

ПРЕДИЗВИКАТЕЛСТВАТА ПРЕД ОБУЧЕНИЕТО ПО ИНФОРМАТИКА В БЪЛГАРСКИТЕ СРЕДНИ УЧИЛИЩА



ФИЛИП ПЕТРОВ

ПРЕДИЗВИКАТЕЛСТВАТА ПРЕД
ОБУЧЕНИЕТО ПО ИНФОРМАТИКА
В БЪЛГАРСКИТЕ СРЕДНИ УЧИЛИЩА

Монография

София • 2021
Университетско издателство „Св. Климент Охридски“

СЪДЪРЖАНИЕ

Научен редактор	доц. д-р Надежда Кучмова Технически университет – София	Предговор от научния редактор	7
Рецензенти	доц. д-р Ангел Ангелов Софийски Университет „Св. Климент Охридски“ доц. д-р Надежда Кучмова Технически университет – София	Увод	10
		Защо да се учи информатика в училище?	16
		Какво се учи по информатика в училище?	32
		Примерен училищен курс по информатика за VIII клас	61
		Примерен училищен курс по информатика за XI клас в профилирана подготовка	123
		Примерен училищен курс по информатика за XII клас в профилирана подготовка	143
		Подготовката на учители по информатика във висшите училища	150
		Използвана литература	172

© 2021 Филип Петров
© 2021 Павел Христов, художник на корицата
© 2021 Университетско издателство „Св. Климент Охридски“

ISBN 978-954-07-5274-7

ПРЕДГОВОР ОТ НАУЧНИЯ РЕДАКТОР

Твърде често читателите прескачат този текст в стремежа си незабавно да се потопят в творбата, но това не е толкова важно, по-важно за мен е удоволствието да се опитам накратко да ви запозная с това, което следва, и с автора, разбира се. Познавам Филип Петров от доста време, а и често чета негови статии на най-различни теми. Бях сигурна, че монографията ще бъде интересна, а не както често се случва – трудно за четене научно пособие. Не се излъгах. Тя разкрива реалната картина в България, представя тенденциите в преподаването на информатика и информационни технологии, засяга наболели проблеми, негативни и положителни тенденции, отчита факти, които не бива да бъдат пропускани. Написана в приятен стил, запознава читателя с редица анализи, направени през годините.

Напоследък в стремежа да се изградят все повече и повече специалисти в областта на информатиката и информационните технологии се пренебрегва значително методиката на преподаване, последователността на поднесения материал, както и качествата на преподавателя, което не води до добри резултати. За съжаление, тенденциите са негативни и въпреки множеството образователни центрове, специалистите не се увеличават значително. В монографията си Филип Петров предлага три примерни учебни програми за VIII, XI, XII клас, чието съдържание позволява бързо навлизане в информатиката, като същевременно не се прескачат етапите на получаване на нужните умения. Анализира-

ни са подходите и материалът за обучение през годините, като са отчетени ефектите от промените, реформите, както и липсата на такива. Предложените примерни програми са съобразени с възрастовите специфики на обучаемите. Авторът е избрал точният момент за поднасяне на материала, като набляга на философията, че заниманието с информатика е творчески процес, а не копиране на готови шаблони и наизустяване. Програмите са базирани на решаване на задачи, упражнения, реализиране на завършени проекти, което създава предпоставка за добиване на умения, а не само на знания в областта, както и ясна представа за професията „програмист“ в по-ранна възраст. Богатият преподавателски опит и наблюдение върху ученици и студенти на автора е видим, което прави преценката му точна и обективна.

Изграждането на специалист в областта на информатиката е дълъг процес на самообучение с натрупване на знания и опит в приблизително сходна последователност. Авторът е съумял да анализира този процес и да предложи подход, който следва логичното надграждане на умения, които да бъдат приложени в образователната система. В монографията се засягат и основни теми като необходимо ли е изучаване на информатика и защо. Отчетен е негативният ефект от премахването на дисциплината като задължителен предмет от масовите средни училища.

Малцина в сферата притежават такава професионална биография като Филип Петров. Той е ерудиран преподавател с дългогодишен опит във водещи висши учебни заведения и училища, специалист в разработване на информационни системи, притежаващ знания и умения в областта на киберсигурността, системното администриране, базите данни и прочее направления в информационните технологии. През годините е успял да привлече мнозина в професията с увлекателните си лекции и упражнения. Автор е на редица публикации не само в сферата на педагогика и информатика. Предвид бурното развитие на софтуерната индустрия в България през последните години, прави чест на Филип Петров, че е избрал да бъде преподавател във висше училище, където да даде своя принос към професионалната подготовка на

бъдещи учители в сферата на информатиката и информационните технологии. Препоръчвам монографичния труд на всекиго, решил да тръгне по не лекия път на учител по информатика и информационни технологии.

Приятно четене!

Доц. д-р Надежда Кучмова
Технически университет – София

УВОД

В края на XX век стана ясно, че с все по-голямата миниатюризация на производствените процеси на електрониката, компютърната техника ще навлезе в огромна част от бита на хората. През 1998 г. у нас се приема Стратегия и национална програма за развитието на информационното общество в Република България [19], в която информационната грамотност на населението е заложена като основен приоритет. Намерението на държавната власт по-късно се препотвърждава с Национална стратегия за въвеждане на информационните и комуникационните технологии в българските училища [27], с която се планира постигане на компютърна и информационна грамотност при учениците чрез цялостно модернизиране на образователна система. Стратегиите намират реално нормативно отражение и в държавните образователни изисквания за учебно съдържание в културно-образователна област: „Математика, информатика и информационни технологии“ [10]. В поредица от стратегии и програми, свързани с внедряване на ИКТ в българското образование [46], [28] и [25] се заявява нуждата от „информатизация на образованието и науката“. Програмата за въвеждане на информационни и комуникационни технологии в училище ([27], [25], [26]) в общи линии е устойчива и по нея се работи с добри темпове (тя [26] започва през 2015 г. и оттогава насам всяка година се преповтаря с допълнителни модернизации и усъвършенствания). Тези програми обаче се фокусират най-вече върху материалната база – осигуряване на средства за достъп до образователни ресурси и развитие на мрежова инфраструктура.

Техническото осигуряване на училищата, разбира се, е важно условие да се случи дигитализацията и съответно повишаването на информационната грамотност на децата. Лесно можем да се досетим, че то само по себе си не е достатъчно – нужно беше учебните планове и учебни програми също да получат необходимото осъвременяване, да се подготвят нови учители за нововъведени учебни предмети, а част от съществуващите учители да получат съответни допълнителни квалификации. Може да се твърди, че като цяло се отделят по-рядко средства по проекти за квалификация на учители и извънкласни дейности за ученици, но все пак подобни съществуват – такъв например е проектът „Образование за утрешния ден“ [40], в чиято дейност 6 е позволено да се финансират „извънкласни дейности (клубове по интереси и допълнителни занимания по ключови дигитални умения – базови и за напреднали, включително и компютърно моделиране/кодиране)“. Обикновено се наблюдава ентусиазъм при училищата за включване в подобни проекти, особено когато има финансиране от Европейски съюз.

В последните няколко години се въведе система от кредити за квалификация на учителите [21], заради която започнаха да се създават и посещават множество курсове в университети и частни школи. Въпреки възможните критики към качеството на немалка част от тях, като цяло стратегията доведе до желания резултат – учителите масово започнаха да посещават квалификационни курсове и да осъвременяват знанията си, като в частност тези по информационни технологии заемат значителен процент и се радват на голям интерес. В крайна сметка макрорамката показва очевидно добри намерения за позитивно развитие: осигуряването на добра материална база, мотивирането за квалификация на учителите и актуализацията на учебните планове са необходимо условие за качествена образователна система.

Реформата на българското образование в посока споменатата дигитализация може да търпи разностранни критики, но като цяло се реализира с положителен резултат. Безспорен факт е, че все повече училища успяват регулярно да си подновяват техниката и много учители добавят използване на информационни тех-

нологии в процеса на обучение, включително и по хуманитарни дисциплини. Така може да се твърди спокойно, че образователната система настига с приемливи темпове технологичния напредък на обществото. Става въпрос обаче за средностатистическото ниво – повечето хора са предимно ползватели на технологиите, а не създатели на такива. Затова и в сферата на образованието тежестта пада върху учебния предмет „Информационни технологии“ (ИТ) като основен носител на минималната дигитална грамотност, която всеки човек трябва да притежава в съвременното общество. От него произтича и огромно поле за междупредметни връзки с други учебни предмети.

В България допълнително можем да се гордеем, че има развит много сериозен отрасъл на създатели на информационни технологии – сектор „Информационни и комуникационни технологии“ (ИКТ) вече е един от водещите в икономиката [2]. За подготовката на специалисти в този сектор се изисква повече от стандартните знания и умения за това как да се използва компютърна техника и софтуерни продукти – необходими са и допълнителни знания как те се създават. Тази област от знания в България исторически е била отделена в отделен учебен предмет наречен „Информатика“. Двата учебни предмета – ИТ и „Информатика“ – естествено се допълват и при добро планиране на учебните програми биха могли да работят в пълна симбиоза помежду си. В много други държави по света те не са разделени, а са част от едно цяло.

Изненада за образователните среди и бизнеса беше новината в края на 2013 г., че се подготвя учебния предмет „Информатика“ да отпадне от списъка със задължителни учебни предмети в гимназиална степен за училища, които не са математически или специализирани гимназии. Това стана факт след приемането на Закон за предучилищното и училищното образование [13] и свързаните с него нови учебни планове. В предмета „Информационни технологии“ като вид компенсация бе пренесена малка част от учебното съдържание на дотогава задължителния учебен предмет „Информатика“ (изучаваше се в IX клас) – става въпрос за темата, свързана с алгоритми, която вече се изучава по ИТ в X клас [65], но

тя от своя страна остана да стои крайно неадекватна спрямо децата, които учат в математически гимназии и имат задължителен предмет „Информатика“ в VIII клас [64]. Това не беше единственият проблем с учебните програми – пропуските в тях ще бъдат коментирани в цяла глава от настоящата монография. В крайна сметка съществена част от знанията в научноприложната област на учебния предмет „Информатика“ – в частност програмирането – изчезна от задължителните предмети на масовите средни училища в България и останаха само в математическите гимназии и специализираните училища, които са относително малко в рамките на Република България.

Тази реформа се появи на фона на пиков за ИТ индустрията в България период – през 2015 г. Министерство на икономиката констатира повече от петкратен ръст за периода 2005–2015 г. и трикратно увеличение на дела от БВП на ИКТ сектора (докладът беше представен от заместник-министър на икономиката Даниела Везиева на Четвърта годишна кръгла маса „Bulgarian ICT Watch“ на ICT Media и Българска асоциация на софтуерните компании БАСКОМ). При това става въпрос за сектор със значима добавена стойност и най-високи средни работни заплати – над 250% над средното за страната [23]. Според прогнози на БАСКОМ от 2019 г. [2] до 2024 г. приходите от софтуерната индустрия ще са 10,3 млрд. лева на годишна база и 7,6% като дял от БВП. Заетите в сектора се очаква да достигнат общо 60 000 души (към края на 2019 г. се оценяват на около 50 000 души [2], като от тях 34 140 са наетите в софтуерния бранш, т.е. хора, които имат досег със знания по програмиране). Освен това всички големи работодатели в сектора заявяват категорично, че се изпитва сериозен недостиг на квалифицирана работна ръка, като в частност един от най-острите проблеми е недостиг на квалифицирани програмисти. Този проблем продължава да е наличен и през 2020 г., а изглед да бъде разрешен в скоро време няма.

До известна степен висшите училища се опитват да компенсират пазарното търсене – специалности, свързани с научноизследователската област „Информатика“ под една или друга фор-

ма, се откриха в много български университети. Тази възможност им е дадена заради относителната им автономия – нещо, което е много по-ограничено при средните училища. Огромен се оказва и ръстът на частния образователен сектор – в последните години се появяват множество частни школи за подготовка на програмисти. Въпреки това остава отворен въпросът защо Министерство на образованието и науката (МОН) се отказва от по-широк достъп на ученици от всякакви училища до базовите знания на една от научните области, в която пазара на труда показва огромен потенциал за ефективно развитие. Естествено, оставена е възможността информатика да продължи да се изучава извън математическите гимназии като избираем предмет и със сигурност по-добрите училища ще предложат подобна възможност за своите ученици, но по-масовата практика беше и вероятно ще продължава да бъде този учебен предмет постепенно да изчезва.

Подобно остро разминаване между нуждите на пазара на труда и планираните кадри от страна на образователното министерство на пръв поглед е изключително нелогичен ход. Би било наивно обаче да упрекваме политиките в некадърност или зъл умисъл. За тези процеси има напълно обективни причини. Основният проблем е, че както има налично текучество на квалифицирани програмисти на пазара на труда, така има и още по-голямо текучество на квалифицирани учители по информатика в училищата. Вероятно ще е дори по-лошо да си затворим очите и да започнем да имитираме дейност, като например по учебен план да се изучава информатика, но реално качеството на обучението да бъде трагично поради липсата на квалифицирани учители в повечето училища. Рядко се признава публично, но всъщност е всеизвестно, че в немалко по-слаби училища часовете в компютърните кабинети се превръщат във време за почивка и игри, а не за учене. Това е особено валидно за гимназиите.

Очевидно е, че опасността постепенно да се въвлечем в спиралата на по-малко подготвени ученици -> по-малко качествени студенти -> още по-малко смислени кадри на пазара на труда -> още по-малко учители -> още по-малко подготвени ученици -> ...

в комбинация с продължаващата демографска криза, в един момент ще доведе до застой или дори срив на ИКТ сектора в България и много големи международни корпорации ще пренасочат активите си към други държави. Това би било жалко при положение, че България е една от първите държави в света, които вкарват обучение по информатика в училище през учебната 1986–1987 г. (тогава X и XI клас), както и това, че държавата е домакин на Първата международна олимпиада по информатика през 1989 г. [8]. Може съвсем спокойно да се твърди, че силният ИКТ сектор в икономиката сега е в голямата си част резултат от натрупване на потенциал и човешки ресурс именно заради тази образователна политика от миналото. Затова вероятно има нужда от цялостно преосмисляне на практическата реализация на заложените стратегии.

В настоящия труд е положено усилие да се даде един по-широк поглед върху проблемите около изучаване на информатика в българските средни училища. Направено е предложение за реструктуриране на учебните програми така, че с минимално разместване на учебно съдържание между различни класове базовите знания по информатика да са по-достъпни. Направен е и преглед на текущото състояние на учебните планове в университетите за специалностите, в които се подготвят учители по информатика, и е споделен по-конкретно практическия ми опит като преподавател от катедра „Обучение по математика и информатика“ в Софийски университет „Св. Климент Охридски“. Изказано е и позитивно мнение относно разрастващия се учебен предмет „Компютърно моделиране“, но с уговорката, че дори сериозно увеличеният хорариум в основното образование не може да компенсира напълно отпадането на информатика от гимназиите.

ЗАЩО ДА СЕ УЧИ ИНФОРМАТИКА В УЧИЛИЩЕ?

Употребата на думите „компетенция“ и „компетентност“ се вижда масово в учебни програми и документи за стратегии, но по мое мнение почти никога не се употребяват коректно. Самите думи произхождат от латински език, където *competentia* означава *някой, който има право да говори и отсъжда*, т.е. в оригинал те предполагат качества на висока интелигентност, опит и мъдрост в даден човек. Естествено, в годините понятията са претърпели известно развитие и интерпретация. Например в [67] са приети дефинициите, че *онези характеристики като знания, умения, нагласи (способности, личностни качества), които се ползват самостоятелно или в различни комбинации и дават резултат в успешно представяне на работното място, се наричат компетенции, а компетентността се равнява на добро представяне, водещо до най-ефикасното постигане на целите на организацията*. В научната литература може да се наблюдава и по-подробно класифициране на различни видове компетентност – социална, емоционална, когнитивна, гражданска и професионална, всяка от които се формира по различен начин и дава различен вид добавена стойност за пазара на труда.

Дефинициите в речниците не са толкова просторно описателни, но в Тълковния речник на БАН например „компетенция“ се дефинира като: *опитност, знания в дадена област*, а „компетентност“ като: *осведоменост, юридическа правоспособност*, т.е. има значе-

ние близко до оригинала. В английските речници „компетенция“ е *важно умение, което е нужно, за да се свърши дадена работа добре* (Кеймбридж), което дава доста широко тълкуване (думите „важно“ и „добре“ могат да се интерпретират), а в речника на Оксфорд: *достатъчност в квалификация; капацитет*. „Компетентност“ в речника на Кеймбридж е дефинирана като *способността да се върши нещо добре*, а на Оксфорд като *достатъчна квалификация; достатъчен капацитет да се работи по дадена тема*.

В училище се добиват знания и умения от дадена научна област по определен ред, който се дефинира на макроравнище чрез учебните планове и на микроравнище от учебни програми, а децата решават различни задачи и научават факти по предварително подбрани от техните учители методи в различни учебни ситуации. Ученето в училище не прави задължително децата „компетентни“, защото компетентността е качество, което трябва да се доказва. Човек, който е завършил училище, може да се нарече компетентен само тогава, когато може да се справя с разрешаването проблеми в работна среда. За повечето хора компетентности се формират по време на творчески труд – тогава, когато са започнали да се сблъскват с проблемни ситуации именно в работна, а не в учебна среда. Въпреки това в съвременното е общоприето, че училищата все пак трябва да се стремят да формират компетенции в обучаемите и затова се говори за т.нар. компетентностен подход в обучението.

При всички положения смятам за неправилно всяко отделно знание или умение да бъде считано за компетенция – нещо, което, за съжаление, се вижда в българските учебни програми. Компетенцията би трябвало да включва множество знания, умения и работно отношение, които да помагат на човека да върши практически полезни дейности на пазара на труда. По същия начин компетентността трябва да се счита за комплекс от множество натрупани компетенции, които да отличават индивида в работна среда спрямо останалите му колеги и да го правят особено полезен на работното му място. Прекалената употреба на тези термини би ги изпразнила от съдържание и ще ги обезсмисли.

Основната цел на училището в рамките на пазарна икономика е да полага основите за развиване на работни компетенции в определен профил, а не да дава готови кадри за пазара на труда в определен брой по определени професии. В този смисъл от тук нататък целенасочено ще избягвам модерната тенденция да се убеждава читателя колко е важно училището да направи децата компетентни в тази или онази научна област (в случая се подразбира, че би ставало дума за информатика). Би било хубаво един завършил средно образование човек, който не е имал и един ден реален трудов стаж, да бъде компетентен програмист, но поставянето на подобна цел със сигурност ще бъде куха и нереализируема в макроплан – ще има няколко светли примери на ученици, които са го постигнали, но огромното мнозинство ще бъде далеч от това. Напротив, смятам, че основната цел на обучението в училище трябва да е да се дадат базовите знания и умения, да се формира правилно отношение към работни проблеми чрез поставяне в подходящи проблемни ситуации в учебна среда и всичко това да се разгледа като процес за **евентуално** развиване на компетенции. Колкото до компетентността, както вече казах, смятам, че тя трябва да се доказва чрез професионализъм на работното място. Затова предлагам следните дефиниции:

Компетенция: пренасяне на наученото (знания, умения, отношение) в разрешаване на проблемни ситуации в извънучебна среда.

Компетентност: множество от компетенции, които формират висока добавена стойност за пазара на труда.

Ако се приеме подобна нагласа, трябва веднага да се забележи, че, за да може да се формират компетенции, в училище се очаква децата и младежите да се докоснат много повече до приложните аспекти на науките, а значително по-малко до тяхната чисто теоретична основа. Идеята е по-скоро да бъде запален интересът към дадена наука, да се види нейната интересна приложна страна и да може децата да се научат да извършват някои практически полезни спрямо тяхното текущо ниво на развитие дейности в нея, а не да бъдат изучавани чисто абстрактни конструкции. Научноиз-

следователската дейност дава изключително важни ползи за пазара на труда (нещо, което сме описали с колега в [32]), но тя може да бъде извършвана ефективно едва след като има вече достатъчно натрупан практически опит.

Поради изложените причини смятам, че в училище трябва да има много повече практика и много по-малко теория. Това е и основната концепция, за която настоящият труд ще се застъпи. Затова и подкрепям мотивите, че трябва да се изучават много повече информационни технологии и много по-малко информатика – информатиката е теоретичната основа, на базата на която са изградени информационните технологии, а е напълно нормално да се очаква в училище да се изучават предимно приложните страни на науките. Въпреки тази теза, не трябва да се стига и до крайности, т.е. не трябва теоретичните знания по информатика да отсъстват напълно.

Информатиката е наука за събиране, преобразуване, пренос и съхранение на информация. В българската литература е особено популярна дефиницията от [3]:

Информатиката е наука, която изучава информацията от гледна точка на нейната структурираност, форми и начини за представяне, количествените ѝ характеристики; информационните процеси като композиция на основните информационни дейности, както и методите и средствата за автоматизирането им.

Терминът всъщност е с френски корени и вероятно се появява за първи път в [69], където авторът го създава като обединение на „информация“ и „автоматика“ (в смисъл като автоматизация на обработката на информация чрез компютърна техника). В английската литература терминът не се използва, а вместо него – компютърна наука (computer science) или по-близкото наука за информацията (information science).

Приложните аспекти на информатиката вече са много и в последните години продължават да нарастват като обем. Например такива са програмиране, софтуерно инженерство, бази от данни, разпределени системи, компютърни мрежи, научни изчисления,

компютърна сигурност, криптография, компютърна графика, компютърни архитектури, машинно самообучение, изкуствен интелект и др. Би било неправилно в училищен курс да се тръгне в посока на стремеж към всеобхватност – подобна грешка би довела до познатото от много други учебни предмети „претрупване“ на учебния материал, който после довежда до „препускане“ през него и в крайна сметка в най-добрия случай до неговото „зазубряне“ (механично наизустяване). Изключително важно е младежите да имат достатъчно време да упражняват изучените знания и по този начин да добиват умения за прилагането им в практиката.

Това мнение се застъпва все по-силно от много източници в педагогическа литература, като например в [18] се казва, че в настоящото време е необходима смяна на педагогическата парадигма – учениците не е толкова важно да трупат големи обеми от знания за различни технологии, а вместо това е по-добре да могат да използват и адаптират в практически задачи по-малък обем от тях, както и да умеят да разрешават специализирани проблеми от различни приложни области. Затова и аз смятам, че всяка учебна програма, в която изучаването на нови знания е 50% или повече от учебното време, е претрупана с информация. При такива програми не остава достатъчно време тези знания да бъдат упражнени чрез самостоятелна работа. Оттам идват и оплаквания на учители, че се „препуска през материала“, а от родителите, че „децата са претрупани с информация“. Всичко това е породено от напълно ненужните напъни да се научи колкото се може повече, за да може да изглежда, че децата са много знаещи, можещи, „компетентни“. Крайният резултат е, че от многото неща, истински научени остават сравнително малко. Така учебните предмети започват да отблъскват учениците, а не да ги мотивират да продължат да се развиват в съответната област.

В контекста на математиката например при едно такова претрупване със знания децата да гледат как учителят решава задачи, а не те самите да ги решават. В часовете по литературата това би означавало да се запознават с готови анализи на произведения, вместо самите те да пишат есета и съчинения, които после да се

обсъждат, анализират и подобряват. За историята това би било изслушването на една поредица от факти за исторически събития, а не провеждане на дискусии около различни интерпретации на отговора на въпроса защо дадено събитие се е случило по точно този начин, а не по друг. Подобни примери могат да се дадат с абсолютно всеки учебен предмет – представете си например по физическо възпитание и спорт децата само да четат правилата на спортовете и да гледат клипчета с професионални спортисти, а не самите те да играят. Всеки ще каже, че това би било нелепо. Също толкова нелепо е и за всички останали предмети, но просто не е толкова очевидно. Знания, които не са упражнени практически чрез самостоятелна работа, а са затвърждавани само чрез преписване на типовите задачи, демонстрирани учителя, са кухи – няма ги истинските умения за тяхното прилагане, т.е. са били безсмислено зазубряни.

Затова в частност, ако се гради един училищен курс по информатика, той трябва да се опита да избегне тези пороци от претрупването с учебен материал. За съжаление, много от учебните програми в българското училищно образование не спазват този принцип – такива са учебните програми по ИТ за V, VI, VII и X клас [61], [62], [63], [65] (първите три от изброените са с цели 56% отделени за нови знания плюс 6% за тяхно „обобщение“), както и модули 3 и 4 на учебната програма за профилирана подготовка по информатика за XII клас [66]. Важи и за наскоро излезлите обновени програми по компютърно моделиране и ИТ [56], [57], [58]. При това тези проценти са само по изчисленията и самооценка на самите автори на въпросните учебни програми – една външна обективна оценка би могла да внесе още по-негативен извод за тяхното съдържание.

Същият проблем, но разгледан от различна гледна точка, е въпросът, пред който неизбежно се изправя всеки един училищен курс в дадена научна област – дали да се направи опит за разностранно изучаване на колкото се може повече приложни области на познание (на принципа „от всичко по малко“) или да бъде поставен фокус върху по-тясна област, която да се изучи в по-дъл-

боки детайли (по-тясна специализация). Тясната специализация, разбира се, е важно нещо и именно тя води до натрупването на необходимото количество знания и умения, които след училище прерастват във формиране на така желаните компетентности, но тя е много трудна и дори опасна за ранен етап в обучението. За да може даден ученик да прецени дали дадена професия е подходяща за неговото бъдеще или не, той трябва да се докосне до поне малка част от нея. Една ранна тясна специализация би лишила децата от досег до много професии, част от които може би са били по-подходящи за тях спрямо тази, в която са били насилствено насочени в ранна възраст. Тук стои и коренът на проблема с претрупването с информация на децата в училище, за който стана дума по-рано – той е стремежът да им бъде показано всичко, за да имат по-широк избор. Именно това е и основният проблем през авторите на учебни програми – хем трябва да се покаже колкото се може повече, хем това води до неефективност и слаби резултати.

Изходът от тази ситуация е да бъдат много внимателно селектирани общите фундаменти на различните приложни области на изучаваната наука и оттам училищното образование да бъде фокусирано именно към тях. По този начин децата ще могат да се докоснат до общите части на множество различни професии накуп. Така например по литература не е дори чисто физически възможно да бъдат прочетени и анализирани всички възможни литературни произведения, но спокойно могат да бъдат подбрани едни от най-добрите такива за основните жанрове и епохи. Непосилно е да се постигне всеобхватност за науката математика и всички нейни области на приложение, но исторически е постигнат известен консенсус кои части от аритметиката, алгебрата, геометрията, анализа, теорията на множествата и логиката могат да се обединят в един общ куп като нейни „основи“ – именно те са улегнали в училищните курсове по този предмет. Същото може да се твърди дори за чисто приложни предмети като изучаването на чужд език и дори изучаването на роден език – например няма как да се изучат всички възможни термини от всички приложни области, но спокойно могат да бъдат селектирани тези, които се използват

най-често в ежедневието (бита) на хората. Същото трябва да бъде направено за всяка научна област, която се преподава в училище и съответно за информатиката в частност.

Информатиката е сравнително нова наука спрямо историята на човечеството, но въпреки това вече има формирани традиции. От една страна, е множеството на чисто практически знания и умения за работа с компютърни системи и технологии – базови познания за работа с различни операционни системи, текстообработка, работа с електронни таблици, уеббраузър, електронна поща и т.н. В България те еволюционно са отделени в обособен учебен предмет „Информационни технологии“ (ИТ), който стана масов и „завладя“ учебните планове във всички класове на прогимназиалния и гимназиалния курс на обучение. Характерното за този тип знания е, че те са насочени към ползватели на готови софтуерни продукти и компютърна техника. Чрез тях учениците добиват знанията и уменията да използват компютърната техника като помощник в бита и бъдещата си професия. Множеството на софтуерни продукти, които се използват масово в почти всички професии, е сравнително лесно за дефиниране. Така в контекста на казаното дотук, ИТ е много полезен учебен предмет, защото сам по себе си може да се разгледа като една силна междупредметна връзка между практически всички потенциални професии – той дава на учениците инструментариум за по-лесно решаване на практически задачи при всички други учебни предмети, независимо дали са свързани с хуманитарни или с природни науки. Именно това е довело и до естествената еволюция предметът ИТ да стане масов и задължителен във всички основни и средни училища.

Втората важна гледна точка при определянето на фундаментите на науката информатика, които би следвало да се изучават в училищата, е свързаната със създаването на тези софтуерни продукти. За държава като България, в която сектор ИКТ е с изключително бърз растеж и носи голяма добавена стойност за икономиката [2], това не е пренебрежима предметна област и не бива да бъде подминавана с лека ръка. Отново по еволюционен път, фундаменталните знания и умения от тази предметна област в

България са обособени чрез учебния предмет „Информатика“. За разлика от ИТ, където определянето на текущо модерните и полезни за изучаване технологии е сравнително лесно, за определяне на фундаментите за информатиката, в комбинация с нуждата от това изучаваното в училище да е максимално практично, са нужни значително повече усилия.

Трудностите при получаването на знания по информатика идват оттам, че по начало създаването на софтуерни продукти е творчески процес. Съставянето на смислена програма не се случва просто с писане на код по готови шаблони, а изисква иновативност, защото този софтуер после трябва да бъде реализиран в конкурентна пазарна среда, т.е. да бъде продаден. Разбира се, твърдението, че програмирането е творческа професия, може да търпи значителна критика, защото неслучайно в последните години се заформира нов термин „софтуерен инженер“, с който се цели да се акцентира именно върху творческите заложби на индивида и да се противопостави на нарастващата тенденция софтуерните продукти да се правят чрез „сглобяване на заготовки“ (нещо като аналогия на поточните линии при модернизацията на редица производства от миналото). Това е вярно и е валидно за всяка професия, която е претърпяла процеси на роботизация и автоматизация, но в крайна сметка висшата цел на всяко модерно образование, което е насочено към пазарна икономика с висока степен на конкуренция, е не само да захранва пазара с работници, но и да формира потенциални бъдещи работодатели. То се случва именно чрез стимулиране на творческо мислене. Не на последно място трябва да се отчете и спецификата на информатиката, свързана с това, че в последните две десетилетия създаването на софтуер се превърна в екипна работа и вече е все по-рядко възможно сам човек да бъде създател на конкурентни програмни продукти. Всички тези неща трябва да бъдат отчетени в един съвременен училищен курс по информатика. В следващата глава тези въпроси ще бъдат разгледани по-подробно.

В крайна сметка отговорът на въпроса „Трябва ли да се учи информатика в училище“ е лесен в рамките на България – да, защото секторът ИКТ в икономиката вече е достатъчно голям и стана

един от най-значимите за бъдещето на държавата, а създателите на информационни продукти в частност са именно тези, които носят най-високата добавена стойност в него. Оттук насетне трябва да бъдат поставени редица последващи сериозни въпроси, а именно – какво и как да се учи, но също така и кой да преподава. На тези въпроси ще бъдат посветени следващите глави на настоящия труд.

Защо тогава информатиката внезапно изчезна като масов предмет от прогимназиалния и гимназиалния курс на обучение в България? Така поставеният въпрос може да получи най-различни спекулативни отговори. Вероятно ще съществуват мнения, че много от професиите въобще нямат нужда от знания по информатика – естествено, че това е вярно, но така може спокойно да се породят безкрайна полемика около съдържанието на практически всички задължителни учебни предмети. Нима в ежедневието си използваме всички знания, които сме получавали в училище? Истината е, че използваме прекалено малко от тях, но това не значи, че училището е било безполезно и ни е учило на неща, които са ни напълно ненужни. Въпросът е, че информатиката и в частност програмирането все пак са държавен приоритет и като такъв е нужно максимално много ученици да имат поне минимален достъп до фундаменталните знания на тази наука. Изчезването на предмета в масовите гимназии и заменянето му с една скромна тема с почти нищожен хорариум по информационни технологии в X клас определено не следваше този приоритет.

Естествено, може да се изтъкне много сериозен и логичен контрааргумент, че все пак в началния курс на обучение влезе нов учебен предмет „Компютърно моделиране“, който в своята същина представлява един лесен, забавен и увлекателен за малки деца начин да се докоснат до основи на програмирането (в България се използва най-вече популярната среда за визуално програмиране Scratch [47] и по-рядко Kodu [6] или други). Така погледнато информатиката не само че не е изчезнала, а даже е увеличила хорариума си. Аргументът е силен, защото няма особено значение какво е формалното наименование на един учебен предмет, а е важно какви знания и умения се добиват с него.

Не само това, но на 25 юни 2020 г. от Министерство на образованието и науката анонсираха предстоящо обсъждане на поредна реформа в учебните планове, а на 1 юли 2020 г. беше обявен за обществено обсъждане [41]. В нея се внесе предложение да се добави нов предмет „Дигитални технологии и творчество“ между V и VII клас, който впоследствие беше логично променен като име на „Компютърно моделиране и информационни технологии“ и получи утвърдени учебни програми [56], [57], [58]. Този предмет практически допълва досегашния „Информационни технологии“, като внася допълнителни теми, за които е предвиден допълнителен хорариум от половин час седмично. Темите са еднозначно от сферата на информатиката:

- Създаване на графични изображения с изучаван език за блоково програмиране.
- Създаване и използване на собствени блокове или подпрограми.
- Създаване на образователен проект със средствата на изучаван език за блоково програмиране.
- Преминаване от език с блоково програмиране към скриптов текстов език.
- Създаване на анимация със средствата на скриптов текстов език.
- Основни типове данни в скриптов текстов език за програмиране.
- Приложение на циклични конструкции.

Видимо по този начин се поставя цел за добавяне на знания от сферата на информатиката в масовите училища, при това не само със „сухи“ теоретични теми, а обвързано с приложно програмиране. Предложеното програмиране на скриптов език (очаквано най-често JavaScript, но не задължително) е предвидено да става чрез съпоставка на вече направени програми чрез блоков език. Силно се надявам, че проблемните теми за алгоритми по ИТ в X клас [65] също ще бъдат премахнати, ако стратегията бъде приета и получи реално приложение с нови учебни програми.

Все пак моето мнение е, че новият предмет „Компютърно моделиране“ в III и IV клас и продължението му с теми между V и VII клас не заместват напълно предишния задължителен предмет по информатика от IX клас. Изучаването на програмиране в ранен етап чрез визуални среди за блоково програмиране не докосва децата до приложното програмиране, което изисква писане на програмен код, а само загатва какво се случва в него. Приоритетът на обучението в началните класове не е децата да се учат да програмират, а да се приучат на основни навици за решаване на задачи в програмна среда [72]. Чрез Scratch или Kodu учениците получават основите на алгоритмичното мислене, което е изключително полезно за едно бъдещо развитие в посока към някоя професия от областта на информатиката, но не е същинско учене на практически полезни знания и умения за професията „програмист“. Именно затова компютърно моделиране и най-вече елементите на изучаване на скриптов език с него може да се приеме като пропедевтика за изучаване на същинския курс по програмиране и няма как да бъде приет като негова пълна замяна. Тоест като цяло вкарването на изучаване на елементарни алгоритми и практическото им реализиране с приложни програми по компютърно моделиране ще даде една чудесна основа за надграждане. Така всеки български ученик ще се е докоснал поне малко до приложно програмиране. В този смисъл приветствам стратегията на проекта за изменение и допълнение на Наредба № 5 [41] и реализацията му с учебните програми [56], [57], [58].

По-основният проблем са междупредметните връзки – в крайна сметка програмирането на софтуер трябва да се прави, за да се моделират и решават практически задачи, които идват от знания по други учебни предмети. Сами може да си направите извод за количеството от знания, върху които може да се правят компютърни модели и които притежава един ученик от основно училище, спрямо тези, които има един гимназист. В първия случай това са деца, които тепърва преминават през основни етапи от света на познанието, а във втория говорим за хора, които вече са започнали професионалното си профилиране и са се насочили към оп-

ределена група от възможни бъдещи професии. Изхождайки от твърдението, че програмирането (или по-ограничено до софтуерното инженерство) е творческа професия, това неизменно изисква обучението по програмиране да бъде извършвано на творчески принцип – чрез моделиране и решаване на практични задачи от реалния свят.

Именно фактът, че професионалното ориентиране на повечето хора се случва в средните, а не в основните училища, трябва да бъде отчетен много внимателно. Най-често учениците се ориентират към това в коя гимназия да следват на базата на свои знания и умения по български език и литература (БЕЛ) и математика в VII клас – това са учебните предмети, по които се извършва националното външно оценяване (НВО) и с които се кандидатства. Дали даден ученик е по-добър по математика или по БЕЛ, може да се приеме за някакво очакване за неговите бъдещи способности и съответно за определящо дали ще бъде по-добър в природните или в хуманитарните науки. Това обаче не трябва по никакъв начин да се приема като категорично „отрязване“ на дадени професионални области за реализация на тези хора. Това, че даден ученик е попаднал в езикова гимназия или друг вид училище, което не е математическа гимназия, не означава, че той няма да се реализира в бъдеще в ИКТ сектора. Напротив. Примерите са много.

Например в огромния световен бизнес за създаване на компютърни игри, в който България има сериозно участие с човешки капитал, не работят само програмисти, а също така графични дизайнери, аниматори, музиканти и др. Тези специалисти не е достатъчно само и единствено да могат да създават красиви графики, анимации или музика – те трябва да имат поне минимални познания за „механиката“ за изработването на продукта, в чийто контекст ще бъде приложен техният труд. Разбира се, че не трябва да се очаква един художник със средно образование да може да програмира, но със сигурност би трябвало да знае какво е програмиране, защото е силно вероятно да работи сред екип за създаване на софтуерни продукти. Така той ще знае потребностите и проблемите на своите колеги програмисти и предизвикателства-

та, пред които те са изправени, и съответно ще може да адаптира продукта от своята работа така, че да ги улеснява максимално. Тоест говорим основно за увеличаване на производителността на труда – показател, за повишаването на който в България има какво да се очаква.

Абсолютно същото важи за много други професии, които са съпътстващи бизнеса с разработка на софтуер – не са малко специалистите от хуманитарни дисциплини, които се занимават с въпроси като достъпност на продукта (функционалности за масовия потребител, за потребителите с физически затруднения и т.н.), писане на документация и потребителски ръководства, правни аспекти около лицензи и договори, икономисти, мениджъри, та дори ако щете и „визионери“ – икономисти с висока философска култура – хора, които предвиждат бъдещото развитие в даден сегмент от пазара. Всички те нямат нужда от задълбочени знания в програмирането на софтуерни продукти, но със сигурност имат нужда от поне основни знания и умения около принципите и начините за тяхното създаване. Още по-силно това важи за професионалните гимназии, т.е. хората от които се очаква да бъдат бъдещите инженери, които да захранват индустрията. За тях също не е непременно важно да могат сами да решават практическите си задачи за програмиране около автоматизация на производствени процеси, но определено трябва да имат основните знания, с които да могат да генерират изпълними задачи за свои колеги програмисти.

Поради изложените причини отпадането от масовите гимназии на иначе скромния курс по информатика, който преди се изучаваше по един час седмично в IX клас и даваше основните знания и умения по програмиране, се прие като лош и непремърен ход от страна на образователното министерство, когато се проведе скорошната реформа в България. Лично аз смятам, че последващото добавяне на компютърно моделиране в началния и основния етап на обучение не е пълен заместител - разглеждам го по-скоро като частично компенсиране. Одобрявам и подкрепям стратегията от [41], но дори с нея не бих казал, че проблемът ще е напълно решен.

Не е необоснована тезата, че „няма свободен хорариум“ – да, вярно е, че се въведе нов предмет „Гражданско образование“, знанията от който са не по-малко важни от вече споменатите, както и че всяка една част от учителската гилдия би се защитавала яростно срещу намаляването на часове по нейния учебен предмет – хорариумите са обективните причини да се каже, че „в крайна сметка все нещо трябва да отпадне“. Един от по-прагматичните погледи върху ситуацията с отпадането на информатика от гимназиалния курс обаче е, че просто няма достатъчно учители по информатика на пазара на труда. Ще бъде много по-лошо учениците да имат формално съществуващ учебен предмет в програмата си, по който всъщност не се учи нищо. Сигнали за такива примери за съжаление е имало и вероятно продължава да има налични, а причините за тях в основата си са свързани с недобре подготвени преподаватели – хора, които трудно могат да бъдат определени като „учители“. Естествено, в никакъв случай не може да се твърди, че всички са били такива – това далеч не е вярно, а напротив: в много училища има светли примери на изключително качествени специалисти. И все пак еднозначно може да се твърди на базата на обявите за свободни работни места по сайтовете на РУО из страната, че беше достигнато едно лошо кадрово положение с учителите по информатика и за него има обективни причини, които са основно икономически – средната работна заплата в ИКТ сектора е несъпоставима с тази на учител по информатика. Още повече, че едно внимателно вглеждане в стратегията [41] ще покаже, че увеличението на часове по компютърно моделиране всъщност се случва за сметка на намаление на часовете по чужд език, а това е другата сфера, където българското образование изпитва голям недостиг на добри учители – и там говорим за същия проблем и сходно решение.

С оглед на гореизложеното, може да се тръгне с разсъждения в две посоки. Едната е да решим, че е много важно информатиката да се изучава и да започне да се предлага да бъде върната отново като масов предмет в средните училища. Другата е да се приеме съществуващото положение като даденост и да се потърсят вари-

анти за вътрешни реформи, така че проблемът да бъде разрешен в бъдеще. По-практичният и реалистичен вариант е вторият. Смятам, че трябва да се направи всичко възможно основните понятия за алгоритми – рекурсивни или итеративни, детерминирани или недетерминирани, крайни или безкрайни, дискретност, масовост, ефективност – не да се изучат и формализират подробно в теоретичен план, но да бъдат показани с чисто практически примери по време на часовете по компютърно моделиране. Това би дало една добра основа за лесно и бързо включване в ефективен курс по програмиране, който може да бъде оставен в полето на математическите гимназии и специализираните училища. За останалите ученици една пауза от 5 години може и да е прекалено голяма, но няма да е непреодолима пречка за развитие в тази област.

Нататък в монографията ще изхожда именно от една такава отправна гледна точка – ще предположа, че стратегията [41] ще се осъществи успешно, т.е. в допълнителните часове по компютърно моделиране колегите учители ще успеят адекватно да развият блоковото програмиране в посока на изучаване на различни по тип и сложност алгоритми, както и ще докоснат учениците до писане на код. Оттам ще предложа реформи в задължителната програма по информатика и профилираната подготовка за средните училища, която да надгражда ефективно точно една такава основа.

КАКВО СЕ УЧИ ПО ИНФОРМАТИКА В УЧИЛИЩЕ?

България има силни традиции в изучаването на информатика в училище. Проблемна група за образование (ПГО) [17], позната още като сендовската система, въвежда информатика в пилотните си проекти още в VI клас с голям хорариум от 64 часа [14]. На първо равнище (задължителната подготовка) прави впечатление, че в тази сравнително ранна възраст е било планирано учениците да се занимават активно с програмиране, като е трябвало да изучават процедури и променливи, рекурсия и някои структури от данни. Използваният език за програмиране е Logo, който е давал възможност за визуално програмиране – концепция, която в България беше забравена за дълго време и се върна обратно чак през 2017 г. с учебните програми [64], [59] и [60], а наскоро и с [56], [57] и [58].

За първи път учебният предмет „Информатика“ се въвежда масово за всички училища през учебната 1986–1987 година в учебните планове на МОН за X и XI клас [8], [14]. България е четвъртата държава в света, която го осъществява [47]. Това се прави още преди да има написани учебници за ученици и поради тази причина в първата година и половина в училищата се използва университетски учебник за подготовка на учители [1]. Все по-мощната автоматизация на производствата и по-масовото навлизане на персоналните компютри в бита на хората наложиха постепенно в курса на обучение да се добавят за изучаване все повече готови софтуерни технологии. Така постепенно в България се формира разби-

рането, че е целесъобразно учебният предмет да бъде разделен на две части – една, в която да се изучават различни технологични средства за решаване на практични задачи, и втора, в която да се запознават учениците с научните фундаменти, свързани с разработката на софтуерни продукти и основните принципи за функционирането на електрониката. През 1994 г. се въвежда за първи път нов учебен предмет „Информационни технологии“. В началото на новия век, през учебната 2000–2001 година, двата учебни предмета стават задължителни за всички училища в IX и X клас. Тогава те се допълват много адекватно и учебните им програми видимо са били писани с визия за използване на много междупредметни връзки помежду си.

В следващите години започва тенденция за разширяване на изучаването на информационни технологии, а развитието около учебния предмет „Информатика“ по-скоро спря. Учебната 2006–2007 година беше много важна за българските училища, защото за първи път „Информационни технологии“ се въвежда и в прогимназиален етап – учебният предмет става задължителен за V, VI и VII клас, с по 30 часа учебен хорариум във всеки от тях. Тогава се акцентира върху нуждата знанията по този предмет да се усвояват на практична основа – чрез решаване на проектни задачи. След тази реформа, в гимназиален етап за общообразователна подготовка информатика продължи да се изучава само в IX клас, а информационни технологии – по 1 час седмично в VIII и IX клас. Видимо беше, че стратегиите на министерството за разширяване на изучаването на информационни технологии получиха мащабна реализация.

В периода от 2006 до 2016 г. може да се твърди, че обучението по информатика и информационни технологии в училищата „отпъква своята територия“ и се утвърди. Изучаването на информационни технологии само по себе си предполагаше силна динамика откъм техническите средства, които се използват в училище – нови версии на масово изучавани софтуерни продукти се появяват почти всяка година, с което и учебното съдържание се нуждаеше от постоянно осъвременяване. Макар и невинаги с нужните темпове,

училищата като цяло се справяха добре с динамиката на пазара и успяваха да отговорят на тези предизвикателства.

За съжаление, положението с информатиката беше по-различно. Частта от предмета, свързана с програмиране, почти не се беше развила спрямо концепциите от 80-те и 90-те години на миналия век – предполагаше се, че децата ще правят първите си стъпки в програмирането на процедурен език в конзолен режим, докато в същото време в практиката масовият софтуер много отдавна беше с графичен интерфейс. Причините бяха комплексни: от една страна, липсата на допълнителни квалификации за действащите учители, от друга, липсата на млади учители по информатика и от трета рестриктивни учебни програми, които не дават възможност за лесна промяна на основната парадигма (естествено, който е запознат с реалното положение, ще знае, че в немалко училища учителите сами се справяха с последния проблем, като си осъвременяваха техническите средства и учебното съдържание сами). Развитието на мобилните технологии допълнително утежни това положение, защото в часовете по програмиране учениците не създаваха програми, които изглеждат „модерно“ – фундаменталните знания, които се добиваха, бяха добри сами по себе си, но практическите примери, с които те се упражняваха, изглеждаха по-скоро като „архаика“.

В края на 2013 г. започна, а по-късно и завърши, пореден ход към много голяма реформа в българското образование, която засегна и обучението по информатика и информационни технологии в частност. Подготвиха се учебни програми, според които учебния предмет „Информатика“ отпада напълно от общообразователната подготовка – според действащите към момента учебни програми той се изучава задължително само в VIII клас в математически гимназии и специализирани училища. Някои от фундаменталните елементи в него – основно по-абстрактната тема за алгоритми – се преместиха при информационни технологии в X клас [65], но по изключително странен начин същата тема остана и при информатика в VIII клас [64] за учениците от профилиите. Подобни макар и дребни разминавания водят до впечатлението, че учебните про-

грами тогава са правени „на парче“ и без обща визия – все едно екипите, които са разработвали различни учебни програми, не са комуникирали помежду си и по този начин не се е получила синхронизация. Иначе учебният предмет „Информатика“ получи така желаното от много време насам осъвременяване на учебната програма, в частта за програмиране на който беше насърчено създаването на приложения с графичен потребителски интерфейс чрез модерна среда (IDE) и с парадигма на обектно-ориентирано програмиране.

От друга страна, както беше споменато в предишната глава, беше въведен нов учебен предмет „Компютърно моделиране“ [59], [60] за III и IV клас, в който на практика се възражда концепцията на ПГО, за която стана дума в началото – децата ще започнат да се учат да създават визуално-атрактивни програми с минимални усилия благодарение на удобна за използване среда (най-често Scratch, по-рядко Kodu или други алтернативни продукти), предоставяща им готови компоненти, с които да „сглобят“ различни пъзели, да местят фигури по екрана, да оцветяват и др. С тези дейности „зад кадър“ всъщност се извършва подготовка за изучаване на основни концепции като „последователност от операции“, „условен оператор“, „цикъл“ и др. Въпреки че не може да се говори за осъзнато изучаване на научните фундаменти, това обучение може да се приеме като една много добра пропедевтика. Действително чрез компютърно моделиране се очаква децата да бъдат мотивирани, че програмирането е приятна професия – изключително лесно се създават много ефектни игри и анимации, а от друга страна, започва формирането на т.нар. алгоритмично мислене у тях.

През 2020 г. се въведе ново предложение, което впоследствие беше прието – разширение на часовете по компютърното моделиране чрез добавяне на допълнителен хорариум в V, VI и VII клас [41]. По-късно се написаха и учебни програми [56], [57], [58]. Те са поредна стъпка на пренасяне на знания от сферата на информатиката в прогимназиален етап. Това би следвало да породии и известни бъдещи промени в съществуващите гимназиални курсове, включително този по информатика за VIII клас в частност. Визията

ми е, че курсът по информатика в VIII клас би могъл да се облекчи откъм учебно съдържание с отпадане на някои по-теоретични теми и вместо тях да се фокусира изцяло в приложно програмиране. Именно тази теза ще застъпвам и нататък.

В крайна сметка към края на 2020 г. не може да се каже, че информатиката изчезва от българските училища. Напротив, може да се твърди, че заради новия предмет „Компютърно моделиране“, както и пренасянето на част от учебното съдържание към информационни технологии, тя до известна степен е разширила обема си като брой часове. Частичен проблем е, че изчезването ѝ като обособен учебен предмет от общообразователна подготовка в гимназиите позволи да се бъде изгубен системният подход при изучаването на знанията от нейната предметна област. Още по-лошото е, че липсата на задължителен предмет се очаква да доведе и до постепенното изчезване на подготвени учители, които имат компетенции с актуално приложно програмиране (въпросът не е само да са преминавали курсове по програмиране в университет, а да го упражняват активно и на практика – в противен случай знанията губят актуалност и се забравят). Така досега до програмиране на един завършил обикновено средно училище или езикова гимназия ученик ще е бил сведен до забавните часове от компютърно моделиране от III и IV клас, надграждането им с по една тема в V, VI и VII клас, заедно с изключително кратко (две-три седмици по 1 час) и сухо (без практическа реализация) теоретично изложение за основни видове алгоритми в X клас. Тоест гимназиалното обучение като цяло ще си остане със силно редуцирано обучение по информатика извън математическите гимназии и специализирани училища.

Би било нередно да се правят оценки за крайния ефект от цялата реформа, преди да са видени поне първите няколко нейни випуска. Също така е ясно, че някои средни училища ще могат да въведат адекватни часове, свързани с програмиране, чрез избираеми предмети (основният фокус на реформата беше свързана именно с намаляване на задължителните и увеличаване на обема на избираемите часове). Така например в 73 СУ „Владислав Грама-

тик“, гр. София, Антоанета Фитнева (дългогодишен базов учител, който приема стажанти от Софийския университет „Св. Климент Охридски“) продължава да преподава обектно-ориентирано програмиране с програмния език C#, макар че информатика вече не се изучава задължително в нейното училище. Със сигурност тя не е и няма да е единствен пример, но въпреки това са налични предпоставки да се изкаже хипотеза, че завършващите ученици в бъдещите поколения ще имат значително по-малко досег с практическо програмиране в общинските и държавните средни училища като цяло, отколкото тези преди тях. Интересът към професията едва ли ще спадне – силно вероятно е голяма част от родителите да записват своите ученици на частни курсове (нещо, което вече е факт), но това не е оправдание за сигналите на безпомощност, които идват от общественото образование в тази научна област.

Пренасяйки се от света на общообразователната към профилираната подготовка, започват да извират допълнителни проблеми, които са свързани с учебните програми. На първо място трябва да се отбележи, че за учениците от математически гимназии има голямо дублиране на учебно съдържание в различни класове. Вече споменах за такова на тема от общообразователна подготовка по информатика за VIII клас и ИТ за X клас (табл. 1). Дори неспециалист може да добие усещане, че екипите, които са разработвали тези учебни програми, не са работили по начин, по който да се стремят знанията да се надграждат и допълват. Дали е вярно, че тези екипи са работили без връзка помежду си, е трудно да твърди, но със сигурност има основателни съмнения, че това е било точно така.

Подобно нещо се случва и при профилираната подготовка по ИТ в XI клас (модул 1, тема 2 за въведение в информационни системи) [55] и информатика в XII клас (модул 3 за релационни бази от данни и модул 4 за информационни системи) [66], където също има дублиране на теми. При това и в този, и в предишния пример учебната програма по ИТ включва предимно теоретични знания и много малко практически дейности – това въобще не е нормално особено за този учебен предмет.

Таблица 1. Дублиране на учебно съдържание между информатика за VIII клас и информационни технологии за X клас

УЧЕБНА ПРОГРАМА ПО ИНФОРМАТИКА ЗА VIII КЛАС (ОБЩООБРАЗОВАТЕЛНА ПОДГОТОВКА)	УЧЕБНА ПРОГРАМА ПО ИНФОРМАЦИОННИ ТЕХНОЛОГИИ ЗА X КЛАС (ОБЩООБРАЗОВАТЕЛНА ПОДГОТОВКА)
1.3. Алгоритми и езици за програмиране: • Дефинира понятието „алгоритъм“ и описва основните му характеристики (резултатност, крайност, детерминираност, масовост). • Описва и проследява несложни, линейни и разклонени алгоритми с различни средства. • Описва същността, структурата и разновидностите на цикличните алгоритмични конструкции.	2.1. Алгоритми: • Определя понятието „алгоритъм“ и основните му свойства: определеност; масовост; крайност и резултатност; дискретност; формалност; сложност. • Описва основните видове алгоритми: линеен, разклонен и цикличен. • Посочва примери за дейности, които могат да се опишат с линеен, разклонен и цикличен алгоритъм. • Познава начин за представяне на алгоритми чрез визуална среда. • Разчита алгоритъм от основните видове, представен словесно чрез псевдокод. • Посочва резултат на кратък алгоритмичен фрагмент, описан с псевдокод и с визуална среда.

Много по-сериозен е проблемът между задължителната информатика от VIII клас и профилираната подготовка от XI и XII клас. В модул 1. „Обектно-ориентирано проектиране и програмиране“ от [66] не са спазени основни принципи при писането на учебни програми за профилирана подготовка в училища – всичко е разписано прекалено подробно до най-малки детайли, което не позволява свобода за творчество за авторите на учебници и дори до известна степен фиксира даже тематичните разпределения на

учителите. Училищата са много и най-различни, което изисква възможност за определена гъвкавост на учебната програма така, че тя да може да бъде адаптирана спрямо характерните особености на конкретно училище и неговите ученици – проблем, който не стои при университетите, защото там се пише отделна учебна програма за всяка отделна специалност във всеки отделен университет. Затова учебните програми в профилирана подготовка на училищата в никакъв случай не трябва да фиксират темите и съдържанието както в общообразователната подготовка – те трябва да са само насоки за това какво да се учи, а учителите и авторите на учебници да са свободни да изберат как да се случи това. Но дори това не е най-големият проблем, пред който са изправени българските училища, където тези предмети започнаха да се провеждат през учебната 2020–2021 година.

Основният проблем, който създава учебната програма по обектно-ориентирано програмиране за XI и XII клас [66] е това, че тя изначално е създадена буквално като един уведен курс по програмиране, който не цели да надгражда изучавани преди него знания, а да си бъде самодостатъчен. В тази програма последователно по дедуктивен път е заложено да се изучават най-основните знания, които не само че се дублират с голяма част от изученото в VIII клас (табл. 2), но и не отчитат факта, че при учениците от някои профили с информатична насоченост се очаква те допълнително да са учили информатика и като избираем предмет в IX и X клас.

Таблица 1. Дублиране на учебно съдържание между информатика за VIII клас и информационни технологии за X клас (профилирана подготовка)

УЧЕБНА ПРОГРАМА ПО ИНФОРМАТИКА ЗА VIII КЛАС (ОБЩО-ОБРАЗОВАТЕЛНА ПОДГОТОВКА)	УЧЕБНА ПРОГРАМА ПО ИНФОРМАТИКА ЗА XI КЛАС (ПРОФИЛИРАНА ПОДГОТОВКА), МОДУЛ 1. „ОБЕКТНО ОРИЕНТИРАНО ПРОЕКТИРАНЕ И ПРОГРАМИРАНЕ“
2.1. Интегрирана среда за визуално програмиране: - Отваря проект в интегрирана среда за визуално програмиране. - Редактира дизайна на графичния потребителски интерфейс на приложение в интегрирана среда за програмиране. - Запазва проект на приложение чрез средствата на интегрираната среда за програмиране. - Стартира приложение с графичен потребителски интерфейс чрез средствата на интегрираната среда за програмиране. - Разпознава основни компоненти на интегрирана среда за програмиране – графичен и текстов редактор, панел с контроли, панел за свойства на обект, панел за съобщения, панел за преглед на структурата на приложението.	4.1. Графични средства на интегрирана среда за създаване на графичен интерфейс: - Използва интегрирана среда за програмиране за създаване на графичен интерфейс - Посочва свойствата на елементи на графичен прозорец. - Създава елементи на графичен прозорец (етикет, текстово поле, бутон). - Задава настройки на свойствата на елементи на графичен прозорец. - Описва добри практики за именуване на идентификаторите на графични елементи. - Открива грешки при именуване на графични елементи.

2.3. Проектиране на графичен потребителски интерфейс: - Знае предназначението на основни контейнери и контроли – форма, етикет, текстово поле, бутон, диалогова кутия. - Разпознава основни свойства на графични обекти контроли – име, състояние, етикет, фон, настройка на шрифт и др. - Проектира несложна форма, съдържаща етикет, текстово поле, бутон. - Настройва основни свойства на форма, етикет, текстово поле и бутон. - Именува обекти контроли съгласно общоприета конвенция. - Задава функционалност на бутон, свързана с извеждането на статично съобщение в диалогова кутия.	4.2. Създаване на елементарен графичен интерфейс: - Реализира програмно достъп до и промяна на свойството „текст“ на графичните елементи. - Реализира програмно обработката на събитие по под-разбиране при натискане на бутон. 8.1. Проектиране на графичен интерфейс: - Проектира графична форма, представяща интерфейс на конкретно приложение. - Моделира свойства на компоненти от графичния интерфейс етикет, едноредово текстово поле, многоредово текстово област, бутон, компоненти за избор, кутия за изображения, меню, контейнери и др. - Спазва добри практики за именуване на идентификаторите на графични компоненти.
---	---

3.1.1. Тип низ • Познава правила за именуване на константи и променливи. • Декларира, описва и инициализира променливи и константи от тип низ. • Присвоява стойност на променлива от тип низ. • Въвежда и извежда данни от тип низ в/от текстово поле. • Извежда данни от тип низ в/от етикет. • Извършва конкатенация на низове. • Използва стандартни методи на интерфейсни компоненти за форматиране на текст.	
3.1.2. Целочислени типове данни: • Декларира, описва и инициализира променливи и константи от целочислен тип данни. • Използва вградени функции за преобразуване на низ в цяло число и обратното. • Въвежда и извежда данни от целочислен тип. • Използва различни целочислени типове данни. • Познава целочислените аритметични операции и техния приоритет. • Конструира аритметични изрази, съдържащи само целочислени данни, спазвайки синтаксиса и семантиката на конкретния език за програмиране.	5.1. Аритметични и логически оператори, релации за сравнение, оператор за присвояване и конкатенация: • Описва синтаксиса, семантиката и приоритета на основните оператори в конкретен език за програмиране. • Разбира асоциативността на оператори с равен приоритет. • Разбира спецификата на целочислената аритметика и аритметиката в смесени изрази. • Разбира семантиката на префиксен, инфиксен и постфиксен запис на оператор • Правилно прилага аритметични и логически операции, както и релации за сравнение в сложни изрази.

• Прилага и анализира резултатите от операциите – събиране, изваждане, умножение, деление, цяла част и остатък от целочислено деление. • Реализира модел за решаване на задачи, базиран на целочислени типове данни. 3.1.3. Реални типове данни • Декларира, описва и инициализира променливи и константи от реален тип. • Използва вградени функции за преобразуване на низ в реално число и обратното. • Въвежда и извежда данни от реален тип данни. • Използва различни реални типове данни. • Познава приоритетите на аритметичните операции при реални типове данни. • Конструира аритметични изрази, съдържащ реални типове данни, спазвайки синтаксиса и семантиката на конкретния език за програмиране. • Прилага и анализира резултатите от операциите – събиране, изваждане, умножение, деление. • Реализира модел за решаване на задачи, базиран на реални типове данни.	• Присвоява стойност на променлива. • Разпознава случаи на неявно преобразуване на типове данни. • Използва явно преобразуване на типове данни. • Конкатенира текстови данни. 5.2. Оценяване на логически изрази: • Разбира пълно оценяване на логически израз. • Разбира частично (съкратено) оценяване на логически израз.
---	---

• 3.1.4. Аритметични изрази и вградени математически функции. Приоритет на операциите: • Оценява числената стойност на аритметичен израз, записан на език за програмиране. • Записва аритметичен израз със средствата на език за програмиране. • Прилага и използва вградени в езика за програмиране математически функции – абсолютна стойност, повдигане на степен, закръгляване, извличане на цялата част на реално число. • Използва приоритет на операциите в аритметични изрази, съдържащи вградени функции. • Създава аритметични изрази, съдържащ различни типове данни, като се съобразява със съвместимостта им. • Описва синтаксис и семантика на оператор за присвояване. • Форматира изхода на реално число.	
--	--

3.3.2. Условен оператор: • Описва синтаксиса и семантиката на кратка и пълна форма на условен оператор. • Описва разклонен алгоритъм с помощта на условен оператор. • Използва условен оператор за проверка на коректността на входните данни за програма. • Използва условен оператор за обработка на свойства на радиобутон и поле за отметка.	6.3. Разклонени изчислителни процеси: • Разбира последователността на изпълнение на разклонен изчислителен процес. • Описва синтаксиса и семантиката на условен оператор „if“. • Разбира семантиката на вложени условни оператори. • Знае и може да прилага законите на Де Морган и закона за двойно отрицание над логически изрази. • Описва ефективно вложени условни оператори. • Описва синтаксиса и семантиката на оператор за множествен избор. • Реализира програмно условен преход със средствата на конкретен език за програмиране • Конструира ефективни условни конструкции • Проследява изпълнението на разклонен изчислителен процес
--	--

3.3.3. Вложени условни оператори: • Обяснява семантиката на вложени условни оператори в кратка и пълна форма. • Проиграва изпълнението на фрагмент на програма, съдържаща вложени условни оператори. • Записва синтактично и логически правилно вложени условни оператори. • Създава модел и алгоритъм за решаване на задача чрез използване на вложени условни конструкции. • Реализира модел за решаване на задача чрез използване на вложени условни оператори. • Заменя вложен условен оператор с единичен и обратно.	
---	--

3.4.1. Циклични алгоритми: • Посочва елементите на циклична конструкция – инициализация, тяло на цикъла и условие на цикъла. • Оценява необходимостта от използването на алгоритми с циклични конструкции с предусловие и постусловие • Записва синтактично и логически правилно оператори за цикъл с предусловие, постусловие и управлявани от брояч. • Прилага алгоритми с циклични конструкции за проверка на входни данни. • Оценява необходимостта от използване на алгоритми с циклични конструкции с условие или управлявани от брояч. • Преобразува програмен код, съдържащ циклична конструкция, управлявана от брояч в циклична конструкция или управлявана от условие. • Открива синтактични и логически грешки в програмния код на алгоритми с циклична конструкция. • Прилага циклични алгоритми за управление на графичен потребителски интерфейс.	6.4. Циклични изчислителни процеси: • Описва синтаксиса и семантиката на оператори за цикъл в конкретен език за програмиране. • Избира подходяща циклична конструкция за решаване на конкретен проблем. • Може да моделира процес описан с една циклична конструкция чрез друг вид циклична конструкция. • Реализира програмно циклични процеси със средствата на конкретен език за програмиране. • Проследява изпълнението на цикличен изчислителен процес. • Разбира семантиката на вложени циклични конструкции.
--	---

Хубаво е да се отвори една голяма скоба и да се каже, че често в учебните програми се дефинират едни неща на теория, но се получава съвсем различна реализация на практика. Така например на теория би трябвало избираемите учебни предмети да са

няколко различни и да водят до мотивиран избор за различна специализация в съответния профил, но на практика често изборът на учениците е сведен до един или два предмета с общ хорариум, равен на задължителния минимум за записване, което ги прави практически задължителни. Ако погледнем избираемите учебни предмети (ИУЧ) в някои математически гимназии, ще видим:

За профил „Софтуерни и хардуерни науки“ в Националната природо-математическа гимназия „Акад. Любомир Чакалов“ в гр. София, Природо-математическа гимназия „Васил Друев“ в гр. Велико Търново и Природо-математическа гимназия „Иван Вазов“ в град Димитровград, както и за профил „Математически“ в Природо-математическа гимназия „Яне Сандански“ в гр. Гоце Делчев в учебния план за IX клас за 2019–2020 г. в раздел „Б“ (избираеми предмети) присъства само и единствено учебният предмет „Информатика“ с годишен хорариум 54.

- За профил „Математически“ в Националната природо-математическа гимназия „Акад. Людмил Чакалов“, Математическата гимназия „Баба Тонка“ в Русе, Математическа гимназия „Д-р Петър Берон“ – гр. Варна, Профилирана природо-математическа гимназия „Акад. Иван Ценов“ – гр. Враца, Математическа гимназия „Гео Милев“ – Плевен, за профил „Софтуерни и хардуерни науки“ в Профилирана математическа гимназия „Константин Величков“ – гр. Пазарджик, както и профили „Математически“ и „Софтуерни и хардуерни науки“ в Профилирана природо-математическа гимназия „Никола Обрешков“ – гр. Разград, избираемите предмети са два, но отново общият брой часове от годишния хорариум са 54 – математика с 18 часа и информатика с 36 часа.
- За профил „Софтуерни и хардуерни науки“ в Математическа гимназия „Гео Милев“ – гр. Плевен и в Профилирана Природо-математическа гимназия „Акад. Иван Ценов“ – гр. Враца, за IX клас избираемите учебни предмети от раздел „Б“ отново са два, но този път са информатика (36 часа) и информационни технологии (18 часа), отново с общ годишен хорариум 54.

- Природо-математическа гимназия „Яне Сандански“ в гр. Гоце Делчев предлага повече от два избираеми предмета в профил „Софтуерни и хардуерни науки“ за IX клас – това са информатика (90 часа), информационни технологии (72 часа) и математика (72 часа), като там прави впечатление, че общият хорариум от избираеми предмети (234) е значително по-голям от всички разгледани.
- В общи линии е пъстро положението с възможности за избираеми предмети в специализираните училища професионална техническа гимназия – гр. Варна, Технологично училище „Електронни Системи“ – гр. София, Професионална гимназия „Васил Левски“ – гр. Ямбол, новата Професионална гимназия по компютърно моделиране и компютърни системи – гр. Варна, и др. В съответните профили на такива професионални гимназии избираемите предмети са много и информатиката се явява в повече от един от тях в различни форми. Тези училища определено имат съвсем самостоятелна визия и е много трудно да бъдат съпоставяни и сравнявани с профилите на традиционните математически гимназии. В общи линии за всяка професионална гимназия трябва да се прави отделен анализ – нещо, което е извън обхвата на настоящия труд. Трябва да се знае обаче, че макрорамката на задължителното образование би трябвало да покрива нуждите и на тези училища.
- В други математически гимназии пък информатиката е изчезнала напълно в IX клас от ИУЧ. Такъв е случаят със Софийската математическа гимназия, Профилираната природо-математическа гимназия – гр. Ловеч, Профилираната природо-математическа гимназия „Екзарх Антим I“ – град Видин и Природо-математическата гимназия „Атанас Радев“ в гр. Ямбол (в последната това е факт дори в техния профил „Софтуерни и хардуерни науки“ – нещо, което изглежда изключително странно). При всяка от тях обаче предметът е наличен като факултативен учебен предмет (ФУЧ). В Софийската математическа гимназия напри-

мер Иванка Зангочева преподава програмиране на C++ в IX клас, а Десислава Рачева води часове с програмиране на C++ дори в V и VI клас.

Без да давам подробно описание, подобно е положението и с информатика като избираем предмет в X клас. Това далеч не са всички математически гимназии в България, но тенденцията видимо е ясна – най-масовият случай е ИУЧ да са със силно редуциран списък и информатиката обикновено присъства в него. Систематизирано направеното сравнение е показано в таблица 3. Освен това там, където няма информатика като ИУЧ, винаги присъства под някаква форма като ФУЧ.

Таблица 3. Дял на учебния предмет „Информатика“ от избираемите дисциплини от раздел „Б“ на учебните планове за IX клас на някои математически гимназии

Училище	Град	Профил	Общ брой избираеми предмети за IX клас	Хорариум за информатика/общ хорариум по ИУЧ
Национална природо-математическа гимназия „Акад. Любомир Чакалов“	София	Софтуерни и хардуерни науки	1	54/54
Природо-математическа гимназия „Васил Друмев“	Велико Търново	Софтуерни и хардуерни науки	1	54/54
Природо-математическа гимназия „Иван Вазов“	Димитровград	Софтуерни и хардуерни науки	1	54/54
Природо-математическа гимназия „Яне Сандански“	Гоце Делчев	Математически	1	54/54

Национална природо-математическа гимназия „Акад. Любомир Чакалов“	София	Математически	2	36/54
Математическа гимназия „Баба Тонка“	Русе	Математически	2	36/54
Математическа гимназия „Д-р Петър Берон“	Варна	Математически	2	36/54
Профилирана природо-математическа гимназия „Акад. Иван Ценов“	Враца	Математически	2	36/54
Математическа гимназия „Гео Милев“	Плевен	Математически	2	36/54
Профилирана математическа гимназия „Константин Величков“	Пазарджик	Софтуерни и хардуерни науки	2	36/54
Профилирана природо-математическа гимназия „Никола Обрешков“	Разград	Математически	2	36/54
Профилирана природо-математическа гимназия „Никола Обрешков“	Разград	Софтуерни и хардуерни науки	2	36/54
Математическа гимназия „Гео Милев“	Плевен	Софтуерни и хардуерни науки	2	36/54
Профилирана природо-математическа гимназия „Акад. Иван Ценов“	Враца	Софтуерни и хардуерни науки	2	36/54
Природо-математическа гимназия „Яне Сандански“	Гоце Делчев	Софтуерни и хардуерни науки	3	90/234

Изводите от така направения преглед са ясни – масовата практика е де юре избираемите предмети да се превръщат де факто в задължителни, а информатиката обикновено е сред тях в IX и X клас за съответните профили. Това показва, че има голяма обществена потребност от този вид знания – училищата се опитват да отговорят на пазарното търсене, което им налагат родителите дори в случаите, когато задължителните учебни програми не поставят подобно задължение. По същество това често се проявява като бламиране на идеята в новите учебни програми да има избираемост – повечето училища се лишават от принципа на вътрешната конкуренция в името на започване на задължително по-ранно профилиране. Това, разбира се, поставя под въпрос и самата концепция за двустепенна система в средното образование (разделянето между VIII–X и XI–XII клас) като цяло, но такъв философски анализ би бил извън рамките на настоящия труд.

Впрочем проблемът с избираеми предмети, които предхождат задължителна учебна програма, не е нов за българското образование. Например дълги години беше наличен в математиката, където профилираната подготовка по тригонометрия предхождаше задължителната програма, в която се учеше тригонометрия. Такъв проблем обаче се появява за първи път в явен вид при учебния предмет „Информатика“. Именно масовото въвеждане на ранно профилиране от страна на гимназиите създаде и неизбежен конфликт между учебните програми по информатика за VIII и XI клас и избираемите предмети в IX и X клас.

Според официалните правила не се допуска теми от ИУЧ да дублират теми от задължителна и профилирана подготовка. Практиката обаче показва, че това не само не се спазва, а, както вече се видя, е налично дори в учебните програми, утвърдени от експерти в МОН. Така проблемът с дублиране на учебно съдържание става много дълбок. Например в Националната природо-математическа гимназия „Акад. Любомир Чакалов“ с профил „Софтуерни и хардуерни науки“ в IX и X клас избираемите учебни часове (ИУЧ) по информатика, които (както вече беше споменато) за съответния профил на практика са превърнати в задължител-

ни, се базират на учебното съдържание от задължителноизбираемите и свободноизбираемите часове по информатика от същото училище, но от предишните им учебни програми по старите учебни планове. В ИУЧ там се изучава обектно-ориентирано програмиране на C++, което се надгражда с изучаване на структури от данни в програмирането. Реално има практически пълно припокриване на учебно съдържание с модули 1 и 2 от XI и XII клас, профилирана подготовка [66], с информатика като задължителна дисциплина в VIII клас и ИУЧ в IX и X клас – разликата се явява само в езика за програмиране (в XI и XII клас се преминава на Java), а заложените фундаментални знания са практически едни и същи.

Оказва се, че ако трябва учебните програми да бъдат следвани стриктно, знанията няма да се надграждат, а ще се дублират, макар и с различно технологично средство. Смятам, че значително по-добре би било за учениците в профила на училището да се инвестира в повече екипни практически задачи в горния гимназиален етап, с които натрупаните знания да се упражняват по време на профилираната подготовка. Всъщност съм убеден, че колегите правят точно това. Също така би било значително по-добре сегашният модул 3 от [66] (изучаване на релационни бази от данни) да не стои изолиран в отделен срок чак в XII клас, а да бъде смъкнат в XI клас и да бъде обвързан с програмирането, което учениците са учили в VIII, IX и X клас. Така би станало възможно в XII клас да се направи систематизиране на знанията и учениците да се фокусират в изработването на цялостен практически полезен и сложен програмен продукт, при това изучавайки и (забележете) **обвързвайки** две различни парадигми на програмиране.

За съжаление, подобни възможности не са предвидени и няма как да бъдат – не е реалистично учебните програми за XI и XII клас да бъдат генерализирани по начин, по който да отговарят на всички училища. Например ако бъдат напаснати спрямо нуждите на НПМГ, те автоматично ще станат невалидни за други училища, където в техните ИУЧ е наложена различна практика и се изучават различни неща.

Естествено, може веднага да се контрира с логичен валиден аргумент, че училищата трябва да променят своите учебни програми в ИУЧ по информатика за IX и X клас спрямо наложената им профилирана подготовка в XI и XII клас така, че последната да стане адекватна за тях. Но трябва ли избираемите предмети да отидат далеч от приложното програмиране, за да не влизат в конфликт със задължителната програма от по-горните класове? Наистина ли ранното профилиране, което видимо почти всички желаят, е толкова вредно? Или ако все пак в тези ИУЧ има програмиране, да бъде ли то с фундаментално различна от обектно-ориентираното програмиране парадигма, за да не влиза в конфликт с по-горните задължителни курсове, и как това ще се отрази на учениците?

Създаденият проблем е фундаментален, защото чрез него се нарушава един от основните принципи в профилирането – тясната специализация чрез избираеми предмети трябва да е последен етап от профилирането, а не да предхожда задължителните учебни програми. Слагането на избираеми предмети в IX и X клас, които предхождат задължителни в XI и XII клас, е системен проблем! Може да се направи справка с изградените традиции във висшето образование, където няма да се намери изключение от правилото, че повечето задължителни предмети са разположени в първите две-три години от обучението на студентите, а по-големите хорариуми, отделени за избираеми дисциплини, се появяват в по-горните курсове. При сегашните учебни планове за математическите гимназии в средните училища подобна хронология не е спазена и те са сбъркани на чисто философско ниво.

Една от най-вероятните причини за създаването на този организационен проблем е прекалено стегнатият хорариум и малкото налични часове в IX и X клас измежду целия куп от други учебни предмети, които трябва да бъдат изучавани задължително. Това се случи заради въвеждането на двустепенната система в средното образование (тоест в резултат от новото НВО в X клас). Бяха въведени нови задължителни учебни предмети в общообразователна подготовка, които заеха допълнителен хорариум („Гражданско об-

разование“), а при други предмети съдържанието от XI и XII клас беше смъкнато надолу към IX и X, за да може да се приключи обучението по тях в първата степен на средното образование (последното от своя страна, разбира се, изиска и повишен хорариум по тези предмети). В крайна сметка в IX и X клас не остана възможност за ранно профилиране на учениците, а, както вече беше споменато, тяхното желание е това да не бъде така. Смятам, че именно от това произлезе фактът, че ИУЧ са де юре избираеми, но де факто задължителни учебни предмети в повечето математически гимназии. Информатиката, бидейки профилиращ, а не задължителен общообразователен предмет, е един от засегнатите от тази недалновидна политика на стремеж да се учи колкото се може повече по всичко. Смятам, че претрупването с учебно съдържание е именно генераторът на споменатите проблеми.

Въпросът оттук насетне е, ако се приеме, че ситуацията с учебните планове е неизбежна (т.е. не може да се освободи хорариум в IX и X клас, така че профилираната подготовка по информатика да се премести в тях, а избираемите часове да останат във втората част на гимназиалната степен, за да може свободно да надграждат изученото преди тях), дали все пак има някакъв начин ефектът от този проблем да бъде намален или заличен. Отговорът според мен е положителен, но при всички случаи ще изисква една или друга степен на автономия за училищата относно изготвянето на учебните програми по профилиращи предмети. Най-общо може да се набележат пет възможни генерални решения:

1. Учебните програми по информатика за XI и XII клас да бъдат пренаписани по начин, по който се залагат много по-малко изисквания към нови знания и умения, а да се постави значително по-силен фокус върху упражнения, работа по проекти и по-адекватно вплитане на сегашния модул 3 с приложно програмиране, което от своя страна ще включва не само реструктуриране на темите от него, но и преместването му в по-ранен етап.
2. Да бъдат разработени две различни учебни програми по информатика за XI и XII клас – една следваща принципите

от предложениния вариант 1, която е предназначена за ученици, които са учили информатика като избираем предмет в IX и X клас, и втора, за такива, които не са имали информатика в ИУЧ/ФУЧ.

3. За ученици, които са изучавали информатика в IX и X клас като избираем предмет, да бъде дадена възможност от страна на училището да се изготви индивидуална учебна програма по информатика в XI и XII клас, която да отговаря по-точно на техните индивидуални потребности и ниво на подготовка.
4. Учебната програма за профилирана подготовка по информатика за XI и XII клас да стане само примерна, а училищата да могат сами по свое усмотрение да я променят и адаптират. Учебният предмет обаче да остане задължителен.
5. Задължителната профилирана подготовка по информатика за XI и XII клас да отпадне напълно дори за профила, а училищата да могат да я предлагат като избираем предмет със своя собствена учебна програма.

Вариант 1 е лесно реализируем спрямо сегашната нормативна база. По същество той представлява решение, с което учебните програми за профилирана подготовка, които не предхождат, а следват избираеми предмети, да се превръщат по-скоро в общи насоки за учителите какво трябва да правят, вместо да бъдат фиксирани рецепти, които трябва да се следват. Така учителят получава свободата сам да прецени какво и как да преподава на учениците си, следвайки методите на изследователския подход в обучението и „учене чрез правене“. С мои студенти и дипломанти много често експериментираме с тези подходи (например [16]), а в учебника по информационни технологии за VIII клас [44] и свързаната книга за учителя [45] с колегите изключително силно наблегнахме именно на тях. Една такава учебна програма по информатика би включвала малко време за изучаване на нови знания и много практически упражнения.

Вариант 2 по същество надгражда идеята от предишния вариант, като дава възможност програмата да бъде по-адекватна в

зависимост от това какво са избирали децата в ИУЧ от IX и X клас. Наличието на алтернативни програми с различни степени на свобода би улеснило учителите, но ще затрудни сериозно авторите на учебно съдържание, които ще трябва да изготвят значително по-комплексни учебници (едва ли издателствата ще инвестират в по два различни учебника, при положение че очакваният тираж на единия от тях тъй или иначе ще е много малък). Вероятно реализирането на такъв вариант ще представлява и юридически казус и няма да е лесен за приемане (реално става дума за реализиране на адаптивност в учебен план спрямо това какви избираеми предмети са изучавали учениците), но със сигурност не е нереализируем.

Възможен за реализиране би бил и вариант 3, но той започва да прилича на вратичка за заобикаляне на иначе задължителните програми, което само по себе си носи свои рискове от злоупотреби. При него се дава само частична автономия за профилираната подготовка в училищата и съответно се реализира вътрешна гъвкавост. Идеята изначално е добра, ако сме идеалисти, защото в същината си тя е да се постигне висока степен на индивидуализация в училищното образование – вместо всички да учат едно и също, учебното съдържание ще се променя спрямо индивидуалните потребности на младежите. На практика обаче ще има немалка критика на подобно предложение – веднага ще се повдигне въпрос, че някои училища могат да злоупотребят с така дадената им свобода, като съответно започнат масово да изготвят индивидуални учебни програми за свои ученици, неотговарящи на заложените от държавата цели. Подобно нещо по-скоро би било изключение, но не трябва да бъде пренебрегвано като проблем, свързан с потенциално широко негативно медийно отразяване и изключително лош обществен отзвук. Много често в България частни случаи се употребяват за остра критика към системата като цяло, а малки проблеми се хиперболизират с лека ръка – тази характерна национална особеност не трябва да се negliжира.

Вариант 4 би бил нещо като „отваряне на кутията на Пандора“ за българското образование като цяло – досега винаги учебните програми, „спускани“ от МОН, са били със задължителен харак-

тер. От това училищата да могат сами да си изготвят учебните програми, естествено, може да се очакват както положителни, така и отрицателни практически примери – на едни места ще бъдат демонстрирани добри иновативни практики, докато на други ще бъде отчетен провал. При всички положения може да се очаква негативен краен отзвук в обществото – много хора са по-склонни да преувеличават проблемите и да омаловажават постиженията. Въпреки вероятно неизбежния негативен образ на подобна реформа, това няма да е лош резултат за образователната система като цяло. Напротив, по-скоро би подобрило положението значително, защото ще се даде голямо поле за конкуренция, а добрата конкуренция в дългосрочен план винаги поражда качество. За постигане на въпросното качество обаче са нужни дълги години за формиране на традиции и отсяване на добри и лоши практики – години, в които не е ясно дали ще има достатъчно политическа воля подобна реформа да издържи на неизбежните критики.

Вариант 5 би бил по-скоро деструктивен в условията на реалностите в България. Рискът учебният предмет просто да изчезне от още повече училища (въпреки повишеното пазарно търсене) не е малък. Има очевидна пазарна потребност и училищата правят всичко възможно да отговорят на нея, но това невинаги може да се получи, особено когато училището е подвластно на централно планиране. Причината е, че училищата в България рядко се конкурират на базата на пазарни принципи – намесват се редица други фактори, като например непазарно лимитиране на брой паралелки от страна на РУО, лимитирани възможности за финансово стимулиране на учители, за чийто учебен предмет има недостиг на квалифицирани специалисти, и други. Липсата на учители по информатика дава потенциал тенденцията за изчезване на информатика да се разрасне и в посока дори към математическите гимназии. Освен това неминуемо ще остане въпросът: Каква е тази профилирана подготовка, в която реално няма заложили профилиращи учебни предмети? Свободата в образованието принципно е хубаво нещо, но тя все пак трябва да следва определените национални стратегии. Затова смятам, че подобен вариант е еднозначно лоша идея.

При всички положения смятам, че текущите учебни програми по информатика за XI и XII клас не отговарят адекватно на профилираната подготовка, за която са предназначени. Това не е защото те сами по себе си са лоши, а защото не се вписват адекватно в учебните планове. Нито е извършен синхрон с учебната програма по информатика за VIII клас, която вече влезе в сила, нито е предвидена каквато и да е гъвкавост за ученици, които са изучавали информатика като избираем предмет в IX и X клас. Дали някога някаква частична или пълна автономия за учебни програми ще бъде дадена на училищата, е въпрос на политическо решение. При всички положения смятам, че учебните програми по информатика за XI и XII клас трябва да бъдат ревизирани основно – пак повтарям: не защото са лоши сами по себе си (напротив, те биха били едни прекрасни уводни курсове в някой университет!), а защото не се вписват по нормален начин в текущите планове. Същото важи и за учебните програми по информационни технологии, защото при тях не само липсва адекватен синхрон с обучението по информатика, но и тези за X и XI клас стоят крайно неадекватно и пораждат недоволство сред учителите в цялата страна. За щастие, със стратегията [41] е видимо, че скоро ще бъдат предприети стъпки към отстраняване на този проблем.

Иначе като цяло смятам, че визията на учебната програма по информатика за VIII клас е много добра. Единственият проблем с нея е свързан с недостатъчно време за упражнения. С оглед на това, че учениците вече ще са учили компютърно моделиране в прогимназията, бих очаквал те да имат достатъчно ниво на теоретична подготовка по алгоритми, т.е. това, което би могло да се приеме като реформа, е обучението в VIII клас да се облекчи от подобни теми и да се фокусира изцяло в приложното програмиране. Именно тази теза и ще застъпя.

Приемам, че наложената практика в съответния профил в IX и X клас да се учи задължително информатика, ще се запази, както и че най-често това ще продължи да включва упражняване, обобщаване и разширяване на изученото в VIII клас. От това би трябвало да следва, че масовият случай ще бъде в началото на XI клас един

ученик от математическа гимназия да има достатъчно подготовка по обектно-ориентирано програмиране, но дебело подчертавам, че това не е задължително да е вярно. Именно затова ще предложа следната генерална реформа на учебните програми за профилирана подготовка за XI и XII клас, която би била опит за разрешаване на набелязаните проблеми:

- В XI клас да започне изучаване на различна парадигма за програмиране – декларативно чрез SQL и съответно изучаване на релационни бази от данни, като то не се изучава самостоятелно и самодостатъчно, а да е обвързано с изученото преди това обектно-ориентирано програмиране, чрез създаване на сравнително прости информационни системи. За ученици, които не са затвърдили и усвоили допълнителни знания и умения по ООП в IX и X клас, да бъде оставена възможност със същата програма за преминаване през същите знания със значително по-опростен курс със система като Microsoft Access или подобна, където приложните програми се създават изключително лесно чрез формуляри. В този смисъл програмата за профилирана подготовка в XI клас трябва да бъде изключително адаптивна и да може да се прилага автоматично към различни видове ученици.
- В XII клас обучението да е изцяло проектно, без задължително изучаване на нови технологии. По този начин ще се извърши систематизиране и обобщаване на вече изучените знания, а учениците ще добият едно от много важните умения в съвременния свят при разработка на софтуерни продукти – работата в екип.

ПРИМЕРЕН УЧИЛИЩЕН КУРС ПО ИНФОРМАТИКА ЗА VIII КЛАС

Прекалената теоретизация на учебното съдържание винаги е била един от факторите децата да смятат даден предмет за „скучен“. Затова смятам за изключително важно в училищните курсове по информатика да се наблегне много повече на индуктивно натрупване на практически знания, отколкото на заучаване на формални дефиниции с дедуктивно изложение на учебния материал. Систематизирането, теоретизирането и разширяването на натрупаните знания може спокойно да останат за полето на висшите училища. В частност вътърпителният училищен курс по информатика в училище не трябва да е някакво подмножество на университетския „Увод в програмирането“, да е фундаментално различен, защото се работи с хора от съвсем различна възрастова група и има различни цели: не да ги въвежда в науката информатика, а да предизвика чисто практичен интерес към професия „програмист“, като да им покаже как могат да се правят лесни, но ефективни примери от сферата на приложното програмиране. Смятам, че в курса по информатика учениците не бива да бъдат засипвани с огромни количества информация за различни програмни конструкти или да бъдат форсирани към изучаване на сложни алгоритми – обучението трябва да се фокусира към създаване на полезни програми „като от реалния свят“ по пътя на най-малкото съпротивление (с колкото се може по-малък набор от теоретични знания). Така ще се спре с претрупването с материал и предметът ще е по-атрактивен.

Друг важен момент е да се позволи учебните програми да са максимално адаптивни. Информатиката е изключително приложна наука, чийто продукт се използва за решаването на задачи от много други области. Поради тази причина не трябва да ограничаваме учителите от възможността да създават силни междупредметни връзки според профила на учениците, а напротив – трябва да ги насърчаваме да го правят. Поради естествената си близост с математиката, много често по програмиране децата се засипват основно с примери за това как може с помощта на език за програмиране да се напишат алгоритми за решаване на математически задачи. Трябва да знаем, че информатика в VIII клас се учи и от деца от други профили – биология, физика, химия, т.н. За тях би било много по-смислено да се използват междупредметни връзки, които са от техния профил, а не само с математика!

Друга визия, която застъпвам, е, че теоретичните знания трябва да се дават „при настъпила нужда“, а не преди решаване на задачите. В училище няма голяма полза от изложение тип лекция-упражнение, т.е. първо да обясняваме нещата теоретично, а чак след това да ги упражняваме. Знанията могат да се натрупват по естествен път чрез т.нар. „изследователски подход в обучението“ – под методическото ръководство на учителя децата сами откриват теоретичните знания, когато се достигне до ситуация, в която те станат неизбежно нужни. Това, разбира се, е по-трудно за реализиране от методическа гледна точка и изисква голяма доза умение за импровизация от страна на учителите, но за сметка на това уроците стават значително по-интересни и по-атрактивни за учениците. Винаги съм застъпвал това свое виждане при преподаването на училищните курсове по ИТ и информатика, които водя в Софийския университет при бакалаври, магистри и в следдипломна квалификация. Всъщност дори тях ги водим традиционно без явно разделение между лекция и упражнение, а ги обединяваме в общ блок. Трябва да се признае, че това се случва като се възползваме активно от факта, че групата от студенти в специалност „Математика и информатика“ във факултета, в който работя, е само една.

Като цяло подкрепям напълно визията на сегашната учебна програма по информатика за VIII клас да се създават директно програми с графичен потребителски интерфейс в среда за обектно-ориентирано програмиране. До голяма степен това, което ще предложи тук, се припокрива с визията на авторите на споменатата учебна програма. Само някои части от теми предлагам да отпаднат (тези свързани с двоична бройна система, както и темата за алгоритми), т.е. така курсът ще се фокусира изцяло в приложно обектно-ориентирано програмиране. Ще представя един примерен ход на курс с прилагане на изследователски подход в обучението, който включва известно (не голямо) преструктуриране на съществуващата учебна програма, като до известна степен (но не напълно) то се явява и нещо като примерно тематично разпределение с методически бележки. Все пак като учебна програма в по-голямата си част съдържателно предположението ми припокрива съществуващото. Освен това ще се съобразя със стратегията от [41] и свързаните с нея програми [56], [57] и [58], и ще предположа, че до VII клас учениците са добили знания за алгоритми и имат основи от някакво приложно програмиране (достатъчно е дори да е било само блоково във визуална среда със Scratch, но би било хубаво да е имало и досег с писане на програмен код на скриптов език, както е заложено в стратегията).

Таблица 4. Тема 1 от примерен училищен курс по информатика за VIII клас

Тема 1. Основи за създаване на приложна програма с графичен интерфейс	
Необходимо учебно време: 3 учебни часа в комбиниран урок	
Знания и умения	Обяснение
Създаване и стартиране на проект в интегрирана среда за визуално програмиране. Добавяне на някои графични обекти с помощта на интегрираната среда	Изучаване на основните функционалности за някоя интегрирана среда за визуално програмиране са неизбежни знания, без които дори не може да се започне. Децата трябва да се научат как да създават проект в средата, да стартират програма с нейна помощ, как да ползват познати от ежедневието готови визуални обекти, които им се предоставят (бутони, текстови полета, т.н.). Съвременните среди за визуално програмиране като Netbeans, Eclipse, Visual Studio т.н. са изключително сложни и предлагат огромен набор от възможности, но не е нужно да се впускаме в излишни подробности – учениците трябва да изучават само това, което им е непосредствено необходимо, за да решат поредната поставена пред тях текуща задача. Няма нужда и всичко, което се използва, да се обяснява в детайли – например, ако изберем Java, няма нужда да обясняваме в подробности какво е package, class, main(), как работи JDK, какво е Ant/Maven/Gradle, каква е разликата между application, library, modular project и т.н. по време на създаването на нашата първа програма – достатъчно е да се каже по изключително елементаризиран начин буквално с 2–3 думи, и то само това, в което сме се фокусирали да използваме.
Проектиране на графичен потребителски интерфейс – създаване на нова и/или изкопиране на визията на съществуваща, добре позната за учениците приложна програма	Проектирането на графичен потребителски интерфейс обикновено е естествената първа стъпка от създаването на целевите програми. Това е и нормалното продължение към темата за интегрираната среда за визуално програмиране – възползваме се именно от нея, за да можем бързо и лесно да „сглобим“ визуалната част на една от първите програми, които ще създаваме.

Имена, типове и характеристики на графичните обекти	В началото няма съществена полза от това да показваме кода, който средата генерира – учениците все още нямат натрупани знания, за да го разберат. Те могат да се запознаят с думата „обект“ в елементаризиран вид, като им се каже, че всички наредени по екрана графични компоненти ще бъдат наричани по този начин – това е далеч от обема на понятието, който стои в обектно-ориентираното програмиране, но този обем ще бъде разширяван впоследствие, като се правят препратки назад. Разглеждайки някои от взетите наготово графични обекти, може да се покаже, че те споделят много общи неща – всеки има уникално име, тип (клас) и набор от характеристики. В никакъв случай не е очаквано да се впускаме в изучаването на всички характеристики – достатъчно е да се покажат няколко от често използваните, както и че често различни по тип и дори визуално коренно различни обекти имат сходни характеристики.
Основни действия с обекти с графичен интерфейс – get и set методи/функции	След проектирането на графичния интерфейс, стартира съществената част от решаването на задачите, а именно написване на програмен код, с който наредените графични компоненти да започнат да работят (например бутонът да върши някаква полезна работа). Най-естественото в случая е да се покаже, че графичните обекти притежават get и set методи/функции, с които съответно можем да четем и променяме техни основни характеристики (текста им, позицията им на екрана, цвета им и т.н.). Важно е да е проектиран интерфейс за практически смислена задача, а не да се показват функционалности само заради тяхната демонстрация – по този начин се очаква обучаемите да получат допълнителна мотивация за учене. Допустимо и до известна степен очаквано е създаденото в тази тема да се използва като входна точка за следващата.

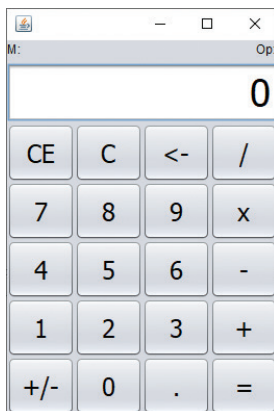
Съхраняване и пренасяне на проект от един компютър на друг	В края на часовете е желателно учителят да покаже как учениците биха могли да прехвърлят проект от един компютър на друг. Подходящо е например той да го направи с разработения от самия него проект, като го прехвърли на работния компютър на ученик, който например е изпитал трудности или е получил малко по-различна визия на програмата си спрямо тази на останалите. Това знание е много полезно за бъдещата работа, защото така ще стане значително по-лесно учениците да синхронизират проектите помежду си и да могат да продължават работата си от една и съща изходна точка.
--	---

Вижда се ясно, че предложената тема 1 е нещо като естественото необходимо условие да се започне да се програмира каквото и да е във визуална среда. Освен основно запознаване със средата, с която ще се работи, се показват стандартни обекти в графичен интерфейс – такива, с които учениците са свикнали от ежедневната си работа с всяка приложна програма на почти всяко компютърно устройство с графичен екран, както и че всеки обект си има име, тип, характеристики и набор от възможни действия. Препоръчвам в основния ход на урока учителят да демонстрира чрез мултимедиен проектор работа директно в интегрираната среда за визуално програмиране, а учениците да повтарят действията му. В никакъв случай не бих съветвал да се използват презентации – много по-добре е учениците да следват на живо и да пресъздават това, което учителят прави. Самостоятелната работа на учениците в такъв урок се свежда до повтаряне на вече демонстрирана от учителя функционалност (например вече е показано как да се добави и настрои визията на един бутон, а учениците да направят същия в своя собствен проект). Трикове за значително повишаване на бързодействието, като например копиране и вмъкване на вече направени обекти (copy-paste), е силно желателно да бъдат усвоени от учениците, но (подразбира се) едва след като са упражнили поне веднъж самостоятелно създаването на поне един обект.

Видимо е, че създаденото в тема 1 няма как да е работеща практически полезна програма – тя може да е само визуален изглед за бъдеща. Препоръката ми е създаденото от тема 1 да се използва като входна точка за тема 2. Това обаче крие своите подводни камъни – съвсем очаквано е поради несъвършенства в действията на учениците да се получи така, че техните проекти в крайна сметка да се различават значително. Основният проблем е, че те няма да се различават само с визията си, но и по имена на обекти и техни характеристики (дори ако отчетем най-малкото тривиални правописни грешки при писането на текст, бъркането на малки и главни букви, т.н.). Ако същите различаващи се проекти се използват в следващата тема, където започва писането на същински програмен код, съвсем очаквано е учителят да изпита огромни трудности, защото със сигурност ще има ученици, които ще преписват механично кода, а той ще се оказва, че няма да работи при тях.

За справяне със споменатия проблем най-често учителите използват подхода да разпространят създадения от тях проект като шаблон по работните компютри на учениците преди започването на следващите часове. Това е работеща и естествена практика, но тук бих препоръчал при наличие на време по възможност да се предприеме друг подход – чрез достъп по FTP или със споделена папка чрез локалната мрежа в класната стая, учениците да бъдат научени как сами да си прехвърлят текущото състояние на проекта от учителския компютър на техния собствен. Това ще отнеме известно време, но в дългосрочен план би спестило много усилия за случаите, в които някой ученик е изостанал много или по време на самостоятелната си работа е достигнал до проблемна ситуация (например грешка в кода), която учителя не успява да разреши бързо и на момента. Подобна практика би била полезна не само по информатика, но и при часовете по информационни технологии.

Една примерна заготовка с графичен интерфейс на компютърна програма, която е постижимо да бъде реализирана от страна на учителя и повечето ученици, но почти никога в „идеален“ вид от страна на всеки един ученик, е например графичен калкулатор.



Първоначалната реакция на всеки би била, че това е прекалено сложен графичен интерфейс за първа програма, но ако се замислите ще установите, че с копиране и вмъкване на бутони или групи от бутони реално той може да се създаде изключително бързо (буквално за няколко минути). Разбира се, напълно очаквано е част от учениците да не успеят да се справят докрай с всичко по подобен интерфейс – за някои размножаването на обектите с копиране и вмъкване, и особено тяхното именуване, би отнело значително повече време спрямо други и е нормално учебният час да не стигне за всеки. В крайна сметка обаче говорим за повторяеми действия, т.е. упражнение едни и същи много прости неща. Тук повторението на операциите до известна степен ще е и досадна задача, но е хубаво да се научи как може да се прави ефективно, защото ще пести огромно количество време в бъдеще.

Добре е учителят да покаже и да насърчи учениците да използват „трикове“ като маркиране, копиране и вмъкване с CTRL+C/CTRL+V не само на един обект, но и на цяла група обекти (например четири бутона по редове), промяна на характеристиките на много обекти накуп и преименуване на имената на променливите с навигация чрез клавиатурата и използване на бутон F2, вместо с поредни натискания с десен и ляв бутон на мишката (така няма да има преместване на ръката от мишка към клавиатура и обратно) и т.н. Това са все неща, които пестят изключително много време и

ще бъдат полезни при всяка следваща задача. При добра организация, подобен графичен интерфейс би могъл да бъде създаден в рамките на до два учебни часа напълно безпроблемно, но отново ще повторя уговорката, че не се очаква самите ученици да могат да го пресъздадат безгрешно и да успеят да го реализират напълно самостоятелно.

Относно операциите с обектите, в началото трябва да бъде поставен фокус върху много елементарни действия. Например да се реализира при натискане на бутон „1“ текста в полето да се смени от „0“ на „1“:

```
currentNumber.setText("1");
```

Съвсем елементарно е учениците да могат да го направят по аналогия с поне част от останалите цифри (няма нужда да го довършват за всички – достатъчно е да се използва еднократно за упражнение).

Друга възможна операция е при натискане на бутон „+“ в текстовото поле за памет да се запише текущата стойност на числото:

```
memory.setText(currentNumber.getText());
```

а при операцията да се запише текста „+“:

```
operation.setText("+");
```

Отново по аналогия учениците могат да направят „-“, „x“ и „/“, като при тях кодът ще е същият, а само текстът на операцията ще е различен. Няма нужда и няма как да се реализират всички бутони още в първия час – достатъчно е да се направят операциите на поне един-два, с което да се покаже, че съществуват get и set методи на обектите.

Учениците най-вероятно ще се възпротивят, че така написаните операции с цифровите бутони не вършат точно това, което се очаква от тях (при повторно натискане на бутон „1“, числото в текстовото поле не става „11“, а си остава „1“) – това обаче ще остане като отворен въпрос да го помислят и ще се реши като задача при следващата тема. Тук отбелязвам друга много важна основна методическа препоръка, зад която винаги се застъпвам в практиката си – в края на урока трябва да се оставят отворени въпроси

за следващия. По този начин част от децата ще са провокирани за самостоятелна работа въщи дори и без да е поставена задължителна домашна работа.

Веднага се вижда, че още в самото начало въвеждаме един никак не добър стил на програмиране – вместо да преизползваме кода и да напишем общ метод/функция за всички цифрови бутони, ние копираме и вмъкваме един и същ програмен код многократно. Хубаво е този проблем да се отстрани възможно най-бързо, но няма как това да се случи още в първите часове. Даже напротив – целесъобразно е учениците първо да се сблъскат с проблема, че при промяна на един изкопиран код, после при нужда от редакции се налага многократна промяна. При положение че функционалността на въпросните бутони все още не е реализирана докрай, това тъй или иначе ще бъде нещо, което предстои да бъде видяно непосредствено.

Много рядко, но все пак понякога се случва ученик да се справи изключително бързо с подобна задача – за такива обучаеми може да се предвидят допълнителни задачи, като например да се проучат допълнителни характеристики на обектите - да се смени фон, да се фиксира максимална, минимална и препоръчителна големина на основния прозорец, да се зададе заглавие (title) на основния прозорец, да се сложи икона на обект и т.н. Подобна практика учителят винаги да има подготвени допълнителни задачи е силно препоръчителна от методична гледна точка. Така вниманието на по-активните ученици е ангажирано през цялото време и се избягва проблема, в който те започват да скучаят и да пречат на другите.

Таблица 5. Тема 2 от примерен училищен курс по информатика за VIII клас

Тема 2. Условен оператор и текстови низове	
Необходимо учебно време: 3 последователни учебни часа в комбиниран урок	
Знания и умения	Обяснение
Променлива – име, тип и стойност Основни действия с текстови низове	Изучаването на типовете от данни е важно, но не трябва да се бърза с изреждането и демонстрирането им, а да се дават постепенно при настъпила нужда по време на решаване на практическа задача. В този смисъл обикновено първият тип, с който учениците се запознават, е текстовият низ – той идва по естествен път чрез съдържанието на текстовите полета (label, text field), които сме подредили в графичния интерфейс и съответно можем да четем и променяме. Тук например можем да покажем как се създава нова променлива от текстови тип, която получава като стойност текст, взет от графичен обект – естествено, се прави и препратка към предишната тема, в която се задаваха имена, типове и характеристики (по аналогия – стойности!) на графичните обекти. Отново няма смисъл от всеобхватно изучаване на всички действия, които можем да извършваме с текстовите низове, а трябва да се изучават само най-основните – тези, с които решаваме проблема от непосредствено поставената задача.

Условен оператор. Пропедевтика за изучаване на булев тип данни	Условният оператор е изключително фундаментален програмен конструкт, без който е много трудно да се създаде каквато и да е смислена програма. Отново няма особен смисъл да се инвестира сериозно време във формални дефиниции, а е напълно достатъчно да се демонстрира неговата употреба многократно първо с прости, а после и с по-сложни (с употреба на логическо “и” и/или “или”) примери. Идеята е на учениците да се припомнят знанията от компютърно моделиране и да се адаптират към новата среда. Относно изразите, които се оценяват, е достатъчно да се обясни на учениците, че се очаква те да връщат резултат „истина“ (true) или „неистина“ (false), като в зависимост от това изпълнението на програмата се прехвърля съответно в първия (на if) или втория (на else) програмен блок. В бъдеще, когато евентуално се изучава по-формално дефиницията за булев тип данни и възможните операции с него, учителят просто ще направи препратка към вече наученото и ще уточни за учениците, че те вече са използвали булевия тип данни многократно навсякъде, където са ползвали условен оператор. Напълно очаквано е учениците да не изпитат сериозен проблем с написването на по-прости изрази като „дали един текст да съвпада с друг“ например, но със сигурност ще имат проблеми при нужда от по-сложни изрази, които включват конюнкция и/или дизюнкция.
Метод/функция	Макар и по-теоретично, знанието за същината на методи/функции и проблемите, които се решават с тях, е важно да бъде въведено колкото се може по-рано. Учениците вече са употребявали и често ще упражняват два вида методи/функции – такива, които са без входни параметри и връщат стойност (get), и такива, които са с входни параметри, но не връщат стойност (set). Това е добра отправна точка за въвеждане за по-общия вид на понятието – входни, изходни данни, тяло с извършени операции и име. Добре е да се напише поне една смислена функция, която решава конкретен набелязан проблем. Не е добре обаче да се за дълбава с много примери – достатъчно на този етап е да се покаже само един пример. Така индиректно започва да се осъзнава и какво всъщност правят get и set методите, които се използват.

Мисля, че визията при тема 2 е пределно ясна – просто се надгражда по напълно естествен път наученото от предишната тема. Хубаво е в края на този урок да се достигне до напълно или почти завършена практически полезна програма. В контекста на примера с калкулатора, който дадох като възможен за тема 1, се появява огромно поле от програмни конструкти, решаващи множество сходни, сравнително прости задачи, в които хем се упражняват различни изрази за условен оператор, хем се натрупват допълнителни знания за обработка на текстови низове. Например:

1. Код на цифров бутон „1“:

```
String temp = currentNumber.getText();
if (temp.equals("0")) {
    currentNumber.setText("1");
}
else {
    temp = temp + "1";
    currentNumber.setText(temp);
}
```

2. Метод, който обединява действията на всички бутони от 0 до 9:

```
void digitBtnPressed(JButton btn) {
    String temp = currentNumber.getText();
    if (temp.equals("0")) {
        currentNumber.setText(btn.getText());
    }
    else {
        temp = temp + btn.getText();
        currentNumber.setText(temp);
    }
}
```

3. Бутон „CE“:

```
currentNumber.setText("0");
```

4. Бутон „C“:

```
currentNumber.setText("0");  
memory.setText(" ");  
operation.setText(" ");
```

5. Бутон „.“:

```
String temp = currentNumber.getText();  
if(temp.contains(".")){  
    return;  
}  
temp = temp + ".";  
currentNumber.setText(temp);
```

6. Бутон „+/-“:

```
String temp = currentNumber.getText();  
if(temp.equals("0")){  
    return;  
}  
if(temp.startsWith("-")){  
    temp = temp.substring(1);  
    currentNumber.setText(temp);  
}  
else{  
    temp = "-" + temp;  
    currentNumber.setText(temp);  
}
```

7. Бутон „<-“:

```
String temp = currentNumber.getText();  
temp = temp.substring(0, temp.length()-1);  
if(temp.equals("") || temp.equals("-")){  
    temp = "0";  
}  
currentNumber.setText(temp);
```

Вижда се, че говорим за сравнително еднообразни задачи, които използват изключително сходни една на друга езикови конструкции и си приличат, но във всяка от тях или се въвежда нещо

ново (нов метод върху текстови низ), или условният оператор се усложнява. Подобни задачи следват стриктно т.нар. „стъпаловиден подход“ за натрупване на знания, който е валиден не само за математиката [71] и в информатиката, а вероятно и за всички други науки. Освен това трябва да се спомене, че в никакъв случай не се цели перфекционизъм – може да се види спокойно, че немалка част от кода е свързана с изчистване на частни случаи, които могат да се пропуснат. Например в бутон „+/-“

```
if(temp.equals("0")){  
    return;  
}
```

би могло да се пропусне, ако приемем, че човек няма да натиска бутона, в случай че на екрана има числото „0“. Аналогично при бутон „.“

```
if(temp.contains(".")){  
    return;  
}
```

може да се пропусне, ако решим, че няма да се натиска бутонът два пъти за едно и също число. Такова опростяване на кода може да се приложи като тактика и за бутоните „+“, „-“, „×“ и „/“. Например „+“ би трябвало да премахва всички нули в края на число с десетична запетая, което би изисквало регулярен израз, готова библиотека или както неоптимално е показано в примера по-долу с цикъл (нещо, което е предвидено да се учи доста по-напред в учебната програма):

```
operation.setText("+");  
String temp = currentNumber.getText();  
if(temp.contains(".")){  
    while(temp.endsWith("0")){  
        temp = temp.substring(0, temp.  
length()-1);  
    }  
}  
if(temp.endsWith(".")){
```

```

        temp = temp.substring(0, temp.
length()-1);
    }
    memory.setText(temp);
    currentNumber.setText("0");

```

но също така може да се опрости до значително по-елементарната последователност от елементарни действия:

```

operation.setText("+");
String temp = currentNumber.getText();
memory.setText(temp);
currentNumber.setText("0");

```

с което не се грижим нито за числа, които имат десетична запетая и завършват с нули, нито за текстове, които завършват с десетична запетая.

Истината е, че в повечето случаи учениците дори не биха забелязали подобни частни случаи – те се появяват само при нестандартна употреба на калкулатора, т.е. когато потребителят не спазва нормална последователност от действия (например не е нормално да искаме да обърнем знака на числото 0 и да го направим отрицателно – ще извършваме подобно действие само ако целенасочено търсим да намерим пропуск в програмата). От методическа гледна точка препоръчвам да се започне от по-простите и бързи решения, без да се акцентира върху частните случаи, въпреки че те няма да са перфектни и ще водят до несъвършен краен резултат. Изглаждането на грешките и вглеждането в детайлите може да се постигне в бъдеще, когато вече има натрупан инструментариум и учениците имат увереност в използването му. Ако целим перфекционизъм от самото начало, това би затрупало децата с прекалено много теоретични знания, би отложило постигането на практически полезните резултати и съответно би направило учебния предмет скучен. Изразът „човек се учи от грешките си“ е валиден и в програмирането и няма нищо лошо да се допускат грешки, стига в бъдеще, когато вече има достатъчно натрупан опит, да се прави препратка назад и те да се посочват.

Например при бутон „.“ може да се започне с много простичко и несъвършено решение:

```

String temp = currentNumber.getText();
temp = temp + ".";
currentNumber.setText(temp);

```

Добавянето на кода за проверка и съответно изход от метода при съществуващо наличие на десетична запетая би могло да се добави впоследствие чак след когато учениците забележат пропуса или учителя ги насочи да го потърсят. Забележете, че този подход ни дава нещо много полезно – задачата става много адаптивна спрямо темпото и нивото на учениците. Ако някои от тях се справят по-бавно с преписване/написване на кода за съответните бутони, задачата ще бъде решена с опростен, неотчитащ частните случаи код (т.е. програмата би могла да се чути при реална употреба и да има бързи). Обратно – ако учениците са по-бързи и се справят с лекота, за тях няма проблем да се допълнят по-дребни нюанси и детайли около частните случаи, които трябва да се обработят с допълнителен код. Също така при един смесен клас, където има както бързи, така и по-бавно справящи се ученици, фините детайли по покриване на въпросните частни случаи биха могли да бъдат допълнителни задачи за онези, които са се справили по-бързо от останалите.

Отново от методическа гледна точка е добре учителят да не показва кода наготово (например както често се практикува с презентация), а вместо това да го пише на живо и учениците да му помагат за неговото написване. Това не само че прави урокът много по-интерактивен, но става също така и по-бърз. Практиката ми е показала, че по време на уроци от тип „лекция-упражнение“, в които учителят първо показва презентация, а учениците чак след изглеждането ѝ упражняват наученото, първата част обикновено е неатрактивна за обучаемите и много от тях изобщо не внимават – това от своя страна губи много време във втората част, която всъщност е същинската (учителят се налага да преповтаря неща, които вече е казал и уж е трябвало вече да са научени).

Единственото неработещо по примерния калкулатор се очаква да остане операцията на бутон „=“. Сама по себе си тя не е сложна, но изисква операции с числа и преобразуване на типове. Поради тази причина е добре да се остави за следващ урок.

Таблица 6. *Тема 3 от примерен училищен курс по информатика за VIII клас*

Тема 3. Основни типове данни	
Необходимо учебно време: 2 последователни учебни часа в комбиниран урок и 4 последователни часа урок за упражнение	
Знания и умения	Обяснение
Някои примитивни типове данни. Преобразуване на данни от един тип в друг	До този момент учениците са работили само с текстови низове и се очаква да познават основни операции с тях. Поради тази причина е целесъобразно именно те да се използват като отправна точка. Ако в графичния интерфейс има текстови компонент, в който е изписано число, вече се знае как съдържанието на текста му да се прочете със съответен <code>get</code> метод/функция. При поставяне на задача, с която това число трябва да се промени на друго чрез математическа операция (например при натискане на бутон да се увеличи с единица или да бъде събрано с друго число, което се вижда на екрана), реално се сблъскваме с тройна задача – първо трябва текстът да се превърне в число, след това математическата операция да се извърши и накрая обратно числото да се превърне в текст. Това само по себе си е лека като код, но по същество сложна за осъзнаване задача и включва множество теоретични обяснения. Поради тази причина темата е почти изцяло свързана с въвеждане на нови знания и много малко практическо упражнение с писане на програмен код.

	Обикновено в такава тема често учителите се изкушават да дадат пълен списък с всички типове данни, но това не е особено нужно на този етап – освен че учениците все още не са достигнали нивото да могат да осъзнаят нуждата от по-малки (<code>byte</code>) и по-големи (<code>long</code>) целочислени типове, те нямат и непосредствена потребност от това – подобно знание би могло да се въведе когато се постави задача, в която неизбежно се прехвърля типът на стандартния <code>int</code> и така да породят естествена нужда от въвеждане на новите знания. По същия начин обяснения от типа „с колко бита се записва числото“ нямат особен смисъл от ранно научаване – потребно е да се учи чак след като е достигната задача, в която обемът на данните в паметта има практическо значение. Същото важи и за числата с плаваща запетая – няма нужда учениците да бъдат въввлечени в изучаване на знания за това как се съхраняват те (мантиса, експонента, знак), нито да им бъдат обяснявани възможните видове с единична и двойна точност (<code>float</code> и <code>double</code>). За началния етап при числата с плаваща запетая е достатъчно да се каже, че те са начин да съхраняваме подобие на реални числа при компютрите – защо е подобие се обяснява със сравнително лесен пример с ирационалното число π , което учениците познават много добре от часовете по математика: само по себе си истинското число π има безкрайно много цифри след десетичната запетая, но компютърът няма безкрайно много памет, в която да ги запише. Именно поради тази причина може изключително елементаризирано да се каже, че <code>float/double</code> при компютрите са приближения до реалните числа, т.е. компютърът ги закръгля, и съответно да се спре дотам, а обясненията за същината на числата с плаваща запетая – мантисата, експонентата и т.н. – да се оставят за изучаване в бъдеще.
--	---

Продължавайки примера с калкулатора, се очаква да е достигнато състояние, в което в `currentNumber` е текущото число в текстовото поле, в `memory` е записано предишното, а в `operation` – знакът на операцията, която трябва да се извърши между тях. Тук, ако се

използва програмният език Java или друг, в който конкатенацията на текстови низове се извършва чрез знак „+“, може много лесно да се подходи със софизъм [11], като учителят започне да решава задачата целенасочено погрешно:

```
String num1 = currentNumber.getText();
String num2 = memory.getText();
if(operation.getText().equals("+")){
    currentNumber.setText(num2 + num1);
    memory.setText(" ");
    operation.setText(" ");
}
```

Очевидно при така извършените операции ние конкатенираме два текста, а не събираме числа. Това не се казва директно на учениците, а се поставя на обсъждане под формата на дискусия с цел те сами да се досетят в какво се корени проблема. Учителят би могъл да ги подпомогне, като вместо да подава конкатенацията num1+num2 директно като параметър на метода setText, преди това да го изчисли чрез заделяне на отделна променлива:

```
String result = num2+num1;
```

След като бъде разяснено откъде идва проблемът, настъпва естествената необходимост да се изучи нов тип данни – в случая този на числа с плаваща запетая, което включва и двата вида преобразуване: от текстови низ в число и обратно:

```
double num1 = Double.parseDouble(currentNumber.getText());
double num2 = Double.parseDouble(memory.getText());
if(operation.getText().equals("+")){
    double result = num2 + num1;
    currentNumber.setText(""+result);
    memory.setText(" ");
    operation.setText(" ");
}
```

При реализиране на другите операции – изваждане, умножение и деление – кодът е напълно аналогичен и учениците могат съвсем лесно да го добавят под формата на вложени условни оператори (if-else-if...) или (ако ходът на урока върви бързо и остане време) дори оператор за многовариантен избор (switch) като техен еквивалент (може да се остави като груповата задача да се проучи как работи switch за ученици, които са се справили по-бързо).

Както се вижда, програмният код в този пример е много малко – значително повече време трябва да се отдели за обяснения и странични примери. Например учителят може да отвори отделен проект, в който да разположи текст (Label) и бутон, с които да демонстрира резултатите от операции с различни типове данни – тип int, операции между такива цели числа и какво се случва когато извършваме операция между число с плаваща запетая и цяло число. Тук убедено смятам, че отново софизмите са много удачен методически инструмент. Следните задачи, които сме събрали в [11], демонстрират някои от важните характеристики при различните типове:

1. Защо върху числата с плаваща запетая може, но не трябва да се изпълнява оператор за точно сравнение (==):

```
if(0.1 + 0.2 == 0.3){
    label.setText("Това очаквахме");
}
else{
    label.setText("Защо стана така?!?");
}
```

2. Еквивалентно на горния пример, при числа с плаваща запетая се губи точност включително поради приоритета на операциите (учениците знаят добре от математиката, че първо се извършват операциите в скобите – поради тази причина няма нужда от изчерпателно обяснение на понятието „приоритет“, а само се припомня и използва по аналогия):

```
if( (0.1+0.2)+0.3 == 0.1+(0.2+0.3) ){
```

```
label.setText("Това очаквахме");
}
else{
    label.setText("Какво?!?");
}
```

3. Защо веднъж се получава коректен резултат 3.5:

```
double a = 7;
double b = 2;
double result = a/b;
label.setText(""+result);
```

но друг път не е така и се получава 3.0 (когато в дясната страна на равенството типа данни е целочислен, резултатът от операцията „+“ е целочислен, който после се превръща в число с плаваща запетая):

```
double result = 7/2;
label.setText(""+result);
```

4. Защо при „сбор“ на символ с целочислен тип се получават неочаквани резултати:

```
int x = 8;
x = x + '0';
label.setText(""+x);
```

Подобен софизъм отива прекалено далеч от текущото ниво на учениците, защото служи за демонстрация на начина, по който се съхраняват символите (че „зад тях“ стоят числа и съответна кодировка), но може да се употреби на по-късен етап.

5. За прехвърляне на максимална стойност на тип и евентуално въвеждане на „по-големи“ типове може да се демонстрира изчисляването на броя микросекунди в един ден, което ще даде грешен резултат:

```
int microseconds_per_day = 24 * 60 * 60 *
1000 * 1000;
```

който всъщност няма да се „поправи“ дори със смяна на типа на long:

```
long microseconds_per_day = 24 * 60 * 60 *
1000 * 1000;
```

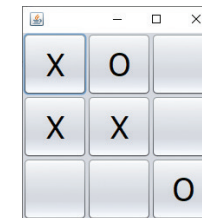
защото вдясно стоят само числа от тип int и резултатът е int. Затова тук правилното решение е:

```
long microseconds_per_day = 24L * 60 * 60
* 1000 * 1000;
```

Софизмите не трябва да бъдат въведението в проблема, а последващи – демонстрация за какво да се внимава чак след като е показан съответният тип при нормална употреба.

С изминаване на този път учениците вече имат достатъчно натрупани практически знания, за да могат да работят сами. Затова оттук насетне е целесъобразно да се пристъпи към упражнение на вече изучените знания, като учениците бъдат оставени в малки групи или самостоятелно да повторят наученото от първите три теми в контекста на визуално различна, но по същество упражняваща същите знания задача. Хубаво е да изградят и графичната част от задачата сами, но това е възможно само ако тя е по-проста – с по-малко компоненти. Ако се цели създаване на визуално по-сложна задача, може да се започне от предварително подготвен шаблон, който учениците трябва да довършат.

Примерна достъпна за текущото ниво, визуално различна от досега правеното и тематично атрактивна за учениците програма, би била да се напише добре познатата от междучасията им занимателна игра „морски шах“ (ХО). Разбира се тя трябва да се реализира във вариант двама играчи да играят на един компютър, като се редуват, а не един да играе срещу компютър, защото последното изисква написването на значително по-сложни алгоритми за изкуствения интелект.



В началото се започва с девет бутона с празен текст. При натискането на бутон, текстът му трябва да се сменя с „X“ или „O“ в зависимост от това кой играч е на ход. Макар и по същество проста, тази задача със сигурност ще затрудни учениците. Имайки опит с калкулатора, те биха могли да напишат код, с който се проверява текстът на съществуващия бутон и се сменя на буквата X, ако е празен.

```
If (btn11.getText().equals("")) {  
    btn11.setText("X");  
}
```

Проблемът, с който неизбежно учениците ще се сблъскат и не се очаква да успеят да разрешат практически без помощ от учителя, е свързан с това как да се прецени кога да се изписва „X“ и кога „O“. Силно вероятно е поне един от учениците да се досети за възможно решение – например да пазим брояч на броя ходове, който да се гледа дали е четен или нечетен, или пък още по-елегантно само променливата, в която да се отчита дали ходът е четен или нечетен. Няма как обаче да го реализират сами, защото макар и просто, това все още не е учено – изисква знание как различни бутони да си споделят информация един на друг. Тук учителят може да помогне като дефинира булевия тип:

```
boolean oddMove = true;
```

след което да използва променливата при извършване на действието на бутона:

```
If (btn11.getText().equals("")) {  
    If (oddMove==true){  
        btn11.setText("X");  
        oddMove = false;  
    }  
    else{  
        btn11.setText("O");  
        oddMove = true;  
    }  
}
```

Остава кодът да се мултиплицира върху всички бутони, като това се случва най-лесно чрез отделянето му в самостоятелен метод – нещо, което вече е учено:

```
void btnClicked(JButton btn) {  
    if (btn.getText().equals("")) {  
        if (oddMove == true) {  
            btn.setText("X");  
            oddMove = false;  
        } else {  
            btn.setText("O");  
            oddMove = true;  
        }  
    }  
}
```

С това практически програмата може и да завърши – макар и примитивна, ще бъде напълно функционална компютърна игра за двама души. Разбира се, при наличие на време и достатъчно бързо справили се със задачата ученици, тя може лесно да се усложни, като се добави условие за проверка на печеливша ситуация след всеки ход. Едно практически ефективно решение би включило сложни проверки, които отчитат върху кой точно бутон е натиснато последно (няма как да има печеливша ситуация извън неговия ред/стълб/диагонал) и последващо обхождане на двумерен масив с цикли (нещо, което още не е учено), но тук трябва да се ограничим с възможно най-простото решение – да се проверят всички редове, стълбове и диагонали за печеливша позиция. Задачата е трудна за начинаещи и изразите в условните оператори ще са много сложни, поради което е добре, ако това въобще тръгне да се прави, учителят да показва на учениците кода, а не те да го пишат сами. Например само проверката на първи ред ще бъде:

```
boolean testWin() {  
    if (btn11.getText().equals(btn12.getText()  
                                && btn12.getText().equals(btn13.
```

```

getText()
    && !btn11.getText().equals("")) {
    return true;
}
...
return false;
}

```

Ясно е, че учениците биха могли да направят редовете, после колоните и накрая диагоналите по аналогия, като само променят съответните индекси. Впрочем именно наименованието на бутоните чрез индекси (btn13 означава „бутон на първи ред, трета колона“) е много силна пропедевтика за бъдещо въвеждане на понятието „масив“ и „индекс на елемент в масив“. При подобни сложни изрази не се очаква всички ученици да се справят безгрешно в собствените си проекти (следенето на индексите е трудно на този етап), а всички да разберат поне какво се прави, защото все пак е на достъпно за тях ниво.

Вижда се, че макар и формално разделени на часове за комбиниран урок и часове за упражнение, в представения пример практически не се получава така – в часовете за упражнение изникват нови неща. Може подобна тактика да изглежда хаотична и непремерена, но всъщност е много практически ефективна. С нея се поддържа интереса на по-напредналите ученици, които се справят бързо със задачите. За такива деца обикновено часовете за упражнения са много скучни, защото те свършват работа бързо и започват да пречат на останалите. Подобни малки трудности в задачите ще поддържат интереса им, докато останалите наваксат тяхното ниво.

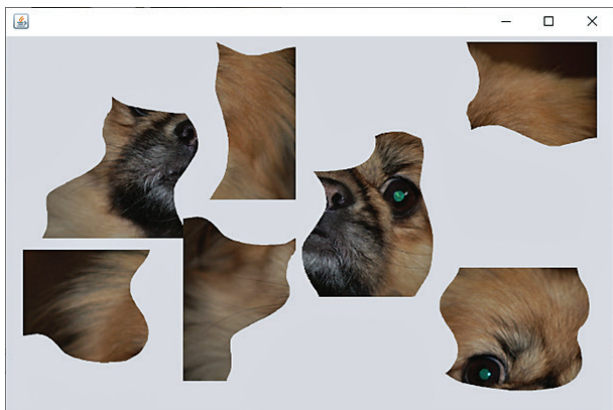
Таблица 7. Тема 4 от примерен училищен курс по информатика за VIII клас

Тема 4. Събития с обекти в графична среда	
Необходимо учебно време: 2 последователни учебни часа в комбиниран урок	
Знания и умения	Обяснение
Събития. Прецизно позициониране на обекти на екрана	До този момент учениците са работили със събития, но неосъзнато – всяко натискане на бутон е генерирало по едно такова. Вече е нормално да се въведе това понятие и да се демонстрира как може самите те да дефинират и генерират свои собствени. Трябва да се започне с генериране на събития, които са предизвикани от човека, който използва програмата – натискане на бутон на мишката или клавиатурата. Неизбежно свързано с управлението на подобни събития е и понятието „обект на фокус“ (който приема събитието). Естествено, както всичко изучавано дотук, понятието се въвежда с елементаризиран спрямо нивото на учениците обем от знания.

В тема 4 по същество се показва, че действията с различните обекти може да се пораздат от различни източници, които ги генерират. Подобно знание би могло да послужи за сравнително лесно изготвяне на редица заготовки за „мини компютърни игри“, които ще са едновременно лесни за създаване и атрактивни за децата.

Една примерна и много лесна за решаване задача е да се направи пъзел, който да може да се сглобява с мишката. Учителят е нужно предварително да подготви разрезите на компютърно изображение като отделни картинки. Това може да се направи тривиално, като изображението се разреже на правоъгълници, но може да стане и по-ефектно – като истински пъзел, чрез по-сложни контури и използване на прозрачен фон (алфа-канал). Подреждането на картинките по екрана с Java в NetBeans може да се случи по най-лесен начин с добавяне на готов компонент JLabel и добавянето на характеристика „иконка“ (icon) за него. Специално ще акцентирам

върху това, че заготовката се прави предварително от учителя, а не от учениците – това е действие, което няма пряка полза за програмирането и не трябва да се измества фокуса с него.



Движението може да се осъществи много лесно със създаване на `MouseDragged` събитие за всяка картинка, което от своя страна извиква следния метод:

```
void move(JLabel image, MouseEvent mouse) {  
    int newPositionX = image.getX() + mouse.  
getX();  
    int newPositionY = image.getY() + mouse.  
getY();  
    image.setLocation(newPositionX, newPosi-  
tionY);  
}
```

С така създадената програма пъзелът е готов и може да се сглобява, но има малко неудобство – влаченето на обектите не се случва съвсем така, както учениците очакват. Причината е, че местим левия горен ъгъл на картинката, а реално натискаме не него, а някъде вътре в нея – това прави едно неплавно прескачане при първото натискане върху екрана. Тази информация не трябва да се дава наготово – учениците трябва сами да помислят и да преценят

защо се получава така. Именно така ще могат сами да преоткрият основни свойства на координатната система, както и какво точно връщат въпросните `get` методи.

Макар и малко, при наличие на време това неудобство не е трудно да бъде решено. За всяка картинка създаваме допълнително второ събитие `MousePressed`, с което запомняме относителната позиция на мишката спрямо горния ляв ъгъл на обекта, върху който е натиснато с левия бутон на мишката:

```
this.draggedAtX = evt.getX();  
this.draggedAtY = evt.getY();
```

И след това изваждаме тези две числа в метода `move`:

```
void move(JLabel image, MouseEvent mouse) {  
    int newX = image.getX() + mouse.getX() -  
draggedAtX;  
    int newY = image.getY() + mouse.getY() -  
draggedAtY;  
    image.setLocation(newX, newY);  
}
```

При тази задача може да влезе в действие изследователския подход – след съвсем малка насока, учениците могат напълно сами да изпробват да допишат формулата, която ще ги отведе до крайния резултат. Обикновено първо я изписват погрешно, като добавят, вместо да извадят относителното отместване, но ефектът на „аха момента“ е много бърз – веднага разбират къде им е грешката. Друг често срещан проблем е свързан с това, че в координатната система оста по вертикала расте надолу, а не нагоре – това често обърква ученици, които са използвали координатни системи в математиката и съответно са свикнали да е с обратна посока. С подобен много кратък като количество код пример се добиват множество основни знания за работа с обекти и тяхното позициониране на екрана:

- координатната система;
- относителни и абсолютни координати;
- осъществяване на взаимовръзка между различни обекти;
- видове събития с мишка.

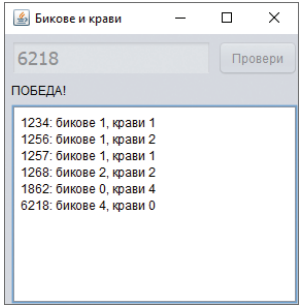
Това е много силна подготовка за следващи по-сложни задачи, в които може не само ще се местят обекти, но и да има някакво взаимодействие между тях. Например често в компютърните игри се налага да се прави „колизия“ (сблъскване на обекти) и тя се изчислява именно чрез техните позиции и размери.

Таблица 5. Тема 6 от примерен училищен курс по информатика за VIII клас

Тема 5. Цикли	
Необходимо учебно време: 3 часа в комбиниран урок 1 час упражнение	
Знания и умения	Обяснение
Цикли. Вложени цикли	Тук се очаква учениците по-задълбочено да изучават по-сложни алгоритми. Отново смятам за важно да бъдат поставени пред реален практически проблем, който да породии нуждата от въвеждане на новите знания.

Като примерна програма, с която да се въведат понятията, предлагам една много популярна игра, която почти винаги учениците познават и често играят на хартия в междучасията (мобилните телефони отдавна завзеха огромна част от игрите, но все още играенето с „хартия и химикалка“ не е изчезнало напълно) – „Бикове и крави“. Това е логическа игра за отгатване на числа, която се играе от двама играчи – в случая ще се реализира като един от играчите е компютър. Компютърът ще си намисли едно тайно четирицифрено число в интервала [1000, 9999], в което всички цифри са различни. Целта на играча е да отгатне това число, като дава последователно свои предположения с налучкване. При всеки опит компютърът ще дава насоки доколко текущото налучкване е близо до верния отговор. Принципът е следният:

- Ако дадена цифра от предположението се съдържа в тайното число и се намира на точната позиция в него, компютърът ще отчете, че има един „бик“.
- Ако дадена цифра от предположението се съдържа в тайното число, но се намира на различна позиция в него, компютърът ще отчете, че има една „крава“.



Съставянето на графичния интерфейс е тривиално, като тук просто се добавя още един досега неизползван нов обект – Text Area (многоредово текстово поле). Работата с него не е сложна и не се очаква да предизвика затруднения. Единственото по-специфично знание, което би могло да затрудни учениците, е поставянето на нов ред чрез специални символи – \n и \r. Програмата може да бъде опростена, ако текстовото поле напълно се премахне и се замени с обикновен Label – в такъв случай няма да се пази история на всички предишни опити, а ще се показва само последният.

Забележете, че въвежданото от потребителя число се пише накуп в едно текстово поле, а не както обикновено интуицията ни води – в четири отделни по-малки кутийки (по една за всяка цифра). Причината е, че ще искаме това число да го приемем като един цял текстови низ, който впоследствие ще обхождаме в цикли на принципа буква по буква. Това е достъпна пропедевтика за бъдещото въвеждане на понятието „масив“. От предишните теми се очаква учениците да са свикнали добре с разликата между типове данни – тук отново се възползваме от това да ги сблъскваме с тях, защото например символът „б“, който ще се вижда на екрана, е нещо различно от числото 6.

Неизбежно ново знание извън предвиденото в програмата при тази задача е генерирането на произволно число от страна на компютъра. Препоръчвам тази задача да бъде оставена за края – след като бъде направено всичко останало и само ако остане време. За начало спокойно може да дефинирате някакво

„работно“ число в кода – така ще е и по-лесно да се тества дали написаните от нас алгоритми са коректни, или имат някаква грешка.

```
String computerNumber = "1234";
```

Цялата работа в програмата ще се извършва от бутона `btnCheck`, който ще трябва да брои биковете и кравите, които потребителят въвежда в текстовото поле `guess`. Освен това ще наречем голямото текстово поле за съобщения `log`, а нередикуемия текст за системни съобщения `sysMessage`. Обикновено интуицията тласка човек, който решава тази задача, да започне моментално да брои биковете и кравите, но тук е добре учителят да спре този порив – това всъщност не са съвсем прости задачи, а преди тях биха могли да се решат една поредица от по-лесни, които да натрупат стъпаловидно поредица знания.

Например първият елементарен за решаване въпрос е: „А вие сигурни ли сте, че това, което е въвел потребителят, е точно с четири символа?“. Лесно може да се напише метод, който проверява това:

```
boolean inputIsFourLetters() {  
    if (guess.getText().length() == 4) {  
        return true;  
    }  
    else{  
        return true;  
    }  
}
```

и впоследствие да се извика при действието на бутона:

```
if(!inputIsFourLetters()){  
    sysMessage.setText("Въведете 4 цифри");  
    return;  
}
```

Кодът на функцията `inputIsFourLetters` може значително да бъде съкратен, като тялото се напише просто с изрази:

```
return (guess.getText().length() == 4);
```

Подобни практични трикове трябва да се показват само ако се прецени, че учениците са усвоили достатъчно добре работата с условните оператори – невинаги е така, защото не е имало достатъчно време за затвърждаване на знанията.

Въвеждането на понятието „цикъл с брояч“ може да дойде непосредствено с поставянето на въпроса: „Вече сме сигурни, че са въведени точно четири символа, но дали те са цифри?“. Да, реално в стандартно (неформатирано) текстово поле потребителят може да въведе каквито си иска символи. НЕ решавайте проблема, заменяйки обикновеното текстово поле с форматирано – използвайте стандартното, за да свършите работата сами и така да покажете нови фундаментални знания за цикъл с брояч, а не просто нов обект, който върши цялата работа наготово.

Кодът на метод, който обхожда текстовия низ буква по буква, може да е следният:

```
boolean inputIsOnlyDigits() {  
    for(int i=0; i<4; i=i+1){  
        char currentLetter = guess.getText().  
            charAt(i);  
        if (currentLetter<'0' || currentLetter>'9'){  
            return false;  
        }  
    }  
    return true;  
}
```

Извикването на функцията става по вече познатия начин при натискане на бутона:

```
if(!inputIsOnlyDigits()){  
    sysMessage.setText("Въвеждайте само  
цифри");  
    return;  
}
```

С тази задача все едно се прави обхождане на масив елемент по елемент, но формалното понятие „масив“ е скрито зад вече познат обект – текстови низ. По желание и при наличие на време учителят може да представи и алтернативно решение на същата задача – нещо, което вече е използвано в първата примерна задача с калкулатора и по този начин да припомни по-стари знания:

```
try {
    int num = Integer.parseInt(guess.getText());
    return true;
} catch (NumberFormatException e) {
    return false;
}
```

Следващият въпрос, който може да бъде поставен, е: „Знаем, че са въведени точно четири цифри, но дали тези цифри са различни?“. Тук е добре учениците първоначално да се опитат сами да решат тази задача. В досегашния ми опит, най-често те започват с писане на много сложни изрази в условен оператор – проверка дали първата буква е различна от втората И първата е различна от третата И първата е различна от четвъртата И втората е различна от третата, т.н.

Точно в момента, в който учителят установи, че учениците са измислили алгоритъма, той може да ги спре (да не губят време да дописват сложния логически израз) и да им покаже по-елегантно-то решение с обхождане на текстовия низ чрез два вложени цикъла. Имайте предвид, че обяснението на този алгоритъм изисква повече време – не всички ученици успяват да го разберат чрез програмния код. Може би най-добрият начин е чрез постъпателно демонстрация чрез конкретен пример на дъската с това какво се случва със стойността на всяка променлива при всяко завъртане – обикновено е удачно да се покаже в табличен вид.

```
boolean allSymbolsAreDifferent() {
    String str = guess.getText();
    for (int i = 0; i < 3; i = i + 1) {
```

```
        for (int j = i + 1; j < 4; j = j + 1) {
            if (str.charAt(i) == str.charAt(j)) {
                return false;
            }
        }
    }
    return true;
}
```

Методът ще бъде извикан при действието на бутона по познатия начин:

```
if (!allSymbolsAreDifferent()) {
    sysMessage.setText("Цифрите трябва да са различни!");
    return;
}
```

Вече можем спокойно да пристъпим и към броенето на биковете. Тук имаме съвсем леко усложнение на вече изучения материал, защото ще обхождаме едновременно два текстови низа – този на компютъра и този от потребителя:

```
int countBulls() {
    int bulls = 0;
    for (int i = 0; i < 4; i = i + 1) {
        char currentComputerLetter = computer-
        Number.charAt(i);
        char currentPlayerLetter = guess.getText().charAt(i);
        if (currentComputerLetter == currentPlayerLetter) {
            bulls = bulls + 1;
        }
    }
    return bulls;
}
```

Алгоритъмът за кравите вече е значително по-сложен. Учениците трябва задълбочено да разискват какво точно трябва да се направи, преди да се пристъпи към реализация. Едно примерно решение е следното:

```
int countCows() {
    int cows = 0;
    for (int i = 0; i < 4; i = i + 1) {
        for (int j = 0; j < 4; j = j + 1) {
            char currentComputerLetter = computerNumber.charAt(i);
            char currentPlayerLetter = guess.getText().charAt(j);
            if (currentComputerLetter == currentPlayerLetter
                &&
                i!=j) {
                cows = cows + 1;
            }
        }
    }
    return cows;
}
```

Рядко всички успяват да разберат какво точно се е случило и затова същата задача трябва да се преповтори по-късно под друга форма и в друг контекст. Може да се използва и тактиката задачата да се реши по алтернативен начин – например да се преброят всички съвпадащи букви без значение от позицията им, а след това от полученото число да се извадят биковите. Предвиждам в рамките на половин час да се припомни алгоритъмът от предишния час, а програмата да се довърши, като първо се приключи с бутона:

```
int bulls = countBulls();
int cows = countCows();
log.append(guess.getText()
    + ": бикове " + bulls
```

```
    + ", крави " + cows + "\n\r");
if (bulls == 4) {
    sysMessage.setText("ПОБЕДА!");
    guess.setEnabled(false);
    btnCheck.setEnabled(false);
    return;
}
```

и само се демонстрира (няма нужда учениците да го упражняват!) как се генерира произволно число от компютъра – в Java е най-удачно за генериране на произволно число в интервала [1000, 9999] да се използва класът `java.util.Random` по следния начин:

```
Random r = new Random();
attempt = 1000 + r.nextInt(9000);
computerNumber = "" + attempt;
```

Учениците много бързо ще се досетят, че това не е достатъчно, защото числото трябва да отговаря на същите критерии, на които и числото на играча – да бъде с различни цифри! Вече написаният метод `allSymbolsAreDifferent` не е универсален – в тялото той работи конкретно с числото, въведено от потребителя, и не може да бъде използван за валидация на това на компютъра. Може методът да бъде дублиран, като се копира втори път под ново име и се смени само името на обекта, с който се работи, но именно в този случай е много по-добре вече написаният код да се промени така, че да може да бъде преизползван и за двете цели:

```
boolean allSymbolsAreDifferent(String str) {
    for (int i = 0; i < 3; i = i + 1) {
        for (int j = i + 1; j < 4; j = j + 1) {
            if (str.charAt(i) == str.charAt(j)) {
                return false;
            }
        }
    }
    return true;
}
```

Като при бутона се изисква много малка модификация:

```
if (!allSymbolsAreDifferent(guess.getText())) {  
    sysMessage.setText("Цифрите трябва да са  
    различни!");  
    return;  
}
```

Вероятно ще се запитате „Защо не го написахме по този начин от самото начало и не си спестихме това допълнително време?“. Отговорът е, че подхождам по този начин нарочно – не е очевидно защо е нужно да правим методите програмно независими от останалата част на кода, докато не се сблъскаме реално с този проблем. Правенето на подобни нарочни грешки се залага целенасочено, за да може впоследствие учениците да се сблъскат с проблем, който да предизвика естествената нужда от добиване на нови знания и да послужи за пример как те са практически полезни. Това чувство не може да се развие достатъчно ефективно, ако не е достигната конкретна проблемна ситуация. Ще забележите, че в досегашните задачи сме допускали тази „грешка“ многократно – много други методи, които описах в примерните задачи, също работят с обръщения директно към обектите, а не са универсални и независими от тях, като само на два пъти се наложи кодът да се напише по начин, по който приема обекта, върху който се извършва действието, като входен параметър. Именно с такъв конкретен пример обаче настъпва момента този подход да се обобщи и затвърди – учениците вече са добили не само усет, но са достигнали и до практически категоричен пример за това, че е полезно от самото начало методите да се пишат така, че да може действията им да се прилагат в различни ситуации и върху различни обекти. Смятам, че е по-добре добрите навици да се придобиват с опит и осъзнаване, а не с налагане на догматични шаблони.

Оттук насетне до края на часа учителят може да покаже приложение на вече познатия от предишни теми цикличен алгоритъм `while`, но вече не безкраен, а с конкретно условие:

```
Random r = new Random();  
int attempt;  
attempt = 1000 + r.nextInt(9000);  
computerNumber = "" + attempt;  
while (!allSymbolsAreDifferent(computerNumber)) {  
    attempt = 1000 + r.nextInt(9000);  
    computerNumber = "" + attempt;  
};
```

Впоследствие да се покаже как същият пример за цикъл с предусловие може да се преработи в по-добрия за конкретния случай алгоритъм с постусловие:

```
Random r = new Random();  
int attempt;  
do {  
    attempt = 1000 + r.nextInt(9000);  
    computerNumber = "" + attempt;  
} while (!allSymbolsAreDifferent(computerNumber));
```

и в крайна сметка трите вида алгоритми – с предусловие, постусловие и брояч – да бъдат обобщени, като чисто в теоретичен план се покаже, че следната конструкция на цикъл с брояч:

```
for(int i=0; i<10; i=i+1){  
    operator1;  
}
```

е еквивалентна на следния цикъл с предусловие:

```
operator1;  
int i=0;  
while(i<10){  
    operator1;  
    i=i+1;  
}
```

и отново еквивалентна на следния цикъл с постусловие:

```
int i=0;
do{
    operator1;
    i=i+1;
while (i<10) ;
```

Естествено, трябва да се акцентира, че невинаги имаме нужда от това тялото на циклите да се изпълнява поне веднъж, т.е. невинаги цикълът с постусловие е удачен (дори всъщност в практиката той се използва сравнително рядко спрямо останалите два вида). За това могат да се дадат и конкретни примери.

При всички положения няма как да се очаква всички ученици да могат да пишат сложни алгоритми, без да се упражняват допълнително – преброяването на кравите беше нещо, което обикновено надхвърля нивото на повечето от тях. Поради тази причина е добре учителят да предостави време за решаване на друга подобна задача в различен контекст. Може да бъде предоставена графичната част и част от кода на програмата наготово, а те да довършват сами обработката на информацията. Моето виждане е, че това затвърждаване на знанията може да се постигне и по време на упражненията по следващата тема, която предлагам да е свързана с масиви – преминаването на обхождане от текстови низ буква по буква до масив от числа елемент по елемент не е значително сложно, защото аналогията е много силна.

Хубаво е да се отбележи и още нещо – подобно на калкулатора, огромна част от кода по тази примерна програма е свързан с валидиране на входните данни. И тук елементаризирането на задачата е изключително тривиално – тази валидация може в една или друга степен да се пропусне. Това прави задачата изключително адаптивна спрямо темпото на учениците и отново дава възможност, ако някои ученици изостават, а други вървят бързо, да бъде ангажирано вниманието на по-бързите с допълнителни самостоятелни задачи, докато през това време учителят помага на учениците, които не се справят.

Таблица 9. Тема 6 от примерен училищен курс по информатика за VIII клас

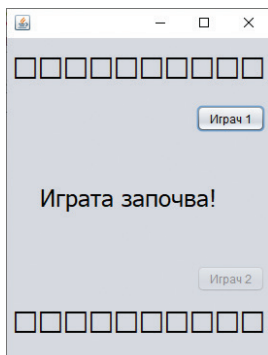
Тема 6. Едномерни масиви	
Необходимо учебно време: 3 часа в комбиниран урок 1 час упражнение	
Знания и умения	Обяснение
Масив	Масивът като основна структура от данни смятам, че трябва да се представи като краен етап от уводния курс. Поставените задачи не трябва да са прекалено алгоритмично сложни, а да са практически смислени и не на последно място – да се усети истинска нужда от научаването им, т.е. примерите да са такива, които трудно се решават без употреба на масиви.

Тривиални задачи са да се даде масив, който първо да се отпечата в прав ред, после в обратен ред, да се сумират елементите му, да се покажат само четните/нечетните числа от него и т.н. Подобни задачи са важни и дават фундаментални за програмирането знания, но ако не бъдат използвани реални (смислени) числа, те стоят изкуствено и ползата от тях не е очевидна за обучаемия. Една класическа, лесна и достъпна задача е човек да си въведе оценките и след това да изчисли своя среден успех. Подобни програми бяха изключително често срещани сред учебните задачи в миналото, когато се пишеха конзолни приложения. Макар и не особено атрактивна, такава може да се реализира съвсем елементарно с минимални усилия с графична среда.

В случая отново ще предложа задача, свързана с елементарна игра за двама. Всеки играч си начертава десет празни кутийки на лист хартия. След това последователно започват, редувайки се, да хвърлят по два зара. При всяко хвърляне резултатът (цяло число от 2 до 6) се записва в последователната празна кутийка. Накрая се сумират резултатите от хвърлянията и играчът с по-висок общ сбор печели.

Визуалната част на така дефинираната игра не е очевиден. Може да се направят множество различни подходи и да се подло-

жат на обсъждане. Обикновено учениците решават да има общо двадесет малки кутийки от текстови полета на екрана, защото това е най-очевидното решение с достъпните готови графични контроли. Не би било трудно да се направи, но те вече имат добит опит и знаят колко много усилия ще отнеме за нареждане и именоване, а и после обработка на това коя кутийка е запълнена и коя не. Тук е и моментът, в който учителя може да поеме водещата роля и да покаже как да се реализира по-лесно. Нека първо всичко се нареди в две редактируеми текстови полета (player1Label и player2Label), в които текстът е представен във вид на последователни символи с правоъгълници (например UTF-8 символ U+25A1). Всеки играч има свой собствен бутон за хвърляне на зар (p1btn и p2btn), като първоначално зарът на първия играч е активен, а на втория – не (след натискане на бутон двете трябва да се обърнат). В средата на екрана ще поставим текст (msg), в който първоначално оставяме някакво приветствие, а след това ще преизползваме за отчитане на съответните хвърляния на зар.



При натискането на всеки бутон ще извикваме метод roll с подаден параметър обектът с кутийките на текущия играч. Например за бутона на първи играч извикването ще е:

```
roll(player1Label);
```

Сега остава да се обсъди как ще бъде реализиран този метод roll. С досегашните знания бихме могли лесно да си дефинираме алгоритъм, при който:

- генерираме произволно число от 2 до 6;
- обхождаме текста и намираме последната свободна кутийка (или си пазим брояч за ходовете, с което си спестяваме това усилие);
- генерираме нов текст, в който първа свободна кутийка е подменена с произволното число;
- заменяме текста на съответното поле с новия;
- активираме чуждия бутон и деактивираме текущия.

Това е достъпно за текущото ниво, но ще се сблъскаме с проблем в края на играта. Когато сме попълнили и последната кутийка, трябва да се сметне общият брой. Учениците знаят, че в текстовите полета записваме букви, а не цифри и съответно събирането няма да е лесно (за тях, иначе има други лесни начини) работа. Все пак дори с досегашните им знания подобен код може да се напише сравнително лесно, като в процеса може да се покажат и някои дребни трикове, например лесната размяна на активацията на бутоните, без да се следи кой точно играч е на ход:

```
void roll(JLabel lbl){
    Random r = new Random();
    int dice = r.nextInt(5)+2;
    for(int i=0; i<10; i=i+1){
        if(lbl.getText().charAt(i) == '□'){
            String newStr = lbl.getText().substring(0, i)
                                +dice
                                +lbl.getText().substring(i+1);
            lbl.setText(newStr);
            p1btn.setEnabled(!p1btn.isEnabled());
            p2btn.setEnabled(!p2btn.isEnabled());
            if(i==9 && p2btn.isEnabled()==false){
                gameOver();
            }
        }
    }
}
```

```

    }
    return;
}
}
}

```

Проблемната ситуация, както вече споменах, настъпва с метода `gameOver()` – в него трябва да обходим текстовете буква по буква и съответно всяка буква да се превръща в число, което да се натрупва в обща сума. Това не е трудна, но определено ще е неприятна задача с тежък код. Познатата `Integer.parseInt` работи с низове, а не с единични символи и оттам насетне, ако не се използват трикове със знания за подредбата на кодировката на символите, ще направи кода изкуствен, далеч от оптимален и трудно четим. Например:

```

void gameOver() {
    plbtn.setEnabled(false);
    int plresult = 0;
    int p2result = 0;
    for(int i=0; i<10; i=i+1){
        plresult = plresult +
            Integer.parseInt(""+player1Label.getText().charAt(i));
        p2result = p2result +
            Integer.parseInt(""+player2Label.getText().charAt(i));
    }
    msg.setText("И1: "+plresult+"", "И2: "+p2result);
}

```

Затова можем да предложим на учениците да подходим по различен начин – да записваме числата директно в числен тип в паралелна (дублираща) структура – именно едномерен масив. Първо дефинираме два едномерни масива с числа:

```

int[] plarr = new int[10];
int[] p2arr = new int[10];

```

и при попълването на буквата добавяме генерираното произволно число на съответната позиция, т.е. метод `roll` се променя по следния начин:

```

void roll(JLabel lbl){
    Random r = new Random();
    int dice = r.nextInt(5)+2;
    for(int i=0; i<10; i=i+1){
        if(lbl.getText().charAt(i) == '□'){
            String newStr = lbl.getText().substring(0, i)
                +dice
                +lbl.getText().substring(i+1);
            lbl.setText(newStr);
            if(plbtn.isEnabled()){
                plarr[i] = dice;
            }
            else{
                p2arr[i] = dice;
            }
            plbtn.setEnabled(!plbtn.isEnabled());
            p2btn.setEnabled(!p2btn.isEnabled());
            if(i==10 && p2btn.isEnabled()==false){
                gameOver();
            }
            return;
        }
    }
}

```

Така в метод `gameOver` може да се опрости до:

```
void gameOver() {
    plbtn.setEnabled(false);
    int plresult = 0;
    int p2result = 0;
    for(int i=0; i<10; i=i+1){
        plresult = plresult + plarr[i];
        p2result = p2result + p2arr[i];
    }
    msg.setText("И1: "+plresult+", И2: "+p2result);
}
```

Имайки вече два масива с числа, условието на същата задача може лесно да бъде променено така, че решението да бъде решаване серия от класически задачи от типа „да се сумират само елементите на четни позиции“ и подобни. Полето от възможни задачи е много голямо.

Поддържането на масивите с числа като паралелни структури от данни в случая не е напълно наложително и може да се направи решение и без тях (даже, както се видя, изхождаме от такова). Все пак учениците лесно осъзнават, че в живота работим често с числа и записването им директно в подходящ за обработка тип ще е удобство, както и виждат, че невинаги данните, с които работим, е задължително да са визуализирани някъде на екрана. При това показваме една аналогия, че текстовият низ прилича на масив от букви (той е нещо много повече, но няма нужда от задълбочаване по темата в теоретичен план) и съответно двете се обхождат по много подобен начин.

Таблица 10. Тема 7 от примерен училищен курс по информатика за VIII клас

Тема 7. Изпълняване на подпрограми в нишки	
Необходимо учебно време: 1 последователен учебен час в комбиниран урок и 5 последователни часа проектна дейност	
Знания и умения	Обяснение
Нишка	Изпълнението на програмен код в подпрограма с нишка ни позволява да създадем софтуер, в който събитията се пораждават не само от действие на човека, но и от заложената програмна логика в самия компютър. Идеята е да се създаде софтуер, в който потребителят да извършва едни действия по негово усмотрение, а компютърът – други, независимо от него. Подобни програми дават значително по-широка визия за това как работят повечето неща, които използваме в ежедневието си на компютър или мобилно устройство. В крайна сметка почти всички софтуер в нашето съвремие използва по един или друг начин паралелно програмиране. С подобна тема задаваме първите стъпки на учениците в тази област.

С предлагане на такава тема съзнавам, че очаквам да получа (а и практически получавам нерядко) критики от колеги, които са дългогодишни специалисти в областта на обучението по програмиране. Наложената практика във всички български университети е паралелното програмиране да се явява в по-горните курсове на обучението. Естествено, това има основателна причина – то не само е трудно, но осъзнатото му използване изисква множество знания как функционира операционната система, как се споделя сигурно памет, как да се избягват т.нар. *race conditions*, „мъртва хватка“, какво е *mutex*, „семафор“ и др.

Макар и вярно, тук трябва да си пречупим мисленето и да приемем, че ние реално не желаем да научим учениците на всякакви иначе важни подробности – само искаме да ги докоснем до основите и да оставим поле за бъдещо развитие. Тоест не просто не искаме, а и не трябва да бъдем изчерпателни! Този подход към елементаризиране на материала е наличен за практически всич-

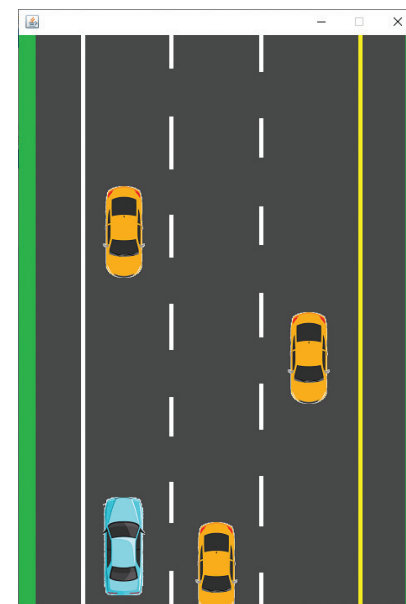
ко, което всеки учи в начален етап. Подходът е различен спрямо изискванията на университетските курсове, където би било недопустимо да не се дадат основни знания за синхронизиране на споделени ресурси – там се търси по-скоро дълбочина и изчерпателност в теоретичен план. В училище положението е обратното – трябва да се набляга почти изцяло на градене на основни знания чрез елементарни практически примери.

Ще дам по аналогия един пример от математиката – още в първи клас децата учат за естествените числа и извършват пресмятания с тях. Нима първокласниците познават аксиомите на Пеано за естествените числа или дефиницията им чрез теория на множествата на Джон фон Нойман? Бихме ли казали, че е недопустимо да позволяваме на учениците да правят сметки, без да се познава аксиоматиката в детайли, защото иначе знанията им са непълни? Очевидно е допустимо – всички винаги са учили на принципа на индуктивно натрупване на знания, а дедуктивното обобщение идва чак след като обучаемите са натрупали достатъчно опит. Допускането на грешки всъщност е нещо не само нормално, но и полезно – дори в народните поговорки се казва, че „Човек се учи от грешките си“. В частност това, че ще дадем самочувствие на децата, че могат да правят много по-сложни програми, отколкото е реалното им ниво, не е в никакъв случай нещо неправилно или лошо – напротив, ще ги мотивира да започнат да правят опити, с които да се сблъскват с нови проблеми, от които сами ще достигат до нови знания.

Желанието ми още в началния курс на програмиране учениците да се запознаят с най-простия начин да се стартират паралелни подпрограми е свързано с това, че в реалното програмиране подобни неща се правят изключително често – в почти всеки софтуерен продукт има някаква форма на многозадачност. Смятам, че ранното формиране на алгоритмично мислене с паралелни алгоритми ще е много полезно за обучаемите и споделям виждането на автори като например [75]. Още повече, самият аз въведох подобен експериментален модел с елементи от паралелно и мрежово програмиране в начален курс по ООП, докато бях асистент в

Техническият университет – София [34]. Експериментът ми не само че отчете успешен резултат, но разработените учебни ресурси и визията за курса като цяло се използват като модел от колегите ми и до днес при преподаването на същия учебен предмет. Тук говорим за училищен, а не за университетски курс – разликата ще бъде основно в това, че задачите за ученици трябва да са значително по-елементарни от алгоритмична гледна точка и с много по-малко програмен код, а теоретичните знания много по-малко както като общо количество, така и като изчерпателност и задълбоченост.

Една примерна стартова програма може да е прототип на елементарна игра. Например играчът управлява автомобил, движейки го наляво-надясно в долния ръб на екрана, а срещу него се задават други автомобили, които равномерно и праволинейно управлява компютърът. Целта е да се избягва сблъсък с чуждите автомобили.



От методическа гледна точка предлагам при решаването на подобна задача да се спазва принципа „разделяй и владей“. Имаме за решаване три фундаментални задачи:

1. Как ние да движим синята кола (нека я наречем car)?
2. Как да се движат чуждите коли (нека ги наречем enemy1, enemy2 и enemy3)?
3. Как да извършим интеракцията между тях (сблъсъка)?

Първата задача е напълно достъпна за обучаемите като вид упражнение, което надгражда знанията от предишния час. Възможно е да се направи движение на колата с издърпване чрез мишката (само по хоризонтала). Също така е може да се направи и движение чрез клавиатурата – например с натискане на бутони наляво и надясно:

```
private void formKeyPressed(java.awt.event.
KeyEvent evt)
{
    if(evt.getKeyCode() == KeyEvent.VK_LEFT){
        car.setLocation(car.getX() - 100, car.
getY());
    }
    if(evt.getKeyCode() == KeyEvent.VK_RIGHT)
    {
        car.setLocation(car.getX() + 100, car.
getY());
    }
}
```

Виждале, че с натискане на бутон наляво, преместваме колата със 100 пиксела наляво, а при натискане надясно ще се отмести със 100 пиксела надясно. Показваме на учениците също така, че с натискането на бутон от клавиатурата се изпраща уникално число (код на бутона) и имаме начин да проверяваме какво е било то. Ако учителят сметне, че същината на събитията все още не е достатъчно добре разбрана, същата функционалност може да се реализира още по-лесно с добавяне на графични бутони.

Втората задача с придвижването на колите трябва да се направи чрез поредица от по-елементарни стъпки. Първата е да убедим учениците, че е нужно да намерим начин да изпълняваме две

операции едновременно и независимо една от друга. А това не е трудно за обяснение: Могат ли например да движат своята кола едновременно наляво и надясно? Естествено, че не – могат да натискат ту единия, ту другия клавиш и операциите ще се изпълняват последователно една след друга. Натискайки ги едновременно, ще се види едно „примигване“ – колата първо ще отиде в едната посока, а после в другата. В същото време не можем и не трябва да чакаме играчът да придвижи своята кола, за да придвижим компютърно управляваната – това би направило играта походова и определено непрактична.

Така по чисто логичен път трябва да насочим учениците към мисълта, че червените коли трябва да се движат, независимо от това какво правим ние. Движенията на колите трябва да са независими от нашите и да се изпълняват паралелно с тях. Освен това те ще се движат до безкрайност (стига да не се сблъснем в тях, но това е задача за по-нататък). Един начин да се реализира придвижването на обектите е с нишка.

```
Thread enemy1Mover = new Thread() {
    public void run() {
        while (true) {
            // ...
            try {
                Thread.sleep(5);
            } catch (Exception ex) {}
        }
    }
};
```

Разбира се, опитните програмисти веднага ще се възпротивят и ще кажат, че използваме Swing, а той не е thread-safe пакет, т.е. би следвало да използваме Timer. Хубавото е, че се спестява писането на try и catch, но за сметка на това кодът е доста претрупан и за начинаещ не е ясно какво точно се случва:

```
Timer enemy1Mover = new Timer(5, new Action-
Listener() {
```

```

    public void actionPerformed(ActionEvent
e) {
        //...
    }
});

```

Поради тази причина може би е добре поне първият път да се раздели на две части:

```

ActionListener enemy1ActionListener = new
ActionListener() {
    public void actionPerformed(ActionEvent
e) {
        //...
    }
};
Timer enemy1Mover = new Timer(5, enemy1Ac-
tionListener);

```

Този шаблонен код трябва да се даде наготово (не е невъзможно, но би изискало много теоретични знания и упражнение, за да може учениците да достигнат сами до него) и ще се ползва оттук нататък при нужда. Учителят уведомява учениците, че кодът на мястото на многоточието ще се изпълнява до безкрайност отново и отново без значение в какво състояние е основната част от програмата и какво правим ние. Разбира се, прави и уточнение, че първият параметър е забавяне за брой милисекунди преди всяко ново изпълнение.

Въпросът е какво искаме да се извършва с този код? Отговорът е: „Да придвижим първата кола надолу“. Даваме идеята, че тя трябва да се мести с по един пиксел, след това да изчака съвсем малко време (няколко милисекунди) и после пак да се придвижи. Именно тези действия ще се повтарят отново и отново:

```

Timer enemy1Mover = new Timer(5, new Action-
Listener() {
    public void actionPerformed(ActionEvent
e) {

```

```

        enemy1.setLocation(enemy1.getX(), ene-
my1.getY() + 1);
    }
});

```

Финално просто трябва да стартираме въпросната нишка в самото начало на програмата чрез съответния метод:

```

enemy1Mover.start();

```

Обикновено при пускането на подобна програма в учебния кабинет настъпва оживление, но и бързо разочарование. Очевидно програмата не е довършена докрай – колата се движи надолу и... в един момент излиза извън екрана и не се връща никога повече (освен при прехвърляне на типа, но при тази програма е непрактично да се чака чак толкова време, а и няма особен смисъл да се показва точно това знание на това място). Тук учениците трябва сами да помислят какво трябва да се направи. Стандартно повечето казват, че трябва да проверим дали колата е излязла извън екрана и ако да, да я върнем обратно в изходна позиция. По-рядко някой дава по-непрактичната идея да пуснем нов автомобил.

```

Timer enemy1Mover = new Timer(5, new Action-
Listener() {
    public void actionPerformed(ActionEvent
e) {
        enemy1.setLocation(enemy1.getX(), ene-
my1.getY() + 1);
        if (enemy1.getY() > 680) {
            enemy1.setLocation(enemy1.getX(),
-150);
        }
    }
});

```

Числата 680 и 150 са подбрани буквално на око с принципа „проба-грешка“ – те са свързани с големината на рамката на основния прозорец и височината на картинката с автомобила. Това

не е правилен подход от гледна точка на приложното програмиране (по-добре е да няма подобни „магически числа“), но изглаждането на подобни проблеми и детайли е удачно за допълнителна задача (само ако остане време). Иначе втората задача е завършена – автомобилът се движи свободно от горе надолу и при излизане от екрана се връща обратно. Остава същото да се повтори за другите два автомобила, което е тривиална задача и учениците спокойно могат да направят сами.

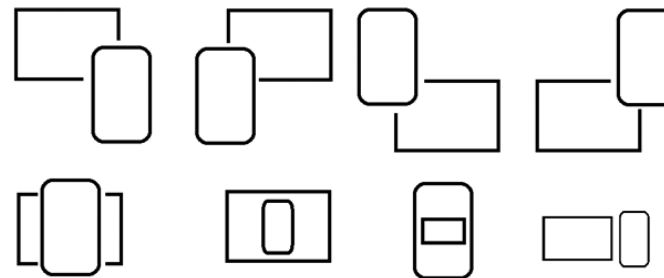
Всъщност вече имаме и още един „порок“ със стила на програмиране – по същество копираме един и същи код за създаване на нов клас за по същество еднакви по тип обекти, което не е правилно. Отново ще повторя – подобни неща е добре да се изглаждат, но само ако има останало достатъчно време. Писането на качествен програмен код и спазването на добри шаблони (design patterns) е много важно в практиката, но в ранен етап на обучение отлага значително достигането до бърз резултат, а това обикновено е разочаровашо за обучаемите и прави предмета да изглежда скучен. Нека тези шаблони бъдат изучавани, преоткривани от самите ученици едва след като са получили реална нужда от тях. Ролята на учителя е да обръща внимание на такива моменти и ако няма време да поправи кода, то поне да посочи проблема и да подсказва, че има по-добро решение, което евентуално ще бъде изучавано на по-късен етап.

В крайна сметка въпросът е философски: Правилно ли е да учим първоначално учениците да програмират грешно, а в по-късен етап да зачеркваме всичко, което са учили и да ги учим да правят същите неща по съвсем нов начин? Макар и да изглежда логически абсурдно, това смятам, че не е лош подход, защото, както вече казах „Човек се учи от грешките си“. При това въобще не е невиджано в обучението по информатика – например в курсовете за програмиране на C в началото масово се използват функции като `scanf`, `gets`, и др., а по-късно практически се забранява тяхната употреба, защото са несигурни и водят до тривиални препълвания на буфера. Защо тогава се използват в началото, а не се започне директно с техните сигурни еквиваленти? Отговорът е ясен – за-

щото с тях е значително по-лесно и изисква значително по-малко обяснение. Затова в горните примери може дори да се ползва `Thread` с безкраен цикъл вместо `Timer` и това няма да е нещо лошо – всъщност достигането до `race condition` при такива малки учебни задачи практически не се случва никога. В по-късен етап всички тези пропуски могат да бъдат обсъждани и отстранявани, когато обучаемият вече е добил необходимата база от теоретични знания и практически умения.

Последната задача – засичането на колизия – всъщност се оказва алгоритмично сложна за учениците. Тя има много бързи и лесни решения с методи от готови библиотеки, но е добре да не се започва с тях, а учителят да се опита поне да даде насока за това, че проблемът не е тривиален за решаване. При липса на време може да не се разреши изцяло с програмен код, а само да се даде формална конструкция и идея. В крайна сметка задачата се свежда до следното: дадени са два правоъгълника, които са дефинирани чрез координати на тяхната горна лява точка и дължините на двете им страни (разполагаме с методи `getX`, `getY`, `getWidth` и `getHeight` за всяка картинка) и накрая се пита кога тези два правоъгълника се пресичат и кога не.

Подобна задача е много „калорична“ за прилагане на метода за решаване на задачи чрез „проба-грешка“. Обучаемите бързо измислят някакво решение, но то обикновено се оказва просто един от възможните частни случаи от всички възможни взаимни положения на двата правоъгълника.



В зависимост от методиката, която приемем, някои от тези случаи ще се окажат еквивалентни спрямо израза, с който правим проверка (т.е. някои от вариантите могат да отпаднат), но при всички положения за решаването на тази задача с текущо наличния инструментариум ще има няколко условни оператора с доста сложен израз. Достатъчно е учителят да покаже поне един от тях и да се увери, че учениците са го разбрали. Оттам насетне може да не се продължава докрай, защото е нормално подобна много сложна откъм програмен код задача да порожи недоволство, разочарование, а у някои дори чувство на безпомощност. Вместо това учителят може да подходи хитро и да каже „очевидно това, което измислихме с вас е много трудно и неприятно – дали не можем да потърсим някакво решение от интернет наготово?“.

А такова се намира изключително лесно с всяка търсачка – при елементарно търсене веднага се попада на редица примери, като например следния:

```
boolean intersects(JLabel l1, JLabel l2) {  
    Area areaA = new Area(l1.getBounds());  
    Area areaB = new Area(l2.getBounds());  
    return areaA.intersects(areaB.getBounds2D());  
}
```

Тук няма нужда да се обяснява подробно какво е Area и как какво прави `getBounds` – всъщност действаме на принципа „копирай-вмъкни-изпробвай“, а обяснението за това какво всъщност прави кодът, следва да е отгоре-отгоре (самите ученици могат сами да предположат какво се случва на базата на имената на класовете и методите). Има и алтернативни, не особено по-сложни решения.

Обикновено реакцията на учениците след подобна демонстрация е да кажат: „Е, защо се мъчихме толкова много, а не потърсихме готово решение още в самото начало“. Отговорът на учителятук е хубаво да бъде: „Да, точно така правете оттук нататък – преди да се мъчите да откривате топлата вода, вижте дали някой вече не я е открил. Винаги препоръчвам учениците да бъдат насърчавани

сами да търсят готови решения на задачите, а не да очакват постоянно да бъдат наставявани кое как се прави. Да, учителят отначало е пътеводител в дебрите на познанието, но в крайна сметка всеки ученик трябва рано или късно да започне да върви сам през него. А колкото до самата задача – остава да се извика въпросният метод вътре в нишката и да се реши какво точно ще се случи след като сме се ударили в автомобил – например може да се показва съобщение: GAME OVER на екрана.

Ще установите, че с натрупаните знания до този момент учениците могат да направят много елементарни игри, които изглеждат визуално коренно различно, но откъм програмни средства не изискват повече усилия от показаната с автомобилите. Хубаво е обаче всяка една от тях да има творчески елемент, с който да се измисли нещо ново и различно. Например:

- Падащи букви – от горния ръб на екрана започват да падат надолу букви. Играчът трябва да натиска съответните букви от клавиатурата, с което да ги „рестартира“ (да ги връща най-горе). Целта е да не допуснем някоя буква да падне най-долу (това ще е край на играта). Едно възможно усложнение спрямо предишната задача е да измислим начин при рестартирането на всяка буква тя да се подменя с произволна нова.
- Пресичане на улица – коли се движат в различни ленти (едни наляво, други надясно), а някакво животинче (командвано от играча) се опитва да пресече улицата, като избегне сблъсък с тях. Това е в общи линии същата задача като представената по-горе, но посоките, в които се движат обектите, са различни. Едно възможно усложнение е да направим така, че играчът да може да се движи не само нагоре-надолу, но и наляво-надясно.
- FlappyBird – пиле се движи нагоре-надолу по левия ръб на екрана, а от горния и долния ръб на екрана срещу него идват препятствия – целта е пилето да се промушва в дупката между тях. Едно възможно усложнение спрямо предишната задача е да добавите „гравитация“ – постоянна

сила в отделна нишка, която дърпа вашето пиле постоянно надолу. Също така препятствията могат да си сменят позициите – дупката между тях да минава по-нагоре или по-надолу при всяко превъртане.

- Прескачане на препятствия – героят стои в долния ляв ъгъл на екрана, а срещу него по хоризонтала започват да се движат препятствия. Целта е с натискане на бутон на мишката да ги прескачате. Усложнението спрямо задачата с колите е, че скачането плавно трябва да е с анимация – постепенно изкачване нагоре, после падане надолу, с което може да се изучи цикъл с условие.
- Стреляне по мишени – през определен период от време на три различни места на екрана се появяват мишени, които след кратко време се скриват. Целта е да се преместим в съответния улей и да стреляме бързо по мишената. Едно възможно усложнение спрямо предишната задача е да измислим как да се случи визуализирането на изстрелването на куршума (той да не достига мигновено до целта при натискане на бутона, а да прелита с кратка анимация).
- Скуош за един човек – от едната страна на екрана движите хилка – например надолу-нагоре по левия ръб, а срещу вас се движи топче, което трябва да отбивате. Ако топчето се удари в друг ръб на екрана – горен, долен или десен, то се отбива по законите на физиката (ъгълът на падане е равен на ъгъла на отражение). Целта е топчето да не мине зад вас. Усложнението спрямо предишната задача е да се измисли алгоритъм за движение на топчето – то вече не се движи само по едната ос на координатната система, а по двете. Най-лесно е за начало да се избере фиксиран ъгъл за градуса за движението и той да не се променя.
- Пинг-понг – същата като предишната игра, но от другата страна на екрана има хилка, която се движи от компютъра. Най-лесният алгоритъм за противниковата хилка е в безкраен цикъл да следи положението на топчето и да се стреми да е на същото място като него по вертикала.

Всъщност подобни игри има изключително много и децата ги познават от ежедневието си – било то на игрови конзоли, на мобилен телефон или на компютър. Всички тези миниигри биха могли да се решат на малки групи на проектен принцип – две или три деца се групират и се опитват на принципа „разделяй и владей“ да си разпределят задачи и всеки един от тях самостоятелно да разреши своята собствена задача. След това с помощта на учителя различните решения се обединяват в едно общо, а децата обясняват един на друг кой как е решил задачата. Изобщо „геймификацията“ в обучението по програмиране е една модерна тенденция, която често се употребява от различни учени [85], [84], [76], [88]. Известни учени от България също проявяват интерес към този подход [86], а от катедра ОМИ имаме и колега, който се занимава активно с темата [73].

В последните няколко години на два пъти направихме подобен експеримент (не задължително със споменатите задачи, но все свързани с някакви видове елементарни компютърни игри), изпращайки наши студенти бакалаври на стажове по софийски училища. Получаваха много добри отзиви не само от самите студенти, но и от базовите учители – учебните часове са се оживявали значително, а някои от учениците се вдъхновявали до степен, че далеч надхвърлили всякакви предварителни очаквания, защото започвали да измислят свои собствени подобрения и дори свои собствени игри. А има ли нещо по-хубаво от това?

Препоръчвам въпросните проекти накрая да получат оценка на базата на формиращо оценяване – в процеса на изграждане на проекта учителят да помага активно на учениците, като им дава насоки за това как могат да направят допълнителни подобрения, с което да си повишат оценката. В крайна сметка целта ни е да научим учениците, а не да ги наказваме когато не са се справили.

С така достигнатото ниво ще видите, че е покрита много голяма част от съществуващата учебна програма по информатика за VIII клас [64] и дори са добавени две съществени нови теми – събития и нишки. Това, което не е покрито спрямо утвърдената програма, е темата „Основи на информатиката“, която включва темите „Числата

и техните представяния“ и „Алгоритми и езици за програмиране“, както и по-задълбочено изучаване на циклични алгоритми – досега е използван само най-елементарният безкраен цикъл, но не са разглеждани цикли с предусловие, постусловие и с брояч.

Други теми също привидно липсват – например тази за тестване и верификация на програма, но моето виждане по въпроса е, че те не трябва да бъдат като самостоятелни уроци, защото тъй или иначе същите знания неизбежно се натрупват и използват в процеса на работа при всички останали (няма как да не се допускат грешки по време на писането на кода, съответно няма как да не се показват и начините за търсенето и отстраняването им). В учебната програма [64] има и леко прикрито включена работа с текстови файлове – това е принципно тежка тема, която смятам, че трябва да бъде обособена отделно.

Относно темите за цикли и едномерни масиви – много удобно е именно чрез цикли с брояч да се обхождат масиви. Затова често учителите използват тактиката тези две теми да се обединят – въвежда се понятието „масив“ и непосредствено след това започва да се обхожда с различни видове цикли. Аз лично съм противник на тази идея – предпочитам задачите да са по-практични (да не се обхождат масиви с безсмислени числа само заради демонстрацията на самото обхождане). Затова предлагам първоначално двете теми да се учат отделно.

Остава да се разгледат отпадналите теми и да се намери тяхното място. Ще започна с темата за алгоритми, която, както вече споменах, се дублира с ИТ в X клас. Смятам, че тя трябва да отпадне както от информатика за VIII клас, така и от информационни технологии за X клас. Вместо това тази тема може спокойно да се приеме в информационни технологии в прогимназията, каквато стратегия има заложена по принцип в проекта [41], а чисто теоретичното формализиране на знанията да остане в полето на висшето образование. По този начин ще бъдат постигнати следните позитиви:

- Ако например темата се изучи в VII клас, това ще доведе до облекчение за учениците, които тъй или иначе са прето-

варени от подготовката за HBO. Ако се наблегне на практически примери и се избегне теоретизацията, темата е разказвателна и не изисква заучаване действия за работа по програмен продукт, т.е. не е обременяваща за учениците (могат да се включват адекватно в нея по всяко време дори без предварителна подготовка). Освен това би имало прекрасна възможност за незабавна междупредметна връзка с компютърно моделиране, където изучаваните алгоритми могат директно да се прилагат.

- Ще има логичен преход към учебния предмет информатика в VIII клас.
- Информационните технологии за X клас ще бъдат значително по-прагматични и по-смислени – извън математическите гимназии темата няма да е висяща абстрактна тема без практическо приложение. Вместо това ще има време за още по-задълбочен фокус върху електронните таблици във възраст, в която нуждата от тях ще бъде значително по-очевидна. Един основен проблем при изучаването на електронни таблици в основното образование е мотивировката защо те са полезни и къде се използват в практиката (учениците нямат достатъчно практически опит, за да видят ползата от автоматизираната обработка на информация – за тях често учебните задачи са безсмислени, защото лесно си ги решават наум).

Вероятно темата за „Числата и техните представяния“ ще породи по-тежки дискусии. Тя е исторически наложена от много години и е била неизменна част от учебната програма по информатика. Действително двоичните числа и математиката свързана с тях са може би най-фундаменталната основа на компютърната техника, без която нищо в съвременните компютри не би било възможно. В такъв случай отпадането на тази тема от задължителната подготовка, което предлагам аз, първоначално звучи като нещо нелогично и вредно.

Не трябва обаче да забравяме, че парадигмата за уводните курсове по програмиране е сменена и вече от самото начало се

програмира на високо ниво (ООП). „Слизането“ към език за програмиране от по-ниско ниво и съответно изучаването на „механиката“ на софтуерните продукти по-близо до хардуера (без абстракции и готови библиотеки) идва значително по-късно (най-често чак в университетите в съответните курсове). В този смисъл предлагам и темата за „Числата и техните представяния“ да остане по-скоро пожелателна евентуално към избираемата подготовка.

Внимателно преброяване на предложения от мен хорариум ще покаже, че са отделени поравно часове за добиване на нови знания и такива изцяло за упражнения. Това обаче не означава, че предметът е претрупан с много теория и малко практика – в предложената методика за преподаване учениците ще извършват практически дейности дори в часовете, в които се усвоява нов материал. Затова и уроците са написани като „комбинирани“ – те са хем за нови знания, хем за упражнение на вече добити стари. Това е много важен фундаментален елемент, който е съществена част от цялостната ми визия. Естествено, внимателен прочит и сметка ще покажат и 4 свободни часа в хорариума. Това е важно технологично време за всеки клас, което обикновено се разпределя за оценяване. Определяне на входно ниво не е нужно, защото предметът се явява изцяло нов и се изучава за първи път.

ПРИМЕРЕН УЧИЛИЩЕН КУРС ПО ИНФОРМАТИКА ЗА XI КЛАС В ПРОФИЛИРАНА ПОДГОТОВКА

Както беше анализирано в предишните глави, до началото на същинското профилиране в XI клас, учениците от математическите гимназии като минимум са имали един уведен курс по програмиране с парадигма на императивно обектно-ориентирано програмиране, а практиката показва, че почти сигурно са го надградили с изучаване на ИУЧ и/или ФУЧ в XI и X клас. Напълно подкрепям визията на учебните програми на МОН, че до края на средното образование е препоръчително учениците да са се сблъскали и с различна парадигма на декларативно програмиране, като също приемам за удачен избора на езика SQL при работа с релационни бази данни. Разбира се, на пазара има огромна плеяда от все повече набиращи популярност различни видове нерелационни бази от данни, но въпреки че лично съм участвал в написването на университетски учебник [33], не смятам, че която и да е от тях би била подходяща за начинаещи. Това мое мнение не е защото твърдя, че са прекалено сложни за изучаване или използване (напротив, както всяко друго нещо, и при тях всичко може да се поднесе пред обучаемия в елементаризиран вид), а защото изначално са създадени да помагат при решението на задачи, които са извън зоната на близкото развитие на начинаещите програмисти – техният фокус и предимства спрямо релационните бази от данни се появяват само при изключително големи информационни системи, които

работят с огромни обеми от информация на множество сървъри, т.е. неща, до които учениците няма да се сблъскат в учебна среда.

Модерните софтуерни продукти и най-вече интернет приложенията са довели до тенденцията за изолиране на информацията от програмния код, а базите от данни и по-конкретно релационните с език SQL са се наложили като най-интуитивно и най-подходящо за момента решение (поне за учебна среда). Затова в софтуерните специалности на вероятно всички университети курсът е основен и обикновено се фокусира именно в релационните бази от данни. Един училищен курс би следвало да се разглежда като пропедевтика за бъдещо изучаване и съответно не трябва да се пренасят механично цели теми и учебно съдържание от университета, а да се представят елементаризирани и опростени откъм теория знания техни адаптации.

Релационните бази от данни са застъпени силно в модул 3 от учебната програма [66], но убедено мисля, че изучаването му в отделен срок от XII клас е по-скоро неуместно. Базите от данни се създават, за да бъдат в помощ и да бъдат достъпвани от софтуерни продукти, а не да са си самодостатъчни. Тоест сегашната визия, че учениците първо ще се научат да работят с бази от данни, а после евентуално ще програмират с тях за изграждане на информационни системи (модул 4 от [66]) е ненужна фрагментация – двата модула могат спокойно да се обединят в един общ училищен курс, в който да се допълват, подкрепят и надграждат. Вярно е, че учителите биха могли да импровизират чрез по-разчупени свои собствени годишни разпределения, но смятам, че би било по-добре това да е заложено още на ниво учебни програми, а не да очакваме те сами да се досещат, че знанията от различните модули всъщност се допълват едни други и е целесъобразно да бъдат обединени.

Освен това смятам, че ще е значително по-полезно, ако тези модули започнат да се изучават по-рано (още в началото на XI клас), за да може знанията от тях да се обвържат с изученото преди това. За сметка на това предлагам те да бъдат надградени с проектна дейност през XII клас. Казвам това с ясното съзнание, че въпросната промяна няма как да бъде направена чисто меха-

нично, а ще се наложи цялостно реструктуриране на тези учебни програми. Съществена част в тази моя визия включва и опростяване на учебното съдържание, като смятам, че фокусът трябва да се измести от теоретично изучаване на сложни информационни системи към създаване лично от страна на учениците на по-прости учебни варианти за такива.

В тази глава ще започна с изложение за вижданията ми как би трябвало да протече един въстъпителен елементаризиран (за училищно ниво) курс по бази от данни, който е обвързан с практическо програмиране на елементарни информационни системи. На практика това ще бъдат опростени версии на модули 3 и 4 от [66], които са преместени в XI клас, без да бъдат предхождани от модули 1 и 2 от [66]. Погледнато по-общо, предложението ми е с изучаването на нови знания от сферата на релационни бази от данни едновременно да се припомня и упражнява изучаваното преди императивно програмиране, като това се случва чрез създаване на елементарни информационни системи с някакъв вече достъпен за учениците ООП език за програмиране. По този начин смятам, че биха се получили следните позитиви:

- Няма да има фрагментация на учебния материал – обединението на модулите в един ще зададе явно връзката между знанията и тясната връзка между тях ще е по-ясно видима.
- Обвързването на релационните бази от данни с изграждането на реален софтуерен продукт ще изясни много по-ясно нуждата от тяхното съществуване.
- Опростяването на учебното съдържание ще направи материалът по-достъпен за чисто практически дейности и ще намали ненужното претрупване с информация.
- Учителите ще са свободни да надграждат програмирането от VIII клас с ИУЧ и ФУЧ в IX и X относително свободно, без да изпитват проблема на рестрикцията на сегашните модули 1 и 2 от XI клас.

Разбира се, големият проблем е, че входното ниво на учениците в различните училища ще е значително различно. От тази

гледна точка една евентуална подробна учебна програма трябва да се създаде изключително внимателно така, че да е максимално адаптивна спрямо потенциално много различни входни нива.

Логиката на стъпаловидно надграждане на знанията [4] подсказва, че е нужно първо да научим обучаемите как да създават програми (изучавайки предмети по основи на програмирането, обектно-ориентирано програмиране и поне някои структури от данни) и чак след като са усвоили тези знания, да ги научим по какъв начин е удачно да се складира и обработва информацията, с която работят в тях. Именно затова поставянето на такъв встъпителен курс в началото на XI клас е много адекватно време – има достатъчно хорариум както да се изучат основите на релационни бази от данни, така и да се упражнят чисто практически.

Както във всяка методика на обучение, и в преподаването на бази от данни учениците трябва да получат отговори на фундаменталните въпроси: „Защо?“, „Какво?“ и „Как?“ [5]. За конкретния учебен предмет можем да ги формулираме като: „Какво са бази от данни?“, „Защо да складираме информация в тях?“ и „Как да го направим?“. Смятам, че за най-ефективни отговори е нужно целенасочено упражнение с максимално практични задачи. Те трябва да са достъпни за учениците – да бъдат практически осъществими с техните текущи знания и умения. Смятам, че създаването на елементарни информационни системи е постижима цел. При това съм убеден, че учебната програма трябва да се направи така, че да е преодолима цел за абсолютно всички – трябва да се заложи и евентуална възможност предметът да бъде изучаван включително от ученици, които не са имали програмиране в ИУЧ и ФУЧ от IX и X клас. В този смисъл е добре да се постави „ниска летва“, а учителите да могат не само свободно да надграждат и усложняват учебното съдържание, но и да бъдат насърчени за това.

Защо да складираме информация в бази от данни? Сравнително лесно могат да се измислят множество прости примери за софтуерни програми, за които има нужда от това да записват данните, с които работят. Тривиалният пример е да направим програма, в която записваме списък с имената на учениците по

класове или оценките на ученик по всички учебни предмети, неговата седмична програма и др. Ясно е, че ако информацията се обработва само в структури от данни в оперативната памет и не се запише на перманентен носител под формата на текстови/бинарен файл или в база от данни, всичко написано ще се загуби при изключване на програмата. Това е бърз и лесен мотив, че трябва да имаме лесен, стандартизиран и достъпен начин за съхранение на данни. За щастие, учениците в днешно време живеят в условия на информационно общество и се сблъскват ежедневно с множество информационни системи, т.е. изобщо не е трудно да се дадат примери от реалния свят.

С уговорката, че някои ученици може вече да са имали опит с работа с файлове, а други – не, отговорът на защо да използваме бази от данни, може да се даде чрез елементаризирано обяснение практически на всекиго. Най-простият пример би бил да се запази обикновен текстови файл, в който се пазят например данните за конкретния клас. Учителят може първоначално да го демонстрира дори с най-обикновен текстови редактор (няма нужда от писане на програмен код) – задачата е информацията да се структурира така, че да може да бъде прочетена и обработена (програмистите употребяват термина „парсната“) впоследствие от някакъв програмен продукт. Смятам, че е част от „зоната на близкото развитие“ [5] на учениците да бъде поставен въпроса: Как различни програми, писани от различни програмисти, да достъпват едни и същи данни по унифициран начин? Това може лесно да се постави и като групов проект, който хем да свърши работа като опресняване на знанията по приложно програмиране, изучавано в предишни класове, хем да постави учениците в ситуация, която изисква знания, които все още нямат.

Например учителят може да раздели класа на две части и да постави задачи за създаване на два различни софтуерни продукта, които обаче да работят с различни подмножества на информация от един и същи файл. С добър подбор и планиране на учебните задачи може да се стигне лесно до ситуация, в която, когато една от програмите обработи файла, той става невалиден за другата

програма и обратно. Така учениците се поставят непосредствено в ситуация, която изисква решение – двата софтуерни продукта някак трябва да синхронизират работата помежду си. Тук се поражда и естествената нужда от изучаване на нови знания от сферата на бази от данни като вид централизирано и универсално хранилище за информация.

Естествените проблеми, които могат да се засегнат на първо място, са разделителите на информацията и нейната подредба – различните програми трябва да се „наговорят“ за единен стандарт – откъде започва името и къде свършва, какъв тип данни ще е например номерът в класа или оценката, т.н. На второ място идва проблемът с излишеството на информация – различните програми могат да боравят с подмножества (например на едни им трябва само имената на учениците от отделен клас, а на други може да им трябва обобщена информация само и единствено за оценките им). Не само това, но едни програми биха могли да поискат да добавят впоследствие допълнителна информация във файловете, която ще промени формата на записаните данни и това не трябва да ги прави невалидни за други програми. Не на последно място стоят проблемите по многопотребителски достъп и сигурност – а ще може ли да стане така, че програмите да се синхронизират, за да работят едновременно с едни и същи данни, без да си навреждат една на друга? Очевидно ако работим директно с най-обикновени текстови файлове, никак няма да е лесно за осъществяване – изправяме се пред купища от проблеми, като най-основният е как файлът да се сподели между различни софтуерни продукти на различни компютри. Ще е нужно нещо ново, което все още не е учено.

Дори начинаещи програмисти веднага ще осъзнаят множеството проблеми. По съвсем естествен път ще се достигне и до идеята за създаване на „интерфейс“ за достъп до данните, чрез който програмите да достъпват информацията – трябва да им минимум правила за достъп и правила за подредба на данните така, че различни програми да могат да работят с тях, без да си навреждат една на друга. Много удачно е тези проблеми да бъдат де-

монстрирани с практически пример – дори най-тривиалното две програми на един компютър да се опитат едновременно да отворят един и същи файл, е достатъчна мотивация за това, че ни трябва някаква система за управление (на бази от данни), която да менажира всичко, вместо ние ръчно да отваряме и пишем по файловете, в които информацията се складира.

Не само това – в практическия пример може сравнително лесно да се покаже как една програма променя файл по такъв начин, че го прави невалиден („чупи го“) от гледна точка на друга. В крайна сметка под една или друга форма ние сблъскваме учениците с елементи от т.нар. аномалии, които стоят в основата на теорията на нормализациите при релационните бази от данни. Дискутирането и практическата демонстрация на следните три въпроса сякаш, че са достатъчна база за мотивиране на въвеждането на нови знания от света на релационни бази от данни:

- Как да съхраняваме информацията по такъв начин, че да улесним бъдещото ѝ използване от други хора или компютърни програми?
- Как да управляваме (съхраняваме и обработваме) информацията си така, че да предотвратяваме в максимална степен допускането на грешки при работа с нея?
- Как да съхраняваме информацията по такъв начин, че да заема максимално малко дисково пространство, но същевременно с това да не отнема съществено количество процесорно време при нейната обработка?

Изучаването на бази от данни обаче не трябва да се свежда само и единствено до подаване на команди към някаква система. След като учениците са подготвени за нуждата от споменатия „интерфейс“ за достъп до информацията, предварително трябва да им поставим въпроса: „А по какъв начин все пак се складира тази информация зад този интерфейс?“. Това е фундаментален въпрос и покрай него може да се достигне до различни нива на задълбочаване.

В училищния курс в никакъв случай не трябва да се цели сериозна специализация, но като минимум трябва да се дадат поне

някакви основни теоретични знания. Например не е целесъобразно да се достига до подробно изучаване на формалните дефиниции на нормалните форми, нито пък да се изучава подробно математическият модел на релационната алгебра, но учениците със сигурност трябва да се докоснали и да получат поне усет за тях. Участвал съм в написване на университетски учебник [7] и съм преподавал 7 години с MySQL в Техническия университет – София и 6 години смесено с Access и MySQL в Софийския университет и отлично осъзнавам колко сложна е теорията за студенти (като минимум претрупана с огромно количество нови термини, както е анализирано в дисертационния труд на моя колега Веселин Дзиев [9]), а оттам и колко още по-трудна би била за усвояване от ученици. Затова не трябва да се надскачаме в поставените цели – напротив, трябва да сме максимално прагматични и още при планирането на курса да си дадем сметка, че много от нещата в него ще се наложи да са нарочно елементаризирани.

От предмета „Информационни технологии“ учениците са добре запознати с табличното форматиране на данни с електронни таблици. Затова е много подходящо в началото да бъде даден пример с таблица, в която има именно проблем с голямо излишество на данни. Такъв пример би бил изключително подходящ и като пропедевтика за по-нататъшното таблично представяне на данните извлечени от релационна база от данни. Обикновено учениците сами успяват да се сетят, че за да се справим с този проблем, е удачно да разделим информацията в две или повече таблици – нещо, което при релационните бази от данни наричаме „декомпозиция“. Рядко обаче успяват сами да измислят теоретичен модел на това как тези таблици биха могли да бъдат свързани логически помежду си. Затова учителите следва да дават лесни и достъпни практически примери.

Такава би била таблица с оценките на ученици от различни класове на различни училища. Учителят може да насочи учениците, че ще има огромно излишество на информация, както и че ще са възможни множество аномалии при иначе стандартни операции (например прехвърляне на ученик от един клас в друг, смяна

на име на училище, т.н.). Декомпозицията в случая би улеснила управлението на информацията и това е нещо, което учениците знаят още при изученото в темите за електронни таблици. Така ние ги докосваме още в началото до нормализацията на релационни бази от данни, но го правим без формалности, а по-скоро по усет.

Тук е мястото учителят да излезе от света на електронните таблици и да покаже аналогия с реална система за управление на релационни бази от данни (СУРБД). Там могат да се видят с графична или дори през конзолна среда как всичко се складира в таблици, но без задълбочаване подробно изследване на тяхната структура. Бих препоръчал също да се избягват ненужни в начален етап нови термини – например смятам за напълно допустимо да се използват простичките „таблица“, „ред“ и „колона“, вместо в началото трудни за осъзнаване „релация“, „кортеж“ и „атрибут“. Да, вярно е, че в базите от данни се цели в крайна сметка да се работи със значително по-високо ниво на абстракция – най-малкото говорим за „обекти и свойства“, вместо за „редове и колонии“. Не е задължително обаче тези знания да се въвеждат в самото начало. Хубаво е до тях да се достигне по естествен път и учениците сами да ги преоткрият. Това се постига по-лесно с множество учебни задачи, които се надграждат и последващото им обобщение, отколкото с въвеждащи лекции, в които се зазубрят дефиниции, които чак накрая се изясняват чрез упражнения.

Съвсем нормално и очаквано е точно на този етап да се напише и елементарна програма на достъпен за тях (вече изучаван) програмен език, с която да се направи проста заявка за извличане на информация от съществуваща таблица. Тоест да се положат основите на създаване на елементарен програмен продукт, обслужващ информационна система. Изхождайки от предишната препоръка за груповата работа по проект с файлове, може вече създадени програми да бъдат преправени по такъв начин, че да обработват и записват същата информация с помощта на споделена между тях система за управление на релационни бази от данни. Така се демонстрира пряко едно от големите предимства на тези системи – как различни приложения могат да споделят общи масиви от информация.

Нуждата от разделяне на информацията в различни таблици върви в комплект и с известна житейска интуиция как да се направи тя. Учениците например смятат за напълно очевидно, че „имената на училищата трябва да се отделят в отделна таблица“ и в началото за тях задачата за разпределяне на информация по различни таблици им се струва лесна – идва интуитивно. Всеки, който се е занимавал професионално с релационни бази от данни, знае, че това невинаги е така. Като продължение на предложената тема, би било удачно ентусиазмът, че проектирането е лесно, да бъде леко охладен. Може да се дадат примери, в които взаимовръзките между обектите нямат ясна йерархична зависимост и не са очевидни. Мисля, че трябва да се подходи внимателно – такъв пример може да бъде даден, но само бегло и без да се задълбава в прекалени теоретични подробности. Спокойно могат да се вземат класически примери за нарушаване на БКНФ (нормална форма на Бойс-Код) или 4НФ, но без да се обясняват теоретичните подробности, свързани с тях, т.е. без да се прави опит да се изяснява в детайли наученото, преди да е натрупана достатъчно практика. Естествено, тези примери не трябва да остават само на ниво проектиране, а е много удачно да бъдат обвързани със създаването на информационна система. Изключително удачно би било например да се направи целенасочено програма, която привидно работи добре, но при въвеждане на определена информация се „чупи“ и това да е резултат именно от лошото проектиране на базата от данни, с която тя работи. Това би предизвикало когнитивен дисонанс към иначе по-скучните теми за проектиране на БД. В зависимост от натрупаните умения по програмиране, подобна програма може да бъде изградена от начало до край или да бъде довършена по шаблон, който е предварително подготвен от учителя.

Оттук насетне предлагам обучението да премине със серия от типови, но надграждащи се и изцяло практически задачи. Дали да бъдат използвани шаблони и дописвани програми с графичен интерфейс, които да работят с базите от данни, или да се използва интегрирана среда, предоставяща готови графични контроли (например Microsoft Access), зависи изцяло от това как учителя пре-

ценява програмистките умения на своите ученици. Ако те са програмирали много и са добили добър опит с някой конкретен ООП език, няма никакъв проблем сами да пишат или дописват програмите, които ще обработват информацията. Ако се прецени, че нямат достатъчно умения за ефективно и бързо писане на софтуер, т.е. уменията им по програмиране не са били достатъчно развити, може да се подходи към „сглобяване“ на готови формуляри за обработка на информацията в интегрираната среда, което реално си е създаване на програмен продукт, но без да се програмира (такава впрочем е наложената традиция по много съществуващи училищни курсове, в които се използва Microsoft Access).

Със сигурност обаче се застъпвам зад идеята поне в началото учениците да програмират с реален код, с който се достъпва и извлича информация от СУРБД – ученето в среда само и единствено в един програмен продукт като Access или подобен не дава достатъчно ясна визия за същината на връзката между бази от данни и приложно програмиране. Споделям и наблюденията от [81], където авторите казват, че обучението с Access се изразжда в изучаване функционалностите на самата среда, а не в учене на бази от данни. Затова, ако е възможно, по-добре да се програмира реално, а не да се използват готови формуляри в интегрирани среди.

Рядко в книгите се дава смесено теорията за проектирането на бази от данни с тяхното практическо използване. Обикновено за практически цели има отделни учебници и книги, най-често свързани с езика SQL. Не са много книгите като тази на Рагху Рамакришнан и Йоханес Герке [80], в които се дава достатъчно последователна информация едновременно за проектирането на бази от данни и практически примери с езика SQL. В един обучителен курс е важно да се направи тази връзка между проектиране и конкретна практическа работа. С това казвам, че не трябва да отделяме много последователно време за изучаване на проблеми с проектирането и чак като учениците са уверени с него да се преминава към практически примери – това би превърнало първите часове в тип „лекция“, т.е. ще бъдат скучни. Обратно, не трябва да изправяме учениците само пред заготовки, с които да работят (изучавайки

SQL или работейки с дизайнер на заявки в интегрираната среда), хубаво е те сами да проектират част от тях, защото така знанията няма да остават непълни.

Езикът SQL би отговорил по един начин на въпроса как да складираме и използваме информация от бази от данни. По тази тема има много учебници и книги, но по мое лично виждане, смея да твърдя, че в повечето от тях има сериозни недостатъци от методическа гледна точка. Най-вече това се вижда в стандартната структура, която почти всички учебници използват като последователност за надграждане на знанията на учащите. В учебниците със смесено съдържание от теория на бази от данни и езика SQL като [80] пък често се забелязва изискване за прекалено високо ниво на предварителна подготовка, най-вече за основните знания за езика SQL – нещо, което всъщност се предполага, че трябва първо да бъде изучено в курса на обучение. За да покажем това, нека разгледаме най-често срещаната последователност на темите в повечето университетски курсове:

- Обучението по SQL започва с прости оператори SELECT за извличане на данни от базата данни.
- Надграждане на операторите SELECT чрез извличане на информация от множество таблици (JOIN).
- Вложен SELECT и извличане на информация от едни таблици на база на съдържание от други.
- Агрегатни функции, групиране и сортиране.
- Оператори INSERT за вмъкване на данни.
- Оператори UPDATE за обновяване на данни.
- Оператори DELETE за изтриване на данни.
- Оператори CREATE, DROP и ALTER за създаване на таблици.
- ... и т.н. следват по-специфични за конструкции като трансакции, VIEW, INDEX, тригери и др.

Пропуснал съм по-теоретичните теми като архитектура на СУРБД, нива на абстракция, релационна алгебра, нормални форми и т.н. Тях обикновено преподавателите в университетските курсове ги изнасят в отделен блок – било то в началото или в края на обучението, но почти винаги без да ги „размешват“ с чисто

практическите теми за езика SQL. Смятам този подход за неправилен от гледна точка на училищен курс – там не трябва да се обособяват отделни самостоятелни теми, а всичко теоретично трябва да се учи и надгражда в симбиоза с практическите упражнения.

От тази последователност се вижда, че обучаемите най-първо се научават да четат информация от бази от данни (оператор SELECT). Това едва ли следва пътя на логиката от практиката, защото проектирането на една база данни в реални условия всъщност следва следната последователност:

- проектиране на системата (ER диаграми и т.н.);
- създаване на базата данни (заявки CREATE);
- попълване на начална информация в базата данни (заявки INSERT);
- обработване на тази информация чрез SELECT, UPDATE, DELETE, INSERT.

Наистина невъзможно е да четем информация от таблица, която все още не съществува. Затова в упражненията преподавателите обикновено дават готови бази от данни, върху които започват да се изпълняват различни заявки. Моята лична практика показва, че този подход води до „накъсване на знанията“. Още в началото учениците изпитват затруднения с това да разберат какви са типовете данни (и защо са такива), как са създадени тези таблици и защо са свързани точно по този начин, откъде идва (как точно е складирана) тази информация и др. Много по-лесно би било да разберат връзките между няколко таблици, ако самите те са ги създали, отколкото, ако са им дадени наготово. Затова според мен прекалено ранното задълбаване в SELECT оператора при такива курсове води до съпътстващи проблеми в усвояването му. Бих препоръчал обучението да започне първо със създаването на таблици (съпътствано разбира се от „микровъпроси“ с казуси около тяхното проектиране), последвани от попълване на примерна информация и чак тогава да се задълбава към по-сложни заявки за нейното четене и промяна.

Като контрапункт на това предложение, колегите преподаватели изтъкват нещо, което до известна степен е вярно – от уни-

верситетските курсове знаем, че проектирането на база от данни всъщност е най-трудната част и съответно е прекалено амбициозно да се започне именно с нея. Наличието на прекалено много CONSTRAINTS и съпътстващи ги чисто теоретични проблеми от проектирането на базата от данни не само са прекалено сложна за начинаещи материя, но също така и много трудна за осъзнаване без достатъчно практически упражнения с другите оператори (SELECT, INSERT, UPDATE, DELETE). Всъщност вече бях споменал, че е най-добре човек първо да се сблъска с проблемите, преди да започне да ги разрешава, а голяма част от тях в проектирането идват от чисто практични ситуации. Именно поради тази причина, при въвеждането в езика SQL се е наложила тенденцията да се започне с най-лесното (прост оператор SELECT), което при това дава практически резултат (на екрана излизат резултатни таблици). Последното е предимство, което не може да бъде подценявано.

Така по естествен път достигахме до моето предложение за „компромисен вариант“. Предлагам първоначално да се започне с оператор CREATE (или тема „създаване на таблица“), но в свой силно опростен вариант – без да се изучава всичко в детайли. Не трябва да се търси никаква изчерпателност при дефинирането на връзките, ограниченията, дори типовете данни. Всякакви специфики ще бъдат натрупвани и изяснявани впоследствие, като това ще са вид препратки назад не само за допълване на знанията, но и за тяхното упражнение. Така проектирането и създаване на таблици няма да бъде тежко от самото начало, а напротив – ще изглежда като нещо много лесно. Така смятам, че ще успеем да се избавим от създаването на традиционния (масов) проблем, който многократно съм виждал при да кажем „по-слаби“ студенти в университета – могат да работят само с оператор SELECT и всъщност успяват да пишат доста сложни заявки с него, а всичко друго е пропуснато или недостатъчно добре научено, защото не е направена адекватна връзка между различните теми.

Поради така изложените причини, предлагам следните принципи за въвеждане на нови знания и съответно последователност от теми:

- Нови типове от данни и показването на тяхната аналогия в езика за програмиране, който познават, може да се дават само при нужда и без излишни подробности.
- В началото няма нужда от сложни примери с база от данни, съставена от голямо количество таблици със сложни взаимовръзки – таблиците трябва да са малко на брой така, че да може учениците лесно да помнят и да си представят мислено взаимовръзките между тях дори без да знаят как формално се правят истински ER диаграми. Освен това е добре самите ученици да са ги създали.
- Не е нужно да се изучават формално ER диаграмите – достатъчно е да се покажат прости примери за такива в средата, в която се работи (например Relationships в Microsoft Access дава много интуитивен изглед, който не изисква много обяснения).
- Понятията „първичен ключ“ и „външен ключ“ в началото е добре да се дадат съвсем елементарно и в най-простата им форма – със сурогатни ключове (id).
- Създаването на таблици трябва да е максимално опростено – в началото може да не се обръща внимание на иначе важни свойства като наличието или не на ограничение NOT NULL, DEFAULT стойности, CHECK правила, каскадно изтриване и обновяване, UNIQUE ключове, индекси.
- След това във вече създадена от учениците база от данни, може да започнат да се тренират заявки INSERT, но отново от най-прост вид и без изчерпателно да се демонстрират множеството проблеми откъм интегритет на данните – такива могат само да се загатват.
- Обучението може да продължи с изучаване на прост оператор SELECT, с който се четат въведените от учениците данни от създадените от самите тях таблици.
- Леко усложняване на знанията чрез добавяне на някои (не изчерпателно!) прости изрази в WHERE, както и евентуално сортиране с ORDER BY.

- Надграждане на оператор SELECT чрез извличане на информация от множество таблици (JOIN) – обикновено само между две таблици и само с INNER JOIN.
- Въвеждане на вложен SELECT (той може и да бъде пропуснат в училищния курс).
- Разглеждане на агрегатни функции, групиране и клауза HAVING.
- Надграждане на знанията за създаване на таблици чрез допълнителни знания, които са придобити чрез операторите SELECT – става дума за целенасочено допуснати грешки при проектирането, които впоследствие довеждат до неправилни резултати. Така могат поетапно да се изучат някои CONSTRAINTS, с които да се грижим за по-добър интегритет на данните.
- Изучаване на прости оператори UPDATE.
- Изучаване на прости оператори DELETE, като учениците трябва да се сблъскат с проблемите при каскадно изтриване на данни. Следва непосредствено „връщане“ към оператор UPDATE, където се отбелязва, че съществуват сходни проблеми, които са още по-трудно забележими от разработчиците (може и да бъде пропуснато, но ако все пак се изучава, няма нужда от множество примери).
- Обобщаване на знанията за проектиране на бази от данни с оглед на натрупания вече практически опит.
- По всички изредени по-нагоре теми да има съпътстващо съставяне на елементарна информационна система, която работи с данните от поставените учебни казуси!

Смятам, че така представената методика ще доведе до много по-плавна и по-ясно усвояване на нужните знания, без да се получава фрагментирането им в измамно изглеждащи самостоятелни теми. Това, че в началото учениците не получават изчерпателна информация за операторите, не означава, че те няма да бъдат ефективно усвоени по-късно по време на курса. Това е и по-адекватният подход за тях спрямо традиционните курсове от висшите училища – може би университетските преподаватели се страхуват

да „свалят нивото“ и да преподават част от материала прекалено опростено за известен период от време. За мен обаче е по-важно обучаемите да разберат материала, та макар и в опростен вид, вместо да го „зазубрят“, без да изпитват вътрешна потребност за това. Първото дава увереност на учениците, а второто ги отегчава, а в определени случаи дори отчайва.

Тук е мястото да отбележа специално учебника на Пол Дюбуа [70], който е изключително добре структуриран за едно целенасочено практическо обучение за системата MySQL. Именно подобен стил на изложение е много подходящ за практическо ръководство. Той не е подходящ за училище, защото е прекалено голям и детайлен, т.е. би изискал хорариуми, с които не разполагаме, но негово подмножество и подобен стил на изложение е изключително адекватен за обучение на встъпителен курс по бази от данни.

Както споменах, тази методика предполага курсът по бази от данни да е встъпителен, но в никакъв случай не изчерпателен. Чрез него учениците биха добили основни практически умения за работа с бази от данни, но не бихме казали, че ще бъдат изградени специалисти. Напротив – теоретичните знания ще са бегли и недостатъчни. Поради тази причина курсът в никакъв случай не замества университетския, а само подготвя хората за него. Обучаемите ще имат понятие за това какво са базите от данни и как се работи с тях, но самите те няма да могат да работят с тях напълно ефективно в практиката, ако не надградят своите знания. Дали това е достатъчно? Бих казал „да“ що се отнася до човек, който ще завършва средно образование в профилирана подготовка.

Лесно може да се забележи, че в предложената концепция предлагам значително опростяване на учебното съдържание спрямо предложения модул 3 и модул 4 в [66]. Тя е свързана с едновременно елементаризиране на учебното съдържание, обвързване на чисто теоретичните теми с по-практичните такива и непосредственото прилагане на всяко изучавано нещо от света на базите от данни в практическа употреба при информационни системи.

Освен това убедено смятам, че курсът трябва да докосне учениците повече до практическата работа с бази от данни, а не да

ги засипва с теоретичните основи на релационния модел. Теоретичните знания са ценни и е хубаво да ги има, но смятам за прекалено амбициозно и лимитиращо да се прави генерализация и да се поставя условие да се знаят от всички деца в съответния профил. Много по-добре е да се наблегне на практични упражнения и да се научи като цяло по-малко и непълно, но да бъде научено, отколкото да се поставят свръхамбициозни програми с куп теоретични знания, които по-скоро няма да бъдат усвоени от повечето ученици. Затова мисля, че огромна част от темите (изучаването на нивата на абстракция, видовете зависимости (функционална, транзитивна, множествена) и нормални форми, задълбочаването в логическото и физическото представяне на данните) и формалното изучаване на основни операции в релационен модел (проекция, селекция, декартово произведение, обединение, сечение, разлика, деление, т.н.), не да отпаднат напълно, но да не бъдат поставяни като формални изисквания за подробно изучаване в учебната програма. Предлагам вместо това да бъдат само загатвани и да се изучават по-задълбочено само според преценката на самия учител.

Смятам, че изучаването на бази от данни и информационни системи в профилирана подготовка, а всъщност и в курсовете по профилирана подготовка като цяло, трябва да дава обща рамка за най-основните неща и съответно да се поставя „ниска летва за прескачане“, а учителят да има свобода да надгражда и допълва по своя собствена преценка спрямо индивидуалните способности на учениците. Това ще стимулира както творчество при учителите, така и индивидуализация в обучението за учениците.

Освен това мисля, че курсът не трябва да си бъде самостоятелен, а да е обвързан със силни междупредметни връзки с всичко преди това изучено. В този смисъл е добре учебното съдържание да е тясно свързано с профила на учениците в съответното училище, а не още на ниво „учебна програма“ учителите да бъдат лимитирани в това какво точно и как да се изучава. Например още в първата тема от модул 4 от [66] се въвеждат понятия като „система за управление на човешките ресурси“ и „система за обработка

на транзакции“ (правилно е да се изписва „транзакция“, но така е написано в документа). Не е невъзможно да се дадат точно тези примери за информационна система, но защо трябва да е нужно учителят да бъде ограничен да ползва точно тях, а не нещо друго по свое собствено усмотрение?

Относно хорариумите, ще се види, че в учебната програма [66] са предвидени 72 часа за модул 1, 72 часа за модул 2, 72 часа за модул 3 и 54 часа за модул 4. Веднага може да се забележи, че с даденото от мен предложение се случва следното:

- Модули 3 и 4 се обединяват в едно и се преместват на мястото на модули 1 и 2, което включва и леко повишаване на хорариума им.
- Опростяването на учебното съдържание във въпросните модули е компенсирано с добавяне на изискване за многократно повече практическо програмиране.

Мисля, че няма нужда от убеждение, че поставената цел е постижима в рамките на 144 часа, колкото е хорариумът за профилирана подготовка в XI клас. Всъщност поставените формални изисквания за количество нови знания няма да са много и съответно учителите ще имат огромно поле за действие за екипни и проектни дейности и/или преподаване на допълнителни знания по съответните модули. Те ще могат да правят това по свое собствено усмотрение. Ценно е в профилирането на учениците и постига висока степен на индивидуализация на обучението им. Въвеждането на нови знания няма да отнема огромната част от учебното време (в текущата учебна програма за практически дейности се препоръчват „до 18 часа“, което е съответно 25% от учебното време за модул 3 и 33% от модул 4), а напротив – значително повече от 50% от учебното време ще бъде отделено за практически дейности. Това е и основният смисъл на метода на „учене чрез правене“, който активно застъпвам през цялото време в настоящия труд.

Не на последно място бих искал да предложа и една концепция [78], с която лично съм експериментирал в университетски курс в Техническият университет – София. При наличие на достатъчно време, учителят би могъл да показва как една и съща задача

може да бъде решена по различен начин. Това обогатява кръгозора на обучаемите и ги подготвя значително по-добре за решаване на по-сложни задачи в бъдеще. Би било добра практика също така при реализиране на софтуер, който работи с базата от данни, да бъдат употребявани параметризирани заявки. Това е важна концепция, защото в практиката се използват изцяло такива. Практически пример за нуждата от употреба на такъв тип заявки може да се направи например чрез демонстрация на инжектиране на нежелан код (SQL Injection), но трябва да се има предвид, че е сложна тема и едва ли ще бъде достъпна за усвояване от всички ученици.

ПРИМЕРЕН УЧИЛИЩЕН КУРС ПО ИНФОРМАТИКА ЗА XII КЛАС В ПРОФИЛИРАНА ПОДГОТОВКА

Последният клас в средното образование е по-особен от останалите поради три основни съображения:

- Държавните зрелостни изпити (матурите) за завършване на образователната степен обикновено се явяват като приоритет за обучаемите.
- Учениците обикновено са фокусирани към кандидатстване във висше училище и съответно се готвят за предстоящи кандидатстудентски изпити.
- Учениците навършват пълнолетие, което води до определен психологичен ефект у тях.

Влизайки в контекста на математическите гимназии и съответния профил, в който се изучава засилено информатика, може да се твърди спокойно, че повечето ученици в XII клас започват да отдават значително по-голямо внимание на изучаването на математика и БЕЛ (и евентуално трети предмет, по който са избрали да се явят на матура). Поради тези причини обикновено някои учебни предмети започват да бъдат отхвърляни на заден план като „второстепенни“. В частност това се случва никак нередко и с обучението по информатика.

Преодоляването на тези обективни негативи налага и внимателно преосмисляне на съдържателната част на училищния курс в този клас. Залагането на високи цели към усвояване на нови зна-

ния и добиване на нови умения в такава ситуация е нереалистично. То по-скоро ще доведе или до конфликтна ситуация (недоволни от строг учител ученици), или до „проформа обучение“ (фиктивно проведени часове), но не и до реални позитивни резултати. Много по-прагматично е да се заложи такава учебна програма, която не обременява учениците със задължително изучаване на големи обеми от нови знания, а по-скоро ги води през активно упражнение и обобщение на вече добити в предишни години знания и умения, покрай които **по тяхна собствена инициатива** се научава и нещо допълнително. Хубаво е дванадесетокласниците да усетят обучението по по-неангажиращите ги със завършване и кандидатстване учебни предмети като лесно и приятно, но същевременно с това да не бъде безполезно.

Освен това убедено смятам, че в профилираната подготовка за XII клас, която е финален етап от гимназиалния курс, трябва да се заложи активно на творчество (създаване на авторски софтуерни продукти) чрез групова работа по проектна дейност. Обучението може да се фокусира изцяло върху реализирането на практически полезен продукт, който не е заложен по предварителни критерии (калѓп, по който трябва да бъде създаден), а самият план за него да се изгражда адаптивно и динамично от самите обучаеми. В този смисъл като нови знания предлагам да се заложи по-скоро на неформално изучаване на типични правила за управление на софтуерни проекти, а не на изучаването на нови (за обучаемите) технологии.

Един подобен курс би могъл да започне с обща дискусия на целия клас, в която чрез метода на „мозѓчна атака“ (brainstorm) или подобен да се активира процес за измисляне на идея за изцяло нов софтуерен продукт, който в течение на учебната година да бъде създаден от всички заедно (в голям екип) чрез проектна дейност. Активното включване на всички участници, споделянето на личния опит на всеки с различни софтуерни продукти и в крайна сметка изборът на такава идея, която да е интересна за всички, е от първостепенно значение. Фантазията трябва да се толерира, а обсъждането на чисто техническите въпроси като „с каква техно-

логия“ и „кое как ще се направи“ трябва да се отхвърли категорично за по-нататъшен етап. В началото е важна само идеята.

След определянето на идеята, на ред е създаването на първоначален план за действие. В него проектът трябва да бъде разбит на йерархична серия от задачи, които впоследствие да се приоритизират по степен на важност. Например, ако сме решили да създаваме автомобил, изключително важно е да се направи адекватен негов двигател, а каква точно тапицерия на седалките ще сложим и с каква боя ще го боядисаме са неща, които могат да почакаат. В крайна сметка се очаква в този план да се открият основните задачи и техни подзадачи, всяка от които би могла да бъде решима сама по себе си (т.е. разбили сме голямата идея на серия от по-малки части на принципа „разделяй и владей“). Именно тези подзадачи, подредени според тяхната важност, ще следва да бъдат решавани по-нататък в курса.

Всяка задача е добре да бъде решима в определен срок – обикновено между 4 и 5 учебни седмици, но (поне по първоначален план) не повече. Ще наричаме тези периоди „цикли за работа“. В началото на всеки цикъл се започва с обща дискусия върху конкретната задача, в която всеки ученик трябва да представи своите идеи и да предложи каква част от работата е способен да извърши според себеоценка на собствените си възможности. В този период се вижда ясно какви знания трябва да се припомнят и какви знания липсват в екипа. Липсващите съответно трябва да бъдат научени с помощ от учителя.

Идеята на подобно събиране и обсъждане е класът да се раздели по естествен път на базата на взаимопомощ. Когато се определят основните части от всяка задача, учителят ще комбинира учениците по такъв начин, че да се обучават и взаимно допълват. Същото нещо се повтаря в началото на всеки нов учебен час от въпросния цикъл – учениците представят не само кой какво е свършил до този момент, а трябва да бъдат насърчени да споделят за трудностите, до които са достигнали, и да търсят помощ от другите, за да ги преодолеят. Това само по себе си може и дори трябва да включва и разместване на хора от една подгрупа в дру-

га. Така се стимулира взаимнообучение между учениците, като едни споделят опита си с други.

Целта на всеки цикъл е да се завърши с готов модул от крайния софтуерен продукт, а този модул трябва да бъде получен чрез сглобяване на по-малките негови парчета, създадени от различните групи от ученици. Това особено в началото се случва трудно чрез процес на адаптиране на написаното от тях към написаното от останалите (не може да се очаква, че абсолютно всичко е извършено коректно според плановете). С подобно разделяне на задача на подзадачи, групиране на обучаеми и последващо обединение на решенията, съм експериментирал по време на курсовете, които съм водил със студенти от първи курс в Югозападния университет „Неофит Рилски“ [79], а подробно редица интерактивни методи за обучение с групови задачи съм разглеждал в докторската си дисертация [29].

Финалната фаза на цикъла включва отново дискусия, в която се цели постигането на рефлексия – самоанализ на постигнатото от всеки поотделно, обсъждане как да се подобри работата в бъдеще при следващия цикъл и подобряване на генералния (общия) план на задачата. Последното е важно, защото именно по време на практическата работа на учениците обикновено им идват допълнителни идеи, някои от които биха могли да доведат до радикални промени. Такива промени не трябва да бъдат парирани, а напротив – добре е да се обсъдят и развият.

Оценките на учениците могат и препоръчвам да се дават на базата на формиращо оценяване. Тоест те не трябва да са наказателни, а е желателно да са поощрителни. Идеята е всяка дейност от всеки цикъл да бъде предварително оценена на определен брой точки, които учениците (извършвайки успешно дадената дейност) да могат да натрупват. Ако даден ученик не успее да се справи с дадена дейност, това няма да води до порицание – ще може отново да изкара точките, като му бъде помогнато, самият той помогне на друг съученик или като бъде пренасочен изцяло към екипа за друга алтернативна задача. Така оценката е за самия процес на работа, а не върху крайния създаден продукт.

Може да се направи аналогия на предложената методология за изготвяне на групов проект с тази на гъвкавите (agile) методологии за разработка на софтуер и по-конкретно със Scrum [83]. Учителят влиза в ролята едновременно на собственик на продукта (product owner), управляващ списъка със задачите, и ръководител (scrum master), организиращ екипите; учениците влизат в ролята на екип от разработчици (development team), а циклите са така наречените спринтове (sprints). Това, разбира се, не е единствената възможна методология, а има още много възможности. При всички положения обаче е важно да се работи по гъвкав адаптивен (Agile [68]) модел, а не по предварително зададени планове и критерии. Важно е колкото се може повече самите ученици да планират и организират работата си, а не да работят само по изпълнение на предварително заложи от учителя задачи.

Научни изследвания като [74], [77] и [82] показват много оптимистични резултати от въвеждането на тази практика. В предложението от мен курс обаче не само се бяга от традиционния водопаден модел (waterfall) на работа по проекти, но и се увеличава изключително много обхватът на самия проект – традиционно за екипна дейност по проекти се отделя съвсем малко време в края на учебната година, докато тук цялата учебна година се отделя за изработката на един или два (по един на срок) големи проекти. Ключово е, че се залага един изключително евристичен дух на работа, в който учениците сами дават идеите, дефинират и разпределят задачите помежду си. Именно това е основната творческа дейност, която ще ги подготви за бъдещо развитие на техни компетенции.

Въпреки че привидно няма заложиено изучаване на нови знания, такива всъщност въобще няма да липсват. Самата методология за разработването на проекта е именно такъв ценен опит. Отделно от това в процеса на работа неизбежно ще се появява отново и отново нужда от изучаване на нови неща. Всъщност работата по такъв модел е толкова динамична, че всякакви предварителни очаквания са просто гадаене. В определени случаи на учениците ще се наложи да изучават допълнително определени структури от данни, в други ще трябва да задълбаят в някои определени ал-

горитми, даже може да се наложи изучаването на изцяло непознати технологии. Важни сами по себе си са процесът на работа, сработването в екип и развитието на евристични способности за решаване на задачи, а не крайните решения на самите задачи и изучените покрай тях нови знания. Никак няма да е неочаквано различните подгрупи на един и същи клас да натрупат изключително разнопосочни знания и умения в зависимост от разпределението на задачите.

Важно е да се отбележи, че при така представената методология се очаква част от учениците да се увлекат да довършват въщи работата, която са започнали в час. Наблюдавал съм този ефект многократно при работата си със студенти, а също така съм го изпитвал и върху самия себе си, докато бях ученик в профил „Програмни продукти и системи“ във вече несъществуващото 151 СОУПИ, където именно работата по проекти и по групи в часовете по информатика беше изключително силно застъпена като практика. Това е чудесно и трябва да бъде насърчавано, но поставя и някои технологични проблеми, а именно възможност разработените програмни модули от часовете да се синхронизират с доработеното въщи, при това продължавайки интерактивността на груповата работа. Тук не говоря само за публикуване на кода в системи за контрол на версиите като Git и SVN, а и за предоставяне на възможност за синхронна и асинхронна комуникация между всички ученици и учители, като това да става публично (за целия клас), групово (вътрешна дискусия между членове на работна група) и лично (между двама участници).

Един очевиден кандидат са LCMS системи като Moodle и Blackboard, които училищата обикновено използват и имат налични като инсталация, но също така е възможно с лекота да се направят и по-специализирани инсталации с безплатни софтуерни продукти. Подобни въпроси разгледах в своя дисертационен труд [29], където реализирах практически такава система за обучение на студенти по линейна алгебра. Популярен термин за това е уеб 2.0 обучение – термин, който съм разглеждал в контекста на математиката в [31].

За щастие, интерактивната комуникация по информатика е значително улеснена, защото липсват голяма част от технологичните проблеми, пред които се сблъскват останалите учебни предмети, като например математиката, за която съм извършвал изследвания в [30], [35]. По-трудно при дистанционното обучение за обучаемите всъщност се явява синхронизирането на кода. Употребата на системи за контрол на версиите, в случая е много важна, защото не само ще им помогне, но и ще ги подготви за реалния свят. Това обаче само по себе си е допълнително време, което трябва да се инвестира. Бих препоръчал среда като Elegit [87], защото по принцип Git е не само една от най-популярните среди в практиката, но е сравнително лесна за начинаещи [20], [43].

ПОДГОТОВКАТА НА УЧИТЕЛИ ПО ИНФОРМАТИКА ВЪВ ВИСШИТЕ УЧИЛИЩА

Професионална квалификация „учител“ и в частност „учител по информатика и информационни технологии“ се дефинира чрез наредба на МОН [22], изменена през месец февруари 2021 г. с постановление [36]. В нея са дефинирани минималният брой на задължителните учебни дисциплини и техният минимален хорариум в академични часове, както следва:

- Педагогика – 60 часа (от които минимум 30 часа лекции).
- Психология – 60 часа (от които минимум 30 часа лекции).
- Методика на обучението по...
 - при един учебен предмет – 90 часа (от които минимум 30 часа лекции);
 - при два учебни предмета – 120 часа (от които минимум 30 часа лекции) общо.
- Приобщаващо образование – 30 часа;
- Информационни и комуникационни технологии в обучението и работа в дигитална среда – 30 часа.
- Компетентностен подход и иновации в образованието – 45 часа (от които минимум 15 часа лекции).

Казусът с минималните хорариуми по дисциплината „Методика на обучението по...“ при обучение на учители с квалификация по два учебни предмета дълги години не беше уреден. В тези случаи беше наложително курсовете по методика на обучението

да бъдат два. Чак с постановление [36] беше ясно дефиниран минимален общ хорариум за двата учебни предмета, което изчисти възможните разнопосочни тълкувания.

Във всички български университети при обучение на учители по информатика и информационни технологии е приета практиката да има допълнително т.нар. училищни курсове, в които се разглеждат основните теми от учебното съдържание в основните и средните училища. В частност за информатика и ИТ те са:

- училищен курс по информатика
- училищен курс по информационни технологии.

В някои университети (например Югозападен, Шуменски и Русенски) тези два курса са обединени в един общ. Също така наредбата дефинира задължително записване на избираеми дисциплини със съответен минимален хорариум от 30 часа в две групи, като обучаващите се задължително избират по две дисциплини от всяка:

- педагогически, психологически и частнодидактически;
- интердисциплинарни и приложно-експериментални дисциплини, ориентирани към ключови компетентности и свързани с професионално-педагогическата реализация на учителите.

Накрая наредбата дефинира задължително записване на поне една:

- факултативна дисциплина (минимум 15 часа).

Може би най-същественият и основен стълб в университетското обучение на учители е отреден за педагогическите практики. Много трудно е един човек да се подготви само чрез аудиторни учебни занятия и самоподготовка – нужно е потапянето му в работна среда. Това особено силно важи за учителите и се знае още от зората на науката методика на обучение. Например още на първата страница в книгата за методика на обучение в прогимназиите на Първан Заимов [12] пише, че „Методиката е безсилна да разкрие творческото обучение, докато практиката не го създаде!“. Поради тази причина и аз смятам, че най-ценното по време на обучението на един учител е именно досегът му с реална учебна среда в ролята на стажант-учител.

В наредбата на МОН задължителните педагогически практики са три:

- Хоспитиране (30 часа) – наблюдение и анализ на педагогически ситуации, уроци и/или други организационни форми в детски градини или училища, осъществявано под непосредственото ръководство на преподавател от висшето училище.
- Текуща педагогическа практика (60 часа) – посещение, наблюдение и провеждане на педагогически ситуации, уроци и/или други организационни форми в детски градини или училища съвместно с учител наставник под ръководството на преподавател от висшето училище с цел подготовка за стажантската практика.
- Стажантска практика (90 часа) – самостоятелно участие на обучаващите се в образователния процес чрез провеждане на педагогически ситуации, уроци и/или други организационни форми в детски градини или училища, провеждани под ръководството на учител наставник и преподавател от висшето училище.

Участвайки в стажантската практика, студентите придобиват статут на стажант-учители. В наредбата е дефинирано, че броят на педагогическите ситуации или уроците, които всеки стажант-учител трябва да проведе самостоятелно, не трябва да е по-малък от 15 и не по-голям от 22. Вижда се, че по дефиниция хоспитирането е отредено само за наблюдение на уроци и последващо тяхно конфеиране, текущата педагогическа практика включва както наблюдение и конфеиране, така и самостоятелно провеждане на поне един урок, а основната тежест за провеждането на уроци от страна на студентите е оставена за стажантската практика. По естествен път това градиране в степента на сложност предполага и в какъв ред е редно да се извършват практиките – първо хоспитиране, след това текуща педагогическа практика и накрая стажантска практика.

При хоспитиране и текуща педагогическа практика обучени-ята обикновено се извършват с група студенти (в Софийския уни-

верситет групите са от 6 до 8 души), докато стажантската практика е самостоятелна. Характерна трудност при груповото посещение на учебни часове по информатика или информационни технологии в училище е, че те се правят в компютърни зали, където пространството обикновено е ограничено и това поражда значителни неудобства на всички. Причината е, че за часовете по информатика и ИТ класовете се делят наполовина, т.е. учениците са по-малко и съответно училищата създават своите компютърни кабинети в зали с по-малко места. Практиката ми показва, че брой от шест стажант-учители отива към границата на разумното за наблюдение на уроци в компютърна зала и всеки допълнителен студент над този брой обикновено води до проблеми от типа къде да се позиционира така, че да не пречи, а в залата става задушно, и т.н. За съжаление, шестима студенти се явява и минимален брой стажанти в група според Правилника на Софийския университет. Затова чисто организационно хоспитирането и текущата педагогическа практика по информатика и ИТ са по-специфични спрямо еквивалентните практики по други учебни предмети.

В Софийския университет „Св. Климент Охридски“, специалност „Математика и информатика“, до 2020 г. включително, „Хоспитиране по информатика и ИТ“ се провежда в четвърти курс, първи семестър, „Текуща педагогическа практика по информатика и ИТ“ в четвърти курс, втори семестър, а „Стажантска практика по информатика и ИТ“ в четвърти курс, втори семестър. Курсът „Училищен курс по информатика“ е в трети курс, втори семестър, а „Училищен курс по информационни технологии“ в четвърти курс, първи семестър. Също така задължителният „Методика на обучението по информатика и ИТ“ се провежда в четвърти курс, първи семестър [50].

Пловдивският университет „Паисий Хилендарски“ има специалности „Математика, информатика и информационни технологии“ и „Информационни технологии, математика и образователен мениджмънт“. Там академичната година е разделена на „триместъри“ – есенен, зимен и пролетен. „Хоспитиране по информатика и информационни технологии“ се провежда в есенен, „Текуща

педагогическа практика по информатика и информационни технологии“ в зимен, а „Стажантска практика по информатика и информационни технологии“ в пролетен триместър на четвърти курс. И при двете специалности „Училищен курс по информатика“ е през есенния триместър на четвърти курс, а „Училищен курс по информационни технологии“ през зимния триместър на четвърти курс. Курсът по „Методика на обучението по информатика и информационни технологии“ се провежда през зимния триместър на трети курс, с което предхожда споменатите практики [49], [51].

Югозападният университет „Неофит Рилски“ има специалност „Педагогика на обучението по математика, информатика и информационни технологии“. Там „Хоспитиране по информатика и ИТ“ е в трети курс, първи семестър, „Текуща педагогическа практика по информатика и ИТ“ в четвърти курс, първи семестър, а „Стажантска практика по информатика и ИТ“ в четвърти курс, втори семестър. „Училищен курс по информатика и информационни технологии“, както подсказва самото име, е общ за двата учебни предмета и се провежда във втори курс, втори семестър. „Методика на обучението по информатика и ИТ“ се провежда в трети курс, първи семестър [52].

Великотърновският университет „Св. св. Кирил и Методий“ има специалност „Педагогика на обучението по математика и информатика“. „Хоспитиране по информатика и информационни технологии“ се провежда в трети курс, втори семестър, „Текуща педагогическа практика по информатика и информационни технологии“ в четвърти курс, първи семестър, а „Стажантска практика по информатика и информационни технологии“ в четвърти курс, втори семестър. „Училищен курс по информатика“ и „Училищен курс по информационни технологии“ се изучават във втори курс, втори семестър. „Методика на обучението по информационни технологии“ се провежда в трети курс, втори семестър [53].

Шуменският университет „Епископ Константин Преславски“ има специалност „Информационни технологии, информатика и математика“. „Хоспитиране по информатика и информационни технологии“ се провежда в четвърти курс, първи семестър, „Те-

куща педагогическа практика по информатика и информационни технологии“ в четвърти курс, втори семестър, а „Стажантска практика по информатика и информационни технологии“ в четвърти курс, втори семестър. „Училищен курс по информатика и информационни технологии“ е общ за двата учебни предмета и е в трети курс, втори семестър. „Методика на обучението по информатика и информационни технологии“ е в четвърти курс, първи семестър [48].

Русенският университет „Ангел Кънчев“ има специалност „Педагогика на обучението по математика и информатика“. „Хоспитиране по информатика и информационни технологии“ се провежда в трети курс, втори семестър, „Текуща педагогическа практика по информатика и информационни технологии“ в четвърти курс, първи семестър, а „Стажантска практика по информатика и информационни технологии“ в четвърти курс, втори семестър. „Училищен курс по информатика и информационни технологии“ е общ за двата учебни предмета и е в трети курс, първи семестър. „Методика на обучението по информатика и информационни технологии“ се провежда в трети курс, втори семестър [54].

Обобщено сравнение на учебните планове на шестте университета е дадено в таблица 11. Забелязва се ясно, че наложената практика във всички университети е педагогическите стажове да се явяват като заключителна фаза от бакалавърската програма. Профилирането с основните училищни курсове и курсът по методика предхождат или в най-лошия случай се провеждат паралелно с тях.

Таблица 11. *Разпределение на основните педагогически дисциплини и педагогически стажове по години и семестри в основните пет университета, които подготвят учители по информатика и ИТ в бакалавърски специалности, свързани с математика и информатика*

Висше училище/ предмет	СУ МИ		ПУ МИИТ, ИТМОМ		ЮЗУ ПОМИИТ		ВТУ ПОМИ		ШУ ИТИМ		РУ ПОМИ	
	Год	Сем	Год	Три	Год	Сем	Год	Сем	Год	Сем	Год	Сем
УКИ	3	2	4	1	2	2	2	2	3	2	3	1
УКИТ	4	1	4	2			2	2				
МОИИТ	4	1	4	2	3	1	3	2	4	1	3	2
ХИИТ	4	1	4	1	3	1	3	2	4	1	3	2
ТППИИТ	4	2	4	2	4	1	4	1	4	2	4	1
СПИИТ	4	2	4	3	4	2	4	2	4	2	4	2

Това, разбира се, не са единствените педагогически специалности, които дават учителска правоспособност по информатика и ИТ – такива има във Физически факултет и Факултет по химия и фармация на Софийския университет, Филиал Силистра на Русенския университет и др. От всички разнообразни учебни планове, единствено този на специалност „Химия и информатика“ в Софийския университет започва по-ранно профилиране с педагогически дисциплини по информатика и ИТ като въвежда „Училищен курс по информационни технологии“ в първи курс, втори семестър, а „Училищен курс по информатика“ във втори курс, първи семестър. В много университети има и задочно обучение за учители, а някои дори са в процес на започване на обучение в дистанционна форма.

Анализът на това кой учебен план е по-добър и по-ефективен е трудна задача. Би било некоректно също да се разгледат резултатите от класациите по рейтинговата система на МОН, защото не е задължително да има корелация между добър учебен план и висок рейтинг – престижът на университета, географското му положение и много други фактори влияят силно върху резултатите за реализацията на студенти на пазара на труда, а освен това рейтингите включват и редица фактори като научна и издателска дейност

и др. При евентуално анкетиране на преподаватели биха се намерили субективни фактори, които са основани на лични интереси на съответни сфери на влияние в отделните факултети. Поради тези причини е много трудно да се говори за обективни критерии за оценка на учебен план. И все пак на базата на натрупания исторически опит и видимото сходство между учебните планове във всички разгледани специалности на различните университети могат да се открият някои основни положения:

1. Поради икономически съображения (но не задължително само такива), най-често в първите две години студентите учат общи за всички специалности на съответния факултет дисциплини, а специализацията им с училищните курсове започва към средата на обучението в бакалавърска степен. В половината от разгледаните университети тясната специализация се случва дори по-късно, като училищните курсове и методиката са сложени в края на трети и дори в четвърти курс.
2. Училищните курсове се очаква да предхождат практиките. Нормално е студентите първо да са натрупали знания и умения в учебна среда, преди да бъдат изпратени да ги практикуват. Това категорично не е спазено в Пловдивския университет и до 2020 г. също така и при единия от училищните курсове в Софийски университет, където се застъпва с хоспитирането.
3. Курсът по методика на обучението по информатика и ИТ се провежда след като бъдат преминати училищните курсове. Това е напълно нормално с оглед на това, че първо трябва да бъдат преговорени, осъвременени и обобщени знанията по тези учебни предмети, а след това да се дискутира методиката за тяхното преподаване. Този принцип не е спазен само в Софийския университет и в Пловдивския университет с курсовете им по УКИТ. Освен това единствено във Великотърновския университет се наблюдава повече от един семестър разлика между училищните курсове и курса по методика – това едва ли е значим

проблем, но може да се помисли за скъсяване на дистанцията, за да може да се получи по-плавен преход между предметите, тъй като те са неизбежно свързани.

4. Курсът по методика на обучението по информатика и ИТ навсякъде без Пловдивския университет се провежда паралелно с хоспитирането (в Пловдив се провежда паралелно чак с текущата педагогическа практика). Това може да се отчете и като потенциално добра практика от гледната точка, че би дало възможност на студентите да правят непосредствена съпоставка между ситуациите, които наблюдават в реално училище, и това, което им се преподава по учебния предмет в университета. Вероятно може да се потърси и историческа връзка между хоспитирането и методиката от по-стари учебни планове, защото на пример такава беше доскорошната практика в Софийския университет, при специалност „Математика и информатика“, двете бяха обединени в един общ предмет, наречен „МОИИТ с хоспитиране“. Основната причина за подобно обединение някога е било разделението между аудиторна и извънаудиторна заетост (преподавателите имат полза от аудиторна и почти никаква за натоварването си от извънаудиторна заетост, каквато до 2020 г. бяха практиките и стажовете).
5. Хоспитиране, текуща педагогическа практика и стажантска практика се явяват крайна фаза от обучението на бъдещите учители. Нормално е те да са в различни семестри и в точно този изреден ред, защото по философия се надграждат една друга – логика, която не е спазена до 2020 г. в Софийския и в Шуменския университет, където последните две практики са сложени в един и същи семестър.
6. Дали училищните курсове са обединени в един или са разделени в два различни учебни предмета не е от особено значение – по-важно е да се покрие поне минимален хорариум, който да позволи на преподавателите да направят всеобхватен преглед на учебния материал в осемте години

на училищното образование (от V до XII клас). По принцип информационните технологии се изучава с много повече часове в основните и средните училища, но информатиката сама по себе си е учебен предмет със значително по-сложно учебно съдържание. Поради тази причина до известна степен се очаква като нормално, ако двата учебни предмета са разделени (както е при половината от разгледаните шест университета), те да са с равен или много близък хорариум. Всички университети спазват този принцип.

Като извод от направения преглед се вижда, че от разгледаните учебни планове, само тези в Софийския и Пловдивския университет имат по-съществени пропуски, които би могло да бъдат коригирани. При специалност „Математика и информатика“ на Софийския университет трябваше да се помисли за изтегляне на хоспитирането в трети курс, текущата педагогическа практика със семестър назад, а двата училищни курса с поне един и съответно два семестра назад в учебния план. И при двете специалности в Пловдивския университет училищните курсове е логично да бъдат преместени в трета академична година, а курсът по методика с един триместър назад. Естествено, при подобни предложени промени остава отворен въпросът за това кои други предмети могат да бъдат преместени по-напред като реципрочна размяна. Това е нелек въпрос, защото обикновено засяга интересите на различни катедри в съответните факултети. Във Великотърновския университет би могло да се потърси начин поне един от двата училищни курса да се премести със семестър напред или (защо не) да се експериментира с преместване на курса по МОИИТ в трети курс, първи семестър.

По моя инициатива на заседание на съвет на катедра „Обучение по математика и информатика“ с протокол №1/2020 г. беше инициирана дискусия за промяна на учебния план на специалност „Математика и информатика“ (редовно обучение). Предложените към Факултетния съвет промени бяха следните:

- Учебният предмет „Училищен курс по информационни технологии“ да бъде преместен във втори курс, втори семестър.

- Учебният предмет „Хоспитиране по математика“ да бъде преместен в трети курс, втори семестър.
- Учебният предмет „Хоспитиране по информатика и информационни технологии“ да бъде преместен в трети курс, втори семестър.
- Учебният предмет „Текуща педагогическа практика по математика“ да бъде преместен в четвърти курс, първи семестър.
- Учебният предмет „Текуща педагогическа практика по информатика и информационни технологии“ да бъде преместен в четвърти курс, първи семестър.

Както се вижда, подобна промяна би разрешила и двата набеязани проблема – ще се получи по-плавна разпределение на педагогическите практики (в различни последователни семестри), а училищните курсове ще предложат практиките и курсовете за методика на обучение. Тези промени бяха приети [42] и влязоха в сила за студентите, приети през учебна 2019–2020 година, редовно обучение.

Освен това бяха обсъдени и приети подробни правила за провеждането на хоспитиране [39], текуща педагогическа практика [38] и стажантска практика [37]. Основен дискутиран проблем при хоспитирането беше свързан с оценяването – в крайна сметка то е само наблюдение на уроци и конферирание. Беше прието за оценка по хоспитиране студентите да попълнят специално подготвена анкетна карта и да разработват план на един от наблюдаваните от тях уроци. При текущата педагогическа практика оценяването да се формира в комисия с базовия учител, като оценката се поставя не само на изнасянето на конкретен урок, но и на предварителната подготовка за него. За целта студентите разработват два или три последователни урока в екип.

Учителска правоспособност би могла да се получи в бакалавърска степен и от студенти от други специалности чрез записване на т.нар. модули за учител под формата на факултативно обучение (студентите много често го възприемат като втора специалност, въпреки че това не е съвсем вярно). Във ФМИ на Софийския универ-

ситет има модул за придобиване на професионална квалификация „Учител по информатика и информационни технологии“. Обучението се провежда в последните три семестъра на бакалавърската степен.

Исторически интересът към модула за учител по информатика и ИТ винаги е бил слаб (при математика напротив – там има значим интерес). Основната причина за това е, че при този вид обучение допълнителните учебни предмети и педагогически практики се явяват допълнителен хорариум – не може основна дисциплина от модула за учител да се отчита втори път като избираема при основната специалност на студента, няма как и избираема дисциплина от модула да се брой за избираема за основната специалност. Това натоварва значително студентите и претрупва седмичния им график до максимум. След изменението в Наредбата за държавните изисквания за придобиване на професионална квалификация „учител“ [22] проблемът се изостри значително, защото практиките увеличиха общия си обем и се добави още един предмет („Приобщаващо образование“). По инициатива на катедра ОМИ, учебните планове на факултативните модули за учител бяха осъвременени и приети на Факултетен съвет с протокол № 4/25.03.2019 г. така, че да отговарят на промените в наредбата за квалификация на учител и същевременно да се облекчи максимално натоварването на студентите. По конкретно с оглед на това, че модулът „Учител по информатика и информационни технологии“ е насочен основно към студенти от специалности в направление 4.6 (Информатика, Компютърни науки, Софтуерно инженерство), се прие за допустимо да се пропуснат учебните дисциплини, които са свързани със систематизиране и припомняне на учебното съдържание от училищните курсове, а именно:

- операционни системи и компютърни мрежи в училище;
- дизайн и публикуване в интернет;
- училищен курс по информатика;
- езици и среди за обучение;
- училищен курс по информационни технологии.

За сметка на това се реши да се наблегне повече на методиката, като курсът „Методика на обучение по информатика и информационни технологии“, който се изучава в бакалавърската програма „Математика и информатика“, беше разделен на два:

- Методика на обучението по ИТ
- Методика на обучението по информатика.

В крайна сметка отпадналите учебни предмети компенсираха добавените допълнителни часове по методика, новия курс „Приобщаващо образование“, увеличените часове за педагогически практики и задължението да се запишат два избираеми учебни предмета (по един от два списъка). За факултативна дисциплина беше приета за допустима опцията да се отчете и такава от основната специалност на студента.

Дали промяната в посока на по-малко задължителни и повече избираеми предмети ще доведе до повишен интерес към записването на факултативния модул, предстои да бъде измерено в предстоящите години. При всички положения остава наличен проблемът с прекалено големият хорариум, който практически изпълва почти до крайност максималната седмична учебна заетост, допустима за студентите от бакалавърска степен. Едно потенциално решение на този проблем би било да се увеличи броят на семестрите за модулите за учител от три на четири, но това на този етап не е възможно да бъде променено.

На обратния полюс са курсовете за следдипломна квалификация (СДК), които в последните години се радват на повишен интерес и при тях се отчитат много добри резултати. Традиционно в едногодишния курс „Учител по информатика и информационни технологии“ във ФМИ на Софийския университет се записват между 15 и 20 души в изцяло платена форма на обучение. Има много и най-различни курсисти от всички възрастови групи, но ако трябва да се групират според някаква обща тенденция, тя е основно в две посоки:

- с учителска правоспособност в други професионални направления, които желаят да добият допълнителна квалификация за учител по информатика и ИТ;

- завършили висше образование и за известен период от време са работили в някоя професия извън сферата на образованието, но впоследствие са решили да започнат кариера като учители.

За да може човек да практикува професията „учител“, трябва да е получил професионална квалификация по реда на наредбата [22], но от много години насам поради недостиг на квалифицирани учители много често се налага такива да бъдат назначавани на срочен договор по реда на чл. 8 ал. 1 от [15], която гласи:

Чл. 8 (1) (Предишен текст на чл. 8, ДВ, бр. 26 от 1996 г.) При липса на правоспособни кандидати директорът сключва срочен трудов договор с лица, които не отговарят на изискванията за заемане на съответната длъжност „учител“ или „възпитател“, до явяването на кандидат с необходимото образование и професионална квалификация.

Именно такива хора се явяват често като обучаеми в курсовете за СДК. Поради тази причина може да се приеме, че мнозинството в тези курсове са вече действащи учители, като от тях най-често са именно по информатика или ИТ. Това прави обучението значително по-различно от това в бакалавърската програма „Математика и информатика“, защото, от една страна, то е целенасочено само в една професионална област, а от друга, се работи с по-възрастни хора, които обикновено имат практически опит в училище, включително и в целевото професионално направление. Това е и причината да бъде възможно придобиване на въпросната професионална квалификация в рамките на само една учебна година, но все пак трябва да се отбележи, че обучението е много интензивно и учебни занятия се провеждат дори през ваканциите.

Фактът, че обучаемите в курсовете за СДК почти винаги работят с трудови договори на пълен работен ден носи и своите негативи в чисто организационен план, защото става трудно часовете да бъдат провеждани през седмицата в работно време. Като се добави допълнителният фактор, че много от тях живеят и практикуват професията си в провинцията, се налага дисциплините в СДК „Учител по информатика и информационни технологии“ във ФМИ да

се провеждат основно интензивно в събота и неделя. Това от своя страна изисква те да се наредят задължително една след друга и не е възможно няколко предмета да се водят паралелно, както се получава в различните семестри на бакалавърската програма. В частност има и отражение върху междупредметните връзки с различното разпределение те се изграждат на съвсем различен принцип.

Трябва да се отчете, че обучаемите в курсовете за СДК невини имат силната математическа подготовка, които бакалаврите от специалност „Математика и информатика“ получават от общите за ФМИ предмети в първи и втори курс (не всички от курсовете по СДК са учили линейна алгебра, висша алгебра, диференциално и интегрално смятане и др.). Накрая, но със сигурност не на последно място, студентите от „Математика и информатика“ са учили предмети като „Увод в програмирането“, „Структури от данни в програмирането“ и „Информационни системи (бази от данни)“, преди да започнат да се готвят по специализираните за професионалната квалификация учебни предмети, докато повечето от курсистите в СДК не са. Именно изброеното множество от причини поражда и разлики в съдържателната част на специализираните учебни предмети – курсовете за бакалаври и курсовете за СДК имат свои неизбежни, понякога много съществени разлики в учебна програма, методика на обучение и учебни материали.

Изброените дотук проблеми на курсистите от СДК – средностатистически по-слабата предварителна подготовка, по-натовареното ежедневие (комбиниране на работа на пълен работен ден с учене), по-напредналата възраст (отговорност към семейство) и по-интензивното обучение (всички специализирани курсове в рамките на една година) – първоначално подсказват, че трябва да има очакване студентите от бакалавърските програми да излизат като по-добре подготвени педагогически специалисти. Истината обаче е, че е точно обратното – завършилите СДК не само се реализират почти на 100% като учители (при бакалавърската програма сравнително малък брой от завършилите работят по специалността си, защото повечето предпочитат да започнат в ИТ фирми

на много по-високи заплати), но по мое лично макар и субективно мнение мотивацията за учене по време на СДК курсовете е значително по-голяма, отколкото при студентите бакалаври, а това често рефлектира и в по-добре подготвени като краен резултат учители. Твърдението ми за мотивацията може да се измери например и по обективен показател за „брой отсъствия“ – при бакалаврите се случва да има часове, в които отсъстват дори повече от 50% от групата, докато при СДК много рядко има отсъствия на обучаеми (обикновено отсъстват не повече от един-двама души и най-честата причина е болест).

Причините за този интересен феномен може да се търсят в няколко посоки, като моите хипотези основно са:

- Голяма част от студентите в бакалавърска степен е възможно да са записали специалността, която дава педагогическа правоспособност, без въобще да имат намерение да се реализират като учители – заради натиск от родителите им или други социално-икономически съображения са решили на всяка цена да следват висше образование, но не са били приети в друга, по-желана от тях специалност. Статистически погледнато това твърдение би могло да бъде подкрепено и с баловите на приемните изпити – традиционно в специалност „Математика и информатика“ във ФМИ балът за прием е значително по-нисък от този на „Компютърни науки“, „Софтуерно инженерство“ и „Информатика“.
- Обучаемите в курсовете по СДК все пак имат голям натрупан практически опит в учебна среда, а това е от голямо значение – всички като минимум са задължително завършили бакалаври, а част от тях имат и магистърска степен. Тази рутина, макар и добита в различна научна област, със сигурност им помага.
- Житейският и най-вече професионален практически опит е много вероятно да е един от най-значимите фактори – мнозинството от обучаемите в курсовете за СДК са вече действащи учители, т.е. те изучават специализираните учебни дисциплини по информатика и ИТ, вече имайки призма-

та на реалните учебни ситуации, с които самите те са се сблъскали на практика, през която пречупват добиваните нови знания.

- Възможно е да се окаже, че за професия „учител“ възрастта всъщност да е предимство, а не недостатък. Много от обучаемите в курсовете по СДК са родители на малки или дори вече пораснали деца – не е нелогично да се приеме, че опитът с отглеждането на собствено дете помага силно при обучението на чужди.
- Курсовете за СДК са платени от самите обучаеми, докато бакалавърската програма е силно субсидирана от държавата. Освен това студентите бакалаври получават издръжка от родителите си, а обучаемите в СДК се издържат сами. Смя да се изкажа твърдение, че това също може някак да влияе върху мотивацията за учене, защото хората обикновено са по-взискателни към продукти, за които са вложили повече лични финансови средства, отколкото към онези, които се получават даром. Тоест в случая говоря за търсене на по-високо качество на образованието като продукт.

В последната точка под „качество на образователния продукт“ по никакъв начин не вменявам, че преподавателите имат по-голям ентузиазъм за работа при платените курсове – напротив, личният ми опит показва, че всичките ми колеги (без изключение) подхождат еднакво съвестно и при платеното, и при субсидираното обучение. Качественото обучение обаче не идва едностранно само и единствено от образователната институция – то много силно зависи от вътрешната мотивация за учене на самия обучаем. Докато въпросната мотивация за учениците в началните, основни и средни училища е отговорност на учителя и родителя, защото обучението там е задължително (по Конституция до 16-годишна възраст), то във висшето образование мотивацията за учене е плод на двустранен процес: доброто и интересно преподаване, от една страна, но и вече изграденото желание за реализация на студентите в дадена професионална сфера, от друга.

В крайна сметка се получава така, че в курсовете за СДК самите обучаеми изискват от лектора да им поднася повече знания, докато при студентите от бакалавърската програма е нерядко обратното – чувства се стремеж при част от тях, ако е възможно да се учи по-малко в часовете, защото по този начин ще бъдат изпитвани на по-малък обем учебен материал в края на курса. Именно последното ми подсказва, че такива студенти може би нямат изразено желание да получават знания и умения, които биха им помогнали за реализация като учители, а те вече са решили, че ще работят друго и в университета пребивават само за да получат диплома за висше образование и/или достъп до евтино общежитие. За съжаление, от подобно отношение ерозира мотивацията и на част от останалите студенти бакалаври, които са около тях.

Подобен ефект може да се наблюдава в явен вид при подготовката за провеждане на уроци при педагогическите практики – в определени случаи аз и колегите ми изпитваме трудност да накараме студентите да мислят адекватно по въпроси, свързани с дидактиката, просто защото самите студенти са приели, че щом могат да решават учебните задачи, те вече знаят всичко необходимо и нямат нужда от това да им се обяснява как да се преподава. Подобно пренебрежимо отношение към дидактически и методически въпроси буквално липсва при курсистите от СДК – напротив, при тях изразено се усеща вътрешно убеждение, че „как да се преподава“ е най-важният въпрос за един учител, защото това е най-трудното за усвояване умение, което изисква жив контакт в педагогически ситуации, докато знанията за технологиите могат да бъдат наваксани по всяко време и чрез самообучение.

Да, но от друга страна, както вече беше споменато, голяма част от курсистите в СДК са вече действащи учители с правоспособност, които се преквалифицират в различно професионално направление – значи все пак някога са преминали през бакалавърски курс на обучение за добиване на професионална квалификация „учител“. Получава се затворен кръг от аргументи и контрааргументи, от който не може да се направи адекватна оценка за това коя от двете коренно различни форми на обучение е евентуално по-до-

бра – сравняваните групи обучаеми са прекалено различни, за да бъдат съпоставяни. Това прави прекалено субективно отчитането на кой от изброените фактори е основен и с най-голяма тежест и кой не е, а оттам и става много трудно да се извадят каквито и да е смислени изводи за причините за по-ниската мотивация на студентите в бакалавърската програма. Затова не бих могъл да твърдя еднозначно коя форма на подготовка на учители по информатика и ИТ е по-ефективна.

Тези фактори все пак могат да се вземат предвид от колеги методисти, които тепърва започват да изготвят обучителни курсове за бакалаври или СДК – редно е да отчитат всички изброени характеристики и особености на обучаемите в различните форми на обучение. Вероятно е добра идея преподавателите да вложат повече внимание и ентузиазъм за изтъкване на повече от добрите страни на професия „учител“ при студентите от бакалавърската програма, а негативните по възможност да се спестяват за обучаемите, защото те допълнително ги отблъскват от тъй или иначе не особено атрактивната спрямо алтернативите на пазара на труда професия. Като вече дългогодишен академичен наставник по програма ЕОС във Факултета по математика и информатика на Софийския университет „Св. Климент Охридски“ винаги съм се опитвал да постигам именно това. Убедено смятам, че е редно университетските преподаватели, които подготвят студенти за дадена професия по съответна държавна поръчка, да бъдат до известна степен и пиари на тази професия, особено в случаите, в които обществените нагласи като цяло не са особено положителни. Обратно, при курсистите в курсовете от СДК би следвало да се набляга много повече на техническите детайли около учебното съдържание, защото именно там те изпитват по-големи затруднения.

Не може да не се отбележи, че в курсовете по СДК съществува и един много тежък „разлом“ в нивото на подготовка на обучаемите. Има огромна разлика между кандидат-учители, които не са имали образование в сферата на информатиката, и такива, които са завършили информатика, софтуерно инженерство, компютърни науки, компютърни системи и технологии, и подобни. Докато

първите са обикновено преквалифициращи се учители и съответно нямат никакви затруднения в педагогическите и методическите дисциплини, но изпитват сериозни трудности в информатичните дисциплини и най-вече с програмирането, при вторите е точно обратното. За всеки преподавател, който има група от студенти, разделени в две крайности – такива с добра предварителна подготовка и такива с никаква – провеждането на адекватно обучение, което да отговори на потребностите на всички, е сериозно предизвикателство. Засага обучаемите, идващи от научни направления, свързани с информатика, са малко на брой (както е обикновено – 3–4) и може да се прилага индивидуален подход. Ако групата им стане по-голяма, тогава мисля, че проблемът ще се изостри много. Затова смятам, че ако в бъдеще количеството на подобни курсисти се увеличи значително, не е лошо да се помисли за разделяне на СДК курса на две части – един за завършили в съответната научна област и един за всички останали.

През учебната 2020 година курсовете СДК във ФМИ получиха значително развитие заради задействането на новата Национална програма „Мотивирани учители“ [24], в изготвянето на която активно участие взе доц. Ангел Ангелов от катедра ОМИ. Идеята на програмата е да се създадат СДК курсове за учителска правоспособност, които са субсидирани от държавата (без такса за обучаемите). В частност от катедра ОМИ създадохме два такива – „Учител по информатика“ и „Учител по информационни технологии“. Докато към първия интересът беше малък, вторият СДК курс се запълни с много курсисти. Вероятно тази разлика в интереса е породена от различния обхват на учебния предмет в училищата – докато от учители по информатика има интерес основно само в математически гимназии, информационни технологии е масов предмет навсякъде. Все още е рано за генерализация и по-детайлен анализ за това дали има качествена разлика между платеното и субсидираното СДК обучение. Моето текущо наблюдение от личен опит с водене на по три учебни предмета на всеки СДК курс е, че на този етап по-скоро няма значима разлика във входното ниво, мотивацията за учене и крайните оценки на курсистите.

Накрая бих искал да дискутирам и един въпрос, който е видим от всички мои колеги, но, за съжаление, рядко се адресира – входното ниво на студентите в бакалавърска степен е много различно между студенти от математически гимназии или специализирани училища и други, завършили езикови гимназии, обикновени средни училища или техникуми. Наблюдавали сме големи разлики дори при предмети като математика, които се изучават масово и задължително във всички училища (в Югозападния университет „Неофит Рилски“ засегнах тази тема като един от изводите в своята дисертация [29]). Наблюдавал съм същия ефект (при това доста по-силно) и по информатика: в Техническия университет – София съм бил свидетел години наред как хора, завършили ТУЕС, СМГ, НПМГ и подобни училища, скучаят в уводните курсове по програмиране, докато нивото на техни колеги се изравни с тяхното. Струва ми се, че при идващите випуски, в които информатиката вече няма да е задължителна за масовите училища, същият проблем ще се задълбочи многократно. Като цяло бих се застъпил и заложил на три възможни различни подхода за решение на проблема:

- Обособяване на различни специалности за студенти от различни видове училища.
- Пожелателен изравнителен курс преди започване на първи курс за студентите, които не са изучавали информатика в гимназиите.
- Допълнителен приеман изпит по програмиране за компютърните специалности (Компютърни науки, Софтуерно инженерство и Информатика), който да е с по-висок коефициент при балообразуването спрямо традиционния по математика.

Без да съм задълбавал юридически по въпроса, имам чувството, че първото предложение е по-скоро непрагматично, а и до някаква степен дискриминационно – по-скоро ще създаде повече проблеми, отколкото ще има полза от този, който евентуално ще разреши. Подобен ефект на елитарност и разлика в специалностите има и в момента – кандидатите с най-висок бал, които са обикновено от математически гимназии, се фокусират в топ спе-

циалността на факултета. Но той се е случил по естествен път, а не с насилствена регулация. Затова по-скоро смятам това предложение за вредно.

Второто е по-практично, но повлича със себе си допълнителни разходи за факултета и създава трудност около заплащането и/или отчитането на часовете към норматива на преподавателите. Основна спирачка е и борбата за надмощие между катедрите – не е еднозначно ясно кой ще води курса и какво точно ще включва той. Изобщо такова решение трябва да търси доста широк консенсус. В катедра ОМИ преди години бяхме предизвикали дискусия в рамките на факултета, но не успяхме да получим необходимото одобрение. Още по-отдавна е имало подобен пилотен експеримент, но той се е провалил поради конюнктурни причини.

Третото решение би било добро не от гледна точка на това, че ще ограничи тези, които не са учили програмиране, от вход във факултета, а че така ще накара повече ученици да се подготвят с кандидатстудентски курсове, включващи информатика. Така за много новопостъпващи студенти ще бъде насърчено въпросното предварително изравняване на нивото с допълнително учене преди първи курс, но разликата с предишната точка е, че в този случай то ще бъде постигнато на пазарен принцип с платено обучение извън университета. Мисля си, че подобна практика би била добра, въпреки че ще търпи известни критики от гледна точка на социално неравенство и подобни. Освен това не е решение, което ще разреши проблема изцяло.

ИЗПОЛЗВАНА ЛИТЕРАТУРА

- [1] Ангелов, А., Гъров, К., Гавраилов, О. Информатика. Част I. София: Печатна база на МНП, 1986.
- [2] БАСКОМ Барометър 2019. Годишен доклад за състоянието на софтуерния сектор в България, декември 2019.
- [3] Бърнев, П., Керпеджиев, Ст. Основни понятия в информатиката. София: „Петър Берон“, 1988.
- [4] Ганчев, И, Грозев, С. За два фундаментални подхода към развитието на научно познание и тяхната употреба в дидактиката по математика. Шеста средиземноморска конференция за обучение по математика 2009 г.
- [5] Ганчев, И., Нинова, Ю., Никова, В. Методика на обучението по математика (обща част), Благоевград, 2002.
- [6] Георгиева-Трифенова, Ц. Въведение в Kodu. София: Фондация „Буквите“, 2018. ISBN 978-619-154-305-2.
- [7] Гоцева, Д., Ганчева, В., Петров, Ф. Бази данни. Учебник по задължителен учебен курс на бакалавърска програма „Компютърни системи и технологии“ в Технически университет – София. София: Издателство към Технически университет – София, 2012. ISBN 978-954-438-809-6.
- [8] Гъров, К. За новото учебно съдържание по информационни технологии в прогимназиален етап на българското училище. Национална конференция „Образованието и информационното общество“, 2006.
- [9] Дзиев, В. Методика за обучение по модул „Бази от данни“. Ди-

сертационен труд по професионално направление 1.3 „Педагогика на обучението по...“, научна специалност „Методика на обучението по информатика и ИТ“, Софийски университет „Св. Климент Охридски“, 2018.

- [10] Държавни образователни изисквания към учебно съдържание, културно-образователна област: Математика, информатика и информационни технологии. Приложение № 3 към чл. 4, т. 3 чл. 4, т. 3 (изм. ДВ, бр. 46 от 2004 г., в сила от 1.07.2004 г., доп., бр. 58 от 2006 г., в сила за учениците, които през учебната 2006–2007 г. постъпват в I и в V клас).
- [11] Жечева, А., Петров, Ф. Онлайн сборник със задачи със софизми по математика, информатика и информационни технологии. – В: списание „Математика, компютърни науки и образование“, том 1, брой 1, 2018. ISSN: 2603-4670, EISSN: 2603-4735.
- [12] Заимов, П. Методика на обучението в прогимназиите проведено през изследване и творчество, Пловдив, 1937.
- [13] ЗАКОН за предучилищното и училищното образование. Обн., ДВ, бр. 79 от 13.10.2015 г., в сила от 1.08.2016 г., изм. и доп., бр. 98 от 9.12.2016 г., в сила от 1.01.2017 г., изм., бр. 105 от 30.12.2016 г., в сила от 1.01.2017 г., бр. 58 от 18.07.2017 г., в сила от 18.07.2017 г.
- [14] Замфиров, М. Исторически предпоставки за внедряването и развиването на обучението по информатика в българските училища. Трети конгрес на физическите науки, секция „История на физиката“, 2016.
- [15] Инструкция № 2 от 29 юли 1994 г. за изискванията за заемане на длъжността „учител“ или „възпитател“ съобразно придобитото образование, професионална квалификация и правоспособност, издадена от министър на науката и образованието. Обн. ДВ, бр. 69 от 26 август 1994 г., изм. ДВ, бр. 83 от 11 октомври 1994 г., изм. ДВ, бр.103 от 24 ноември 1995 г., изм. ДВ, бр.26 от 26 март 1996 г., изм. ДВ, бр.81 от 17 септември 1997 г., изм. ДВ, бр.81 от 12 септември 2003 г.
- [16] Карабельова, Г., Петров, Ф. Хардуерна лаборатория и музей в класната стая. – В: сборник „Математика и математическо образование“. СМБ, 2020. ISSN 1313-3330.

- [17] Кендеров, П., Андреев А. Академик Благовест Сендов (1932–2020) – В: сборник „Математика и математическо образование“, 2020. ISSN 1313-3330.
- [18] Киселев, Г. М., Бочкова, Р. В. Информационные технологии в педагогическом образовании. Учебник 2-е издание, переработанное и дополненное. Рекомендовано органом по сертификации образовательной деятельности „Центр качества профессионального образования“ в качестве учебника для студентов вузов, обучающихся по направлению подготовки „Педагогическое образование“ (квалификация — бакалавр). Москва: Издательско-торговая корпорация „Дашков и К°“, 2014.
- [19] МОН. Национален институт по образование. Стратегия и национална програма за развитието на информационното общество в Република България, 1998. <http://planipolis.iiep.unesco.org/upload/Bulgaria/Bulgaria-ICT-Policy.pdf> (последно посетен на 28.04.2021).
- [20] Москалева, Ю. П., Сейдаметова, З. С. Централизованные и распределенные системы управления версиями как учебная платформа. Информационно-компьютерные технологии в экономике, образовании и социальной сфере, (3), 112–116, 2017.
- [21] Наредба № 12 от 01.09.2016 г. за статута и професионалното развитие на учителите, директорите и другите педагогически специалисти. Обн. ДВ, бр. 75 от 27.09.2016 г., в сила от 27.09.2016 г., Издадена от министъра на образованието.
- [22] Наредба за държавните изисквания за придобиване на професионална квалификация „учител“. Обн. ДВ, бр. 89 от 11.11.2016 г., в сила от учебната 2017–2018 година; изм. и доп., бр. 105 от 18.12.2018 г., в сила от 18.12.2018 г.
- [23] Национален статистически институт. Средна брутна месечна заплата на наетите лица по трудово и служебно правоотношение през 2019 г.
- [24] Национална програма „Мотивирани учители“, одобрена с Решение № 285 от 30 април 2020 г. на Министерския съвет.
- [25] Национална програма „ИКТ в училище“ (2008–2013).
- [26] Национална програма „Информационни и комуникационни технологии (ИКТ) в системата на предучилищното и училищното образование“ (2019).
- [27] Национална стратегия за въвеждане на ИКТ в българските училища 2004 г.
- [28] Националната стратегия за въвеждане на ИКТ в българските училища (2005–2007).
- [29] Петров, Ф. Организационен модел за приложение на интерактивни методи в обучението по линейна алгебра. Дисертация за придобиване на научно-образователна степен „доктор“ по направление 1.3. Благоевград: Университетско издателство „Неофит Рилски“, 2014.
- [30] Петров, Ф. Технологии за въвеждане и изобразяване на математически формули в уеб 2.0 приложения. – В: сборник „Образованието в информационното общество“. Пловдив: Асоциация „Развитие на информационното общество“ и „Институт по математика и информатика“ към БАН, 200–209, 2011, ISSN 1314-0752.
- [31] Петров, Ф. Уеб 2.0 обучение по математика. – В: сборник „Синергетика и рефлексия в обучението по математика“, Пловдив: Университетско издателство „Паисий Хилендарски“, 445–452, 2010. ISBN 978-954-423-621-2.
- [32] Петров, Ф., Андреев, Д. Ползи от научноизследователската дейност на студенти и докторанти по информатика и информационни технологии извън университета. – В: Сборник с доклади от научна конференция на младите изследователи 13 май 2016 г., Велико Търново: Фабер, 7–13, 2016. ISBN 978-619-00-0476-9.
- [33] Петров, Ф., Георгиева-Трифенова, Ц. Нерелационни бази от данни – практическо ръководство. София: Университетско издателство „Св. Климент Охридски“, 2019, ISBN 978-954-07-4821-4.
- [34] Петров, Ф., Джолев, Д. Въвеждане на елементи от паралелно и мрежово програмиране в курс по обектно-ориентирано програмиране. – В: сборник „Математика и математическо образование“ СМБ. София: Издателство на БАН, 255–261, 2016, ISSN 1313-3330.

- [35] Петров, Ф., Дурева-Тупарова, Д. Практически пример на технологични средства за реализиране на уеб 2.0 обучение по математика. – В: Математика и информатика – научно-методическо списание, година LVI, кн. 3, 258–266, 2013. ISSN 1310-2230.
- [36] Постановление № 27 от 1 февруари 2021 г. за изменение и допълнение на Наредбата за държавните изисквания за придобиване на професионална квалификация „учител“, приета с Постановление № 289 на Министерския съвет от 2016 г.
- [37] Правила за провеждане на стажантска практика. Катедра ОМИ, Софийски университет „Св. Климент Охридски“. <https://omi.fmi.uni-sofia.bg/practices/internships/>
- [38] Правила за провеждане на текуща педагогическа практика Катедра ОМИ, Софийски университет „Св. Климент Охридски“. <https://omi.fmi.uni-sofia.bg/practices/tpp/>
- [39] Правила за провеждане на хоспитиране. Катедра ОМИ, Софийски университет „Св. Климент Охридски“. <https://omi.fmi.uni-sofia.bg/practices/hospitirane/>
- [40] Проект № BG05M2OP001-2.012-0001 „Образование за утрешния ден“. Министерство на образованието и науката (конкретен бенефициент) и ИА на Оперативна програма „Наука и образование за интелигентен растеж“ (ОП НОИР) 2014–2020 г.
- [41] Проект на Наредба за изменение и допълнение на Наредба № 5 от 30 ноември 2015 г. за общообразователната подготовка. <http://strategy.bg/PublicConsultations/View.aspx?lang=bg-BG&Id=5278>
- [42] Протокол № 7/27-29.04.2020 г. на ФС на ФМИ, Софийски университет „Св. Климент Охридски“.
- [43] Сейдаметов, С. В. Распределенные системы контроля версий: проблемы начинающих разработчиков. Информационно-компьютерные технологии в экономике, образовании и социальной сфере, (3), 39–44, 2019 г.
- [44] Стефанова, Е., Николова, М., Митева, Д., Петров, Ф., Николова, Н. Информационни технологии за 8. клас със CD. София: Просвета, 2017. ISBN 978-954-01-3429-1.
- [45] Стефанова, Е., Николова, М., Митева, Д., Петров, Ф., Николова, Н. Книга за учителя по информационни технологии за 8. Клас. София: Просвета, 2017. ISBN 978-954-01-3497-0.
- [46] Стратегия за ефективно прилагане на информационни и комуникационни технологии в образованието и науката на република България (2014–2020).
- [47] Табакова-Комсалова, В., Глушкова, Т. Учебно съдържание на задачите в обучението по компютърно моделиране за III клас. – В: Научна конференция „Иновационни софтуерни инструменти и технологии с приложения в научни изследвания по математика, информатика и педагогика на обучението“, 23–24 ноември, Пампорово, 2017 г.
- [48] Учебен план на специалност „Информационни технологии, информатика и математика“. Протокол на Академичния съвет № 7/29.03.2019 г. Факултет по математика и информатика. Шуменски университет „Епископ Константин Преславски“.
- [49] Учебен план на специалност „Информационни технологии, математика и образователен мениджмънт.“, Протокол на Факултетния съвет № 17/19.04.2017 г., Факултет по математика и информатика. Пловдивски университет „Паисий Хилендарски“.
- [50] Учебен план на специалност „Математика и информатика“. Протокол на Факултетния съвет № 7/24.06.2019 г. Факултет по математика и информатика, Софийски Университет „Св. Климент Охридски“.
- [51] Учебен план на специалност „Математика, информатика и информационни технологии“. Протокол на Факултетния съвет № 17/19.04.2017 г. Факултет по математика и информатика. Пловдивски университет „Паисий Хилендарски“.
- [52] Учебен план на специалност „Педагогика на обучението по математика, информатика и информационни технологии. Протокол на Факултетния съвет № 9/28.06.2017 г. Природо-математически факултет. Югозападен университет „Неофит Рилски“.
- [53] Учебен план на специалност „Педагогика на обучението по математика и информатика“. Протокол на Факултетния съвет № 6/19.6.2017 г. Факултет по математика и информатика, Великотърновски университет „Св. св. Кирил и Методий“.

- [54] Учебен план на специалност „Педагогика на обучението по математика и информатика“. Протокол на Академичен съвет от 23.4.2019 г. Факултет „Природни науки и образование“, Русенски университет „Ангел Кънчев“.
- [55] Учебна програма по информационни технологии (профилирана подготовка). Заповед № РД09-4022/21.12.2018 г. на министъра на образованието и науката за утвърждаване на учебни програми за осигуряване на профилираното обучение на учениците, които през учебната 2020–2021 година постъпват в XI и през учебната 2021–2022 година постъпват в XII клас, съгласно приложенията, изменена със Заповед № РД09-515/24.01.2019 г., Заповед № РД 09-594/04.02.2019 г. и със Заповед № РД09-1283/02.06.2020 г.
- [56] Учебна програма по компютърно моделиране и информационни технологии за V клас (общообразователна подготовка). Приложение към заповед № РД09-2819/09.10.2020 г.
- [57] Учебна програма по компютърно моделиране и информационни технологии за VI клас (общообразователна подготовка). Приложение към заповед № РД09-2819/09.10.2020 г.
- [58] Учебна програма по компютърно моделиране и информационни технологии за VII клас (общообразователна подготовка). Приложение към заповед № РД09-2819/09.10.2020 г.
- [59] Учебни програми за III клас, утвърдени със Заповед № РД09-1093/25.01.2017 г. на министъра на образованието и науката. Учебна програма по компютърно моделиране за III клас (общообразователна подготовка).
- [60] Учебни програми за IV клас, утвърдени със Заповед № РД09-5778 от 22.11.2017 г. на министъра на образованието и науката. Учебна програма по компютърно моделиране за IV клас (общообразователна подготовка)
- [61] Учебни програми за V клас в сила от учебната 2016–2017 година, утвърдени със Заповед № РД09-1857 от 17.12.2015 г. Учебна програма по информационни технологии за V клас (общообразователна подготовка).
- [62] Учебни програми за VI клас в сила от учебната 2017/2018 година, утвърдени със Заповед № РД09-300 от 17.03.2016 г. Учебна програма по информационни технологии за VI клас (общообразователна подготовка).
- [63] Учебни програми за VII клас в сила от учебната 2018–2019 година, утвърдени със Заповед № РД09-1093/25.01.2017 г. Учебна програма по информационни технологии за VII клас (общообразователна подготовка).
- [64] Учебни програми за VIII клас в сила от учебната 2017–2018 година, утвърдени със Заповед № РД09-301 от 17.03.2016 г. Учебна програма по информатика за VIII клас (общообразователна подготовка).
- [65] Учебни програми за X клас, утвърдени със Заповед РД09-30/11.01.2018 г. на министъра на образованието и науката. Учебна програма по информационни технологии за X клас (общообразователна подготовка).
- [66] Учебни програми за профилирана подготовка. Заповед № РД09-4022/21.12.2018 г. на министъра на образованието и науката за утвърждаване на учебни програми за осигуряване на профилираното обучение на учениците, които през учебната 2020–2021 година постъпват в XI и през учебната 2021–2022 година постъпват в XII клас, съгласно приложенията, изменена със Заповед № РД09-515/24.01.2019 г. и със Заповед № РД 09-594/04.02.2019 г. Учебна програма по информатика (профилирана подготовка).
- [67] Чантов, В. Компетенции и компетентност (между знанието и можещото). – В: конференция: „Нови информационни технологии в образователния процес“, Охрид, 2012.
- [68] Anand, V. R., Dinakaran, M. Popular agile methods in software development: review and analysis. *International Journal of Applied Engineering Research* 11.5, 3433–3437, 2016.
- [69] Dreyfus, Ph. *L'informatique. Gestion*, Vol. 5, 240–241, 1962.
- [70] DuBois, P. *MySQL Cookbook*, O'Reilly, October 2002.
- [71] Gantchev, I., Grozdev, S. On Two Fundamental Approaches to the Development of Scientific Knowledge and Their Implementation in Didactics of Mathematics. *Proceedings of the 6th Mediterranean Conference on Mathematics Education*, Plovdiv, 2009.

- [72] Gülbahar, Y., Kalelioğlu, F. The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective. – In: Journal Informatics in Education – An International Journal, 33–50, 2014.
- [73] Ivanov, S. Using gamification methods in information technology education, EDULEARN19 Proceedings, 2712–2718, 2019. ISBN 978-84-09-12031-4/ISSN 2340-1117.
- [74] Kastl, P., Kiesmüller, U., Romeike, R. Starting out with projects: Experiences with agile software development in high schools. – In: Proceedings of the 11th workshop in primary and secondary computing education, 2016.
- [75] Kessler, C. W. Teaching parallel programming early. – In: Proceedings of Workshop on Developing Computer Science Education – How Can It Be Done? Linköping university, Linköping, Sweden, March 10, 2006.
- [76] Maraffi, S., Sacerdoti, F., Paris, E. Learning on Gaming: A New Digital Game Based Learning Approach to Improve Education Outcomes. US-China Education Review A, 7, 2017. doi:10.17265/2161-623X/2017.09.003
- [77] Missiroli, M., Russo, D., Ciancarini, P. Learning agile software development in high school: an investigation. – In: Proceedings of the 38th International Conference on Software Engineering Companion, 2016.
- [78] Petrov, Ph., Djolev, D. Combining the procedural and the set-based approaches in the teaching of SQL SELECT statements in the introductory databases course. – In: Proceedings Seventh International Scientific Conference Computer Science' 2015, 249–254, ISBN 978-619-167-177-9.
- [79] Petrov, Ph., Gyudzhenov, I., Tuparova, D. Adapting interactive methods in the teaching of Linear Algebra – results from pilot studies. – In: Procedia-Social and Behavioural Sciences Journal, 142–146, 2014. ISSN: 1877-0428.
- [80] Ramakrishnan, R., Gehrke, J. Database management systems (second edition), 2002.
- [81] Renauld, K., Biljon, J. Teaching SQL – Which Pedagogical Horse for this Course? Key Technologies for Data Management, 3112, 244–256, 2004.
- [82] Romeike, R., Göttel, T. Agile projects in high school computing education: emphasizing a learners' perspective. – In: Proceedings of the 7th Workshop in Primary and Secondary Computing Education, 2012.
- [83] Schwaber, K., Beedle, M. Agile software development with Scrum. Upper Saddle River: Prentice Hall; Feb 2002.
- [84] Shabalina, O., Malliarakis, C., Tomos, F., Mozelius, P. Game-Based Learning for Learning to Program: From Learning Through Play to Learning Through Game Development. 11. Graz, Austria: European Conference on Games Based Learning, 2017.
- [85] Sung, K. Computer Games and Traditional CS Courses, 52, 2009. doi:10.1145/1610252.1610273
- [86] Tuparova, D. & Nikolova, E., Tuparova, E. Integrated game-based learning in Informatics secondary course -is there a difference between girls' and boys' achievements?, 2020.
- [87] Walker, E., Connelly, J., Musicant, D. Elegit: Git Learning Tool for Students. – In: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education 642–642, 2017.
- [88] Wolz, U., Barnes, T., Parberry I. Digital gaming as a vehicle for learning. ACM SIGCSE Bulletin, 38, 394–395, 2006. doi:10.1145/1121341.1121463

Филип Петров
ПРЕДИЗВИКАТЕЛСТВАТА ПРЕД ОБУЧЕНИЕТО ПО ИНФОРМАТИКА
В БЪЛГАРСКИТЕ СРЕДНИ УЧИЛИЩА

Българска
Първо издание

Научен редактор
доц. д-р Надежда Ангелова Кучмова
Рецензенти
доц. д-р Ангел Илиев Ангелов
доц. д-р Надежда Ангелова Кучмова

Редактор
Татяна Димитрова
Художник на корицата
Павел Христов

Предпечат
Иво Ников

Формат 60x84/16
Печ. коли 23

Университетско издателство „Св. Климент Охридски“
www.unipress.bg

