RESEARCH PAPER ACCE CONFERENCE 2008

Which programming language makes it easier for students to learn to program?

ABSTRACT

A comparative examination of the effectiveness of five introductory programming languages - PhP, Visual Basic, Gamemaker, Alice, and RoboLab. Based on the degree of visual processing involved, this study explains some reasons that different programming languages are more or less effective in teaching programming concepts.

Introduction

The choice of computing language used to teach programming is often a personal one and based primarily upon the experiences of the teacher. While significant research has been conducted in the area of learning to program, the study from which this paper is drawn (Zagami, 2008) focused on the use of visualisation by programming languages to support learning. This paper is reports on the placement of programming languages on to a continuum based on the degree to which cognitive processing is conducted in the auditory or visual processing portions of the brain. The paper begins with a summary of cognition as it relates to the paper, an overview of the study as it pertains to this paper, a comparative overview of the programming languages examined in the study, and concludes with a presentation of the continuum developed for the study.

Cognition

The association of text with auditory processing and the comparison of this to the visual processing supported by programming languages is the focus of this paper. Spoken language is predominantly one dimensional, sequential and sentential (Crapo, 2002) and preprocessed in the auditory centres of the brain (Card, Moran, & Newell, 1983) before temporary storage in working memory. Text, used as a substitute for spoken language (Carpo, 2002), is likewise one dimensional, sequential and sentential and also preprocessed in the auditory centres of the brain before temporary storage in working memory. Visual images are conversely multi-dimensional and visual input requires preprocessing in the visual centres of the brain before temporary storage in working memory, these processes are summarised in the Model Human Processor model of cognition (Figure 1).

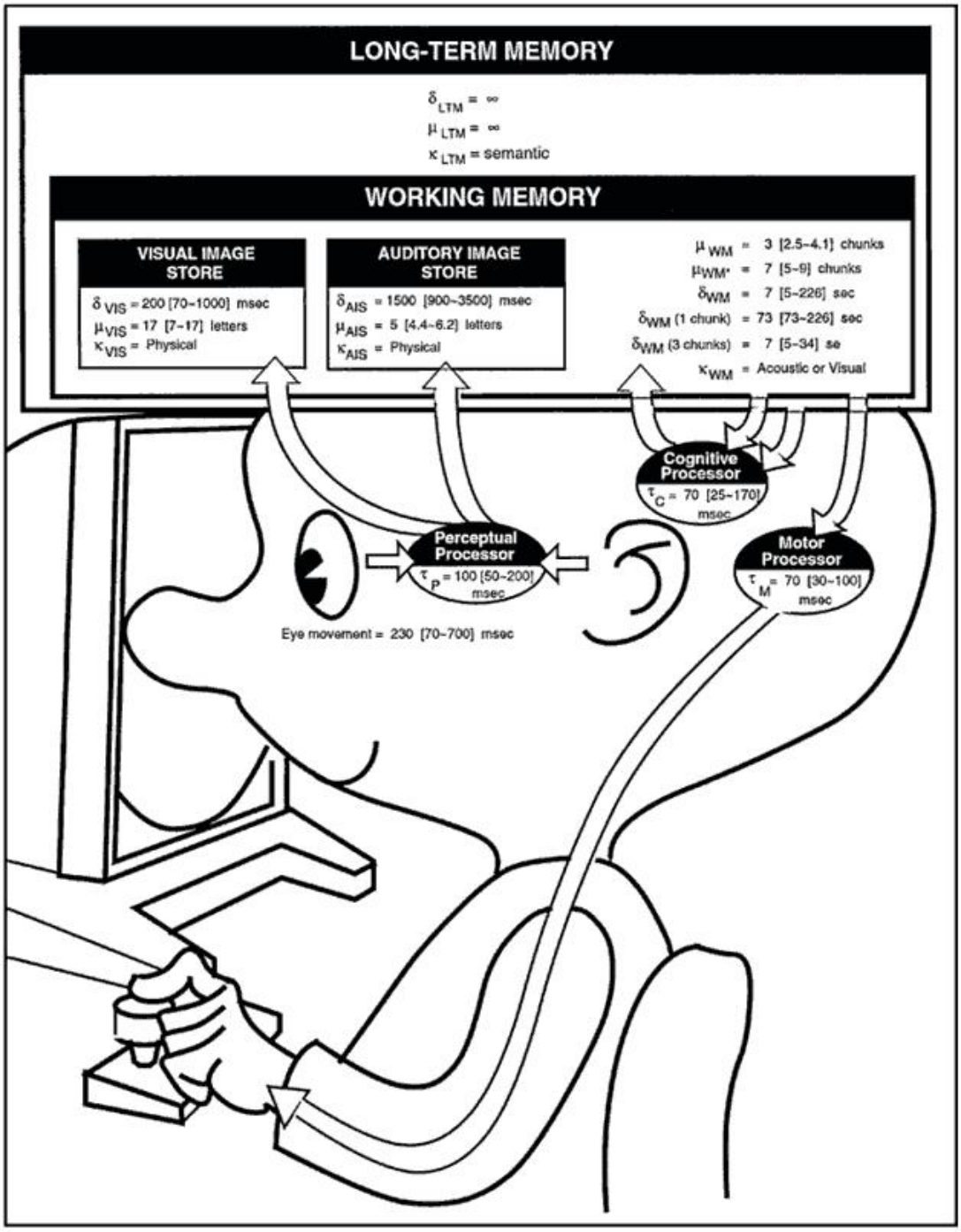

Figure 1: Model Human Processor (MHP) model of cognition (Card, Moran & Newell, 1983)

The importance of cognitive processing is highlighted by theories of cognitive load. Cognitive processing of working memory draws upon a schema or mental model of the concept stored in long term memory (Sweller, 1994). This progressive mental model is modified based upon new material in working memory as it supports or challenges existing understanding during cognitive processing. Cognitive load occurs when the capacity of working memory is exceeded, resulting in a reduction in the brains cognitive processing capacity. From the perspective of memory, cognitive load is the sum of the load on three kinds of working memory: intrinsic, germane, and extraneous (Paas, Renkl, & Sweller, 2003).

Germane cognitive load relates to the association of other concepts involved in the modification of the overall mental model and can be influenced by motivational factors. Intrinsic cognitive load is an immutable characteristic of the complexity of the concept under study and is not generally modifiable by the programming language used. The instructional environment, in this case the programming language, can however vary the extraneous cognitive load. Studies have shown (Crapo, 2002) that minimising the impact of extraneous cognitive load can be achieved through a reduction in the input required to generate mental representations. Textual representations, processed as auditory sequences, can place a greater burden on working memory than visual representations. Visual representations do not require significant translation as they already exist in a format conducive to visual manipulation of the processes involved in the concept. In addition, the reduction in the input to working memory required to contextualise textual representations reduces extraneous cognitive load and provides greater working memory capacity for cognitive processing and development of the mental model of the concept.

The study (Zagami, 2008) measured student mental model development of four programming concepts using five programming languages. The concepts ranged in complexity from low (sequence) to relatively high (modularity). For low complexity concepts, it was found that the choice of programming language did not make a significant difference in the development of student understanding of the concept. In these cases, the working memory capacity used by inputs from auditory (textual) or visual inputs did not exceed overall working memory capacity. This resulted in sufficient working memory capacity remaining for cognitive processing and the improvement of the students mental model of the concept.

As the complexity of the concept studied increased, demand on working memory capacity also increased (intrinsic cognitive load). For example, visualising a sequence of steps requires a less complex mental representation than visualising the steps involved in nested iterations. Likewise the amount and complexity of text required to describe a sequence of steps is less than that required to describe nested iterations. This in turn demonstrates the advantages with respect to working memory capacity of diagrammatic or visual representation over a textual or auditory representation. As the complexity of the concept increased, the working memory used for the mental representation of the input also increased. Where he input was textual, it required greater working memory than when the input was visual. In addition, as the complexity of the concept increased, the cognitive processing required to develop the mental model increased for not only the mental representation produced by the input but also related mental models (Germane cognitive load). For example, to develop a mental model of selection, a student must draw upon their understanding of sequence, accessing their mental model of sequence and requiring the use of additional working memory capacity.

In general, increasing the complexity of the concept was found to reduce available working memory. Programming languages that required substantial working memory to generate the required visualisations, those predominantly making use of textual representations, more rapidly exceeded working memory capacity resulting in a reduction in the available cognitive processing capacity. This increased the time taken for students to improve their mental model of the concept and reduced the complexity of the model that was achievable in the time available. Programming languages that used predominately visual representations, was found to place less demand on available working memory capacity, and as a result provided sufficient working memory capacity for cognitive processing and the development of student mental models. This resulted in less time taken for students to improve their mental model of the concept or in the complexity of this model achievable in the time available.

Study

This paper reports on part of a qualitative analysis of collective case studies (Zagami, 2008) of programming languages as taught to thirty one adolescent females at a suburban private school studying a variety of courses that included computer programming over a three year period. An application of mental model theory (Pirie & Kieren, 1989) was modified to progressively track their cognitive development (Zagami, 2006) of four programming concepts. These concepts were sequence, iteration, selection and modularity. Five programming languages were compared in the study and the degree of support each language provided to student mental visualisation of a concept was measured. An application of mental model theory (Zagami, 2006) provided a scaled framework to measure this development through a process of stimulated recall facilitated by video and screen recording (Figure 2).

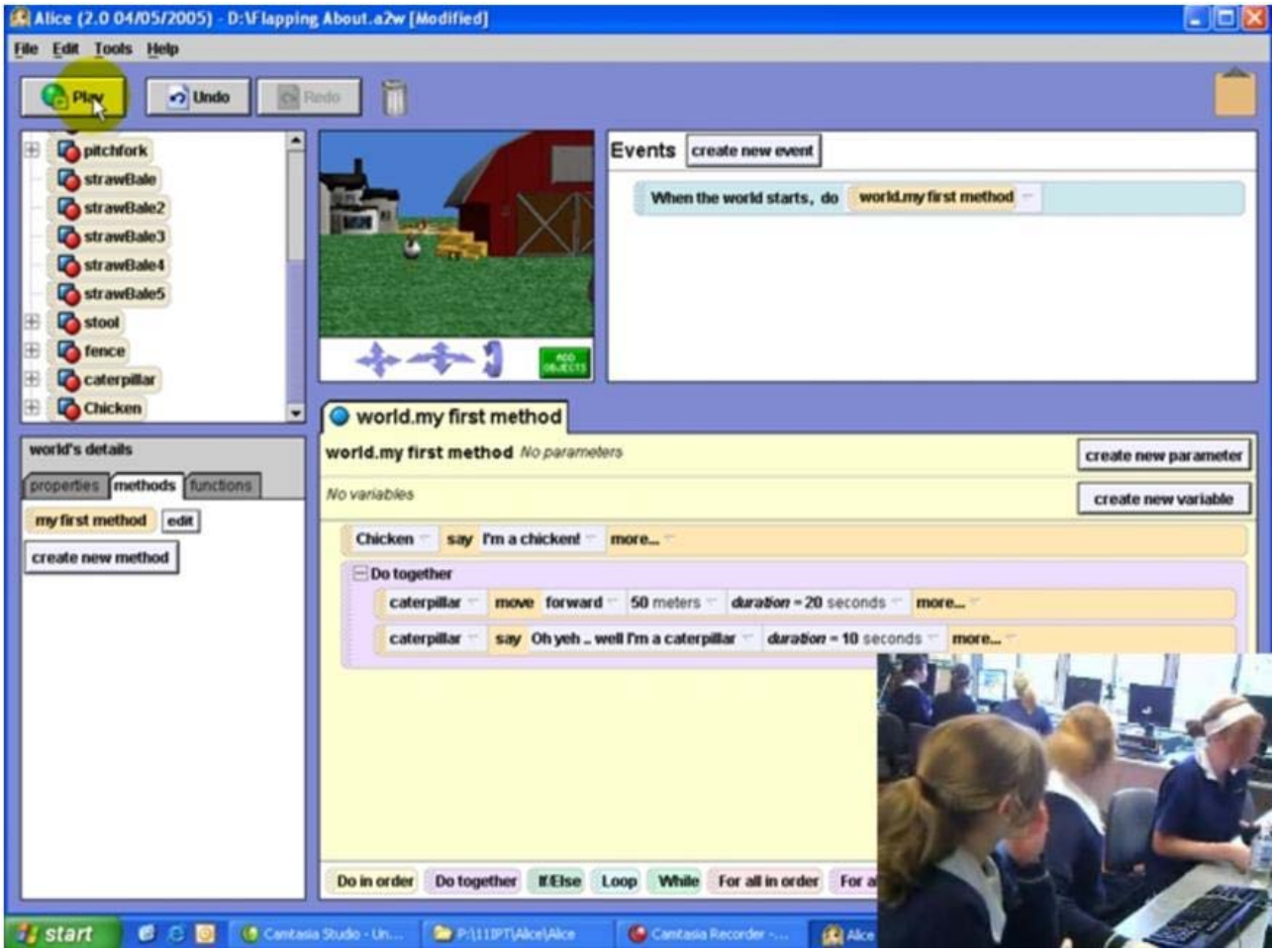

Figure 2: Screen and video recording for stimulated recall

This paper reports on the placement of the programming languages into a continuum based on the degree to which cognitive processing is conducted in the auditory or visual processing portions of the brain. The study used a collective case study methodology in which each programming language formed a specific case. In measuring the degree to which each language used auditory (textual) or visual processing, thirty one students studied one or more of five programming languages. Through a wide range of problem based tasks in which student responses were measured by speak aloud peer discussion, and stimulated recall using screen and video recordings (Figure 2), measures of the degree to which students used the textual or visual aspects of the program language were made.

Programming Languages

The study examined five programming languages commonly used to teach introductory programming concepts, PhP, Visual Basic, Alice, GameMaker, and RoboLab. The four programming concepts of sequence, iteration, selection, and modularity were studied using these languages and the effectiveness of the languages in developing student understanding was determined, measured by the complexity of their mental model of the concept (Zagami, 2008). To assist analysis of these cases, each programming language was placed upon a continuum from predominantly auditory processing to predominantly visual processing as determined by case studies of the individual programming language.

The PhP programming language (Figure 3) was found to predominantly involve auditory (textual) processing with some visual relationships through indenting and colour highlighting. For low complexity concepts such as sequence and simple iteration students processed these concepts into mental visualisations without difficulty. For relatively complex concepts such as complex iteration, selection and modularity, students experienced considerable difficulties in the translation of textual representations into mental visualisations of the processes involved and the PhP language was found to place sufficient demands on student working memory to impede the development of their understanding of these concepts.

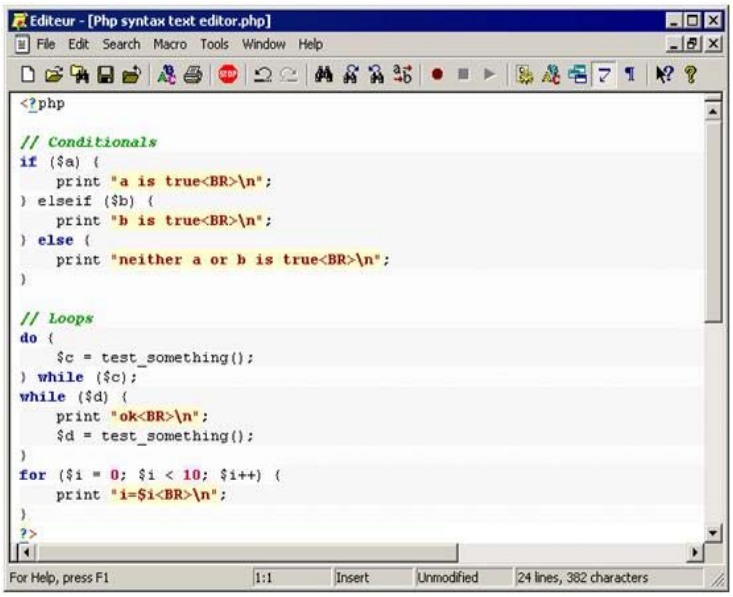

Figure 3: PhP Programming Language

The Visual Basic programming language (Figure 4) provided a visual interface but it was found that this predominantly involved the graphical prompting of textual commands and property setting. For low complexity concepts such as sequence and simple iteration the Visual Basic language was found to also be effective in developing student understanding of these concepts. For relatively complex concepts such as complex iteration, selection and modularity, the auditory translation of the Visual Basic language was also found to be placing sufficient additional demands on student working memory to impede the development of their understanding of these concepts. While the visual aspects of the interface were found to assist in reducing errors in the language syntax, it was not found to assist in the development of mental visualisations for these concepts.

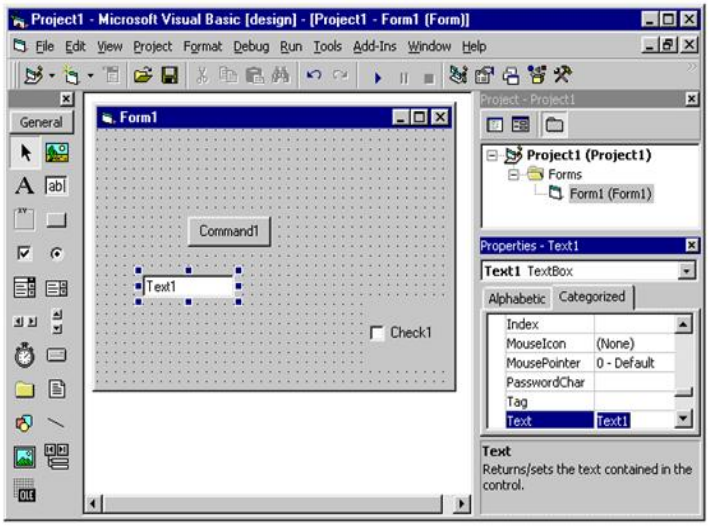

Figure 4: Visual Basic Programming Language

The Alice programming language (Figure 5) was found to use labeled icons and a visual representation of the structure of the program processes. For low complexity concepts such as sequence, iteration and simple selection the Alice language was found to be effective in developing student understanding of these concepts. The visual representation of programming processes assisted in the development of student mental visualisations when the process involved was clearly visually articulated by the interface, but as the complexity of the problem sets increased, translation of textual elements came to increasingly dominate the processes involved in student understanding of the concept. For relatively complex concepts such as complex selection and modularity, the Alice language still placed additional demands on student working memory that impeded the development of their understanding of these concepts.

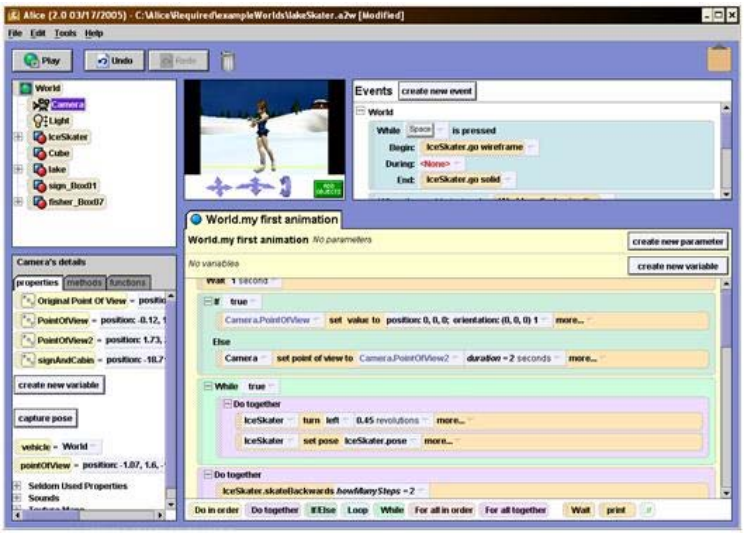

Figure 5: Alice Programming Language

The GameMaker programming language (Figure 6) was found to mostly use unlabeled icons but still required the use of text for parameter setting. For concepts such as sequence, iteration and selection the GameMaker language was found to be effective in developing student understanding of these concepts. For the more complex concept of modularity, the GameMaker language was found to be less effective in assisting the visualisation of this concept as the interface could not fully display the elements involved and

this impeded student mental visualisation of the distinct elements involved in problems involving modularity and difficulties in sustaining these elements in student working memory was found to impede the development of their understanding of this concept.

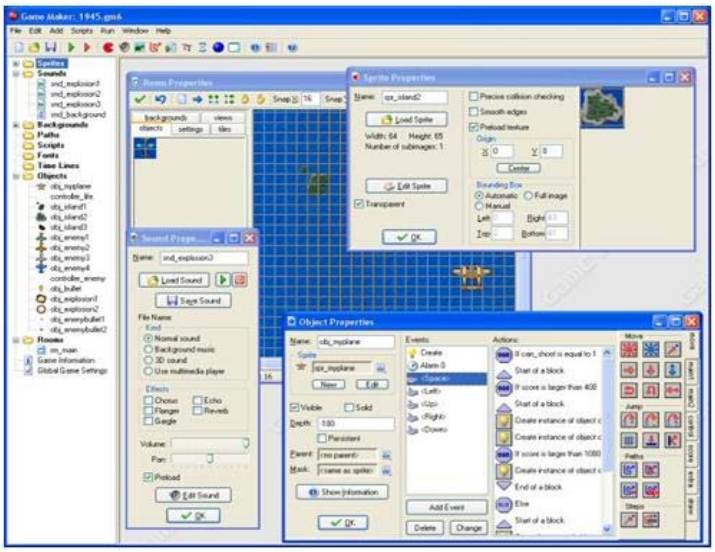

Figure 6: GameMaker Programming Language

The RoboLab programming language (Figure 7) did not use text at all for the concepts studied, and relied entirely on visual representation of the processes involved. For low and high complexity concepts the RoboLab language was found to be the most effective in developing student understanding of the concepts and was not found to make sufficient demands on student working memory to impede the development of their understanding of these concepts. The reliance on visual representation in the language interface and the graphical abstraction of language control structures was found to assist students in developing their mental visualisation of the concepts.

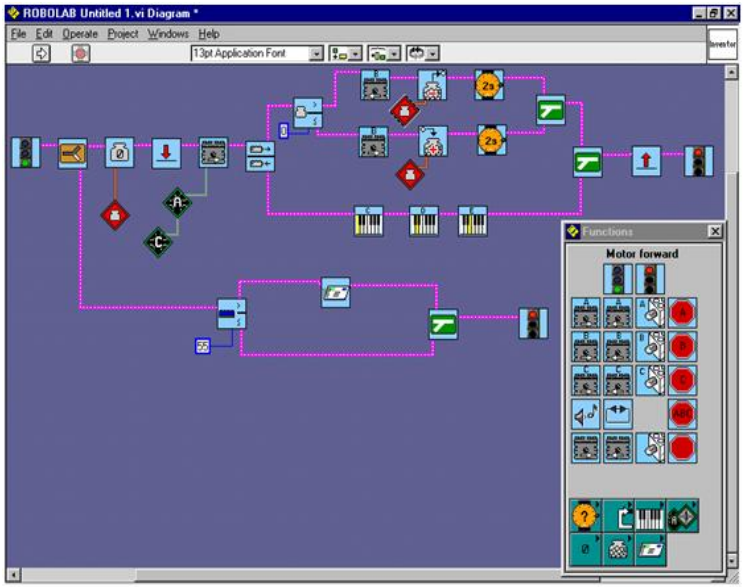

Figure 7: RoboLab Programming Language

Conclusion

The resulting analysis placed the selected languages on a continuum (Figure 8) from predominantly supporting an auditory mental representation of the concept, through to predominately supporting a visual mental representation of the concept. This provided a basis in the wider study on which to examine the effectiveness of the programming language to support the development of programming concepts using a cognitive framework.

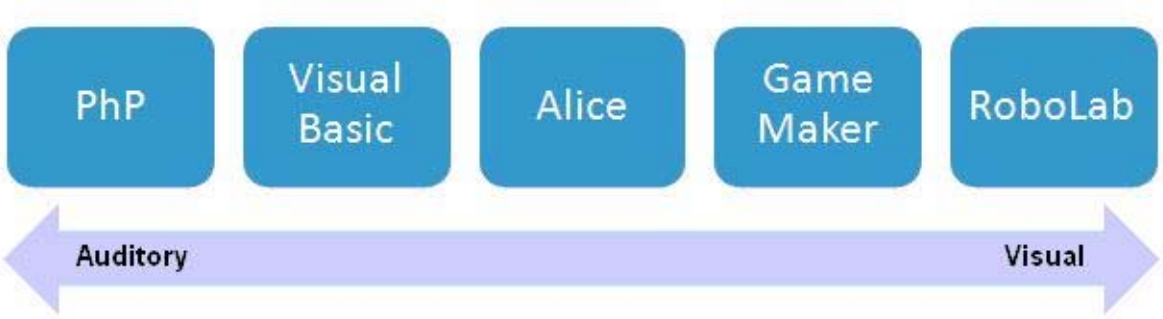

Figure 8: Auditory to visual continuum

The choice of which programming language to use for teaching programming is often the subject of passionate but unsupported debate. This paper seeks to clarify the selection of programming languages from one perspective, the degree of visualisation supported by the language, and place the language on a continuum in relation to the other languages examined. In the wider study from which this paper is drawn (Zagami, 2008), programming languages that rely predominantly on visual representations were shown to be more effective in supporting student understanding of introductory programming concepts than programming languages relying predominately on text.

This paper details the placement of the five programming languages used in the study on a continuum from predominantly auditory processing to predominantly visual processing. While the study and the continuum as presented suggests increased effectiveness in teaching introductory concepts as the degree of visualisation supported by the language increases, this is presented from one narrow perspective, that of visualisation. Many other factors can influence the effectiveness and choice of a programming language such as the languages paradigm - procedural, object-oriented, functional, or logic. The degree to which the language supports the development of various types of applications is another consideration - web based, games, multimedia, information systems, mathematical calculations, or control systems. Finally, the level of instruction must be considered. This study examined introductory programming concepts only and found that the complexity of the concept under study influenced the degree to which a programming language was successful or not in developing a students understanding of a concept. More complex concepts, such as recursion, with subsequently greater demands on working memory may not follow the trends identified in this study.

REFERENCES

Card, S., Moran, T., & Newell, A. (1983). The psychology of human-computer interaction. Hillsdale, NJ: Lawrence Erlbaum Associates.

Crapo, A. (2002). A cognitive-theoretical approach to the visual representation of modelling context. (Doctoral dissertation, Rensselaer Polytechnic Institute, 2002). Dissertation.

Paas, F., Ranki, A., & Sweller, J. (2003). Cognitive load theory and instructional design: Recent developments. Educational Psychologist, 38(1), 1-4.

Pirie, S., & Kieren, T. (1989). A recursive theory of mathematical understanding. For the Learning of Mathematics, 9(4), 7-11.

Zagami, J. (2006). Use of visual programming environments to learn introductory programming concepts. Proceedings of ACEC, the Australian Computers in Education Conference, July 2006, Cairns, Australia.

Zagami, J. (2008). Seeing is understanding: The effect of visualisation in understanding programming concepts. (Doctoral dissertation, Queensland University of Technology, 2008). Dissertation.