CRITERIA AND GUIDELINES FOR THE SELECTION AND IMPLEMENTATION OF A FIRST PROGRAMMING LANGUAGE IN HIGH SCHOOLS

L. GOOSEN B.Sc. HED(P) B.Ed. M.Ed.

Thesis submitted for the degree Philosophiae Doctor in Subject Didactics at the North-West University (Potchefstroom Campus)

Promoter: Prof. H.D. Nieuwoudt

Co-promoter: Dr. E. Mentz

2004

ACKNOWLEDGEMENTS

A word of heartfelt thanks to the following persons, who, each in a particular capacity, contributed to making the production of this thesis possible:

- Various friends and family members for support and encouragement.
- Subject advisors and administrative personnel in the various provincial offices for permission to carry out the empirical study, together with supplying some data and information needed.
- All respondents who completed questionnaires.
- Dr. Ellis from the Statistical Consultation Service of North-West University (Potchefstroom Campus) for advice on the questionnaire, as well as statistic calculations carried out.
- Thys and Lizette de Jager for final editing and proof reading.
- Dr. Mentz for showing the way computer-wise and always responding quickly.
- Prof. Nieuwoudt for having been there all the way for all this time.

SUMMARY

CRITERIA AND GUIDELINES FOR THE SELECTION AND IMPLEMENTATION OF A FIRST PROGRAMMING LANGUAGE IN HIGH SCHOOLS

The choice of a first programming language is critical for fostering problem solving skills and remains a concern in programming instruction that should be taken into account during discussions on the Computer Science curriculum. Criteria that should be considered in selection, therefore, need to be established. Few precedents exist for the implementation of such a language, and guidelines for this process ought to be introduced. This study aims to institute valid criteria and guidelines for the selection and implementation of a first programming language in high schools. Criteria for selection were established and guidelines for implementation introduced through a literature study, after which the validity of these was tested empirically.

Care was taken to ensure the relevance of criteria, which were established regarding the development of thinking and programming skills, requirements for the programming language and its software development environment to make it appropriate for learners, new tendencies in programming, issues influencing programming used in practice, affordability, training and resources, and programming for various purposes.

Guidelines were introduced for the implementation of a first programming language in high schools regarding appreciating teachers' roles in implementation, issues surrounding pilot testing, considering cost implications at macro-implementation level, introducing a new curriculum, and guidance, support and assessment.

The validity of selection criteria and implementation guidelines identified was empirically verified within the South African context, in that all items in a questionnaire based on criteria and guidelines received averages rating these as 'fairly important'.

Effect sizes designating practical significance for differences between

- the reported importance and application/usage of criteria and guidelines and
- the use of implementation guidelines as regarded by groups consisting of policy makers and teachers respectively

indicate a need for more careful consideration of what is important and practical in the world of the teachers who implement a selected language.

Key terms: selection criteria, implementation guidelines, first programming language, high schools, Computer Studies, curriculum development, curriculum implementation, curriculum dissemination

OPSOMMING

KRITERIA EN RIGLYNE VIR DIE SELEKSIE EN IMPLEMENTERING VAN 'N EERSTE PROGRAMMERINGSTAAL IN HOËRSKOLE

Die keuse van 'n eerste programmeringstaal is kritiek vir die bevordering van probleemoplossingsvaardighede en omdat dit van uiterste belang is vir programmeringsonderrig, behoort dit tydens besprekings van die Rekenaarwetenskapkurrikulum in ag geneem te word. Om dié rede moet kriteria gevestig word wat by die keuse van 'n eerste programmeringstaal oorweeg moet word. Omdat daar voorheen min gevalle was waar so 'n taal geïmplementeer moes word, behoort riglyne vir sodanige proses ingestel te word. Hierdie studie is daarop gerig om geldige kriteria en riglyne vir die seleksie en implementering van 'n eerste programmeringstaal in hoërskole daar te stel. Nadat 'n literatuurstudie voltooi is, is kriteria vir sodanige seleksie gevestig en riglyne vir implementering daargestel. Die geldigheid van hierdie kriteria en riglyne is daarna empiries getoets.

Sorg is gedra om die toepaslikheid van kriteria te verseker. Kriteria is ingestel vir die ontwikkeling van denk- en programmeringsvaardighede, vereistes vir die programmeringstaal en die gepaardgaande programontwikkelingsomgewing wat dit vir leerders geskik maak, nuwe tendense in programmering, kwessies wat programmering in die praktyk beïnvloed, bekostigbaarheid, opleiding en hulpbronne, en programmering vir 'n verskeidenheid doeleindes.

Riglyne vir die implementering van 'n eerste programmeringstaal in hoërskole is ingestel vir die erkenning van onderwysers se rol in die implementering, kwessies aangaande loodstoetsing, oorweging van koste-implikasies op makro-implementeringsvlak, ingebruikneming en disseminasie van 'n nuwe kurrikulum, en leiding, ondersteuning en assessering.

Die geldigheid van die gekose seleksiekriteria en implementeringsriglyne is empiries binne die Suid-Afrikaanse konteks bevestig. Alle items in 'n vraelys wat op kriteria en riglyne gebaseer is, het gemiddeldes aangeneem wat dit as 'taamlik belangrik' klassifiseer.

Effekgroottes het beduidende verskille met praktiese implikasies aangedui tussen

- gerapporteerde belangrikheid en toepassing/gebruik van kriteria en riglyne, en
- die gebruik van implementeringsriglyne vir groepe wat onderskeidelik uit beleidmakers, onderwysers en opleiers bestaan.

Hierdie verskille dui daarop dat daar 'n behoefte bestaan aan deegliker oorweging van dít wat die onderwysers wat 'n gekose taal moet implimenteer, as belangrik en prakties beskou.

Sleutelterme: seleksiekriteria, implementeringsriglyne, eerste programmeringstaal, hoërskole, Rekenaarstudie, kurrikulumontwikkeling, kurrikulumimplementering, kurrikulumdisseminasie

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

STATEMENT OF THE PROBLEM AND RESEARCH METHOD

1.1 INTRODUCTION AND ORIENTATION

Important changes in computer and communication hardware and software have occurred frequently (Westrom, 1992) and recent developments have given many people access to powerful computers in the form of desktops, laptops and embedded systems (Van Rossum, 1999). Computers and information technologies are being infused into every aspect of our global culture (Adams, 2003:285) and computer technology is being increasingly depended upon to facilitate processes taking place in homes, workplaces and schools.

We are rapidly entering an age where information appliances, wearable computers and deeply networked, embedded central processing units (CPUs) in everyday objects offer users control over their physical and information environments. Hardware is now also fast and cheap enough to make mass computer education possible - the next big change will occur when most computer users have the knowledge and power to create and modify software. However, while few people doubt the fact that technological skills have become increasingly important in the 21st century (Madison & Fifford, 2003:217) and many people nowadays use computers, only a very small percentage of these users are computer programmers.

The first computers only started appearing in schools around 1965, which gives them a recent history in this milieu (Westrom, 1992). As more computers began to move into schools and classrooms in the mid and late 1970s and early 1980s, the term "computer literacy" became very common in educational circles (Deek & Kimmel, 1999:98).

The majority of high school[1] computer classes focuses on what schools refer to as computer literacy courses, where learners learn keyboarding or data entry skills (Bernardo & Morris, 1994) and basic computer software applications, such as word processing, spreadsheets and databases.

The other primary component of the typical computer curriculum offered in most high schools appears to be programming (Deek & Kimmel, 1999:100). Although Computer Science is an ever-expanding field that is characterised by a combination of theory, practice, knowledge and skills, and includes many activities beyond programming (SIGCSE, 2001a), programming will, due to the focus of this study, feature prominently.

---

[1] In light of the title of this thesis, the term "high school(s)" will be used consistently to refer to all such institutions, including, but not restricted to, ones classified as secondary schools.

In South African (SA) schools, learners can learn about programming by taking the subject Computer Studies.[2] The subject is offered in Grades 10 – 12, i.e. the final 3 years of formal schooling, on two grades, where the Higher Grade (HG) concentrates on problem solving using programming as vehicle (Chiles, 2001). For the majority of these learners, the first programming language that they come into contact with is the one that they learn at school.

Another matter that needs to be cleared up pertains to the confusion that exists between Computer Science as a discipline and instructional technology as a tool for teaching and learning (Deek & Kimmel, 1999:91). Generally, technology can be viewed as a tool that cuts across all subjects (Deek & Kimmel, 1999:92). However, where technology is the auxiliary tool in most subjects, it is the primary tool for instruction in Computer Science. Therefore, within the context of this discussion, Computer Science must be considered as subject matter as opposed to a tool: the computer is the tool and Computer Science is a discipline that uses it, naturally, among other tools.

1.2 STATEMENT OF THE PROBLEM AND MOTIVATION

Computer Studies is an important subject at school level, because it not only conveys technical aspects of computers and programming, but also offers a new approach to problem solving by which the thought processes of learners can be drastically enriched (Von Solms, 1979:11). The programming taught in Computer Studies is linked to the development of practical abilities in problem analysis, design of solutions, practical implementation - which includes coding in a computer programming language - and evaluation (Kirkwood, 2000:509). Computer programming also provides a very rich and opportune environment for metacognitive development (Thomas & Upah, 1996).

Programming instruction is used as an instructional environment to promote the above-mentioned higher order thinking skills (Palumbo, 1990:65) and the programming language as a medium for developing these skills. Although problem solving, programming methodology and support are the important issues, Deek and Kimmel (1999:98) believe that any discussions on Computer Science curriculum should consider the choice of which first language to use. Palumbo (1990:83) concurs that this choice remains one of the concerns of programming language instruction.

The language used in the 1990's in South African high schools to teach programming was

---

[2] The subject currently known as Computer Studies, like many other subjects in the Further Education and Training (FET) band of learning, is undergoing revisions with regard to content etc. A part of this process entails a name change for the new subject to Information Technology. Worldwide, various subjects with much the same content which specifically teach programming, can be found. Subjects' names will be used as they appear in sources.

Turbo Pascal (TP). Similarly, in Australia Pascal[3] was still being taught in 1995 (Kay *et al.*, 2000:110). However, towards the end of that decade, all sorts of questions regarding Computer Studies were raised:

> "Why are you teaching Pascal when industry requires C++ (or whatever) programmers?"
>
> "What are you doing about object-oriented programming?" (Chiles, 1999a.)

TP as a language was a bit out-dated (Shear, 2000). King (2000) has been involved in the computer industry for several years and has a large number of contacts in the industry. Many of these contacts, as well as some of the parents at the school where he teaches, had been shocked that TP was still being used. While he understood the problems associated with changing, he had great difficulty in convincing the parents that the teaching of TP would help their children later in life. Mayers (2000a) agrees with this line of thinking when he states that he doesn't think that teachers were being relevant by teaching TP.

Not only was TP considered an 'old' language, but at a conference held in Pretoria (South Africa) in May 1998, it was indicated that development on TP had stopped and that users would start experiencing compatibility and other problems (Chiles, 1999a). Many schools teaching TP had upgraded their equipment to Pentium IIs and above, and were now having difficulties running TP: it no longer operated properly on machines with faster processors, and needed to be patched. With regard to software developments, Windows 2000 didn't support DOS-based applications (Chiles, 2000c) such as TP.

Suppliers told other schools starting up with Computer Studies that TP for Windows and Borland Pascal with Objects were no longer available and that they would have to settle for TP version 7 for DOS, which would also shortly be removed from the market (Chiles, 1999a). This was confirmed by Darren Crowder, senior product manager for the Inprise Corporation, Borland Southern Africa (Crowder, 2000), as well as in a quote from the Borland/Inprise website by Breakey (2000): "Turbo Pascal for DOS is available only in Europe, while supplies last."

Pascal was developed as a language to teach programming and designed to encourage good programming techniques (Choi & Repman, 1993). Even though these properties made TP an extremely useful and effective medium for teaching problem solving (Tweedie, 2000), an acute awareness of Pascal's shortcomings had developed for several years (Kay *et al.*, 2000:110) and the language was obviously on its last legs. It was felt that a move was urgently needed and the search was on for a new programming language to teach in schools.

---

[3] It is not possible to determine in some sources whether TP or its older predecessor, Pascal, is being referred to. In such cases, as well as where the reference is specifically to the original language (see e.g. start of last paragraph above) Pascal is used and not TP.

As is the case for a number of other international programmes, the Interim Syllabus for Computer Studies (DoE, 1994) does not impose a specific language (Gal-Ezer & Harel, 1999:117), nor does it describe the syntax of any language (Westrom, 1992). The only specification of consequence states that a high-level computer language (Rogers, 2000) needs to be used. This paves the way for the future, where teams might be found developing courseware that uses other languages.

In order for input to be obtained from all quarters, a discussion on "Where to after Pascal?" was started on the comp-studies list (Chiles, 1999a), which is an electronic communications medium for Computer Studies teachers through which they can join in discussions on various issues pertaining to the subject (Chiles, 2000b). Although the list was originally set up for teachers in the Western Cape (one of the nine South African provinces), it now involves teachers from all over the country with a couple of teachers from Zimbabwe and elsewhere in the world. All examiners and moderators (both departmental and SAFCERT[4]) in South Africa, as well as many of the Subject Advisers for Computer Studies in the provinces subscribe to the list.

Participants in the debate were asked to contemplate what the most important points would be that needed to be considered when selecting and implementing a new language (Chiles, 2000k). Blignaut (2000) is convinced that the issue of changing to a new language has three equally important aspects:

- object oriented programming
- visual programming environment
- database connectivity

In the United States of America (USA) a move to a new programming language was also being debated. During a meeting of the Special Interest Group for Computer Science Education (SIGCSE) in 2000 a panel discussion was held on appropriate high-level goals for introductory programming courses (Walker, 2000), and three main principles seemed to emerge:

- emphasis on object-orientation
- a need for safety in a language and environment
- a desire for simplicity

Additionally, it is important that the decision making process followed is conducted in a scientific manner (Chiles, 2000c), by

- giving consideration to what learners need to be taught through Computer Studies and
- also taking into account issues such as support, costs and training.

---

[4] SAFCERT used to be the South African Certification Council. This organisation has recently been renamed the Umaluzi Council, which means 'shepherd'. In documents referred to in this thesis, the name SAFCERT was still relevant, and it will therefore be referred to as such.

The adoption of TP as programming language in schools was an easy one, as there were not too many contenders and it presented something that other languages may not have offered at that stage, for example, it was structured (Gibson, 2000). Thus, when it was realised that a new language was needed, the choice of such a language was found to be very difficult (Kay *et al.*, 2000:110): no selection criteria existed against which different candidate languages could be assessed, nor did guidelines exist for the way in which the selected language(s) could be implemented successfully. Although many opinions exist, as is clear from the preceding discussion, very little evidence of specific research into this area can be found in either national or international searches.

This study aims to establish such criteria and guidelines. In order to narrow down the field of research, the following questions that describe the problem were set for this study:

- What criteria should be considered when selecting a first programming language to teach in high schools?
- What guidelines can be introduced for the implementation of a first programming language in high schools?
- Are these selection criteria and implementation guidelines valid for the South African context?

1.3 AIMS OF THE RESEARCH

In light of the aforementioned research questions, the following aims were set for this study:

- To establish criteria that should be considered when selecting a first programming language to teach in high schools.
- To introduce guidelines for the implementation of a first programming language in high schools.
- To verify the validity of selection criteria and implementation guidelines identified empirically within the South African context.

1.4 IMPORTANCE OF THE STUDY

Because "computer technology develops at an alarming rate" (Barrow *et al.*, 2002:v) and the rate at which computer languages develops is much higher than was seen in the past (Gibson, 2000), the cutting-edge computer industry re-tools more frequently and programmers will cover many more languages in their lifetime than ever before. This implies that while learning practical programming skills in a particular programming language, it is even more important for the learner as novice programmer to develop a sound theoretical understanding of programming in general, and so to prepare for later learning future languages and environments. Learners must

be prepared "for lifelong learning that will enable them to move beyond today's technology to meet the challenges of the future" (Chiles, 2002).

The rapid changes in computer technology and programming languages mentioned at the start of the previous paragraph also leads Mayers (2000b) to the opinion that in five years' time the language issue will have to be re-evaluated and languages changed again. Criteria and guidelines established in this study could prove to be indispensable in such a case.

1.5 RESEARCH METHOD

1.5.1 Literature study

Through a literature study, using mainly electronic media, criteria to be considered when selecting a first programming language to teach in high schools, were established and guidelines for the implementation of such a language introduced. An HSRC-NEXUS search on current and completed research in South Africa in related fields and an ERIC-DIALOG search were undertaken. Additional searches were undertaken as needed. The following keywords or phrases were used:

programming language; computer science; Computer Studies; school education; teaching; learning; object oriented programming

1.5.2 Empirical study

1.5.2.1 Method

A field survey using a questionnaire was undertaken.

1.5.2.2 Target population and sampling

The population for the study consists of all role players in the curriculum process of Computer Studies, who can be involved in various sectors connected to the subject:

- Team members of the Writing Committee for the National Curriculum Statement (NCS) for Information Technology [N=7].
- Subject Co-ordinators representing their provinces with regard to the finalisation of the NCS and implementation issues [N=8][5].
- Examiners and moderators for Matriculation examination papers in all provinces [N=18].
- Curriculum planners and advisors in all provinces [N=7].

[5]A number of people are involved in Computer Studies in various capacities. Numbers presented here are such that persons already incorporated in previous structures are not counted again.

- Persons involved with Computer Studies at provincial level in an administrative capacity [N=7].
- Persons involved in the training of Computer Studies teachers at tertiary institutions [N=10].
- Teachers from all schools offering Computer Studies (HG and/or programming) [N=445].

All sectors of the population, apart from the last, were small enough to target all members. However, the number of teachers of the subject (last sector of population mentioned) would not only make it difficult to process all potential respondents, but the size of this sector of the population relative to other sectors is also disproportionately large. It was, therefore, decided to apply sampling to this sector, by contacting only teachers who act as markers for Matric papers. In three of the provinces in South Africa, the Eastern Cape, Free State and Northern Cape, due to small numbers of learners, all marking is done by one person each – in these cases, experienced teachers recommended by Subject Advisors or Chief Examiners were approached.

1.5.2.3 Research instrument

The questionnaire used contained Likert type responses and shorter answers to open questions. The aim was to ascertain empirically to what extent various role players in the South African context substantiate the authority of the selection criteria and implementation guidelines identified.

1.5.2.4 Statistical processing

Mainly descriptive statistics were used, but inferential statistics were also used to establish the significance of similarities and differences. The design of the questionnaire and the statistical processing of data were done in conjunction with the Statistical Consultation Service of the North-West University (Potchefstroom campus).

1.6 PROCEDURE

The following research procedure was followed:

- A literature study was conducted and the questionnaire set in accordance with the study.
- The necessary permission to have the questionnaire completed was obtained from the Subject Advisors (Computer Studies) in the various provinces.
- Role players were informed of the study and questionnaires were distributed, completed and gathered.
- The gathered data was processed statistically and the results were evaluated. Findings were reported and conclusions and recommendations were formulated accordingly.

1.7 ORGANISATION OF REMAINING CHAPTERS

In chapter 2 an introduction to terminology that will be used throughout the research is provided, with applicable explanation and discussion.

Chapter 3 presents research comprising the establishment of criteria that should be considered in the process of selecting a first programming language to teach in high schools.

Guidelines for the implementation of a first programming language in high schools are proposed in chapter 4.

The empirical study and its results are described in chapter 5, together with discussion and interpretation of these results.

In chapter 6 a final summary of research is presented. Selection criteria and implementation guidelines identified in chapters 3 and 4 are compared to results obtained in the empirical study, implications of the study for the selection and implementation of a first programming language in high schools are examined, as well as suggestions made for further investigations.

CHAPTER 2

PROGRAMMING IN CONTEXT

2.1 INTRODUCTION

According to Bill Gates, the international technology industry has already seen an unprecedented growth phase from 1998 to 2000 (Kok, 2003:3). Now, a new era lies ahead for the IT industry where developments of hardware will fuel a new growth phase that should continue until 2009. Gates however advises that for this growth to realise,

> "Our perceptions of the role that technology plays, have to change." (Brand, 2003:10.)

In order to facilitate an understanding of this process, this chapter supplies an explanation of applicable terminology, including programs, programming and programmers. Various types of programming languages are introduced, followed by a description of the role of abstraction in the evolution of programming languages.

Object-orientation is presented as another form of abstraction. The origins of object-orientated programming (OOP) are outlined, along with the basic characteristics of a pure approach to OOP. The key principles of OOP are delineated, succeeded by an investigation into the merits of OOP.

Definitions are provided for miscellaneous terms surrounding graphical user interfaces, visual and event-driven programming, and the way in which these improve the user's interaction with applications, is briefly sketched.

2.2 PROGRAMS AND PROGRAMMING LANGUAGES

At this stage it is necessary to place the discussion in context by taking a look at various terms that will frequently appear in what follows.

2.2.1 Programs, programming and programmers

Computer programs consist of an organized collection of statements (Eckel, 2000) that, when executed while the program is running (Barrow *et al.*, 2002:19), cause the computer to behave in a predetermined manner (Jupitermedia Corporation, 2003f). A program usually also contains a list of variables, and the set of directions tells the computer exactly what to do (Kamin & Reingold, 1996:1) with the variables, so that programs can be seen as the medium used to communicate with the computer. However, viewing a program only as a sequence of instructions for a machine presents a rather narrow outlook.

The much broader view should be taken in which programs are descriptions of values, properties, methods, problems and solutions. The role of the machine is to speed up the manipulation of these descriptions to provide solutions to particular problems. Computer programming can therefore be described as a term that not only refers to implementation, but to

the whole process of expressing a solution to a problem in the language of the computer (Thomas & Upah, 1996) that also includes analysis and design.

The first responsibility in programming is to clarify the problem, by asking certain questions:

- Exactly what is required?
- How can the task be broken down into manageable pieces?
- What algorithm is appropriate to solve the problem? An algorithm is a formula or sequence of step-by-step instructions representing a constructive solution to a particular problem (Deek & Kimmel, 1999:95; DoE, 2003a:30) without reference to technical peculiarities. To be an algorithm, a set of rules must be precise, unambiguous and have a clear stopping point (Bishop, 1998:11).
- How can this algorithm be turned into a program? In this context, programming (coding) may be considered as an implementation of an algorithm (Grigas, 1994).
- Does the program work?
- Is it written as clearly as it can be?
- Is it fast?
- Can it be changed easily if needed?
- Does it demand too great a fraction of the computer's resources?

These are the questions confronting the computer programmer, the person who designs and writes the program. The programmer's task consists of devising the appropriate step-by-step instructions, which when carried out by the computer, will accomplish the desired task (Schneider *et al.*, 2000:393). In order to achieve this, the programmer needs to

- describe what is to be computed
- organise the computation sequencing into small steps
- organise memory management during the computation (Joyner, 1996)

The most basic activities of programmers include editing source code, running the program to test it, and debugging the program (Van Rossum, 1999). This whole process of designing, writing and debugging a computer program is an exciting activity (Kamin & Reingold, 1996:xxiii). As exciting as it is, however, it is fraught with frustrations that come from needing to know an enormous number of details immediately, before even simple tasks are possible. It is an exacting and complex task, rich in subtleties that contribute both to the inherent excitement and to the frustrations.

2.2.2 Programming languages

Programs can be written in many different programming languages (Bishop, 1998:13). A programming language is an artificial, formal language for specifying sequences of directions for the computer (Kamin & Reingold, 1996:1). It provides a language of description which serves as

an interface between the models in the human mind and those in computing hardware (Goldberg & Robson, 1983:vii).

Computers are inflexible machines that understand what is entered only if it is typed in the exact format that the computer expects (Jupitermedia Corporation, 2003h). The definition of a particular programming language provides this expected form in terms of syntax and semantics. In the milieu of computer programming languages, the term semantics refers to the meaning of the language constructs (DoE, 2002a:33) which is used to differentiate the meaning of an instruction from its format (Jupitermedia Corporation, 2003g).

The format, which covers the spelling and grammar of programming language components and the rules controlling how components are combined, is called the language's syntax. Each program defines its own syntactical rules, which define how the various symbols of the language may be combined, and that control which words the computer understands, which combinations of words are meaningful, and what punctuation is necessary. For example, if a command is misspelled, it is a syntax error. If, on the other hand, a legal command is entered that does not make any sense in the current context, it is a semantic error.

Historically, programming languages have had a limited role, that of writing executable programs. As programs have grown in complexity, this role alone has proved insufficient, and many design and analysis techniques have arisen to support other necessary roles. According to Joyner (1996), a programming language should

- provide a formal, yet readable, notation to support consistent descriptions of systems that satisfy the requirements of diverse problems
- aid reasoning about the design, implementation, extension, correction, and optimisation of a system by supporting the exchange of ideas, intentions, and decisions between project members
- provide methods for automated project tracking and notations to formally document a system

Programming languages function at many different levels and have many roles. They should be evaluated with respect to those levels and roles, because each programming language was designed to meet specific needs and is appropriate for different types of applications and different types of computers. Because of these differences, a particular "language may be better suited than others for writing certain kinds of programs" (Schneider *et al.*, 2000:392).

One of the distinctions that can be made with respect to programming languages is that they can either be typed or untyped. Typed languages can further be statically typed or dynamically typed. Static typing ensures that only valid operations are applied to an entity, in other words, the compiler makes sure that all assignments are legal (Cook, 2003). In dynamically typed languages, type inconsistencies are not detected until run-time.

Programming languages can also be categorised as either general or domain-specific (Van Rossum, 1999). The domain-specific group contains everything from command line argument syntax to email headers and HTML. The distinguishing factor is the presence of a relatively narrow application domain. A typical property of domain-specific languages is that they provide excellent control in the application domain for which they were intended, but very little freedom in unanticipated areas.

The term general is used to include functional programming languages and logic programming languages to the extent to which they are usable as a general programming tool. General languages usually aren't as good in any particular domain. However, general languages make up for this through their Turing-completeness, which makes it possible to solve any problem that might come up (assuming availability of sufficient resources). General languages are therefore ideal when used in combination with domain-specific languages.

2.2.3 The role of abstraction in the evolution of programming languages

According to Aho and Ullman (1992:1) "an important part of the [Computer Science] field deals with how to make programming easier and software more reliable. But fundamentally, Computer Science is a science of abstraction - creating the right model for a problem and devising the appropriate mechanizable techniques to solve it." It can be argued that the complexity of the problems one is able to solve in a specific programming language is directly related to the kind of abstraction (what it is that one is abstracting), as well as the quality of the abstraction. As abstraction is clearly a fundamental concept in computing, all programming languages provide abstractions (Eckel, 2000).

In order to give a reflection of the progression in different levels of abstraction, programming languages are grouped into generations, starting with machine language (first generation) and assembly language (second generation). High-level programming languages comprise the third generation.

2.2.3.1 Machine language – first generation

Eventually, all programs must be translated into *machine language* that the computer can understand (Jupitermedia Corporation, 2003f). However, machine language was designed specifically with the bare machine in mind, which makes it complicated - it uses binary notation and only allows numeric memory addresses. These properties also make it difficult to understand and change, and to create data (Schneider *et al.*, 2000:240-241).

2.2.3.2 Assembly languages – second generation

The development of assemblers and *low-level languages* called *assembly languages* provided a small abstraction of the underlying machine. Low-level languages are still close to the language used by a computer, and use operators, which make them more like assemblers (Joyner, 1996).

On the positive side, they made symbolic naming and addressing possible (Coggins, 1996) and data items could now be assigned abbreviated names.

However, in assembly languages, a name could still refer only to one memory location. This is in contrast to the fact that a programming language should provide superior support for abstraction to the extent that it enables a single name to refer to a larger, more complex body of code (Zimmer, 1985:9,10). Therefore, individual assembly language statements, though easier to read than their binary machine language counterparts, can be no more powerful than the underlying machine instructions.

For the same reason, assembly language programs are machine-specific. The programmer still has to manually manage the movement of data items between memory locations at the low level of the machine-oriented deployment domain, which handles the particulars of how a specific computation is to be achieved. By specifying how things are to be done in one environment, portability to other platforms is hindered. Certain bookkeeping tasks also arise from having to specify how a computation is done, as a microscopic view of a task has to be taken by breaking the task down into tiny subtasks at the level of what is going on in individual memory locations.

Finally, assembly language statements are not English-language-like. Although operations are given mnemonic code words as an improvement over a string of bits, instructions remain rather stilted.

2.2.3.3 High-level languages – third generation

In order to overcome the deficiencies of assembler languages, *high-level programming languages* were created (Schneider *et al.*, 2000:296). In a high-level language

- the programmer has no need to manage the details of the movement of data items within memory or pay any attention to exactly where those items are stored
- the programmer can take a macroscopic view of tasks at the level of problem solving, so that the primitive operations used as building blocks in algorithm construction can be larger
- programs written in a high-level language are portable rather than machine-specific
- programming statements in a high-level language are closer to standard English, and uses standard mathematical notation

High-level programming languages function at the level of what it is that needs to be computed, which describes the problem domain. The most significant way in which high-level languages replace bookkeeping is by using a declarative approach. Additional advantages are also obtained, as automating the bookkeeping tasks enhances correctness, compatibility, portability and efficiency.

Many so-called procedural[1] languages (also known as imperative languages) that followed, such as C, Pascal, BASIC, Fortran and COBOL, were abstractions of assembly language. These languages are big improvements on assembly language, but their primary abstraction still requires one to think in terms of the structure of the computer rather than the structure of the problem one is trying to solve. The programmer must establish the association between

- the machine model in the solution space, which is the place where the problem is being modelled (such as a computer)
- the model of the problem that is actually being solved in the problem space, which is the place where the problem exists

The effort required to perform this mapping, and the fact that it is extrinsic to the programming language, produces programs that are difficult to write and expensive to maintain.

The alternative to modelling the machine is to model the problem one is trying to solve. Early languages chose particular views of the world, such as LISP - a functional programming language (Schneider *et al.*, 2000:412) which ultimately sees all problems as lists - and APL (all problems are algorithmic). PROLOG is a logic programming language (Schneider *et al.*, 2000:417) that casts all problems into chains of decisions. Languages have been created for constraint-based programming and for programming exclusively by manipulating graphical symbols. Each of these approaches is a good solution to the particular class of problem they're designed to solve, but as soon as one steps outside that domain, these approaches become awkward.

2.3 OBJECT-ORIENTATION

The object-oriented (OO) approach goes a step further in the pursuit of abstraction by providing tools for the programmer to represent elements in the problem space. The problem domain is modelled using object-classes (Maih, 1997) and the elements of the problem space and their representations in the solution space are referred to as objects. An object can represent any abstract or real world entity (Finch, 1998) that is relevant to the system, while "classes are a means for describing the properties and capabilities of the objects in real life that a program has to deal with" (Bishop, 1998:23). In this way, any conceptual component in the problem that is being solved can be represented as an object in that program.

The idea is that the program is allowed to adapt itself to the solution of the problem by adding new types of objects, so that when one reads the code describing the solution, one is reading words that also express the problem. In using this approach, object-oriented programming (OOP) offers a more flexible and powerful language abstraction than has been available previously, which allows programmers to describe the problem in terms of the problem, rather than in terms of the computer where the solution will run. A connection back to the computer still

[1] "A program written in a **procedural** language consists of sequences of statements that manipulate data items; that is, they change the contents of memory cells." (Schneider *et al.*, 2000:393.)

exists. Each object behaves like a small computer, having a state and operations that it can be asked to perform. This compares favourably to objects in the real world, which all have characteristics and behaviours.

2.3.1 The origins of object-oriented programming

The origins of OOP can be traced to a set of code packaging conventions that were enabled by advances in compiler technology and certain technical innovations in programming language design. These conventions, which supply a method for developing, organising, imposing and maintaining structure on medium to large scale software systems (Lambert & Nance, 1997:13) that involve several programmers (Kay *et al.*, 2000:110), provide several useful software structuring innovations (Coggins, 1996) that enunciate the intellectual foundation for what became OOP. The innovations

- established an organising principle for project decomposition, thereby minimising communication and dependencies among modules
- provided a useful definition of the module that is supported by syntactic structures in the programming language and checked by the compiler
- enforced a useful formal separation between architecture, implementation and realisation.

The term "object oriented programming" was largely borrowed from the SmallTalk community (Johnson, 1994). Smalltalk was the first successful object-oriented language that showed the usefulness of the OO approach in a programming environment. It is a dynamically typed language (Joyner, 1996) that provides a revolutionary combination of a graphical user interface[2] and a powerful and flexible interpretive programming environment[3] (Goldberg & Robson, 1983:viii). Eckel (2000) refers to a summary of five basic characteristics of Smalltalk, that represent a pure approach to object-oriented programming, which will be elaborated on in the following paragraphs.

2.3.1.1 Everything is viewed as an object

An object is a distinct entity that can be thought of as a compound variable that has a unique name or identifier, and basically bundles together its own state or data, and actions or behaviours (Standish, 1998:21; Barrow *et al.*, 2002:476).

An object's internal state is defined by the values of its attributes, which can be thought of as a collection of variables and data. The concept of attributes is central to object-oriented programming (Barrow *et al.*, 2002:3) and all OOP languages associate a particular set of attributes with an object.

---

[2] **graphical user interface**: see *2.4.1.3* for definition

[3] **Programming environment:** A set of tools for programming that is commonly provided with a computer's operating system (DoE, 2002a:33). Frequently these tools are supplemented by an application development system.

Objects not only store data, but requests can also be made to particular objects, asking them to perform certain operations. In object-oriented programming, a method is a procedure, function or routine definition that is executed when an object receives such a message (Standish, 1998:22; Jupitermedia Corporation, 2003d). It consists of a named sequence of programming instructions to the computer that tells the object how to perform a particular operation (Barrow *et al.*, 2002:40). A method is always associated with a class and attached to an object, as methods express the behavioural capabilities defined for a class and its objects as a set of predefined operations (Bishop, 1998:13). Methods therefore specify the way in which an object's data can be manipulated (Martin & Odell, 1992:17) to change its state.

2.3.1.2 A program consists of a collection of objects telling each other what to do by passing messages between each other to accomplish a task

These messages can be thought of as requests to call functions that belong to a particular object. When requesting a service, a method call invokes an object's method by sending a message to the object, in other words, invoking a member function of the class (Coggins, 1996).

2.3.1.3 Each object has its own memory made up of other objects

Put another way, a programmer can simply create a new kind of object by making a package containing other existing objects (Cook, 2003; DoE, 2003a:32). In this way complexity can be built into a program, while hiding it behind the simplicity of objects.

2.3.1.4 Every object has a type and is implemented as an instance of a class

Defining complex types is a central concept of object-oriented programming (Joyner, 1996). Classes are analogous to a derived type in a procedural language (DoE, 2002a:30) in that they offer a prototype for an object in an object-oriented language, so that "designing an object really means designing a class and its members" (Bishop, 1998:228). It is often useful to think of a class as a user-defined data structure[4], combined with the procedures that will operate on that data structure. An object class may also be considered to be a grouping of these objects that share a common structure and behavior.

The structure of a class is determined by the class definition that acts as an object's template, by furnishing descriptions of both:

- The memory structure of the class variables that represent the state of an object of that class.
- The behavioral components, consisting of the set of allowed operations (methods) associated with objects of the class.

The process of generating new objects during the running of the program is referred to as creating instances of the "blueprint" specified by its class. Put another way: "an object is a

concrete realization of a class description" (Bishop, 1998:23). Once an object has been instantiated from a class, the object's properties and its behaviour can be set (Barrow *et al.*, 2002:466). Each instance of a class can have its own attribute values, even though the whole class shares the set of attribute names (LACS, 2000:2).

2.3.1.5 All objects of a particular type can receive the same messages

This makes the kind of messages/requests that can be handled the most important distinguishing characteristic of a class, which is one of the most powerful concepts in OOP.

2.3.2 Key principles of object-oriented programming

Some of the notable advantages offered by object-oriented programming are related to three key principles of OOP referred to as encapsulation, inheritance and polymorphism (Miah, 1997). These advantages are obtained by defining systems of objects using these principles, together with various information-hiding properties (Eckel, 2000), the important idea of abstraction (Cook, 2003) and sub-classing, genericity or other forms of polymorphism, and overriding.

The object-oriented meaning of encapsulation refers to the process where classes enable a programmer to package the different parts of logically related abstract data structures (i.e. the declarative aspects of an object) and behavioural operation routines (definitions, methods and procedures that operate on data) together under a single name in the operational definition of one module by organising data and code into a class capsule (Joyner, 1996).

In this way encapsulation also provides the means to separate the abstract interface of a class, which presents the visible surface of the capsule and serves as interface for other objects to communicate with objects of a specific class (Chakravarty & Lock, 1997:122) from its implementation that is hidden in the capsule. An object thus has control over the way in which other objects can access its data and methods, and can hide anything that it wants to keep private from other objects. This allows for information hiding to be achieved, as access to the object's attributes is only granted through its own methods (Martin & Odell, 1992:17). A higher degree of abstraction for system management and integration is supplied, but it is an enhanced aspect of data abstraction, and not the defining property of object-oriented programming. When using data abstraction in conjunction with encapsulation, the data-hiding concept is supported and extended in a programming language by defining a complex collection of data as a single user-defined type with a single name.

Although the term object-oriented programming is often used as a synonym for data abstraction, it properly refers to the way in which commonalties among abstract data types are made explicit by using object-oriented design. Use of OO design permits class definitions to be hierarchically arranged (Barrow *et al.*, 2002:476), leading to the powerful inheritance programming technique.

---

[4] "In programming, the term **data structure** refers to a scheme for organizing related pieces of information." (DoE, 2002a:30.)

Wirfs-Brock *et al.* (1990:24) define inheritance as "the ability of one class to define the behaviour and data structure of its instances as a superset of the definition of another class or classes."

Programmers make use of the process of inheritance by abstracting all the common features into a high-level member class that represents the characteristics that are shared by all its descendants. A high-level object class can then be specialised into sub-classes (classes at lower levels in the hierarchy) by creating new sub-classes under the super class, which would inherit data and behavior (member functions) derived from the super/base class or their ancestors at higher levels, and then add characteristics of their own (Barrow *et al.*, 2002:469). In this way, inheritance allows code and data structure definitions to be shared among classes with similar structure (Coggins, 1996), leading to the possibility of existing code being reused. By expressing the common shared theme only once, using such programming techniques also saves effort, because programmers need to indicate only the individual differences that define the variations (Standish, 1998:23).

Abstraction also comes into play in the field of inheritance, where it "enables a class or method to concentrate on the essentials of ... its behavior and interface to the world, and to rely on the details being filled in at a later stage" (Bishop, 1998:267). Implementing abstraction in this way allows the programmer to focus on defining a class in terms of what is known about it, and to leave open an option to define additional versions of it later on (Bishop, 1998:268). These versions will inherit the properties and characteristics of the original class, and can therefore be smaller and neater.

The term polymorphism originates from the Greek words "poly morph" and suggests the capacity to appear in many forms (Jupitermedia Corporation, 2003e). In the context of object-oriented programming, the concept of polymorphism refers to a programming language's ability to allow different objects to react to the same stimuli (i.e. message) differently, depending on their data type or class, by redefining methods for derived classes. An object can operate interchangeably with an ancestor in relation to the attributes and methods it inherits from the ancestor (Barrow *et al.*, 2002:465).

Genericity allows the reuse of one standard class, while overriding allows a subclass to "decide to supply its own version of a method supplied by a superclass" (Bishop, 1998:280).

2.3.3 Is the investment being made in object-oriented programming worth it?

The power of mature technologies and paradigms is routinely underrated, often with disastrous results, or at least expensive side effects (Richfield, 2000g). This, however, does not seem to be the case with object-oriented programming: although its origins stretch back many years (Standish, 1998:21), its use has recently become widespread.

OOP has been described as being "in the 1980's what structured programming was in the 1970's" (Coggins, 1996). Structured programming, a statement-level code organising discipline and the associated stepwise refinement design method, were brought to general attention at about the same time as data abstraction. A particular pattern was observed for the integration of structured programming into the general programming world:

- At first it was just a topic for debate among programming language researchers.
- It became a topic of study in graduate school.
- It migrated into upper-division undergraduate courses.
- Finally it was introduced into introductory courses.

Now, many computer science students do not specifically know what structured programming is, because it has been fully integrated into how they learned programming. In much the same way, OOP is on its way to becoming the way programming is naturally done and taught, and is now beginning to be used in introductory programming courses at many universities.

According to Barrow *et al.* (2002:447) object-oriented programming has also become important, because it is one of the best approaches available for dealing with many problems facing software developers. The accessibility of object-oriented languages and techniques to the massive personal computer market has further intensified the interest in OOP. Many believe in it and maintain that they practice it (although they practice it differently), and products claim to support or use it, but little seems to be known about what it exactly is. A point has been reached where understanding what object-orientation involves is important to many system analysts, designers and programmers. Some over-claiming by enthusiasts and in advertising also tends to make some sceptics wonder whether the term object-oriented is anything more than a high-tech synonym for good.

Questions in this regard that need to be considered include:

- Why is the programming world embracing this technology?
- Why do they think it is better?
- Why has industry made the investment in OO?
- Are these reasons for schools to teach OO (Cook, 2000a)?

In order to address these questions, some of the most common claimed benefits of OOP will subsequently be critically discussed.

2.3.3.1 Naturalness of analysis and design

One of the frequently stated benefits of OOP is that it offers a natural, and therefore more understandable, way of thinking about problems and their solutions, which is assumed to be cognitively similar to the way human beings perceive and understand the real world. As an example of this way of thinking, Martin and Odell (1992:31) state that "the (OO) way of thinking is more natural for most people than the techniques of structured analysis and design. After all,

the world consists of objects." It is thus postulated that it should be more natural for developers and programmers to decompose a problem into objects and classification hierarchies, at least as compared to traditional structured languages.

However, when the design performance of novice programmers using object oriented, procedural and data design methodologies was compared, it was reported that novices performed best using procedural methods and worst using object-oriented methods. Miah (1997) also reports that both novices and experienced OO programmers generally had difficulties in appropriately structuring classes. Some of the key difficulties experienced by novices when using OO include difficulty in identifying classes, associating methods with appropriate classes and structuring solutions.

Problems experienced with the formation of hierarchies may be connected to the inheritance model of OOP, which tends to assume the world can be cleanly modelled as hierarchical classification and/or nested structures, or at least into mutually-exclusive divisions (a flat tree), when in fact this is often not the case (Jacobs, 2002a).

These observations show that

- decomposing a problem by identifying classes and articulating objects and functions is not easy
- designing a solution to a problem with an object-oriented language is not as natural as it would appear at first glance

The hypothesis that object-oriented systems are natural representations of the world is not confirmed. Although it appears that object models seem close to reality, they don't represent the world more directly than is the case for other modelling techniques.

2.3.3.2 Higher quality code

Another alleged advantage for OOP is higher quality code, as some mistakes made while using older methods are harder to make and easier to detect using OOP. Coggins (1996) points out that using OOP has the side effect of improving the average sophistication and quality of code produced by a typical programmer.

The net productivity of a programmer is also enhanced by

- teaching programmers good software design strategies and practices, such as explicit public interfaces, constructors and destructors, and decoupling and cohesion
- performing more comprehensive consistency checks on implementations earlier in the development process
- providing an easy-to-use, reliable mechanism for implementing dynamic binding as a consequence of inheritance

Johnson (1994), however, warns that basic code quality depends on intelligent design, an effective implementation process and aggressive testing. OOP does not address the first or last

stages at all, and falls short in the implementation stage. OOP also does not by itself make the coding task easier, as a significant simplification of the coding task requires environment and library support.

2.3.3.3 Impact on software development

OOP is the latest in a long series of technological developments in programming language and translator technology that have led to improved programming methodologies. Each new methodology was in its day a great conceptual breakthrough. Although object-oriented design offers an important enhancement to programming methodology, this is due to a normal evolutionary development of programming methodology, as opposed to OO design being a conceptual breakthrough. OOP is not new, it does not tender a new paradigm and it is not a revolutionary advance in the way software is developed.

The biggest gains in using OOP result from applying principles that are older than, and largely independent of, OOP. It also benefits from important miscellaneous programming language enhancements implemented in object-oriented languages, which are not really related to OOP, such as

- strong typing
- function overloading
- placing declarations anywhere
- comments
- reference arguments to procedures and reference return values
- constant declarations
- simpler, cleaner storage management using new and delete operators
- inline specifications

These enhancements make some OOP languages better than some of the older, non-OOP languages, independent of the OOP enhancements, but the benefits of these enhancements are often not distinguished from the benefits of OOP.

The use of new object-oriented programming languages can, however, have a long-term strategic impact on software development, as they make it easier and more productive to generate large serious high quality software products by providing capabilities for

- secure network programming
- multi-threaded program execution
- clean data-encapsulation
- easy definition of graphical user interfaces (Standish, 1998:21)

Johnson (1994), however, argues that OOP has little to offer in the area of GUI design and implementation. OOP also does not provide good answers to the problems of asynchronous input/output (I/O) or concurrent processing that occur in user interfaces (Coggins, 1996).

Proponents of object-oriented programming further argue that using techniques such as encapsulation and inheritance provide for a means of organising software systems that

- define effective modular software components that hide their internal details
- define, present and manage clean software and user interfaces to their users
- allow for later modification and customisation to suit the needs of software maintainers, leading to improved software maintenance (Standish, 1998:21)
- permit substitution of new data representations when needed for improving efficiency
- allow prefabricated software subsystems to be built in which, by filling in blanks, working subsystems can be generated with minimal effort (Eckel, 2000)

These measures not only vastly reduce the time and expense required to develop reliable software subsystems (Standish, 1998:23), but also reward effective designs with easier development and maintenance, and low performance costs.

2.3.3.4 Software reuse

The ability to support broad algorithmic software reuse is perhaps the principal publicised long-term benefit of OOP techniques over procedural programming techniques. Advocates of OOP maintain that this feature provides effective mechanisms to allow for software to be reused, as programmers are enabled to create modules that do not need to be changed when a new type of object is added (DoE, 2002a:32). To facilitate this reuse, well-designed objects are implemented in object-oriented systems as the basis for systems to be assembled largely from reusable modules, resulting in an improvement of productivity (Miah, 1997) that comes from code reuse. Martin & Odell (1992:31) confirm that the use of OO "leads to a world of reusable classes, where much of the software construction process will be the assembly of existing well-proven classes." "The construction of reusable, modular components" has become "the Holy Grail of programming" (Cook, 2003).

Jacobs (2002a) is however convinced that if programming code is specifically designed to be reused, it doesn't need object-oriented techniques to be effective. Very little research exists on whether non-OOP languages can be used such that reuse and flexibility can be increased to the same levels as properly used OOP. A study to investigate this problem should

- compare OOP to other techniques and languages, not just self-standing OOP successes
- be sponsored by an organization that has no vested interest in OOP technology
- use a wide variety of applications
- consider factors like existing training levels and curves
- use unbiased sampling techniques that do not bypass case successes or failures due to embarrassment, fear of obsolescence, public relations agendas or other social factors
- be carefully reviewed by secondary entities
- preferably be open to the public

2.3.3.5 Extensibility and modification

It is claimed that the OO approach to software development allows software to be extended and modified easily at later stages. Khoshafian (1990: 274) states that "object oriented programming techniques allow the development of extensible and reusable modules". Graham (1994:37) similarly notes that "inheritance, genericity or other forms of polymorphism ...improve the extensibility of systems."

The fact that encapsulation allows object classes to be modified, or even added to new systems without requiring additional modification to other classes in the system, also makes object-oriented programs easier to modify (Jupitermedia Corporation, 2002b). A component based software industry where classes can be purchased, and plugged in, is envisioned. Ideally, one would like to take these "off the shelf, customise them, and plug them together to construct an application, with a bare minimum of additional coding and additional compilation" (Cook, 2003).

There are several different types of modification of data structures required in OO software development of which the following four are equally important:

1. refining existing classes
2. abstracting from existing classes
3. composing classes from existing components
4. decomposing classes into new components

However, a definite lack of support for these modifications is reported by Fischer *et al.* (1995:89), in that only the first, and to a lesser degree the third, was effectively supported, with no support provided for the second and fourth. It is suggested that inadequate support for these types of modifications is perhaps due to a lack of understanding of the software evolution process.

The implication is that extensibility needs to be anticipated in advance and objects structured appropriately. This is not a major obstacle, since OOP requires most object behaviour to be declared in advance as part of encapsulation anyway, as a predetermined set of methods is associated with an object.

Object-oriented software is also harder to change than it should be, as some of the common changes required in the development of OO applications fundamentally change the abstractions embodied in existing object classes, and relationships among those classes (Miah, 1997). Similar problems are reported with maintaining relationships for newly added classes. The use of protocol coupling, where one class depends on the protocol of another class to work, implies that if the original class is changed or removed, the risk is run of ruining the dependant one. OOP offers weak support for relationships, is no better optimised to deal with dynamic feature relationships and changes than competitor paradigms, and in many cases seems to be messier (Jacobs, 2002a).

Although OOP may provide better extensibility support than procedural languages, it does not go far enough.

2.3.3.6 Encapsulation versus division of data and behaviour

In a pure OO model, no support is provided for ad-hoc queries, which where made popular with Structured Query Language (SQL) in relational systems (Jacobs, 2002a), as such support would conflict with encapsulation. By using encapsulation, OOP combines data and behaviour, while competing techniques, specifically procedural programming, separate these (Jacobs, 2002b).

The primary contention of OOP proponents regarding the integration of data and behaviour is that it allows one to swap behaviour in place of data without bringing about many changes. However, integration sometimes makes it hard to swap parts for compatible alternatives, and procedures are especially not easy to transfer.

Different programming languages and systems very often have to share data in the real world. If there is a division between data and behaviour, different programming languages can easily work with the same data. The integration of data and behaviour, on the other hand, makes sharing objects among different programming languages and paradigms complex. If the data is mixed in with methods, the programmer is stuck with the current OO programming language to interpret the data. It is therefore usually safer to keep large data pools separate from methods, and OOP should not be used when sharing data is important.

2.3.3.7 Unresolved problems and issues in object-oriented programming

One of the main weaknesses in the current formulation of object-oriented design and programming is that OOP tends to encourage bottom-up design and toolkits of tiny classes (Coggins, 1996). Although an object should in theory represent a real-world entity (Finch, 1998), most programmers think of objects as processing entities. This gives rise to the problem that most OO development is not truly object oriented, but rather programming with small pieces of predefined code. Programmers are also tempted to define classes in libraries based on what is convenient to implement, rather than based on the capabilities that are needed to advance systems' intended objectives.

Other unresolved problems and issues identified in connection with OOP include the following:

- OO is weak at defining problems in ill-structured discovery domains (Miah, 1997).
- Professed enhanced communications for OO is not true in ill-defined domains.
- OO does not provide adequate support for multiple and/or conflicting views.
- The OOP philosophy is code centric (Jacobs, 2002a) as it attempts to manage the complexity of the world with structures programmed into a programming language. This is used together with an approach that more easily allows the programming language code to manipulate and manage these structures (Jacobs, 2002b). While the OOP approach makes

it easier for application programmers to write the software to manipulate structures, a different approach would make most of these manipulations unnecessary, or far simpler.

- Viewing everything as objects (see 2.3.1.1) is not appropriate in the database domain (Miah, 1997) and OOP specifically does not map well to relational databases (Jacobs, 2002a). In this case, the division between data and behaviour in procedural/relational applications has software design benefits, as it allows a certain contract between the database and the application programmer as database user.

Many OO books, and even some admirers of OO, suggest that it only performs well under fairly ideal conditions (Jacobs, 2002a), which may include:

- Sufficient budget and incentives for good long-term planning.
- Good code viewer/editor tools.
- OO-friendly database system.
- Sufficient training and mentoring: While learning any OO language does not involve any revolutionary change, there are new concepts and features to learn and new mechanisms to master (Coggins, 1996). There is also a long learning curve before many people grasp the power of OOP, because its benefits can't really be taught, or at least understood, from a book. Colligan (1997) confirms that learning programming languages is difficult to accomplish directly from a book.
- Programmers must have realistic expectations of OOP.
- Sufficient understanding of the domain/task.

Although there is a great deal of promise in the object-oriented approach, it seems that object-oriented programming by itself is not adequate to easily solve all programming problems. In the light of this, some language designers advocate the combination of various approaches into multi-paradigm programming languages (Eckel, 2000).

There is also much of value in old-fashioned structured programming (Barrow *et al.*, 2002:v) and many programming fundamentals remain essential, irrespective of the approach one adopts. It seems prudent to introduce learners to valuable aspects from all applicable paradigms, since this knowledge will be important for many years to come.

2.4 GRAPHICAL USER INTERFACES, VISUAL AND EVENT-DRIVEN PROGRAMMING

Crucial obstacles to the usability and ultimate success of programming systems are found in the way users access the functionality of these systems, interact with them and understand the results of these interactions (Drakos, 1995b). Huge strides have however been taken recently in improving the ease of these interactions for the user. Terminology connected to these issues that will also reappear in consequent chapters, will now be briefly introduced and explained.

2.4.1 Graphical user interfaces

An interface is a boundary across which two independent systems meet and act on or communicate with each other (Jupitermedia Corporation, 2003c). In computer technology, there are several types of interfaces. For the sake of this discussion, only user interfaces will be explained further.

A user interface refers to those features of a program or computer, such as a set of commands or menus (Jupitermedia Corporation, 2003i) which govern the way people interact with the program or computer (DoE, 2002a:34). This interface usually consists of the keyboard, mouse and menus of a computer system, and allows the user to communicate with the operating system. In this way, the user interface acts as a language of interaction that "matches the human communication system to that of the computer" (Goldberg & Robson, 1983:vii).

A graphical user interface (GUI) is a program interface that takes advantage of the computer's graphics capabilities in terms of components such as windows, icons and pop-up menus to make the program easier to use (DoE, 2002a:31). GUI's have become standard on personal computers as they allow the user to "see the result on the screen exactly as it will look to the application" (Johnson, 1994).

Many programming languages currently being used support the easy and rapid development of graphical interfaces under operating systems such as Windows (Barrow *et al.*, 2002:v) by even beginner programmers. This not only allows users to derive a sense of achievement and motivation from writing good-looking programs, but also to concentrate on the principles and programming behind the user interface.

2.4.2 Visual programming

Visual programming refers to any system that allows the user to specify a program in at least two dimensions (Drakos, 1995a), and includes the use of graphical programming languages to create programs.

The definition of visual programming languages will be taken as those languages that allow programming with visual expressions (Drakos, 1995a). Visual programming languages may be further classified according to the type and extent of visual expression used. A graphic language adapted to its users provides them with a domain-specific visual vocabulary, which they can apply and understand immediately (Drakos, 1995b).

Visual programming environments provide graphic elements and visual user interface components that appear on the screen (Barrow *et al.*, 2002:262), and can be manipulated by the user in an interactive way according to some specific spatial grammar for program construction (Drakos, 1995a). A basic form is usually provided on which the user can place various components (Drakos, 1995c) that allow the user to interact with the program while it's running.

2.4.3 Event-driven programming

When GUI's are programmed, the user, and not the program, is in charge (Barrow *et al.*, 2002:262). The program often cannot be written to anticipate in advance what the next event will be and so it must be ready at any time to respond to any event that may occur. An event in this case refers to an action or occurrence detected by a program (Jupitermedia Corporation, 2003a). The events involved can be actions initiated by the user of the program interacting with a specific object in the GUI or connected to the application, such as clicking a mouse button or pressing a key, in which case it is known as a user event. System events respond to occurrences happening in the computer itself without any direct action by the user (Barrow *et al.*, 2002:252), such as running out of memory.

A style of programming that accommodates this is event-driven programming, which provides a natural and congruent design approach for a large number of modern applications (Barrow *et al.*, 2002:v). Such applications are said to be event-driven, because they are designed to respond to events.

Whenever an event is initiated while an application is running, an event handler is automatically executed in response to the associated event occurring (Barrow *et al.*, 2002:19,249). An event handler is a particular form of a more general programming structure called a procedure (Barrow *et al.*, 2002:248), function or method. This leads to event handlers also sometimes being called event-handling procedures or event procedures (Barrow *et al.*, 2002:7).

Programmers control how a program responds to particular events that it has been programmed to recognise (Barrow *et al.*, 2002:261) by inserting the necessary blocks of program code statements into the associated event handlers (Jupitermedia Corporation, 2003b).

2.5 CONCLUSION

An explanation of applicable terminology, including programs, programming and programmers, was supplied. Various types of programming languages were also introduced.

In order to give a reflection of the progression in different levels of abstraction, programming languages are grouped into generations, starting with machine language (first generation) and assembly language (second generation). High-level programming languages comprise the third generation.

Object-orientation was presented as another form of abstraction. The origins of object-orientated programming were outlined, along with the five basic characteristics of a pure approach to OOP:

1. Everything is viewed as an object (Miah, 1997).
2. A program consists of a collection of objects telling each other what to do by passing messages between each other to accomplish a task (Coggins, 1996).
3. Each object has its own memory made up of other objects (Cook, 2003; DoE, 2003a:32).

4. Every object has a type and is implemented as an instance of a class (Joyner, 1996).
5. All objects of a particular type can receive the same messages.

Object-oriented programming is a coding discipline consisting of a collection of mechanisms, rules and pragmas to govern the structure of small to medium sized bodies of code. It refers to a special type of programming where programmers define not only the data type of a data structure, but combines with the types of operations (functions/methods) that can be applied to the data structure (DoE, 2002a:32) to create re-usable objects. These objects are therefore self-contained entities that encapsulate both their own data and functions (Barrow *et al.*, 2002:466). In addition, programmers can create relationships between objects.

The term object-oriented is generally used to describe a system that deals primarily with different types of objects, and where possible actions depend on the type of object that is being manipulated (Jupitermedia Corporation, 2002a).

Some of the notable advantages offered by object-oriented programming are related to three key principles of OOP referred to as encapsulation, inheritance and polymorphism. Defining systems of objects using these principles, together with various information-hiding properties, the important idea of abstraction and sub-classing, genericity or other forms of polymorphism and overriding obtain these advantages.

When the question is posed whether the investment being made in OOP is worth it, it repeatedly seems that the features that make OOP unique are also the ones that cause the most problems (Jacobs, 2002a):

- Although it appears that object models seem close to reality, the hypothesis that object-oriented systems are natural representations of the world is not confirmed.
- Using OOP has the side effect of improving the average sophistication, quality and reliability of the final code (Johnson, 1994) produced by a typical programmer, and also enhances net productivity. Basic code quality, however, still depends on intelligent design, an effective implementation process and aggressive testing. Higher quality leads to faster development times.
- OOP impacts on software development in that it helps gaining economic advantage by building systems at low cost from reusable software components in a flexible, well-organised fashion.
- OO advocates claim that software developed using OOP is easy to extend and modify. Some types of extension needed in practice are however not supported.
- No support for ad-hoc queries is provided in pure OO models, as this would conflict with encapsulation.
- Viewing everything as objects is not appropriate in the database domain.

Finally, the way in which GUI's, visual and event-driven programming can improve the user's interaction with applications was illustrated.

Using programs, programming and abstractions in programming languages that enable programmers to create software that is easy to use and manage "would stimulate economic activity through the creation of a strong local capacity for information technology development and innovation" (Petje, 2003:3). "The aim is to improve productivity. If people's productivity can be improved by 10% to 20%, we can have a big impact on the economy" - Bill Gates (Brand, 2003:10).

CHAPTER 3

CRITERIA FOR THE SELECTION OF A FIRST PROGRAMMING LANGUAGE IN HIGH SCHOOLS

3.1 INTRODUCTION

Since computers have made their way into educational systems around the world, one of the predominant uses of computers, especially at the high school level, is the instruction of learners in various programming languages (Palumbo & Reed, 1991:343). Several languages are available for teaching programming to learners at this level (Choi & Repman, 1993), and many good arguments have been made for adopting any one of the many possible styles of programming in a first course.

The language of choice for introductory programming courses (Madison & Fifford, 2003:227) - the "mother tongue" - however remains controversial (Gal-Ezer & Harel, 1999:117), with the problem of selecting an implementation language all too often taking "on the character of a religious war that generates far more heat than light" (SIGCSE, 2001a). It is clearly of the utmost importance that the choice of a new language is made in a scientific manner (Chiles, 2000c).

The argument should address issues like the needs of the learners (Cook, 2000a), and the question of which of these programming languages would be most beneficial to learners' development (Palumbo & Reed, 1991).

As computer and software performance show rapid and spectacular progress (Mendelsohn *et al.*, 1990:176), the decision on which tool(s) to introduce into the classroom for teaching programming should be taken with great care: choosing the wrong implementation(s) could mean that not only its use, but also its teaching will be outdated in as little as five years' time.

It quickly becomes clear that even if organizational constraints, such as support, costs, training etc. (Chiles, 2000c) can be put aside, language choices must be based on many different criteria. The aim of this chapter is, therefore, to establish criteria for the selection of a first programming language in high schools.

3.2 REQUIREMENTS THAT CRITERIA NEED TO SATISFY

With regard to aspects considered in order to establish criteria for selecting a first programming language, the emphasis will be on eventually setting selection criteria that will be universal, in that these will have relevancy in all circumstances. Set criteria should also not date, but should still be relevant to select a first language in ten years' time. However, these criteria, although

general in terms of worldwide relevancy, should, in light of the context of this study, be specifically suitable in the South African (SA) context.

In line with these guidelines, aspects that should be considered within a universal, worldwide context will be discussed. Next, aspects regarding relevance and new tendencies in programming will be considered, followed by a review of aspects that have specific relevance for the South African context.

3.3 ASPECTS THAT SHOULD BE CONSIDERED WITHIN A UNIVERSAL CONTEXT

3.3.1 The nature and purpose of the subject Computer Studies

The discussion of the aspects that should be considered when choosing a first programming language will be placed in perspective by giving consideration to what it is that learners need to be taught through Computer Studies (Chiles, 2000c). The purpose of Computer Studies will be investigated, which should provide an outline of the significant matters in the subject and provide an avenue to explore various issues that need to be considered when changing to a new language.

Delport (2000) considers the primary purpose of Computer Studies at school level to be to lead learners to discover and integrate Information and Communication Technologies (ICT) concepts holistically:

"Computer Science is about man-machine interface boundaries, machine-software boundaries, code-model boundaries, engineering principles, architecture and team worker roles" (Delport, 2000).

This might make the field seem overpoweringly vast. The South African subject statement does narrow things down a bit when it states that the purpose of Computer Studies is to "enable learners to understand ... the principles of computing, through the use of a current programming language, ... and how these apply in their daily lives, in the world of work and in their communities." (DoE, 2002b:6.) The statement then elaborates that one of the ways in which this understanding is to be achieved is by providing learners with opportunities to "work competently in a dynamic computer-using environment", which includes the use of "problem solving approaches". Grigas (1994) agrees that the subject is a tool for thinking, reasoning and problem solving. The accent is clearly on problem solving skills, making it a problem solving discipline, centred on the development of solutions to problems (Van den Berg, 1990:30).

An important common thread for this discussion is hereby presented: the urgent need to "concentrate on problem solving" (A. Bezuidenhout, 2000b). But what type of problem solvers are involved in this situation, and "what sort of learner will study Computer Studies" (DoE, 2002a:10)?

3.3.2 The level and nature of learners

"... it must be remembered at all times that the users are learners. Their needs, their knowledge and their abilities are different from those of experienced programmers" (Mendelsohn *et al.*, 1990:196).

According to Palumbo (1990:67), one of the central questions in programming research focuses on the age range when learners' ability levels will coincide with the necessary cognitive skills, and they will have sufficient background, in order to effectively benefit from programming language instruction.

Most educational programming is done at the high school level. This means that the majority of learners most often exposed to programming language instruction are between 15 and 18 years of age (DoE, 2002a:10). With regard to the developmental aspects of these learners enrolled in programming instruction (Palumbo, 1990:84), it is important to remember that such learners are maturing cognitively as sophistication of thought processes are expanded. At this level, learners are further developing abstract thought as they concentrate on thinking in abstract terms. Problems presented to learners for solution should therefore effectively make use of, and extend, these learners' cognitive development levels (Eccles & Midgley, 1989:141), as learners can analyse events and develop some understanding of higher level thinking skills with appropriate support.

3.3.2.1 Developing and supporting higher order thinking and problem solving skills

Higher order thinking is a term used to describe when a person is engaged in sustained cognitive effort directed at solving a complex problem (Norris & Jackson, 1992) using such skills as planning, problem decomposition and systematic elimination of errors (Singh, 1992). In contrast to guided learning, higher order thinking involves self-regulation of the thinking process, as well as the application of multiple, conflicting criteria. The quality of problem solving is related to the tendency to reflect on the validity of problem solving when several alternative solutions are available, and a correct solution is not immediately obvious (Van Merrienboer, 1990). Using higher order thinking contributes to this quality, as it yields multiple solutions, each with distinct costs and benefits, rather than unique solutions (Deek & McHugh, 1998: 172). For that reason, learners' growing abilities to use higher-level cognitive strategies should be supported and further developed, so that it can be utilised for further learning.

Research shows that young children do not demonstrate significant improvement in higher order thinking skills when exposed to programming instruction. One of the reasons for subjects' failure to show improved results may centre on their stage of development: many of these learners would be classified as concrete operational (Palumbo, 1990:84), which implies that they are more capable of acquiring practical skills than learning theoretical concepts (Dagiene, 2003:180).

On the other hand, high school learners are at, or developing capabilities towards achieving, the formal operational stage (Choi & Repman, 1993). It has been shown that such learners can significantly benefit from a programming language treatment, in that their ability to solve problems can be increased through systematic exposure and interaction with programming languages. The importance of this finding is emphasised by the fact that logical thinking and problem solving skills are some of the characteristics that learners who are most likely to succeed in the subject, should show evidence of (DoE, 2002a:12).

Significant changes in problem solving skills may also not be possible for young children, as they do not benefit from a long learning history (Palumbo & Reed, 1991). They have thus not acquired much prior knowledge about a variety of problem solving domains to which to transfer the problem solving strategies used in programming. Without such a history, it seems unlikely that transfer of training is likely to occur. Some of the success shown in problem solving transfer in high school learners may indeed be attributed to the participants' more extended history of dealing with a variety of problem solving domains.

These findings not only provide support for continued programming language instruction at high school level, but also offers a means by which the problem solving skills for the group most often criticised for their lack of problem solving ability, can be improved (Palumbo, 1990:86).

3.3.2.2 Developing and supporting critical thinking

It is also important to get learners at this development stage focused on critical and creative thinking skills (DoE, 2002a:10), as a person's ability to think critically is not only recognized as an important and pervasive educational outcome, but it is also known to be a key success factor in many kinds of occupations.

Many research studies dealing with programs aimed at teaching critical thinking skills indicate that while there is some success when critical thinking skills are taught directly, there is little transfer of the problem solving skills learned in one discipline to problem solving in another discipline. However, critical thinking is indeed enhanced by completing a programming course (Norris & Jackson, 1992).

As concern grows for the promotion of critical thinking skills among learners, so does the need for careful investigation into the instruction involved in computer coursework (Bernardo & Morris, 1994).

3.3.3 The nature of Computer Studies instruction

3.3.3.1 The role of programming in the subject

One of the most hotly debated questions in Computer Science education is the role of programming in the introductory curriculum (SIGCSE, 2001a). To give an indication of the range of this debate, Deek and Kimmel (1999:95) contend that programming is "an important topic and

must be part of any high-school computer science curriculum". In direct contrast to this statement, Choi and Repman (1993) quote a question by Wiburg: "Does programming deserve a place in the school curriculum?"

In order to shed more light on the issues in the debate, Thomas and Upah (1996) suggest that programming instruction should receive careful study to determine

- its educational value
- appropriate outcomes that computer programming should enable learners to reach
- instructional methodologies with which programming should be taught in order to accomplish the identified outcomes

Programming instruction poses many educational opportunities because programming is a constructive, inventive activity (Wirth, 1986:11). However, acquiring and developing knowledge about programming is a highly complex process (Rogalski & Samurçay, 1990:157) requiring high-level skills that can be difficult to learn (Van den Berg, 1990:38). If programming is indeed so difficult to learn and/or teach, why should it be taught at school level?

- Programming is an important skill, as an individual who is in control of the computer has a powerful problem solving tool at hand (Thomas & Upah, 1996).
- Knowledge of programming provides a necessary background for dealing with issues of modern society. It not only enables efficient learning and effective use of software tools, but also empowers the learner/user by providing useful and novel ways of thinking about problems.
- As a result of programming instruction and practice, learners develop and gain general cognitive skills that are identifiable and transferable to other situations (Mendelsohn *et al.*, 1990:179). More specifically, learning computer programming develops and significantly increases learners' problem solving abilities (Choi & Repman, 1993).
- In terms of the development and promotion of metacognitive skills, learning to program can also serve as a source of self-knowledge - a mirror for watching one self solve problems (Mendelsohn *et al.*, 1990:195).
- Norris and Jackson (1992) justify the teaching of computer programming in terms of applying knowledge to other fields, as a person's capacity to acquire new knowledge and skills, and to use these in problem solving situations, is improved through completion of a computing course that includes programming.

Some of the issues regarding the educational value of programming instruction have now been addressed. Next, the outcomes that should be kept in mind when programming languages are used at school (Mendelsohn *et al.*, 1990:176) will be considered.

3.3.3.2 The outcomes envisioned when learning a first programming language

A programming language is considered as a medium that creates new ways of dealing with existing knowledge (Mendelsohn *et al.*, 1990:179) and using programming languages as a medium for developing problem solving skills (Palumbo, 1990:67). In this scenario, the computer language offers learners "a programming environment rich with opportunities for problem solving" (Singh, 1992) where programming language instruction is used as an instructional environment to promote higher order thinking skills, such as problem solving (Palumbo, 1990:66).

When using instruction specifically designed to facilitate problem solving transfer from programming language experience, the types of skills likely to be transferred will be those that centre on specific components and attributes of the programming language features (Palumbo & Reed, 1991). If learners are therefore instructed in a particular first programming language, especially those training components that involve very similar concepts could more easily be used in near[1] transfer to a new programming language environment (Palumbo, 1990:70-71). Learners should consequently be able to adapt readily to other common programming languages and environments (Barrow *et al.*, 2002:1). Gibson (2000c) agrees that if a learner is taught any language in an appropriate fashion, it must be easier to move on to a second language, as opposed to learning a first language.

Programming languages are used to write programs, in order to solve problems, and the subject should thus be moving towards a language that would encourage more, if not all, learners at school level to participate in the fundamentals of problem solving, using a programming language (Janse van Rensburg, 2000). Although the acquisition of basic programming language features is necessary to use a language, it is not sufficient for complex problem solving (Husic *et al.*, 1989:571) which requires abilities beyond the mere command of the syntax and semantics of a programming language (Deek & McHugh, 1998:130). Therefore, the role of problem solving in programming instruction should be examined.

3.3.3.3 The role of problem solving in programming instruction

The instruction of learners in various programming languages centres on the professed relationship between programming language instruction and the promotion of higher order thinking skills, such as problem solving (Palumbo & Reed, 1991:343). Problem solving and higher order thinking skills, as well as critical thinking, reasoning and logical thinking, are often used interchangeably to refer to a person's ability to analyse a problem situation and come to an appropriate conclusion or solution (Norris & Jackson, 1992).

[1] Transfer of skills and expertise to a new problem solving domain that is similar to the domain in which proficiency and skill have already been obtained and established.

Problem solving and design skills are not only documented as essential for college and university (Stephenson, 2002), but also seemed to be the "number one on the wish-list" of tertiary institutions as to what learners should learn at school (Wasserman, 2000a). In terms of the workplace, problem solving ability is a necessary precursor to success in most professions, as well as in everyday living (Norris & Jackson, 1992):

"The expectation of today's computer users is that the whole computing community will assist in the solution of day-to-day problems" (Bishop, 1998:xix).

As the role of problem solving in programming is extensive (Deek & McHugh, 1998: 172) and problem solving skills are absolutely key to program development, these must form an integral part of the body of knowledge a learner acquires when first learning programming (Deek & McHugh, 1998: 162). Any comprehensive approach to programming instruction should clearly involve instruction in problem solving (Thomas & Upah, 1996) where learners study suitable and relevant problems with the outcome of developing problem solving skills (Chiles, 2001).

Polya was the first to describe a problem solving model based on classroom experience (Stephenson, 2002). In his now famous book *How to solve it*, he gives a popular version of his work on techniques for solving problems in mathematics, which is highly suggestive for programming (Weinberg, 1971:178). He recommends a heuristic strategy that consists of four steps (Bernardo & Morris, 1994):

- figuring out / understanding the problem
- devising a plan
- carrying out the plan
- looking back

Programming also involves a series of different sequential steps or stages (Thomas & Upah, 1996; Dagiene, 2003:181). These steps comprise an extended restatement of Polya's method for solving problems, combined with various elements from the software engineering process (Stephenson, 2002). Using the steps in problem solving is one of the primary outcomes for the subject (Chiles, 2001) which empower learners to

1. analyse the problem to be solved into fundamental steps, by developing models of its conditions and requirements
2. solve the problem in a way that can be expressed in a computer language, by preparing a design of the steps needed in a set of instructions (develop an algorithm)
3. verify the algorithm in terms of not only correctness, but also effectiveness and efficiency
4. translate the algorithmic solution by writing a program (Deek & Kimmel, 1999:95) which expresses and implements the solution in a language that the computer can ultimately execute
5. debug computer solutions for algorithms by isolating and removing syntax and logical errors
6. test and validate the program, both for valid and invalid data

7. assess and review the solution program by using techniques such as walkthroughs and peer reviews, to ensure accuracy
8. reflect on the process
9. produce adequate internal and external documentation, both online and written (DoE, 2002a:61), which makes maintaining the program easier over time

This process for problem solving as well as basic concepts and techniques, such as the principle of dividing a problem into parts (commonly known as the "divide and conquer" technique), should definitely be covered in instruction (Deek & Kimmel, 1999:95; Dagiene, 2003:181). Learners should not learn these programming principles by-the-way while developing programs for homework (Van den Berg, 1990:30), but because it is explicitly taught.

It is also important that learners are taught problem solving strategies that can be used not only in programming, but also in other areas (Van den Berg, 1990:31). The ability to accommodate and apply assimilated information to new scenarios (Scordilis, 2000b) and use problem solving skills learned in one context in another, is known as transference or generalisation (Norris & Jackson, 1992). Transference is, however, not easily achieved, and especially difficult when processes are not appropriate.

When the improvement of problem solving and critical thinking abilities is a desired outcome, justification of programming instruction is strengthened by the fact that instruction in programming gained in a computing class both improve these abilities and make transfer of the abilities to other multiple problem situations possible (Norris & Jackson, 1992). Some of these results might be explained by evidence found in the literature of similarities between the thinking processes involved in problem solving and those thinking processes employed in programming the computer (Bernardo & Morris, 1994). The fact that instructional emphasis on thinking processes, instead of the program produced, promoted transfer of learning, can also contribute to the success of this kind of research.

SIGCSE (2001a) believes that it is possible to develop introductory experiences that accomplish outcomes such as those described above, by

- not only introducing learners to fundamental problem solving concepts, but also by
- facilitating the development of cognitive models for these concepts, and
- encouraging learners to develop the skills necessary to apply the conceptual knowledge.

Considering the ways in which learners use their own knowledge (Van den Berg, 1990:39) could serve as a starting point. They should learn to explore and devise their own understanding and representation of the use of problem solving heuristics and techniques (Deek & Kimmel, 1999:95) such as sub-goals, reflection, trial and error, and regular patterns of analysis and synthesis (Bernardo & Morris, 1994). When learners are appropriately exposed to computers in an instructional setting (Palumbo, 1990:65), they develop various new and necessary cognitive skills, such as the ability to discover relationships, observe structure and

develop plans for solutions (Van den Berg, 1990:37). As problem solving experience accumulates, systematic thinking will give way to an intuitive understanding of concepts, together with the formulation of rules and strategies for solving problems (Singh, 1992).

These changing, ever-improving abilities are both facilitated and encouraged by the changing cognitive demands placed on learners as they learn to program. When teachers integrate problem solving theory with their instruction, they are able to guide their learners much farther along the so-called "chain of cognitive accomplishments" (Husic *et al.*, 1989:570).

When implementing the chain of cognitive accomplishments, teachers need to

- emphasize a strong foundation and adequate training in specific language features of the programming language (Palumbo, 1990:78-79).
- focus on instructional practices and experiences that foster the development of design skills for effective and efficient programs, based on combining and clustering commands and language features to solve specific programming problems (Husic *et al.*, 1989:571). These design skills also represent a major motivational and conceptual turning point in the acquisition of programming knowledge.
- give learners practice in learning to solve more complex and difficult problems, by using procedural skills such as planning, testing, and reformulating their solutions, and also by incorporating templates into their thinking, especially when they learn to develop, refine and apply their own templates.
- expect learners to generalize their knowledge to novel problems (Husic *et al.*, 1989:572).

Critical components for fostering general problem solving skills include the choice of programming language, explicit attention to design and debugging methods, and the kinds of programming tasks assigned to learners (Palumbo, 1990:70). The extent to which experience with general problem solving skills will begin to transfer the knowledge gained in the programming language environment to other, non-programming domains (Palumbo, 1990:78), and provide a framework for use in novel problem situations (Palumbo, 1990:66), are critically connected to learners' abilities to monitor their own progress, reflect on alternative approaches for solving complex problems, and autonomously seek information (Husic *et al.*, 1989:571).

3.3.3.4 The role of self-regulation and metacognition in learning (and) programming

The types of activities mentioned above are classified as self-regulated/autonomous learning, which refers to the cognitive and self-management procedures learners deploy while attending class, doing assignments, and studying for tests (Rohwer & Thomas, 1989:584). Included are those activities that

- facilitate comprehension, memory, or content integration
- serve to effectively manage time and effort
- support problem solving

Instructional practices are therefore needed to encourage this self-regulated approach to solving problems. Explicitly training learners to apply the specific types of self-regulation and self-explanation strategies found to be used by high performers, can improve learners' study strategies (Bielaczyc *et al.*, 1995:247), which in turn can improve the performance of learners on problem solving tasks (Bielaczyc *et al.*, 1995:221).

As an illustration of the difference between the way that less able learners and high performers use self-regulation in their approaches to solving problems, lower ability subjects

- attempt to remain in contact with a problem's details
- often examine the details of a problem at length before discovering its purpose
- regularly employ trial-and-error as a problem solving approach (Madison & Fifford, 2003:226)

To these learners, trial-and-error methods of identifying effective means of attacking programming problems are evidently important (Rohwer & Thomas, 1989:592).

In contrast, more able learners

- discern the general purpose of a program before examining the details
- show little evidence of relying on trial-and-error as a problem solving approach

This attitude agrees with that of expert programmers, who contend that the use of trial-and-error to locate effective approaches to solving programming problems, is much less productive than planning ahead.

To foster and encourage such outcomes as autonomous learning (Husic *et al.*, 1989:581), teachers must also achieve a delicate balance between providing opportunities for independent problem solving and modelling explicit problem solving strategies. Programming teachers do indeed differ widely in the extent to which they model strategies for analysing programming problems and planning solutions to these (Rohwer & Thomas, 1989:584). Too little teacher guidance frustrates learners, while too much teacher-imposed structure reduces assignments to uninteresting, step-by-step translations of instructions in syntax, with the result that learners are less attentive and involved (Husic *et al.*, 1989:570). Rigid structuring of computer work may also increase engagement in dysfunctional preservation (Rohwer & Thomas, 1989:592). On the other hand, a heavy emphasis on writing programs may discourage engagement in analytical planning, an activity that appears productive in less difficult courses.

During the programming process, the learner is led not only to be self-regulating, but also to understand his or her own thought processes, and be involved in metacognition (Bernardo & Morris, 1994). Metacognition refers to learners' awareness and knowledge of their own cognitive processes, their associated products, and how these are related to learning outcomes and the ability to evaluate and control the thought processes (Goosen, 1999:20).

Learners should possess the metacognitive awareness and managing strategies to monitor and direct their own studies and problem solving (Thomas & Upah, 1996; Goosen, 1999:23). These strategies govern the activities that learners engage in to manage or enhance their own learning as they carry out the work of their courses (Rohwer & Thomas, 1989:584) by

- facilitating the learning of programming constructs (Pirolli & Recker, 1994)
- guiding the use and application of programming knowledge in the construction of computer programs, by invoking lateral and metacognitive thinking, which are both critical to successful problem solving (Bernardo & Morris, 1994)
- providing a sound platform that enables learners to use and transfer the knowledge and skills developed in learning programming, to similar learning and problem solving tasks in other areas of interest (Scordilis, 2000b)

Results by Pirolli and Recker (1994:235) suggest that the acquisition of cognitive skills is facilitated by high degrees of metacognition, which includes higher degrees of monitoring states of knowledge, more self-generated explanation goals and strategies, and greater attention to the instructional structure. In the light of these findings, the primary instructional focus in first programming courses should include building metacognitive strategies (Thomas & Upah, 1996) and developing metacognition by emphasizing the processes of reflection, articulation and exploration (Kirkwood, 2000: 509). Especially the metacognitive component of problem solving should receive direct attention during instruction, and care should be taken that assignments are selected that would not only emphasise metacognitive behaviour, but would also show the need for this behaviour (Fernandez *et al.*, 1994:196).

3.3.3.5 The role of problem solving in the errors learners make when programming

Although understanding the constructs of programming languages is a crucial link in the cognitive chain of learning programming (Madison & Fifford, 2003:217), these constructs are not the biggest stumbling block for beginner programmers, but instead the compilation and coordination of the components of the program (Van den Berg, 1990:31). Researchers studying novice programmers have found that most faulty answers arise from a systematic application of knowledge the learner already has, and not simply as a result of slips in the mechanics of program construction (Madison & Fifford, 2003:218). Concentrating on the mechanistic details of programming constructs does, however, often leave learners to figure out the essential character of programming through an ad hoc process of trial and error (SIGCSE, 2001a). In order to facilitate the development of an overall understanding of effective program development in learners, analysis and design methodologies and problem solving skills should be learned.

Some of the most fundamental obstacles to learning programming are also related to problem solving (Stephenson, 2002), and programming teachers should refrain from interpreting

program errors as simple carelessness or a lack of attention or concern on the learner's part (Madison & Fifford, 2003:226). It has been established that many of the difficulties learners experience in introductory programming can often be attributed to a lack of problem solving abilities (Walker, 1996:4). Errors in learners' programs are also commonly related to deficiencies in problem solving strategies and insufficient planning, and not just to syntactical misunderstandings (Deek & McHugh, 1998:131). Even studies that specifically address the difficulties novice programmers have with syntax, have concluded that greater emphasis is needed in teaching planning and design strategies. These research findings emphasise the fact that problem formulation, planning and design are essential prerequisite tasks to coding and testing language syntax (Deek & McHugh, 1998: 160).

However, in direct contrast to these findings, most introductory programming courses, throughout the history of the discipline of computer science, have focused primarily on the development of programming skills (SIGCSE, 2001a), syntax and the particular characteristics of a programming language. These courses also often oversimplify the programming process to make it accessible to novice programmers, and pay too little attention to design, analysis, and testing relative to the conceptually simpler process of coding.

The superficial impression learners take from these experiences of programming skills mask fundamental shortcomings that will limit their ability to adapt to different kinds of problems and problem solving contexts in the future. Concentrating on the relatively unimportant details emphasised in these types of courses, rather than the underlying algorithmic skills, also means that many learners fail to comprehend the essential algorithmic model that transcends particular programming languages.

In a programming course for the new century, the focus should therefore move from studying programming via the explanation of various details pertaining to the language syntax of a specific programming language, to include techniques for problem solving (Van den Berg, 1990:90; Bishop, 1998:xix), as well as the identification of models that focus on algorithmic concepts (SIGCSE, 2001a).

3.3.3.6 The role of algorithms in problem solving

Learning about algorithms not only develops learners' aptitude for problem solving, as they learn to analyse, specify, and formulate problems, but by mastering algorithms, broader goals are also achieved, including problem solving and thinking skills (Grigas, 1994).

Most definitely, courses that do not deal with the design and implementation of algorithms, or with the components that underlie the implementations of these, cannot pretend to instil CS education (Richfield, 2000f). Learners should be led to examine abstract material on classic algorithms for searching and sorting arrays in internal memory (Blignaut, 2000), data definition, relevant hardware principles and the like. According to the South African subject statement

(DoE, 2002b:7), algorithmic design will indeed remain one of the areas included in Computer Studies.

M.L. Bezuidenhout (2000a) is of the opinion that irrespective of the language being used, there would be only one way of doing (for instance) bubble sort or selection sort. Richfield (2000d), however, argues that there are several techniques, and that the choice of algorithm and hardware makes a difference, because learners need to be made aware of the circumstances under which the different applications should be used.

However, learners who are only introduced to the abstract aspects of what computers or systems do, end up with strange pathologies of why things work, or fail to work, or work badly. If learners are just taught the kind of elegant algorithmic theory that is language-independent, then a gap is left in terms of practical ability.

Computer Studies is an applied subject. It does have its theoretical underpinnings, but if a matriculant is unable to achieve any practical objective on a computer system, then Richfield (2000c) does not see how teachers can claim to be doing what they are supposed to. There are certain concrete skills that can be taught and must be taught if the curriculum is to mean anything (Richfield, 2000a). Design and practical skills should undoubtedly be borne in mind (Richfield, 2000g), together with the need to effectively teach these skills, as well as programming structures and style (Chiles, 2001).

3.3.3.7 Didactical principles involved in the teaching and learning of the subject

"Beginning programmers ... often have only a vague notion of what programming entails and surprisingly little experience that prepares them to undertake it." (Thomas & Upah, 1996.) In the domain of programming, these learners are therefore not only faced with the need to both learn and recall the exact syntax of programming commands in the new programming language and determine how to effectively combine commands (Bielaczyc *et al.*, 1995:248), but must also quickly build a strategy for directing their understanding of the discipline. Teachers therefore need to design introductory classes to accomplish outcomes characteristic of learning any programming language for the first time (Husic *et al.*, 1989:572).

Given a field that changes as rapidly as computer science, pedagogical innovation is necessary for continued success (SIGCSE, 2001a). Because of this, pedagogical approaches are in considerable flux, as teachers and researchers in the programming field define and assess current teaching practices and introduce alternative approaches to programming instruction (Husic *et al.*, 1989:570). Instructional practices vary from unguided discovery to explicit description of problem solving procedures or lectures on abstract programming concepts. A pedagogical agenda is needed to enable Computing Studies teachers to adopt teaching approaches and methods at schools which are based on the development and enhancement of

learners' general capacities to solve problems, and to use creative thinking and learning habits (Kirkwood, 2000:519; Grigas, 1994).

It could be argued that problem solving, higher order thinking and metacognition are already present to some degree in examinable Computing Studies courses, but may not be reflected in teaching since course documentation contains no advice on teaching methods (Kirkwood, 2000:519). This lack in instruction presents a major dilemma, as studies about learning programming indicate that the attainment of higher level objectives, such as transfer and increased problem solving abilities, are highly dependent on instructional methodology (Palumbo, 1990:67,73).

Traditionally, introductory programming instruction has been a teacher-directed process of concentrating on presenting the language features, the syntax and semantics, and the function of statements. A typical instructional sequence might include acquiring declarative knowledge from reading instructional materials (Recker & Pirolli, 1995:4) which can contain text and show example programs, in which the statements are used. The example solutions typically found in programming instruction, however, often present a concrete set of actions or completed code, with little explanation for the underlying rationale (Bielaczyc *et al.*, 1995:248).

Learners will then probably be supplied with a programming assignment to complete (Thomas & Upah, 1996) for solving conventional programming problems, and are usually free to choose their own problem solving strategy. In this instructional strategy, learners are primarily engaged in the generation of small computer programs (Van Merrienboer, 1990) in which they apply recently learned language features. Especially lower ability learners however tend to make many errors while programming, due to an inability to construct coherent explanations of new material.

Although some instruction in such introductory programming courses introduce new language features, teachers focus primarily on skills development (Husic *et al.*, 1989:572). These so-called "skills courses" date from a time when programming was regarded primarily as a tool (SIGCSE, 2001a) and implement traditional instruction that has been designed to specifically develop skills, perhaps at the expense of understanding (Madison & Fifford, 2003:227).

Nieuwoudt (1998:65) however emphasizes that classroom instruction is effective to the extent that it purposefully makes possible and facilitates relevant and meaningful learning with comprehension.

How can problem solving and programming be effectively taught to learners (Deek & McHugh, 1998:130)?

For effective learning to take place, Polya (1981: 103) advises that learners should

- be interested in the material that is being learnt. Maintaining strong and ongoing student interest in learning, results in the development of problem solving skills and program design

skills (Grigas, 1994). It is the responsibility of the teacher, as the 'salesman' of knowledge, to convince learners that the point under discussion is, in fact, interesting, and that the problem that they are supposed to work on, deserves their effort (Polya, 1981: 105).

- find enjoyment in the learning activity. Learners should "enjoy working with the programming language" (DoE, 2002b:82), which should be inspirational to both teachers and learners.
- discover as much of the learning content, as is possible under specific circumstances, themselves.

This last point should not be seen as an endorsement of the many anecdotal assertions in favour of allowing learners to learn programming on their own. Mayer (1994: 983) sites a range of research results that established that learners, who work on their own in non-directed, practical instruction programs, are often not successful in discovering even the basic principles of programming. Discovery model instruction has also often failed to produce the desired cognitive effects (Palumbo, 1990:78): an unguided, free-exploration approach does not develop a deep understanding of the structure and operation of a programming language, and thus does not require learners to use or develop high level thinking skills (Singh, 1992).

The overwhelming consensus is that, for most learners, a practical discovery environment should be complemented by instruction that emphasises structure, interventional facilitation by the teacher, and direct instruction in the programming language (Goosen, 1999:80).

An inquiry-based approach could still be used, as long as learners receive structured, systematic instruction on particular and well-defined aspects of problem solving skills, together with structured experiential learning, which affords a learner the opportunity to exercise and reinforce recently acquired skills and knowledge (DoE, 2002b:8). The language must not only be taught in a way that focuses on top-down programming design (Palumbo, 1990:83), but appropriate activities to reinforce the instruction must also be top-down and strategic, so that problem solving skills might be accessed and developed (Choi & Repman, 1993).

Teacher-mediated learning techniques for dealing with learner/teacher interactions in the programming environment (Palumbo, 1990:74) are also used as learners can provide helpful information about what they understand, as well as their misunderstandings, if they are provided with appropriate opportunities (Madison & Fifford, 2003:226). Teachers might be alerted to learner difficulties by using a face-to-face grading scheme (Madison & Fifford, 2003:227) while journals or other classroom assessment techniques could also identify learners who would profit from added interchange. During these interchanges, learners who explain a program's function and justify design decisions could obtain the opportunity to elaborate their conceptions. Learners would also be less inclined to submit a program they did not fully understand, if they knew that they might be required to explain the program's functioning to their teachers. When learners are prompted to use productive thinking to solve programming problems (Palumbo, 1990:75,78), they have been shown to have fewer misconceptions about the programming

language and also exhibit a greater understanding of programming concepts.

The use of direct instructional techniques (Palumbo, 1990:74) is also one of the attributes that is critical to achieving success in teaching programming language and problem solving. Kirkwood (2000: 509) therefore advises that the direct teaching of problem solving strategies should be interweaved within the problem-based learning (PBL) methodology for teaching computer programming at an introductory level. PBL as a teaching approach seems to offer a lot of promise, and according to Kay *et al.* (2000:113) is characterized by:

- *the placement of problem solving and the development of metacognitive skills at the heart of the curriculum.* This means that problems and their solutions, the modeling of the solution processes, as well as the teacher's response to solutions, require urgent action (Grigas, 1994). In addition to an understanding of elementary programming concepts, principles and skills, an environment designed for fostering problem solving and metacognitive skills, as well as higher order thinking, is also needed (Kirkwood, 2000:510).
- *focusing instruction on particular aspects of problem solving abilities* (Palumbo, 1990:74) and metacognitive skills (Kay *et al.*, 2000:112), with courses that actively and explicitly teach these abilities and skills. The most common methods used for teaching problem solving and critical thinking skills are hands-on experience, real-world problems and cooperative learning (Stephenson, 2002) which should be integrated with on-going formative assessment. Explicit instruction and frequent feedback (Palumbo, 1990:81) are both important for modifying and analysing computer programs, while problem solving practice and extensive on-line access to the computer that performance on writing computer programs. These are all critical features of effective programming language instructional practices that lead to improved programming performance (Husic *et al.*, 1989:570). Pedagogy that focuses on designing computer programming language instruction using an explicit, problem solving approach to teaching and learning programming, delivers much better results for learning and transfer of problem solving skills from the programming language context to related areas (Choi & Repman, 1993) and non-computer environments (Palumbo, 1990:74).
- *an involvement with much broader, open-ended, authentic, substantial problems, which drive the learning and involve a larger set of problem solving skills* (Kay *et al.*, 2000:111). Learners work on problem solving assignments that extend over long periods of time (Rohwer & Thomas, 1989:585), on their own or in small groups, according to their personal preferences. It is common for different learners to be at different stages of an assignment at any given time. They also consult others freely at any time and are no more likely to look to the teacher for assistance than to other learners.

However, to teach problem solving effectively, more should be done than just presenting learners with problems (Goosen, 1999:59). The outcome of problem solving should also be

explicitly stated (Geary, 1994:272). Especially in PBL, where the very general problem statements mentioned are used as a starting-point, it is all the more critical to clearly indicate to learners how they will be assessed (Kay *et al.*, 2000:122). One of the well-acknowledged problems that learners report in PBL is that they are unsure of how much they need to know, how far to explore their problems, and when they can reasonably stop learning. This difficulty can be addressed by using a carefully crafted set of assessment requirements.

When teachers of introductory courses craft opportunities where learners can participate in the construction of assessment criteria for programming success, which are more comprehensive than merely "finding the right answer", more learners are enabled to construct understanding that moves beyond the stage of intuitive conception to the stage of principled understanding of the programming process (Madison & Fifford, 2003:227).

Learners should also receive proper instruction in the strategies that can be used to solve problems. Expert programmers not only have a sufficient knowledge base, comprising a solid command of the factual, declarative, or linguistic base necessary to solve the problem presented, but also a collection of effective strategies for employing that base when solving related problems (Palumbo, 1990:70). The outcome of programming instruction should therefore be learners who have acquired the command structure of a particular language (declarative knowledge) and are then able to employ this information and a wide variety of strategies in strategically different ways (procedural knowledge) when faced with learning and problem solving (Recker & Pirolli, 1995:1) in new and challenging situations (Palumbo, 1990:66).

Care should be taken to initially develop and establish a strong linguistic base of the declarative knowledge required in the programming language to effectively write programs (Palumbo & Reed, 1991) by including adequate coverage of basic control and data structures (Deek & Kimmel, 1999:95). According to Mayer and Dyck (1989:23), the most productive way to learn a new procedural programming language is to first learn the underlying concepts of the semantics in one's natural language, independently of the language syntax. In this way, learners are able to focus on understanding new concepts during the initial sessions with natural language, and can then tie these concepts to the syntax of the programming language in the subsequent sessions. At this stage, learners can be given the opportunity to develop programs to solve specific programming problems that require the use of not only the declarative knowledge base, but also procedural knowledge, as they link program commands together to form effective and efficiently coded programs.

New commands can slowly be added to the declarative knowledge base. If sufficient time is allotted for the learners to integrate the declarative knowledge into programs of their own creation, by using structured activities and assignments, the procedural knowledge base will be equally increased.

The procedural knowledge features of programming (Palumbo, 1990:74) represent cognitively more demanding skills, the acquisition of which is dependent on evolving mental models of the task domain, including models of code structures, programming abstractions, and their functionality, purposes, and operation, as well as how these can be most efficiently employed in the operations of programming tasks, such as initial and ongoing program planning, followed by the debugging of various kinds of programming structures and/or existing program codes (Pirolli & Recker, 1994:273).

Instruction that is structured in a way that both the declarative and procedural aspects of programming receive attention, is particularly important, as learning to program in any language is not an easy task (Rogalski & Samurçay, 1990:157), and many learners find computer programming courses difficult (Madison & Fifford, 2003:217). Programming teachers should, therefore, be well aware, and need to constantly remind themselves, of the myriad difficulties that learners as novice programmers face in acquiring knowledge of various concepts (Westrom, 1992). Learners often have difficulty in understanding the concept of variables, data representation, manipulation and storage, iteration in its many forms, sub-routines, and parameters. Indeed, many teachers and learners attest that parameter passing is the most challenging concept covered in the introductory programming course (Madison & Fifford, 2003:218).

Because of these difficulties, the educational integrity possible in the language (Richfield, 2000d) is of paramount importance. There are no magic languages, but there are such things as bad languages, and bad or inappropriate didactic use of good ones (Richfield, 2000h). Although there is no substitute for didactically and professionally sound use of languages, the use of systems that are didactically beneficial and favourable (Richfield, 2000f) and languages that offer certain educational advantages (Palumbo & Reed, 1991), would be preferred.

This would demand a suitable curriculum and available, amenable learners with the necessary aptitudes (Richfield, 2000h). Learners' ability across the board could act as constraints: "How difficult would a particular language be for learners, and what particular difficulties would it involve" (Tweedie, 2000e)?

3.3.4 Need for stability and safety in language and environment

Not many educational outcomes can be attained if would-be learners are frustrated by unnecessary difficulties with the programming language and environment (Mendelsohn *et al.*, 1990:195). Although the need to emphasise high-level concepts might make it necessary to offer the power of a sophisticated programming language to novice programmers (Palumbo, 1990:83), it should also not be easy for learners to fall into sophisticated loopholes that might be built into a language for the use of professionals. It also necessitates the use of a safe, stable (Richfield, 2000f) and controlled language and environment (Walker, 2000). M.L. Bezuidenhout

(2000a) likewise argues for the use of a language with a high level of control and structure, as learners require structured programming in order to generate effective code (Palumbo, 1990:83). A lack of an inherent structure may be addressed by teaching a language in a structured format, but additional strain is still placed on learners by programming that requires structuring.

A suitable software development environment, such as an Integrated Development Environment (IDE), "which allows for programming, compiling and testing of programs" (Barrow *et al.*, 2002:18) for the language, should also be available (DoE, 2002b:81). An integrated development environment typically consists of a compiler embedded within a collection of supporting routines, such as editors and debuggers (Schneider *et al.*, 2000:471). A wide array of compiler tools can greatly simplify the programmer's task, and increase his or her productivity. Sophisticated on-line debuggers, containing extensive tools for tracing and debugging, help programmers locate and correct errors with spelling and/or error correction, as well as well-defined and easily-understandable error messages (Mendelsohn *et al.*, 1990:196). Good implementations of re-usable software component libraries and packages (DoE, 2002b:82; Cook, 2003) which contain a large collection of already-written program units for the language, should be readily available. Another measure that can be considered to ensure that learners are suitably protected, is that compilers should provide extensive checking, so that any attempt to misinterpret data is detected at compile time or generates a well-specified error at run-time (Chiles, 2001). It should also be possible to execute the language in an environment that prohibits it from deleting or modifying other files, or otherwise performing data destroying and computer crashing operations (DoE, 2002b:82).

3.3.5 Desire for simplicity of runtime model

While the desired concepts and problem solving approaches should be supported, both the language and programming environment should be suitable for novice programmers (Mendelsohn *et al.*, 1990:196) in that they are reasonably simple (Walker, 2000). A programming language with a runtime model that is accessible enough, and sufficiently simple to work with, should therefore be encouraged (DoE, 2002b:82) for use at high school level (Palumbo & Reed, 1991), as a simple and clear context can encourage learners to develop high level thinking skills.

Because they are novice programmers, learners can easily be side tracked by too much awkward syntax, complex language semantics or expansive programming environments. One wants to avoid languages that enforce many new abstract ideas, or have too many "programming tricks" to be learnt (Mendelsohn *et al.*, 1990:176). The complexity of the programming language will affect the level at which learners will work, and will partly determine how much of their time is spent on learning programming, and how much on using that

knowledge to aid other kinds of learning. The language should therefore be considered easy to learn and offer relative simplicity of commands and the use of limited logical constructs, designed for easy instruction as a beginning programming language (Palumbo, 1990:83). Learners should also be able to install the language themselves, so that they can help themselves at home (Delport, 2000).

3.3.6 Life long learning

Cook (2000a) is concerned about what learners are being prepared for. According to the purpose of Computer Studies in South Africa (DoE, 2002b:6), learners should be provided with opportunities to "prepare for a career path, further education and lifelong learning, thus enabling learners to become effective members of a computer-using society".

Computer Science is a fast-changing discipline (SIGCSE, 2001b) in which new computer designs and software are inevitable. The quick pace at which change in the discipline happens, implies that learners must not only keep up to date, but should have the tools and be ready to assess the impact and implications of modern developments. It is therefore critical to prepare learners for long careers (LACS, 2000:4) and a lifetime of change by providing them with the necessary logic skills to adapt to change (Scordilis, 2000b). The outcome should be to ensure that learners have the flexibility and broad-mindedness necessary to adapt and prosper over the duration of their careers (LACS, 2000:5).

Scordilis (2000b) feels that any new language chosen should allow flexibility for learners to learn the basics and/or expand/extend their knowledge according to their abilities and interests. "Give learners the basics, and many will run with them, beyond realms, that teachers have energy or time for." Learners should not only be provided with an orientation to further study in this field (DoE, 2002b:8), but be supplied with a basis for continuing learning in Higher Education and Training. The learner that emerges must value, have access to, and succeed in lifelong education and training of good quality (DoE, 2002b:4). Teachers' primary concern should however be the richness of the experience of the learners they teach as they prepare them for life, not for a specific university course.

Because the field is evolving so rapidly, learners out of school for only a few years will find themselves struggling to stay on top of the latest developments and research (Rappaport & Scharnhorst, 1996). One of the most important outcomes of a Computer Science program should therefore be to produce learners who are life-long learners (SIGCSE, 2001b). As "the curriculum ... seeks to create a lifelong learner" (DoE, 2002a:56), classroom instruction should emphasize and facilitate the mastery and development of processes that deliver required and lasting outcomes of lifelong learning (Nieuwoudt,1998:10) and help learners foster a resolve to continuously develop (Dagiene, 2003:180).

3.4 ASPECTS REGARDING RELEVANCE AND NEW TENDENCIES IN PROGRAMMING THAT SHOULD BE CONSIDERED

Any new language chosen "should be relevant to the current programming paradigm" (DoE, 2002b:81), in that learners are taught the principles of current programming (Scordilis, 2000b) in "a modern programming language". Some of the new trends in programming will now be discussed. Although these aspects will most likely date, and therefore don't fully fall under the heading of universal aspects, they do represent the current, modern view of programming and what should be taught with regard to programming.

3.4.1 Object-oriented programming

Object-oriented programming will only be briefly discussed here, as it was extensively covered in chapter 2. Specifically, some comments related to the instruction of OOP at school/introductory level, will be conveyed.

In Computer Science programming language circles, the accepted fundamental problem solving and programming paradigms currently used include structured, modular, procedural and object-oriented ones (Walker & Schneider, 1994; DoE, 2003a:61).

A relevant language should offer strict adherence to the OOP paradigm (DoE, 2002b:81). While using programming in "a current object-oriented programming language" (DoE, 2002b:7) and "supporting objects" (Chiles, 2000i), good compliance with the OO model (Cook, 2003) necessitates the use of, and language support for

- encapsulation
- inheritance
- polymorphism
- abstraction
- multi-threading (DoE, 2002b:82)
- information hiding and exception handling (Walker & Schneider, 1994)

The language should also be taught in a way that focuses on modularity (Palumbo, 1990:83), with adequate coverage of modularisation (Deek & Kimmel, 1999:95).

Object-oriented systems show strong promise as introductory programming languages for use in high schools (Westrom, 1992). Some of these languages are inexpensive and enable learners to rapidly become capable of producing useful results. They use object-oriented concepts and lend themselves readily to the use of group and project work, as well as some, but not all, structured programming principles.

Lambert and Nance (1998:30), however, warn that object-oriented design "is not necessarily the easiest and most straight-forward way to learn to solve problems on the computer. In particular, the need for objects will not become apparent until we begin to define our own data types".

Given the centrality of object-oriented programming in curriculum requirements, and the difficulty learners have in learning to program in an object-oriented style (SIGCSE, 2001a), Blignaut (2000) believes that at school level, predefined objects should be used as far as possible. He would be more than happy to leave the design of user-defined objects for a second year programming course at university, as he considers this a difficult topic and a minefield for the uninitiated.

SIGCSE (2001a) also cautions that certain didactical problems can be exacerbated when an objects-based model is used, as many of the languages used for object-oriented programming in the industry involve significantly more detail complexity than classical languages. Unless teachers take special care to introduce the material in a way that limits this complexity, such details can easily overwhelm introductory learners.

3.4.2 International trends

A similar trend with regard to a focus on objects and OO related facets is observed in the United States of America (USA). The current advanced placement (AP) CS course and examination may be described as object-based, meaning that classes and objects are used throughout, but object-oriented design is not included in the present syllabus. Respondents to a survey (Walker, 2000) suggested that first year university courses were becoming more object-oriented, including an emphasis on higher-level abstraction, object-oriented design, encapsulation, inheritance, and polymorphism.

Gal-Ezer and Harel (1999:115) also report that in Israel one of the future possibilities include object-oriented programming.

Given this correspondence, international trends with regard to the programming languages used in high school education in other parts of the world should be given consideration (DoE, 2002b:82). One could, for instance, look at what is happening in Europe (Chiles, 2001). Although the language(s) acknowledged by the International Olympiad in Informatics (IOI) does not impact directly on most learners (except for those who reach the final round of the Computer Olympiad), these could also be considered (Chiles, 2000l).

3.4.3 Software development process

The software development process also has implications for the relevancy of the language: the question should be posed whether there is continuing development of a specific language by the developer, "or if it is on the verge of being withdrawn from the development scene" (DoE, 2002b:82). "Is the programming language internationally standardised", "well supported and long-lived" (Richfield, 2000f)? The ability to keep abreast with the latest version of the language and latest language developments (Chiles, 2001) might have important implications for costing.

3.4.4 Internet[2] orientation and applicability to the Web

Another important point that needs to be considered when selecting a new language is net-centricity (DoE, 2002b:82) and applicability to the Web. The Internet is revolutionising the way that information is transferred (Rappaport & Scharnhorst, 1996), and it plays an increasingly important role as an educational resource and learning tool. Nowadays, the trend is towards the development of systems for the Internet, with a large part of commercial application development involving programming for the Internet and the World Wide Web (Barrow *et al.*, 2002:492). A language suitable for the Internet, which facilitates this process, should be used (Delport, 2000) as the whole world is moving towards the Internet, with e-commerce becoming a major aspect (Chiles, 2000i). Pieterse (2000) disagrees; he feels that the value of being able to program for and to the Internet is over-estimated, as most programming for businesses is still done for things like salaries, security, planning, etcetera. The Internet is mainly used as a window through which statistics and advertisements are shown, as well as for e-commerce and the interchange of data.

In terms of applicable environments, transportability across multiple platforms (Chiles, 2001) is required, and a natural choice is needed for development of programs that run under DOS, Windows, and other environments, without prejudice in favour of Web-related functions (Richfield, 2000c). Although learners will be quite familiar with Web-based applications, LACS (2000:3) recommend that the specification of a Web focus be removed, and that the application instead be allowed to change with the times.

3.4.5 Database connectivity

Blignaut (2000) is adamant that future teaching of programming at school level should include proper database education. The current trend in industry is to move away from low level programming and text-based, strictly procedural languages, using traditional text and binary files, to rapid application development (RAD) with database access. Gibson (2000a) also felt swayed by choosing a language that interfaces as naturally as possible with a database, as teachers need to revisit that aspect of their teaching. He believes that there has been a lack of attention to database programming, and learners (and teachers) have not really learnt to design and process databases. These will always be major skills in, and components of, the computer industry. Whatever language is adopted, Gibson (2000a) urged teachers not to skip out on these steps by being engulfed in difficult OO concepts, because a lot of IT revolves around databases, irrespective of the Internet. A programming language, which includes access to a relational database using query language calls built into the language for use in database development (DoE, 2002a:61), should definitely be considered. If proper database connectivity

[2] The **Internet** is the biggest computer network in the world, reaching millions of people, on thousands of interconnected networks (DoE, 2002a:31).

is done, with a focus on normalised database design and SQL, learners will be able to develop useful applications in a professional way (Blignaut, 2000).

One way of avoiding the inclusion of programming for databases is to exclude it from the normal programming environment and do it in MS Access, for example. Badenhorst (2000) however prefers that it is included in the normal programming environment.

3.4.6 Visual programming

Recker and Pirolli (1995:33) found that during instruction, learners' actions appeared to be mostly driven by features present on their screens. Learners may however have conceptual and representational difficulties in constructing dynamic mental models of what the computer is doing when executing a program (Singh, 1992). Such difficulties may constrain the level of programming skill. Instructional methods that help the learner to build a memory image of the computer system (Goosen, 1999:82-83) will help to address some of these difficulties, and thereby improve learners' ability to use a programming language to solve problems. Some of the areas and operations for which models need to be constructed, include

- a simplified version of the main areas where processing is carried out
- the objects that operations are carried out on
- the actions that are executed within the computer

Visual development environments facilitate the development of such models by

- using graphics to let the programmer see what is happening (Schneider *et al.*, 2000:471)
- offering excellent display of execution behaviour in the visibility of different parts of the program (Mendelsohn *et al.*, 1990:196) as they are executed
- ensuring that learners receive immediate, concrete feedback for their efforts (Mendelsohn *et al.*, 1990:176; Madison & Fifford, 2003:227)

Graphics can also make it much easier to understand how a program should be used (Schneider *et al.*, 2000:361). A move away from text-oriented systems to more powerful and user-friendly graphic user interfaces means that instead of having to learn dozens of complex text-orientated commands for such things as copying, editing, deleting, moving, and printing files, learners are presented with simple, easy-to-understand visual metaphors. A graphic environment thus adapted to the needs of users, provides them with a visual vocabulary that they can immediately understand and apply (Drakos, 1995b).

A visual developing environment not only enables the novice programmer to concentrate on the principles and programming behind the user interface, but also "to derive a sense of achievement and motivation from writing good-looking programs" (Barrow *et al.*, 2002:v.).

3.5 ASPECTS THAT SHOULD BE CONSIDERED WITHIN THE SOUTH AFRICAN CONTEXT

3.5.1 The outcomes expected of the subject

Within the SA context, the way in which the South African Qualifications Framework (SAQF) and the new educational dispensation are going to make different demands with regards to outcomes (Cook, 2000a) should be considered, as the outcomes of the subject will determine the direction (Chiles, 2000i).

The SA subject statement for Information Technology has four learning outcomes:

- Learning Outcome 1: Hardware and system software
- Learning Outcome 2: e-Communication
- Learning Outcome 3: Social and ethical issues
- Learning Outcome 4: Programming and software development

Of these learning outcomes, the last is more heavily weighted, "because it is the crux of the subject" (DoE, 2003:6). It is also most relevant to this study in terms of the selection of a programming language. "This learning outcome focuses on the design and development of appropriate computer-based solutions to specific problems using programming (in an object-oriented way which incorporate appropriate structured data types), databases, spreadsheets, websites and their inter-connectivity." (DoE, 2003:5.) In an elaboration of the outcome it states that after having had "practical experience in the design and implementation of solutions using a set of core development tools", learners should be "able to design, implement, test and deliver efficient and effective solutions to problem situations."

Cook (2000d) is of the opinion that the following four points should be contemplated when outcomes for schools are considered:

1. the teaching of problem solving skills and algorithms
2. a new thrust that places more emphasis on semantics
3. the demonstration of good programming habits
4. the demonstration of understanding of the programming paradigm philosophy

The first two of these points have already been discussed extensively in terms of international viewpoints. With specific reference to the South African situation, Computer Studies has concentrated on teaching problem solving using a computer through the vehicle of a programming language ever since its introduction as a subject in South African high schools in 1978 (DoE, 2002b:15). The programming language acts as a tool to solve problems (Wasserman, 2000a) and a resource in problem solving (Richfield, 2000h). It is used to meet the requirements of what teachers wish to achieve through the teaching of problem solving skills (Chiles, 2000i), and not necessarily just to teach programming in a specific language for the sake of programming (Richfield, 2000d).

With regard to the third point mentioned by Cook, the question needs to be asked whether a

particular language complies with educational aims with regard to preparing learners to have a firm foundation in good programming style (Badenhorst, 2000) for well-designed programs (Stephenson, 2002). The primary purpose of using a specific language should be to teach good principles of programming, such as the use of naming conventions and indentation to indicate structure, as opposed to unstructured and haphazard code (Blignaut, 2000).

The last of Cook's points has also already been discussed in section 3.4.1 of this chapter, which stated both the international and South African perspectives. International corroboration for the paradigm shift was also further elaborated on in section 3.4.2.

3.5.2 Financial considerations

This leads to the following question that is a cause for great concern: "What about cost implications?" (Chiles, 2000l.) The fact that cost is a characteristic of great importance in developing countries (Grigas, 1994), makes this is a crucial consideration, as there doesn't seem to be funds available to

- supply schools with adequate hardware and software, and/or
- (re)train teachers (Tweedie,1999).

3.5.2.1 Financial situation in South African schools

"As ... schools themselves may well have to carry the costs (mentioned in the previous paragraph), consideration needs to be given to a language that is either free or has a relatively low cost" (DoE, 2002b:82.). Richfield (2000f) similarly argues for the use of systems that are acceptably affordable, and urges that something should be chosen that would run on an environment that practically all schools can afford (Richfield, 2000h). Teachers of Computer Studies across South Africa are used to the responsibility of drawing up a budget for their centre, including costs for hardware and software (Miller, 2000a). Now an added element will enter into these calculations: where would the money come from to buy the new packages, and upgrade hardware (if needed for the new language)?

3.5.2.2 Hardware updates

The hardware platforms available in schools across the board will need to be considered: are the hardware requirements such that the programming language will adequately run on the computers in the schools?

- "Bearing in mind the hardware constraints imposed by a language, the environment and language size need to be seriously considered" (DoE, 2002b:81). What are the minimum specifications for the computers a school has to have, in terms of the programming language, IDE and database (Badenhorst, 2000; Chiles, 2000l)?
- "How many schools have the hardware to run the new language" (Wasserman, 2000a)?

- "How long would a school running Turbo Pascal on 386s have before they must upgrade or stop teaching CS" (Chiles, 1999b)?

In 1999, most schools in SA that offered CS as a subject had 486's or Pentium I's, but some schools had even more outdated equipment. Moving to a Windows-based language may mean that schools with outdated equipment may have to stop offering the subject (Chiles,1999b). It would be unfair to simply tell some schools to upgrade, because many of these schools are from disadvantaged communities. However, Blignaut (2000) is of the view "that a school, who cannot afford to do a subject properly, should not do it".

3.5.2.3 Cost of software

The cost per workstation should be compared with regard to software, including programming language, IDE and database (Badenhorst, 2000). Delport (2000) is of the opinion that South Africa is a poor country with an increasingly bad exchange rate. She feels that there really is no other economic choice but to choose the Open Source[3] route. If the language(s) chosen operated within the Open Source domain, it would be free to learners and schools. She also maintains that going the visual route would in the long run be intellectual and economic suicide.

3.5.2.4 Cost of retraining teachers

As many of the provincial departments are concentrating on training teachers in the GET band (Gr. 1 - 9), they may not have funds to re-train CS teachers (Chiles, 1999b) in aspects of the new language(s). This means that

- the teachers will have to pay for their own re-training
- the schools will have to bear the additional cost (over and above the changes to hardware and software)
- a sponsor will have to be found (e.g. those companies that believe that a move should be made away from TP)

3.5.3 Training and support of teachers

Teacher supply and the logistics of finding educators to teach the language (Breakey, 2000a) is another crucial problem, which consists of two parts:

- Teachers currently teaching Computers Studies will need to be re-trained in the language.
- New teachers just starting to teach the subject need to know the language.

---

[3] **Open-source software** is a method and philosophy for software licensing and distribution, designed to encourage the use and improvement of software written by volunteers, by ensuring that anyone can copy the source code and modify it freely (DoE, 2002a:32). The **open source** movement claims that peer review of software by thousands can greatly improve the quality of software (Van Rossum, 1999). Philosophers in the Open Source/Freeware/Shareware arena believe that scientific advances in ICT should be the responsibility of a collective intellectual effort by all like-minded people to the benefit of all. The Open Source movement is supported by companies such as Sun, IBM, HP, Oracle and Corel.

3.5.3.1 In-service training for current Computers Studies teachers

Staff training in the CS area, or perhaps more appropriate in some cases, the lack thereof, is already a problem (Miller, 2000a) - there simply aren't sufficient skills available. The problem is that the more relevant the subject is made, the higher the demands and standards expected of teachers become (Gibson, 2000a).

Retraining and/or retooling appears to be the biggest fear of teachers of the subject (Gibson, 2000b), as OO and event-driven programming is regarded as being a hurdle by all TP converts (Gibson, 2000a):

- Booth (2000) wanted to know whether training courses were going to be available, be they correspondence, on the Internet or centralised somewhere for those teachers who do not have access to a nearby university, or to training institutions which are readily available in the bigger centres.
- Tweedie (2000f) didn't think that there could be change without providing courses, as some teachers would not cope at all. He thought that that would be a recipe for disaster. Sample questions would have to be provided well in advance of the first examinations. Otherwise, one could easily end up with programming questions that many learners couldn't answer.

If sufficient retraining is not offered to all CS teachers, some individual schools would still probably thrive and provide a service second to none, but many would suffer (Tweedie, 1999). The end result would be that a standard suitable to all could not be defined. CS would die very quickly.

3.5.3.2 Language(s) in which teachers new to the subject are trained

Another consideration in this regard is the languages being used in Computer Science and Information Systems departments at local tertiary institutions (DoE, 2002b:82). It is likely that graduates from these institutions will be teaching in the schools, so that it makes sense to choose a language with which they are familiar. If many teachers are drawn from the local universities, it would not necessarily be appropriate to teach A++ if universities are producing Z-trained teachers (Chiles, 2000e).

3.5.3.3 Support

From a strategic IT point of view, current and future user bases should be considered (Delport, 2000). What about support - both for teachers new to a language, not to mention the teaching of that language, as well as for the learners? According to Chiles (2000e), it would be advantageous to adopt a language that could be supported by local tertiary institutions and private training organisations (Chiles, 2001). These institutions could offer beginner and follow-up courses, and/or make their Web sites available to teachers who need support. Teachers

could then post questions to the sites and have knowledgeable people answer them (Chiles, 2000i).

3.5.4 Resources available to teachers

"The availability of resource material for the language" (DoE, 2002b:82), with some of it developed locally, should be heeded. These resource materials include (Chiles, 2001; Chiles, 2000i)

- books and tutorial matter, some of which can be down loaded from the Web, and in the case of the tutorial matter, even be executable on the Web
- readily available on-line Web documentation on the Internet, which is implemented together with the selected language, because learners should learn to break from the previous generations' bad habits of working to and from paper (Delport, 2000)
- other Web-based resources, including adequate Internet support structures, such as frequently-asked-questions (FAQ) sites and user groups for the language.

The printed text is the oldest and most widely used source of knowledge (Grigas, 1994), and printed material is very effective as a means for studying. Writing and using textbooks, however, takes a great deal of time. Another problem presents in the fact that although Computer Studies textbooks generally serve as handy resources of subject knowledge (Jacobs & Bezuidenhout, 1994:22), typical textbooks used for first programming courses tend to only concentrate on the syntax and semantics of a specific language (Van den Berg, 1990:31).

Contemporary information technologies offer a more powerful means of information presentation, including hypertexts, electronic books, and multimedia. These make it possible to structure instruction in a nonlinear form (Recker & Pirolli, 1995:2), so that learners would have a choice about how they move through instruction, as well as the option of viewing explanatory elaborations when they are unable to generate them on their own.

However, these advanced techniques are not yet available to many learners, especially in developing countries. Some difficulties can also arise when information is complicated, lengthy, and/or requires deep mental analysis. In these cases, a hard copy is usually produced.

3.5.4.1 Outcomes based education orientation

Outcomes based education (OBE) is a learner-centred, outcomes oriented design, based on the conviction that all individuals can learn (Boschee & Baron, 1994: 193). It recognises that learners are often at different levels of physical and cognitive development, and are therefore capable of responding to diverse degrees of challenge. Teachers have to respond by adjusting the curriculum and instruction, and employing resource materials that would smooth the progress of learners towards the achievement of outcomes. Optional extra activities and

materials that cater for these differences between learners (DoE, 2002a:36) could also be provided.

Using OBE also requires the establishment of the skills, knowledge and thought patterns that people need to function in the real world. This implies that the language taught should allow for formal instruction, OBE and/or taking on a role (Scordilis, 2000b) that facilitates the teaching of programming concepts that are generally applicable (Janse van Rensburg, 2000).

3.5.5 General purpose programming

It is important that a language is selected that can handle the requirements of the type of applications for which it is to be selected (Delport, 2000), both in terms of the quality of the language, and its appropriateness to its purpose (Richfield, 2000f). Although the language should support many tools, both academically and commercially, the course taught should be of a general nature. The language used should, therefore, be a general purpose programming language, and not one specifically developed for a certain environment, e.g. scientific or commercial (DoE, 2002b:81).

3.5.6 Demand and training for industry

According to SIGCSE (2001b), the primary purpose of the subject should be to prepare a skilled workforce for the information technology profession. In this scenario, teachers must ensure that the curriculum they offer gives learners the necessary preparation for their eventual academic and career paths. To produce learners who will be competitive in tomorrow's industry, the curriculum must shift to meet the demand that industry places on it (Rappaport & Scharnhorst, 1996).

If these needs are to be met, the language used must relate to this requirement: "For which language is there a demand?" (DoE, 2002b:82.) According to Adlard (2000b), the battle lines seem to be drawn between industry standard languages, as opposed to sound problem solving languages, and, depending on which journal/newspaper one reads, the mood swings to one or the other. He is of the opinion that this is a moot point, as the reason why certain languages are used in industry is precisely because they are good problem solving languages.

Gibson (2000c) warns that a decision should not "be swayed too much by the popularity of any existing language". Kaasboll (2000:21) emphasises that the process of learning programming should not be confused with the process of software development, and the current commercial use and/or popularity of particular languages shouldn't be seen as essential for selection. Teachers' jobs in CS are to enable learners to learn and master relevant concepts, not study the flavour of the month (Richfield, 2000c).

"What skills are (teachers) trying to teach learners" (Cook, 2000a)? Learners have to be equipped "to do something with their knowledge after school" (Blignaut, 2000). It seems as though some teachers are trying to prepare learners for tertiary study and also for a vocational direction (Gibson, 2000a).

One should consider "what happens to candidates post school in reality" (Gibson, 2000c). If most learners are not directly prepared for the market place, but will need to do post Matric and tertiary courses (Scordilis, 2000b) first, this has a major impact on the choice of language (Gibson, 2000c). While studying Computer Studies, many learners also have other careers in mind (Scordilis, 2000b).

At the moment most learners are not equipped to move into the market area, and many learners never do formal computer courses again (Blignaut, 2000). This majority of learners will not have the skills to do something for themselves, and will most likely do some in-house or self-taught training, with unstructured and haphazard code as a result! If learners move into industry without further training, a move needs to be made to a language that is actually used in industry. This will open doors for learners, even without tertiary qualifications. Badenhorst (2000) would want to prepare learners thus trained for the market. Scordilis (2000b), however, does not believe that school subjects should be just training/skills based. Machanick (2000:1) also indicates the need to educate rather than to train, because training is too skills-specific, and can change too quickly, usually at great cost (Scordilis, 2000b).

Computer Studies must then not be a training course: the emphasis in CS should be on teaching concepts (Chiles, 2000i). It is not, and has never been, the intention to teach programming, or train programmers, for industry or commerce (Chiles, 1999b; Tweedie, 1999), and the decision on which language to adopt should not be reliant on the training needs of industry.

3.5.7 Tertiary institutions and their expectations

It seems as if the majority of CS learners are studying further, and move on to take up some form of computers at tertiary level (Gibson, 2000c). In that case, "what skills would tertiary institutes expect learners to have" (Cook, 2000a)? The ACM/IEEE-CS college-level recommendations consider programming as a prerequisite for college students majoring in computer science, implying that learners are expected to take programming in high school (Deek & Kimmel, 1999:95). It, however, seems as if the choice of a language is not that important to tertiary institutions: as long as learners have been trained in (any language) solving problem, tertiary institutions are prepared to teach learners (many) other languages (Wasserman, 2000a).

Depending on the language taught at school, there seems to be two possible scenarios at tertiary level (Gibson, 2000c):

- Linking the language taught at schools to that taught at the tertiary institutions in their area, will assist those learners that move to these institutions (DoE, 2002b:82), as learners could (in line with the concepts of OBE) receive some recognition (Cook, 2000b) of prior learning and be fast-tracked, as many institutions do.
- If the first-year language is different to the one taught at school, learners are still at an advantage, albeit slightly less.

3.5.8 External assessment and options for a language-independent examination

Something else that needs to be considered when deciding on the programming environment and language to be studied, is that the South African Department of Education (DoE) is working at introducing a single Senior Certificate examination - one examination for all grade 12 candidates (Chiles, 2000d) in all provinces. A single language should thus be considered (Wasserman, 2000a) in order to set a standard throughout the country (Blignaut, 2000).

In direct contrast to this, no one specific programming language or group of programming languages has yet been determined (DoE, 2002b:81). In line with the spirit of OBE, it is entirely conceivable that each school offering this subject could teach a different programming language. This is likely to create a number of problems:

- If different languages are taught at different schools, it would make setting external examinations more difficult (Chiles, 2000e). However, as will be discussed shortly (see following page), there are several ways to work around this difficulty.
- Schools need to be assisted in the form of training and support groups (Wasserman, 2000a). As the main facilitators in this process, curriculum advisers/implementers will need to know a range of programming languages.
- If schools are clustered together geographically (all the schools from one area), in order to assist one another, it would not make sense for the schools to teach different programming languages. However, groups of teachers offering the same language could work together (A. Bezuidenhout, 2000b).
- The number of learners studying a specific programming language may not justify the development of resource material by publishers.
- If an educator for some reason leaves a school, it is possible that the replacement might not know the language being taught at the school.

Given these arguments, Chiles (2000e) would be hesitant to agree to a choice between languages within one province. It is, therefore, recommended that a model of provincialisation be followed, with each examining authority setting the programming language for the schools in their province (DoE, 2002b:81).

Blignaut (2000) doesn't see how it would be possible to do proper external moderation and ensure equal standards ("it is difficult enough as it is currently") if every province follows its own

way and uses a different language. The external moderator would have to be familiar with a variety of programming environments to be able to comment on the correctness of solutions. He feels that such a situation is intolerable, and he is not prepared to agree to this.

For at least as long as programming is examined externally (Wasserman, 2000a), examination papers could remain as they currently are used. Questions could just be set, in both the practical and the theoretical examinations, in such a way that it is not language specific, but could be answered in any language (Rogers, 2000a).

The practical part of the examination could demand practical applications of problems in the schools' respective environments (Richfield, 2000f).

Most of the theoretical paper could be the same for all (Breakey, 2000b): a written examination on such things as hardware and information theory, algorithms, data bases, general technical knowledge and the like (Richfield, 2000c). In the last part of the written paper, a move away from language specific problem solving questions to more generic ones (Gibson, 2000a) should be made. This would typically be a record processing application, involving arrays to set up data for a report, which could, hopefully, be answered in any language, if syntax issues etcetera are ignored.

Agreement on certain basic standardisations of pseudo-code or flowcharting notation needs to be reached for examination purposes, so that examiners and learners can communicate. Such standards should be pliable enough for easy use and comprehension, but tight enough to be unambiguous (Richfield, 2000c). Teachers however cannot be expected to handle programming specific questions in the absence of a standard (and preferably executable, compilable) coding notation (Richfield, 2000d). Notation does matter, and it does leave teachers with problems of

- comparability
- how much notation to include in examinations
- how comfortable to feel with the problems

However, to Wasserman (2000a) it seems very unlikely that a paper can be set that is language neutral and fairly comparable between learners who have worked on different systems (Richfield, 2000c). Some concepts simply are not common to all members of certain sets of systems and languages, or are trivial in one system and major undertakings in others. The setting and marking of papers for such a highly creative activity is extremely difficult (Blignaut, 2000) as it is very hard to set CS questions that are at once:

- markable
- non-trivial
- technically relevant
- educationally relevant
- conducive to the learning of serious abstract or applied skills

It is of interest to Breakey (2000b) that a board of examiners can set a paper that is language neutral. This implies that experts in all of the languages should be involved in moderating the paper. He doesn't believe this to be a healthy scenario at all. Ignoring the logistics, he would much rather see a board of examiners set a paper in each of the languages (Breakey, 2000b). Maybe, due to a lack of resources, this could be done at a national level.

Blignaut (2000) thinks that it is very important that a final decision on the programming language is not made before a draft set of examination papers have been set. This will include the programming/problem solving question in the theory paper, the practical paper itself, as well as guidelines of how to mark a project. These papers should then be actually written by a second person (who is of course knowledgeable of the new language) and then marked properly according to a memorandum. In doing so, issues might be discovered that was not thought of beforehand. Copies of first-year examination papers from various CS and IS departments at universities, representing the different languages considered, might be obtained in order to guide one in spotting possible pitfalls.

In light of the above, Bezuidenhout (1999b) and Tweedie (2000c) agree that it would be best to remove the assessment of programming language specific questions from the theoretical examination. Practical examinations and projects could then be done in different languages (A. Bezuidenhout, 2000b) with an internally marked, regionally moderated practical and portfolio (Adlard, 2000b).

Another option would be to remove the programming language from the external examination papers (Adlard, 2000b), so that programming would no longer be examined in a theoretical or a practical examination (Tweedie, 1999). Programming would then be assessed and/or examined internally by using one or more projects only (Tweedie, 1999; Chiles, 2000d). The use of any platform and language could be allowed for the project component (Adlard, 2000b), and the mark allocation increased (Tweedie, 2000c) for the project based assessment component. Blignaut (2000) is, however, not in favour of more than 100 marks to be allocated for project work, where an external moderator has not been involved.

The format (of independent type projects) should be such that the product should be self-installing or should boot stand-alone. It should include documentation, source and a discussion of the project as seen by the learner. Such material should be fairly easy to mark and to justify the marks given. A list of acceptable projects and criteria for marking would have to be compiled in advance. Schools that wish to use any particular language or system should satisfy themselves that it is practical and adequate for the examination (Richfield, 2000c). Schools would then have the option of choosing which language they want to use and up to what level they want to teach. This will allow for some freedom of choice within a school (Adlard, 2000b).

The Independent Examinations Board (IEB) of South Africa's situation in 2000 was such an open one: the IEB did not prescribe any language and it seemed as though schools could

choose any language (Chiles, 2000e) as long as the teacher was able to assess the practical examination as well as the portfolio in that language.

The IEB has made significant progress in generalising the examination to suit any language, and work is continuing on making the examinations as generic as is reasonably possible. There have been an number of issues involved, as the type of question asked is not always language neutral, but most learners who wrote the IEB examination seemed to have coped quite well (Rogers, 2000a). Adlard (2000a) however mentions that some difficulty was still experienced in training learners to deal with a standard practical examination.

3.6 CONCLUSION

Within a universal context, selection criteria that have worldwide relevancy in all circumstances, now and in a decade, should be set.

The purpose of Computer Studies is the development and use of problem solving approaches and skills, with a focus "on activities that deal with the solution of problems" (DoE, 2002b:6).

With regard to the level and nature of learners of Computer Studies, the language adopted should adequately match the abilities of its users, the learners (Mendelsohn *et al.*, 1990:176). The majority of programming takes place at high school level, where learners' cognitive development levels are growing towards the formal operational stage. The chosen language should therefore provide an instructional environment that promotes the development of higher order thinking and problem solving skills, as well as critical thinking.

The role of programming in Computer Studies instruction is justified in that

- programming is an important skill
- knowledge of programming provides a needed background for dealing with issues of modem society
- learners develop general cognitive skills that are identifiable and transferable to other situations
- studying programming promotes the development of metacognitive skills, the knowledge of which can be applied in other fields.

The outcomes envisioned when learning a programming language in conjunction with the role that problem solving plays in instruction, make it necessary that any programming language selected should be appropriate in the sense that it can be used as a medium for developing problem solving skills (Palumbo, 1990:67) by being as variously problem and solution oriented as possible (Richfield, 2000f).

The language should encourage a self-regulated approach to solving problems by promoting strategies for analysing programming problems and formulating solutions to them (Van den Berg, 1990:28). As the acquisition of cognitive skills is facilitated by high degrees of

metacognition, the language should facilitate the use of the metacognitive components of problem solving during instruction, so that learners' understanding of their own thought processes and their involvement in metacognative behaviour would be developed.

As errors in learners' programs are commonly related to deficiencies in problem solving strategies and insufficient planning (Deek & McHugh, 1998:131), an overall understanding of effective program development, including analysis methodologies, planning and design strategies and problem solving skills and techniques should be encouraged. The role of algorithms in problem solving further emphasises the necessity to focus on underlying skills for the design and analysis of algorithms (Van den Berg, 1990:30).

Due to the didactical principles involved in the teaching and learning of the subject, a first programming course should use a language that is beneficial for shaping problem solving (Palumbo & Reed, 1991). Programming principles such as procedure and data abstraction, top-down design with steps-wise refinement, good programming style, testing and debugging should form an integrated part of the language.

As the compilation and coordination of the components of a program represent some of the greatest stumbling blocks for novice programmers, learners should be provided with a safe, stable, structured and controlled programming language and environment that does not frustrate them due to unnecessary difficulties with the language and environment. A suitable software development environment is required, with compiler tools that simplify learners' task, such as debuggers to locate and correct errors and supply well-defined and easily understandable error messages.

The language and programming environment should be suitable for novice programmers in that it is easy to learn. The programming environment shouldn't be too expansive, but instead should be sufficiently simple to work with and offer relative simplicity of commands. Awkward syntax and complex language semantics should be avoided.

Learning about the programming language should produce learners who are life-long learners, who have been provided with the necessary tools and skills to adapt to a lifetime of change and will prosper over the duration of long careers.
Even though criteria should not date, topics that are relevant and observe new tendencies in programming should none-the-less be considered.

OOP is currently one of the accepted problem solving and programming paradigms being used. The language should therefore offer possibilities for object-oriented design, encapsulation, inheritance and polymorphism.

International trends with regard to the programming languages used in high-school education in other parts of the world should be given consideration.

In terms of the software development process the language should be standardised and have reasonable prospects for continued development.

The language should be suitable and have capabilities for working on and with the Internet to "facilitate electronic communication" (DoE, 2002b:5).

As a lot of programming revolves around programming for databases, the language should have sufficient capacity for database connectivity.

Visual programming languages present learners with an environment that is easy to use, gives them concrete feedback, and allow them to see what is happening in terms of execution behaviour.

Although criteria should be general in terms of relevancy, these should be specifically suited to the South African context.

The South African subject statement for Information Technology has four learning outcomes. Of these, Learning Outcome 4, on programming and software development, is more heavily weighted, "because it is the crux of the subject" (DoE, 2002b:6). It is also most relevant to this study in terms of the selection of a programming language. According to this outcome, the language should provide access to "a set of core development tools" (DoE, 2002b:5) in order for learners to practically experience the design and implementation of solutions using said tools.

The financial situation of schools make it necessary to consider whether schools themselves or education departments will carry the cost for the upgrade of hardware and software. The costs associated with the acquisition of the software required, including the programming language, IDE and database, should be within reasonable reach (Sanz-Casado *et al.*, 2002:141). In this regard Open Source software can be considered. Costs with regard to sufficient in-service training of current CS teachers should also be carefully considered.

Local tertiary institutions expect learners to be able to solve problems in any programming language. If the languages offered at these institutions are the same as those used at school level, students will be slightly advantaged; if these are different, students will at least know programming principles. Teachers new to the subject should be able to learn the language used in school during their training at local tertiary institutions. Due to the training that they provide in different programming languages, both tertiary and private computer training institutions could offer teachers training in, and support for teaching, the language.

It is imperative that learning, teaching support materials and other resources, particularly textbooks, for the language, should be available to teachers. Specifically, resources appropriate to an OBE approach to the subject (DoE, 2002a:5) should be selected.

A general purpose programming language that supports many academic and commercial tools should be used. Many of the languages used for object-oriented programming in

industry involve significantly more complexity than more traditional languages (SIGCSE, 2001a). This, together with the fact that the purpose of the subject is not to provide training, makes it unnecessary to consider the popularity and/or demand for specific languages in industry when selecting a first programming language for high schools.

The new language should provide adequate avenues for assessment (Scordilis, 2000b). In terms of external assessment, options for a language-independent examination (Walker, 2000) need to be considered if provinces/schools implement different languages. Problem solving questions could however still be completed by using a common pseudo-code. Another option is for the programming section to no longer be assessed externally, but instead to use only projects for this purpose.

Once a first programming language for high schools had been selected according to set criteria, such a language needs to be implemented in schools. The next chapter introduces guidelines for the implementation of a first programming language in high schools.

CHAPTER 4

GUIDELINES FOR THE IMPLEMENTATION OF A FIRST PROGRAMMING LANGUAGE IN HIGH SCHOOLS

"If all difficulties were known at the start of a long journey, most of us would never start out."

- William F. Buckley, Jnr. (2003:43.)

4.1 INTRODUCTION

Information technology (IT) has been evolving rapidly during the last half of the 20th century (Sulistyo-Basuki, 1999:353). The curriculum of such a young and vibrant discipline must respond to the enormous and rapid pace of technical change (SIGCSE, 2001b) by growing and evolving in response to new developments and ideas (Walker & Schneider, 1994). It is, therefore, important that curricular programs be assessed and updated on a regular basis to keep up with the rapid changes in the field, and schools then prompted to implement relevant computer science programs (Deek & Kimmel, 1999:91).

From an educational point of view, all reform initiatives should improve teaching practice and learners' academic performance (Maes *et al.,* 1999:661). Computer science educators should therefore not only take a fresh look at programming courses in light of recent advances in software and computer interfaces, but also at research findings in programming instruction and learning in other areas (Thomas & Upah, 1996). Following this line of reasoning, a move to a new programming paradigm and/or language could provide additional impetus for re-examining the way in which programming is taught (Kay *et al.*, 2000:110).

In practice, however, the review and redesign of an entire curriculum is a tremendous job. Appropriate actions to bring about the desired change(s) have to be planned, these actions implemented (Bencze & Hodson, 1999:536) and then assessed to establish to what extent the envisioned outcomes for change have been achieved. Even just the introduction of a new area within an existing subject, such as a new paradigm or language, is an arduous task at tertiary level (Deek & Kimmel, 1999:111), which is exacerbated at high school level.

A central requirement for the acceptance of a new programming language is the availability of efficient implementations. In this instance, the implementation strategy should facilitate the implementation of the whole OO class of languages in a uniform way, instead of being geared towards a specific language (Chakravarty & Lock, 1997:122). The implementation techniques for different paradigms are however largely dissimilar and do not blend easily (Chakravarty & Lock, 1997:121). Fundamental changes in current teaching practices will also be required, because the new paradigm departs sharply from existing beliefs and practices in schools (Spillane & Callahan, 2000:401).

The question, therefore, needs to be asked how existing, efficient implementation techniques can be re-used for integrating the operational concepts needed to implement the programming paradigm. The leap from curriculum to reality, however, is a difficult one, for there are literally few precedents, few models and no guidelines (Vesilind & Jones, 1998:757). Although each discipline has its own identity, other subject areas could still be used as models in the process, with reform parallel to what has occurred in disciplines such as Science and Mathematics (Deek & Kimmel, 1999:91).

In light of the preceding discussion, this chapter will introduce guidelines for the implementation of a first programming language in high schools, starting by looking at initial planning, the different role players in the development and implementation of new curricula and the establishment of outcomes for implementation.

4.2 PLANNING

4.2.1 Initiation

The process of curriculum development (Carl, 1995:47) is initiated when an introductory investigation is launched. Planning for the new curriculum is started off by doing a situation analysis, which can be effectively supplemented by using other methods, such as reviewing existing curriculum and literature, consulting with colleagues, organising focus groups for academic program development and observing national and international trends (Ocholla, 2001:166).

It is important that a new curriculum shouldn't be a summary of previous ones, but should plan ahead for the next decade (Van den Berg, 1990:29). Aspects that influence the field and the role that computers and Computer Studies will play in the future should be considered. Seeing that the field changes so rapidly, skills quickly date and programming competencies only are not enough any more. For this reason, underlying principles are so important.

4.2.2 Role players in the development and implementation of new curricula

The process for developing and implementing a new curriculum brings together different stakeholders with a range of interests in the nature of change in schools (Kirk & Macdonald, 2001:553). Such an arrangement for curriculum reform is complex, as some of the partners' interests can potentially be in conflict with those of others.

It is, therefore, especially important that all role players get together at the start of such a project to establish feelings of co-operation and provide an opportunity to develop good relationships between different contributors, irrespective of the level at which their participation will be required (Mostert, 1986:154). Such a meeting should also be used to explicitly indicate what the roles and relationships of everybody involved would be. The power relations that will inevitably

exist within collaborative processes can also be cleared up, together with the importance of possibilities for all parties to contribute appropriately to curriculum reform.

This cannot be a one-way process where outsiders prescribe and pass on rules (Carl, 1995:141). An interactive process should be promoted with regard to planning and implementation, in which outside organisations and persons are involved and work together on a team approach basis with the eventual implementers (Carl, 1995:139). It is also essential that personal, face-to-face contact, not memoranda, be used between the advocates and the users of new programs (Pratt, 1994:333).

Government education administrators and representatives are usually members of a consortium that manages a curriculum development project (Kirk & Macdonald, 2001:557). In the past, it was a common occurrence that state officials followed a model of curriculum management where an existing curriculum was reviewed, alternatives developed and recommendations for implementation made.

Specialist curriculum writers, their line managers and other stakeholders, including curriculum specialists, teaching methodologists, subject matter experts, university based researchers and scientists, are involved in the innovation of the curriculum (Verhoeven & Verloop, 2002:94) as experts who handle the development task at the level of constructing the curriculum specification. Their contributions to the production of new curricula tend to take the form primarily of subject matter expertise or expertise in methods of delivery of subject matter (Kirk & Macdonald, 2001:558).

Traditionally, the mentioned administrators and experts were the only participants in the process of curriculum change, as teachers were regarded as no more than receivers of curricular wisdom (Carl, 1995:3) who should readily change their ways in response to a new curriculum. Curriculum change models viewed teachers as merely passively standing on the periphery and being onlookers with regard to things that are done for them, and implementing decisions that others, removed from their local environment, have taken (Parke & Coble, 1997:774; Bencze & Hodson, 1999:525). Teachers' activities in the curriculum development field were thus positioned explicitly and tightly as a straightforward process of reproducing (Kirk & Macdonald, 2001:565) and delivering a curriculum that had already been decided on on their behalf. Teachers in a study done by Vesilind and Jones (1998:765) did not talk about themselves as curriculum developers, but saw themselves as mediators between a given curriculum and their learners.

Curriculum researchers and designers have all too often overlooked or simply ignored, but only rarely taken into account, the influence of teachers as implementers of personal practical knowledge when new curricula are designed (Manouchehri & Goodman, 1998:27; Spillane & Callahan, 2000:420).

The anchoring of teachers' authoritative voice to speak on issues relating to curriculum matters within development projects is rooted in the "local context of implementation" of the reforms (Kirk & Macdonald, 2001:561): their intimate knowledge in relation to their learners in their classes, their colleagues, their school structures, politics and work conditions, and the resources, facilities and programs available to them. This knowledge, together with their ability to make judgements about appropriate learning experiences and assessment activities for particular learners in particular learning contexts (Bencze & Hodson, 1999:524), circumscribes the primary form of expertise teachers bring to the reform process, which can only, with considerable difficulty, be challenged by agents who work primarily in the research field.

Because of the nature of their work, teachers make an important and invaluable contribution to the curriculum reform process through their adaptation for their own practice of the ideas and materials advanced by reformers to fit their own local contexts (Kirk & Macdonald, 2001:564). In the processes of making sense of the new instructional arguments and understanding the new curriculum, teachers are inevitably involved to some degree in locating, transforming and reconstructing the innovative ideas embedded in reform materials.

In moving beyond the texts and materials to apply the reforms to their contexts (Kirk & Macdonald, 2001:558), a process is started for delivering a curriculum that not only makes sense to teachers, but also holds possibilities for teacher ownership (Parke & Coble, 1997:776) of curriculum change. They see more options for making choices when their 'own' curricula operate within their school culture.

Curriculum designers should recognise that a crucial aspect of effective curriculum development is the multiple roles which teachers are required to play (Bencze & Hodson, 1999:523). They need to take the trouble to reflect on, and develop an understanding of the attributes and responsibilities of the people who will be involved in the implementation and delivery of a curriculum (Pratt, 1994:280) - what they know and do, and what the dynamics of their interactions look like.

Teachers can be involved in the production of new curricula and curriculum guides at all stages, as

- curriculum-writers (Kirk & Macdonald, 2001:556). A team of curriculum writers employed to prepare the prototype curriculum and other support materials can consist of teachers seconded for the duration of the project from their schools, together with senior teachers who might have experience of curriculum development. These curriculum writers could make occasional visits to trial schools.
- representative members of curriculum-in-development advisory committees who give direct feedback to curriculum writers.
- participants in school-based trials of curricula and curriculum materials. Groups of teachers are invited to contribute to the curriculum-in-development-process-evaluators' reports by

exploring the usability of the curriculum in their work at their schools when they translate the curriculum into units of work that are to be taught, monitored and assessed (Kirk & Macdonald, 2001:557). These teachers' experiences are relayed to curriculum writers.

Most teachers not only want to give more input into the implementation programme (Vesilind & Jones, 1998:765), but are in fact dedicated and willing to investigate and develop programs in response to needs and published recommendations (Deek & Kimmel, 1999:97).

Education ministries usually set the degree of control and freedom that might be given to teachers with regard to curriculum development and implementation at the level of teachers' practice in the classroom (Neves & Morais, 2001:451,456). These control issues also dominate the whole question of the extent to which possibilities exist for the inclusion in, and genuine participation by, teachers (Carl, 1995:2) in partnerships for reforming education. Teachers should be given more autonomy and decision making power at the school level, not only in the design of the curriculum, but also in the latitude they have in introducing and implementing the curriculum (Suárez *et al.*, 1998:657; Chow-Hoy, 2001:670,671).

It is important to widely consult with teachers in an open-minded way (Pratt, 1994:329), as reforms that seek to by-pass teachers will not succeed (Kirk & Macdonald, 2001:551). Teachers could oppose the proposals of curriculum boards (Verhoeven & Verloop, 2002:93) in a move against what they perceive as an imposition (Vesilind & Jones, 1998:768). They are also sceptical about constant mandates for change and tire of new ideas that are constantly being delivered to them from central offices, which they are expected to implement (Parke & Coble, 1997:774).

The successful implementation of an innovative curricular program is dependent on the full and active participation in substantive ways of the classroom teachers who will eventually be expected to implement proposed curriculum reforms (Parke & Coble, 1997:774). Although involving teachers in the associated decision making process places a great amount of trust in the hands of the teachers (Chow-Hoy, 2001:674), it also facilitates conversations between teachers, helps eliminate blaming and finger pointing and opens up greater possibilities for continuous improvement of the curriculum. By ceding a substantial measure of control over the curriculum to teachers, a sense of ownership, that is essential for effective and long-lasting change, is created (Bencze & Hodson, 1999:525).

It is imperative that there is constant clear communication between the different role players during a reform project (Mostert, 1986:154) in the form of a great deal of discussion, interviewing and debriefing to explain terminology, to illustrate roles and to supply answers (Carl, 1995:170). It is especially important that regular meetings of everybody involved in the reform are held, as opportunities where teachers can report back, discuss problems that they might be experiencing and give vent to their grievances (Carl, 1995:139).

Parke and Coble (1997:775) point to research findings that identify teachers as the obvious key agents of change in the classroom. The notion needs to be considered that teachers participating in the process can become significant curriculum makers by acting as active co-developers and collaborative architects in the curriculum process when they join forces with experts and other partners. In this way, expertise in the creation, design and implementation of new curricula (Bencze & Hodson, 1999:521) can be shared to facilitate the production of new, school based curriculum documents (Kirk & Macdonald, 2001:564).

Teachers are indeed being asked to develop the skills and traits necessary to serve as effective agents for educational change (Vesilind & Jones, 1998:757). It is, therefore, critical that teachers are empowered to take on their roles as fully-fledged and effective curriculum agents (Carl, 1995:3) who do not regard the curriculum as a set standard from which one may not deviate, but rather as an opportunity to be made into something relevant and meaningful.

Kirk and Macdonald (2001:555) used a process where curriculum development and assessment practices were brought into play as a vehicle that simultaneously constituted teachers' professional development experience and sustained school reform. Making the many choices inherent in curriculum development not only builds the competence, trust and empowerment that characterises effective people and organisations, but also the kind of empowerment that will ensure the continued professional growth and development of participating teachers (Bencze & Hodson, 1999:534).

Interviews with teachers revealed that their experiences while participating in programs involving curriculum development (Deek & Kimmel, 1999:106) were of significant value as they reflected on current research about teaching and learning to make decisions concerning the design of challenging classroom environments (Parke & Coble 1997:773). Used in this way, practitioner research may be a valuable contribution to an individual teacher's professional development (Kirk & Macdonald, 2001:551). This places teachers at the centre of curriculum construction and place their learning central to efforts of improving education.

After such experiences, teachers know how to learn about educational issues, formulate their own views on significant matters and critique and develop their own educational practices (Parke & Coble, 1997:776). The cycle of dependency on curriculum experts has been broken.

4.2.3 Establishment of outcomes for implementation

The establishment of outcomes for implementation (Lenzi, 1995) in terms of curriculum design must be guided by the outcomes that the implementation of such a curriculum hopes to achieve (SIGCSE, 2001b). Throughout the process of defining a Computer Science curriculum, it is essential to consider the various characteristics that must go into the curriculum in order for it to be successful with all learners, together with crucial elements that support the successful implementation (Alper *et al.*, 1997:148) of these innovative curricular outcomes (Verhoeven &

Verloop, 2002:93). The outcomes of the program in terms of the specific capabilities that learners must have at its conclusion, together with the associated techniques for determining whether the outcomes are met, provide the foundation for the entire curriculum.

4.2.3.1 Ambiguity

Some of the factors that could lead to a discrepancy between intended and executed policy are ambiguousness of the legislation and/or policy concerned, and outcomes that are not clearly formulated (Mostert, 1986:167). Documents that are very clear in terms of how the learning area is set out on paper (Kirk & Macdonald, 2001:563) as well as a commitment to clearly and commonly identified outcomes (Pratt, 1999:188) are therefore needed - the benefits of clarity and the importance of articulating curriculum intentions clearly (Pratt, 1994:66,67) cannot be over estimated. As an example, module titles need to be clearly indicated for effective focus (Ocholla, 2001:165).

4.2.3.2 Concise outcomes

Curriculum outcomes should be concise (Kirk & Macdonald, 2001:564), but at the same time, readers of such outcomes should be able to understand these without further explanation (Pratt, 1994:69). This requirement implies both risks of trivialisation on the one hand (Pratt, 1994:67), and worries about documents seeming to be too verbose on the other.

4.3 DEVELOPMENT

One of the essential and interlocking components in the process followed to provide a framework for curriculum development in education systems and schools (Kirk & Macdonald, 2001:555) that will result in a viable reform program, is efforts at state level to recognise and establish content standards (Deek & Kimmel, 1999:93,111). As soon as criteria for the selection and classification of content have been determined, these content standards for the new language need to be developed into a model (Lenzi, 1995) for implementation, and adopted (Deek & Kimmel, 1999:91). Curriculum frameworks aligned with these content standards can then be developed for the classroom.

4.3.1 Specification and classification of instructional learning content

The specification and classification of instructional learning content (Pratt, 1994:176) and refinement of outcomes entails:

- ensuring that the changes to be implemented in terms of content reflects and meets recognised, meaningful, significant and relevant needs of learners
- choosing content that the particular learners can relate to and are interested in (Chow-Hoy, 2001:665)
- sequencing content in order of familiarity, difficulty and logical progression

In this way, a new reviewed curriculum could consolidate previous content that had been illogically scattered throughout programs and the content of which was inadequately articulated (Ocholla, 2001:165).

4.3.2 Methodological issues

For teachers, implementation is a demanding task (Maes *et al.,* 1999:661) that requires them to change their teaching practice to include particular new activities at classroom level (Maes *et al.,* 1999:673). In order to achieve this, they have to acquire specialized new knowledge, needed skills (Alper *et al.*, 1997:150), beliefs and attitudes that they are unlikely to have gained in their own education or in their previous classroom work. Under such circumstances, it could sometimes be a major problem to get teachers to take the necessary responsibility for their own learning (Bencze & Hodson, 1999:534).

Many teachers' concerns originate primarily from their lack of knowledge about the long range content outcomes of programs, the way in which units are organized, the extent to which ideas and topics are addressed in various units and new instructional practices demanded of them (Manouchehri & Goodman, 1998:36). Didactic guidelines should therefore be supplied about the form in which curriculum content is to be transmitted in the classroom context (Neves & Morais, 2001:455). This necessitates the development of friendly and reliable working methods that can be used in the classroom, because the inclusion of such content in educational curricula is essential to ensure their subsequent implementation in professional practice (Sanz-Casado *et al.*, 2002:135).

The various strategies proposed by policy makers should be studied in light of the significant role played by individual teachers' experiences and perceptions of their involvement. These can either influence the professional learning process of teachers by enhancing or inhibiting teachers' abilities to act on their personal knowledge, beliefs and skills in their daily work, or strengthen the bureaucratic control of instruction at the expense of teachers' working conditions (Maes *et al.*, 1999:662). Working conditions at school level should, therefore, be created that support teachers' work by making changes, new activities and learning possible at classroom level (Maes *et al.*, 1999:673). Ocholla (2001:165) found visits to work environments to be rewarding for understanding the context, environment and work atmosphere that is essential for reviewing curriculum and teaching methods.

One of the issues of paramount interest is that although a new program might bring about numerous exciting and enriching activities, changes in teachers' beliefs and practices do not occur merely by placing new programs at their disposal. As curriculum programs are only as promising as teachers' abilities to realize engaging and demanding interactions for the learning and teaching process occurring in their classrooms, researchers must examine the needs of teachers as they relate to their content (Manouchehri & Goodman, 1998:39) in order for

curricula to potentially change school practice. Often teachers themselves have to decide how to deal with the content at hand and connect ideas between various activities and units. The development of teachers' pedagogical understanding of new curricula should be initiated and guided by providing them with

- detailed methods on how to address content development (Manouchehri & Goodman, 1998:36)
- more in-depth assistance and opportunities to grow and consider alternative representations of content and ideas
- possibilities for any connections among these
- pedagogical practices conducive to effective implementation.

Teaching-learning models recommended by state authorities should not only tell the teacher how to follow such models (Neves & Morais, 2001:451), but should also support implementation by adopting a cognitive perspective on the implementation process (Spillane & Callahan, 2000:401). Teachers have to be provided with a conceptual knowledge base, along with a substantial knowledge of learners' cognition (Manouchehri & Goodman, 1998:37). This will enable them to understand how to deal with learners' thinking, learners' multiple approaches to problems and the misconceptions that learners might encounter as they are faced with different ideas. Model documentation could include sections that carefully orient teachers to the potential of materials by studying how their learners learn and by interpreting their responses, together with how the knowledge, skills and attitudes that learners need, can be taught effectively.

Other problems that teachers could face as they teach a new curriculum include not only a lack of conceptual understanding of subject concepts, but also a lack of teaching approaches that support conceptual understanding (Manouchehri & Goodman, 1998:27). More professional development activities should be pursued and systematically assessed in teacher training programmes that focus on making a concerted effort to help teachers develop the nature and extent of their instructional skills, abilities and conceptions. The resultant teachers' understanding will not only be reflected in planning for instruction, but will also enable them to find ways to transform their understandings into classroom practice that impact on learners (Lederman, 1999:917).

The development of a wide variety of instructional routines and schemes that allow teachers new to teaching a specific language to feel comfortable with the organisation and management of instruction, appears to be a critical prerequisite for any efforts to assist teachers' attempts to promote learners' understanding (Lederman, 1999:927). Maes *et al.* (1999:664) stress the importance of including the necessary general methodologies required for the implementation process, as there is clearly a need for extensive coverage of the strategies and tactics necessary to implement the new language, including significant discussion of the practical details of implementation (Chang *et al.*, 2000) and explicit suggestions of teaching methods

(Vesilind & Jones, 1998:768). In this way, teachers involved as potential users can be made aware of teaching methods that can be used to teach the learning content (Mostert, 1986:186) and can be trained to apply a knowledge of the techniques learned practically and effectively (Sanz-Casado *et al.*, 2002:141). This will enable them to consolidate objective information that is available and truly valuable for the definition of strategies and effective decision making (Sanz-Casado *et al.*, 2002:133,134). This would however require a questioning of existing practice and skilful presentation of an array of alternative, proven educational ideas, practices and approaches (Pratt, 1999:186; Bencze & Hodson, 1999:525) such as using integrated cross-curricular thematic instructional methods (Suárez *et al.*, 1998:657; Chow-Hoy, 2001:671).

Teachers do not want to be directed by others, nor do they wish to be left entirely to their own devices. They should not feel that their teaching style is being criticised (Vesilind & Jones, 1998:767), but rather that they are being supported to feel comfortable with practising new ways of teaching, informed by theoretical perspectives and aligned with best teaching practices suggested in reform literature (Parke & Coble 1997:773). On the other hand, instructors should carefully guard against creating conditions under which teachers accept the legitimacy of particular practices on the authority of a facilitator, rather than through critical reflection (Bencze & Hodson, 1999:535), as such teachers could also easily become preoccupied with achieving predetermined outcomes.

It is also necessary to create a closer alignment between how teachers interpret innovative curricular outcomes, what teachers think and believe, and what they do in practice (Verhoeven & Verloop, 2002:91,92,100). In terms of assessment, success will be determined by the extent to which teachers internalise the importance of the stated outcomes of the reform, together with their ability to implement these outcomes in their classrooms (Lederman, 1999:928). Practical and learner-centred learning methods are encouraged and outcome-based methods of assessment are emphasised (Ocholla, 2001:165).

When an attempt is made to offer teachers a curriculum that they would like to use, it is important that they not only have some say in what should be taught, but are also not constrained in how they will do that (Chow-Hoy, 2001:673,676). Reforms that seek to be overly prescriptive (Kirk & Macdonald, 2001:551), together with demands by central office that curriculum should be implemented exactly as stipulated, indicate a lack of trust in schools and the professionalism of local teachers, and will not succeed. Neves and Morais (2001:468) make an appeal for a more flexible attitude towards teaching methodologies so that the specific conditions of teachers' school contexts could be taken into account. They support the provision of methodological guidelines that focus on methodological guidance (Neves & Morais, 2001:466) for theoretical and methodological issues in education and curriculum development (Ocholla, 2001:143). Access to the pedagogic principles on which curricula are based and to the reasons which justify certain curriculum options and relate to the construction of a curriculum,

may help teachers to respond more efficiently to directives established at a higher level (Neves & Morais, 2001:452,456,457,471).

The extensiveness of curriculum text can be viewed as a measure of the degree of explicitness of the content contained in the curricula: the more extensive, detailed and directive the curriculum text and content is about teachers' actions, the greater the explicitness (Neves & Morais, 2001:459). On the other hand, relatively specific and/or slightly directive text makes the curriculum content less explicit. Curriculum content will be mostly implicit, control would be weaker and there would, therefore, be greater space for intervention by teachers when directives are more limited, vague and flexible. In such a case, teachers can change content according to their experience (Neves & Morais, 2001:458).

Teachers can literally learn by doing (Deek & Kimmel, 1999:91) in that their work setting, the classroom, serves as a learning environment for ideas and perspectives about teaching (Manouchehri & Goodman, 1998:39). All decisions regarding methods and materials used to design and organise the learning environment (Parke & Coble, 1997:785) should be guided by classroom experience (Alper *et al.*, 1997:150). The ultimate outcome should be that those who will be teaching a new Computer Science curriculum would utilise many of the ideas and approaches covered in training programs (Aiken & Snelbecker, 1991) aimed at reform.

4.3.3 Developing new curricular materials

A nation's curriculum documents provide a description of the intended scope, sequence and relative emphasis expected in classrooms (Romberg, 1997:133). Although curriculum documents describe what the intended policy is, there usually is little correspondence between these intentions and the way in which the curriculum is implemented in classrooms (Verhoeven & Verloop, 2002:91). Textbooks take up the challenge of aligning curriculum and materials (Vesilind & Jones, 1998:765) and are often more detailed sources of what subject content is covered than are curricular frameworks. Many teachers comment that their primary guide for classroom practice is the adopted textbook (Parke & Coble, 1997:774). Despite their known faults, good textbooks are invaluable learning tools (Pratt, 1994:264). They are relatively cheap, portable, lightweight, can store enormous amounts of information, do not require electricity and can be used at the learner's own pace.

Whether or not useful, high-quality instructional materials accompany a particular curriculum is one of the major factors in the successful implementation of curriculum innovations. Time should, therefore, be available for the design, composition, production and development of teaching material, which may include the redesigning of existing material (Carl, 1995:155). These materials can be included in the curriculum itself or published in a companion document. Accompanying materials can consist of teacher handbooks, question banks, instructional packages including overhead transparencies and masters of these, reference sources including

books, magazines and newspapers (Carl, 1995:155; Pratt, 1994:261), workbooks and study guides, self-tests and computer software.

Instructional support should incorporate technology for the presentation of new material (Walker, 1997), including facilities for the writing, reviewing, editing and distribution of ideas. At a more ambitious level, support might expand beyond writing to oral presentations and multimedia materials. Some of the ways that technology can assist teachers' efforts include textbook style, single-thread computer based presentation of materials, or hypertext with learner-initiated links.

4.4 CURRICULUM ASSESSMENT AND TESTING

4.4.1 Curriculum assessment

Curriculum assessment involves determining the worth of a curriculum document (Pratt, 1994: 297) once it is designed.

Initial assessment comprises the use of internal assessment, expert appraisal and confidential review:

- Curriculum writers themselves do internal assessment. A couple of weeks after a curriculum document has been completed, the writers go back to the document to check whether they would like to, for example, state a particular outcome in a slightly different way, or add/delete something.
- Using expert appraisal entails the submission (Carl, 1995:47) of the curriculum to one or more experts in curriculum development and some experts on the subject of the curriculum (Pratt, 1994:298), in order to obtain their opinions on the curriculum process and the document produced.
- Confidential review involves informal submission of the curriculum to some of the people who are in positions to influence or block implementation (Pratt, 1994:301), who could include government officials, administrators, community leaders, teachers or representatives of teachers' associations.

After this part of curriculum assessment has been conducted, the next step involves the testing of the new curriculum in the form of a small scale pilot testing, followed by typical-use field-testing (Pratt, 1994:296).

4.4.2 Try-out

In most countries, trying out a curriculum on a small scale in preliminary testing before full-scale implementation (Pratt, 1994:302) is considered to be a part of the curriculum development process (Mostert, 1986:152).

Teachers involved in a try-out should be prepared for the instructional task (Carl, 1995:47) by being exposed to a thorough orientation and training programme to handle reform elements

before testing is started (Mostert, 1986:152,158). This in-service training ahead of a trail programme requires extensive organisation and substantial costs (Mostert, 1986:159), but the fact that the success of the curriculum depends on this training, nevertheless justifies these organisational and cost implications.

4.4.3 Pilot testing

Pilot testing involves small scale testing (Pratt, 1994:303) which is conducted with a relatively small number of schools in this testing phase (Mostert, 1986:156). Kay *et al.* (2000:118) explored the implementation of a new programming language via a pilot project with a small group of students.

Some of the benefits of such a trial include:

- A careful assessment of the trial can inform the decision to move to the new programming language, or not.
- An opportunity is needed to learn how to run a course using the new programming language and a trial group is better for this than the larger population of learners.
- When everything is ready for a move to full implementation of the new programming language, the trial can provide solid answers to teachers who are concerned about difficulties they might encounter.
- The trial results can provide help when challenges are met in the full implementation and the wisdom of implementing the new programming language might be questioned.

When preparations are made for trial implementation, the first thing that has to be done is to project the duration and scope of the trail, and identify a target group in terms of team size and geographical span and implications (Lenzi, 1995; Carl, 1995:155), as to keep this part of the implementation manageable.

With particular reference to the duration of such a project, Pratt (1999:189) advises that educational planners should design the implementation of innovations with a clear timetable that aims to institutionalise the innovation before initial enthusiasm and interest begin to fade. Although sufficient time should be allowed to perform a thorough analysis, it should none the less be emphasised that the trail programme constitutes a temporary system (Pratt, 1999:177), as people bring more energy to bear on a time-limited task, than they do to ongoing obligations.

It is important that schools involved in the trial of a prototype curriculum based on national documents (Kirk & Macdonald, 2001:556) do so voluntarily. Teachers' participation in the trial implementation should especially be voluntary (Bencze & Hodson, 1999:534), as effective change is more likely in such a situation (Vesilind & Jones, 1998:766). Suitable project schools and teachers can, therefore, be found by inviting schools to take part in the project and a final selection can then be made from the available schools (Mostert, 1986: 153,155). Although

incentives to join the group could be considered, it should be borne in mind that interventions that consist of financial reward for participation impede implementation (Mostert, 1986:180).

Generally, successful, more experienced teachers, who were not involved in the development of a curriculum, are selected to teach the developmental content (Mostert, 1986:152). These teachers not only have the confidence to want to make a contribution (Carl, 1995:139), but will also put forward ideas, valuable tips and guidelines due to their insights and experience and come up with their own guidelines and recommendations that could be used in the implementation of programs (Deek & Kimmel, 1999:108). The intention to test the curricular material is sharply focused on the experimental environment if all teachers' influences are constant factors with regard to experience. At the other end of the spectrum, younger teachers, with their recent training and boundless enthusiasm, might also have a contribution to make if a variety of teachers with regard to experience are involved (Mostert, 1986:157).

Pilot testing is often better conducted with learners of above average and below average aptitude. Faster learners may be more articulate about shortcomings of the curriculum, while slower learners will probably make more errors, which will point to unexpected problems (Pratt, 1994:305) and areas of weakness in the curriculum.

If assessment shows that some of the pre-determined outcomes had not been achieved, measures should be put in place to ensure that the reasons for failing to meet the outcomes would not be repeated, and steps taken to rectify the situation (Pratt, 1999:188). Any re-work time needed should be identified (Lenzi, 1995) and scheduled. As follow-up activities and studies are essential for curriculum development and educational improvement (Ocholla, 2001:165), a cycle that consists of testing the model, analysing results and modifying for retest (Pratt, 1994:304), is repeated until the curriculum is as good as it is thought that it can be made. As an example of this procedure, a curriculum developed by Alper *et al.* (1997:150) went through at least three preliminary drafts, with each draft tested in classrooms.

When the curriculum is ready for field testing, teachers who took part in a first trail round can help to train and advise teachers who will be participating in ensuing, more wide-spread implementation (Mostert, 1986:159).

4.4.4 Field-testing

Field-testing tries out a new curriculum under conditions of typical use, with both teachers and learners being representative of those for whom the curriculum is intended. It is the participating teachers' responsibility to track the trial project precisely (Lenzi, 1995) and carefully note the course of the trail as it enfolds by keeping good records during field testing. Of particular interest are anecdotal comments by teachers and learners, reports about interesting and significant incidents, including oddities and anomalies, and especially problems and shortcomings that

might emerge (Mostert, 1986:161). As everything should be noted and recorded, heavy investment could be required for printing, administration and record keeping (Pratt, 1999:183).

Teachers should also be given considerable control of resources and freedom of action. Initiative, independence and adaptation to local conditions must be encouraged (Pratt, 1999:189). Although rewards for special contributions could be considered (Lenzi, 1995), participants can and should strive to exceed the minimum requirements proposed (Chang *et al.*, 2000).

4.4.5 Programme assessment

Programme assessment comprises a review of the activities that occur when a curriculum is implemented (Pratt, 1994:297) in classrooms. Lenzi (1995) strongly advises that such a "post-mortem" be done to learn from a trail experience. Confirming this recommendation, the trial implementation carried out by Kay *et al.* (2000:118) was complemented by an extensive assessment undertaken by staff with professional expertise in the assessment of higher education courses.

Said assessment addressed three issues:

1. Assurance that the learners who had studied the new programming language were no less competent at programming than main group learners.
2. Evidence that the additional generic skills that the new programming language is designed for, were in fact being learnt.
3. Data about learners' attitudes to and perceptions of the new programming language, since these can have a marked effect on learning.

Programme assessments may be conducted in formative (Carl, 1995:47) or summative formats (Pratt, 1994:311) and could include unobtrusive measures (Pratt, 1994:312) such as observation, together with interviews, feedback from learners and written course assessments. In the curriculum document itself, requests can also be made for input from users regarding additions, deletions, successful teaching strategies, new resources and other amendments (Pratt, 1994:317).

The greatest strengths, weaknesses and deficiencies of the program should be identified (Carl, 1995:154), as these would determine the types of training and professional development programmes needed for teachers to be able to deliver the new instructional program (Deek & Kimmel, 1999:108).

After testing and programme assessment, the revised and improved curriculum is readied for full implementation (Mostert, 1986:153). As soon as a commitment has been made to go forward with full implementation, a start can be made to implement the new language (Dagiene,

2003:178) by establishing the necessary infrastructure for advisory programmes on the integration of the program into schools and initial teacher training.

4.5 IMPLEMENTING CURRICULUM CHANGE

Macro implementation is the execution of a curriculum decision as determined at national level by curriculum authorities (Carl, 1995:169), namely, the application of policy and curriculum initiatives (Mostert, 1986:166) that are distributed and applied countrywide. Factors that can impede macro implementation and make the process unsure, include:

- Differences with regard to outcomes.
- The degree of influence and authority executed.
- Communication problems between organisations. As effective implementation is dependent on good administrative support, structure, usage and procedures (Carl, 1995:155) and good channels of communication are essential to enable administration to progress smoothly, the use of these should be advocated and rewarded (Mostert, 1986:182).
- A lack of funds. Costs are important in the thinking of most people and organizations (Pratt, 1994:290). All educational programs must take costs into account at some level, and cannot do everything that they might wish to do if they were somehow freed from economic constraints. Especially higher education is always subject to resource limitations of various kinds (SIGCSE, 2001b). A reform campaign cannot, however, succeed without substantial resources of money and materials (Pratt, 1999:188). The important challenge is to create a high quality program in the presence of potentially very limited resources (Walker & Schneider, 1994), especially when substantial curriculum change coincides with severe cuts in the education budgets (Bencze & Hodson, 1999:521).

  It is important to note that it is not the quantity of money spent, but the way in which it is done, that makes all the difference between success and failure: the amount of money invested should clearly signify economic commitment (Deek & Kimmel, 1999:93,111) and seriousness of intent. As the economic benefits of education are also notoriously elusive (Pratt, 1999:186), program planners need to calculate, and work toward achieving the critical mass of precisely targeted financial support necessary to launch a program initiative with the confidence that effective program implementation (Pratt, 1999:189) will be achieved.

The first stage in the macro-implementation of policy concerns administration, which entails an authoritative decision that leads to the acceptance of a national program. Others consist of adoption and micro-implementation, that will be discussed in detail in the following part of this chapter, while technical validation, where the implemented practice leads to outcomes, will be addressed in the assessment of a curriculum.

4.5.1 Adoption

The state program leads to the adoption of a project at a more local level, which may show slippage from the authoritative program. As many curriculum initiatives have miscarried when curriculum developers have underestimated the importance of implementation (Carl, 1995:170), enough time should be allowed to spend on the adoption plan at local level.

Factors that can directly influence the adoption of a new program:

- Availability of funds.
- Availability of information about, and quality of, the program. Deek and Kimmel (1999:112) believe that curriculum dissemination is the critical first phase of implementation. Curriculum dissemination entails a thorough introduction to a reform or new program, whether it involves a new curriculum and/or training programme (Mostert, 1986:173,185,190). Social action is used that comprises the preparation of all those involved (Carl, 1995:136) through the distribution of information, thoughts and concepts, together with opportunities for input by interested parties (Carl, 1995:138).

  Word needs to be spread by supplying teachers with information about implementation (Gal-Ezer & Harel, 1999:114) in terms of what is being done, as well as why it is being done (Lenzi, 1995), as this dissemination of ideas, information and material will make informed decision making in schools possible (Carl, 1995:147).

  Time will be spent by various role players in making the eventual users of a curriculum aware of the need to change, creating the climate for envisaged change and establishing a climate of trust in, and acceptance of, the intended curriculum (Carl, 1995:135,166).

- Recommendation by authorities and teachers. It should be noted that the fact that academic or pedagogical experts support the need for a new curriculum, does not carry much conviction with teachers (Pratt, 1994:328).
- One of the most important strategies for curriculum dissemination (Carl, 1995:146) involves clearly demonstrating to participants that their efforts will be supported (Mostert, 1986:182), by identifying support services and resources, and making these accessible to teachers (Vesilind & Jones, 1998:766). When it comes to implementing a curriculum, clear specification of required curriculum resources, including sufficient materials, equipment, facilities, personnel, time and cost, can make a difference between success and failure (Pratt, 1994:258). Support staff should also be continuously available to offer material assistance and encouragement. Organizations that can be of assistance to teachers and learners can be listed in the curriculum document (Pratt, 1994:287).

4.5.2 Micro implementation

Micro implementation is the process during which local decisions are taken (Carl, 1995:169). The locally accepted project, which must be implemented at school and classroom level by

subject teachers, leads to the initial curriculum initiative being established as implemented practice.

There is no single formula for success in implementing a Computer Science curriculum (SIGCSE, 2001b), and this task is a difficult one, in part because of the fact that curriculum implementation requires significant local adaptation of a range of recommendations and strategies that have been validated by practice, to match the characteristics of particular, individual institutions.

The environments (Lenzi, 1995) of different schools vary dramatically, according to their contexts, in

- their resources and traditions
- how teachers are prepared and their practices are judged
- how schools are organized and time is spent
- how outcomes are interpreted and learners' performance assessed (Romberg, 1997:138)

The development of education is now based on the framework of an open and flexible curriculum (Suárez *et al.*, 1998:657) in which schools and their teachers design, create and develop their own individual curricular projects and programmes (Dagiene, 2003:176). Curricula are shaped according to the particular socio-economic and cultural context of each school, and relates to the actual needs of the learners and teachers in that school (Dagiene, 2003:180).

Creating a workable curriculum therefore requires the development of the organisational structures that allow for its implementation (Chow-Hoy, 2001:655), in conjunction with a set of outcomes that captures the essence of the field in a way that can be adopted by all schools. This necessitates the use of requirements which are small and broadly supported (Chang *et al.*, 2000) within the flexibility of the curriculum (Romberg, 1997:134).

It is also very important that the best means possible are established and given to schools so that teachers can be placed in charge of defining the outcomes, teaching content and methods proposed by the administration in terms of their own needs and facilities. In this way, schools can determine what gets priority, rather than having it imposed upon them (Kirk & Macdonald, 2001:559).

4.5.2.1 Identification of differences between old and new practices

Differences between old and new practices should be identified, as teachers might grapple to understand how they can reshape their existing practice around reform ideas (Lappan, 1997:233).

It cannot simply be assumed as a matter of course that educational reform, such as a new or revised curriculum, will be implemented faithfully (Mostert, 1986:165). The message contained in a curriculum or the meaning of curriculum content may not be read, or followed, by teachers

(Neves & Morais, 2001:455,456). Implementation problems could also be a result of implementers' efforts to ignore, sabotage and/or adapt interventions to fit their local agendas and preferences.

According to Parke and Coble (1997:774), literature reveals that when teachers feel vulnerable as a result of uncertainty as to what the new curriculum contains, if they lack understanding of the nature and extent of the envisaged change, when they feel pressured to make changes, or they do not have a voice in curriculum decisions, they simply remould the new curriculum to fit their traditional practice. It is all too easy to fall back on the security of previous and existing teaching and learning experiences (Kay *et al.*, 2000:117).

Innovations must normally be implemented by staff who are and will remain, part of an existing structure and who have typically made an accommodation to things as they are (Pratt, 1999:190). As teachers have invested human and material resources in their school's current programmes, many are keen to adapt what they already have in the school so that they don't have to make major changes (Kirk & Macdonald, 2001:560). Typically, teachers rely heavily on materials, content and methods that they know to be effective from previous years (Pratt, 1994:336) while trying out, judging and using parts of new curricula in light of the repertoire of strategies, methods and materials they need to develop (Manouchehri & Goodman, 1998:39).

Teachers as implementers tend to gravitate towards ideas that are familiar, and/or ideas that enable them to package outcomes in familiar instructional forms (Spillane & Callahan, 2000:419). Using familiar language and presenting new content in ways that resonate with ideas that are already familiar to implementers, are more likely to get the attention of local actors (Spillane & Callahan, 2000:420) and thus find their way more easily into local schools' programmes. However, these familiar ideas about instructional change are likely to be understood through the existing schemata that local implementers have for them and as a result, the intent of the ideas for curricular design might not be apprehended or appreciated by implementers. It might be possible that teachers do not perceive the shift as a fundamental innovation in terms of curricular outcomes. This not only makes it easy for them to ignore less familiar or more novel ideas, but also to continue pursuing the same outcomes as they did in the past (Verhoeven & Verloop, 2002:93,100).

In order to address the concerns of misrepresentation of intent and adaptation to existing practice, documents used during implementation might explicitly state what is not meant by certain key ideas.

4.5.2.2 Ensuring that relative advantage of reform is clear to teachers

This can be achieved by presenting innovations in a form that is easy to understand, making it easy to get hold of supplementary sources of information (Carl, 1995:147) and it should also be easy to make the connection with what is expected in practice (Mostert, 1986:182).

4.5.2.3 Teachers' expectations

It is important to ensure that the expectations that teachers have for training and implementation are addressed right from the start (Lenzi, 1995). It should be made clear to participants what would be expected of them during and after implementation. A few teachers might expect to become quasi-professional programmers as a result of completing the program (Aiken & Snelbecker, 1991). Others could assume that they will learn about a wide range of instructional uses of computers, while also getting extensive understanding of the programming language. On the other hand, there is a danger of teachers having expectations that are too precise and too narrow (Bencze & Hodson, 1999:535).

4.5.2.4 Factors influencing successful implementation

Factors which are important for determining whether successful implementation will be facilitated or inhibited (Carl, 1995:167), together with variables that appear to enhance or impede the implementation of the program by participating teachers (Manouchehri & Goodman, 1998:28), should be identified.

The quality of the planning, design and dissemination done before implementation largely determine success during implementation. Effective dissemination must first take place before there can be effective implementation, as the latter depends on the extent to which all the relevant role players were informed and have been prepared for the envisaged change, and whether they are also prepared to associate themselves with it. Only if there had been no indistinct and/or faulty dissemination can the implementation phase that follows be successful.

Properties of implementation strategies and interventions that have proven to lead to effective implementation in education (Mostert, 1986:180) make provision for the following:

- Requests from teachers which lead to sufficient practice oriented in-service training (Carl, 1995:170) that is well received (Pratt, 1994:316,317), because it focuses on concrete problems of users during the implementation of a reform.
- Active and effective involvement of all role players interested in the curriculum, especially teachers (Carl, 1995:138), as meaningful curriculum renewal and change can only be successful if this is the case. Dynamic leadership also needs to be promoted: Pratt (1999:186,191) underscores the significance of management by inspiration, as opposed to trying to invent curriculum "in a vacuum of curriculum leadership" (Vesilind & Jones, 1998:763) with a total lack of, or insufficient, professional support and progressive leadership (Manouchehri & Goodman, 1998:27) from educational leaders.

  Principals can exercise leadership in curriculum improvement by mobilising resources and administration in support programs, to ensure that teachers have the resources they need (Pratt, 1994:324). For heads of departments and subject leaders, the human resources

available to them among their teachers, in terms of utilising people's abilities to the best and getting team work going, is a matter of considerable significance in relation to the implementation of new programmes (Kirk & Macdonald, 2001:559).

In order to create and further promote an appropriate renewal climate while implementation takes place (Carl, 1995:139), a conscious attempt needs to be made to cultivate a high group morale amongst teachers and to instil in them enthusiasm and dedication for the implementation. Teachers should also have the experience that they are truly dealt with as professional people.

- Follow-up through local organisations, as well as regional and national forums on the issues of implementation, which can further the process of dissemination and provide opportunities for sharing and discussion of successful implementations. Local regional group meetings can assist teachers during the review and implementation of a curriculum. The purpose of such meetings is to provide teachers with a local professional network for exchanging ideas and sharing suggestions for better implementation.

  Computer networks could also be used for information exchange among educational institutions (Dagiene, 2003:179) to encourage mutual contact between them. These could also serve as forums for group interaction by utilising such services as bulletin boards and list servers (Walker, 1997) to facilitate ongoing discussion meetings (Carl, 1995:170) and make many contact opportunities with teachers available on a continuous basis (Mostert, 1986:182).

  Additionally, the regional setting can be used to engage teachers in small discussion groups (Pratt, 1994:335) on subject reform and its implications for their professional practice. Teachers need to debate different visions and methods of teaching and learning, publicly articulate outcomes and reflect on curriculum (Vesilind & Jones, 1998:765,766). Although this might unsettle some teachers, they could also become more open to new information and additional models to observe.

  They can then be allowed to learn by watching and seeing what is available by observing instances where the reform has already been implemented (Mostert, 1986:181) in other teachers' classrooms (Maes *et al.,* 1999:662), and/or observing each other teaching. Viewing videotapes of the new program in action, talking with teachers who had already implemented reforms and/or visiting their classrooms will be more effective than listening to lectures on the merits of the innovation. Examples of best practice in the form of a set of modules, work samples (Kirk & Macdonald, 2001:556), lesson plans and assessment instruments developed by other teachers can also be reviewed.

- Excellent teaching materials that have been developed locally (Pratt, 1994:262). A curriculum that comes with insufficient support materials is unlikely to succeed, as it implies

that teachers will be expected to spend large amounts of time acquiring or creating new materials. The development and production of educational materials is labour-intensive, requires a lot of creative ingenuity (Chow-Hoy, 2001:659) and could be one of the most time consuming tasks for teachers. Walker (1997) indicates that the initial preparation of materials and courses based on collaborative learning specifically requires a great deal of time. Many teachers none the less do produce large amounts of such materials in the form of worksheets, other handouts and bulletin board displays (Pratt, 1994:267,336).

Curriculum designers usually assume that teachers will be able to isolate time, planning and resources at the level presumed by implementation documents (Kirk & Macdonald, 2001:564). According to Pratt (1994:336) the actual provision of the necessary time, material and other resource support during implementation is probably the most critical factor in curriculum innovation. Specifically, a shortage of time is "the single most frequently cited barrier to implementation" (Pratt, 1994:335), and a premier reason for not implementing curriculum changes (Pratt, 1994:326).

Teachers in high schools note the constraint of available time to devote to new projects (Kirk & Macdonald, 2001:560). A lack of adequate time for planning and instruction (Manouchehri & Goodman, 1998:36) is a critical issue for successful implementation and one of the major challenges for teachers, regardless of the extent of their experience and familiarity with authentic instruction techniques. If teachers complain of difficulty with time allocation (Pratt, 1994:316), this does not mean that they do not use their time well (Suárez *et al.*, 1998:662). Teachers, and those who support teachers, require the extra time to

- focus on the requirements of the new task (Maes *et al.*, 1999:662), understand the reform ideas and figure out what they might mean for their existing practice and school
- learn the knowledge (Lappan, 1997:233)
- design lessons, conduct classroom activities, assess learner progress, plan additional activities to help learners develop the necessary skills to achieve outcomes and pace their instruction to groups of learners
- reflect on their attempts to carry out reform (Lappan, 1997:208)

Teachers feel that they are not allowed sufficient time as individuals or teams to plan for implementation. There is a lack of time for teachers to talk (Vesilind & Jones, 1998:765), and they "don't have time ... to sit together and plan collaboratively" (Bencze & Hodson, 1999:535).

If participants do not receive any release time for their participation in reform and to execute these activities, they will have to complete all this additional work on their own time, and they will feel that they don't have sufficient time to teach the new language properly (Bencze & Hodson, 1999:524).

Many teachers who work in non-supportive and isolated environments also use the time demand as a rationale to either argue against the value and practicality of a program, or to shy away from using new materials (Manouchehri & Goodman, 1998:36).

It is essential to identify obstacles that might impede a narrowing of the gap between suggested and real practice, and in so doing, prevent the new curriculum from being realised in practice (Mostert, 1986:185,187,191). Obstacles to implementation that might be encountered in this phase can be practical and/or psychological (Mostert, 1986:173).

- When a new curriculum is being introduced, obstacles of political support (Pratt, 1999:189) and power could be faced, as cultural and value differences often make it difficult to obtain consensus on educational issues (Pratt, 1999:186). Tradition can lead to opposition by experts (Pratt, 1999:177), as well as problems of professional and bureaucratic resistance (Pratt, 1999:190). As a result of such problems, there could also be scepticism as to the credibility (Carl, 1995:170) of the new curriculum.
- It is possible that implementation could fail because local implementers lack the know-how or capacity to carry out policy makers' proposals (Spillane & Callahan, 2000:402). Although the quick answer would probably be to train teachers well, those who design implementations may need to begin where local teachers are, rather than dwell solely on the brave new world for education they want to create. Teachers' existing ideas and conceptions about the new programming language and teaching it may be a more appropriate starting place for developing alternative ideas about, and visions for, education among teachers.
- The anxieties that teachers experience typically reflect obstacles with regard to logistical aspects (Carl, 1995:154). Teachers recognise the often-considerable institutional constraints of the educational circumstances within which they work - the factors limiting what might be possible given their internal school structures, large numbers of learners in their classes (Kirk & Macdonald, 2001:560) and various demands placed on their time and energy. Some teachers regard a lack of required equipment in many schools (Dagiene, 2003:178) as an insurmountable obstacle. Teachers could also face obstacles centred on the demands of an overcrowded curriculum that contain an unrealistic amount of material that needs to be covered and has too few suggestions for possible learner assessment, together with a climate of accountability (Vesilind & Jones, 1998:763) that prioritises the acquisition of factual knowledge.

  These obstacles impact teachers' practice (Bencze & Hodson, 1999:521) by influencing how they teach and assess content (Manouchehri & Goodman, 1998:27), and could also significantly affect their willingness and ability to integrate innovative methods and materials into their classrooms. Teachers can handle obstacles differently, and their eagerness and

ability to deal with any obstacles that are encountered (Manouchehri & Goodman, 1998:39) depend on their previous personal and professional experiences, as well as the nature of professional expectations within their working environments.

Developing feasible classroom activities entails acknowledging the legitimacy of these constraints, but attempting to overcome them. Some of the coping strategies that teachers employ include:

- Teaching as little of the new programming as possible.
- Avoiding all but the simplest hands-on work.
- Using outside experts whenever possible.
- Emphasising managerial aspects of their practice and concentrating on getting through the content (Bencze & Hodson, 1999:523).
- Concentrating on areas in which confidence is highest, while avoiding those subject units that they do not feel comfortable teaching, or those that they do not regard as significant (Manouchehri & Goodman, 1998:38).
- Some teachers undoubtedly rely excessively on textbooks and worksheets to structure their curriculum (Pratt, 1994:264).

4.5.2.5 Addressing obstacles and resistance to change

The challenge lies in minimising threats by identifying shortcomings, as well as factors that could lead to, and sources of, resistance to change that might be encountered well ahead of time. These can then be thoroughly taken into account during dissemination (Carl, 1995:137) and dealt with on a continuous basis in such a fashion that later implementation will progress smoothly (Carl, 1995:140). Problems can be addressed by discussing them and making suggestions for possible solutions (Deek & Kimmel, 1999:89). Procedures should be designed and executed to endeavour to break down (Carl, 1995:170) and overcome any obstacles to classroom implementation that have been identified (Mostert, 1986:188). Although enormous amounts of energy, ingenuity, patience and perseverance might be necessary to overcome these obstacles, curriculum programmes can ultimately be completed successfully.

4.5.2.6 Teacher training

A crucial factor is that teachers who are expected to teach the new programming language be given the background and experience to do so (Aiken & Snelbecker, 1991). Intensive training would be necessary to prepare all teachers (Verhoeven & Verloop, 2002:100) who need in-service training to enable them to respond to change, irrespective of their experience, qualifications and competence (Mostert, 1986:159). Plans should be made to provide systematic in-service training before implementation that will produce skilled and qualified classroom teachers (Deek & Kimmel, 1999:93,111).

Aiken and Snelbecker (1991) advise that anyone developing retraining programmes ought to work closely with representatives of target school districts from conception through completion, as school districts play an important role in the actual implementation of policy (Spillane & Callahan, 2000:401).

Teachers' accumulated professional and personal experience, or relative lack of experience and expertise, together with their self-perceptions of their expertise, play a significant part in their interpretations during their first encounters with the new language and their attempts to understand what is required of them. The wide difference in terms of teachers' level of experience, teaching contexts, intentions and perceptions of learners (Lederman, 1999:916,927) is therefore of critical importance. Training should be structured and presented in such a way that provision is made for teachers with diverse experience, including some with many years of experience and others who had been teaching for only a few years (Verhoeven & Verloop, 2002:94) and might lack the experience to make the links across the curriculum that are required (Kirk & Macdonald, 2001:561,563,564).

Experienced Computer Science teachers, who work in the school system and participated in the trial implementation, should form part of the project from the beginning, because having people available who know the pitfalls of teaching this material at the secondary level is very valuable. Though instructors know the Computer Science and instructional design material, it is sometimes difficult for them to grasp the particular problems faced by participants when they return to their classrooms.

Teachers need to be given the opportunity to share and discuss their fears, anxieties, reservations and doubts about curriculum innovations (Pratt, 1994:335; Carl, 1995:155). As local teachers who participated in the trial implementation and/or field-testing of a new curriculum are more credible advocates than distant experts, they can be invited to respond to the doubts of teachers who are considering adopting the innovation (Pratt, 1994:335), discuss the details of implementing the innovation and explain why they did what they did (Vesilind & Jones, 1998:766).

Utilising outside speakers provides a good means for presenting different ideas, as well as introducing topics for which the presenters of the training might not have the required expertise. In order to make the most of the situation when employing such speakers, it is necessary to make certain that they know what they are expected to cover, to ensure that they address the issues pertinent to the purposes of the programme.

With regard to the in-service training in the use of the programming language, Aiken and Snelbecker (1991) note that there is a great deal of material to cover and participants should have enough time to do justice to the course. As all teachers should have a solid foundation in the principles, it is especially important for all participants to have ready access to computers

outside formal classes (preferably at home) with substantial amounts of time to explore and apply the ideas and principles they learn in class on a computer.

One should not change the level at which the material is presented, but it is sometimes necessary to repeat important concepts more often, and/or to clear up any doubts that arose during a presentation (Suárez *et al.*, 1998:665). It is very helpful to present concepts in ways that teachers can relate to. The pace of the course could also be adjusted.

As participants might at times appear to be overwhelmed by the details of the syntax of the language, new approaches to help teachers view their acquisition of the new language in a broader context might need to be explored. Discussions and reflection can be used to make it clear to participants how they could apply their learning and understanding of essential details about the language when carrying out projects (Lederman, 1999:928). The ideas about programming and the new language that teachers construct from subject documents and training, could also be examined by means of discussions and reflection, as these ideas can contribute to the new language being implemented in ways that miss or misrepresent the original intent of policy proposals (Spillane & Callahan, 2000:401).

The usual way of providing in-service training is by requesting teachers from each school in a region to travel to a central point in a curriculum area (Vesilind & Jones, 1998:758). When they return to their schools, these teachers may be asked to organise teachers development, conduct teacher workshops, peer coach, or in other ways try to effect change among colleagues.

The formats in which training can be presented include

- workshops (Alper *et al.*, 1997:151)
- discussion groups (Mostert, 1986:188)
- conferences (Deek & Kimmel, 1999:109)
- systematising, structuring and organising the respective know-how into a series of practical short summer courses and seminars (Sanz-Casado *et al.*, 2002:141)
- continuing educational programmes (Ocholla, 2001:164), including a full degree in Computer Science Education, or even postgraduate courses (Verhoeven & Verloop, 2002:100), which could be presented on campus or in a distance learning format

In conjunction with teacher preparation programmes (Deek & Kimmel, 1999:91) to shape teachers' initial and ongoing understanding of what the new language requires of them (Kirk & Macdonald, 2001:562), long term workshops to improve the skills of educators (Dagiene, 2003:179) can also be organised.

Kirk and Macdonald (2001:557) used teacher workshops, together with many smaller-scale cluster meetings serviced by school liaison officers from their project. Likewise, Manouchehri and Goodman (1998:28) arranged quarterly two-day curriculum workshops and concurrent

project sponsored conferences. The intent of these workshops, cluster meetings and conferences were to provide teachers with opportunities to share their work, experiences and insights about the program with other teachers and project partners and, in so doing, to shape implementation.

Teachers can also be formally introduced to reform materials through workshops (Kirk & Macdonald, 2001:561), which are viewed as an essential means of providing teachers with the appropriate messages (Kirk & Macdonald, 2001:563) about these materials at the moment when they first encounter the materials and attempt to make sense of them. The authors of such materials might think that the materials are self-explanatory, but some teachers will probably not perceive it as such.

Not only do teachers resent being "talked at" in workshops (Vesilind & Jones, 1998:766), but using only documents and oral presentations offer a very weak approach to the introduction of a major curriculum change (Verhoeven & Verloop, 2002:100), as the mental shift required is hard to achieve from merely reading papers (Kay *et al.*, 2000:117) and listening passively. If documents or books are simply given to teachers to read, without doing anything practical in a workshop situation, essential details might not come across. As teachers always want to know what the point is, they won't feel that their time is being wasted when they are given a practical approach (Chow-Hoy, 2001:674). Materials have to be talked about and walked through with someone who's been through it before and then can lead other teachers through. In such circumstances there shouldn't be many problems.

Together with individual programming assignments, training courses could also include group projects (Aiken & Snelbecker, 1991). Some of the major benefits of using groups in training and implementation include the development of linkages (Ocholla, 2001:164) through extended partnerships for cooperation, resource sharing and networking (Sulistyo-Basuki, 1999:360; Kirk & Macdonald, 2001:564). Participants can benefit from group membership and projects in that:

- Teachers do not feel alone in their endeavours (Chow-Hoy, 2001:671) when they learn that their colleagues are also experiencing problems. Fellow teachers not only understand the programming problems, but can also sympathise with many of the frustrations that they have also experienced.
- Not only does the social interactions appeal to teachers, but it also provides a structure through which colleagues can collaborate to help one another (Vesilind & Jones, 1998:757,773). Teachers get the feeling that there are others who would be willing to tell them what they think could work, as each person could contribute knowledge and skills about computing to their mutual benefit. They are also able to bounce ideas off one another in an attempt to present material to learners in an interesting fashion and to make materials available if other teachers are interested (Vesilind & Jones, 1998:767).

- They are very willing to share information and discuss with colleagues experiences and perceptions about instructional planning and problem solving activities employed by individual teachers. Such discussions can lead to shared views (Maes *et al.*, 1999:673) and a shared sense of responsibility, which can be one of the deepest personal lessons from the campaign.
- A supportive environment can be created and sustained in which groups of teachers, who know the learners, locality and school environment well, can work together on theoretical and practical issues related to the design and implementation of learning experiences (Bencze & Hodson, 1999:524). These groups also have the necessary organisation to attempt classroom or school innovations that would exhaust the energy, skill or resources of an individual teacher (Vesilind & Jones, 1998:772) and overcome obstacles by virtue of the persistence and energy of a collection of gifted people.
- If critical thinking, creativity and skilful problem solving are to be developed by learners, it is essential that those who are responsible for that education also possess these attributes. If learners are to acquire critical human qualities, including the ability to work productively in a collaborative mode across the boundaries of culture and ideology (Pratt, 1999:190), their teachers must also possess these abilities.

4.5.2.7 Distribution of materials

An essential aspect of effective curriculum dissemination is the successful distribution of quality curriculum documents and teaching materials, which have been tested, to all teachers (Carl, 1995:155).

The design and distribution of curriculum materials to change instruction is not a new concept, but in fact has a long tradition. History has however shown that a parade of innovative materials in education does not significantly change school practice. The problem might lie in how teachers are actually able to use such documents in their schools (Kirk & Macdonald, 2001:559). As they have particular approaches that they use (Kirk & Macdonald, 2001:561), very few teachers will actually run with new materials as they are published - most of them like to put their own stamp on it.

The way in which teachers perceive, assess and value materials in new programs (Manouchehri & Goodman, 1998:27,38), together with the extent to which these materials are consistently implemented and used in classrooms, are also strongly related to what teachers know about content background, and the connection between teachers' views of their work and the confidence that they have in their knowledge of the subject in terms of innovative pedagogical practices (Chow-Hoy, 2001:677).

These innovations within the classroom include carrying out a variety of activities to facilitate the use of activity guides, the use of more than a textbook and the use of a large amount of support

material that the teachers themselves take to the classroom (Suárez *et al.*, 1998:665). This requires a focus on having teachers develop their own materials (Chow-Hoy, 2001:661) that they will use in the classroom (Aiken & Snelbecker, 1991). Together with the materials that they themselves developed (Dagiene, 2003:177), teachers also report the use of supplemental textbooks and other resources that they can turn to.

It is especially important that teachers are made aware of all new materials on hand, as schools often have more materials, equipment, books and teachers' curriculum guides (Spillane & Callahan, 2000:402) available than teachers realise (Bencze & Hodson, 1999:524), while guidelines or policy statements, often giving detailed schemes of work, are not used, and in some cases their existence is unknown.

4.5.3 User implementation

Factors that influence the plan for implementation by users (Mostert, 1986:174,175) include:

- Explicitness with which reform is spelled out.
- The time allowed for implementation.
- The amount of external support from district authorities in the form of materials and advice. Teachers' practice in schools will not change if teachers are not well prepared for the change and when only poor, marginal support is made available from the educative administration (Parke & Coble, 1997:774) in solving problems that have arisen from the new reform (Suárez *et al.*, 1998:657).

  Education reform can be especially difficult, and teachers are reluctant, even afraid, to teach the new language, because most of them believe that they have little preparation in content or pedagogy (Vesilind & Jones, 1998:758). They experience a lack of both content knowledge and procedural expertise (Bencze & Hodson, 1999:524), and feel that they are especially ill prepared in terms of practical skills (Ocholla, 2001:164).

  There is little support from central offices. Teachers are given neither sufficient guidance by the Ministry of Education, nor adequate support by school boards (Bencze & Hodson, 1999:524). Few school districts provide teachers development programs or instructional materials to support curricula (Vesilind & Jones, 1998:758), and many schools merely show a complacent acceptance of the status quo (Sanz-Casado *et al.*, 2002:135).
- The nature and extent of in-service training of teachers. In-service training seminars, workshops and discussion groups should therefore be organised on an on going basis during implementation. Kay *et al.* (2000:117) found that intensive teacher development sessions provide an excellent starting point for the first year of full implementation. These were run for three days, with mornings devoted to classroom sessions and afternoons to practical activities.

Learners' needs and abilities, features of classroom life and teachers' practical skills are very real and significant considerations when teachers attempt to introduce reforms into their classrooms (Kirk & Macdonald, 2001:560).

- The ways in which teachers approach and interpret curriculum documents, together with the specific ways that they use these documents, are very much determined by teachers' knowledge of the learners they work with, as well as these learners' needs and capabilities (Kirk & Macdonald, 2001:559; Verhoeven & Verloop, 2002:101). Teachers need to establish the appropriateness of specific content for their learners, rate the suitability of new materials in terms of what their learners could possibly do (Kirk & Macdonald, 2001:558), and go with what they think is ideal and their learners need. In this way, teachers provide an important reality check of what is thought to be suitable in terms of the curriculum and what they believe is appropriate in terms of their day-to-day interactions with learners.
- The resources available to teachers in the physical environment with regard to facilities and equipment are a further local consideration that teachers raise in relation to their interpretations of new curriculum materials.
- A third dimension is the practical realities of teachers' work, including issues of power and politics at work in the structures within schools as institutional forms. Where the institutional set up is a co-determinant in implementation, differences in nature and scope will have an effect on implementation (Mostert, 1986:165). As an illustration of this, learner assessment (Spillane & Callahan, 2000:402) has become a key concern. However, in the complex environments in which teachers are attempting to introduce new assessment practices, the practicality issue comes into play when a teacher asks: "How can you accurately record observations of 26 learners after every lesson?"

Once the year has started, support for teachers can be continued with detailed scripts and notes about resources, including hand-outs from the project. A support service could also be provided that, for example, spells out time scheduling. The first few weeks have very detailed scripts, up to eight pages long. As the weeks progress, the guidelines suggest more choices for the teacher to make and there is less detail. By the end of the first semester, the outline for a week's activity is usually two pages long and in the second semester, formal scripts are not provided. Teachers' feedback indicates that those teaching for the first time appreciate this level of support.

Teachers should also be supplied with assistance by structuring the course into sections, with an experienced and committed person as cluster leader (Kay *et al.*, 2000:118). Teachers who serve on project committees, together with additional people from the school or community who could assist with the program (Pratt, 1994:316) can act as curriculum advisors for an area in their immediate vicinity. Such co-ordinators can supply all teams with information at the beginning of each section of instruction (Grigas, 1994) by having frequent meetings with the

teachers in their cluster. This also supplies teachers with opportunities for report back and discussion of ways to handle implementation problems that they and their learners are having. These problems might not have been anticipated, but can be identified by participants (Mostert, 1986:189). It also helps participants to feel that they do not stand alone during the reform.

4.5.4 Institutionalisation

Institutionalisation is linked to the stabilisation of change (Mostert, 1986:175) in a process whereby users' understanding of, and their confidence in, implementation increases as a curriculum gradually takes root and becomes established and consolidated into an accepted part of curricular practice. As an example of this process, Kay *et al.* (2000:122) mention that in three years of full implementation, courses have been refined considerably. This is to be expected with any new course, even ones which only involve a move to a new programming language.

The local capacity to supply teachers with sustained supporting factors such as finance, material, resources and time given for real meaningful involvement which they require to engage in curriculum change and sustain good practice once it is in place, is central to implementing reform (Kirk & Macdonald, 2001:551). It is therefore a matter of some urgency that ways are found to restructure programs of support for teachers and to help transform schools into institutions that accept collegial responsibility for curriculum development by facilitating and encouraging the professional development of their teachers (Bencze & Hodson, 1999:535).

Maintaining a focus on institutional growth is best facilitated by a school climate that provides collaborative staff development that develops an ethos of collegiality through shared experiences. This needs to be done in conjunction with effective social support systems for teachers from colleagues (Pratt, 1994:317) in terms of sufficient professional networks for communication, material trials, deliberation and material resources (Kirk & Macdonald, 2001:564). Teachers' working conditions should allow them to approach the job with more deeply analytic, interactive planning and classroom research, so that learning from their practice on a continual basis can become a reality (Lappan, 1997:233).

When teachers work in environments where they have few opportunities and no incentives to learn, they are less likely to implement the new paradigm (Spillane & Callahan, 2000:402). On the other hand, groups that help by providing contextually appropriate guidance and advice (Ocholla, 2001:164), together with support by supplying (Carl, 1995:155) encouragement to teachers and incentives that operate at the organisational level, reinforce and promote teachers' individual professional learning and development (Bencze & Hodson, 1999:536).

Effective supporting cultures for teacher learning and professionalism build on teachers' existing beliefs and knowledge, take into account the realities of teachers' school situations and offer

coherent and connected instruction and support over time (Lappan, 1997:208). When things go wrong, there is a comprehensive system of support, sensitive to the unique needs of teachers, in place (Chow-Hoy, 2001:658,671).

By providing praise and acknowledgement that translates into other aspects of intrinsic compensation and motivation (Carl, 1995:168), opportunities are created for teachers to further their formal and informal training (Maes *et al.*, 1999:662) and to grow professionally by way of a more extended perspective and responsibility.

It is also crucial to build teachers' confidence and expertise (Aiken & Snelbecker, 1991), so that they can act in adaptable and self-reliant ways in the face of new demands (Vesilind & Jones, 1998:772). The increased confidence they subsequently feel in their personal theories, in their ability to make their own curriculum decisions (Bencze & Hodson, 1999:534) and in expressing their views about educational issues, leads to feelings of enhanced credibility concerning their own educational practice.

4.6 FINAL SUMMATIVE ASSESSMENT OF PROGRAMME IN TERMS OF IMPLEMENTED PRACTICE

Pratt (1994:306) recommends that a curriculum document should contain criteria and procedures for the assessment of that particular programme.

Implementation assessment determines the didactical and technical validity of the curriculum in terms of the extent of agreement between (Carl, 1995:192,193)

- what was intended according to the anticipated curriculum, in terms of anticipated content, instructional actions and learning experiences, and
- how the implemented practice of the curriculum lead to the realisation of learning outcomes, in terms of delivered content and actual instructional activities and learning experiences.

The message carried by a curriculum becomes more meaningful when mechanisms exist in the educational system for the formal and informal assessment of the quality of the curriculum implementation (Neves & Morais, 2001:474) as such. This part of curriculum assessment is carried out in conjunction with teachers, in order to determine to what extent the new curriculum was indeed implemented.

The extent to which educational administrators are able to maintain the fidelity of a new curriculum depends on the capacity within their education systems to impose approaches, which ensures that the stated curriculum will be followed (Lederman, 1999:919), and to mandate practices that hold schools and teachers accountable for their practice (Kirk & Macdonald, 2001:565).

From teachers' perspective, the implementation of the directions of a curriculum depends on multiple factors, such as the social contexts of the schools (Neves & Morais, 2001:456) within

which they work, the amount and quality of their experience, their professional knowledge base about curriculum and instruction, and their own personal theories about effective teaching and learning practices for the subject.

These characteristics of teachers, together with the extent to which they can generate new and better practices in their schools, impact implementation, as the environment of the classroom in which an innovative curriculum project is implemented can positively or negatively influence implementation (Suárez *et al.*, 1998:655). An analysis of the perceptions of learners, teachers and external observers about the teaching-learning environment in classrooms, therefore, comprise an important source of data for the direct assessment of the implementation process and, indirectly, of its quality (Suárez *et al.*, 1998:656).

Such observations of teachers at work can prove to be extremely useful for validating the reliability of performance results obtained (Ocholla, 2001:166). These results not only indicate whether the proposed method allows for the efficient implementation of a new language (Chakravarty & Lock, 1997:121), but also afford additional criteria to assess scientific and technological development (Sanz-Casado *et al.*, 2002:134).

Although the use of qualitative design provides better insight and is more rewarding when obtaining results in a follow-up study (Ocholla, 2001:165), a few quantitative measures of overall performance, closely followed by maintaining frequent assessment, are more valuable than assessing a broad spectrum of many different aspects of program execution. It is, therefore, essential to focus on a stance that emphasise explicit outcomes (Pratt, 1999:186), in contrast with earlier efforts that measured success in terms of inputs:

"...education in general continues to endeavour to manage itself less by specifying learning outputs than by manipulating such inputs as teacher quality and compensation, instructional materials and catalogues of curriculum content" (Pratt, 1999:188).

In order to not only make it possible to gauge progress and feed evidence of success back to teachers (Pratt, 1994:331), but also to assist management directly in pinpointing shortfalls and critical gaps, and in analysing failures, there is a need to

- set quantifiable targets
- establish the ability to utilise explicit outcomes against which the local implementation work in the field (Pratt, 1999:187) can be assessed to improve the program's operation
- train teachers well in the assessment of these outcomes (Verhoeven & Verloop, 2002:101)

There seems to be no alternative to the establishment of limited scope and clear criteria, founded on specific, quantifiable, realistic, dynamic and flexible curricular outcomes and measures for effective management control. The development of such assessment mechanisms (Carl, 1995:47) need, however, to be handled with care (Pratt, 1994:309), as they have important implications for the successful implementation of reform (Lederman, 1999:916).

4.7 CONCLUSION

Instituting sufficient, efficient guidelines for curriculum implementation (Pratt, 1999:326) may facilitate the successful implementation of a new curriculum. Planning should be started by doing an introductory investigation, followed by a situation analysis in conjunction with various other methods of gathering information about the process. All role players in the curriculum process ought to work together (Pratt, 1999:189) for a common outcome, with the needed appreciation for the role of teachers in implementation. Outcomes that need to be achieved (Mostert, 1986:191) have to be identified and formulated. It is of great importance that these outcomes are stated clearly, and in a concise way.

Specifying and classifying learning content that is of consequence to learners can initiate the development of a curriculum. Teachers' concerns regarding content and conceptual knowledge have to be addressed by providing practical advice about teaching the new curriculum. Documentation in this regard must however not be overly prescriptive, but should allow teachers enough leeway to implement according to their specific circumstances. Materials, including, but not limited to, textbooks, that accompany and support the new curriculum are supposed to be developed.

Curriculum assessment benefits from the use of internal assessment, expert appraisal and confidential review. Preliminary testing of the curriculum could take place in the form of pilot testing, using volunteer, experienced teachers. It is important to emphasise that this part of a reform programme constitutes a temporary system (Pratt, 1999:177). Programme assessment is supposed to be carried out in order to revise and improve the curriculum to a standard suited to implementation.

At the macro implementation level of curricular change (Pratt, 1999:320), cost can be an important factor. Micro implementation takes place at classroom level, and granting teachers the necessary freedom facilitates this local adoption. The dissemination of information regarding the new curriculum is required to be done in a manner that will overcome resistance to change and convince participants that their implementation efforts will be supported. Especially teachers are supposed to be made aware of the differences between old and new practice, the relative advantage of reform must be made clear to them and their expectations for implementation have to be gauged.

Factors and variables that will either further or impede implementation need to be identified. Some influence is to be expected in terms of

- the quality of dissemination done before implementation
- effective leadership
- in-service training

- discussion of implementation
- reform campaign managers that need to be able to obtain sufficient resources and teaching materials
- time available for implementation
- obstacles regarding teachers' working conditions

Sources of resistance to change and obstacles to implementation identified should be handled in such a manner that implementation can proceed smoothly.

Curriculum documents and materials are required to be distributed to schools. Teachers must be given the time and resources to be retrained properly (Aiken & Snelbecker, 1991). They would also benefit from guidance and support provided during user implementation, and this support ought to be continued as implementation becomes institutionalised.

The curriculum can finally be assessed in terms of the extent to which outcomes have been realised, as well as the quality of implementation (Mostert, 1986:190,191). Program management needs to focus on outcomes rather than inputs (Pratt, 1999:177).

Reform in teaching and learning, with the underlying need to empower teachers and learners, is neither simple, nor straightforward (Manouchehri & Goodman, 1998:39). The process needs to be accompanied by changes in teacher preparation and development, as it requires teachers to enter a very uncertain world: They must create and test solutions to education problems by researching and using new curriculum materials and systematically integrating new instruction techniques that impose a drastically different structure on learning environments. The competence, motivation and experience of teachers will, therefore, ultimately govern the success of all such reform programmes (Pratt, 1999:189).

"We need to find ways to select for this endeavour those professionals who have a passion for the undertaking in their hearts. And we need to find ways continually to revive, maintain and augment that passion in all educators and their learners." (Pratt, 1999:192.)

CHAPTER 5

EMPIRICAL STUDY, RESULTS AND INTERPRETATION

5.1 INTRODUCTION

Selection criteria that should be considered when selecting a first programming language to teach in high schools, together with guidelines for the implementation of such a language, were identified in chapters 3 and 4 respectively. If the criteria and guidelines identified are valid, they should facilitate the selection and implementation of a language. The aim of this chapter, therefore, is to verify the validity of these criteria and guidelines empirically within the South African context.

In order to address this aim, the empirical study that was carried out is reported on. The research design of the questionnaire used and procedure followed therewith are handled first, followed by a description of results, discussion and interpretation of these results and a report of findings made in this regard.

The reader should note that conclusions and recommendations with regard to this study will be addressed in chapter 6. It is, however, necessary to first describe and motivate the research programme followed.

5.2 EMPIRICAL STUDY PROGRAMME

A descriptive study was designed and carried out in order to verify the validity of criteria and guidelines empirically within the South African context. By utilising a questionnaire completed by role players with varied involvement in Computer Studies, data was gathered on the importance, applicability and use of identified criteria and guidelines.

5.2.1 Aim of the empirical study

The aim of the empirical study was to determine

- how important various respondents rated different criteria for selection when choosing, and guidelines when implementing, a programming language for South African high schools;
- to what extent various criteria and guidelines have been applied and used to select and implement programming languages for South African high schools

5.2.2 Background to study

In order to present an illustration of the size of the subject Computer Studies in the various South African provinces, table 5.1 (see next page) supplies the number of learners enrolled for the subject on higher grade (HG) in each province. Most of these numbers were obtained from a single source (Matthews, 2003) representing the number of matriculants who wrote the HG paper in the subject at the end of 2002, except for those for Limpopo, Mpumulanga and the

Northern Cape. The numbers for the latter, together with confirmation and/or updates for the 2003 numbers in some provinces, were obtained from subject advisors or chief examiners.

Table 5.1: Number of learners enrolled for Computer Studies HG

Province:	Number of learners	Percentage
Eastern Cape	190	3%
Free State	119	2%
Gauteng	2121	34%
Kwazulu-Natal	2079	33%
Limpopo	450	7%
Mpumulanga	255	4%
Northern Cape	55	1%
North-West Province	385	6%
Western Cape	610	10%
Total:	**6264**	

The target population and sampling used in the study was described in chapter 1 (see 1.5.2.2). The research procedure and response rates for various sectors of the target population will be discussed in section 5.3.

5.2.3 Permission

Permission to distribute questionnaires to teachers in the different provinces was obtained from Subject Advisors, or designated persons in those provinces where no Subject Advisors are available. Authorisation was mainly obtained telephonically.

5.3 RESEARCH DESIGN AND PROCEDURE

5.3.1 Questionnaire

A descriptive study was designed to gather data using a questionnaire aimed at various participants in the Computer Studies arena. The questionnaire was designed by the researcher, piloted with a small number of participants and improved based on revision suggestions from these early participants. It consists of ten questions with subdivisions, divided into six sections (see Appendix A). A further explanation regarding the format of specific sections is supplied just before the results obtained for those sections are presented (see 5.4.3).

5.3.2 Ethical issues

Possible participants in the study were informed that participation was voluntary prior to distribution of the questionnaire. On the first page of the questionnaire (see Appendix A) participants were pointed to the aim of the questionnaire, encouraged to provide answers to questions that reflect their opinions, and reassured that responses would be used only for the purposes of this enquiry and all information would be treated as strictly confidential. All data and

opinions offered by respondents were also reported anonymously.

5.3.3 Procedure for gathering data

In Gauteng questionnaires were handed out and completed at a meeting held on November 22, 2003 in Pretoria that was attended by more than fifty role players in the province, including most senior teachers. In four of the provinces (KwaZulu-Natal, Limpopo, North-West Province and the Western Cape) questionnaires were sent to one person (Subject Advisor/Chief Marker), who distributed questionnaires to all persons forming part of the target population in a particular province, and sent the completed questionnaires back to the researcher. In the remaining provinces (Eastern Cape, Free State, Mpumulanga and Northern Cape) questionnaires were e-mailed or faxed to participants for completion and returned by the same mode. This last category of potential respondents, together with all Curriculum Advisors and members of the Writing Committee for the National Curriculum Statement, were initially contacted, and also followed up where needed, telephonically.

5.3.4 Response rate

A total of 124 usable questionnaires were obtained. In Gauteng six teachers' questionnaires could not be used, as they had not served as markers (see 1.5.2.2 regarding sampling used for teachers). In those provinces handled by one person each, all possible respondents were reached. For e-mailed questionnaires 17 were returned out of 29 sent out (59%), while for faxed questionnaires 2 of the 4 sent out were returned (50%).

Two members of the Writing Committee for the National Curriculum Statement and one Curriculum Advisor did not return questionnaires, while one Curriculum Advisor was unreachable for the duration of the data-gathering period.

Results as obtained from the various sections of the questionnaire used are represented next.

5.4 RESULTS

In this section results with regard to individual items are discussed, while factor analysis will be discussed in the next. Data was mainly analysed using descriptive statistics, including frequencies, percentages, averages and standard deviations.

5.4.1 Biographical data

The demographic description of respondents is represented in table 5.2 (see next page), followed by a note regarding the provincial spread of respondents, and reports of the responses of those respondents who had supplied options other than those offered in the questionnaire.

A comparison between table 5.1 and the provincial analysis in table 5.2 shows that the percentages for the numbers of respondents from the various provinces fairly closely match the percentages representing the size of the subject in the provinces.

Table 5.2: Demographic description of respondents

Province:	Numbers	Percentage
Eastern Cape	4	3%
Free State	2	2%
Gauteng	49	40%
KwaZulu-Natal	34	27%
Limpopo	10	8%
Mpumulanga	1	1%
Northern Cape	4	3%
North-West Province	9	7%
Western Cape	11	9%
Respondents' highest qualification in Computer Science:		
First year level	18	14%
Second year level	33	27%
Third year level	45	36%
Honours level	7	6%
Masters level	6	5%
Doctoral level	0	0%
Other	15	12%
Respondents' highest academic qualification:		
B-degree	50	45%
Honours degree	23	21%
M-degree	16	14%
D-degree	3	3%
Other	19	17%
Respondents' highest educational qualification:		
Higher Education Diploma	74	62%
Further Diploma in Education	11	9%
B.Ed	17	14%
M.Ed	5	4%
D.Ed / Ph.D	2	2%
Other	10	9%

The following explanations were supplied by some of the respondents who indicated that their highest qualification in Computer Science is something other than the options offered on the questionnaire:

- 4 specified that they had no qualifications in Computer Science.
- 4 possessed a Higher Education Diploma (HED), three of which indicated that they had had Computer Science as a specialisation subject for the four years of their training.

- 2 held Further Diplomas in Education (FDE), of which one was specifically in Computer Science; this person also has a diploma (Visual Basic).
- National Diploma in computer data processing.
- Diploma in datametry.
- N6 Computer practice.
- Self taught with a few courses attended.

One respondent who indicated that another qualification was applicable did not specify, while two respondents who did have CS qualifications also listed a National Diploma in Information Technology and certificates and dip(lomas) respectively.

From those respondents who indicated that their highest academic qualification was something other than the options offered on the questionnaire, the following was supplied:

- 11 Higher Education Diplomas, of which 2 was specified as Secondary and Senior primary respectively.
- 2 had Matric, one specifying an exemption.
- Diploma.
- International experience in Finland and London.
- National Professional Diploma in Education (NPDE).
- Technical Higher Education Diploma (THED).

Two respondents who indicated that another qualification was applicable did not specify.

From those respondents who indicated that their highest educational qualification was something other than the options offered on the questionnaire, the following was supplied:

- 2 indicated that they had no educational qualifications.
- Adult Basic Education Training (ABET).
- B.com.
- BPAED[1].
- Corporate training.
- NPDE.
- Primary Teachers' Diploma.
- STD[1]

5.4.2 Involvement of respondents in various Computer Studies capacities

The number of years that respondents indicated to have been involved in different capacities in Computer Studies structures is represented in table 5.3 (see next page).

In this section of the questionnaire, many respondents only completed those items applicable to their particular situation, e.g. someone who only has experience as a teacher of Computer

[1] Some abbreviations used by respondents are unclear and were reported as supplied.

Table 5.3: Respondents' involvement with Computer Studies

	Number of years:	0		1-2		3-4		5-6		7-10		More than 10	
		f	%	f	%	f	%	f	%	f	%	f	%
5.3.1	Writing Committee for the National Curriculum Statement	**73**	88	**7**	9	**2**	2	**0**	0	**0**	0	**1**	1
5.3.2	Curriculum Advisor	**72**	87	**5**	6	**2**	2	**3**	4	**0**	0	**1**	1
5.3.3	Administrative capacity - provincial level	**70**	89	**2**	3	**2**	3	**1**	1	**0**	0	**3**	4
5.3.4	Provincial Matric Examiner / Moderator	**52**	61	**11**	13	**2**	2	**14**	16	**3**	4	**3**	4
5.3.5	Teacher of Computer Studies	**3**	3	**3**	2	**15**	12	**21**	17	**31**	25	**50**	41
5.3.6	Training of Computer Studies teachers	**57**	68	**8**	10	**9**	11	**3**	4	**5**	6	**1**	1
5.3.7	Other	**7**	32	**4**	18	**5**	23	**2**	9	**2**	9	**2**	9

Table 5.4: Respondents' involvement other than CS

Number of years:	1-2	3-4	5-6	7-10	More than 10
Provincial Marking of Matric Papers (7) [2]	3	2		1	
Computer training for provincial gifted child education					1
Training of teachers in computer literacy		1			
International Study Tours		1			
Teaching Computer Science at University			1		
Delphi for Teachers[2]					
Intel facilitator[2]					
Corporate training - Future Kids		1			
Private sector management, financial, CEOs			1		
Programming and systems analysis					1

[2] See comment under 5.4.2 next page - respondents did not all supply amount of experience.

would only complete that item and leave the other items blank.

There seems to have been some confusion amongst some respondents regarding the item concerning the Writing Committee for the National Curriculum Statement. Although the committee had only seven members (Chiles, 2002b), of whom two did not complete the questionnaire (see 5.3.3), ten respondents indicated involvement at this level. This committee has also only been active for two years, yet two respondents indicated that they have been involved for three to four years, and another indicated involvement for more than ten years. The latter also did not complete any other items in this section against which this indication could be controlled.

Due to the way in which the population for this study had been chosen, and specifically with regard to the sampling for teachers, the amount of experience respondents have with regard to teaching Computer Studies is quite high. The three respondents who had indicated that they have no experience in this regard consists of two Curriculum Advisors for the subject and a lecturer at a university involved in training Computer Studies teachers. Of these three however, two have a HED each, in combination with a masters degree and third year level in Computer Science respectively, and while the other has no qualifications in Computer Science, he does have both a D-degree and a M.Ed.

Responses of participants who indicated that their involvement with Computer Studies was something other than the options offered on the questionnaire are presented in table 5.4 (see previous page). The numbers in the various columns show the number of respondents who had indicated a specific amount of experience. Some respondents however only indicated the type of experience, but not the amount (e.g. Delphi for Teachers, Intel facilitator).

Now that the demographic composition of respondents has been established, a look can be taken at what their opinions are regarding selection criteria and implementation guidelines.

5.4.3 Selection criteria

In this part of the questionnaire respondents were presented with 41 statements regarding selection criteria (see Appendix A). They were asked to indicate in a column on the left hand side of each statement how important it would be to use each of these criteria for selection when choosing a programming language for South African high schools. However, although some or all of the selection criteria listed might be important, it might not be practical to use some of these selection criteria when selecting a programming language. Respondents were therefore asked to indicate in the column on the right hand side to what extent each of the selection criteria have been applied to select programming languages for South African high schools. Table 5.5 (see next page) presents the complete results obtained regarding the sections on selection criteria in the questionnaire.

Table 5.5: Selection criteria for a first programming language for South African high schools

		Not important at all	Fairly unimportant	Fairly important	Very important	Average[3]	Standard deviation	Selection criteria for a programming language for South African high schools should	Not applied at all	Sometimes applied	Usually applied	Always applied	Average	Standard deviation
5.5.1	f	4	13	36	69	3.393	0.809	• have worldwide relevance	4	13	36	69	2.598	0.97
	%	(3%)	(11%)	(29%)	(57%)				(3%)	(11%)	(30%)	(57%)		
5.5.2	f	5	10	39	67	3.388	0.810	• be relevant for a reasonable period	5	10	39	67	2.66	0.85
	%	(4%)	(8%)	(32%)	(56%)				(4%)	(8%)	(32%)	(55%)		
5.5.3	f	4	13	44	59	3.32	0.799	• suit specific needs relevant to the South African context	4	13	44	59	2.602	0.87
	%	(3%)	(11%)	(37%)	(49%)				(3%)	(11%)	(37%)	(49%)		
								The programming language and its software development environment should						
5.5.4	f	3	13	33	73	3.44	0.78	• suit the needs of novice programmers	3	43	47	22	2.77	0.79
	%	(2%)	(11%)	(27%)	(60%)				(3%)	(37%)	(41%)	(19%)		
5.5.5	f	0	6	24	91	3.70	0.56	• be safe, stable, structured and controlled	1	29	53	32	3.01	0.76
	%		(5%)	(20%)	(75%)				(1%)	(25%)	(46%)	(28%)		
5.5.6	f	3	10	27	81	3.54	0.75	• not frustrate learners because of features suited to professional programmers	6	33	55	21	2.79	0.80
	%	(3%)	(8%)	(22%)	(67%)				(5%)	(29%)	(48%)	(18%)		
5.5.7	f	0	5	33	84	3.65	0.56	• provide effective debugging tools	4	33	50	27	2.88	0.81
	%		(4%)	(27%)	(69%)				(3%)	(29%)	(44%)	(24%)		
5.5.8	f	1	1	34	85	3.68	0.54	• provide understandable error-messages	1	32	56	23	2.90	0.72
	%	(1%)	(1%)	(28%)	(70%)				(1%)	(29%)	(50%)	(20%)		
5.5.9	f	2	11	26	83	3.56	0.73	• be easy to learn	3	40	48	24	2.81	0.79
	%	(2%)	(9%)	(21%)	(68%)				(2%)	(35%)	(42%)	(21%)		
5.5.10	f	3	8	42	68	3.45	0.73	• offer relative simplicity of commands	4	31	56	23	2.86	0.77
	%	(2%)	(7%)	(35%)	(56%)				(4%)	(27%)	(49%)	(20%)		

[3] Most values for averages and standard deviations are given rounded to two meaningful digits after the decimal point. However, in the case of the first three statements for this part of the questionnaire, values obtained are so close together that values are shown to three digits to discriminate.

Table 5.5: Selection criteria for a first programming language for South African high schools (continued)

		Not important at all	Fairly unimportant	Fairly important	Very important	Average	Standard deviation	The chosen programming language should	Not applied at all	Sometimes applied	Usually applied	Always applied	Average	Standard deviation
5.5.12	f	1	16	42	63	3.37	0.74	• adequately match the abilities of learners	6	37	48	20	2.75	0.82
	%	(1%)	(13%)	(34%)	(52%)				(6%)	(33%)	(43%)	(18%)		
5.5.13	f	3	2	37	80	3.59	0.65	• provide an instructional environment promoting development of higher order thinking skills	3	31	55	23	2.88	0.76
	%	(2%)	(2%)	(30%)	(66%)				(3%)	(28%)	(49%)	(20%)		
5.5.14	f	1	3	33	83	3.65	0.57	• provide an instructional environment promoting development of critical thinking skills	5	30	52	26	2.88	0.81
	%	(1%)	(2%)	(28%)	(69%)				(4%)	(27%)	(45%)	(23%)		
5.5.15	f	0	2	17	102	3.83	0.42	• provide an instructional environment promoting development of problem solving abilities	4	22	51	35	3.04	0.81
	%		(2%)	(14%)	(84%)				(4%)	(20%)	(46%)	(31%)		
5.5.16	f	1	7	37	76	3.55	0.64	• facilitate the development of learners' understanding of their own thought processes	9	37	48	19	2.68	0.85
	%	(1%)	(6%)	(30%)	(63%)				(8%)	(33%)	(42%)	(17%)		
5.5.17	f	0	6	34	81	3.62	0.58	• encourage a self-regulated approach to solving problems	6	34	50	24	2.81	0.83
	%		(5%)	(28%)	(67%)				(5%)	(30%)	(44%)	(21%)		
5.5.18	f	2	16	37	66	3.38	0.78	• encourage programming principles such as abstraction	10	37	44	22	2.69	0.89
	%	(2%)	(13%)	(31%)	(54%)				(9%)	(33%)	(39%)	(19%)		
5.5.19	f	2	14	38	68	3.41	0.76	• promote top-down design with step-wise refinement	7	28	49	28	2.88	0.86
	%	(2%)	(11%)	(31%)	(56%)				(6%)	(25%)	(44%)	(25%)		
5.5.20	f	0	3	27	90	3.73	0.50	• support good programming style	3	20	62	28	3.02	0.73
	%		(3%)	(22%)	(75%)				(2%)	(18%)	(55%)	(25%)		
5.5.21	f	1	6	38	77	3.57	0.63	• support new tendencies in programming	12	35	38	30	2.75	0.96
	%	(1%)	(5%)	(31%)	(63%)				(10%)	(31%)	(33%)	(26%)		
5.5.22	f	0	5	26	91	3.70	0.54	• offer possibilities for object-oriented design	17	22	39	37	2.83	1.04
	%		(4%)	(21%)	(75%)				(15%)	(19%)	(34%)	(32%)		
5.5.23	f	0	11	42	70	3.48	0.66	• offer possibilities for visual programming	20	29	34	31	2.67	1.06
	%		(9%)	(34%)	(57%)				(18%)	(25%)	(30%)	(27%)		
5.5.24	f	0	12	44	64	3.43	0.67	• offer possibilities for encapsulation	19	30	36	26	2.62	1.03
	%		(10%)	(37%)	(53%)				(17%)	(27%)	(33%)	(23%)		

Table 5.5: Selection criteria for a first programming language for South African high schools (continued)

		Not important at all	Fairly unimportant	Fairly important	Very important	Average	Standard deviation	The chosen programming language should	Not applied at all	Sometimes applied	Usually applied	Always applied	Average	Standard deviation
5.5.25	f	3	12	41	64	3.38	0.77	• offer possibilities for inheritance	20	33	32	26	2.58	1.04
	%	(3%)	(10%)	(34%)	(53%)				(18%)	(30%)	(29%)	(23%)		
5.5.26	f	5	17	42	56	3.24	0.85	• offer possibilities for polymorphism	19	32	38	22	2.57	1.00
	%	(4%)	(14%)	(35%)	(47%)				(17%)	(29%)	(34%)	(20%)		
5.5.27	f	1	13	39	69	3.44	0.72	• be internationally standardised	18	37	37	22	2.55	0.98
	%	(1%)	(11%)	(32%)	(56%)				(16%)	(32%)	(32%)	(20%)		
5.5.28	f	2	4	20	95	3.72	0.61	• have reasonable prospects for continued support from its developers	14	29	47	23	2.70	0.93
	%	(2%)	(3%)	(16%)	(79%)				(12%)	(26%)	(42%)	(20%)		
5.5.29	f	4	15	52	52	3.24	0.79	• enable Internet programming	28	41	34	12	2.26	0.95
	%	(3%)	(13%)	(42%)	(42%)				(24%)	(36%)	(30%)	(10%)		
5.5.30	f	0	5	42	76	3.58	0.57	• support database programming	18	30	40	27	2.66	1.01
	%		(4%)	(34%)	(62%)				(16%)	(26%)	(35%)	(23%)		
5.5.31	f	1	13	38	70	3.45	0.72	• follow international trends with regard to programming languages used in high school education	23	37	40	15	2.41	0.95
	%	(1%)	(11%)	(31%)	(57%)				(20%)	(32%)	(35%)	(13%)		
5.5.32	f	0	4	22	97	3.76	0.50	• be affordable	15	38	41	19	2.57	0.92
	%		(3%)	(18%)	(79%)				(13%)	(34%)	(36%)	(17%)		
5.5.33	f	0	3	24	95	3.75	0.49	• enable affordable in-service training of CS teachers	23	42	35	14	2.35	0.94
	%		(2%)	(20%)	(78%)				(20%)	(37%)	(31%)	(12%)		
5.5.34	f	5	10	38	70	3.41	0.81	• be compatible with tertiary establishments' choice of programming language	15	48	39	13	2.43	0.86
	%	(4%)	(8%)	(31%)	(57%)				(13%)	(42%)	(34%)	(11%)		
5.5.35	f	1	4	31	85	3.65	0.59	• be included in the pre-service training of CS teachers	17	52	30	17	2.41	0.91
	%	(1%)	(3%)	(26%)	(70%)				(15%)	(44%)	(26%)	(15%)		
5.5.36	f	0	1	30	92	3.74	0.46	• have learning and teaching support materials and resources, such as question banks, instructional packages, workbooks and study guides, available	20	47	28	19	2.39	0.96
	%		(1%)	(24%)	(75%)				(17%)	(41%)	(25%)	(17%)		
5.5.37	f	0	3	30	90	3.71	0.51	• have particularly textbooks suited to learning the language at high school level available	16	47	36	16	2.45	0.90
	%		(3%)	(24%)	(73%)				(14%)	(41%)	(31%)	(14%)		

Table 5.5: Selection criteria for a first programming language for South African high schools (continued)

		Not important at all	Fairly unimportant	Fairly important	Very important	Average	Standard deviation	The chosen programming language should	Not applied at all	Sometimes applied	Usually applied	Always applied	Average	Standard deviation
5.5.38	f	4	8	34	77	3.50	0.76	• have resources appropriate to an OBE approach to the subject obtainable	22	50	31	10	2.26	0.87
	%	(3%)	(6%)	(28%)	(63%)				(20%)	(44%)	(27%)	(9%)		
5.5.39	f	1	4	43	75	3.56	0.60	• support programming for various purposes	12	37	50	16	2.61	0.86
	%	(1%)	(3%)	(35%)	(61%)				(10%)	(32%)	(44%)	(14%)		
5.5.40	f	1	8	44	69	3.48	0.66	• be able to be used with academic tools	15	38	47	13	2.51	0.87
	%	(1%)	(6%)	(36%)	(57%)				(13%)	(34%)	(42%)	(11%)		
5.5.41	f	0	16	51	55	3.32	0.70	• be able to be used with commercial tools	19	38	44	13	2.45	0.90
	%		(13%)	(42%)	(45%)				(17%)	(33%)	(39%)	(11%)		
5.5.42	f	3	13	33	73	3.44	0.78	• be popular in industry, where there is a demand for such programmers	22	34	42	17	2.47	0.97
	%	(2%)	(11%)	(27%)	(60%)				(19%)	(30%)	(36%)	(15%)		

Table 5.6 presents some of the results from table 5.5 in a slightly simplified format by showing a ranking of statements according to the average score calculated for importance and application based on respondents' indications.

Table 5.6: Relative ranking of selection criteria

		Importance		Application	
	The chosen language should	**Average**	**Rank**	**Rank**	**Average**
15[4]	• provide an instructional environment promoting the development of problem solving abilities	3.83	1	1	3.04
32	• be affordable	3.76	2	**29**	2.57
33	• enable affordable in-service training of CS teachers	3.75	3	**39**	2.35
36	• have learning and teaching support materials and resources, such as question banks, instructional packages, workbooks and study guides, available	3.74	4	**38**	2.39
20	• support good programming style	3.73	5	2	3.02
28	• have reasonable prospects for continued support from its developers	3.72	6	17	2.70
37	• have particularly textbooks suited to learning the language at high school level available	3.71	7	**33**	2.45
22	• offer possibilities for object-oriented design	3.70	8[5]	10	2.83
5	• be safe, stable, structured and controlled	3.70	9	3	3.01
8	• provide understandable error-messages	3.68	10	4	2.90
14	• provide an instructional environment promoting development of critical thinking skills	3.65	12	6	2.88
7	• provide effective debugging tools	3.65	13	5	2.88
13	• provide an instructional environment promoting development of higher order thinking skills	3.59	15	7	2.88
38	• have resources appropriate to an OBE approach to the subject obtainable	3.50	22	**41**	2.26
10	• offer relative simplicity of commands	3.45	26	**9**	2.86
19	• promote top-down design with step-wise refinement	3.41	31	**8**	2.88
29	• enable Internet programming	3.24	41	40	2.26

The table contains the ten statements that received the highest averages, as well as the statement that received the lowest average, for importance and application respectively. The statements in the two categories with the highest average obtain a ranking of 1 and the statement for the lowest average a ranking of 41. Where the ranking in terms of application for a specific statement differs greatly from that for importance, the application ranking is shown in bold.

The first statement in table 5.6, regarding the provision of an instructional environment promoting the development of problem solving abilities, obtained the highest average in terms

[4] Numbers in this column of table 5.6 refer to the same numbers of items as used in tables 5.5, e.g. item 15 here is the same item as 5.5.15.

[5] While rounded averages are shown here, ranking is based on the complete values of averages obtained.

of both importance and application. The last statement, with regard to enabling Internet programming, also shows fairly consistent averages, in that it received the lowest average for importance and the second lowest for application. The item that received the lowest average in terms of application (5.6.38) ranks close to the fiftieth percentile for importance.

The items that received the second, third, fourth and fifth highest averages with regard to application rank slightly lower with regard to importance. However, the items that received the second, third, fourth and seventh highest averages with regard to importance rank significantly lower with regard to application - particularly the averages for items 5.6.33 and 5.6.36 are third and fourth from the lowest respectively. The reverse of this trend is true for item 5.6.19, which has a fairly high average for application, but a low one for importance.

The following comments were received from respondents regarding specific items in the section for selection criteria (numbering refers to the numbers used for statements in table 5.5):

5.5.12 A respondent poses the question whether the chosen programming language should adequately match the abilities of "average" learners, as learners' abilities vary. This person feels that offering possibilities for the spectrum is also important, and it should be offered at multiple levels.

5.5.33 In-service training should be essential!

5.5.42 Regarding the popularity of a specific language in industry and whether there is a demand for programmers in a language, one respondent comments that it should not always be a matter of supply and demand, but of an "entrepreneurial approach for career and sustainability".

5.4.4 Implementation guidelines

In this part of the questionnaire respondents were presented with thirty statements regarding guidelines for implementation (see Appendix A). Following the same procedure as used in the sections concerning selection criteria (see 5.4.3), respondents were asked to complete two columns to the left and right of statements about implementation guidelines. The column on the left hand side was again used to indicate the importance of a statement, while an indication was given in the column on the right hand side regarding the extent to which each implementation guideline has been used to implement programming languages in South African high schools. Table 5.7 (see next page) presents the complete results obtained regarding the sections on implementation guidelines in the questionnaire.

The following comments were received from respondents regarding specific items in the section for implementation guidelines (numbering here refers to numbers of statements in table 5.7):

5.7.13 Although one respondent does not think that it is important to convince teachers that they will be supported, she does consider it to be extremely important to actually support teachers.

Table 5.7: Implementation guidelines for programming language

		Not important at all	Fairly unimportant	Fairly important	Very important	Average	Standard deviation	Implementation guidelines for programming language	Not used at all	Sometimes used	Usually used	Implemented fully	Average	Standard deviation
5.7.1	f	2	10	57	50	3.30	0.70	Perform a situation analysis, using various methods for gathering information about the implementation process.	17	54	31	9	2.29	0.82
	%	(2%)	(8%)	(48%)	(42%)				(15%)	(49%)	(28%)	(8%)		
5.7.2	f	0	6	38	75	3.58	0.59	Show appreciation for teachers' roles in implementation at all levels in curriculum process.	20	54	28	8	2.22	0.83
	%		(5%)	(32%)	(63%)				(18%)	(49%)	(26%)	(7%)		
5.7.3	f	3	7	41	68	3.46	0.72	Consider cost implications at macro-implementation (national/provincial) level.	17	44	39	11	2.40	0.87
	%	(3%)	(6%)	(34%)	(57%)				(15%)	(40%)	(35%)	(10%)		
5.7.4	f	0	4	24	91	3.73	0.52	Make guiding documentation about implementation of a programming language available to teachers.	24	44	34	9	2.25	0.89
	%		(3%)	(20%)	(77%)				(21%)	(40%)	(31%)	(8%)		
5.7.5	f	0	5	41	72	3.57	0.58	Grant teachers the necessary freedom to implement at classroom level.	17	44	46	5	2.35	0.79
	%		(4%)	(35%)	(61%)				(15%)	(40%)	(41%)	(5%)		
5.7.6	f	0	1	21	97	3.81	0.42	Develop learning and teaching materials and resources to accompany and support a new curriculum.	31	43	31	8	2.14	0.91
	%		(1%)	(18%)	(81%)				(27%)	(39%)	(27%)	(8%)		
5.7.7	f	0	0	25	94	3.79	0.41	Develop particularly textbooks suited to high school level to accompany and support a new curriculum.	22	36	44	10	2.38	0.90
	%			(21%)	(79%)				(20%)	(38%)	(39%)	(9%)		
5.7.8	f	0	2	25	92	3.76	0.47	State outcomes that need to be achieved clearly and in a concise way.	14	40	46	11	2.49	0.84
	%		(2%)	(21%)	(77%)				(13%)	(32%)	(41%)	(10%)		
5.7.9	f	1	8	40	70	3.50	0.66	Perform preliminary testing of a new curriculum in the form of pilot testing.	36	39	30	5	2.04	0.89
	%	(1%)	(7%)	(33%)	(59%)				(33%)	(36%)	(27%)	(5%)		
5.7.10	f	3	12	42	62	3.37	0.77	Participation of teachers in pilot testing takes place on a voluntarily basis.	36	38	35	6	2.10	0.93
	%	(3%)	(10%)	(35%)	(52%)				(33%)	(35%)	(32%)	(5%)		
5.7.11	f	4	8	33	74	3.49	0.77	Use experienced teachers to conduct pilot testing.	35	30	37	8	2.16	0.96
	%	(3%)	(7%)	(28%)	(62%)				(32%)	(27%)	(34%)	(7%)		
5.7.12	f	0	3	39	77	3.62	0.54	Carry out program assessment in order to revise and improve a new curriculum to a standard suitable for implementation.	30	35	32	12	2.24	0.98
	%		(2%)	(33%)	(65%)				(28%)	(32%)	(29%)	(11%)		
5.7.13	f	1	6	33	79	3.60	0.63	Convince teachers that their implementation efforts will be supported.	35	44	24	8	2.05	0.91
	%	(1%)	(5%)	(28%)	(66%)				(31%)	(40%)	(22%)	(7%)		

Table 5.7: Implementation guidelines for programming language (continued)

		Not important at all	Fairly unimportant	Fairly important	Very important	Average	Standard deviation	Implementation guidelines for programming language	Not used at all	Sometimes used	Usually used	Implemented fully	Average	Standard deviation
5.7.14	f	1	7	33	78	3.58	0.64	Make teachers aware of the differences between old and new practices.	23	49	28	12	2.26	0.91
	%	(1%)	(6%)	(28%)	(65%)				(20%)	(44%)	(25%)	(11%)		
5.7.15	f	0	4	23	93	3.74	0.51	Address teachers' knowledge of content and concepts in a new curriculum by providing practical advice about teaching.	30	47	23	9	2.10	0.90
	%		(3%)	(19%)	(78%)				(28%)	(43%)	(21%)	(8%)		
5.7.16	f	1	10	41	65	3.45	0.69	Gauge teachers' expectations for implementation.	25	47	26	10	2.19	0.90
	%	(1%)	(8%)	(35%)	(56%)				(23%)	(44%)	(24%)	(9%)		
5.7.17	f	1	6	33	76	3.59	0.63	Specify and classify learning content that is of consequence to learners to initiate the development of a curriculum.	20	48	31	7	2.24	0.83
	%	(1%)	(5%)	(28%)	(66%)				(19%)	(45%)	(29%)	(7%)		
5.7.18	f	0	1	37	81	3.67	0.49	Distribute information regarding a new curriculum in a way that will overcome resistance to change.	28	43	30	9	2.18	0.91
	%		(1%)	(31%)	(68%)				(26%)	(39%)	(27%)	(8%)		
5.7.19	f	0	4	34	81	3.65	0.55	Make the relative advantage of reform clear to teachers.	23	50	25	12	2.24	0.91
	%		(3%)	(29%)	(68%)				(21%)	(45%)	(23%)	(11%)		
5.7.20	f	1	5	36	74	3.58	0.62	Identify factors/variables that will advance or slow down implementation.	15	53	28	9	2.30	0.82
	%	(1%)	(4%)	(31%)	(64%)				(14%)	(50%)	(27%)	(9%)		
5.7.21	f	0	4	28	86	3.69	0.53	Have educational leaders provide progressive leadership during curriculum improvement by mobilising administration and resources in support of teachers.	21	55	27	6	2.17	0.80
	%		(3%)	(24%)	(73%)				(19%)	(50%)	(25%)	(6%)		
5.7.22	f	1	4	39	75	3.58	0.604	Create opportunities for teachers from different schools to discuss implementation.	21	51	26	21	2.26	0.90
	%	(1%)	(3%)	(33%)	(63%)				(19%)	(46%)	(24%)	(11%)		
5.7.23	f	1	4	38	76	3.59	0.602	Handle sources of resistance to change and obstacles to implementation identified in such a manner that implementation can proceed smoothly.	25	45	33	4	2.15	0.82
	%	(1%)	(3%)	(32%)	(64%)				(23%)	(42%)	(31%)	(4%)		
5.7.24	f	0	0	19	100	3.84	0.37	Give teachers the time and resources to be retrained properly.	36	37	24	9	2.06	0.95
	%			(16%)	(84%)				(34%)	(35%)	(23%)	(8%)		
5.7.25	f	0	0	17	102	3.86	0.35	Distribute curriculum documents and materials to schools.	24	34	35	14	2.36	0.98
	%			(14%)	(86%)				(22%)	(32%)	(33%)	(13%)		
5.7.26	f	0	0	14	104	3.88	0.32	Provide teachers with guidance and support during implementation.	28	43	23	12	2.18	0.95
	%			(12%)	(88%)				(26%)	(41%)	(22%)	(11%)		

Table 5.7: Implementation guidelines for programming language (continued)

5.7.27	f %	0	1 (1%)	27 (23%)	90 (76%)	3.75	0.45	Continue support for teachers as implementation becomes established and consolidated into an accepted part of curricular practice.	27 (25%)	49 (46%)	21 (20%)	9 (9%)	2.13	0.89
5.7.28	f %	0	4 (3%)	30 (26%)	84 (71%)	3.68	0.54	Involve teachers in the assessment of a new curriculum.	22 (21%)	55 (52%)	19 (18%)	10 (9%)	2.16	0.86
5.7.29	f %	1 (1%)	4 (3%)	32 (27%)	81 (69%)	3.64	0.59	Assess a new curriculum according to the extent to which its implementation leads to the achievement of learning outcomes.	21 (20%)	50 (47%)	28 (27%)	6 (6%)	2.18	0.82
5.7.30	f %	2 (2%)	7 (6%)	39 (33%)	70 (59%)	3.50	0.69	Determine to what extent a new curriculum was indeed implemented.	18 (17%)	51 (49%)	27 (26%)	8 (8%)	2.24	0.83

5.7.24 One respondent reports that teachers went on a one-week course at the beginning of 2002 and that was it. The rest she had to teach herself.

5.7.25 She hasn't received a curriculum or guidelines for the implementation of the programming language used in their province, and, therefore, has no idea what to teach her learners, or whether she is concentrating on the work that will be tested in the Matric examinations.

For this reason another respondent feels that curriculum documents and materials should be distributed to schools in the year prior to implementation, so that teachers can plan for the year ahead. This is not the case. Regulations are in some cases sent weeks before the implementation date, which is utterly unfair and counter-productive and unnecessarily adds to the already heavy workload of teachers. Referring to all the statements regarding pilot testing (5.7.9 - 5.7.11), this respondent is of the opinion that if pilot testing was done, the problem just mentioned might not occur and proper planning would be possible.

Commenting on this part of the questionnaire as a whole, one respondent stated that as far as implementation was concerned, they had good guidance from a particular individual in his capacity as teacher and examiner. However, from the side of the Department of Education they had no guidance in terms of in-service training. They attended a training course at a local university requested by them and at their own cost.

Concerns about costs are echoed by another respondent from a province where no teacher training was offered in the language being implemented. "Training was the teacher's responsibility and not that of the Department of Education."

Table 5.8 (see next page) presents some of the results from table 5.7 in a slightly simplified format by showing a ranking of statements according to the average score calculated for importance and usage based on respondents' indications. The table contains the ten statements that received the highest averages, as well as the statement that received the lowest average for importance and usage respectively. The statement in each of the two categories with the highest average obtained a ranking of 1 and the statement for the lowest a ranking of 30. Where the ranking in terms of usage for a specific statement differs greatly from that for importance, the usage ranking is shown in bold.

The statements with the highest, and third and fourth highest, averages with regard to usage similarly obtained relatively high averages for importance. However, other statements that showed high averages for importance obtained fairly low averages for usage, e.g. the statement with the third highest average for importance has the third lowest average with regard to usage, and even the statement rated as being the most important to use during implementation places on the verge of the bottom third for actual implementation. This type of pattern also holds for the

Table 5.8: Relative ranking of implementation guidelines

		Importance		Usage	
		Ave	Rank	Rank	Ave
26[6]	Provide teachers with guidance and support during implementation.	3.88	1	**19**	2.18
25	Distribute curriculum documents and materials to schools.	3.86	2	4	2.36
24	Give teachers the time and resources to be retrained properly.	3.84	3	**28**	2.06
6	Develop learning and teaching materials and resources to accompany and support a new curriculum.	3.81	4	**24**	2.14
7	Develop particularly textbooks suited to high school level to accompany and support a new curriculum.	3.79	5	3	2.38
8	State outcomes that need to be achieved clearly and in a concise way.	3.76	6	1	2.49
27	Continue support for teachers as implementation becomes established and consolidated into an accepted part of curricular practice.	3.75	7	**25**	2.13
15	Address teachers' knowledge of content and concepts in a new curriculum by providing practical advice about teaching.	3.74	8	**26**	2.10
4	Make guiding documentation about implementation of a programming language available to teachers.	3.73	9	10	2.25
21	Have educational leaders provide progressive leadership during curriculum improvement by mobilising administration and resources in support of teachers.	3.69	10	**20**	2.17
22	Create opportunities for teachers from different schools to discuss implementation.	3.58	19	**8**	2.26
14	Make teachers aware of the differences between old and new practices.	3.58	19	**9**	2.26
20	Identify factors/variables that will advance or slow down implementation.	3.58	22	**6**	2.30
5	Grant teachers the necessary freedom to implement at classroom level.	3.57	23	**5**	2.35
9	Perform preliminary testing of a new curriculum in the form of pilot testing.	3.50	24	30	2.04
3	Consider cost implications at macro-implementation (national/provincial) level.	3.46	27	**2**	2.40
1	Perform a situation analysis, using various methods for gathering information about the implementation process.	3.30	30	**7**	2.29

statements with the fifth through ninth highest averages in terms of usage, which all obtained averages for importance that places them in the bottom third. This is especially true for statement 5.8.3: according to respondents' indications, cost implications should be one of very last considerations when implementing a language, but it is in fact used second most.

[6] Numbers in this column of table 5.8 refer to the same numbers of items as used in tables 5.7, e.g. item 26 here is the same item as 5.7.26.

5.4.5 Miscellaneous statements

In the last section of the questionnaire used, respondents were presented with eight statements regarding the selection and implementation of a programming language (see Appendix A). They were then asked to indicate to what extent they (dis)agree with each of these statements. The results for this section are presented in table 5.9 (see next page).

Respondents most strongly agree with the statement that learners should be provided with the necessary tools and skills to adapt to a lifetime of change. If learners develop general thinking skills identifiable and transferable to other situations when learning a programming language, these learners would become life-long learners and should be able to prosper over the length of long careers.

Just more than half of respondents (52%) strongly agree that it is more important that a programming language facilitates the learning of programming principles than to learn a language learners will encounter in tertiary or private training.

Almost three-quarters of respondents (71%) agree to some extent that the programming knowledge that learners gain when studying Computer Studies provides them with the needed background to deal with modern society issues.

Although respondents generally agree to some extent that options for a language-independent examination need to be considered if different provinces implement different languages, they however disagree with reservation with the notion that the programming section currently contained in written papers should no longer be assessed externally, but instead only projects used for this purpose.

The following two comments were received from respondents regarding specific items in this section (numbering here refers to numbers of statements in table 5.9):

5.9.7 One respondent questions the notion that a programming problem can only be answered as written code in an answer book. Instead it might be possible to answer the whole paper in e.g. MSWord, with the programming section answered on a computer while the written examination is taking place. Although space and hardware might present a problem, printouts and/or screen captures with Windows could make it much easier.

5.9.8 Another respondent queries how the "science of implementation" can be differentiated from a specific language. Practical experience at this science is important, but not language specific details.

Table 5.9: Miscellaneous statements

			Strongly disagree	Disagree with reservations	Agree with reservations	Strongly agree	Average	Standard deviation
5.9.1	Programming knowledge provides needed background to deal with modern society issues.	f %	8 (7%)	27 (22%)	67 (54%)	21 (17%)	2.82	0.79
5.9.2	Learners develop general thinking skills identifiable and transferable to other situations.	f %	2 (2%)	7 (6%)	42 (34%)	72 (58%)	3.50	0.68
5.9.3	Learning the programming language should produce learners who are life-long learners.	f %	1 (1%)	11 (9%)	38 (31%)	73 (59%)	3.49	0.69
5.9.4	Learners should be provided with the necessary tools and skills to adapt to a lifetime of change.	f %	2 (2%)	4 (3%)	32 (26%)	84 (69%)	3.62	0.63
5.9.5	Learners should be able to prosper over the length of long careers.	f %	3 (2%)	5 (4%)	46 (38%)	68 (56%)	3.47	0.69
5.9.6	Options for a language-independent examination need to be considered as provinces implement different languages.	f %	13 (11%)	10 (8%)	48 (40%)	49 (41%)	3.11	0.96
5.9.7	The programming section currently contained in written papers should no longer be assessed externally, but instead only projects used for this purpose.	f %	34 (29%)	32 (27%)	41 (34%)	12 (10%)	2.26	0.99
5.9.8	It is more important that a programming language facilitates the learning of programming principles than to learn a language learners will encounter in tertiary or private training.	f %	4 (3%)	13 (11%)	41 (34%)	62 (52%)	3.34	0.80

5.4.6 Additional comments

At the end of the questionnaire respondents were given the opportunity to make additional general comments. The comments and remarks received in response are reported here.

- Implementation of a programming language is very important and also very dangerous. It does not help if the person(s) making the choice(s) do(es) not take into account what the learners as well as the teachers will be able to grasp. Personal preferences are sometimes a handicap. Dramatic changes can lead to dramatic chaos. He doesn't think that all companies and industry would discard their preferences so that only one kind of solution is available for custom designed applications. To implement a single programming language nationally, all students at tertiary institutions would need to be trained the same way and these institutions would not be able to have a preference. Teachers at schools must be trained intensively for a period of one or two years, so that all teachers can be at the same level of development. Only then can a new programming language be rolled out. A question is also posed regarding the average life span of a programming language in this day and age, in the light of the dramatic and quick changes among programming languages. This implies that a speed wobble while rolling out a programming language could also lead to chaos.
- Another respondent picks up on the theme that it is very important that all schools across the country use the same programming language, since students move between schools when parents move.
- Since they haven't introduced the language yet, it was difficult to complete the questionnaire, but he hopes that they will get better support from the Department of Education than in the past.
- She enjoys teaching a specific language used in that province and thinks it was a good choice. It is just difficult to teach at a double medium school where there are no textbooks available in Afrikaans and English.
- One respondent mentions funding, the retraining of educators, and that programming at schools should be language independent (psuedo-code/flowchart only).
- In the same vein, another respondent mentions teacher training, as well as the provision of equipment.
- Educators should be given ample time to master the use of a programming language prior to implementation - it should not be a rushed process. The cost of retraining educators should be fully borne by the employer (state).
- One respondent commented that opinions expressed in her completion of the questionnaire reflects the opinions of a teacher in a specific province, which would differ from those of teachers in other provinces where the programming language is different and where sufficient teacher training took place.

- She had not answered the right hand columns since she was never exposed to any decision making about the use of programming languages. When she was a lecturer at a teachers training college, she decided to introduce Java as programming language in 1998 solely because it was new and she personally had a desire to learn it, and nobody questioned her decision. She anticipated that the local Department of Education would probably change to C++ (which would not be too difficult to learn with a Java background) since they changed from Basic to Pascal at a stage when Pascal was phased out in industry in favour of C++. She thought that they would probably introduce C++ when industry rejects it in favour of Java, and figured that if the students were well trained in basic principles, they would be able to teach those principles, regardless of what language they have to use to do that.

This brings to an end the reporting of results obtained from respondents using the questionnaire. In the next part of the chapter a factor analysis of results is reported.

5.5 FACTOR ANALYSIS

Table 5.10 consists of extracts from tables 5.5 and 5.7. These extracts indicate that the ranges across averages with regard to importance for selection criteria and implementation guidelines are very similar. On the other hand, averages for the application of selection criteria and usage of implementation guidelines respectively have distinctly different ranges, which differ between application and usage, and also from those for importance.

Table 5.10: Minimum and maximum averages for separate items

	Selection criteria		Implementation guidelines	
	Importance	**Application**	**Importance**	**Usage**
Minimum	3.24	2.26	3.30	2.04
Maximum	3.83	3.04	3.88	2.49

Similarly, an investigation of the standard deviations of separate items shows that the standard deviations for all items are higher for the application/usage section than those for importance. This would imply that respondents rated the importance sections much more uniformly than those represented by the column on the right hand side of the questionnaire (see 5.4.3). Significant differences across responses would therefore more likely occur for the sections for the application of selection criteria and usage of implementation guidelines.

For this reason factor analysis was carried out with regard to these sections. Calculations and manipulation needed for the factor analysis, as well as other statistic results mentioned in the rest of this chapter, were implemented using the SAS program (SAS Institute Incorporated, 1999a). Results in this regard are subsequently reported and discussed.

5.5.1 Selection criteria

When a factor analysis was carried out on statements in the questionnaire with regard to the application of selection criteria, seven factors emerged, which explained 75.14% of the variance. These factors were then rotated according to the Varimax rotation method, producing the matrix represented in table 5.11 (see next page).

As all final communality estimates are larger that 0.5, the factors identified can be considered to be a successful extraction.

The variance explained by each factor is presented in table 5.12:

Table 5.12: Variance explained by factors - selection criteria

Factor 1	Factor 2	Factor 3	Factor 4	Factor 5	Factor 6	Factor 7
15%	13%	12%	12%	11%	6%	5%

When assigning individual questionnaire items to factors, those items that only show a factor loading under a single factor were obviously assigned to that factor. If an item showed more than one factor loading, it was assigned to the factor for which it shows the highest loading, with the exception of 5.11.12 and 5.11.41. The latter two items were grouped with factors where they fitted more logically. In table 5.11 the factor loadings shown in **bold** indicate assignment to a specific factor.

Each of the factors identified will now be presented in terms of a short heading, followed by a description of the items that make up the factor.

Factor 1: New tendencies in programming

> The chosen programming language should support new tendencies in programming by offering possibilities for object-oriented design, including the notions of encapsulation, inheritance and polymorphism, as well as the opportunity to make use of visual programming.

Factor 2: Developing thinking and programming skills

> The chosen programming language should adequately match the abilities of learners in that it provides an instructional environment promoting the development of problem solving abilities as well as higher order and critical thinking skills. In this way, it facilitates the development of learners' understanding of their own thought processes and encourages a self-regulated approach to solving problems. It should also support good programming style in that it encourages programming principles such as abstraction and promotes top-down design with step-wise refinement.[7]

[7] Text continues on page 128

Table 5.11: Factor analysis for the application of selection criteria

	F1	F2	F3	F4	F5	F6	F7	Selection criteria for a programming language for South African high schools should
5.11.1	.[8]	.	.	.	.	.	**0.55**	• have worldwide relevance
5.11.2	0.42	.	.	.	.	.	**0.71**	• be relevant for a reasonable period
5.11.3	.	.	.	.	.	.	**0.69**	• suit specific needs relevant to the South African context
								The programming language and its software development environment should
5.11.4	.	.	.	**0.76**	.	.	.	• suit the needs of novice programmers
5.11.5	.	.	.	**0.67**	.	.	.	• be safe, stable, structured and controlled
5.11.6	.	.	.	**0.69**	.	.	.	• not frustrate learners because of features suited to professional programmers
5.11.7	.	.	.	**0.64**	.	.	.	• provide effective debugging tools
5.11.8	.	.	.	**0.62**	.	.	.	• provide understandable error-messages
5.11.9	.	.	.	**0.80**	.	.	.	• be easy to learn
5.11.10	.	.	.	**0.77**	.	.	.	• offer relative simplicity of commands
								The chosen programming language should
5.11.12	.	**0.41**	.	0.47	.	.	.	• adequately match the abilities of learners
5.11.13	.	**0.62**	.	.	.	0.50	.	• provide an instructional environment promoting development of higher order thinking skills
5.11.14	.	**0.73**	.	.	.	0.46	.	• provide an instructional environment promoting development of critical thinking skills
5.11.15	.	**0.63**	.	.	.	.	.	• provide an instructional environment promoting development of problem solving abilities
5.11.16	0.47	**0.54**	.	.	.	.	.	• facilitate the development of learners' understanding of their own thought processes
5.11.17	.	**0.75**	.	.	.	.	.	• encourage a self-regulated approach to solving problems
5.11.18	0.40	**0.57**	.	.	.	.	.	• encourage programming principles such as abstraction.
5.11.19	.	**0.78**	.	.	.	.	.	• promote top-down design with step-wise refinement
5.11.20	.	**0.71**	.	.	.	.	.	• support good programming style
5.11.21	**0.53**	0.52	.	.	.	.	.	• support new tendencies in programming
5.11.22	**0.81**	.	.	.	.	.	.	• offer possibilities for object-oriented design
5.11.23	**0.77**	.	.	.	.	.	.	• offer possibilities for visual programming
5.11.24	**0.80**	.	.	.	.	.	.	• offer possibilities for encapsulation
5.11.25	**0.81**	.	.	.	.	.	.	• offer possibilities for inheritance
5.11.26	**0.82**	.	.	.	.	.	.	• offer possibilities for polymorphism

[8] Values less than 0.4 are not printed, but represented by a dot (.) in a cell

Table 5.11: Factor analysis for the application of selection criteria (continued)

	F1	F2	F3	F4	F5	F6	F7	The chosen programming language should
5.11.27	0.44	.	**0.64**	.	.	.	.	• be internationally standardised
5.11.28	.	.	**0.69**	.	.	.	.	• have reasonable prospects for continued support from its developers
5.11.29	.	.	**0.78**	.	.	.	.	• enable Internet programming
5.11.30	0.43	.	**0.50**	.	.	.	.	• support database programming
5.11.31	.	.	**0.48**	.	.	.	.	• follow international trends with regard to programming languages used in highschool education
5.11.32	.	0.48	.	.	**0.53**	.	.	• be affordable
5.11.33	.	.	.	.	**0.82**	.	.	• enable affordable in-service training of CS teachers
5.11.34	.	.	**0.66**	.	0.51		.	• be compatible with tertiary establishments' choice of programming language
5.11.35	.	.	.	.	**0.70**	.	.	• be included in the pre-service training of CS teachers
5.11.36	.	.	.	.	**0.80**	.	.	• have learning and teaching support materials and resources, such as question banks, instructional packages, workbooks and study guides, available
5.11.37	.	.	.	.	**0.73**	.	.	• have particularly textbooks suited to learning the language at high school level available
5.11.38	.	.	0.43	.	**0.53**	.	.	• have resources appropriate to an OBE approach to the subject obtainable
5.11.39	.	.	.	.	.	**0.59**	.	• support programming for various purposes
5.11.40	.	.	0.48	.	.	**0.55**	.	• be able to be used with academic tools
5.11.41	.	.	0.51	.	.	**0.46**	.	• be able to be used with commercial tools
5.11.42	.	.	**0.63**	.	.	.	.	• be popular in industry, where there is a demand for such programmers

Factor 3: Issues influencing programming used in practice

In terms of the software development process the chosen programming language should be internationally standardised and have reasonable prospects for continued development from its developers. The language should be suitable and have capabilities for working on and with the Internet and support database programming. International trends with regard to programming languages used in high school education in other parts of the world should be given consideration, together with compatibility with tertiary establishments' choice of programming language and the popularity and/or demand for specific languages in industry.

Factor 4: Requirements for the programming language and its software development environment to make it appropriate for learners

The programming language and its software development environment should be suitable for novice programmers in that it is easy to learn. Learners should be provided with a safe, stable, structured and controlled programming language and environment that does not frustrate them because of features suited to professional programmers. A suitable software development environment should provide effective debugging tools and understandable error-messages. It should also offer relative simplicity of commands.

Factor 5: Affordability, training and resources

The chosen programming language should not only be affordable, but should also enable affordable in-service training, and be included in the pre-service training of CS teachers. It is imperative that learning and teaching support materials and other resources, particularly textbooks, for the language should be available to teachers. These resources should specifically be appropriate to an OBE approach to the subject.

Factor 6: Programming for various purposes

A general purpose programming language that supports many academic and commercial tools should be used.

Factor 7: Relevance of selection criteria

When establishing criteria for selecting a first programming language for South African high schools, the emphasis will be on setting selection criteria that will be universal, in that these will have worldwide relevancy in all circumstances. Set criteria should also not date, but need to be relevant to select a programming language for a reasonable period. However, these criteria should suit specific needs relevant to the South African context.

The factor analysis carried out with regard to statements in the questionnaire concerning the usage of implementation guidelines are subsequently presented.

5.5.2 Implementation guidelines

When a factor analysis was carried out on statements in the questionnaire with regard to the usage of implementation guidelines, four factors were identified, which accounts for 72.91% of variance in data. These factors were then rotated according to the Varimax rotation method, producing the matrix represented in Table 5.13 (see next page).

As all but one of the final communality estimates are larger that 0.5, the factors identified here can be considered to be a successful extraction. The only statement with a final communality estimate of less than 0.5 (0.37) was 5.13.3 regarding cost implications, the handling of which is described shortly.

The variance explained by each factor is presented in table 5.14:

Table 5.14: Variance explained by each factor - implementation guidelines

Factor 1	Factor 2	Factor 3	Factor 4
23%	19%	18%	13%

Items with regard to implementation guidelines were assigned to factors in the same way as described for selection criteria (see 5.5.1). One of the two items that were not assigned to the highest factor loading is 5.13.7, which was grouped with a factor where it fitted more logically. The other, 5.13.3, has not only already been mentioned as having a low final communality estimate, but also shows a fairly low factor loading – it was decided to treat this item as a separate additional factor.

Each of the factors identified will now be presented in terms of a short heading, followed by a description of the items that make up the factor.

Factor 1: Introducing a new curriculum

Specifying and classifying learning content that is of consequence to learners can initiate the development of a curriculum. Factors and variables that could advance or slow down implementation need to be identified, such as the extent to which educational leaders provide progressive leadership during curriculum improvement by mobilising administration and resources in support of teachers and create opportunities for teachers from different schools to discuss implementation. Gatherings of teachers can also be used to gauge teachers' expectations for implementation, make the relative advantage of reform clear to teachers, make teachers aware of the differences between old and new practices and to address teachers' knowledge of content and concepts in a new curriculum by providing practical advice about teaching. Information regarding a new curriculum needs to be distributed and sources of resistance to change and obstacles to implementation identified, handled in such a manner that implementation can proceed smoothly.

Table 5.13: Factor analysis for the use of implementation guidelines

	F1	F2	F3	F4	
5.13.1	.	.	0.52	**0.53**	Perform a situation analysis, using various methods for gathering information about the implementation process.
5.13.2	.	.	.	**0.73**	Show appreciation for teachers' roles in implementation at all levels in curriculum process.
5.13.3	0.40	.	.	.	Consider cost implications at macro-implementation (national/provincial) level.
5.13.4	.	.	.	**0.58**	Make guiding documentation about implementation of a programming language available to teachers.
5.13.5	.	.	.	**0.82**	Grant teachers the necessary freedom to implement at classroom level.
5.13.6	.	0.40	0.42	**0.55**	Develop learning and teaching materials and resources to accompany and support a new curriculum.
5.13.7	.	0.40	0.56	**0.47**	Develop particularly textbooks suited to high school level to accompany and support a new curriculum.
5.13.8	.	0.42	**0.43**	.	State outcomes that need to be achieved clearly and in a concise way.
5.13.9	.	.	**0.78**	.	Perform preliminary testing of a new curriculum in the form of pilot testing.
5.13.10	.	.	**0.86**	.	Participation of teachers in pilot testing takes place on a voluntarily basis.
5.13.11	.	.	**0.87**	.	Use experienced teachers to conduct pilot testing.
5.13.12	.	.	**0.68**	.	Carry out program assessment in order to revise and improve a new curriculum to a standard suitable for implementation.
5.13.13	.	0.45	**0.61**	.	Convince teachers that their implementation efforts will be supported.
5.13.14	**0.71**	.	.	.	Make teachers aware of the differences between old and new practices.
5.13.15	**0.72**	.	.	0.41	Address teachers' knowledge of content and concepts in a new curriculum by providing practical advice about teaching.
5.13.16	**0.69**	.	.	.	Gauge teachers' expectations for implementation.
5.13.17	**0.63**	.	.	.	Specify and classify learning content that is of consequence to learners to initiate the development of a curriculum.
5.13.18	**0.68**	.	.	.	Distribute information regarding a new curriculum in a way that will overcome resistance to change.
5.13.19	**0.64**	0.41	.	.	Make the relative advantage of reform clear to teachers.
5.13.20	**0.68**	0.44	.	.	Identify factors/variables that will advance or slow down implementation.
5.13.21	**0.71**	.	.	.	Have educational leaders provide progressive leadership during curriculum improvement by mobilising administration and resources in support of teachers.
5.13.22	**0.55**	0.45	.	.	Create opportunities for teachers from different schools to discuss implementation.
5.13.23	**0.74**	.	.	.	Handle sources of resistance to change and obstacles to implementation identified in such a manner that implementation can proceed smoothly.

Table 5.13: Factor analysis for the usage of implementation guidelines (continued)

	F1	F2	F3	F4	
5.13.24	.	**0.55**	.	.	Give teachers the time and resources to be retrained properly.
5.13.25	.	**0.78**	.	.	Distribute curriculum documents and materials to schools.
5.13.26	0.42	**0.78**	.	.	Provide teachers with guidance and support during implementation.
5.13.27	0.42	**0.69**	.	.	Continue support for teachers as implementation becomes established and consolidated into an accepted part of curricular practice.
5.13.28	0.42	**0.77**	.	.	Involve teachers in the assessment of a new curriculum.
5.13.29	0.45	**0.63**	.	.	Assess a new curriculum according to the extent to which its implementation leads to the achievement of learning outcomes.
5.13.30	.	**0.64**	.	.	Determine to what extent a new curriculum was indeed implemented.

Factor 2: Guidance, support and assessment

Provide teachers with guidance and support before implementation by giving them the time and resources to be retrained properly and distributing curriculum documents and materials to schools. Support for teachers should be continued as implementation becomes established and consolidated into an accepted part of curricular practice. Teachers can also be involved in the assessment of a new curriculum, both in assessing it with regard to the extent to which its implementation leads to the achievement of learning outcomes and determining to what extent it was indeed implemented.

Factor 3: Issues surrounding pilot testing

As soon as outcomes that need to be achieved have been stated clearly and in a concise way, preliminary testing of a new curriculum can take place in the form of pilot testing, using volunteer, experienced teachers. Program assessment is supposed to be carried out in order to revise and improve the curriculum to a standard suited to implementation. Appropriate implementation of these steps can convince teachers that their implementation efforts will be supported.

Factor 4: Appreciating teachers' roles in implementation

Performing a situation analysis, using various methods for gathering information about the implementation process can aid the development of learning and teaching materials and resources, particularly textbooks suited to high school level, to accompany and support a new curriculum. These steps, together with making guiding documentation about the implementation of a programming language available to teachers, show appreciation for teachers' roles in implementation at all levels in the curriculum process and grant them the necessary freedom to implement at classroom level.

Additional factor[9]:

Consider cost implications at macro-implementation (national/provincial) level.

5.5.3 Cronbach Alpha reliability of factor analysis

In terms of the Cronbach Alpha reliability of the factor analysis, the results obtained for raw and standardised scores were indistinguishable when rounded to two meaningful digits. These scores are presented in table 5.15 (see next page).

An acceptable reliability coefficient is suggested to be 0.70 (SAS Institute Incorporated, 1999b: 295). As all values obtained in this case are above 0.8, these factors represent reliable measuring instruments.

[9] See explanation 5.5.2

Table 5.15: Cronbach Alpha reliability of factor analysis

Application of selection criteria		**Reliability**
Factor 1	New tendencies in programming	0.96
Factor 2	Developing thinking and programming skills	0.93
Factor 3	Issues influencing programming used in practice	0.92
Factor 4	Requirements of programming environment for learners	0.90
Factor 5	Affordability, training and resources	0.90
Factor 6	Programming for various purposes	0.91
Factor 7	Relevance of selection criteria	0.81
Usage of implementation guidelines		
Factor 1	Introducing a new curriculum	0.95
Factor 2	Guidance, support and assessment	0.94
Factor 3	Issues regarding pilot testing	0.92
Factor 4	Appreciating teachers' roles in implementation	0.89

5.5.4 Differences between importance and application/usage per factor for groups

As indicated earlier (see 5.5) there are distinct differences between the ratings that respondents gave to items with regard to importance and application/usage respectively. To determine the extent of these differences, the difference between importance and application/usage were calculated for each factor. The effect size is obtained by dividing the difference between each set of means (mean: importance and mean: application/usage) by the standardised standard deviation for the differences, in the formula $d = \frac{\overline{x}_{diff}}{s_{diff}}$ (Steyn, 1999:3). Guidelines for the interpretation of the effect size obtained from Cohen (1988:25) are portrayed in table 5.16:

Table 5.16: Effect sizes for different values of d

Value of d	**Effect size**	**Indicator**
d < 0.2	insignificant	-
d ≈ 0.2	small effect	•
d ≈ 0.5	medium effect	**
d ≈ 0.8	large effect	***

Values for d ≥ 0.8 are considered to be practically significant, as a result of there being a large enough difference to have an effect in practice.

When the differences between importance and application/usage for factors regarding both selection criteria and implementation guidelines were calculated for each factor for all respondents, effect sizes with practical significance were obtained for all factors (see table 5.17 next page).

Table 5.17: Effect sizes for differences between importance and application/usage per factor for groups

	Selection criteria								Implementation guidelines				
	New tendencies in programming	Developing thinking and programming skills	Issues influencing programming used in practice	Programming environment requirements for learners	Affordability, training and resources	Programming for various purposes	Requirements that selection criteria have to satisfy		Introducing a new curriculum	Guidance, support and assessment	Issues regarding pilot testing	Appreciating teachers' roles in implementation	Considering cost implications
All respondents	0.82	0.73	1.00	0.74	1.30	0.97	0.79	Mean	1.42	1.54	1.41	1.38	1.07
	1.01	0.69	0.90	0.70	0.83	0.88	0.90	Standard deviation	0.86	0.91	0.89	0.79	1.18
	0.81	1.07	1.11	1.07	1.58	1.10	0.87	d	1.65	1.70	1.58	1.75	0.91
	***	***	***	***	***	***	***	Effect	***	***	***	***	***
Group 1: Policy-makers	0.35	0.46	0.37	0.61	0.62	0.43	0.51	Mean	0.83	0.92	0.76	0.81	0.47
	0.70	0.51	0.51	0.70	0.75	0.47	0.68	Standard deviation	0.92	0.93	0.79	0.61	0.94
	0.50	0.92	0.72	0.87	0.83	0.92	0.75	d	0.89	0.99	0.96	1.32	0.50
	**	***	**	***	***	***	**	Effect	***	***	***	***	*
Group 2: Teachers	1.01	0.81	1.16	0.76	1.46	1.10	0.88	Mean	1.54	1.67	1.56	1.51	1.20
	1.06	0.72	0.89	0.72	0.77	0.89	0.93	Standard deviation	0.85	0.9	0.88	0.81	1.27
	0.95	1.12	1.31	1.05	1.90	1.24	0.95	d	1.82	1.86	1.78	1.86	0.95
	***	***	***	***	***	***	***	Effect	***	***	***	***	***
Group 3: Trainers	0.39	0.64	0.85	0.82	1.27	0.86	0.61	Mean	1.44	1.52	1.40	1.36	1.13
	0.73	0.61	0.96	0.60	0.86	0.94	0.94	Standard deviation	0.59	0.67	0.79	0.59	0.81
	0.54	1.05	0.89	1.37	1.47	0.92	0.65	d	2.46	2.26	1.76	2.30	1.40
	**	***	***	***	***	***	**	Effect	***	***	***	***	***

• indicates a small effect that is of no practical significance

** indicates a medium effect – a difference between groups that might be of practical significance

*** indicates a large effect – a difference between groups that is large enough to be important in practice

Grouping was then implemented in order to establish whether there were distinguishing patterns of differences amongst certain respondents. Respondents were divided into policy makers (those who indicated involvement in the Writing Committee for the National Curriculum Statement, as Curriculum Advisors and/or who had been involved in an administrative capacity at provincial level), trainers of Computer Studies teachers, and teachers. In terms of the size of the different groups, the policy makers group consisted of 18 persons (15%) and the group of trainers of 19 (also 15%), while 87 teachers made up the rest (70%).

With regard to these groupings, differences between importance and application/usage for factors for both selection criteria and implementation guidelines still show an effect with practical significance for the vast majority (83%) of differences. Exceptions are medium effects (**) shown for the first and last factors for selection criteria for both policy makers and trainers, as well as a medium effect for the third factor for policy makers. The only small effect shown (please note that the actual number is 0.4989) is for cost implications by policy makers – this would indicate that there is little difference between the rated importance and usage of policy makers for cost implications when implementation takes place.

5.5.5 Significance of differences between groups

Results reported above indicate differences between the way in which the importance and application/usage respectively of factors were rated, which were also considered separately for three different groups. This section looks closer at how much difference there was between the different groups in the way that each group rated the importance, application and usage of each factor.

Effect sizes were calculated using the formula $d = \frac{|\bar{x}_i - \bar{x}_j|}{\sqrt{MSE}}$ (Steyn, 1999:3) with the difference between each pair of group means divided by the square root of the mean square error (RMSE) of analysis of variance as obtained from ANOVA test.

Differences between group means were only calculated for those factors for which the values of the F-ratio exceeds the critical value at 10% probability levels ($p < 0.1$)[10], as this was considered to indicate that there is a statistically significant difference between at least one pair of means. The results for these factors are reported in table 5.18 (see next page), with the interpretation of effect size for each difference supplied according to the guidelines given in table 5.16.

With the exception of two factors, the largest effects for the differences between groups occur for the way in which policy makers and teachers respectively rate the importance, application and usage of factors. Four of the factors regarding the usage of implementation guidelines exhibit differences with practical significance between these two groups, while another seven

[10] In fact, only two exceed $p < 0.05$, while five have values consistent with $p < 0.01$.

Table 5.18: Significance of differences between groups

				Mean			Differences					
							Policymakers - Teachers		Teachers -Trainers		Policymakers - Trainers	
	Factor	Section	ANOVA RSME	Policy-makers	Teachers	Trainers	d	Effect	d	Effect	d	Effect
5.18.1	Affordability, training and resources	Importance	0.35	3.47	3.73	3.69	0.73	**	0.09	-	0.64	**
5.18.2	New tendencies in programming	Application	0.90	3.04	2.49	3.13	0.61	**	0.71	**	0.10	-
5.18.3	Developing thinking and programming skills	Application	0.63	3.13	2.75	2.99	0.60	**	0.37	*	0.23	*
5.18.4	Issues influencing programming used in practice	Application	0.75	2.96	2.37	2.61	0.79	**	0.32	*	0.48	*
5.18.5	Affordability, training and resources	Application	0.71	2.85	2.30	2.46	0.78	**	0.23	*	0.55	**
5.18.6	Programming for various purposes	Application	0.80	2.96	2.41	2.67	0.69	**	0.32	*	0.37	*
5.18.7	Appreciating teachers' roles in implementation	Importance	0.35	3.53	3.62	3.79	0.27	*	0.49	*	0.76	**
5.18.8	Introducing a new curriculum	Usage	0.70	2.76	2.08	2.38	0.98	***	0.44	*	0.54	**
5.18.9	Guidance, support and assessment	Usage	0.76	2.80	2.07	2.30	0.97	***	0.30	*	0.67	**
5.18.10	Issues regarding pilot testing	Usage	0.76	2.64	2.05	2.31	0.78	**	0.35	*	0.44	*
5.18.11	Appreciating teachers' roles in implementation	Usage	0.65	2.75	2.13	2.46	0.95	***	0.51	**	0.44	*
5.18.12	Considering cost implications	Usage	0.83	3.00	2.23	2.56	0.93	***	0.40	*	0.53	**

- indicates an insignificant effect

* indicates a small effect

** indicates a medium effect – a difference between groups that might be of practical significance

*** indicates a large effect – a difference between groups that is large enough to be important in practice

factors show differences with medium effect. These relatively large differences could be explained by considering that policymakers' views represent the intended importance, application and usage of various factors, but they might be far removed from the actual situation as perceived by teachers at school level.

Differences between the way in which teachers and their trainers rate factors are generally small, with only two factors showing differences with medium effect. One of these, regarding the application of new tendencies in programming, are in fact the largest of the three effects for this factor, while the difference between trainers and policy makers is insignificant. This result might be explained by considering that items included in this factor, such as object-oriented design, encapsulation, inheritance, polymorphism and visual programming, could be more familiar to people in the policy makers and trainers groups, while teachers, especially in those provinces where implementation of a new language have only recently started, have not had much contact with these concepts. These would also be the concepts that teachers need the most training in.

Between policymakers and trainers, effect sizes for differences are spread equally between small and medium effects. The only factor from the implementation guidelines for which the importance was considered in this section, regarding the appreciation of teachers' roles in implementation, shows the largest effect for the difference between trainers and policymakers. If a situation analysis of the implementation process is not carried out, not only will teachers' roles in implementation at all levels in the curriculum not be appreciated, but problems such as a lack of guiding documentation about the implementation and textbooks suited to high school level, will only be revealed when teachers come together, as would be the case at training opportunities.

5.6 CONCLUSION

In this chapter the empirical study programme was introduced, including the aim of the empirical study, the background to the study and the procedure followed in terms of permission needed to obtain data. Next, the research design in terms of the questionnaire used and procedure for gathering data was discussed, together with the response rate.

Results as obtained from the various sections of the questionnaire used were subsequently represented. Firstly, a demographic description of respondents was given in terms of their provincial spread, highest academic and educational qualifications, as well as highest qualifications in Computer Science. The number of years of respondents' involvement with Computer Studies was also provided.

Statements regarding selection criteria for a first programming language for South African high schools were presented according to the average score calculated for importance and application, based on respondents' indications. The statement regarding the provision of an instructional environment promoting the development of problem solving abilities obtained the

highest average in terms of both importance and application, while a statement with regard to enabling Internet programming received the lowest average for importance and the second lowest for application. Although the two items just mentioned, as well as several others not cited here again, ranked fairly consistently across importance and application, there were also numerous items that ranked high on one scale and very low on the other.

When a similar ranking as described for selection criteria was used for implementation guidelines, comparable results were obtained, i.e. some items ranked fairly alike for both importance and usage, while others were quite dissimilar. An example of the latter is a statement regarding cost implications: according to respondents' indications, this should be one of very last considerations when implementing a language, but it is in fact used second most.

Regarding the section in the questionnaire containing miscellaneous statements, respondents most strongly agreed with the statement that learners should be provided with the necessary tools and skills to adapt to a lifetime of change. They however disagreed with the notion that the programming section currently contained in written papers should no longer be assessed externally, but only projects used for this purpose instead.

Additional comments received from respondents centre around support for teachers from the various provincial education departments, particularly with regard to the provision of retraining for educators, and the cost thereof, as well as a need for textbooks. Although some respondents made a case for the use of the same programming language in all schools across the country, another argued that if pre-service teachers are well trained in basic principles, they would be able to teach those principles, regardless of what language they have to use to do so.

A factor analysis for the application of selection criteria and usage of implementation guidelines was undertaken, yielding the following factors:

Application of selection criteria

1. New tendencies in programming
2. Developing thinking and programming skills
3. Issues influencing programming used in practice
4. Requirements of programming environment for learners
5. Affordability, training and resources
6. Programming for various purposes
7. Relevance of selection criteria

Usage of implementation guidelines

1. Introducing a new curriculum
2. Guidance, support and assessment
3. Issues regarding pilot testing
4. Appreciating teachers' roles in implementation

The amount of variance explained by the factors for both selection criteria and implementation guidelines were shown, and the Cronbach Alpha reliability of the factor analysis was confirmed. One additional item factor, regarding cost implications, was also identified and used in subsequent manipulations.

An investigation into the differences between the importance and application/usage per factor for groups consisting of policy makers, teachers and trainers shows effect sizes with practical significance for the majority of differences. Lastly, four of the factors examined exhibit differences with practical significance between the way in which policy makers and teachers respectively rate the usage of factors for implementation guidelines.

In the last chapter, which follows, a final summary of research is presented. Results obtained in the empirical study are compared to selection criteria and implementation guidelines identified in chapters 3 and 4. Implications of the study for the selection and implementation of a first programming language in high schools are examined, as well as recommendations made regarding these processes.

CHAPTER 6

FINAL SUMMARY OF RESULTS, IMPLICATIONS AND RECOMMENDATIONS

6.1 INTRODUCTION

In this chapter a final summary of research is presented. Selection criteria and implementation guidelines identified in chapters 3 and 4 are compared to results obtained in the empirical study, and implications of the study for the selection and implementation of a first programming language in high schools are presented. Recommendations are also made regarding the processes for selecting and implementing a first programming language in high schools.

6.2 AIMS OF THE STUDY AND RESEARCH METHOD

This study was undertaken in order to

- establish criteria that should be considered when selecting a first programming language to teach in high schools
- introduce guidelines for the implementation of a first programming language in high schools
- verify the validity of selection criteria and implementation guidelines identified empirically within the South African context

Through a literature study criteria were established that should be considered when selecting a first programming language in high schools, and guidelines for the implementation of such a language introduced.

Care was taken to ensure the relevance of criteria for the selection of a first programming language to teach in high schools. Criteria were established with regard to

- the development of thinking and programming skills
- requirements for the programming language and its software development environment to make it appropriate for learners
- new tendencies in programming
- issues influencing programming used in practice
- affordability, training and resources
- programming for various purposes

Guidelines were introduced for the implementation of a first programming language in high schools regarding

- appreciating teachers' roles in implementation
- issues surrounding pilot testing
- considering cost implications at macro-implementation level
- introducing a new curriculum
- guidance, support and assessment

A descriptive study was designed and carried out in order to verify the validity of criteria and guidelines identified empirically within the South African context. By utilising a questionnaire completed by role players involved at different levels in various structures connected to Computer Studies, data was gathered to determine

- how important various respondents rated different criteria for selection when choosing, and guidelines when implementing, a programming language for South African high schools
- to what extent various criteria and guidelines have been applied and used to select and implement programming languages for South African high schools

The validity of all selection criteria and implementation guidelines identified in earlier chapters were confirmed in the empirical study, as all items based on these obtained averages for importance of between 3.24 and 3.88 (see table 5.10) which in terms of the scale used in the questionnaire (see appendix A), classify all as 'fairly important'.

All aims set for this study have thus been achieved.

However, the extent to which criteria was applied for language selection and guidelines used for the implementation of languages was fairly low, and the correspondence between the rated importance of specific items and the extent to which these were applied or used varied widely.

Averages for both selection criteria and implementation guidelines are also spread over a range of values within the classification above, and based on these different averages, criteria and guidelines can be arranged as being more important relative to others for those with the highest averages and less important for those that obtained the lower averages within the respective ranges.

During the selection process of a first programming language for high schools, as well as during the subsequent implementation of such a language, participants in these actions would do well to pay close attention to those criteria and guidelines identified as most important. At the other end of the scale, time and effort could potentially be saved by assigning less importance to criteria and guidelines with lower averages.

In the next sections a closer examination of the selection criteria and implementation guidelines as developed from the literature study, how these relate to results obtained in the empirical study, and what implications the combined results have for the selection and implementation of a first programming language in high schools, will be presented. Where applicable, suggestions are also made regarding further investigations that could be considered in light of these findings.

Most of the limitations of this study would relate to the relevance of selection criteria to a specific situation other than the South African context used – these are addressed in the first part of the next section. It should be noted that in the following discussions text representing items from the questionnaire used is presented in *italics* to indicate said items.

6.3 RESULTS FOR SELECTION CRITERIA, IMPLICATIONS AND RECOMMENDATIONS

6.3.1 Relevance of selection criteria

When establishing criteria for selecting a first programming language for South African high schools, emphasis could be placed on setting selection criteria that

- will be universal, in that these will have worldwide relevancy in all circumstances
- should not date, but would still be relevant to select a first language for a reasonable period
- although general in terms of worldwide relevancy, should, in light of the context of this study, specifically suit needs relevant to the South African context

The three items in the questionnaire regarding the relevance of selection criteria received averages for importance that places them in the lowest quarter of results – specifically the item regarding suiting needs relevant to the South African context has the third lowest average of all items in the sections for selection criteria. In terms of application, the item with regard to relevance for a reasonable period received an average slightly higher than those for the other two items, but all three items still places in the lower half of the results.

According to the respondents in the empirical study, the relevance of selection criteria comprised some of the least important issues and was also applied very little. However, if selection criteria established in this study were to be used in situations other than the South African context applicable to this study, some of these might need to be reconsidered.

Criteria with universal relevancy ought to be applicable worldwide, while many aspects contemplated within the South African context, including financial considerations, training and support of teachers and resources available to teachers, would probably remain valid for most situations. Other criteria regarding the outcomes expected of the subject, the OBE orientation of resources, general-purpose programming, demand and training for industry, tertiary establishments and their expectations and external assessment and options for a language-independent examination, could be considered depending on the applicable context. Finally, the relevance of criteria in the 'new trends' section would depend on the amount of time that has expired since the completion of the current study.

6.3.2 Developing thinking and programming skills

Compared to other items regarding selection criteria in the questionnaire, respondents felt that the adequate matching of the adopted programming language to the abilities of learners of Computer Studies with regard to both level and nature is one of the least important items. Its average for application however places it in the top half of items. Despite the low importance awarded to this item by respondents, its application shows that the fact that the needs, knowledge and abilities of learners as computer users differ from those of experienced programmers, should be borne in mind. This then boils down to the question of which programming language would be most beneficial to learners' development.

At high school level, learners' cognitive development levels are growing towards the formal operational stage. It has been argued (see 3.3) that the subject Computer Studies serves as a problem solving discipline and a tool for thinking and reasoning, centred on the development of solutions to problems. In line with this stated purpose of the subject, respondents agree that the chosen language should *provide an instructional environment that promotes the development of higher order thinking and problem solving skills, as well as critical thinking.* Specifically, the averages for the item regarding the development of problem solving skills distinguish it as both the most important and most applied item. These findings not only provide support for continued programming language instruction at high school level, but also offer a means for improving the problem solving skills for the group most often criticised for their lack of problem solving ability.

Concern for the promotion of critical thinking skills among learners is growing, as a person's ability to think critically is not only recognized as an important and pervasive educational outcome, but it is also known to be a key success factor in many kinds of occupations. It is therefore important to get learners at this development stage focused on critical and creative thinking skills. Respondents agreed with this sentiment in that items regarding the development of higher order and critical thinking skills, as well as *encouraging a self-regulated approach to solving problems* by promoting strategies for analysing programming problems and formulating solutions to them, achieved averages placing them in or close to the top third of items for both importance and application.

As the acquisition of the mentioned cognitive skills is facilitated by high degrees of metacognition, the language should facilitate the use of the metacognitive components of problem solving during instruction. The average for an item that refers to the development of *learners' understanding of their own thought processes* and their involvement in metacognitive behaviour, places it in the middle relative to others with regard to both importance and application.

Almost two-thirds of respondents strongly agreed that when the improvement of problem solving and critical thinking abilities is a desired outcome, the role of programming in Computer Studies instruction is justified and strengthened by the fact that programming instruction both *develops learners' general* cognitive *skills* in an identifiable way and makes *transfer* of these abilities to multiple *other* problem *situations* possible. Another statement in this regard, that *programming knowledge provides needed background to deal with modern society issues*, received an 'agree with reservations' rating from more than half of respondents.

Due to the didactical principles involved in the teaching and learning of the subject, consideration should be given to whether a particular language used in a first programming course complies with educational aims with regard to preparing learners to have a firm foundation in good programming style for well-designed programs. Respondents supported this

opinion, as an item in this regard was applied second most and is the fifth most important selection criterion.

Another primary purpose of using a specific language should be to teach underlying programming principles, as in a field that changes so rapidly, skills quickly date and programming competencies only are not enough any more. If programmers were well trained in basic principles, they would be able to implement these principles regardless of what language they have to use to do that (see 5.4.6). *Encouraging programming principles such as* procedure and data *abstraction* and *promoting top-down design with step-wise refinement* are some of the least important items, but were applied much more, with the latter placing in the top quarter of averages for application. It is possible that the low importance afforded to abstraction and the promotion of top-down design by respondents might be due to some respondents not being aware of the integral role that abstraction plays in OOP (see 2.3) on the one hand, and/or being under the impression that using top-down design forms part of the 'old' way of programming. These notions could be investigated in subsequent investigations, and if proven to have substance, need to be addressed during training sessions.

Another consequence of the rapid evolution of the field (see the start of the previous paragraph) is that almost 70% of respondents strongly agreed with the statement that *learners should be provided with the necessary tools and skills to adapt to a lifetime of change* (see 3.3.6). More than 55% of respondents also strongly agreed that some of the most important outcomes of a Computer Science programme should be that *learning* about *the programming language* would produce *learners who are life-long learners* by ensuring that *learners* have the flexibility and broad-mindedness necessary to adapt and *prosper over the length of long careers.*

6.3.3 Requirements for the programming language and its software development environment to make it appropriate for learners

Arguments have been presented in favour of using a language with a high level of control and structure (see 3.3.4), which agrees with results for the items with the third, fourth and fifth highest averages for application. These items refer to learners being provided with a *safe, stable, structured and controlled* programming language and a suitable software development environment containing compilers that simplify learners' tasks, in that they supply well-defined and *easily understandable error messages* and offer *effective debugging tools* to locate and correct errors. Although these three items received averages for importance that place them slightly lower, they still place within the top third of results. The relatively high rankings of said items for both importance and application are also advisable in the light of the fact that the compilation and coordination of the components of a program represent some of the greatest stumbling blocks for novice programmers – effective debugging tools and easily understandable error messages will aid learners in these tasks.

Items with regard to the programming language and its software development environment being *suitable for novice programmers* in that it is *easy to learn* and sufficiently simple to work with in offering *relative simplicity of commands* (see 3.3.5), as well as *not frustrating learners because of* unnecessary difficulties and *features suited to professional programmers* (see 3.3.4) and the programming environment not being too expansive, are not considered as important, as the averages for these items place them in the lower half of results. However, these items were applied to a greater extent than would be expected for low importance, as application averages for these items place them in or close to the top third of results for application. Results for application conform much better with recommendations that the properties that these items refer to should be encouraged for use at high school level, as a simple and clear context can promote learners' development of high-level thinking skills.

6.3.4 New tendencies in programming

Supporting new tendencies in programming (see 3.4) is considered to be of moderate importance and to have been applied to the same extent, as averages for this item for both importance and application fall just inside the upper half of results.

With regard to topics with current relevance, OOP is considered to be one of the presently accepted problem solving and programming paradigms being used (see 3.4.1), and the chosen programming language *offering possibilities for object-oriented design* received averages for importance and application placing it in the top quarter of results for both these sections. Warnings indicating that programmers experience difficulty in learning to program in an object-oriented style and that object-oriented design is not easy (also see 2.3.3.1) should however be carefully taken into account. Teachers also need to exercise special care in introducing OO material in a way that limits the detail complexity involved in many of the languages used for object-oriented programming in industry, as such details could easily overwhelm learners.

Offering learners the opportunity to make use of *visual programming* received averages for importance and application placing this item just inside the lower half of results. However, using a visual language that can provide learners with a visual vocabulary, which they can apply and understand immediately (see 3.4.6), could offset some of the difficulties with regard to OOP mentioned in the previous paragraph.

The concepts of *encapsulation, inheritance* and *polymorphism* were considered to be some of the least important selection criteria, with averages placing them in the lowest quarter of results. These items, although applied slightly more, still obtained averages placing them in the lower half of application. These results could be explained in a variety of ways:

- Using techniques such as encapsulation and inheritance provide for a means of organising software systems in industry (see *2.3.3.3*), which might not be necessary at school level, as such large systems are not built.

- Especially teachers might not be very familiar with these concepts yet, as these occur in the curriculum (in terms of when these would be taught) towards the end of the last school year - in provinces where languages have only recently been implemented, this content would not have been reached yet.
- The need to use these techniques will not become apparent until user-defined objects are implemented, which is considered to be a difficult topic (see 3.4.1).

A coping strategy that teachers sometimes employ (see *4.5.2.4*) entails concentrating on areas in which confidence is highest, while avoiding those subject units that they do not feel comfortable teaching (because they are not familiar with it or they regards it as difficult), or those that they do not view as significant. Could it be that teachers do not view encapsulation, inheritance and polymorphism as significant?

Further investigations could be launched to explore the validity of the above notions. The extent to which encapsulation, inheritance and polymorphism would eventually be used as part of selection criteria would depend on the outcomes of such investigations, as well as the importance afforded to the employment of user-defined objects by role players in a curriculum project.

6.3.5 Issues influencing programming used in practice

In terms of the software development process (see 3.4.3), the chosen programming language having *reasonable prospects for continued development from its developers* has an average that places it as the sixth most important selection criteria, while its average for application is in the higher half of results. Whether or not a language has sufficient capacity for database connectivity also places in the higher half of results for both importance and application, as a great deal of programming revolves around *programming* for *databases* (see 3.4.5).

The *international standardisation* of a programming language, together with *international trends with regard to programming languages used in high school education in other parts of the world* (see 3.4.2), places in the lower half of results for importance and in the lowest quarter for application. Considering *the popularity and/or demand for specific languages in industry* when selecting a first programming language for high schools received similar results for importance and application. This makes sense in the light of the fact that

- the commercial use and/or popularity of particular languages shouldn't be seen as essential for selection when learning programming (see 3.5.6)
- the purpose of the subject in South Africa is not to provide training for industry (also see result at the end of the next paragraph)
- many of the languages used for object-oriented programming in industry involve significant complexity (see 3.4.1)

Local tertiary establishments expect learners to be able to solve problems in any programming language (see 3.5.7). If the languages offered at these institutions are the same as those used

at school level, students will be slightly advantaged; if it is different, students will at least know programming principles. In line with this, the *compatibility of a programming language with* these establishments' *choice of programming language* is one of the least important and applied items. Just more than half of respondents also strongly agreed that it is more important that a programming language facilitate the learning of programming principles than to learn a language learners will encounter in tertiary or private training.

The language being suitable and having capabilities for working on and with the *Internet* to facilitate electronic communication is the least important selection criteria and was also applied second least. This could be considered to be a confirmation of the opinion expressed (see 3.4.4) that the value of being able to program for and to the Internet is over-estimated, as most programming for businesses is still done with regard to traditional applications such as salary systems.

6.3.6 Affordability, training and resources

With regard to selection criteria that are specifically suited to the South African context, the *affordability of the chosen programming language* is considered by respondents to be the second most important item of all considered. The financial situation of schools makes it necessary to consider whether schools or education departments will carry the cost for the upgrade of hardware and software (see 3.5.2). The costs associated with the acquisition of the software required, including the programming language, IDE and database, should be within reasonable reach. In this regard Open Source software could be considered. However, in spite of all this, considering the affordability of a chosen programming language was applied so little that it places at the lowest quarter of results.

The availability of *affordable*, sufficient *in-service training of* current *CS teachers*, the ability of teachers new to the subject to learn the language used in school during their *pre-service training* at local tertiary institutions (see *3.5.3.2*), and having *learning and teaching support materials and other resources, particularly textbooks, for the language available to teachers*, (see 3.5.4) received averages that place them as the third, eleventh, fourth and seventh most important items. However, the averages for these items for application are well within the lowest quarter of results, with the first and third of the items mentioned in this paragraph specifically obtaining the third and fourth lowest averages in this regard.

Of particular concern is how little criteria mentioned in the last paragraph were actually applied during language selection - in any future situations the importance of these criteria, specifically with regard to the ability of teachers to eventually successfully implement a language, warrants much more careful consideration.

Affordable, sufficient in-service teacher training is considered to be essential, both in literature (see *3.5.2.4* and *3.5.3.1*) and by respondents to the questionnaire (5.4.3 and 5.4.4). Teachers

from the latter group mention a province where no teacher training was offered in the language being implemented, while four refer to the retraining of educators, three of whom bring up the cost involved in retraining – they feel that these costs should be fully borne by education departments, and not by teachers, as had already occurred in some cases. The feeling was also expressed that education departments should be responsible for arranging training opportunities.

Although having *resources appropriate to an OBE approach to the subject* available to teachers has an average placing it in the middle of results, it was in fact applied the least. This last result is very discouraging, since employing such resource materials is meant to smooth the progress of learners towards the achievement of outcomes (see *3.5.4.1*).

In line with OBE principles, the chosen programming language should also provide adequate avenues for assessment (see 3.5.8). In terms of external assessment, most respondents agreed to some extent that *options for a language-independent examination need to be considered* if *provinces*/schools *implement different languages*. Using a common pseudo-code could still be used to complete problem solving questions. However, with regard to the notion that *the programming section currently contained in written papers should no longer be assessed externally, but only projects used for this purpose instead*, respondents were highly divided: although just more than a third agreed with reservations with this idea, more than half disagreed to some extent. During further investigations an effort can be made to determine which factors contribute to the large difference of opinion for this last item, upon which this issue might be laid to rest.

6.3.7 Programming for various purposes

In the South African context, the programming language curriculum being taught is of a general nature (see 3.5.5). The language to be used should therefore be a general purpose programming language that supports many academic and commercial tools, and not one specifically developed for a certain setting, such as scientific or commercial environments.

Using a general purpose programming language that supports *programming for various purposes* received averages for both importance and application placing it inside the lower half of results, as does the average for the importance of being able to use the language with *academic tools*. The average for the application of the latter, as well as the application average for being able to use the language with *commercial tools*, places these in the lowest quarter of results, while being able to use the language with *commercial tools* is the fourth least important item amongst selection criteria.

6.4 RESULTS FOR IMPLEMENTATION GUIDELINES, IMPLICATIONS AND RECOMMENDATIONS

6.4.1 Appreciating teachers' roles in implementation

Starting planning by doing an introductory investigation, followed by *a situation analysis* in conjunction with *various other methods for gathering information about the implementation process* (see 4.2.1) is considered to be the least important item of all amongst the implementation guidelines. However, it was used seventh most when implementation took place. In light of the low rating for importance the amount of use is heartening, as clear specification of required curriculum resources, including provision of sufficient materials, equipment, facilities, personnel, time and cost, can make a difference between success and failure when it comes to implementing a curriculum.

A similar phenomenon is observed for the item regarding allowing micro-implementation to take place at classroom level by granting *teachers the necessary freedom* that will facilitate this local adoption (see 4.3.2): although its average for importance places it firmly inside the lowest third of results, its average for usage situates it in the fifth place from the top. The low importance assigned to this item is in contrast to literature indicating that reforms will not succeed if education departments demand that curriculum should be implemented exactly as stipulated (see 4.2.2).

The development of learning and teaching materials and resources, including, but not limited to, particularly textbooks suited to high school level, to accompany and support a new curriculum (see 4.3.3) include some of the most important contributions to implementation, placed fourth and fifth in terms of importance. The development of textbooks received a lot of attention during the implementation process (third most used item), which could be due to the fact that textbooks align curriculum as the intended policy and the way in which the curriculum is implemented in classrooms by being more detailed sources of what subject content should be covered. In this way, textbooks become many teachers' primary guide for classroom practice. Concerns have however been raised regarding the availability of books for high school level in languages other than English (in South Africa there is a significant number of learners who speak Afrikaans, and also double medium schools, where classes are presented in both Afrikaans and English).

The development of other resources, however, fell to the wayside, with an average placing it in the lowest quarter of results for use. This is distressing, as one of the major factors in the successful implementation of curricular reform concerns whether or not useful, high-quality instructional materials accompany a particular curriculum. If a curriculum comes with insufficient resources, it is unlikely to succeed, as teachers will need to spend large amounts of time acquiring or creating materials, the development and production of which is labour-intensive and time-consuming (see *4.5.2.4*).

Making *guiding documentation about implementation of a programming language available to teachers* received averages for both importance and usage in the top third in each section, while showing appreciation for the role of teachers in implementation at all levels in the curriculum process place around the middle of results for both. The inclusion of didactic guidelines about the format in which curriculum content is to be transmitted in classrooms in educational curricula is essential to ensure the subsequent implementation of such curricula (see 4.3.2). If teachers don't receive guidelines for the implementation of the programming language used in their province, thy will have no idea what to teach their learners, or whether they are concentrating on the correct content. Curriculum documents and materials should be distributed to schools in the year prior to implementation, so that teachers can plan for the year ahead. Sending regulations mere weeks before implementation dates unnecessarily adds to the already heavy workload of teachers.

6.4.2 Issues surrounding pilot testing

Stating *outcomes that need to be achieved clearly and in a concise way* as soon as these have been identified (see 4.2.3), was used the most during implementation, and it was also awarded a fair amount of importance (sixth highest average). These results are encouraging, as outcomes that are ambiguous and/or not clearly formulated could lead to a discrepancy between intended and executed policy.

Preliminary testing of a new curriculum in the form of pilot testing, using volunteer, *experienced teachers* received very low averages for both importance and usage, all of which fall inside the lowest third of results. Respondents are not convinced of the merits of pilot testing, despite the fact that a trial can provide solid answers to teachers who are concerned about difficulties they might encounter (see 4.4.3) in full implementation. One respondent, however, is of the opinion that if pilot testing was done, proper planning would be possible (see 5.4.4).

With regard to the use of volunteer teachers in pilot testing, trail organisers should note that effective change is more likely in a situation where teachers' participation in a trial implementation is voluntary. If it is a requirement of the experimental environment that all teachers' influence should be constant with regard to experience, successful, more experienced teachers have the confidence to want to make a contribution, will put forward ideas due to their experience and come up with recommendations that could be used to implement programmes. If teachers with varied experience are involved, younger teachers, with their recent training and enthusiasm, might also have a contribution to make.

Convincing *teachers that their implementation efforts will be supported* has the second lowest average for use, but is considered more important with an average for importance that places it at the middle of results. The latter rating agrees better with the opinion (see 4.5.1) that one of the most important strategies for curriculum dissemination involves clearly demonstrating to

participants that their efforts will be supported, by identifying support services and resources, and making these accessible to teachers.

Carrying *out programme assessment in order to revise and improve a new curriculum to a standard suitable for implementation* has averages for both importance and use that places it at the middle of results. This is in line with a recommendation made (see 4.4.5) that the greatest strengths, weaknesses and deficiencies of a new instructional programme should be identified, as these would determine the types of training and professional development programmes needed for teachers to be able to deliver the programme.

6.4.3 Consider cost implications at macro-implementation level

Considering *cost implications at macro-implementation (national/provincial) level* is a contentious issue, rated as the fourth least important implementation guideline, but was in fact used second most. Based on these results, it could be argued that the consideration of cost implications should be used far less than was the case as reported in this study. However, all educational programmes must take costs into account at some level (see 4.5), and especially higher education is always subject to resource limitations of various kinds. On the other hand, any reform campaign cannot succeed without the infusion of substantial amounts of money - the challenge lies in creating a high quality programme with limited resources.

6.4.4 Introducing a new curriculum

Addressing *teachers' concerns regarding content and* conceptual *knowledge in a new curriculum by providing practical advice about teaching* (see 4.3.2) and the extent to which *educational leaders provide progressive leadership during curriculum improvement by mobilising administration and resources in support of teachers,* received averages for importance placing these items in the top third of results. However, for usage both these items have low averages.

Specifically, the low average for addressing teachers' knowledge concerns should be particularly disturbing to those in charge of implementation interventions, in light of the warning issued (see 4.5.3) that implementation could fail if teachers lack the content and/or conceptual knowledge to carry out policy makers' proposals - reform is difficult, and teachers are reluctant to teach a new programming language, if they believe that they had little preparation in content or pedagogy.

Likewise, the low level of providing leadership flies in the face of the fact that one of the strategies that has proven to lead to effective implementation underscores the significance of dynamic curriculum leadership in providing sufficient, professional support programmes to ensure that teachers have the resources they need (see *4.5.2.4*).

The reverse is true for the items relating to making *teachers aware of the differences between old and new practices*, creating *opportunities for teachers from different schools to discuss implementation* and identifying *factors and variables that* could either further or impede *implementation*: these items received averages for use that place them in the top third of results, but averages for importance around the middle of results.

Differences between old and new practices (see *4.5.2.1*) should be pointed out to teachers, as literature reveals that teachers who don't understand the extent of change, feel pressured to make changes, or do not have a voice in curriculum decisions, simply remould a new curriculum to a format that allows them to fall back on previously existing teaching practices.

It is especially important that regular meetings of everybody involved in reforms are held, as teachers need to be provided with opportunities where they can report back, share and discuss their fears and reservations about curriculum innovations, as well as problems that they might be experiencing, and give vent to their grievances (see 4.2.2). Connections made at such meetings could also supply teachers with a local professional network for exchanging ideas, discussing successful implementations and sharing suggestions for better implementation.

The following items show averages for both importance and use close to the middle of results:

- specifying and classifying *learning content that is of consequence to learners to initiate the development of a curriculum* (see 4.3.1)
- making *the relative advantage of reform clear to teachers*
- managing the dissemination of *information regarding a new curriculum in a manner that will overcome resistance to change* and make informed decision making in schools possible (see 4.5.1)
- handling *sources of resistance to change* and *obstacles to implementation* that have been *identified in such a manner that* resistance to change will be overcome and *implementation can proceed smoothly*

Obstacles to implementation can be practical and/or psychological, while teachers' anxieties regard logistical aspects (see *4.5.2.5*). It is essential to identify these obstacles, as they could prevent a new curriculum from being realised in practice.

Gauging *teachers' expectations for implementation* also has a moderate rating for use, but an average for importance placing it very close to the lowest quarter of results.

6.4.5 Guidance, support and assessment

Providing *teachers with the time and resources to be retrained properly* and distributing *curriculum documents and materials to schools* are examples of the *support and guidance* that they could benefit from before and *during* user *implementation* (see 4.5.3). *Support for teachers* should be continued *as implementation becomes established and consolidated into an accepted*

part of curricular practice. Teachers can also be involved *in the assessment of a new curriculum*, by assessing it with regard to *the extent to which its implementation leads to the achievement of learning outcomes* (see 4.6).

All items in the questionnaire with relevance to the statements above, received averages for importance placing them in the top half of results. Specifically, items referred to in the first sentence of that paragraph have the top three averages for importance. However, with the exception of the item regarding the distribution of documents (which places fourth for use), all other items have much lower averages for use, placing them at and below the middle of results.

Educators should be given ample time to master the use of a programming language prior to implementation (see 5.4.6), as a lack of adequate time for planning and instruction is the single most frequently cited barrier to implementation, and a premier reason for not implementing curriculum changes (see *4.5.2.4*).

The actual provision of the necessary resource support during implementation can be the most critical factor in curriculum innovation, and one teacher specifically expressed the hope that teachers will get better support from the Department of Education than in the past. Teachers need sufficient practice-oriented in-service training that focuses on concrete problems that they experience during the implementation of reform (see *4.5.2.6*).

The successful distribution of quality curriculum documents and teaching materials, which have been tested, to all teachers, is an essential aspect of effective curriculum dissemination (see *4.5.2.7*), while the local capacity to supply teachers with sustained supporting factors is central to implementing reforms (see 4.5.4).

Although determining *to what extent a new curriculum was indeed implemented,* received a relatively low average for importance compared to the other items in this factor, its average for use places it just outside the top third of results. Based on these results, this item could potentially be used much less.

In the analysis of data, mainly descriptive statistics were used, the results of which have been summarised in this and the previous section of this chapter. However, inferential statistics were also used to establish the significance of differences, and a précis of these results follows.

6.5 FACTOR ANALYSES AND SIGNIFICANCE OF DIFFERENCES

Factor analyses were carried out on statements in the questionnaire with regard to the application of selection criteria and use of implementation guidelines. For selection criteria, seven factors emerged (see 5.5.1), while for implementation guidelines four factors were identified (see 5.5.2). Based on the final communality estimates, these factors can be considered to be a successful extraction. One item regarding cost implications showed low values for both its final communality estimate and factor loading – it was decided to treat this

item as a separate additional factor. The reliability of these factors as measuring instruments was also confirmed by calculating Cronbach Alpha scores (see 5.5.3).

When the differences between the ratings that respondents gave to items with regard to importance and application/usage respectively for each factor were calculated, effect sizes indicated that all factors displayed differences between importance and application/usage that are large enough to have an effect in practice (see 5.5.4). Grouping of respondents as policy makers, trainers or teachers reveal less significant differences between importance and application for policy makers and trainers regarding new tendencies in programming and the relevance of criteria. Policy makers also show less significant differences with regard to issues influencing programming used in practice and little difference between the rated importance and actual implementation of cost implications.

The fact that there are generally practically significant differences between the rated importance and application/usage of factors point to a situation where issues that are important to the majority of respondents (in this case teachers) are not applied and/or used in the selection and implementation of the programming language. This impression is underlined when these differences are smaller for policy makers, which could indicate that policy makers are under the impression that the issues that they deem to be important dictate the extent to which these are applied and/or used.

Differences between the ways in which policy makers, trainers or teachers respectively rated the importance, application and usage of factors for which there is a statistically significant difference between at least one pair of means (see 5.5.5), was also investigated. With the exception of issues surrounding pilot testing, four factors regarding the use of implementation guidelines exhibit differences with practical significance between the way in which policymakers and teachers respectively rate the use of these factors.

These results indicate that teachers at school level experience that implementation guidelines are used to a far lesser extent than policy makers would assume in their ratings. In order to address this discrepancy, implementation will need to be recalibrated so that the use of guidelines is not only stepped up, but deficiencies that have developed could be dealt with in ways that will facilitate the eventual acceptance of a new programming language.

6.6 FINAL CONCLUSIONS

The successful implementation of an innovative curricular programme is dependent on the full and active participation in substantive ways of the classroom teachers who will eventually be expected to implement proposed curriculum reforms. Teachers' authority to speak on issues relating to curriculum development is rooted in their knowledge of their learners, schools and resources available to them. Policy makers need to take this extensive knowledge base that teachers have into account when new curricula are designed.

Based on this knowledge, teachers can be involved in the process of producing new curricula in the form of senior teachers who act as co-authors in preparing a curriculum and support materials, and as participants in school-based trials of curricula and curriculum materials. The latter group contribute to the development process by exploring the usability of the trial curriculum at their schools where they teach, monitor and assess.

Local teachers who had participated in the trial implementation and/or field-testing of a new curriculum are more credible advocates than academic or pedagogical experts, whose support of a new curriculum does not carry much conviction with teachers. 'Trial' teachers should, therefore, rather be invited to discuss the finer details of implementing the innovation and respond to the doubts of teachers who are considering its adoption.

Those who design implementations need to consider teachers' existing ideas and conceptions about the new programming language and teaching it, and structure implementation in such a way that teachers will be enabled to isolate time and resources at the level presumed by implementation documents.

In order for curricula to change school practice, all role players in the curriculum renovation arena must consider the fact that teachers' practices will not change if they are scarcely prepared and only marginal support is made available from the educative administration to solve problems that have arisen due to reform efforts.

Only "reforms that take into account teachers' beliefs, perspectives and knowledge, that actively involve teachers in planning and decision making, and that treat teachers as professionals are ... likely to succeed." (Fennema *et al.*, 1991:28).

BIBLIOGRAPHY

ACM SPECIAL INTEREST GROUP IN COMPUTER SCIENCE EDUCATION (SIGCSE). 2001a. Introductory courses. Chapter 7. (*In* Computing Curricula 2001: Computer science volume.) [Web:] http://www.acm.org/sigcse/cc2001/index.html [Date of access: 2 April 2003].

ACM SPECIAL INTEREST GROUP IN COMPUTER SCIENCE EDUCATION (SIGCSE). 2001b. Institutional challenges. Chapter 13 (*In* Computing Curricula 2001: Computer science volume.) [Web:] http://www.acm.org/sigcse/cc2001/index.html [Date of access: 20 March 2003].

ADAMS, N.B. 2002. Educational computing concerns of postsecondary faculty. *Journal of research on technology in education*, 34(3):285-303, Spring.

ADLARD, J. (James@mhs.kzn.school.za) 2000a. Re: Language again. Comp-studies@wcape.school.za. May 26.

ADLARD, J. (James@mhs.kzn.school.za) 2000b. Re: The language. Comp-studies@wcape.school.za. May 26.

AHO, A.V. & ULLMAN, J.D. 1992. Foundations of computer science. New York : Computer Science Press.

AIKEN, R.M. & SNELBECKER, G.E. 1991. Hindsight: reflections on retraining secondary school teachers to teach computer science. *Journal of research on computing in education*, 23(3):444-451, Spring. [In EBSCOHost : Academic Search Premier, Full Display: http: www-sa.ebsco.com] [Date of access: 21 March 2003].

ALPER, L., FENDEL, D. & FRASER, S. 1997. Designing a high school mathematics curriculum for all students. *American journal of education*, 106(1):148-178, November.

ANON. 1999. Not every Java programmer is a true Java programmer. *Computing SA*, 19(26):38, July 12.

BADENHORST, J. (jjcb@majuba.ofs.gov.za) 2000. New programming language. Comp-studies@wcape.school.za. July 31.

BARROW, J., GELDERBLOM, J.H. & MILLER, M.G. 2002. Introducing Delphi programming: theory through practice. 3rd ed. Cape Town : Oxford University Press.

BASSON, E. (philerna@netactive.co.za) 2000. Re: Rekenaarstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. fransiesmit@mweb.co.za. August 17.

BENCZE, L. & HODSON, D. 1999. Changing practice by changing practice: toward more authentic science and science curriculum development. *Journal of research in science teaching*, 36(5):521-539, May.

BERNARDO, M.A. & MORRIS, J.D. 1994. Transfer effects of a high school computer programming course on mathematical modeling, procedural comprehension, and verbal problem solution. *Journal of research on computing in education*, 26(4):523-536, Summer. [In EBSCOHost: Academic Search Elite, Full Display: http: www-sa.ebsco.com] [Date of access: 21 March 2003].

BEZUIDENHOUT, A. (abez@netactive.co.za) 2000a. Delphi book. Comp-studies@wcape.school.za. May 26.

BEZUIDENHOUT, A. (abez@netactive.co.za) 2000b. Programmeringstaal. Comp-studies@wcape.school.za. August 3.

BEZUIDENHOUT, M.L. (Bezuid@oudhs.wcape.school.za) 1999a. Where to after Pascal. Comp-studies@wcape.school.za. April 12.

BEZUIDENHOUT, M.L. (Bezuid@oudhs.wcape.school.za) 1999b. What after Pascal. Comp-studies@wcape.school.za. April 16.

BEZUIDENHOUT, M.L. (Bezuid@oudhs.wcape.school.za) 2000a. Nuwe sillabus. Comp-studies@wcape.school.za. April 6.

BEZUIDENHOUT, M.L. (Bezuid@oudhs.wcape.school.za) 2000b. Re: Java. Comp-studies@wcape.school.za. July 17.

BEZUIDENHOUT, M.L. (Bezuid@oudhs.wcape.school.za) 2000c. Re: Irishman & programming language. Comp-studies@wcape.school.za. July 26.

BEZUIDENHOUT, M.L. (Bezuid@oudhs.wcape.school.za) 2000d. How do we teach Java? Comp-studies@wcape.school.za. August 7.

BIELACZYC, K., PIROLLI, P.L. & BROWN, A.L. 1995. Training in self-explanation and self-regulation strategies: investigating the effects of knowledge acquisition activities on problem solving. *Cognition & instruction*, 13(2):221-252, June 1.

BISHOP, J.M. (jbishop@cs.up.ac.za) 2000. Interface 2000. [E-mail to:] Southern African Computer Lecturers' Association. (SACLA@wwg3.uovs.ac.za) February 14.

BLIGNAUT, P. 2000. Debate on programming language for computer studies HG. (Presentation made at the SAFCERT meeting of examiners and moderators held on August 18.) Boksburg. [Attachment to e-mail from:] Blignaut, P. (pieterb@wwg3.uovs.ac.za) [E-mail to:] Goosen, L. (leila@netactive.co.za) 21 August.

BOOTH, B. (booth@iafrica.com) 2000. Re: New language. Comp-studies@wcape.school.za. August 10.

BORLAND INPRISE. 2000. Adopting Linux without abandoning Windows. [Web:] http://www.borland.com/ [Date of access: 22 August 2000].

BORLAND INTERNATIONAL. 1993. Turbo Pascal user's guide. Scotts Valley : Inprise Corporation.

BOTHA, J. (joland@hotten.wcape.school.za) 2000. Re: Programming language. Comp-studies@wcape.school.za. July 26.

BRAND, D. 2003. Microsoft steun swart bemagtiging. *Rapport, IT-sake*:10, September 28.

BREAKEY, L.N. (lbreakey@mweb.co.za) 2000a. CS HG - Delphi the route to go? Comp-studies@wcape.school.za. May 16.

BREAKEY, L.N. (lbreakey@mweb.co.za) 2000b. Changing from Pascal. Comp-studies@wcape.school.za. May 18.

BREAKEY, L.N. (lbreakey@mweb.co.za) 2000c. Programming language. Comp-studies@wcape.school.za. August 6.

BRITTEN, R. 2000. Watter programmeringstaal in dié Babelse verwarring? *Bylae tot Die Burger, Beeld en die Volksblad*:3, August 15.

BUCKLEY, W.F. 2003. Food for thought. *Getaway*, 15(7):43, October.

CANTO, M. 2000. Comparing OOP Languages: Java, C++, Object Pascal. [Web:] http://www.marcocantu.com/papers/ooplang.htm [Date of access: 23 November 2000].

CARL, A.E. 1995. Teacher empowerment through curriculum development: theory into practice. Kenwyn : Juta.

CHAKRAVARTY, M.M.T. & LOCK, H.C.R. 1997. Towards the uniform implementation of declarative languages. *Computer languages*, 23(2-4):121-160.

CHANG, C., ENGEL, G., ROBERTS, E. & SHACKELFORD, R. 2000. Letter to the Liberal Arts Computer Science Consortium, August 18. [Web:] http://www.math.grin.edu/~walker/coll-learning/index.html [Date of access: 21 March 2003].

CHILES, M. (mikec@iafrica.com) 1999a. Syllabus revision. Comp-studies@wcape.school.za. April 11.

CHILES, M. (mikec@iafrica.com) 1999b. Where to after Pascal? Comp-studies@wcape.school.za. April 11.

CHILES, M. (Mchiles@pawc.wcape.gov.za) 2000a. Turbo Pascal. [E-mail to:] Rodokanakis, M. (mrodokanakis@inprise.co.za) January 11.

CHILES, M. (mikec@iafrica.com) 2000b. Re: Launch of comp-studies list. Comp-studies@wcape.school.za. March 21.

CHILES, M. (mikec@iafrica.com) 2000c. Turbo what? Comp-studies@wcape.school.za. March 23.

CHILES, M. (mikec@iafrica.com) 2000d. New syllabus. Comp-studies@wcape.school.za. March 26.

CHILES, M. (mikec@iafrica.com) 2000e. Interface 2000 conference. Comp-studies@wcape.school.za. May 21.

CHILES, M. (mike@pawc.wcape.gov.za) 2000f. Change in programming language in the HG. Comp-studies@wcape.school.za. July 14.

CHILES, M. (mike@pawc.wcape.gov.za) 2000g. Change in programming language in the HG. Comp-studies@wcape.school.za. July 14.

CHILES, M. (mike@pawc.wcape.gov.za) 2000h. Change in programming language. Comp-studies@wcape.school.za. July 17.

CHILES, M. (mike@pawc.wcape.gov.za) 2000i. Programming language. Comp-studies@wcape.school.za. July 24.

CHILES, M. (mike@pawc.wcape.gov.za) 2000j. Re: The language debate: comments & Q's. Comp-studies@wcape.school.za. July 28.

CHILES, M. (mike@pawc.wcape.gov.za) 2000k. A first programming language at school level. [E-mail to:] Southern African Computer Lecturers' Association. SACLA@wwg3.uovs.ac.za. August 3.

CHILES, M. (mike@pawc.wcape.gov.za) 2000l. Re: Rekenaartaal 2001. Comp-studies@wcape.school.za. August 7.

CHILES, M. (mike@pawc.wcape.gov.za) 2000m. New programming language. Comp-studies@wcape.school.za. August 21.

CHILES, M. (mike@pawc.wcape.gov.za) 2000n. Re: Hoërgraad. Comp-studies@wcape.school.za. August 21.

CHILES, M. (mike@pawc.wcape.gov.za) 2000o. Java meeting. Comp-studies@wcape.school.za. November 28.

CHILES, M. 2001. To Java or not to Java? ... That is the question! (Presentation made at the Africa Connects conference at the University of Cape Town on July 11.) Cape Town.

CHILES, M. 2002a. Principles on which ICT syllabus will be based. [Attachment to e-mail from:] Chiles, M. (mike@pawc.wcape.gov.za) [E-mail to:] Comp-studies@wcape.school.za. 13 May.

CHILES, M. (mike@pawc.wcape.gov.za) 2002b. Computer science subject statement. Comp-studies@wcape.school.za. 12 September.

CHILES, M. 2003. Telephonic discussion with author on January 20. Pretoria.

CHOI, W.S. & REPMAN, J. 1993. Effects of Pascal and FORTRAN programming on the problem-solving abilities of college students. *Journal of research on computing in education*, 25(3):290-302, Spring. [In EBSCOHost: Academic Search Elite, Full Display: http: www-sa.ebsco.com] [Date of access: 21 March 2003].

CHOW-HOY, T.K. 2001. An inquiry into school context and the teaching of the virtues. *Journal of curriculum studies*, 33(6):655-682, November.

COGGINS, J.M. 1996. Subject-oriented programming. [Web:] http://iraf.noao.edu/iraf/web/ADASS/adass_proc/adass_95/cogginsj/cogginsj.html [Date of access: 27 January 2003].

COHEN, J. 1988. Statistical power analysis for the behavioral sciences. Rev. ed. Orlando, Fla. : Academic Press.

COLLIGAN, J.K. 1997. Introduction to programming JAVA Applets (software review). *Mathematics teacher*, 90(6):496-497, September.

COOK, D. (dc@cs.uct.ac.za) 2000a. Re: The language. Comp-studies@wcape.school.za. May 22.

COOK, D. (dc@cs.uct.ac.za) 2000b. Re: Programming language. Comp-studies@wcape.school.za. August 7.

COOK, D. (dc@cs.uct.ac.za) 2000c. Re: How do we teach Java? Comp-studies@wcape.school.za. August 7.

COOK, D. 2000d. Java as a programming language at school level. (Powerpoint presentation made at the SAFCERT meeting of examiners and moderators held on August 18.) [Attachment to e-mail from:] Chiles, M. (mike@pawc.wcape.gov.za) [E-mail to:] Comp-studies@wcape.school.za. August 21.

COOK, D. (dc@cs.uct.ac.za) 2000e. Re: Hoërgraad. Comp-studies@wcape.school.za. August 21.

COOK, D. 2003. Java as a programming language at schools. (Powerpoint presentation made at the WCED Java course run at University of Cape Town from 13 - 16 January.) [Web:] www.cs.up.ac.za [Date of access: 25 January 2003].

CROWDER, D. (dcrowder@inprise.co.za) 2000. Re: Turbo Pascal. [E-mail to:] Chiles, M. (Mchiles@pawc.wcape.gov.za) January 11.

CUSENS, L. (kankamp@iafrica.com) 2000a. Re: Rekenaatstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 17.

CUSENS, L. (kankamp@iafrica.com) 2000b. Re: Rekenaatstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 17.

DAGIENE, V. 2003. The model of teaching Informatics in Lithuanian comprehensive schools. *Journal of research on technology in education*, 35(2):176-185, Winter 2002-2003.

DEEK, F.P. & KIMMEL, H. 1999. Status of computer science education in secondary schools: one state's perspective. *Computer science education*, 9(2):89-113, August.

DEEK, F.P. & MCHUGH, J.A. 1998. A survey and critical analysis of tools for learning programming. *Computer science education*, 8(2):130-178, August.

DE VILLIERS, P. (pja@cs.sun.ac.za) 2000. Re: Oberon. [E-mail to:] Blignaut, P. (pieterb@wwg3.uovs.ac.za) April 4.

DELPORT, A. (alidad@csc.co.za) 2000. Re: [Fwd: A first programming language at school level]. [E-mail to:] Chiles, M. (mike@pawc.wcape.gov.za) August 22.

DoE see SOUTH AFRICA. DEPARTMENT OF EDUCATION

DRAKOS, N. 1995a. Graphical programming concepts. [Web:] http://cbl.leeds.ac.uk/nikos/tex2html/examples/concepts/node31.html [Date of access: 27 January 2003].

DRAKOS, N. 1995b. Why visualise? [Web:] http://cbl.leeds.ac.uk/nikos/tex2html/examples/concepts/node31.html [Date of access: 27 January 2003].

DRAKOS, N. 1995c. Visual programming paradigms. [Web:] http://cbl.leeds.ac.uk/nikos/tex2html/examples/concepts/node35.html [Date of access: 27 January 2003].

DRAKOS, N. 1995d. Visual Basic. [Web:] http://cbl.leeds.ac.uk/nikos/tex2html/examples/concepts/node44.html [Date of access: 27 January 2003].

DRAKOS, N. 1995e. Visual C++. [Web:] http://cbl.leeds.ac.uk/nikos/tex2html/examples/concepts/node45.html [Date of access: 27 January 2003].

DU PLESSIS, A. (mmicro@mailbox.co.za) 2000. Re: Rekenaarstudie-vakvergadering. [E-mail to:] Smit, F. (smitap@mweb.co.za) August 16.

DYCK, J.L. & MAYER, R.E. 1989. Teaching transfer of computer program comprehension skill. *Journal of educational psychology*, 81(1):16-24, March.

ECKEL, B. 2000. Thinking in Java. 2nd ed. [Web:] http://www.mindview.net/TIJ2/Introduction.html [Date of access: 24 April 2000].

FENNEMA, E., CARPENTER, T.P. & PETERSON, P. 1991. Teacher's decision making and cognitively guided instruction: a new paradigm for curriculum development. *Pythagoras*, 27:27-35.

FINCH, L. 1998. So much OO, so little reuse. [Web:] http://www.ddj.com/documents/s=909/ddj9875g/9875g.htm [Date of access: 27 January 2003].

FISCHER, G., REDMILES, D., WILLIAMS, L., PUHR, G.I., AOKI, A. & NAKAKOJI, K. 1995. Beyond object-oriented technology: where current approaches fall short. *Human-computer interaction*, 10(1):79-119.

GAL-EZER, J. & HAREL, D. 1999. Curriculum and course syllabi for a high-school CS program. *Computer science education*, 9(2):114-147, August.

GELLING, J. (gelling@stjohns.wits.ac.za) 2000. Teaching Delphi. Comp-studies@wcape.school.za. May 26.

GIBSON, K. (gibsonk@mail.arhs.ecape.school.za) 2000a. New programming language. Comp-studies@wcape.school.za. July 27.

GIBSON, K. (gibsonk@mail.arhs.ecape.school.za) 2000b. Some thoughts on the new language from the Eastern Cape. Comp-studies@wcape.school.za. August 1.

GIBSON, K. (gibsonk@mail.arhs.ecape.school.za) 2000c. New language thoughts. Comp-studies@wcape.school.za. August 10.

GOOSEN, L. 1999. Doeltreffende onderrig van rekenaarstudie in die sekondêre skool, met spesifieke verwysing na uitkomsgebaseerde onderwys. Potchefstroom : Potchefstroom University for Christian Higher Education. (Dissertation - M.Ed.)

GOLDBERG, A. & ROBSON, D. 1983. Smalltalk-80: the language and its implementation. Reading, Mass. : Addison-Wesley.

GRAHAM, I. 1994. Object-oriented methods. 2nd ed. Wokingham : Addison-Wesley.

GRIGAS, G. 1994. Distance teaching of informatics: motivations, means, and alternatives. *Journal of research on computing in education*, 27(1):19-28, Fall. [In EBSCOHost: Academic Search Elite, Full Display: http: www-sa.ebsco.com] [Date of access: 26 March 2003].

GROBLER, R. (dirkg@lantic.co.za) 2000. Re: Rekenaarstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 17.

HAVENGA, M. (havengam@iafrica.com) 2000. Re: Programmeringstaal. [E-mail to:] Bezuidenhout, A. (abez@netactive.co.za) August 4.

HILL, H. (bastion@iafrica.com) 2000. Re: Rekenaarstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 23.

HILL, L. (lhill@ingulube.bortech.ac.za) 2000. New computer language. [E-mail to:] Gibson, K. (gibsonk@mail.arhs.ecape.school.za) August 3.

HUSIC, F.T., LINN, M.C. & SLOANE, K.D. 1989. Adapting instruction to the cognitive demands of learning to program. *Journal of educational psychology*, 81(4):570-583, December.

JANSE VAN RENSBURG, N. (nance@crawfordcollege.co.za) 2000. Research done on Python. Comp-studies@wcape.school.za. August 28

JARDINE, L. (jardine@icon.co.za) 2000. Re: Rekenaarstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 16.

JOHNSON, S.C. 1997. Objecting to objects. [Web:] http://www.usenix.org/publications/library/proceedings/sf94/johnson.html [Date accessed: 27 January 2003].

JOYNER, I. 1996. C++??: a critique of C++ (3rd ed.) and programming and language trends of the 1990s. [Web:] http://www.elj.com/cppcv3/s2/.html [Date of access: 27 January 2003].

JUPITERMEDIA CORPORATION. 2002a. Object oriented. [Web:] http://webopedia.internet.com/TERM/o/object_oriented.html [Date of access: 20 January 2003].

JUPITERMEDIA CORPORATION. 2002b. Object-oriented programming. [Web:] http://webopedia.internet.com/TERM/o/object_oriented_programming_OOP.html [Date of access: 20 January 2003].

JUPITERMEDIA CORPORATION. 2003a. Event. [Web:] http://webopedia.internet.com/TERM/E/event.html [Date of access: 23 September 2003].

JUPITERMEDIA CORPORATION. 2003b. Event handler. [Web:] http://webopedia.internet.com/TERM/E/event_handler.html [Date of access: 23 September 2003].

JUPITERMEDIA CORPORATION. 2003c. Interface. [Web:] http://webopedia.internet.com/TERM/I/interface.html [Date of access: 23 September 2003].

JUPITERMEDIA CORPORATION. 2003d. Method. [Web:] http://webopedia.internet.com/TERM/M/method.html [Date of access: 23 September 2003].

JUPITERMEDIA CORPORATION. 2003e. Polymorphism. [Web:] http://webopedia.internet.com/TERM/P/polymorphism.html [Date of access: 23 September 2003].

JUPITERMEDIA CORPORATION. 2003f. Program. [Web:] http://webopedia.internet.com/TERM/P/program.html [Date of access: 23 September 2003].

JUPITERMEDIA CORPORATION. 2003g. Semantics. [Web:] http://webopedia.internet.com/TERM/S/semantics.html [Date of access: 23 September 2003].

JUPITERMEDIA CORPORATION. 2003h. Syntax. [Web:] http://webopedia.internet.com/TERM/S/syntax.html [Date of access: 23 September 2003].

JUPITERMEDIA CORPORATION. 2003i. User interface. [Web:] http://webopedia.internet.com/TERM/U/user_interface.html [Date of access: 23 September 2003].

KAASBOLL, J. 2000. Learning and teaching programming. (*In* Interface 2000. Computing at schools, technikons and universities. Papers read at the Interface 2000 conference held in Pretoria on 19 and 20 May. Pretoria. p. 24-31.)

KAMIN, S.N. & REINGOLD, E.M. 1996. Programming with class; A C++ introduction to computer science. New York : McGraw-Hill.

KAY, J., BARG, M., FEKETE, A., GREENING, T., HOLLANDS, O., KINGSTON, J.H. & CRAWFORD, K. 2000. Problem-based learning for foundation computer science courses. *Computer science education*, 10(2):109-128, August.

KENNING, G. (gkenning@mweb.co.za) 2000. The language debate: comments & Q's. Comp-studies@wcape.school.za. July 27.

KHOSHAFIAN, S. 1990. Insight into object-oriented databases. *Information and software technology*, 32(4):274-289, May.

KING, A. (alakin@tvh.co.za) 2000. Repost of bounced message. Comp-studies@wcape.school.za. July 27.

KIRK, D. & MACDONALD, D. 2001. Teacher voice and ownership of curriculum change. *Journal of curriculum studies*, 33(5):551-567, September.

KIRKWOOD, M. 2000. Infusing higher-order thinking and learning to learn into content instruction; a case of secondary computing studies in Scotland. *Journal of curriculum studies*, 32(4):509-535, July.

KNAGGS, R. (Richard Knaggs@parklands.co.za) 2000a. Java resources. Comp-studies@wcape.school.za. July 17.

KNAGGS, R. (Richard Knaggs@parklands.co.za) 2000b. Java. Comp-studies@wcape.school.za. July 26.

KNAGGS, R. (Richard Knaggs@parklands.co.za) 2000c. Learning Java. Comp-studies@wcape.school.za. August 7.

KOK, L. 2003. Tegnologiebedryf gaan weer blom - tot 2009, sê Gates. *Sake-Beeld*:3, September 25, Thursday.

KRAUSE, R. 2003. Telephonic discussion with author on March 4. Pretoria.

KURATA, D. (deborahk@insteptech.com) 2000. Re: Teaching of VB in schools. [E-mail to:] Shear, S. (sshear@commerce.uct.ac.za) August 22.

KUYPER, S. (s.snyman@mweb.co.za) 2000. Re: Rekenaarstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 18.

LAMBERT, K.A. & NANCE, D.W. 1997. Understanding programming and problem solving with C++. Minneapolis: West Pub. Co.

LAMBERT, K.A. & NANCE, D.W. 1998. Fundamentals of C++ programming: understanding programming and problem solving. Cincinnatti, Ohio : South-Western.

LAPPAN, G. 1997. The challenges of implementation: supporting teachers. *American journal of education*, 106(1):207-239, November.

LEDERMAN, N.G. 1999. Teachers' understanding of the nature of science and classroom practice: factors that facilitate or impede the relationship. *Journal of research in science teaching*, 36(8):916-929, October.

LENZI, M. 1995. The Object World Conference. *Computer conference analysis newsletter*, 369:1-11, August 25 [In EBSCOHost: Business Source Premier, Full Display: http: www-sa.ebsco.com] [Date of access: 25 March 2003].

LIBERAL ARTS COMPUTER SCIENCE CONSORTIUM (LACS). (members@lacs.edu) 2000. Letter to the CC2001 Steering Committee, July 24. [Web:] http://www.math.grin.edu/~walker/coll-learning/index.html [Date of access: 21 March 2003].

LöTTER, A. (tygerweb@yebo.co.za) 2000. Programming language. Comp-studies@wcape.school.za. August 17.

MACHANICK, P. 2000. On the value of qualifications: why go to university or technikon? (*In* Interface 2000. Computing at schools, technikons and universities. Papers read at the Interface 2000 conference held in Pretoria on 19 and 20 May 2000. Pretoria. p. 1-9.)

MADISON, S. & GIFFORD, J. 2002. Modular programming: novice misconceptions. *Journal of research on technology in education*, 34(3):217-229, Spring.

MAES, F., VANDENBERGHE, R. & GHESQUIEÁ, P. 1999. The imperative of complementarity between the school level and the classroom level in educational innovation. *Journal of curriculum studies*, 31(6):661- 677, November.

MALAN, K.M. (Malankm@unisa.ac.za) 2003. Re: Java for teachers. [E-mail to:] Goosen, L. (lgoosen@gk.up.ac.za) March 11.

MANOUCHEHRI, A. & GOODMAN, T. 1998. Mathematics curriculum reform and teachers: understanding the connections. *Journal of educational research*, 92(1):27-41, September/October.

MARTIN, J. & ODELL, J.J. 1992. Object-oriented analysis and design. Englewood Cliffs, N.J. : Prentice-Hall.

MAYER, R.E. & DYCK, J.L. 1989. Teaching for transfer of computer program comprehension skill. *Journal of educational psychology*, 81(1):16-24, March.

MAYERS, P. (pmayers@admin.dc.wcape.school.za) 2000a. Re: Irishman & programming language. Comp-studies@wcape.school.za. July 27.

MAYERS, P. (pmayers@admin.dc.wcape.school.za) 2000b. Re: New language thoughts. Comp-studies@wcape.school.za. August 10.

MENDELSOHN, P., GREEN, T.R.G. & BRNA, P. 1990. Programming languages in education: the search for an easy start. (*In* Hoc, J.-M., Green, T.R.G., Samurçay, R. & Gilmore, D.J., *eds.* Psychology of programming. London : Academic Press. p. 175-200.)

MEYER, I. (imeyer@csir.co.za) 2000. Re: Rekenaatstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 17.

MIAH, S. 1997. Critique of the object oriented paradigm: beyond object-orientation. [Web:] http://members.aol.com/shaz7862/critique.htm [Date of access: 27 January 2003]

MILLER, P. (pam@miller.wcape.school.za) 2000. Re: Syllabus. Comp-studies@wcape.school.za. March 26.

MOLLENTZE, A. (amollentze@kingsley.co.za) 1999. Where to after Pascal? Comp-studies@wcape.school.za. April 13.

MOSTERT, J.M. 1986. Riglyne vir kurrikulumontwikkeling. Pretoria : Human Sciences Research Council.

NEVES, I.P. & MORAIS, A.M. 2001. Teacher's 'space of change' in educational reforms: a model for analysis applied to a recent reform in Portugal. *Journal of curriculum studies*, 33(4):451-476, July.

NIEUWOUDT, H.D. 1998. Beskouings oor onderrig: implikasies vir die didaktiese skoling van Wiskundeonderwysers. Potchefstroom : Potchefstroom University for Christian Higher Education. (Thesis - Ph.D.)

NOOME, C. 2000. Programming language comparison guide. [Attachment to email from:] Smit, F. (fransiesmit@mweb.co.za) [E-mail to:] Goosen, L. (leila@netactive.co.za)

NORRIS, C. & JACKSON, L. 1992. The effect of computer science instruction on critical thinking skills and mental alertness. *Journal of research on computing in education*, 24(3):329-336, Spring. [In EBSCOHost: Academic Search Elite, Full Display: http: www-sa.ebsco.com] [Date of access: 21 March 2003].

OCHOLLA, D.N. 2001. Curriculum response to a changing national and international information environment: theoretical and methodological paradigms on review and revision. *Education for information*, 19(2):143-167.

PALUMBO, D.B. 1990. Programming language/problem-solving research: a review of relevant issues. *Review of educational research*, 60(1):65-89, Spring.

PALUMBO, D.B. & REED, W.M. 1991. The effect of BASIC programming language instruction on high school students' problem solving ability and computer anxiety. *Journal of research on computing in education*, 23(3):343-345, Spring. [In EBSCOHost: Academic Search Elite, Full Display: http: www-sa.ebsco.com] [Date of access: 21 March 2003].

PARKE, H.M. & COBLE, C.R. 1997. Teachers designing curriculum as professional development: a model for transformational science teaching. *Journal of research in science teaching*, 34(8):773-789, October.

PETJE, M. 2003. Schools to go online on time. *Gauteng news*:3, September.

PIETERSE, R. (PIETERSR@boksburgcouncil.co.za) 2000. Re: Programming language. Comp-studies@wcape.school.za. July 26.

PIROLLI, P. & RECKER, M.M. 1994. Learning strategies and transfer in the domain of programming. *Cognition & instruction*, 12(3):235-275, September 1.

POLYA, G. 1981. Mathematical discovery. New York : Wiley.

PRATT, D. 1994. Curriculum planning: a handbook for professionals. Fort Worth : Harcourt Brace.

PRATT, D. 1999. Lessons for implementation from the world's most successful programme: the global eradication of smallpox. *Journal of curriculum studies*, 31(2):177-194, February.

PRETORIUS, E. (gapretorius@xsinet.co.za) 2000a. Re: Programming language. Comp-studies@wcape.school.za. August 6.

PRETORIUS, E. (gapretorius@xsinet.co.za) 2000b. Re: Programmeringstaal. Comp-studies@wcape.school.za. August 13.

PUNT, W. (wpunt@mweb.co.za) 2000. Java. Comp-studies@wcape.school.za. September 15.

RAPPAPORT, T.S. & SCHARNHORST, A.A. 1996. Educating engineers for a wireless world. *Electronic engineering times*, 896:C8, August 4. [In EBSCOHost: Academic Search Premier, Full Display: http://www-sa.ebsco.com] [Date of access: 21 March 2003].

RECKER, M.M. & PIROLLI, P. 1995. Modelling individual differences in students' learning strategies. *Journal of the learning sciences*, 4(1):1-38, January 1.

RICHFIELD, J. (jonrichfield@msgto.com) 2000a. Syllabus. Comp-studies@wcape.school.za. March 23.

RICHFIELD, J. (jonrichfield@msgto.com) 2000b. Next generation CS environment. Comp-studies@wcape.school.za. March 26.

RICHFIELD, J. (jonrichfield@msgto.com) 2000c. Brave new syllabus. Comp-studies@wcape.school.za. April 5.

RICHFIELD, J. (jonrichfield@msgto.com) 2000d. Re: Brave new syllabus en ook Turbo what? en Nuwe sillabus. Comp-studies@wcape.school.za. April 7.

RICHFIELD, J. (jonrichfield@msgto.com) 2000e. Re: CS HG - Delphi the route to go? Comp-studies@wcape.school.za. May 17.

RICHFIELD, J. (jonrichfield@msgto.com) 2000f. Re: CS HG - Delphi the route to go? Comp-studies@wcape.school.za. May 19.

RICHFIELD, J. (jonrichfield@msgto.com) 2000g. Re: CS HG - Delphi the route to go? Comp-studies@wcape.school.za. May 22.

RICHFIELD, J. (jonrichfield@msgto.com) 2000h. Re: The language. Comp-studies@wcape.school.za. May 23.

ROBBERTSE, H. (rihette@lantic.net) 2000. Re: Nuwe tale. [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 17.

ROGALSKI, J. & SAMURÇAY, R. 1990. Acquisition of programming knowledge and skills. (*In* Hoc, J.-M., Green, T.R.G., Samurçay, R. & Gilmore, D.J., *eds.* Psychology of programming. London : Academic Press. p. 157-174.)

ROGERS, D.H. (David@mhs.kzn.school.za) 2000a. Re: CS HG - Delphi the route to go? Comp-studies@wcape.school.za. May 17.

ROHWER, W.D. & THOMAS, J.W. 1989. The role of problem-solving activities in learning to program. *Journal of educational psychology*, 81(4):584-593, December.

ROMBERG, T.A. 1997. The influence of programs from other countries on the school mathematics reform curricula in the United States. *American journal of education*, 106(1):127-147, November.

SAINSBURY, P. (paul.sainsbury@iname.com) 2000. Edited version. [E-mail to:] Gibson, K. (gibsonk@mail.arhs.ecape.school.za) July 31.

SANZ-CASADO, *E.,* SUAREZ-BALSEIRO, C., GARC´ÝA-ZORITA, C., MART´ÝN-MORENO, C. & LASCURAIN-S´ANCHEZ, M.L. 2002. Metric studies of information: an approach towards a practical teaching method. *Education for Information*, 20(2):133-144.

SAS INSTITUTE INCORPORATED. 1999a. The SAS System for Windows Release 8.02 TS Level 02M0 Copyright© 1999-2001 by SAS Institute Inc., Cary, NC, USA.

SAS INSTITUTE INCORPORATED. 1999b. SAS OnlineDoc®, Version 8, Cary, NC, USA.

SCHNEIDER, G.M., GERSTING, J.L. & BAASE, S. 2000. An invitation to computer science. Pacific Grove : Brooks/Cole.

SCORDILIS, M.C. (mikescor@iafrica.com) 2000a. Re: Programming language. Comp-studies@wcape.school.za. August 12.

SCORDILIS, M.C. (mikescor@iafrica.com) 2000b. Re: Programming language? The choice? Training, teaching or facilitation. Comp-studies@wcape.school.za. August 12.

SEGAAR, T. (segaar@icon.co.za) 2000. Re: Rekenaarstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 17.

SHEAR, S. (sshear@commerce.uct.ac.za) 2000. Teaching of VB in schools. [E-mail to:] Kurata, D. (deborahk@insteptech.com) August 22.

SINGH, J.K. 1992. Cognitive effects of programming in Logo: a review of literature and synthesis of strategies for research. *Journal of research on computing in education*, 25(1) 88-104, Fall. [In EBSCOHost: Academic Search Elite, Full Display: http: www-sa.ebsco.com] [Date of access: 21 March 2003].

SMIT, F. (fransiesmit@mweb.co.za) 2000a. Rekenaarstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Goosen, L. (leila@netactive.co.za) August 15.

SMIT, F. (fransiesmit@mweb.co.za) 2000b. Re: SOS aan RS onderwysers. [E-mail to:] Goosen, L. (leila@netactive.co.za) November 2.

SOUTH AFRICA. DEPARTMENT OF EDUCATION. 1994. Interim syllabus for computer studies. Pretoria : Government Printer.

SOUTH AFRICA. DEPARTMENT OF EDUCATION. 2003a. National curriculum statement Grades 10-12 (Schools): Guidelines for the development of learning programmes - Computer science. *Government Gazette*, 4342, October 28. (Regulation Gazette No: 3434.)

SOUTH AFRICA. DEPARTMENT OF EDUCATION. 2003b. National curriculum statement Grades 10-12 (Schools): Information technology (Computer science) subject statement. *Government Gazette*, 4342, October 28. (Regulation Gazette No: 3434.)

SPILLANE, J.P. & CALLAHAN, K.A. 2000. Implementing state standards for science education: what district policy makers make of the hoopla. *Journal of research in science teaching*, 37(5):401-425, May.

STANDISH, T.A. 1998. Data structures in Java. Reading, Mass. : Addison-Wesley.

STEPHENSON, C. 2002. Software engineering in the computer science curriculum. (Presentation at the ACM Symposium on Teaching Computer Programming held at St John's College, Johannesburg on 19 and 20 September.) [Web:] http://www.compprogsa.tk.html [Date of access: 30 September 2002].

STEYN, H.S. (jr.). 2000. Practical significance of the difference in means. *Journal of industrial psychology*, 26(3):1-3.

SUÁREZ, M., PÍAS, R., MEMBIELA, P. & DAPÍA, D. 1998. Classroom environment in the Implementation of an innovative curriculum project in science education. *Journal of research in science teaching*, 35(6):655-671, August.

SULISTYO-BASUKI, L. 1999. Information technology and library education in Indonesia: recent developments in the curriculum. *Education for information*, 17:353-361.

THOMAS, R.A. & UPAH, S.C. 1996. Give programming instruction a chance. *Journal of research on computing in education*, 29(1):96-118, Fall. [In EBSCOHost: Academic Search Elite, Full Display: http: www-sa.ebsco.com] [Date of access: 10 August 2001].

TRUTER, A. 2000. A comparison between Pascal and Java programming. (*In* Interface 2000. Computing at schools, technikons and universities. Papers read at the Interface 2000 conference held on May 19 and 20 in Pretoria. Pretoria. p. 24-31.)

TWEEDIE, S. (Irish_Man@dc.wcape.school.za) 1999. Re: Where to after Pascal? Comp-studies@wcape.school.za. April 14.

TWEEDIE, S. (root@tweedie.wcape.school.za) 2000c. Re: CS HG - Delphi the route to go? Comp-studies@wcape.school.za. May 17.

TWEEDIE, S. (root@tweedie.wcape.school.za) 2000d. The language. Comp-studies@wcape.school.za. May 21.

TWEEDIE, S. (irishman@abbottsc.wcape.school.za) 2000e. ?Programming language? Comp-studies@wcape.school.za. May 25.

TWEEDIE, S. (irishman@abbottsc.wcape.school.za) 2000f. Comp-studies@wcape.school.za. .

VAN BILJON, A. (anelize@wwg3.uovs.ac.za) 2000. Rekenaartaal 2001. Comp-studies@wcape.school.za. August 7.

VAN DEN BERG, A. 1990. 'n Programontwerpsbenadering en 'n rekenaargesteunde programontwikkelingsomgewing vir onderrig in rekenaarwetenskap. Pretoria : University of Pretoria. (Thesis - MSc.)

VAN DER MERWE, J. (johanvdm@intekom.co.za) 2000. Re: Rekenaarstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 18.

VAN MERRIENBOER, J.J.G. 1990. Instructional strategies for teaching computer programming: interactions with the cognitive style reflection-impulsivity. *Journal of research on computing in education,* 23(1):45-53 , Fall. [In EBSCOHost: Academic Search Elite, Full Display: http: www-sa.ebsco.com] [Date of access: 20 March 2003].

VAN ROSSUM, G. 1999. Computer programming for everybody. [Web:] http://www.python.org/doc/essays/cp4e.html [Date of access: 30 October 2000].

VENTER, A. (shs@sutherlandhs.co.za) 2000a. Re: Rekenaatstudie onderwysers: SOS - Antwoord asb dringend! [E-mail to:] Smit, F. fransiesmit@mweb.co.za) August 17.

VENTER, A. (shs@sutherlandhs.co.za) 2000b. Re: VB as programming language. [E-mail to:] Smit, F. (fransiesmit@mweb.co.za) August 18.

VERHOEVEN, P. & VERLOOP, N. 2002. Identifying changes in teaching practice: innovative curricular objectives in classical languages and the taught curriculum. *Journal of curriculum studies*, 34(1):91-102, January.

VESILIND, E.M. & JONES, M.G. 1998. Gardens or graveyards: science education reform and school culture. *Journal of research in science teaching*, 35(7):757-775, September.

VICTOR, K. (koosv@iafrica.com) 2000. Java. Comp-studies@wcape.school.za. July 26.

VON SOLMS, S.H. 1979. Enkele gedagtes oor die onderrig van rekenaarstudie op sekondêre en tersiêre vlak. Johannesburg : Randse Afrikaanse Universiteit.

WALKER, H.M. 1996. A brief history of the computing curriculum at Grinnell College. Last revised: February 8. [Web:] http://www.math.grin.edu/~walker/coll-learning/index.html [Date of access: 21 March 2003].

WALKER, H.M. 1997. Collaborative learning: a working paper. (Draft position paper written in preparation for the working group on "Computer Supported Collaborative Learning" at ITiCSE, Conference on Integrating Technology into Computer Science Education, at Uppsala Sweden, June 1-5, 1997.) [Web:] http://www.math.grin.edu/~walker/coll-learning/index.html [Date of access: 21 March 2003].

WALKER, H.M. 2000. AP CS update. [Web:] http://www.math.grin.edu/~walker/sigcse-ap/update-6.16.00.html [Date of access: 10 August 2001].

WALKER, H.M. (walker@cs.grinnell.edu) 2000. Survey for AP CS. [E-mail to:] SIGCSE.MEMBERS@acm.org. August 1.

WALKER, H.M. & SCHNEIDER, G.M. 1994. A revised model curriculum for a liberal arts degree in computer science. (Draft of an article that appeared in the December 1996 issue of the Communications of the ACM.) [Web:] http://www.math.grin.edu/~walker/coll-learning/index.html [Date of access: 21 March 2003].

WALTERS, S.A. (fanus@walters.wcape.school.za) 2000. Re: New syllabus. Comp-studies@wcape.school.za. March 27.

WASSERMAN, U. (uwasserman@sacte.ac.za) 2000a. Re: The language. Comp-studies@wcape.school.za. May 22.

WASSERMAN, U. (uwasserman@sacte.ac.za) 2000b. Re: Teaching Delphi. Comp-studies@wcape.school.za. May 26.

WEBOPEDIA see JUPITERMEDIA CORPORATION

WEINBERG, G.M. 1971. The psychology of computer programming. New York : Van Nostrand Reinhold Company.

WESTROM, M. 1992. Teaching structured programming in the secondary schools. *Journal of Research on computing in education*, 25(2):274-276, Winter.

WIRFS-BROCK, R., WILKERSON, B. & WEINER, L. 1990. Designing object-oriented software. Englewood Cliffs, N.J. : Prentice-Hall.

WIRTH, N. 1986. Algorithms and data structures. Englewood Cliffs, N.J. : Prentice-Hall.

ZIMMER, J. A. 1985. Abstraction for programmers. New York : McGraw-Hill.

APPENDIX A: QUESTIONNAIRE **(Office use - Questionnaire number: [ ][ ][ ]) (1-3)**

CRITERIA AND GUIDELINES FOR THE SELECTION AND IMPLEMENTATION OF A FIRST PROGRAMMING LANGUAGE IN HIGH SCHOOLS

- **In an effort to empirically verify within the South African context the validity of selection criteria and implementation guidelines identified, your friendly cooperation is requested in response to this questionnaire.**
- **There are no "wrong" or "right" answers to any of the questions - answers to questions should reflect your opinion.**
- **Responses will be used only for the purposes of this enquiry – it will in no way be used to evaluate you or your school/department.**
- **All information will be treated as strictly confidential.**

INDICATIONS: *Complete each of the following questions by circling your choice, or supplying further information, where required, in the blocks.*

Section A (Biographical data)

1. Province: (4)

Eastern Cape	1
Free State	2
Gauteng	3
KwaZulu-Natal	4
Limpopo	5
Mpumulanga	6
Northern Cape	7
North-West Province	8
Western Cape	9

2. Your highest qualification in Computer Science: (5)

First year level	1
Second year level	2
Third year level	3
Honours level	4
Masters level	5
Doctoral level	6
Other (please specify):	7

3. Your highest academic qualification: (6)

B degree	1
Honours degree	2
M degree	3
D degree	4
Other (please specify):	5

4. Your highest educational qualification: (7)

HED	1
FDE	2
B.Ed	3
M.Ed	4
D.Ed/Ph.D	5
Other (please specify):	6

5. Please indicate the number of years (rounded to nearest completed) that you have been involved in each of the following activities:

	Number of years:	0	1-2	3-4	5-6	7-10	More than 10	
5.1	**Writing Committee for the National Curriculum Statement**	0	1	2	3	4	5	(8)
5.2	**Curriculum Advisor**	0	1	2	3	4	5	(9)
5.3	**Administrative capacity - provincial level**	0	1	2	3	4	5	(10)
5.4	**Provincial Matric Examiner / Moderator**	0	1	2	3	4	5	(11)
5.5	**Teacher of Computer Studies**	0	1	2	3	4	5	(12)
5.6	**Training of Computer Studies teachers**	0	1	2	3	4	5	(13)
5.7	**Other (please specify):**	0	1	2	3	4	5	(14)

6. **SELECTION CRITERIA FOR PROGRAMMING LANGUAGE**

Section B (Importance): Please indicate in the column on the left hand side how important it would be to use each of the following criteria for selection when choosing a programming language for South African high schools:	**Section C (Applicability)**: Although some or all of the selection criteria listed below might be important, it might not be practical to use some of them when selecting a programming language. Please indicate in the column on the right hand side to what extent each of them (have been/are being) applied to select programming languages for South African high schools:

	Not important at all	Fairly unimportant	Fairly important	Very important	SELECTION CRITERIA FOR PROGRAMMING LANGUAGE	Not applied at all	Sometimes applied	Usually applied	Always applied	
					Selection criteria for a programming language for South African high schools should					
6.1	(15)	1	2	3	4	have worldwide relevance	1	2	3	4	(18)
6.2	(16)	1	2	3	4	be relevant for a reasonable period	1	2	3	4	(19)
6.3	(17)	1	2	3	4	suit specific needs relevant to the South African context	1	2	3	4	(20)

		Not important at all	Fairly unimportant	Fairly important	Very important	SELECTION CRITERIA FOR PROGRAMMING LANGUAGE	Not applied at all	Sometimes applied	Usually applied	Always applied	
						The programming language and its software development environment should					
6.4	(21)	1	2	3	4	suit the needs of novice programmers	1	2	3	4	(44)
6.5	(22)	1	2	3	4	be safe, stable, structured and controlled	1	2	3	4	(45)
6.6	(23)	1	2	3	4	not frustrate learners because of features suited to professional programmers	1	2	3	4	(46)
6.7	(24)	1	2	3	4	provide effective debugging tools	1	2	3	4	(47)
6.8	(25)	1	2	3	4	provide understandable error-messages	1	2	3	4	(48)
6.9	(26)	1	2	3	4	be easy to learn	1	2	3	4	(49)
6.10	(27)	1	2	3	4	offer relative simplicity of commands	1	2	3	4	(50)
						The chosen programming language should					
6.12	(29)	1	2	3	4	adequately match the abilities of learners	1	2	3	4	(52)
6.13	(30)	1	2	3	4	provide an instructional environment promoting development of higher order thinking skills	1	2	3	4	(53)
6.14	(31)	1	2	3	4	provide an instructional environment promoting development of critical thinking skills	1	2	3	4	(54)
6.15	(32)	1	2	3	4	provide an instructional environment promoting development of problem solving abilities	1	2	3	4	(55)
6.16	(33)	1	2	3	4	facilitate the development of learners' understanding of their own thought processes	1	2	3	4	(56)
6.17	(34)	1	2	3	4	encourage a self-regulated approach to solving problems	1	2	3	4	(57)
6.18	(35)	1	2	3	4	encourage programming principles such as abstraction	1	2	3	4	(58)
6.19	(36)	1	2	3	4	promote top-down design with step-wise refinement	1	2	3	4	(59)
6.20	(37)	1	2	3	4	support good programming style	1	2	3	4	(60)
6.21	(38)	1	2	3	4	support new tendencies in programming	1	2	3	4	(61)
6.22	(39)	1	2	3	4	offer possibilities for object-oriented design	1	2	3	4	(62)
6.23	(40)	1	2	3	4	offer possibilities for visual programming	1	2	3	4	(63)
6.24	(41)	1	2	3	4	offer possibilities for encapsulation	1	2	3	4	(64)
6.25	(42)	1	2	3	4	offer possibilities for inheritance	1	2	3	4	(65)
6.26	(43)	1	2	3	4	offer possibilities for polymorphism	1	2	3	4	(66)

		Not important at all	Fairly unimportant	Fairly important	Very important	SELECTION CRITERIA FOR PROGRAMMING LANGUAGE	Not applied at all	Sometimes applied	Usually applied	Always applied	
						The chosen programming language should					
6.27	(67)	1	2	3	4	be internationally standardised	1	2	3	4	(83)
6.28	(68)	1	2	3	4	have reasonable prospects for continued support from its developers	1	2	3	4	(84)
6.29	(69)	1	2	3	4	enable Internet programming	1	2	3	4	(85)
6.30	(70)	1	2	3	4	support database programming.	1	2	3	4	(86)
6.31	(71)	1	2	3	4	follow international trends with regard to programming languages used in high-school education	1	2	3	4	(87)
6.32	(72)	1	2	3	4	be affordable	1	2	3	4	(88)
6.33	(73)	1	2	3	4	enable affordable in-service training of CS teachers	1	2	3	4	(89)
6.34	(74)	1	2	3	4	be compatible with tertiary establishments' choice of programming language	1	2	3	4	(90)
6.35	(75)	1	2	3	4	be included in the pre-service training of CS teachers	1	2	3	4	(91)
6.36	(76)	1	2	3	4	have learning and teaching support materials and resources, such as question banks, instructional packages, workbooks and study guides, available	1	2	3	4	(92)
6.37	(77)	1	2	3	4	have particularly textbooks suited to learning the language at high school level available	1	2	3	4	(93)
6.38	(78)	1	2	3	4	have resources appropriate to an OBE approach to the subject available	1	2	3	4	(94)
6.39	(79)	1	2	3	4	support programming for various purposes	1	2	3	4	(95)
6.40	(80)	1	2	3	4	compatible with academic tools	1	2	3	4	(96)
6.41	(81)	1	2	3	4	compatible with commercial tools	1	2	3	4	(97)
6.42	(82)	1	2	3	4	be popular in industry, where there is a demand for such programmers	1	2	3	4	(98)

6.43 If you have comments/remarks about some of the statements used in this part of the questionnaire, please refer to a particular statement by quoting the appropriate number, e.g. 6.12, in front of that statement:

7. IMPLEMENTATION GUIDELINES FOR PROGRAMMING LANGUAGE

Section D (Importance): Please indicate in the column on the left hand side how important it would be to use each of the following guidelines for implementation when implementing (a) programming language(s) in South African high schools:		**Section E (Usage)**: Although some or all of the implementation guidelines listed below might be important, some of them might be very difficult to implement effectively. Please indicate in the column on the right hand side to what extent each of them (have been/are being) used to implement programming languages in South African high schools:

		Not important at all	Fairly unimportant	Fairly important	Extremely important	IMPLEMENTAION GUIDELINES FOR PROGRAMMING LANGUAGE	Not used at all	Sometimes used	Usually used	Implemented fully	
7.1	(99)	1	2	3	4	Perform a situation analysis, using various methods for gathering information about the implementation process.	1	2	3	4	(112)
7.2	(100)	1	2	3	4	Show appreciation for teachers' roles in implementation at all levels in curriculum process.	1	2	3	4	(113)
7.3	(101)	1	2	3	4	Consider cost implications at macro-implementation (national/provincial) level.	1	2	3	4	(114)
7.4	(102)	1	2	3	4	Make guiding documentation about implementation of a programming language available to teachers.	1	2	3	4	(115)
7.5	(103)	1	2	3	4	Grant teachers the necessary freedom to implement at classroom level.	1	2	3	4	(116)
7.6	(104)	1	2	3	4	Develop learning and teaching materials and resources to accompany and support a new curriculum.	1	2	3	4	(117)
7.7	(105)	1	2	3	4	Develop textbooks particularly suited to high school level to accompany and support a new curriculum.	1	2	3	4	(118)
7.8	(106)	1	2	3	4	State outcomes that need to be achieved clearly and in a concise way.	1	2	3	4	(119)
7.9	(107)	1	2	3	4	Perform preliminary testing of a new curriculum in the form of pilot testing.	1	2	3	4	(120)
7.10	(108)	1	2	3	4	Participation of teachers in pilot testing takes place on a voluntarily basis.	1	2	3	4	(121)
7.11	(109)	1	2	3	4	Use experienced teachers to conduct pilot testing.	1	2	3	4	(122)
7.12	(110)	1	2	3	4	Carry out programme assessment in order to revise and improve a new curriculum to a standard suitable for implementation.	1	2	3	4	(123)
7.13	(111)	1	2	3	4	Convince teachers that their implementation efforts will be supported.	1	2	3	4	(124)

		Not important at all	Fairly unimportant	Fairly important	Extremely important	IMPLEMENTAION GUIDELINES FOR PROGRAMMING LANGUAGE	Not used at all	Sometimes used	Usually used	Implemented fully	
7.14	(125)	1	2	3	4	Make teachers aware of the differences between old and new practices.	1	2	3	4	(142)
7.15	(126)	1	2	3	4	Address teachers' knowledge of content and concepts in a new curriculum by providing practical advice about teaching.	1	2	3	4	(143)
7.16	(127)	1	2	3	4	Gauge teachers' expectations for implementation.	1	2	3	4	(144)
7.17	(128)	1	2	3	4	Specify and classify learning content that is of consequence to learners to initiate the development of a curriculum.	1	2	3	4	(145)
7.18	(129)	1	2	3	4	Distribute information regarding a new curriculum in a way that will overcome resistance to change.	1	2	3	4	(146)
7.19	(130)	1	2	3	4	Make the relative advantage of reform clear to teachers.	1	2	3	4	(147)
7.20	(131)	1	2	3	4	Identify factors/variables that will advance or slow down implementation.	1	2	3	4	(148)
7.21	(132)	1	2	3	4	Have educational leaders provide progressive leadership during curriculum improvement by mobilising administration and resources in support of teachers.	1	2	3	4	(149)
7.22	(133)	1	2	3	4	Create opportunities for teachers from different schools to discuss implementation.	1	2	3	4	(150)
7.23	(134)	1	2	3	4	Handle sources of resistance to change and obstacles to implementation identified in such a manner that implementation can proceed smoothly.	1	2	3	4	(151)
7.24	(135)	1	2	3	4	Give teachers the time and resources to be retrained properly.	1	2	3	4	(152)
7.25	(136)	1	2	3	4	Distribute curriculum documents and materials to schools.	1	2	3	4	(153)
7.26	(137)	1	2	3	4	Provide teachers with guidance and support during implementation.	1	2	3	4	(154)
7.27	(138)	1	2	3	4	Continue support for teachers as implementation becomes established and consolidated into an accepted part of curricular practice.	1	2	3	4	(155)
7.28	(139)	1	2	3	4	Involve teachers in the assessment of a new curriculum.	1	2	3	4	(156)
7.29	(140)	1	2	3	4	Assess a new curriculum according to the extent to which its implementation leads to the achievement of learning outcomes.	1	2	3	4	(157)
7.30	(141)	1	2	3	4	Determine to what extent a new curriculum was indeed implemented.	1	2	3	4	(158)

8. **Section F (Agreement):** Please indicate for each of the following statements to what extent you agree/disagree with a particular statement:

		Strongly disagree	Disagree with reservations	Agree with reservations	Strongly agree	
8.1	Programming knowledge provides needed background to deal with modern society issues.	1	2	3	4	(159)
8.2	Learners develop general thinking skills identifiable and transferable to other situations.	1	2	3	4	(160)
8.3	Learning the programming language should produce learners who are life-long learners.	1	2	3	4	(161)
8.4	Learners should be provided with the necessary tools and skills to adapt to a lifetime of change.	1	2	3	4	(162)
8.5	Learners should be able to prosper over the length of long careers.	1	2	3	4	(163)
8.6	Options for a language-independent exam need to be considered if provinces implement different languages.	1	2	3	4	(164)
8.7	The programming section currently contained in written papers should no longer be assessed externally, but instead only projects used for this purpose.	1	2	3	4	(166)
8.8	It is more important that a programming language facilitate the learning of programming principles than to learn a language learners will encounter in tertiary or private training.	1	2	3	4	(167)

9. If you have comments/remarks about some of the statements used in this questionnaire, please refer to a particular statement by quoting the appropriate number, e.g. 7.12, in front of that statement:

10. Any other comments/remarks:

THANK YOU FOR YOUR COOPERATION!

APPENDIX B: ABBREVIATIONS

ABET	Adult Basic Education Training
AP	advanced placement
CPU	central processing unit
CS	Computer Studies / Science
DoE	Department of Education
FAQ	frequently-asked-questions
FDE	Further Diploma in Education
FET	Further Education and Training
GET	General Education and Training
GUI	graphical user interface
HED	Higher Education Diploma
HG	Higher Grade
HTML	Hyper text markup language
I/O	Input/Output
ICT	Information and Communication Technologies
IDE	Integrated Development Environment
IEB	Independent Examinations Board
IOI	International Olympiad in Informatics
LACS	Liberal Arts Computer Science Consortium
NCS	National Curriculum Statement
NPDE	National Professional Diploma in Education
OBE	Outcomes-based education
OO	object-oriented / object-orientation
OOP	object-orientated programming
PBL	problem-based learning
RAD	rapid application development
RMSE	root of the mean square error
SA	South Africa(n)
SAQF	South African Qualification Framework
SIGCSE	Special Interest Group for Computer Science Education
SQL	Structured Query Language
THED	Technical Higher Education Diploma
TP	Turbo Pascal
USA	United States of America