Developing multimedia courseware for EFL (ESOL) purposes

Ioulia Katsika
University of Thessaly, Greece

Abstract

Despite some scepticism related to the effectiveness of Computer-Aided Instruction (CAI), Computer Assisted Language Learning (CALL) is generally considered to be valuable and there is a wide range of software materials that could be used for the implementation of the approach. Taking into consideration the value of the CALL approach, its close relation to the software programs used for its implementation and the need for programs tailored to the needs of a specific class, this paper attempts to delineate the potentials and the limitations for the EFL teachers to create their own electronic software using authoring packages.

Keywords: computed assisted language learning, multimedia resources, software development, teaching

1. Introduction: The methodology involved in CALL

There are three developmental stages in CALL Methodology: the ‘behaviouristic’, the ‘communicative’ and the ‘integrative’ approach.

According to Hubbard (1987 quoted in Murphy 2000: 16), the behaviourist approach to CALL presents vocabulary and structure appropriate to the learner’s level through pattern (positive and negative) reinforcement. It aims to maintain the learner’s attention to the task and provides sufficient material for mastery and over-learning to occur. However, despite the well-founded peda-
gogical value that behaviouristic CALL displays, based on learner reinforcement and hence motivation, it has some obvious drawbacks which Murphy (2000: 17) summarises as follows: "focus on form rather than on meaning, sketchy and vague help or feedback, the computer as evaluative task-master that asks all the questions and judges all the answers and discrete points of grammar or vocabulary, mostly out of context and devoid of any real meaning". Due to these disadvantages the behaviouristic approach is no longer influential nowadays.

Communicative CALL on the other hand, which should be viewed within the general communicative approach to teaching, stresses the importance of authentic contexts and language in use. Underwood (1984 quoted in Murphy 2000: 17) illustrates a set of principles for the approach in question. He outlines that communicative CALL focuses on communication rather than on the form and avoids drill, teaches grammar implicitly through the lesson rather than explicitly and allows and encourages the student to generate original utterances rather than merely manipulate prefabricated language without judging or evaluating everything the student does.

The communicative approach has been very influential, but with the advent of software technology, another approach, integrative CALL, has stolen the limelight.

Integrative CALL is the most recent approach to CALL and it is closely associated with the developments in the field of multimedia design. Integrative CALL offers a more holistic approach to language teaching and learning. Warschauer and Healy (1998) argue that integrative CALL seeks both to integrate the various skills of language learning (listening, speaking, writing and reading) and to integrate the technology more fully into language teaching. This is the type of software one can create using programming languages or authoring tools such as Director, combining, as aforementioned, multimedia and hypermedia. Thus, the language learning software not only integrates listening, speaking, writing and reading, but it also contains a variety of exercises, while it is attractive in form, stimulating the learners' interest and motivation by giving them the chance to learn in a context that somewhat replicates an electronic game.

2. An overview of CALL materials

As far as CALL resources are concerned, there is a wide range of materials that could be used for the implementation of the approach. Jarvis (2000: 63)
states that "CALL includes not only multimedia software products, but also other language work applications, such as the word processor, and it also exploits the use of Internet, focusing mainly on e-mail and the World Wide Web (WWW)". CALL is closely linked to the creation and exploitation of software resources. Nowadays, there are a vast number of materials available in CD-ROM format or on the Internet, most of which are multimedia packages, programs which combine video, sound, graphics and animation, specially designed to promote language acquisition and learning. Jarvis (2000: 62) illustrates that "traditional CALL concerns software programs designed specifically for teaching and learning languages (...) CALL software essentially continues to come in two forms: off the shelf and ready-to-use programs known as dedicated CALL and programs which allow the teacher to create exercises known as authoring CALL". Ready-to-use software can be very attractive, but the teacher or the user should be very careful in their selection, because this type of software is made for general teaching purposes, and it is difficult to find a program with pedagogical value that matches the specific needs of a particular pupil or class.

Authoring packages like 'Wida Authoring Suite', 'Hot Potatoes' or 'Quia' on the other hand, allow teachers to create their own exercises exploiting the existing models without having to code in programming languages. Such authoring packages are useful, because even teachers with limited computer knowledge can create exercises adapted to the needs of a particular audience. Nevertheless, as Spanou (2001) asserts, "the drawback of such programs is that the teachers are limited in the types of exercises they can perform with their material, as they have to restrict themselves to the ones offered by the program". The disadvantage that Spanou points out is important, because the activities that the teacher can create (i.e. multiple-choice, gap-filling, drag-and-drop exercises, crosswords etc.) are predetermined by the program and this can be frustrating for the teachers because it limits their creativity and resourcefulness.

Another option for teachers who are keen on developing their own multimedia software is the use of programming languages such as Visual C++, Visual Basic and Java or high level authoring tools such as WinCALIS, HyperCard, Toolbook Assistant, Macromedia Authorware and Macromedia Dire-

---

1 These are some representative examples of the authoring packages available in the market. For more information, see http://www.wida.co.uk/noframes/auth.htm (for Wida), http://web.uvic.ca/hrd/halfbaked/ (for Hot Potatoes) and http://www.quia.com/products/quia_web.html (for Quia).
ctor. As Warschauer (1996) asserts, these tools are very flexible and give the teacher the opportunity to create language teaching material that combines multimedia (text, graphics, sound, animation and video) and hypermedia since the multimedia resources are all linked together and learners can navigate their own path simply by pointing and clicking a mouse. The results one can achieve with these tools are impressive, and the most important thing is that the teacher can produce programs that cater for the needs of a specific class and at the same time include the full range of activities needed. Thus, the programs created with the authoring tools in question are pedagogically more effective than the ‘readymade’ programs or those created with the less flexible authoring packages.

The disadvantage of the authoring tools mentioned in the previous paragraph is that they demand a lot of time and effort to master. As Gimeno-Sanz and Davies (1979) aptly put it, “multipurpose authoring packages require a considerable degree of programming expertise in order to get the best results. A language teacher can achieve reasonable results with such packages if he/she is willing to persevere, but the time taken to learn a complex package such as Macromedia Director will probably outweigh the benefits”. Beyond doubt, developing programming skills is a painstaking process for the language teacher whose academic background does not include computer programming or multimedia design. Nonetheless, if the teacher is keen on working hard to understand how these tools work so that s/he can realise her/his ideas, then the whole effort is rewarding. Creating a multimedia course is a productive process and furthermore a computer expert language teacher is likely to create a very competent program, being able to combine appropriate use of technology with an efficient methodological approach.

3. A technical approach to software creation

Designing and creating multimedia CALL material is not an easy task. As Gimeno-Sanz and Davies (1979) state,

it calls upon a range of skills and meticulous attention to detail.
Such is the complexity of computer programs these days that it is

---

highly unlikely that a single person will have all the necessary skills
to undertake a CALL development project alone. Team work is
therefore essential.

It is obvious that teamwork is the best possible solution when it comes to
creating this kind of material, because the language teacher does not have to be
a computer expert or to know any programming languages. The allocation of
tasks is necessary, so that different professionals are responsible for different
aspects of the software (i.e. the teachers for the pedagogical implications and
the developers for the implementation). This way much time and effort is
saved, because the teacher does not have to go through the painstaking and
time-consuming task of mastering a programming language or an authoring
tool.

Nevertheless, the ideal situation where a lot of experts work on different
aspects of the same well-organised and sponsored project is not always the case.
Frequently, the teachers have to create the material on their own. Thus, if they
aspire to produce software tailored for the needs of a specific group of students,
they have to learn how to program in a computer language or how to use an
authoring package. Apart from the basic platform, the development of software
material demands the use of a number of other programs, mainly image and
sound editors as well as animation and interaction generating software.

4. User interface and design considerations

In order for a piece of software to be effective, there are many issues the de-
veloper has to bear in mind on top of the technicalities pertinent to the explo-
tiation of the platform, especially with regard to the user interface. According to
Bonsiepe (1995: 43) interface has been defined as “a specification of the look
and feel of a computer system. This includes what types of objects the user sees
and the basic conventions for how the user interacts with those objects”. The
interface is what stands between the program and the user, and consequently its
display is of paramount importance because the interaction between the user
and the program is based on what the user can see on the screen. The tools used
for the creation of multimedia programs, either visual programming environ-
ments or authoring software, are fronted with a graphical user interface (GUI),
that is, they use pictures rather than just words to represent the input and
output of a program. The design of the GUI is a complicated matter because it
encapsulates a lot of design considerations.

Before proceeding with the design considerations though, it is considered
useful to present the notion of a window, that is the major screen object that nowadays is almost synonymous with GUI. According to Hix and Hartson (1993: 59), “a window provides an arena for presentation of, and interaction with, other interaction objects. (…) via windows, a user can organise work by tasks and can work on several tasks at once”. The window form of the program renders it user friendly because nowadays the prevalent computer operating systems (OS) are windows based as well, and thus this layout is very familiar to the user.

The GUI’s effectiveness depends on various factors, both general and specific, according to the needs of the target audience of the software. First of all, the interface should be user-friendly, clear and attractive. It should not confuse the learners, it should be functional and there should be something more than a flashy display. As Bonsiepe (1995: 51) aptly points out, “it is essential to find a balance between such extremes as the audio-visual overload derived from video games and the visual poverty found in command-line interfaces”. Multimedia design is a challenge when it can produce programs that are efficient both in terms of practicality and smartness. After all, as Macro (1987: 165) holds, “the best approach to the problem of design is to regard it as an exercise in reducing complexity”. For this purpose, the program in question, although it is designed for children, and hence its effectiveness relies a lot on its appearance, tries to keep an interface as simple and comprehensible as possible, with rather self-explanatory buttons and icons without unnecessary graphics or “glitzy stuff”, as called by Bonsiepe (1995: 51).

Important features that reinforce the effectiveness of the software are also the balance between image and text (Bonsiepe 1995: 51) and the use of colours and consistency. It is very important that the screen should not be congested either with text or pictures, because then it becomes unattractive and even confusing for the users. The imbalance between pictures and words can result in loss of meaning. As Bonsiepe (1995: 51) outlines, “there is a danger of being fascinated by the technical effects of animation and losing sight of the objectives which the selection, arrangement and presentation of the information should serve for effective communication”. This equilibrium between graphics and text is not always an easy task, especially on teaching software targeted for children, which should contain a lot of pictures and passages.

As regards colour, a multi-coloured scheme can be more appealing to the young learners. For efficiency purposes one can follow the guidelines given by Fowler and Stanwick (1995) and the ICT4LT project (module 3.2 http://www.ict4lt.org/en/index.htm). Thus, it is generally preferable to use pictures that have the same brightness and spectrum and backgrounds in pastel colours,
so that the dazzling and tiredness of the eyes is avoided. Furthermore, colour can be used for the reinforcement of meaning so that the user knows where to focus his/her attention. Nevertheless, the use of extensive colour-coding should be avoided in case there are colour blind users who confuse certain colours and hence would not take notice of it. For this purpose, the parts of the software should be explicit and colour should be used for the reinforcement and not the conveyance of meaning.

It is also essential that the use of buttons and icons be consistent. The same buttons or icons should convey the same meaning wherever they are found in the software.

On top of these general considerations, there are more design requirements relevant to the nature of the software and the specifications of the audience it addresses. The design (and of course the content) of language teaching material, such as the one in question, is influenced by “factors such as the age of students, their linguistic and cultural background, the status of the target language in the country where it is being taught, and the reasons for learning it” (Mccarthy 1994).

Moreover, Xix and Hartson (1993: 30) advise involving the user via preparatory design. They assert that “getting users involved in interaction development is now widely recognised as a key to improve the usability of the software. Designers benefit from the problem domain knowledge that only the users have”. For this reason, the projects should be examined by future users during the testing procedure and their comments should be taken into serious consideration and constitute the basis for many design changes.

5. Testing the software

Testing the software is very important, as it results in the program being robust. Macro (1987) argues that “nowadays it is generally understood that the proportions of work involved in the principal phases of the software lifecycle are approximately, specification 15%, design 35%, coding 15% and testing 35%”. Thus, it is evident that testing is a vital part in software development.

The test plan for the program normally involves two stages: on-going modular testing and final testing. The former entails that each individual part of the system is tested by itself using ‘compile all scripts’ or similar commands before the next part of the program is built. The people working on the project should test separately all the sections of the program for usability and errors on a regular basis. During the modular testing procedure several instances of malfunctions can be detected and remedies should be sought.
Once the latest version of the software is complete, a full test plan must be drawn up and conducted to test the overall software. The final testing procedure involves two aspects: testing for the pedagogical value of the program, and testing for technical inconsistencies. For this reason both software engineers and language teachers should test it. The program should also be tested by future users, that is persons who belong to the target group of the software, because users have anticipations and know what they expect; thus their comments are very relevant and useful (Hix and Hartson 1993). To perform the testing, subjects can be given a copy of the program and a questionnaire to write their opinions and comments. For better results, different questionnaires should be given to different groups of testers.

6. Conclusion

All in all, creating an integrative CALL piece of software is a difficult but gratifying process for the language teacher. The considerations one should take into account are several, the amount of time and effort needed is vast, but there are major advantages. Generally, the overall process is very creative and gives the teacher a sense of accomplishment, especially if the result meets with her/his expectations. Additionally, if the teacher herself/himself creates the software, then (s)he can tailor it according to the needs of a given target audience. Thus, the software created becomes a lot better than those commercially available, because it caters for the needs of a specific group of learners, and it is not something that was created for general commercial purposes. Of course, this observation does not intend to undermine the importance of the commercially available software; it just aims to emphasize the pedagogical value of software design. Especially nowadays, that the Computer Assisted Language Learning approach is widely followed, the importance of teachers competent enough to design their own software for their teaching purposes is enormous.

References


Gimeno-Sanz A. and G. Davies. “CALL software design and implementation”. In The ICT4LT Project, Module 3.2.


