*Proceedings of the 9th International Conference on Applied Informatics*
*Eger, Hungary, January 29–February 1, 2014. Vol. 2. pp. 75–84*
doi: 10.14794/ICAI.9.2014.2.75

# Evaluation of development environments from educational perspective

**Péter Szlávi, Viktória Heizlerné Bakonyi**

ELTE IK
szlavip@elte.hu
hbv@inf.elte.hu

**Abstract**

In this paper we want to examine the tools we need for writing a program, from the simplest editors to the professional integrated development environments (IDE). We do not say that we have found the "philosopher's stone;" thus, you must not expect us to name a given IDE, which is perfect from all viewpoints. Instead, we would like to sketch a system of viewpoints, putting two questions in focus: which features are preferable on different levels of education and what can be the didactic result of choosing one or the other.

*Keywords:* Programming languages, Development Environments, Programming methodology, Elementary and Secondary School

*MSC:* 68U99

## 1. Introduction

### 1.1. Why is this viewpoint interesting?

In Hungary, the areas of literacy that have to be developed in public education are defined by NAT (National Educational Plan). To provide an approximate description about how it works, NAT lists the main competencies students have to reach at the end of their studies in the given each area. NAT, however, does not specify what to teach and how to teach it. We can say that it only provides frame-curricula, which have to be filled with content locally, by schools, or rather, by the teachers. [1] Since informatics knowledge changes quickly, [2] informatics teachers have to make newer and newer curricula. When creating a new curriculum, the teacher needs to consider a lot of things in order not to fall into certain traps; for example, introducing too much new info the first steps of learning, or encountering difficulties in managing tools, just to name a few. That is why it is very important to know features which can help assess the learnability and usability of a new learning

content. A bad choice may ruin the whole learning process, and the correction of it may take too much time. Hungarian informatics teachers especially have to make use of the available time as efficiently as they can, [3] because most of the students learn only 1-2 hours of informatics per week, finishing their IT studies at the age of sixteen. This situation is very unfortunate, because the ability of abstraction evolve only at the end of puberty (18-19 year). [9] some of the students, however, can learn the subject in special courses, which means 2 extra hours per week in the last two years. Finally, we do have to mention that there exist some schools where, where pupils can learn more than 10 hours of informatics per week. Taking into consideration all the above cases, it must be clear that we cannot work with the same content and same expectations.

We do not want to focus on each sub-area of informatics knowledge (like information society, application usage, etc.); instead, we will only consider the issue of problem solving, which is one of the most complex processes we encounter in education. If one wants to solve a problem, one must understand it in detail (by making a specification), create an algorithm (by using an algorithmic language like structograms), and finally write the code (by knowing at least one programming language [4] and an editor, debugging tool, etc.). At this point, we would like to bring into focus the issue of development environments (DE[1]), which can help our code-writing, or in some other cases, they can make our work more difficult. At first sight, the question of choosing a DE does not seem too interesting in the learning process of programming. Similarly, one might find a wheel in a car as something not too exciting, but who can use a car without a wheel? Having pointed out the importance of DEs, we hope that the proposed viewpoint system of evaluating DEs will help teachers' decision about which development environment to use.

# 2. Development Environments

## 2.1. Classification of development environments

In public education, we have to form a general picture about DEs. To give an analogy, when we introduce a specific text editor, we make sure to speak not just about the specific text editor, but to give an overview to editing in general. It is, thus, very important to be informed what services we can expect from that specific DE we chose, during code editing, executing, testing, or debugging. We must be conscious that there is a wide range of DEs, from the quite simple to the professional ones. One DE can be a simple console-like editor; another may have graphical user interface (GUI); or it might be able to build up code by dragging controls (4GL). We need to keep in mind that by using the most simple console editor we can make the same applications as with the help of a 4GL editor, which only presupposes a bit more work and a deeper knowledge from the developer. Some of the professionals, as a consequence, prefer simple editors instead of the

[1] We will use the expression development environment for integrated development environments too.

"polymath" ones, where the development of the tiniest parts can be held in their own hands.

In the following, we are going to classify DEs by their functionalities, also providing some examples for each class (just to show what we are thinking of):

- Simple editors (e.g. NotePad, Edit, vi)
- Code-editors (e.g. NotePad++)
- Special editors (e.g. online Bootstrap editor, Scratch, PWF)
- Development environments (e.g. Geany, Emacs)
- Integrated development environments (e.g. Visual Studio, Eclipse)

A simple editor offers the functionality of creating documents like copying, deleting, searching, etc. What are the advantages of using them for code-writing? There is no need to install new applications; with it we can spare resources. If the students already used them before, we do not have to waste time on explaining their usage. Usually they are not too complicated, but there are no warnings and error messages, which requires pupils to work punctually. (To teach punctuality, a lot of school teachers use NotePad for editing HTML codes.) Since the students have to write down each character of the code, they may be able to remember better the key-words, the structure, and the "philosophy" of the programming language, through the constant repetition. But this can be the disadvantage of their usage as well. If the students already understand the basis of programming, such an editor can slow down their working process.

A code-editor will help us while typing the code, by highlighting the key-words or signaling code completion. But in this case we have to use a single compiler and debugger for working, which is not too convenient. The advantage of these code-editors is that their usage is not too difficult. Our personal opinion is that, if there is enough time for it, we should show the process of creating a program from a code step by step, from writing a code, through compiling it, to executing it at least once. The students would better understand what happens in the background; they do not have to imagine it as a mystification.

Into this group we categorize some "strange" editors. With the help of these editors, we can create a code without knowing the syntax of the programming language/code in question. For example, by using online Bootstrap editor you can create CSS files without the knowledge of CSS syntax. In the case of Scratch, you can make a program by dragging some structures to the programming area and you do not have to write any code. The advantage here is to teach only algorithmic thinking without using a programming language. We have to mention PFW, with which one can edit "algorithm", which not only creates some code but documentation as well. [5]

Finally, let us examine DEs (development environment) and IDEs (integrated development environment) together. There is no sharp borderline between them, but we can agree that DEs have less functionality. It must be added, though, that

there are great differences in functionalities among IDEs too. It is a fact that the more functionality is built in, the more difficult it is to use. It is not at all certain that the most professional one is the best choice in all cases! Everything depends on the age-group of the students, their fields of interest, and the available time-frame of the classes.

Another viewpoint of classification should be whether DEs have built-in compiler/interpreter or not. They can be, as we stated before, simple console-like editors, or more complex ones, say, with GUI or 4GL.

# 3. Evaluation of Development Environments [8]

## 3.1. General expectation towards development environments

**Human expectations:** First of all, it is vital that the difficulty of managing the chosen DE must fit the knowledge of the age-group. If it is too complex, it may also be too difficult to explain, which can easily discourage students from learning. Another important viewpoint is that the software must be free (like CodeBlocks or Visual Studio Expression). If a student wants to use it at home, it is an advantage if he or she does not have to buy it. In special classes where several programming languages are taught, it can be useful if the same DE can be applied for all of them. (Here is a list of some of the DEs and the assigned PLs: Geany – Pascal, PHP, Java, Python...; Visual Studio-C++, C#, BASIC...; CodeBlocks – C, C++.) Moreover, it is a good point if the DE enables the creation of applications to several platforms (console, graphical, web, mobile). If the above-mentioned expectations are fulfilled, you still need something more. You need documentation or books as well, in order to be up-to-date about the details.

**Computer expectations:** In special cases, the computer resources are not adequate for the usage of certain DEs, usually the most complex and professional ones. If so, the teacher faces the dilemma to choose between possibilities and wishes. It is a benefit if the DE can run on different operating systems, which enables students to make their homework at home on Linux or on Windows as well (e.g. CodeBlocks). A negative example can be Lazarus, which supposedly works both on Windows and Linux, but in reality it is uncertain on Linux. It must ne added that there is also a possibility to work independently from platforms, by using some DEs on the web through a browser like Cloud9 (for HTML, JavaScript, and PHP) or WebSharper (for F#).

**Further expectations:** Secondary school teachers have a role in helping students plan their future career. Firstly, they need to prepare students for passing their matriculation exam, [6] and secondly, they ought to help students to excel in vocational trainings too. Therefore, they have to give a standard for improvement, which is why it is important to use a DE with which we can edit projects containing several files. Besides, teachers need to be up-to-date and know which DEs are

used in current higher education (usually professional ones like Eclipse, NetBeans, Visual Studio, etc.).

## 3.2. Functionality

### 3.2.1. Creating code

The usual functions in DEs are code editing, compiling (+building)[2], executing, debugging, built-in test possibilities, refactorization, or even generating documentation.

Below, we collected the most frequent functionalities of DEs. Naturally, teachers themselves have to make a decision which ones are useful or necessary for their specific work, or which ones can disturb their students with their complexity. (Referring back to a former remark of ours in the introduction, the teacher has to be aware of the students' level of abstraction.)

Functionalities in connection with creating codes are:

- Typing the code
  - automatic syntax bubbles (CodeBlocks)
  - code completion (PowerShell ISE)
  - code snippets (Visual Studio)
  - automatic code generation (VS Ultimate (installing Architecture and Modelling Tools), BootStrap on-line editor)
- Helping to orient ourselves in the code
  - highlighting (NotePad++)
  - helping (static, position sensitive, frequency)
  - ToDo list
  - code lensing, code mapping (Visual Studio)
- Helping the readability of the code
  - indention (LCN Logo, CodeBlocks, ELAN, Visual Studio )
  - refactorization (Eclipse, Visual Studio)
- Team work
  - versioning, message sending

[2]In this article we mention compilers for the sake of simplicity, but we have to note that there are DEs for interpreted languages as well.

In the first group we listed the functionalities that regard the typing of the code. Automatic syntax bubbles or the usage of so-called IntelliSense/content-assist are both very useful in public education. They help students remember the details of the functions, their signature, and their short explanations. In some DEs the IntelliSense offers the most often used possibilities at first (VS). Code-completion speeds up the typing process, which is why it ensures efficient work. Modern, complex DEs usually offer code snippets as well, but this functionality can be neglected in general classes, because its usefulness reveals itself only when the code is long and consists of repeating parts. This function belongs more to professional usage in our opinion. Finally, we have to mention automatic code generation in two different ways. Professional IDEs usually support code creation from UML, or test lines (e.g. "lorem ipsum") by using hot-keys. Next to this, in certain cases we can also speak about code generation as well. The editor does not show the generated code, which means you can use the drag and drop technique, for example, without actually knowing the syntax of the language (e.g. BootStrap on-line editor).

In the next category we grouped the functionalities that either assist in finding and selecting the right instructions and functions, or help in orienting ourselves in the program code. The simplest and most useful functionality is code highlighting, which shows the developer the recognized instructions, without which mistyping would occur more often. Almost all of the code-editors offer code-highlighting except for some special ones like Imagine. To turn to functionality within this group, the help a DE provides has a great impact on educability. In fact, it can pose a great problem if the "available" assistance is not more than an internet address. This was the case with the first versions of FreePascal and Lazarus. It can be very problematic if students – mainly smaller children – cannot read help instructions in their native language. A "menu-controlled" help needs the most "humane input"; lengthy searching processes and incomprehensible, poorly or half-translated sentences hinder understanding, thus, guarantee student fiasco. Softwares with a menu controlled by a special key are mainly position-sensitive, so they help problem-solving. There are systems that publish the method of transcription, in order to make the generation of natural lingual versions easier (menu, help). NotePad++ (above mentioned as a code-editor) is a good example for this. Another helping function can be reminders about what to do later. Signing code lines with bookmarks or creating task lists can also help code-writing, which all appear in VS or in CodeBlocks, adding special remarks (ToDo) to places where we want to go back. Finally, we have to mention CodeLens or code mapping, which help us understand the structure of the program.

The last two functionalities are very useful in the case of huge programs, but usually in public education, except for some special classes, they are not necessary.

In the third group, we can find the functionalities that assist developers in creating more readable program codes. Let us mention indention briefly. One would assume that we could live without using indention, but we need to point out that such a code would certainly be unreadable. A quintessential service of CodeBlocks, among others, is that it re-organizes the code to a given "style" (to paragraph-

structure). With indention LCN Logo forces the expression of a subordinating structure, just like Python, but this latter editor makes it somewhat awkwardly. If the programmer steps out from the given level of code, he or she can return to make corrections only in a very complicated way. Professional IDEs even have refactorization functionalities as well, with which we can reorganize program code.

The last group of functionalities helps team work, which we think belongs more to professional work. Therefore, they are needed mainly in vocational training or in very special classes. It is needless to stress that nowadays all modern, professional DEs come with such functionalities.

Functionality of compiling, debugging, testing, documenting

Besides code-writing, DEs usually help us compile, execute, debug, or test our programs. We group some of the features in the following:

- compiling, executing, deploying
- debugging (Geany, Delphi)
- customizing (CodeBlocks)
- documenting
  - code export to pdf, html ... (CodeBlocks)
  - object classes from code (Visual Studio)
  - automatic generation from (Delphi)
- testing, analyzing, measuring (Visual Studio)

Most of the DEs use external programs for compiling and debugging. Certainly, it is more advantageous if we can choose which compiler or debugger to use. (For example CodeBlocks recognizes a big amount of C++ compilers, or in VS we can install compilers for several programming languages). After compilation DEs can execute the code and offer some services like showing the execution time (e.g. CodeBlocks). In the case of web programs, it is better if we can choose the browsers in which we want to try the code (it is well known, for example, that a JavaScript code will work differently in IE or in Firefox). Visual Studio, for instance, has a built-in feature to publish the web application on the web-server. In the case of mobile programs, it is beneficial if the DE can emulate the executional platform, while also deploying the application to the device.

Geany does not offer debugging functions (only after installing AddIn), while CodeBlocks, NetBeans GUI, and most of the DEs do. Interestingly, Visual Studio offers debugging services when the projects were written in different programming languages.

The role of customization is that we can use our own style of "programming," meaning, for example, indention style. A DE can offer a choice between safe code and efficient code, globally (for the whole program code) or locally (focusing only on some code details). This is an important question, because safe codes are needed

during the developing process, while efficiency is a good feature of a completed cone. It is an essential expectation that debugging mechanism (index out of bounds, bad type usage) can be built in or be rather switchable. [7]

A very important feature of DEs is how (much) they can facilitate documentation. The simplest way is to export the written code into html, pdf or any other useful format. A more complex solution is to create the graph about the connections among objects and classes. Finally, to mention another related functionality, existing in professional DEs like in Delphi, we can even create a sketch of documentation from the code.

In some DEs there can be automatic testing and measuring features as well (e.g. Visual Studio), but we do not go into details here, because these functionalities are used principally in professional contexts.

## 4. Availability

In our "shareware-world," it is a crucial question whether we have free access to (the info about) a DE, or we are left "alone" and dependent on others when it comes to information. The standards and/or documentations are necessary (but not definitely sufficient) conditions when considering the introduction of a DE into education. We also need to consider if the DE has a stable presence or not. It is not worthy to educate a DE which can disappear soon. Prevalence and a developing community on the web can guarantee stability.

## 5. Summary

First of all, we have to state that before making a decision that will determine our curriculum for years, we teachers must consider as many things as possible. If a teacher can choose a programing language like Imagine Logo, he or she has also chosen a DE. In any other cases, teachers have to think about the age-group of the students, their field of interest, and the overall goal of the classes (like whether students want to take a matriculation exam form informatics or not). These viewpoints will determine whether simplicity, learnability, complexity, or professionalism is more important when picking the DE. In the initial stages learning, that is, mainly in primary schools, the main viewpoint must be simplicity, along with the ability of shaping proper notions in the students. On the contrary, at the level of vocational trainings (ISCED 5) learnability as a viewpoint can be neglected for obvious reasons.

By learnability we mean the followings:

- The menu structure and/or ToolBox should be clear-cut.
- The user interface should be written in the native language of the students or should be manipulated with graphical objects (4GL).

- The help should also be written in their native language, while the operations should be simple.
- Debugging must be supported.
- It should be stable and reliable.

The most important professional viewpoints are as follows:

- It should support the creation of bigger programs, possibly consisting of several modules.
- It should help making documentations.
- It should have some code-writing help-services (e.g. ToDo-list, refactorization, etc.).
- It should support automatic code-generation.
- It should able to create different types of applications.
- It should support different PLs and compilers.

Last but not least, let us close the article with a summative table that lists the main features of the previously mentioned DEs, along with a brief evaluation.

| DE | simple | multi OS | chooseable compiler | multi pr. language | debugging | highlight | code completion | intellisense | native help | indention | multi-platform | custom-ization | many resources | document creation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **NotePad++** | + | - | - | + | - | + | -/+ | - | + | -/+ | - | + | - | - |
| **Imagine** | + | - | - | - | -/+ | - | - | - | + | - | + | -/+ | - | - |
| **Geany** | + | + | + | + | + | + | + | - | - | -/+ | + | -/+ | - | + |
| **CodeBlocks** | -/+ | + | + | - | + | + | + | - | - | + | -/+ | + | - | -/+ |
| **Lazarus** | -/+ | + | - | - | + | + | + | + | - | + | - | + | - | - |
| **V. Studio** | - | - | - | + | + | + | + | + | - | + | + | + | + | + |
| **Eclipse** | - | + | + | + | + | + | + | + | - | + | + | + | + | + |

# References

[1] Nemzeti Alaptanterv (2012), `www.kormany.hu/download/c/c3/90000/MK_12_066_NAT.pdf` (Accessed 15 April 2014)

[2] Heizlerné, V., Illés, Z., Menyhárt, L., Szempontok az informatika oktatási tartalmak kialakításához, in InfoDidact'12 Konferencia (2012)

[3] `njszt.hu/neumann/hir/20121031/az-informatika-es-az-uj-kozoktatasi--kerettanterv` (Accessed 15 April 2014)

[4] Szlávi, P.; Heizlerné, V., Evaluation of programming languages in education, in ICAI 2014, Eger, Conference lecture (2014)

[5] Nassi-Schneiderman diagram in HTML based on AML, Acta Didactica Napocensia (2013) Volume 6, Number 3 pp. 19-26. ISSN: 2065-1430

[6] Az informatikai alapismeretek vizsgatárgy írásbeli és szóbeli érettségi vizsgáihoz – A 2012. év vizsgaidőszakaira érvényes szoftverek listája (2012), `www.oktatas.hu/pub_bin/dload/kozoktatas/erettsegi/info2012osz/infalapism_irasbeli_szoftverlista_2012okt.pdf` (Accessed 15 April 2014)

[7] Szlávi, P., Feltételes fordítás alkalmazása (Turbo) Pascalban (e-manuscript) (1998), `people.inf.elte.hu/szlavi/ProgModsz/FeltFord.pdf` (Accessed 15 April 2014)

[8] Szlávi, P., Általános célú szoftverek értékelése iskolai szempontok szerint (e-manuscript) `people.inf.elte.hu/szlavi/InfoOkt/SzoftErt/SzoftverekErtekelese.pdf` (Accessed 15 April 2014)

[9] Bernáth, L., Solymosi, K. (ed.), Fejlődéslélektani olvasókönyv, Budapest, 1997. p. 21.