ПРОГРАМИРАНЕ И ПРОГРАМНИ ПАРАДИГМИ – ДИДАКТИЧЕСКИ АСПЕКТИ

Ивайло Дончев

1. Понятието „програмиране”

Програмирането е ключов елемент в информатиката – важно практическо умение и, съответно, съществена част от бакалавърските програми. Въпреки това, сред академичната общност няма единно мнение дори за дефиниция на понятието „програмиране“. Самите програмисти считат, че „всеки опит да се даде кратко определение води или до нещо толкова неясно и неразбрано, че само по себе си няма смисъл, или до разтегнат текст който дава смисъл в общия случай и след това оборва сам себе си с всички частни случаи, за които се е сетил авторът“ [10]. За това и в учебниците рядко се дава дефиниция на това понятие, а по-скоро се коментират елементи от неговия обем и съдържание.

В исторически план точно обемът и съдържанието на понятието „програмиране“ търпят развитие. До появата на алгоритмичните езици процесът на програмиране е представлявал подготовка на изчисления за машината – съставяне на график за последователността на отделните операции, необходими за изчисленията. Разликата между програмата и другите форми на изчисления е, че последователността от команди се изпълнява автоматично, без намесата на човек. С появата на алгоритмични езици, програмирането вече се разглежда и като „процес на превеждане от удобен за човека език в език, удобен за компютъра“ [11]. В тези ранни години програмите решават предимно математически задачи и програмирането е „черната работа“ по намирането на точно математическо формулиране и метод за решаването на задачата, в нотацията на подходящ проблемно-ориентиран език, чиито символи са в тясна връзка с решавания проблем. И днес сред студентите е широко разпространено убеждението, че програмирането е „създаване на последователност от команди, които да позволят на компютъра да свърши нещо“ [16] или „да кажеш на компютъра как трябва да си свърши работата“ [17].

В течение на годините развитие, програмните приложения излизат далеч извън областта на математиката, в посока към обработка на много по-обща информация. Променят се и същността на основните операции и езиковите конструкции. Еволюират методите за формулиране

и решаване на проблемите. Същността на процеса на програмиране се променя от описание на изчисления, през дефиниране на функции, към създаване и работа с обекти [21], но основата е същата – да се пишат „последователности от кодирани инструкции“, които се подават на компютъра; да се „подреждат данните, така че да могат да бъдат обработени“ и да се “зарежда програмата в компютъра” [7]. Това обаче обхваща само една част от дейностите, които трябва да извършва професионалният програмист – техническата част от неговата работа (дизайн и кодиране). Социалният контекст на програмистката работа предполага умения за създаване и интерпретиране на документи със спецификации, дискусии по дизайна, преценка на необходимите разходи и усилия за реализиране на проектите и др.

Друг аспект на понятието „програмиране“ е свързан с количеството програмен код, необходим за имплементирането на системата. Макар че основният процес на разработка по същество е един и същ, терминът „програмиране“ обикновено се прилага към изграждането на малки до средно големи софтуерни приложения [22, с. 7]. За дейности по конструирането на средни, до големи софтуерни системи се предпочитат термините „софтуерно инженерство“ или „системно инженерство“. Разликата идва от там, че по-големият мащаб на решавания проблем предполага, освен „конвенционално“ програмиране, да се прилагат и по-сложни техники и инструменти.

В последно време нараства и броят на неспециалистите, на които се налага да програмират, като част от служебните им задължения. Тенденцията се обуславя от факта, че всички значими софтуерни приложения включват скриптов или макроезик, използван за конфигуриране и персонализиране на поведението на програмите. Така е и с повечето операционни системи. Може би най-типичен е примерът с масово използваните електронни таблици. Сами по себе си те са декларативен език за програмиране. С тяхна помощ много крайни потребители (*end-users*) създават първоначално малки приложения, улесняващи тяхната лична работа, които много често еволюират до сложни бизнес приложения, стратегически за цялата фирма. На практика се преминава през целия цикъл жизнен цикъл – от спецификациите, през дизайна, кодирането, тестването и поддръжката. Феноменът е известен с термина „*end-user programming*“. Освен електронните таблици, крайните потребители (икономисти, счетоводители, лекари, инженери, и др.) често се изявяват

като програмисти в областта на базите от данни и web-приложенията. Така границите на програмистката професия се размиват още повече. Към това можем да прибавим и свободния контекст, в който хората използват думата „програмиране“. В ежедневната реч често с “програмиране” се означават дейности, които доста се разминават с представата ни за изграждане на големи софтуерни приложения, например програмиране на *HTML*. Дори нещо повече, хората „програмират“ климатика, готварската печка, *DVD*-рекордера, радиото в колата. Макар и толкова обикновени и битови, тези дейности имат допирни точки с програмирането – за тях също е характерно, че потребителят не манипулира директно видими неща, но определя поведение, което ще се прояви в бъдещ момент [5]. Така че може би най-широката дефиниция е „човешка дейност, означаваща действията по разширяване или променяне на функционалността на дадена система“ [17, с. XXIX]. Тази дейност може да се извършва от неспециалисти (например, потребители, които променят настройките на смартфона) и специалисти (компютърни програмисти).

Що се отнася до традиционното „компютърно програмиране“, то е определено в [19] като „преодоляване на пропастта между спецификациите и работещата програма“. Това включва проектирането на архитектурата и абстракциите в приложението и кодирането им на език за програмиране. И така, въпреки че няма общоприета дефиниция и ограничавайки обема на понятието до традиционните представи, с термина “програмиране“ е коректно да означим „целия процес на разработване на софтуер, който започва с откриването на основните характеристики на проблема, преминава през разработването на систематичен метод за автоматизираното му решаване и, в крайна сметка, представянето на този систематичен метод в точните изразни конструкции на конкретен език за програмиране” [12].

2. Програмни парадигми

Програмирането е обширна дисциплина и за това преобладаващата част от езиците за програмиране са доста сложни. Важните концепции, заложени в тях, обаче, са прости. Програмното решаване на проблем изисква да се изберат точните концепции. Всички проблеми, с изключение само на най-тривиалните, изискват различен набор от концепции за решаването на различни части от проблема. Ето защо много от популярните езици позволяват да се програмира в два (някои и в повече) различни стила (парадигми). Понятието „програмна парадиг-

ма“ можем да дефинираме като „подход към програмирането на компютър, базиран на математическа теория или съгласуван набор от принципи“ [20]. Тя е фундаментален стил, определящ методологията на проектиране и програмиране, включително конструкциите на езика, взаимодействието между структурите от данни и операциите с тях, структурата на програмата и как най-общо се анализират и решават проблемите.

Всяка парадигма подкрепя множество от концепции, които я правят най-подходяща за определен кръг от проблеми. Например обектно-ориентираното програмиране (ООП) е най-подходящо за случаите, когато имаме голям брой свързани абстракции на данни, организирани в йерархии, а логическото програмиране е най-удобно за преобразуване или управление на сложни символни структури, в зависимост от определени логически правила. Езици, които имплементират тези две парадигми са, съответно, *C#* и *Prolog*.

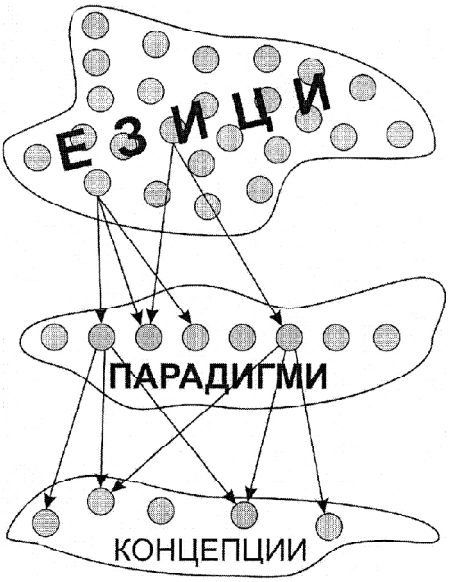
Парадигмата определя и начина, по който програмистът „вижда“ изпълнението на програмата. Именно това го насочва какви абстракции да търси в проблемната област. Например при ООП програмистът мисли за работещата програма като за съвкупност от взаимодействащи си обекти и търси за моделирането ѝ обекти, класове от обекти и връзки между тях. При функционалното програмиране програмата се разглежда като последователност от пресмятания на функции, които нямат свои състояния. Единствената възможна декомпозиция тук е в термините на функции.

Най-добрият вариант би бил един език да поддържа много и добре реализирани концепции, така че програмистът да може да избере точно тези, които са най-подходящи за конкретния случай, без да се ограничава до предлаганите такива от една или максимално две парадигми. Такъв стил на програмиране се нарича мултипарадигмен (*multiparadigm programming*). За съжаление популярните езици все още не поддържат директно подобни механизми, а само библиотеки, разширяващи тяхната функционалност. Въпреки това, познаването на повече парадигми и разбирането на лежащите в основата им концепции може да подобри стила и ефективността на програмиране дори ако се ползват езици, които не поддържат директно желаната парадигма – както е възможно ООП на *C* [15] или *Fortran* [4] със съответната нагласа на програмиста.

Познати са десетки парадигми. Концепциите са основните примитиви, от които се изграждат парадигмите. Повечето парадигми се отли-

чават по съвсем малко на брой концепции, но това може да доведе до коренна разлика в начина на програмиране.

Важно е да се подчертае връзката между езиците за програмиране, програмните парадигми и концепциите, залегнали в тях (фиг. 1). Парадигмите са много по-малко на брой от езиците. Например *Java*, *Javascript*, *C#*, *Ruby*, и *Python* са пет различни езика, които имплементират една парадигма – обектно-ориентираната. Един език може да имплементира няколко парадигми, най-често две. Парадигмите от своя страна са конструирани от много по-малко на брой концепции. Тоест основният градивен блок е концепцията. Концепцията може да бъде или понятие (клас, обект, нишка), или механизъм (наследяване, предаване на съобщения, предефиниране). Тъй като концепциите са най-малобройното множество, доста от парадигмите имат общи концепции.



Фиг. 1. Връзки между езици, парадигми и концепции

Можем да определим като „главни“ 4 от парадигмите [6] [8] [14]: императивно (*imperative*), логическо (*logical*), функционално (*functional*) и обектно-ориентирано (*object-oriented*) програмиране. Всички останали представляват разширения на главните парадигми и съдържат сходни концепции. Дори функционалното и логическото програмиране могат да

се разглеждат като разширения на по-общата декларативна (*declarative*) парадигма. Така основните парадигми могат да се редуцират до три. Ще направим уточнението, че тези основни парадигми се отнасят само до последователно (*sequential*) програмиране. В съвсем друга категория попада паралелното (*parallel*) програмиране. Така че е възможно разделянето и само на две групи парадигми. Разбира се, такова разделяне може да бъде само условно, доколкото е възможно паралелно програмиране в декларативен или обектно-ориентиран вариант.

3. Мултипарадигмено програмиране

Всяка програмна парадигма има своите предимства и недостатъци. Необходимостта от владеенето на повече от една се подчертава и в двата фундаментални доклада – CC2001 [3, с. 89] и CS2008 [2, с. 19]. Там, обаче, става въпрос за това, че изборът на парадигма може чувствително да повлияе на начина, по който се мисли за проблемите и се представят решенията. Препоръчва се изучаването на език, който поддържа поне две различни парадигми (повечето съвременни са такива), но акцентът е върху идеята да се използват различни езици за различни цели и студентите, познавайки различията в парадигмите и дизайна на езиците, да могат по-лесно и бързо да усвояват нови езици [3, с. 113].

Идеята на мултипарадигменото програмиране е друга – за всеки решаван проблем да се използва най-подходящата концепция, независимо към коя парадигма принадлежи тя и в една програма (написана на един език) да могат да се смесват концепции от различни парадигми. Тази идея не е нова – можем да отнесем нейните корени към дизайнерите на езиците *Lisp* и *Scheme*.

В съвременното висше образование по компютърни науки се преподават само няколко парадигми, като все още обектно-ориентираната доминира. По-малко внимание се отделя на функционалното и логическото програмиране. Всяка от тези парадигми се преподава отделно от останалите и така студентите не могат да видят връзките между тях и как могат да се използват заедно, а това, според [18], в последствие ще повлияе на компетентността на програмистите и качеството на тяхната работа. В [18] се дава пример със студенти, които учат за паралелизма само от *Java*. Те остават с впечатлението, че тази концепция е винаги трудна и скъпо струваща, което е заблуда – използвайки парадигмата, конкурентно програмиране може да направи задачата толкова проста, колкото и последователното програмиране. Правилният

подход е програмирането да се преподава от гледна точка на концепциите, а не да се базира на отделен език или парадигма. След това основните парадигми съвсем естествено се появяват, в зависимост от използваните концепции. Така студентите имат много по-задълбочено и обосновано виждане за дизайна на програмите си, за тяхната коректност и сложност.

Някои автори, например [9] поддържат тезата, че студентите по компютърни науки трябва да бъдат обучавани в повече от една програмна парадигма още от първата година на своето следване, като се аргументират, че естественият начин на декомпозиране на проблема е на подпроблеми, в зависимост от необходимите концепции за тяхното решение. Така всяка от използваните парадигми допринася за дизайна на цялостното решение. За да е възможно такова обучение трябва да се използва или мултипарадигмен език (например *Leda* който има елементи на функционален, обектно-ориентиран и логически език), или да се преподава множество от езици, които могат да се използват заедно за имплементиране на цялостния проект (например с *Microsoft Visual Studio*).

Като мултипарадигмен език можем да определим и широко използвания в уводните курсове по света и у нас език *C++*. В последната си версия от 2011-та година той поддържа директно процедурно, обектно-ориентирано, паралелно, функционално и генерично (родово) програмиране. С помощта на допълнителна библиотека, към тези парадигми можем да добавим и логическо програмиране (библиотеката *Castor*) [13].

Изучаването на всички тези възможности на *C++* в уводния курс е невъзможно, поради изключително сложните синтаксис и семантика. Приложимостта на един мултипарадигмен език зависи от това колко добре са интегрирани в него отделните парадигми [17, с. XXXI]. Добри резултати има от използването на друг мултипарадигмен език – *Oz* и по-конкретно неговата имплементация *Mozart*. *Oz* директно поддържа декларативно, функционално, обектно-ориентирано, конкурентно (вкл. разпределено) и логическо (вкл. *constraint*) програмиране.

4. Програмните парадигми в обучението на информатиците

Голямото разнообразие от програмни парадигми затруднява избора на тези от тях, които трябва да присъстват в учебните планове на конкретните специалности. Сред колегията няма консенсус дори дали основна парадигма като функционалното програмиране трябва да е задължителна за всички бакалавърски програми по компютърни науки [2, с. 19]. Тя се препоръчва безусловно само за специалности, подгот-

вящи кадри за научна работа. Вместо функционално програмиране, за специалностите, подготвящи web-програмисти се препоръчват скриптови езици. Няма спор за необходимостта от изучаване на процедурно и обектно-ориентирано програмиране – тези парадигми са задължителни и влизат в ядрата от теми на всички специалности, като повишено внимание и грижа трябва да се отделя на основните програмистки техники (*basic programming*) [2, с. 17]. В частност това се отнася до сигурността. Студентите трябва да са наясно защо е необходимо да се програмира по такъв начин, че да не остават „вратички" за заобикаляне, по отношение на сигурността.

В областта от знания „Основи на програмирането" (*Programming Fundamentals*) на CS2008 попадат „концепции, знания и умения, които са съществени за всяка програмистка практика, независимо от прилежащата парадигма" [2, с. 40]. В CC2005 обаче същата област е определена като „основни концепции на процедурното и обектно-ориентираното програмиране" [1, с. 55]. По същество става въпрос за едни и същи теми:

– **основни императивни понятия**: променливи, типове, изрази, присвояване, условни и итеративни управляващи конструкции);

– **процедурни и структурни концепции**: функции и предаване на параметри, структурна декомпозиция;

– **основни структури от данни**: масиви, низове, указатели, динамична памет, свързани списъци, стекове, опашки, хеш таблици, графи, дървета;

– **алгоритмични стратегии**: включително за дебъгване;

– **рекурсия**;

– **събитийно програмиране**;

– **ООП**;

– **основи на информационната сигурност и защитено програмиране** (*secure programming*).

Това още веднъж доказва важната роля на двете парадигми – процедурна и обектно-ориентирана. В сравнение с препоръките от CC2001 [3, с. 89], в CS2008 се забелязва отчетливо акцентът върху сигурността – за нея в ядрото от задължителни знания са отделени най-малко 6 от общо 47 часа. В по-старите препоръки сигурността фигурира само като тема за напреднали (*advanced topic*), с сега вече е част от ядрото и навлизането в тази тематика започва още от уводните курсове за всички студенти. В приложение Appendix C от CS2008 [2, с. 107] са

описани препоръки към учебното съдържание на отделен курс за напреднали, озаглавен „Въведение в сигурността“ или „Въведение в компютърната сигурност“. Някои от темите спокойно могат да бъдат включени в уводните курсове, като се отчетат особеностите на избрания за целта подход.

Разделът „Основи на програмирането“ не покрива целия спектър от знания по програмиране, които бакалаврите по компютърни науки трябва да притежават. Други основни области, в които попадат свързани с програмирането теми от ядрото са езиците за програмиране и софтуерното инженерство, но повечето от темите, включени там се отнасят предимно за по-горните (*advanced*) курсове.

Засилващата се роля на паралелизма и конкурентното програмиране се отчита в CS2008, но фокусирането върху тази парадигма при организирането на обучението попада в рамките на алтернативните подходи, наред с програмирането на игри [2, с. 27 – 28], тоест все още се счита за екзотично. Очакваме тази ситуация скоро да се промени, предвид на бурното развитие на хардуера в посока към паралелизъм. Това ще засили влиянието и на декларативните парадигми, особено на функционалното програмиране.

Фактът, че популярните езици (*C++*, *Java*) и среди за разработка включват поддръжка на все повече програмни парадигми е индикация за това, че назрява моментът да се експериментира по-смело с прилагането на „чисти“ мултипарадигмени езици още в уводните компютърни курсове.

5. Заключение

Програмирането е основна дисциплина в обучението на информатици. То развива способности за решаване на проблеми и осигурява начин за изразяване на абстрактни идеи. Това го прави нещо повече от професионално умение. Програмирайки, студентите усвояват много умения, важни за всички професии – критично и аналитично мислене, отделяне внимание на детайлите и др. За да станат добри програмисти, обаче, те трябва да познават в детайли особеностите на различните програмни парадигми и да могат да преценяват кои концепции и механизми са подходящи за решаването на конкретните, поставени пред тях задачи.

Посоката, в която се развива изчислителната техника е към масово навлизане на многоядрени и многопроцесорни устройства и разпределени системи. Това обуславя засилващото се влияние на функционал-

ното и паралелното програмиране, както в индустрията, така и в образованието. Необходимо е интегрирането на тези парадигми в курсовете по програмиране от всички нива, включително уводното. За това спомага и развитието на традиционно използваните в обучението езици в посока към директна поддръжка на тези парадигми.

ЛИТЕРАТУРА

1. ACM/AIS/IEEE-CS Joint Task Force for Computing Curricula 2005, Computing Curricula 2005. The Overview Report, In Proceedings of the 37th SIGCSE technical symposium on Computer science education, ACM New York, 2006.

2. ACM/IEEE CS2008 Review Taskforce, Computer Science Curriculum 2008: An Interim Revision of CS 200*1*, Technical report, ACM/IEEE CS, 2008.

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

4. *Akin, Ed.* Object Oriented Programming Via FORTRAN 90/95, Cambridge University Press New York, 2003.

5. *Blackwell, A.* What is programming, In Proceedings of Psychology of Programming Interest Group, 2002, pp. 204 – 218.

6. *Bolshakova, E.* Programming Paradigms in Computer Science Education, International Journal on Information Theory and Applications, Volume 12, Number 3, 2005, pp. 285 – 291.

7. *Collins:* English Dictionary Definition (Meaning) of 'program', available at http://www.collinslanguage.com

8. *Gabbrielli, M., S. Martini.* Programming Languages: Principles and Paradigms (Undergraduate Topics in Computer Science), Springer, 2010.

9. *Gewali, L., J. Minor:* Multi-Paradigm Approach for Teaching Programming, In Proceedings of the 2006 International Conference on Frontiers in Education: Computer Science & Computer Engineering, FECS 2006, Las Vegas, Nevada, USA, June 26-29, 2006, pp. 141 – 146.

10. Griffin. *Базис*. Що е програмиране?, http://prog.ludost.net/base/programming.html

11. *McCracken, D.* Digital computer programming, Wiley, 1957

12. *McGettrick, A., R. Boyle, R. Ibbett, J. Lloyd, G. Lovegrove*. Grand Challenges in Computing: Education – A Summary, The Computer Journal (2005) 48 (1): 42 – 48.

13. *Naik, R.* Introduction to Logic Programming in C++, 2010, available at http://www.mpprogramming.com/resources/CastorTutorial.pdf

14. *Nurmark, K.* Object-oriented Programming in C# for C and Java programmers, Department of Computer Science, Aalborg University, Denmark, 2010, available at http://www.cs.aau.dk/~normark/oop-09/pdf/all.pdf

15. *Schreiner, A.* Objektorientierte Programmierung mit ANSI C, Hanser, 1994.

16. TheFreeDictionary, Dictionary/thesaurus – Programming, available at http://www.thefreedictionary.com/programming

17. *Van Roy, P., S. Haridi.* Concepts, Techniques, and Models of Computer Programming, The MIT Press, Cambridge, Massachusetts, 2004.

18. *Van Roy, P., J. Armstrong, M. Flatt, B. Magnusson.* The Role of Language Paradigms in Teaching Programming, In Proceedings of the 34th SIGCSE technical symposium on Computer science education, ACM, New York, 2003, pp. 269 – 270.

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

20. *Van Roy, P.* Programming Paradigms for Dummies: What Every Programmer Should Know, In G. Assayag and A. Gerzso (eds.) New Computational Paradigms for Computer Music, IRCAM/Delatour, France, 2009.

21. *Weinberg, G.* The Psychology of Computer Programming: Silver Anniversary Edition, Dorset House, 1998

22. *Winder, R., G.* Roberts. Developing Java Software (third edition), Wiley, 2006.

ПРОГРАМИРАНЕ И ПРОГРАМНИ ПАРАДИГМИ — ДИДАКТИЧЕСКИ АСПЕКТИ

ИВАЙЛО ДОНЧЕВ

Резюме

В тази статия се коментират различните гледни точки към понятието „програмиране“, проследява се историческото развитие на схващанията за това какво представлява процесът на програмиране. Посочени са нетрадиционни дейности, които също могат да бъдат определени като програмиране. Обяснява се защо е трудно да се даде точна дефиниция на това понятие. Анализирано е влиянието на програмните парадигми върху процеса на програмиране и значението им за обучението на студенти-информатици.

Ключови думи: програмиране, програмни парадигми, мултипарадигмено програмиране, обучение по програмиране.

PROGRAMMING AND PROGRAMMING PARADIGMS – DIDACTICAL ASPECTS

IVAYLO DONCHEV

Summary

This article discusses the different perspectives to the concept of programming. It tracks the historical development of conceptions of what constitutes the programming process. There are pointed non-traditional activities, which may also be defined as programming. The article also explains why it is difficult to give a precise definition of this concept. The impact of programming paradigms over the programming process and their significance to the training of computer science students are analyzed.

Keywords: programming, programming paradigms, multiparadigm programming, education