# ТАКСОНОМИЯТА НА БЛУМ И ОБУЧЕНИЕТО В СТИЛ НА ПРОГРАМИРАНЕ

***Теодоси Теодосиев***

*Шуменски университет*
*t.teodosiev@fmi.shu-bg.net*

***Резюме**: В работата е използвана таксономията на Блум за определяне на целите на обучението в стил на програмиране. Показани са елементи на стила на програмиране, в които може да се обучават студентите още в началото на изучаването на програмирането. Елементи на стила на програмиране се намират на различни нива в пирамидата на Блум.*

***Ключови думи**: таксономия на Блум, програмиране, обучение, стил на програма*

## 1. Въведение

Целта на образованието неминуемо се свързва с образователния идеал на времето и мястото и се обуславя от влиянието на много фактори. Целите на образованието и обучението не обслужват единствено дейността на професионално заетите в образованието лица, а всички потребители на образователен продукт, т.е. цялото общество.

В традиционното обучение, обучението в дейност, ако има такова, се ограничава до минимум и основното съдържание остава "усвояване на готови знания". Затова традиционното обучение се нарича "обучение в знание". В него господстващо място заемат знанията. Смята се, че в името на знанието се осъществява обучение. Такова разбиране за целите на обучението се счита за нормално и не предизвиква съмнение. Заблудата на този подход започва да се разбира от края на 50-те години на миналия век, когато психологът и педагог П. Галперин в своите изследвания показва, че целта на обучението е да се дадат на човека умения да действа, а знанията трябва да станат средство за обучението в действия.

Подходът, който се използва при обучение по програмиране на студентите, е паралелно изучаване на алгоритмизацията и език за програмиране. Този подход има своите предимства. Желателно е студентите възможно най-рано да имат възможност да проверяват точността на своите алгоритми на компютър. А за тази цел те трябва да се запознаят с език за програмиране, да овладеят системата за програмиране [3]. Освен това се разчита на начална подготовка от средното училище.

Непрекъснатото обучение играе основна роля в курса по програмиране. Причината за това е, че разбирането на следващите понятия, които се

преподават, до голяма степен зависи от разбирането на тези, които преди това са усвоени. Изучаването на езика за програмиране е в контекста на решаваните задачи, т.е. новите средства на езика се въвеждат по необходимост за справяне с друг тип задача.

Възниква въпрос дали не може паралелно с горните дейности да се обучава и в стил на програмиране? Ще се опитаме да обосновем тази възможност, като разгледаме елементите на стила на програмата в терминологията на таксономията на Блум.

## 2. Таксономии на целите

Цел на педагогическите таксономии е да се определят и подредят педагогическите цели на обучението, реализацията на които води до мислене от високо ниво (по-висок ранг). Според автора в [10] мисленето от по-висок ранг се характеризира с: липса на алгоритъм (пътят на дейността не е предварително напълно определен); комплексност (пътят не е видим от една гледна точка); наличие на множество решения (всяко с различна стойност и предимства); нюансирана преценка и тълкуване; прилагането на множество критерии (някои в конфликт помежду си); безпорядъчна структура; несигурност и саморегулиране на процеса. Когато човек има умения за мислене от високо ниво, той не само разбира същността на изучаваните предмети и явления, но и открива закони и закономерности, принципи и правила, разкрива тенденции за развитие и бъдещо поведение, участва в конструктивна, съзидателна и творческа дейност, съзнателно и разумно ръководи и контролира собственото си индивидуално и социално поведение.

Таксономиите играят съществена роля в теорията на обучението. Те са важни, защото позволяват:

- ✓ Правилно да се определят и степенуват целите и задачите на обучението;
- ✓ Разностранно да се планират дейностите в процеса на обучението;
- ✓ Да се обособяват познавателните структури в обучението;
- ✓ Да се формулират проблеми и да се поставят задачи пред обучаемите;
- ✓ Да се подбират оценъчни инструменти, адекватни на поставяните цели;
- ✓ По резултатите от обучението и изпитаните от обучаемите трудности да се коригира процеса на обучение [12].

Когнитивната таксономия на Блум е изградена като йерархия от шест равнища от запомняне и възпроизвеждане на изучения материал до решаване на проблеми, в хода на което е необходимо да се преосмислят наличните знания, да се изграждат нови техни съчетания с предварително изучени идеи, методи, процедури (начини на действие), включително и да се пристъпи към ново

решение. Насочена е към формиране на интелектуалните умения. Блум класифицира интелектуалното поведение в шест категории: знание, разбиране, приложение, анализ, синтез, оценка [7].

От публикуването на тази таксономия досега са намерени редица нейни слабости и практически ограничения [5, 8 и др.]. През 2001 г. група изследователи, начело с Андерсън, публикуват ревизирана версия на таксономията на Блум [6].

В обновената версия, познанието не е едноразмерно, а има две измерения: знание и познавателен процес. Знание е съдържанието по предмета, а познавателният процес показва какво трябва да се направи с предметното съдържание. Познавателният процес отразява различни форми на мислене, а тъй като мисленето е активен процес се използват глаголни форми. Това измерение има шест категории, както и в оригиналната таксономия, но те са преименувани и преобразувани [2]. Тъй като знанието може да бъде резултат или продукт на мисленето, а не форма на мислене, оригиналната категория знание е заменена със запомняне. Категориите разбиране, приложение, анализ и оценка са запазени, но под формата на отглаголни съществителни: разбиране, прилагане, анализиране, оценяване. Според авторите в [6] в процеса на творчество индукцията е по-сложен процес от дедукцията. Затова авторите променят реда на категориите: от синтез към оценка, на оценяване – създаване.

✓ Запомняне – запомняне и възпроизвеждане на изучаван материал. Отнася се от факти до теории. Основното е способността да се възпроизведе необходимата в момента информация.

✓ Разбиране – способността да се представи смисъла на изучаването чрез различни форми, интерпретации, сравнения. Представя собствено мнение за изучавания материал. Проявява се при трансформиране на знанията чрез обяснение и обобщение. Използва се информация от вече изучавани области.

✓ Прилагане **–** способност да се използва наученото в нови, конкретни ситуации. Свързва се с прилагането на принципи, правила, концепции, методи, теории.

✓ Анализиране – способност за разделяне на дадена материя на съставните ѝ части, за да се разбере и изследва структурата ѝ. По-високото ниво за схващане на смисъла се свързва освен със съдържанието и със структурата на материала.

✓ Оценяване – способността да се отсъжда по отношение на изучаваното, като се използват външни обективни или вътрешни субективни критерии. Резултатите от учебния процес изискват най-високо ниво в когнитивната

област, защото съдържат елементи от всички други категории, плюс съзнателно оценяване на стойности.

✓ Създаване – обединяване на новите части, за да се получи ново цяло. Определяне на компонентите на новата структура. Изисква се творческо поведение с акцент върху разработката на нови модели и структури.

## 3. Построяване на стила на програмиране върху пирамидата на Блум

Програмиране в тесния смисъл на думата, разбирано като кодиране на изучавания език на готови алгоритми, без да се вниква в тяхната същност и без умения за тяхното развитие, не може да бъде цел на общообразователен курс (още по-малко на профилиран курс). Основната цел на изучаването на език за програмиране е не толкова самият език, колкото придобиването на знания и умения за алгоритмизация в нейния структурен вариант, разработване на методи за решаване на клас от задачи, които традиционно се решават на подобни езици.

Доколкото стилът на програмиране е неформализирано понятие от висок ред, строгото определение е невъзможно. Все пак, има няколко елемента, общи за голям брой програмни парадигми[5]. Под стил на програмиране разбираме набор начини или методи на програмиране, които се използват от опитните програмисти, за да получат правилни, ефективни, удобни за прилагане и лесни за четене програми. Когато програмист усвои определен стил на програмиране неговата програма става значително по-лека за възприемане [1].

Възможността да се напише хубава и елегантна програми е вече много важно умение, което не винаги е във фокуса на компютърната наука и курсове по програмиране. Има едно добро наблюдение направено от Керниган и Пайк [ 9] за стила на програмата:

*“В свят... на непрекъснат натиск за повече от всичко, единственото което можем да пропуснем са основните принципи – простота, яснота и общност – които формират здравата основа на добрия software.”*

[5] Парадигма на програмиране — това е съвкупност от идеи и понятия, определящи стила на написване на програма. Парадигмата на програмирането определя в какви термини програмистът описва логиката на програмата. Например, в императивното програмиране програмата се описва като последователност от действия, във функционалното програмиране се представя във вид на изрази и множество от дефиниции на функции. В популярното ООП програмата се разглежда като набор от взаимодействащи обекти. Важно е да се отбележи, че парадигмата на програмиране не се определя еднозначно от езика за програмиране – много съвременни езици допускат използване на различни парадигми [4].

Елементите на стила могат да се разделят на три групи:

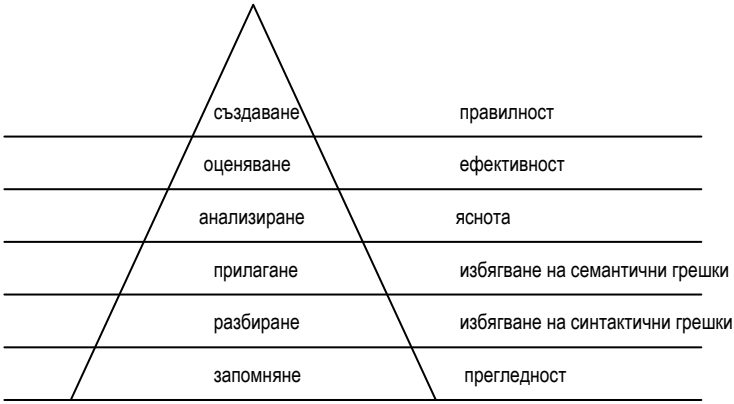
✓ *Имащи отношение към прегледността и стандартизацията на оформяне на кода;*

✓ **Подпомагащи откриването на грешки и предпазването от тях;**

✓ Имащи отношение към правилност и ефективност.[6]

Веригата, по която се развива процесът за овладяване на стила на програмиране е следната:

Прегледност -> избягване на синтактични грешки -> избягване на семантични грешки -> яснота -> ефективност -> правилност. Това са шест етапа в усвояването на стила на програмиране и сякаш е естествено да се съпоставят горните елементи на стила на нивата в пирамидата на Блум (Фигура 1). И тук, за да се премине на по-високо ниво трябва да се усвоят по-долните.




*Фигура 1.*

Първите три етапа могат да се реализират с императивни правила. От казаното по-горе става ясно, че у обучаемите не се формират умения за мислене на високо ниво, ако те: само запаметяват и възпроизвеждат информацията; разбират и обясняват най-общо понятията и идеите; нито пък ако прилагат информацията и прилагат правила в познати ситуации. На тези нива се справят и по-слабо успяващите студенти.

Последните три обаче изискват активното и съзнателно участие на обучаемите. Умения за мислене на по-високо ниво се развиват, когато обучаемите са ангажирани и анализират информация за проучване на знания и взаимовръзки; оценяват решението или начина си на

[6] Форматирането по-надолу показва принадлежност към съответната група.

действие; създават нови идеи, продукти, гледни точки. Така високите нива на мислене са свързани с трите категории на познавателния процес: анализиране, оценяване и създаване в ревизираната таксономия на Блум.

Може да се твърди, че по тази схема върви и обучението по програмиране. Започваме с оформянето на програмния код, преминаваме през коригирането на грешките, за да стигнем до правилните програми.

По-внимателното разглеждане обаче показва, че в дълбочина нещата са по-различни. Оказва се, че елементи на стила на програмиране от различните групи се появяват на различни нива в пирамидата. Ето например, как на различните нива от йерархията на Блум могат да се включат примери от различните групи елементи на стила.

1. Ниво Запомняне.

*Оформянето на програмния код.* Акцентът пада върху акуратното форматиране (използване на отстъпи и празни редове за разкриване на програмната структура). Тези стилови особености имат препоръчителен характер.

**Превантивно използване на съставен оператор.** Докато не се разбере непосредствено езика за програмиране, не може да се видят очевидните синтактични грешки. В течение на първата фаза на изучаване започват да се разпознават неща, които обикновено се наричат “стил на кодиране”. Пишейки програмата, се използват такива правила, които правят грешките “очевадни” или предпазват от такива!! Спазвайки определени правила, се избягват най-типичните за “новаците” грешки.

Използване винаги на операторни скоби в структурите за контрол, спестява грешки при употребата на съставни оператори.

2. Ниво Разбиране.

*Стандартизация на оформянето на програмния код – използване на носещи смислова стойност имена на обекти.* Подходящият избор на имена на обекти се разглежда като крайъгълен камък за добър стил. Идентификаторите (т.е. имена на променливи, функции, типове, структури и т.н.) съобщават на читателя чрез името на обекта функционалната му роля.

**Използване на ляво сравнение в структурите за контрол.** В езиците, които използват един символ *(=)* за присвояване и друг *( ==)* за сравнение (например C/C++, Java, PHP, Perl), когато присвояванията могат да бъдат направени в рамките на структурите за контрол, предимство е мястото на константи или изрази да е вляво на всяко сравнение. Изразите с ляво сравнение *(например 6+к == с)* в C/C++ биха предизвикали грешка при компилация, при пропускане на едното = (обичаен пропуск при новаците), което няма да се случи при дясното (стандартно) сравнение [11].

**Излишната „точка и запетая”.** Това е много характерна грешка за новаците в програмирането. Ако има някаква грешка в if, те добавят „;” в очакване това да реши проблема. Неприятното е, че това действие не води до сигнал от компилатора и грешката се открива трудно.

Да се избягва смесване на типовете данни. Студентите могат да разберат, че резултатът от числов тип с плаваща запетая трябва да се отреже, ако се съхранява в цяла променлива, но е по-трудно да се разбере загубата на точност при съхраняване на стойност на израз с целочислено деление в променлива от тип double или float. Особено внимание е необходимо при използване на изрази, включващи операция деление с цели операнди, защото резултатът е цяло число. Проблемът се дължи на разликата между математическите знания и компютърната интерпретация на аритметичните операции. Различието става явно след компилиране и изпълнение на такъв код.

3. Ниво Прилагане.

Когато се разглежда темата за **област на действие на обектите,** се сблъскваме със затруднения по отношение областта на действие на променливите дефинирани в do-while блокове и в главата на for-цикли. Само имащите отношение към цикъла контролни изрази трябва да бъдат включени в конструкцията `for ( )`. Прави се ясно разграничение на това, което се контролира и това, което се повтаря в цикъла.

Съобразяване с особеностите на машинната аритметика. Обучаемите изпадат в противоречие с математическите си умения/знания, защото машинната аритметика се различава от традиционната. Написването на програмата трябва да е съобразено и с особеностите на компютърните пресмятания. Препоръчително е да се избягват аритметични операции с числа с много различни порядъци. За да се намалят грешните резултати при изчисления с реални числа следва: да се избягва деление на големи по модул числа на малки, особено, ако последните са с неголяма точност; събирането /изваждането на дълги последователности от числа да започва с най-малките.

Не е препоръчително да се използва сравнение за равенство на реални числа.

Спазвайки тези препоръки, освен постигане на по-добър стил, в много случаи се предпазваме от грешки при изчисленията.

4. Ниво Анализиране.

*Правилно използване на коментари.* Коментарите трябва да пояснят само неща, които не са очевидни и да помагат на програмиста да се ориентира в алгоритъма и в действието на програмата. Когато се проверява дадена програма, се преминава от ролята на съставител към тази на читател на програмата. В тази ситуация стилно написаният код е по-лесен за четене.

**Област на действие на обектите.** Опасността от локални декларации в тялото на функция е още по-голяма. В *C++* променливата може да се декларира/дефинира като глобална, нелокална или локална променлива във функция. *C++* позволява на локална променлива да се даде същото име както на нелокална. Опасността се дължи на покриването на нелокалната от локалната променлива. Това може да доведе до грешни резултати, които трудно се откриват.

Разбиване на задачата на подзадачи. Прилага се структурна (функционална) декомпозиция за разделяне проблема на части. Избират се и се използват подходящи структури от данни, отделните групи от изчислителни стъпки се разделят в отделни функции.

Най-важната идея на структурното програмиране е идеята за модулност. Всъщност става дума за способността да се разложи сложен проблем на по-прости компоненти. В този случай, задачата се разделя на информационно затворени части (блокове), които имат самостоятелно значение, и след съставяне на първоначалната схема, свързваща частите на задачата, се прави детайлизация на отделните компоненти. Принципът на модулност, който в действителност е мисловен подход, има голямо общообразователно и възпитателно значение.

5. Ниво Оценяване

**Подбор на подходящи тестови примери.** При подбор на тестови примери математическите знания позволяват правилно подбиране на тестови данни и лека проверка за коректност на резултата.

Изнасяне на операции извън цикъла и включване в инварианта на цикъла. Реализацията на циклични процеси е фундаментална тема в програмирането. Генезисът на тези процеси е многократното изчисление. Трябва обаче да се избягва преизчисляване на един и същ израз (между впрочем валидно и в математиката).

Стилната програма влияе съществено на ефективността на програмата. Чрез изменение на представянето на данните и алтернативни алгоритми се прави сравнение на програмите по време и памет**.**

Освен за подреденост и “красота” на програмния текст, стилът на програмиране има роля и за проясняване на идеите на решението, което е особено важно за обучението. В стилната програма лесно се виждат основните елементи на решението.

6. Ниво Създаване.

Избягване на излизане извън границите на допустимите стойности. При разглеждане на управляващите конструкции най-важният елемент е внимателното конструиране на условията (булеви изрази) и правилното им използване. Когато в едно условие се съдържа поне една логическа операция,

то се нарича съставно условие. Понякога подреждането има само препоръчителен характер, но в други непрецизното подреждане може да доведе до грешки [11].

Неправилното подреждане може да доведе до излизане извън границите на индекса на масив или дефиниционното множество на функция. А това може да предизвика обработка на елементи извън допустимите граници и получаване на грешки при изпълнение или още по-лошо на грешни резултати.

Параметри на функции. Трябва внимателно да се проверява съответствието между формални и фактически параметри при писане на програма и това да се прави преди тестването. Да се ограничава броя на параметрите, предавани по референция**.**

От написаното могат да се направят следните изводи:

✓ Различните стилови елементи могат да се представят на различни етапи от обучението по програмиране. Оказва се, че елементи, касаещи оформянето на кода, се появяват на високите нива на пирамидата (напр. коментарите). От друга страна, елементи, касаещи правилността и ефективността (напр. особености на машинната аритметика), могат да се представят в първите часове от изучаването на програмирането и то като императивни правила в долните нива на пирамидата.

✓ Елементите на стила от трите групи са разположени на различни нива в пирамидата на Блум. Следователно обучението в стил може да бъде непрекъснато.

✓ С прости програмни средства, без отклоняване от основната цел (обучението по програмиране), може да се покаже на студентите ползата от стила на програмата.

Това дава утвърдителен отговор на поставения в края на въведението въпрос, т.е. едновременно с обучението по програмиране може да се усвоява и стил на програмиране.

**Заключение**

Обучението по информатика изисква разработване на методики, които да осигурят не само възпроизвеждане на голям обем от знания, но преди всичко формиране и развитие у обучаемите на способности да овладяват тези знания, да изграждат умения за самостоятелно придобиване на нови знания и критическото им осмисляне.

Един методичен подход за изграждане на умение за критично мислене в обучението по програмиране е чрез реализиране на различни начини и подходи при решаване на конкретна задача. Целенасоченото трениране на

критично мислене позволява на обучаемите да преодолеят репродуктивното ниво на осмисляне на учебния материал и дава възможност за по-дълбоко проникване в същността на възникващите пред тях проблеми и нестандартни подходи за решение на задачи.

**Благодарности**

Работата е финансирана по проект РД 08-257/14.03.2013 на Шуменския университет „Епископ К.Преславски”.

**Литература**

1. Ван Тассел, Д. Стиль, разработка, эффективность, отладка и испытание программ, Пер. с англ. — 2-е изд., испр., — М., Мир, 1985.—332 с, ил.
2. Генджова, А. Мисловни умения на високо и ниско ниво, заложени в български и канадски учебни програми, Chemistry: Bulgarian Journal of Science Education, Volume 21, Number 1, 2012, 60-70
3. Лапчик М. П. и др. Методика преподавания информатики, Москва, Академия, 2001.
4. Роганов, Е.А. Основы информатики и программирования, М., МГИУ, 2001
5. Amer A. Reflections on Bloom‘s revised taxonomy, *Electronic J. Res. Educ. Psychology, 4*, 2006, 213 – 230.
6. Anderson, L.W, Krathwohl, D.R, Airasian, P.W, Cruikshank, K.A, Mayer, R.E., Pintrich, P.R, Raths, J. & Wittrock, M.C. *A taxonomy for learning teaching,* Boston, Allyn & Bacon, 2000.
7. Bloom, B., Engelhart, M., Furst, E., Hill, W. & Krathwohl, D. Taxonomy of educational objectives: the classification of educational goals, Handbook I: Cognitive domain, NewYork: David McKay, 1956.
8. Krathwohl, D. A revision of Bloom’s taxonomy: an overview, *Theory into Practice*, *41*, 2002, 212-218.
9. Kernighan, B. W., Pike, R.: The Practice of Programming. Addison-Wesley, 1999.
10. Resnick, L. *Education and learning to think*. Washington, National Academy Press, 1987.
11. Teodosiev T., Nachev A. Some Pitfalls in Introductory Programming Courses, International Journal “Informatics in Education”, 2012, Vol. 11, No. 2, 241–255
12. http://www.tuj.asenevtsi.com/EL09/EL31.htm (последно посетена 29.04.2013)

## BLOOM'S TAXONOMY AND TRAINING IN PROGRAMMING STYLE

***Teodosi Teodosiev***

***Abstract:*** *The presented work is using Bloom's taxonomy to set the goals of teaching programming. Here are shown the elements of programming style, in which you can teach novices. Elements of programming style are at different levels of Bloom's pyramid.*