СРАВНИТЕЛЕН АНАЛИЗ НА ЕЗИЦИ ЗА ПРОГРАМИРАНЕ*

ТЕОДОСИ К. ТЕОДОСИЕВ

COMPARATIVE ANALYSIS OF PROGRAMMING LANGUAGES

TEODOSI K. TEODOSIEV

ABSTRACT: *The current paper comments on the requirements of programming languages and their usage. Furthermore, the complexity of choosing a programming language is justified. The article presents a comparative analysis of the most popular programming languages as well. The paper elaborates on the different criteria in choosing a language for the business and university.*

KEYWORDS: *Programming Language, Introduction to Programming, Comparative Analysis, Teaching.*

1. Въведение. Процесът, посредством който организациите и хората възприемат тенденциите в развитието на технологиите, е сложен, защото включва в себе си множество фактори. Тази обща констатация се отнася в частност и до езиците за програмиране, където много внимателно разработени езици, имащи превъзходни технически атрибути, не се разпространяват широко за разлика от езици, започващи със скромни амбиции и ограничени възможности, които постигат широко приложение в промишлеността и научните кръгове.

В предложената работа са коментирани изискванията към езика за програмиране и неговото използване. Обоснована е сложността на избора на език за програмиране. Направен е кратък сравнителен анализ на най-популярните езици за програмиране. Коментирани са различните критерии при избор на език за програмиране в бизнеса и университетите.

2. Дефиниции и характеристики. Познатият термин "език" придобива нов смисъл в съзнанието ни във връзка с програмирането. Около този термин се изгражда цяла система от научни понятия. Понятието “език” е едно от най-важните системообразуващи понятия в курса по информатика.

Алгоритмите, приведени във вид годен за автоматичното им изпълнение се наричат програми, а формалните средства, използвани за записа на програмата, се наричат „езици за програмиране”. Език за програмиране (ЕП) се нарича всяка система от означения, годна за описание на алгоритми и структури от данни.

Нотацията като набор от изразни конструктивни средства – това са елементите, от които се изгражда програмния текст. Също както естествените езици, всеки ЕП има свои синтаксис и семантика, но тук те са постулирани пределно точно. Синтаксисът определя правилата на записа на отделните конструкции, а семантиката изразява значението (смисъла) на обектите на езика и отразява правилата за композиция на обектите. Програмата, правилно написана от гледна точка на ЕП, трябва да удовлетворява както синтактичните, така и семантичните правила на езика.

При разработката на ЕП се ръководим на първо място от това ”какво може машината”. Впрочем, отчитайки, че ЕП е мост между потребителя и машината и

* Работата е частично финансирана по договор РД-08-98/05.02.2016 към Шуменския университет

фактически може да се разглежда като негов инструмент, изглежда много важно да се разглежда и „какво Човек може да измисли". Заради това самата нотация трябва да бъде от най-високо качество. Нотацията трябва да способства максимално за „техническото" програмиране, да помага за развитие на добър стил. При неудачна нотация много внимание се отделя на „подводните камъни" на нотацията, т.е. нейната нерегулярност и дефекти на дизайна, така че начините за избягването им излизат на преден план и затъмняват съдържателния аспект на програмирането. Добрата нотация е толкова важна, че такъв безспорен авторитет в света на информатиката като Н. Уирт, специално създава собствени езици, за да осигури ефективност и мотивираност на курса по програмиране.

Възможността за използване на ЕП като средство за взаимодействие със съществуващите автоматични изчислителни системи дълго време се разглежда като най-важното им свойство. Ефективността, с която се изпълняват програмите, стана главен критерий за качествата на ЕП. Вследствие от това, недостатъци на машините старателно се възпроизвеждат в ЕП, при това в ущърб на интелектуалната управляемост на програмата.

Най-важно, но в същото време и най-незабележимо свойство на всеки инструмент, се явява неговото влияние върху формирането на навиците на хората, които го ползват. Когато този инструмент е ЕП, неговото влияние, независимо от нашето желание, се отразява на начина ни на мислене. И както отбелязва Дейкстра в [1] "...формализмите, които използваме, формират нашия начин на мислене в добра или лоша посока, а това значи, че трябва да сме много внимателни в избора на това какво да учим, защото после да отвикнем е невъзможно".

Общоприета е гледната точка, че основна цел на ЕП се явява възможността на програмиста да формулира своите мисли в термините на абстракции, съответстващи на решаваната задача, а не в термините на възможностите на апаратните средства [2].

ЕП е главен инструмент на програмиста за написване на програми. От една страна ЕП оказва силно влияние върху мисленето на програмиста, а от друга – трябва да осигури ефективно изображение в машинен код. ЕП играе роля на мост над пропастта, разделяща човешкото мислене и начина на работа на компютъра.

Ще цитираме три мнения за избора на ЕП. Те са показателни за залитанията и сложността на този избор.

Известният информатик Е. Дейкстра коментира следното в [3]: "... инструментите, които използваме, и езика и означенията, с които изразяваме или записваме нашите мисли, се явяват основен фактор, който определя за какво можем да мислим и какво можем да изразим. Анализът на влиянието, което ЕП оказва на потребителите си, и признанието на факта, че в настоящето време мощта на нашия мозък се явява най-недостигащия ресурс, съвместно дават нов набор от еталони за сравнение на относителните достойнства на различните езици за програмиране. ... Езиците за програмиране на утрешния ден ще се отличават значително от днешните, те ще ни подтикват към отразяване в структура на това, което пишем, всички абстракции, необходими за концептуалния обхват на сложността на предмета на нашата разработка".

Защо съображение от типа, че даден език е „широко използван в практиката" и т.н не е добър аргумент? Ограничаваме се с мнението на Джоел Сполски „...Това не са професионални училища! Подготовката на хората за работа в бизнеса не е тяхна задача... Университетите трябва да дават на студентите фундаменталните средства, за да преживеят живота си, а не да ги готви за първите две седмици на работното им място..." [4].

Още аргументи в тази насока може да се намерят в [5]: "За да задържат студентите в колежа, колежите (университетите) обичат да преподават популярни езици. Това кара студентите да мислят, че те изучават правилното нещо, което в крайна сметка се използва...".

Прави впечатление, че неизменно се посочват като съществени критерии за избора на ЕП простотата на синтаксиса, удобството за четене и близост до матерния език.

Това, което цени начинаещият програмист в ЕП – достъпност, яснота и гаранции от грешки – на системния програмист му се струва излишно и даже го дразни.

3. Сравнителен анализ на езиците за програмиране. След казаното дотук ще направим сравнителен анализ на най-популярните езици за програмиране (C/C++, ADA, Java, Python и др.) В този анализ езикът C/C++ е използван като сравнителна единица при съпоставка с другите езици за програмиране, което е признание за мирогледната значимост и широкото му приложение от една страна, а от друга поради господстващото му място като ЕП в обучението в настоящия момент.

В средата на 70-те години на XX век се създава и впоследствие широко се разпространява в различни варианти (в т.ч. знаменития Linux) много мощната операционна система Unix. На „неин гръб" в света на програмирането получава разпространение езика C, използван за нейното написване. Езикът C бил съчинен (именно съчинен, тъй като това не може да се нарече проектиране) от безхитростно проинтегрирани практически съображения като замяна на Асемблер в написването на ОС Unix. Разработен за писане на операционни системи, компилатори и други инструменти на системата, езикът C става почти напълно доминиращ.

По-нататък на негова основа без отстраняване на многобройните му дефекти се построява C++ по пътя на добавяне към C на модните конструкции от обектно-ориентираното програмиране (за недостатъците на C++ са написани цели трактати [6]).

Разбира се проблемите на C/C++ се усещат и в индустрията. Именно затова възниква Java, представляващ сам по себе си опит да се направи това, което не успяха авторите на C/C++, а именно не просто безсмислено добавяне на модни средства върху ненадеждната основа на C, а да се направи обмислен подбор, а също и препроектиране на „основите" на езика, изключвайки коварните капани, с които е знаменит C. Строгата типизация, автоматичното управление на паметта, отказа от множественото наследяване – фактически, това е отказ от главните култови свойства на C, правещи го толкова опасен („пълната свобода на програмиста, включваща свободата да прави всякакви грешки"). Развитието от C към C++ говори за ориентация към ограничаване на голямата свобода. Експериментите показват, че ЕП със силна типизация способства за увеличаване на яснотата и надеждността на програмите.

В [7] авторът прави изводи: "C и Java са важни за понятията, които въплъщават, за перспективата за професионална реализация и за класовете задачи, които решават. Студентите трябва да познават основите на тези езици. Те обаче не са добри за преподаване на студенти, недобре запознати с програмирането."

Привърженици на езика ADA твърдят, че "ADA има няколко езикови особености, които правят правилен нашия избор за първи език. Силното типизиране е много изгодно и за неспециалисти, както и за специалистите. Силната типизация позволява откриване на програмните грешки още при компилация вместо по време на изпълнение. И това ни води до по-бърз успех. На слабо типизираните езици компилацията е по-лека, но грешките се търсят трудно при изпълнение" [8].

Авторът в [9] се съгласява, че ADA превъзхожда Java, защото избягва объркващия синтаксис. Поддръжката от ADA на подтипове и изброими типове много превъзхожда

Java, C++, или C#. Изброимите типове в ADA могат да се използват като индекси на масиви за разлика от Java, C++ и C#.

Накрая ADA поддържа императивната и обектно-ориентираната (OO) парадигми, което е съществено за преподаване на "Увод в програмирането". Java изисква от обучаемите, които са слабо запознати с програмирането да изучат основите на обектно-ориентирания синтаксис едновременно с изучаването на основите на информатиката. (Виж например [10, 11]).

В [12] авторът мотивира изборът на Java: "В нашия университет ние имахме привърженици и на трите езика VB, Java и C. Избрахме Java заради господството му в Web приложенията, по-малката сложност и лекота за проверка (debugging) от C, и синтаксис основан на C/C++, облекчаващ студентите, които вече са запознати със C".

Java е най-популярният ЕП в света и има много работни места за Java програмисти. Обширната библиотека класове създава чувство, че за всяко нещо си има клас и голяма част от „програмирането в Java" се състои в търсенето на правилния клас. Пак там [12] обаче авторът прави и следния коментар. "Даже след двегодишен опит, виждам, че не мога да направя много без справка с документацията".

Java наследява голям брой конструкции от C++, но добавя и голяма библиотека от системни обекти, които трябва да се научат добре. Това нарушава краткостта и яснотата и не позволява избягване на излишните езикови конструкции.

Строгата структурираност на езика може да пречи на обучението и преподаването. В това отношение Java не се отличава от другите езици за програмиране, основани на компилация с изискването на строга проверка на типа. Студентите, имащи опит със scripting езици, където има нетипизирани променливи и по-малко синтактични правила, се дразнят от това.

Успехът на маркетинга на Java привлича достатъчно разработчици, за да може да се проявят преимуществата на Java в производителността, заради отказа от култовата свобода на C. След Java се появява и „близнака" ѝ C#. Но тези два езика, заради конкурентната война, непрекъснато се усложняват, което прави почти невъзможна преносимостта, заради привързването към определена платформа. Непригодни за общите училищни курсове са и Java и C#, заради своя C-образен синтаксис; направени са прекалено сложни по тактически маркетингови съображения.

В Oberon (наследник на Pascal) се постига точният синтез на „старите" достижения на структурното и модулното програмиране с „новите" обектни методи.

Може да се говори за формиране под влиянието на Oberon на стандартната парадигма на процедурното програмиране (впрочем Oberon излиза извън рамката на класическото процедурно програмиране, заимствайки най-ценните характеристики на функционалните езици за програмиране). В орбитата на тази парадигма се оказват също Java и C#, лежащи в основата на проектите Java и .NET.

Python поддържа много програмни стилове – от много прости до обектно-ориентирани методи. Python е интерактивен език от високо ниво с динамична семантика. Python има много добри качества: предписва добър програмен стил; достъпен обектно-ориентиран подход, но не задължителен; използване на изключения, но не задължително; нито игрови, нито академичен език – много реални приложения са написани на Python; Python позволява концентрация върху алгоритъма и задачата, а не на езиковите особености и недостатъци; безопасен – има динамична проверка на типа и проверка на границите на масива.

Вградените структури от данни в съчетание с динамичната типизация и динамичното свързване правят Python много привлекателен за бърза разработка на

приложения (Rapid Application Development). Тъй като няма компилация, цикъла редактиране-тестване-проверка протича невероятно бързо. Python е интерпретатируем език и това го прави по-бавен. Динамичната проверка на границите, динамичният изход и други умни неща в Python го забавят още повече. Python може да бъде на порядък по-бавен от еквивалентен код на C, но изисква много по-малко време за разработка. Python–програмите са типично 5-10 пъти по-къси от кода на C++! (3-5 пъти по-къс код от еквивалентния на Java). Тази разлика може да се обясни с вградените типове данни на високо ниво и динамичната типизация в Python.

В [13] авторът заключава: "Java и C са обичайно използваните езици и те са съответни за повечето дисциплини, но не и (аз вярвам в това) за абсолютно начинаещи. Аз смятам, че Python е много по-добър избор за новаци."

4. Използване на ЕП в бизнеса и университетите. Компанията TIOBE Software (http://www.tiobe.com) предлага една от най-пълните и най-актуални проучвания за използването на езиците за програмиране. Това проучване се базира на използването на онлайн ресурси за оценка на използването на езиците за програмиране в индустриалната практика и се обновява на ежемесечна база. В този рейтинг през последните десет години начело са Java, C, C ++, C#, Python и PHP.

Ако се опрем и на други проучвания и се опитаме да разделим използването на езиците за програмиране в бизнеса и в научните среди, могат да се направят следните констатации.

C, Java и C ++ са сред първите 4 ЕП както в бизнеса така и в научните среди. Но докато C заема #1 в бизнеса (може би заради остаряло програмно осигуряване), то е на 4-то място в научните среди.

Учебните заведения трябва да предоставят повече свобода за смяна на езиците за програмиране в сравнение с бизнеса. Разпределението на ЕП в научните среди е по-еднородно (Java заема първо място с огромните над 44%), в същото време C заема първо място в бизнеса само 17% [14].

Друг интересен случай е Python. В университетите неговото разпространение е почти колкото на C и Java, докато в бизнеса значително им отстъпва.

За по-точна представа кои ЕП получават по-широко разпространение в университетите и кои губят позиции, може да се проследи колко университета са сменили ЕП за курса по увод в програмирането, което показва, че Python има най-положителна динамика, независимо, че има по-ниско разпространение.

За разлика от уводния курс по програмиране, в който вниманието е насочено към преподаване на дисциплината на програмиране, уводния компютърен курс (за университетите, които го предлагат, например в САЩ [14]) запознава студентите с широка гама от изчислителни теми и като правило се използва за следващите курсове по информатика. Езиците за програмиране за тези курсове се избират, съобразявайки се с много изисквания като удобство, лекота за обучение и простота за използване и донякъде релевантност с бизнеса. Най-използвани са MatLab, Python, Visual Basic, Scratch. Най-популярните и използвани езици за програмиране (Java, C++, C) са доста по-назад. Затова не е чудно, че малко университети използват един и същ ЕП за уводния компютърен курс и за увод в програмирането.

За уводния курс по програмиране се използват някои езици за програмиране, които не са популярни в бизнеса. Основание за използване на тези езици е това, че при обучението искаме езика да поддържа дисциплината на програмиране, а когато студентите усвоят строга дисциплина, преходът към друг език е проста работа [15].

Сред първите десет езика намираме и Пролог (#2), който заема доста висока позиция, независимо че използването му е доста ограничено в бизнеса [14]. Този език се използва в качеството на средство за обсъждане на логиката на програмирането. Друго впечатляващо наблюдение е групата на функционалните езици (Scheme, Haskell, ML, Lisp, OCaml и др), която заема място в първата десетка на рейтинга и подържа практиката на функционалното програмиране.

Интересен е въпросът за взаимното влияние между бизнеса и университетите при избора на език за програмиране. Бизнесът може да упражни водеща роля, като изисква от университетите да подготвят студентите за пазара на труда, обратно университетите могат да определят езика за програмиране, произвеждайки поколение студенти, владеещи този език, който с времето се приема и от бизнеса.

Може да се коментира следния въпрос: защо бизнесът продължава да използва езици за програмиране от края на шестдесетте/началото на седемдесетте години на миналия век (C), а също и техните наследници (C ++, Java) за сметка на съвременните езици за програмиране, които представят съвременни идеи за дизайна, имат интересни атрибути, като поддържане на модулност, обработка на изключенията, скриване на информация и др. Отговорът може да се търси в две ортогонални повърхности:

Първо, вътрешните атрибути на програмните артефакти играят незначителна роля при избора на ЕП за сметка на външните фактори, отнасящи се до обстоятелствата, при които възникват артефактите. Как иначе може да се обясни, че език като C, разработен от двама системни програмисти, за да им помогне в разработката на операционна система (Unix), достига световен успех и така оказва влияние на много от следващите езици за програмиране.

Второ, приемането на ЕП в бизнеса зависи от много ограничения, които не се прилагат в научните среди. Към тях се отнасят:

✓ Цената на обучението на програмистите и аналитиците на нов ЕП заедно с нови среди за програмиране и програмно осигуряване.

✓ Загубите свързани с по-ниската производителност на труда и намаляване на качеството на продукцията в резултат на въвеждането на новия ЕП, докато се достигне до овладяване на този език.

✓ Необходимост от поддържане на квалифицирани сътрудници за езиците, които се използват за унаследеното програмно осигуряване.

В [16] авторите стигат до извода, че "съществуващия код, наличния опит и библиотеките с отворен код са основен фактор за избора на ЕП ". Интересно е, че те считат, че опитните програмисти са по-отворени към приемането на нови езици за програмиране, отколкото младите програмисти, при условие, че студентите в университетите са по-млади от средната възраст в програмисткато съсловие. Още повече, че натрупания опит е по-голям сдържащ фактор в бизнеса, отколкото в университета. В противовес на това буди недоумение подобен избор в научните среди в светлината на следните наблюдения:

✓ Тези езици, особено C са неподходящи за начално обучение на програмиста: те са много сложни, има много изключения и са зависими от конкретната реализация, за да служат за преподаване на дисциплината на програмиране.

✓ Тъй като разработчиците изучават нови езици често и бързо, студентът може да се научи да програмира на един език и в бъдещата си практика да разработва програмно осигуряване на друг език с минимални усилия. Ето защо университетите не са длъжни да се съобразяват с бизнеса, а да осъществяват изборът си по собствени критерии.

5. Заключение. От изложеното могат да се направят следните изводи:

Може да се твърди, че академичните среди, взимащи решения, са длъжни да поемат водещата роля в определяне на насоката на програмисткото образование посредством разумен избор на езици за програмиране, които са предназначени за тази цел, които помагат на студента да си създаде ясна дисциплина на програмиране и които ще помогнат в крайна сметка да се повиши нивото на програмното осигуряване.

Изучаването на различни езици – толкова колкото е възможно – е добра стратегия. Помага на студентите да се абстрахират от избрания език и да мислят повече за това какъв е проблемът и как да се представи решението. Надяваме се, че учейки студентите да мислят за проблема извън ЕП, ги учим и как да учат нови езици за програмиране.

Когато можете да програмирате на един език е сравнително лесно да научите друг за няколко дни. Така че за уводния курс по програмиране е важно да се накарат студентите да създават програми.

Първите стъпки в нова дейност трябва да се правят внимателно с прости, евтини и леки за използване помощни средства, които да ни предпазват от грешки. Има ли смисъл на дете да се купи "Ферари"? Не, защото от една страна това е много скъпа играчка, а от друга – детето не може да се възползва в пълен обем от мощта и възможностите му.

ЛИТЕРАТУРА

1. Дейкстра Э, Ответы на вопросы студентов отделения программного обеспечения, 2000
2. Wirth N., Programming Languages: What to Demand and How to Assess Them, Software Engineering, ed. by R. H. Perrott, Academic Press, New York, 1977
3. Дейкстра Э, Смиренный программист, Commun. ACM 15, 1972
4. Spolsky J. (2005). Making Wrong Code Look Wrong. Joel on Software. http://www.joelonsoftware.com/articles/Wrong.html (accessed 30/07/16)
5. Programming and Teaching CS http://foo.ewu.edu/jefu/other/essays/ language.html (accessed 1/04/2009)
6. Joyner I., C++?? A C++ Critique, The Modulator, Nr. 10&11/Nov&Dec-1992. http://www.modulaware.com/mdlt28.htm (accessed 30/07/16)
7. A Very Quick Comparison of Popular Languages for Teaching Computer Programming, 2006 http://www.ariel.com.au/a/teaching-programming.html (accessed 30/07/16)
8. Sward R. E., Carlisle M. C., Fagin B. S., and Gibson D. S., The Case for Ada at the USAF Academy, Department of Computer Science 2354 Fairchild Dr, Suite 6G101 USAF Academy, CO 80840 1-719-333-3590
9. Brosgol, Benjamin M. A Comparison of Ada and Java as a Foundation Teaching Language, Ada Core Technologies, http://ftp.informatik.rwth-aachen.de/ftp/pub/mirror/cs.nyu.edu/pub/gnat/jgnat/papers/ada-java-teaching-comparison.pdf (accessed 30/07/16)
10. Reges, S., Conservatively Radical Java in CS1. In Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education, Austin, TX, March 2000, pp 85-89
11. Hristova, M. et al, Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. In Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education, Reno, Nevada, February 2003, pp 153-156. 36
12. Pendergast, M., Teaching Introductory Programming to IS Students: Java Problems and Pitfalls, 2006.
13. Elkner Jeffrey, Using Python in a High School Computer Science Program http://ftp.python.org/workshops/2000-01/proceedings/papers/elkner/elkner.html (accessed 30/07/16)
14. Latifa BEN ARFA RABAI, Barry COHEN, Ali MILI, Programming Language Use in US Academia and Industry , Informatics in education, Vol 14, Num 2, 2015, 143-160
15. Yi, P., Li, F., Mili, A., Modeling the evolution of operating systems: an empirical study. *Journal of Systems and Software*, 80(1): 1–15, 2007.
16. Meyerovitch, L.A., Rabkin, A. (2013). Empirical analysis of programming language adoption. In: OOPSLA'13, Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications. ACM, New York, 1–18.