З.С. Сейдаметова, Л.М. Меджитова
Кримський інженерно-педагогічний університет

Моделі навчання основ програмування на молодших курсах комп'ютерних спеціальностей університетів

В світі для позначення «будь-якої цілеспрямованої діяльності з використанням комп'ютера [1, 9], [2, 14] використовується термін «комп'ютінг» (computing). Швидкість змін, які відбуваються в комп'ютінгу, ставить перед університетами, кафедрами і викладачами завдання постійного оновлення навчальних курсів, підходів до навчання, зміщуючи акценти від застарілих технологій до таких моделей навчального процесу, на основі яких можна розвивати у студента здібності самостійного надбання знань і навичок. Навчання студентів в сучасних умовах передбачає такі процедури, що стимулює їх до розвитку здібностей самонавчання впродовж всієї професійної кар'єри (continued study throughout a career).

У монографії [2, 40–47] представлений базисний корпус знань BOK2001, який відображений в CS 2001 і становить 132 дискретних частини, з яких 64 є обов'язковими. Мінімальна кількість лекційного часу, виділеного на вивчення обов'язкових частин, складає 280 годин. BOK2001 впорядкований за 14 базисними тематиками: Дискретні структури (DS); Основи програмування (PF); Алгоритми і складність (AL); Архітектура і організація (AR); Операційні системи (OS); Мережевий комп'ютінг (NC); Мови програмування (PL); Робота людини з комп'ютером (HC); Графіка і візуальний комп'ютінг (GV); Інтелектуальні системи (IS); Інформаційний менеджмент (IM); Соціальні і професійні питання (SP); Програмний інжиніринг (SE); Обчислювальні науки і обчислювальні методи (CN). Цей набір базисних дисциплін може бути по-різному структурований для формування тієї або іншої педагогічної моделі навчання основ професії.

Мета, що переслідувалась при підготовці статті: аналіз різних моделей і підходів навчання основ програмування студентів молодших курсів комп'ютерних спеціальностей; аналіз і вивчення базисного корпусу знань, відповідного цим моделям і підходам; аналіз педагогічних концепцій, що використовуються при навчанні студентів молодших курсів комп'ютерних спеціальностей основам програмування.

Моделі навчання міжнародних освітніх стандартах CS 2001 [3], CS 2008 [4] визначені цілі навчання студентів–першокурсників, вивчені сильні і слабкі сторони традиційного підходу «спочатку програмування», тобто такого підходу, при якому вивчення предметної галузі комп'ютінга починається з навчання студентів якій-небудь мові програмування. Також в CS 2001 запропоновані альтернативні підходи навчання і проаналізована їх відповідність цілям освітнього процесу. Розроблено кілька послідовностей ввідних курсів для вирішення проблеми вирівнювання рівня студентів, які поступили на комп'ютерні спеціальності. У результаті в [3] був представлений короткий перелік альтернатив для першого року навчання на комп'ютерних спеціальностях. В цілому в міжнародних освітніх стандартах CS 2001 [3], CS 2008 [4] представлені шість підходів до ввідних курсів з комп'ютерних наук. Відмітимо, що кожний із запропонованих варіантів реалізації апробований багаторічним педагогічним досвідом одного або кількох університетів США. Також відзначимо, що в Україні при навчанні студентів комп'ютерних спеціальностей основ програмування в основному використовується підхід, який базується на навчанні програмування як процесу обчислення у вигляді інструкцій. Ознайомлення освітньої громадськості України з ширшим колом альтернативних підходів сприятиме формуванню загальніших і адекватніших національних моделей навчання.

Для ввідних курсів в освітніх стандартах CS 2001, CS 2008 запропоновано шість реалізацій (підходів): імперативна, об'єктна, функціональна, широка, алгоритмічна і апаратна. Така структура, на думку авторів CS 2001, дає велику гнучкість при складанні навчальних планів.

«Підхід до навчання» визначається за педагогічними аспектами, що висуваються на перший план: «перш за все програмні інструкції», або «перш за все – об'єкти», «перш за все – алгоритми» і т.п.

Перший підхід – імперативний (the imperative-first approach) – базується на програмуванні в традиційному імперативному стилі, оскільки це широко використовується і є інтегральною частиною об'єктно-орієнтованого програмування. Імперативне програмування – це парадигма програмування, яка описує процес обчислення у вигляді інструкцій, за якими змінюється стан програми. В цьому випадку пропонуються два варіанти реалізації: (1) трисеместровий з набором дисциплін (з 1-го по 3-ій семестри) $CS101_I$ «Основи програмування»→ $CS102_I$→ «Об'єктно-орієнтована парадигма»→ $CS103_I$ «Структури даних і алгоритми»; (2) двосеместровий (1-й і 2-й семестри) з дисциплінами $CS111_I$ «Введення в програмування»→ $CS112_I$ «Абстракція даних». В обох варіантах реалізації як мові програмування віддається перевага одній з об'єктно-орієнтованих мов програмування, приділяючи при цьому переважну увагу імперативним аспектам мови (вирази, структури управління, процедури і функції і інші центральні елементи традиційної процедурної моделі), а не методиці об'єктно-орієнтованого проектування. Перший варіант імперативного підходу має на увазі більший тематичний обхват, ніж другий. На рис. 1 подано зведення моделей і підходів до навчання студентів молодших курсів основ програмування. Імперативному підходу, описаному вище, відповідає перша колонка.

Другий підхід – об'єктний (the object-first model) (друга колонка на рис. 1). Центральним моментом цього підходу, як і першого, є програмування. Об'єктна реалізація включає принципи об'єктно-орієнтованого програмування і проектування з перших днів навчання студента. Запропоновано два варіанти реалізації: (1) трисеместровий – з набором дисциплін (з 1-го по 3-ій семестри) $CS101_O$ «Введення в об'єктно-орієнтоване програмування»→ $CS102_O$ «Об'єкти і абстракція даних»→ $CS103_O$ «Алгоритми і структури даних», (2) двосеместровий (1-й і 2-й семестри) з дисциплини $CS111_O$ «Об'єктно-орієнтоване програмування»→ $CS112_O$ «Об'єктно-орієнтоване проектування і методологія». Перші дисципліни обох варіантів починаються з понять об'єктів і спадкоємства, рано занурюючи студентів в об'єктно-орієнтовану філософію, після чого вводяться більш традиційні структури управління в контексті загальних понять об'єктно-орієнтованого проектування. У подальших дисциплінах вивчаються алгоритми, фундаментальні структури даних, деякі питання програмної інженерії.

Третій підхід – функціональний (the functional-first style) базується на функціональній парадигмі програмування (третя колонка на рис. 1). Функціональне програмування – парадигма програмування, в якій процес обчислення трактується як обчислення значень функцій в математичному розумінні. Функціональне програмування є частиною «декларативного програмування» (стиль програмування, за яким програмах описується спосіб розв'язування поставленої задачі, а не пропонуються кроки для отримання результату). Цей підхід був апробований в Массачусетському технологічному інституті (MIT) в 1980-х роках. Особливість цього підходу полягає у використанні простої функціональної мови на першому курсі навчання. Припускається тільки один двохсеместровий варіант реалізації: $CS111_F$ «Введення у функціональне програмування»→ $CS112_F$ «Об'єкти і алгоритми». В цій реалізації функціонального підходу мається на увазі, що за послідовністю $CS111_F$→$CS112_F$ слідує дисципліна, що охоплює об'єктно-орієнтоване програмування і проектування. Зазвичай при навчанні дисциплін функціональної моделі ввідного циклу використовується проста функціональна мова. Прихильники функціонального підходу вважають, що використання такої мови виправдане, оскільки вона рідко використовується в промисловій розробці програмних продуктів, дозволяє швидко вирівнювати відмінності доуніверситетського досвіду програмування у студентів. Функціональні мови мають мінімальний синтаксис, що дозволяє у процесі навчання приділяти увагу фундаментальним питанням програмування. При цьому з'являються можливості деякі складні для першокурсників ідеї (рекурсія, зв'язані структури даних, функції як перший клас об'єктів даних) природним чином включати в програми дисциплін педагогічної реалізації функціонального підходу. Вважається, що функціональний підхід сприяє розвитку абстрактного мислення у студентів.

Четвертий підхід – широкий (the breadth-first approach) (четверта колонка на рис. 1). Широкий підхід, крім орієнтації на програмування, припускає формування у студентів цілісного погляду на комп'ютінг. Пропонуються два варіанти реалізації цього підходу: (1) односеместровий курс $CS100_B$ «Попередній розгляд Computer Science», який випереджає традиційну програмістський набір; (2) трисеместровий набір $CS101_B$ «Введення в Computer Science»→ $CS102_B$ «Алгоритми і методи програмування»→ $CS103_B$ «Принципи об'єктно-орієнтованого проектування». Перший варіант дуже складний в реалізації, оскільки не так просто за один семестр охопити всю різноманітність тематик комп'ютінгу. Дисципліна $CS100_B$ «Попередній розгляд Computer Science», при вивченні якої реалізується цей варіант, включає широкий діапазон тем дискретної математики, введення в мови програмування, розв'язування алгоритмічних задач, аналізу алгоритмічної складності, основних концепцій апаратного забезпечення, операційних систем, мереж, графіки і введення в соціальний контекст комп'ютінга.

П'ятий підхід – алгоритмічний (п'ята колонка на рис. 1). В алгоритмічному підході ввідного рівня навчання представляються основні концепції комп'ютінга не на реальній мові програмування, а за допомогою псевдокоду. Студенти освоюють широкий діапазон типів даних і структури управління не на комп'ютері, а на папері і в уяві. Вони не вивчають специфічні синтаксичні конструкції, властиві якій-небудь мові програмування. Вважається, що завдяки алгоритмічному підходу традиційне програмування освоюється студентами значно швидше, ніж при інших підходах. При такому підході з'являється можливість надалі за рахунок ознайомлення студентів з широким спектром структур даних та структур управління більш глибоко вивчати практичні питання ефективного програмування, що сприяє поліпшенню навичок налагодження програм. Зазвичай для цього підходу використовується двосеместрова реалізація, що складається з дисциплін $CS111_A$ «Введення в алгоритми і застосування»→ $CS112_A$ «Методологія програмування». Перша дисципліна охоплює питання, пов'язані з алгоритмами, їх застосуванням, основами об'єктно-орієнтованого програмування. В другій дисципліні піднімаються питання методології програмування.

Шостий підхід – апаратний (the hardware-first approach) (шоста колонка на рис. 1). Апаратний підхід є останнім із запропонованих у ввідному рівні навчання. У цьому підході звертається увага на основи інформатики – від машинного рівня до абстрактніших концепцій програмування. Модель цього підходу реалізується двосеместровим набором дисциплін $CS111_H$ «Введення в архітектуру комп'ютера»→ $CS112_H$ «Методи об'єктно-орієнтованого програмування». При вивченні першої дисципліні розглядається будова комп'ютера, а вивчення другої з використовуванням відомостей з першої сприяє розвитку навичок програмування і введенню в методи об'єктно-орієнтованого програмування.

Безсумнівно, що окрім цих підходів можна запропонувати безліч інших моделей, які університети могли б використовувати як педагогічні підходи. Разом з тим, яку б модель університет

не вибрав, вона повинна охоплювати всі базисні тематики знань BOK2001 [2, 31–47], [3].

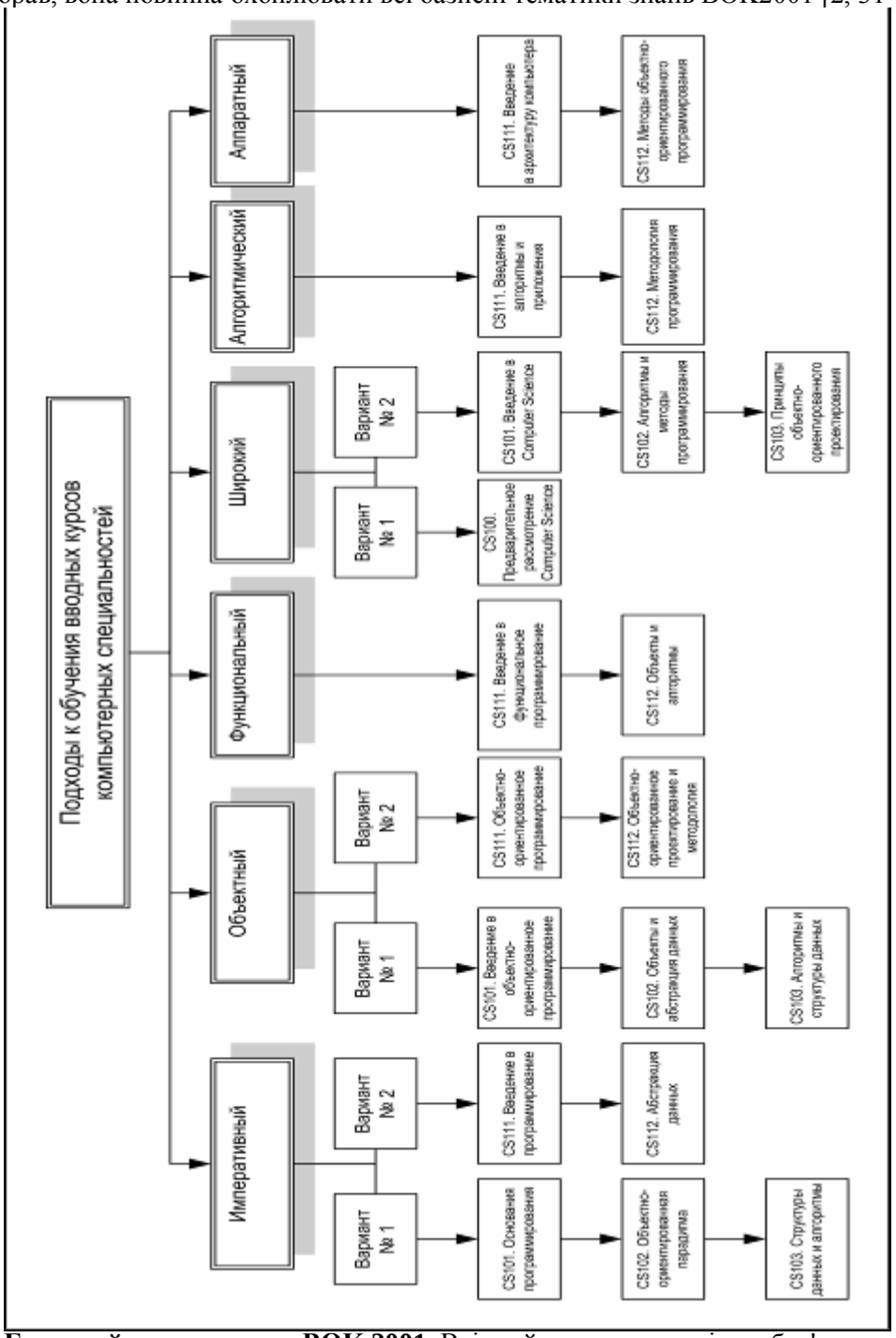

Рис. 1. Зведення підходів до навчання ввідних курсів програмування

Базисний корпус знань BOK 2001. Ввідний цикл дисциплін в обов'язковому порядку повністю торкається наступних тематик BOK2001: DS1. Функції, відносини і множини; DS2. Основи логіки; DS4. Основи обчислень; DS6. Дискретна ймовірність; PF1. Основні конструкції програмування; PF4. Рекурсія; PL1. Огляд мов програмування; PL2. Віртуальні машини; PL4. Змінні і типи даних; PL5. Механізми абстракції; SP1. Історія комп'ютінга. У цьому циклі також рекомендується вивчення деяких проблем, які містяться в наступних темах: DS3. Методи доведення (тут рекомендується розгляд тематик – структура формальних доведень; пряме доведення, доведення за допомогою контрприкладу, доведення через зіставлення, доведення через зведення до протиріччя; метод математичної індукції); PF2. Алгоритми і розв'язування задач (стратегії розв'язування задач; роль алгоритмів в процесі розв'язування задач; концепція і властивості алгоритмів; стратегії налагодження); PF3. Фундаментальні структури даних (примітивні типи; масиви; записи; рядки і опрацювання рядків; подання даних в пам'яті; статичне, стекове і пірамідальне розміщення в пам'яті; управління пам'яттю під час виконання програми; посилання і покажчики; зв'язані структури); AL1. Основи аналізу алгоритмів (нотація О-велике; стандартні класи складності; емпіричне вимірювання ефективності; компроміси часу і простору в алгоритмах); AL3. Фундаментальні алгоритми комп'ютінгу (прості числові алгоритми; алгоритми послідовного і двійкового пошуку; квадратичні і O(NlogN) алгоритми сортування; хешування; дерева двійкового пошуку); AR1. Цифрова логіка і

цифрові системи (логічні вентилі; логічні вирази); PL6. Об'єктно-орієнтоване програмування (об'єктно-орієнтоване проектування; інкапсуляція і приховування даних; розділення поведінки і реалізації; класи, суперкласи, спадковість; поліморфізм; ієрархії класів); SE1. Проектування програмного забезпечення (фундаментальні концепції і принципи проектування; об'єктно-орієнтований аналіз і проектування; проектування з метою повторного використання); SE2. Використання програмних інтерфейсів додатків (API) (API програмування; браузери класів і відповідний інструментарій; програмування на прикладах; відладка API оточення); SE3. Інструментарій і оточення програмного забезпечення (програмні оточення; інструменти тестування); SE5. Вимоги і специфікації програмного забезпечення (важливість специфікацій в процесі розробки програмного забезпечення); SE6. Валідізация програмного забезпечення (основи тестування; генерація тестових прикладів).

Всі ці тематики присутні в різних варіаціях в моделях і підходах навчання основ програмування студентів молодших курсів.

Концепції навчання. Педагогічні концепції, що покривають ввідний рівень навчання, поділяються на концепції, пов'язані з алгоритмічним мисленням, основами програмування, а також оточенням комп'ютінга.

У концепції, пов'язаній з формуванням алгоритмічного мислення, мається на увазі навчання студентів алгоритмічнич обчислень, розуміння алгоритмічної ефективності і формування умінь використовувати ресурси. Тому в дисциплінах ввідного циклу при поданні навчального матеріалу необхідно розглядати алгоритми як моделі обчислювальних процесів, ілюструвати сказане прикладами загальновідомих алгоритмів. Студенти 1-2 років навчання повинні проводити простий аналіз алгоритмічної складності, оцінювати вибір альтернативних варіантів, знати методи оцінювання алгоритмів. Студенти повинні уміти складати, читати, пояснювати алгоритми, застосовувати стандартні алгоритми, обговорювати питання, пов'язані з коректністю алгоритму, уміти оцінювати час роботи за алгоритмом і використовувану пам'ять, експериментально оцінювати ефективність алгоритму. У зв'язку з цим необхідні контрольні завдання і завдання для самостійної роботи, що разом з діагностуючою функцією дозволило б формувати вміння абстрагуватися від конкретної мови програмування при розв'язування поставленого завдання і формувати стійкі навички самостійної роботи, які зазвичай недостатньо розвинені у студентів початкових курсів. Як приклад можна навести нерозуміння студентами необхідності вивчення різних методів сортування, оскільки при вивченні певної мови програмування вони звикають використовувати готові бібліотеки алгоритмів і структур.

Для формування у студентів знань основ програмування на ввідному рівні навчання важливим є вивчення моделей даних, що включають стандартні структури подання даних, абстрактні і конкретні описи даних. Студенти повинні вміти читати і пояснювати коди програм, застосовувати і модифікувати програми, вміти використовувати при цьому різні структури даних. Під час лабораторних робіт корисно навчити їх обґрунтовувати вибір в своїй програмі тієї або іншої структури. Тут доцільно використовувати тестові завдання, як відкриті, так і з множинним вибором. У таких завданнях можна запропонувати модифікувати заданий програмний код. Наприклад, якщо заданий фрагмент програми сортуювання масиву в порядку зростання його елементів, то можна запропонувати внести такі зміни, щоб сортування відбувалося в порядку спадання. Також як завдання в тесті можна запропонувати пояснити, для чого саме призначений заданий фрагмент програми. При цьому можна підготувати на вибір декілька варіантів відповіді.

При вивченні основ програмування також важлива концепція структур програмних об'єктів управління, в якій мається на увазі розуміння операцій програми, уміння пояснювати результати виконання операцій, застосовувати і описувати операції. Окрім цієї концепції, суттєва концепція, в якій описується порядок виконання структур управління (порядок виконання, вибір операції, розгалуження, ітерація, виклик функцій, передавання параметрів).

При вивченні основ програмування мають бути засвоєні також концепції, пов'язані з інкапсуляцією і зв'язками між її компонентами. В концепції інкапсуляції мається на увазі, з одного боку, невидимій клієнтові зв'язок об'єктів програми, приховані дані, а з іншого боку, внутрішні деталі програми, видимі розробникам. Зв'язок між компонентами інкапсуляції виводить на перший план питання, пов'язані з призначенням інтерфейсу як засобу, за допомогою якого здійснюється управління процесом реалізації програми. При навчанні основ програмування необхідно також приділяти увагу питанням тестування і налагодження програм, необхідно знайомити студентів із стратегіями налагодження, учити розробляти ефективні тести. Тут корисним може виявитися завдання, яке полягає в обміні студентами своїми програмами з метою їх тестування з написанням специфікації тестів і докладним описом умов виникнення помилок. Необхідно навчити критично оцінювати програму в цілому і передбачати можливі уразливі місця при її виконанні.

Тематика ввідного курсу «Оточення комп'ютінга» включає п'ять концепцій. Перша концепція пов'язана з рівнями абстракції. Тут розглядаються комп'ютерні системи в ієрархії віртуальних машин. У концепції «Мови і парадигми програмування» важливе формування розуміння ролі мов програмування, процесів трансляції програм. У концепції, що стосується тематики «Основні апаратні засоби і подання даних», мається на увазі формування умінь пояснювати машинну організацію комп'ютера, подання даних на машинному рівні. У концепції «Інструменти» передбачається ознайомлення студентів з компіляторами, редакторами і налагоджувачами програм. Тут також доцільне вивчення компонентів супроводу готових програмних продуктів: довідкова система,

настанови для користувача, інсталятор і т.п. В останній концепції як прикладні програмні засоби, необхідні для вивчення студентами, розглядаються браузери, текстові процесори, електронні таблиці, бази даних, поштові системи, інструменти підготовки презентації готового програмного продукту. Корисно проводити захист студентських робіт перед аудиторією для вироблення уміння представляти і захищати свій проект.

В Україні при навчанні студентів молодших курсів комп'ютерних спеціальностей основ програмування в основному переважає імперативний підхід. Протягом останніх кількох років на факультеті інформатики Кримського інженерно-педагогічного університету проводилося анкетування і тестування студентів першого курсу спеціальності «Інформатика». Результати опитування показують, що необхідно приділяти значну увагу формуванню алгоритмічного мислення. Це пов'язано з тим, що в школі більша увага приділяється прикладним програмам і виробленню навичок користувача. Наприклад, тема «Алгоритмізація і програмування» в школі вивчається на рівні синтаксису і семантики певної мови програмування (частіше всього Pascal). Таким чином, якщо першокурсник може записати код на основі готового алгоритму, то розв'язування задачі «з нуля» для нього стає проблематичним. Що стосується навичок написання програм, доцільно з самого початку приступати до вивчення принципів об'єктно-орієнтованого програмування і проектування. Це пов'язано з відмінностями в початковому рівні підготовки першокурсників. Частина з них вже знайома з імперативними мовами, а частина не має навичок програмування взагалі.

Висновок. Таким чином у ввідному циклі при навчанні студентів основ програмування можна використовувати одну з шести представлених вище моделей навчання. У всіх шести різних підходах до реалізації моделі навчання приділяється увага розвитку у студентів алгоритмічного мислення, концептуальних знань щодо програмування, а також ознайомленню студентів з оточенням комп'ютінга.

ЛІТЕРАТУРА

1. Computing Curricula 2005. The Overview Report. – A volume of the Computing Curricula Series. – A cooperative project of the ACM, the AIS, the IEEE-CS. 30 September 2005. – 55 p.
2. Сейдаметова З.С. Подготовка инженеров-программистов по специальности «Информатика». – Симферополь: Крымучпедгиз, 2007. – 480 с. – Бібліогр.: с. 329–363.
3. Chang C., Denning P.J. (chairs) et al. Computing Curricula 2001: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science / A volume of the Computing Curricula Series – IEEE CS Press, ACM Press, 2001. – 240 p.
4. Computer Science Curriculum 2008 (CS 2008): An Interim Revision of CS 2001 (draft) – ACM, IEEE-CS, 2008. – 102 p.