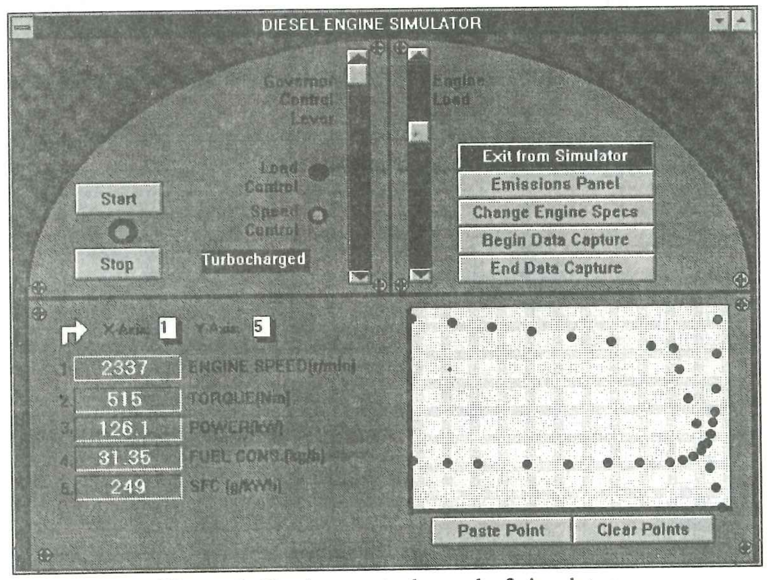
the objective of providing students with an exploratory tool for learning how a diesel engine responds to adjustments in governor control lever position and to changes in engine load. The simulation was programmed for the Windows environment using a multimedia authoring package. The engine functioned in accordance with simple textbook principles. The first part of the simulator required the input of engine performance parameters that were used to define the bounds within which the engine was expected to operate. For the second part, a full screen engine panel illustrated in Figure 1 was designed to incorporate various elements including push buttons, slider buttons, digital meters and a graph panel to allow the student to interact directly with the engine and to observe its responses on a dynamic and instantaneous basis.

This type of interaction is ideal for visual, active, sensing and global learners, and if used correctly, can exercise their application, analysis, synthesis and evaluation skills.

Analysis

In this category, one of the most commonly used software packages is spreadsheets, which allow the user to assemble relatively complex models that rely on suitable input data, equations and macros where necessary, to generate results and graphs. Another group of software packages that fall into the category of analysis are the equation solvers such as Mathematica, MATLAB and Mathcad. Students are able to set up a system of equations with inputs in an appropriate sequence and obtain a solution in numerical and graphical form. The interactive nature of these programs allows the student to use a significant amount of time exploring complex problems by varying system parameters and operating conditions and observing the results almost immediately rather than spending tedious time programming the system of equations used to model the physical system as well as the numerical techniques needed to solve these equations. With the emphasis on analysis rather than programming, the student not only learns through discovery from the results of varying parameters, but he/she has the opportunity to practise his/her creativity and synthesis skills.

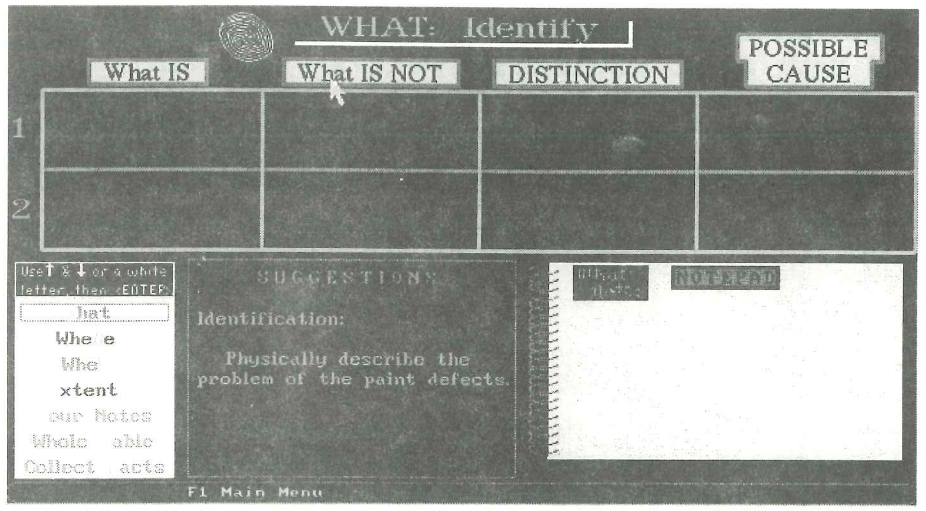


1. 2337 ENGINE SPEED(r/min)
2. 515 TORQUE(N-m)
3. 126.1 POWER(kW)
4. 31.35 FUEL CONS.(kg/h)
5. 249 SFC (g/kWh)

Figure 3 Engine control panel of simulator

This process has been taken a step further by combining textbooks with equation solvers, in which solutions to specific problems in the text are provided. An example is the Schaum's Interactive Outline series published by McGraw Hill and MathSoft Inc., which incorporate a Mathcad Electronic Book. Such applications address all six levels of Bloom's taxonomy and help virtually all learner types in Table 1.

Another example involving the development of qualitative problem solving skills is a set of computer modules developed to supplement a problem solving textbook by Fogler and LeBlanc (1994). An example of a screen from the problem analysis module is shown in Figure 4.



Identification:

Physically describe the problem of the paint defects.

Figure 4 Problem analysis sample screen (Fogler and LeBlanc, 1994)

Conclusions

Software that has been developed more recently for instruction of engineering students addresses many of the educational objectives put forward in Bloom's taxonomy. In future development of computer-based instructional materials, it is vital that appropriate pedagogical objectives and evaluation methods be identified right from the start to ensure that the software is effective.

References

Andrews, SJ, 1994. Some cultural and perceptual implications of courseware development and the use of technology within a multicultural, multilingual society (a cautionary tale). Proc. International Conference on Computer-Assisted Education and Training in Developing Countries, Eskom Conference and Exhibition Centre, Halfway House, Johannesburg.

Felder, RM and Silverman, LK, 1988. Learning and teaching styles in engineering education. Engineering Education, 78(7), 674-681.

Fogler, HS and LeBlanc, SE, 1994. Strategies for creative problem solving. Publ. Prentice-Hall PTR, New Jersey, USA.

Grayson, DJ, 1996. Using education research to develop waves courseware. Computers in Physics, 10(1): 30-37

Hansen, AC, Reid, JF and Goering, CE, 1995. Diesel engine simulator as a multimedia educational tool. Proc. Annual Conference of American Society for Engineering Education, Anaheim, California, USA. Vol 1: 140-144.

Lown, JB, Prather, TG and Peters, PA, 1993. Multimedia training for loggers, arborists and woodcutters. ASAE paper 937523, Winter Meeting. American Society of Agricultural Engineers, Chicago, Illinois, USA.

Montgomery, S, 1995. Addressing diverse learning styles in a large classroom. Proc. Annual Conference of American Society for Engineering Education, Anaheim, California, USA. Vol 1: 344-349.

Montgomery, S and Fogler, HS, 1996. Selecting computer-aided instructional software. Journal of Engineering Education, 85(1): 53-60.

Reeves, TC, 1994. A model of the effective dimensions of interactive learning. Proc. International Conference on Computer-Assisted Education and Training in Developing Countries, Eskom Conference and Exhibition Centre, Halfway House, Johannesburg.

Rosati, PA and Felder, RM, 1995. Engineering student responses to an index of learning styles. Proc. Annual Conference of American Society for Engineering Education, Anaheim, California, USA. Vol 1: 739-743.

Sheldon, EJ, Field, WE and Tormoehlen, RL, 1993. CAI/Multimedia approach to farm tractor and machinery safety certification. ASAE paper 933540, Winter Meeting. American Society of Agricultural Engineers, Chicago, Illinois, USA.

SOFTWARE ENGINEERING DEVELOPMENT METHODOLOGIES APPLIED TO COMPUTER-AIDED INSTRUCTION

Paula Kotzé and Ruth de Villiers
Department of Computer Science and Information Systems
University of South Africa, P O Box 392, Pretoria, 0001
Email: {kotzep,dvillmr}@alpha.unisa.ac.za

Abstract

The research reported in this paper aims to integrate software engineering approaches with instructional factors in the requirements, analysis, design and production phases of instructional software development. The integration has resulted in the evolution of a branch of software engineering (SE) called courseware engineering. The paper reports on the results of an independent study we have undertaken into the aspects of SE that are appropriate for the development of courseware. Examples of other research efforts combining the disciplines of SE and instructional system development (ISD) are given, where applicable. Two SE methods were identified as being particularly useful and appropriate to ISD: prototyping and object-oriented (OO) design. Factors that support the use and applicability of these two approaches are discussed.

Introduction

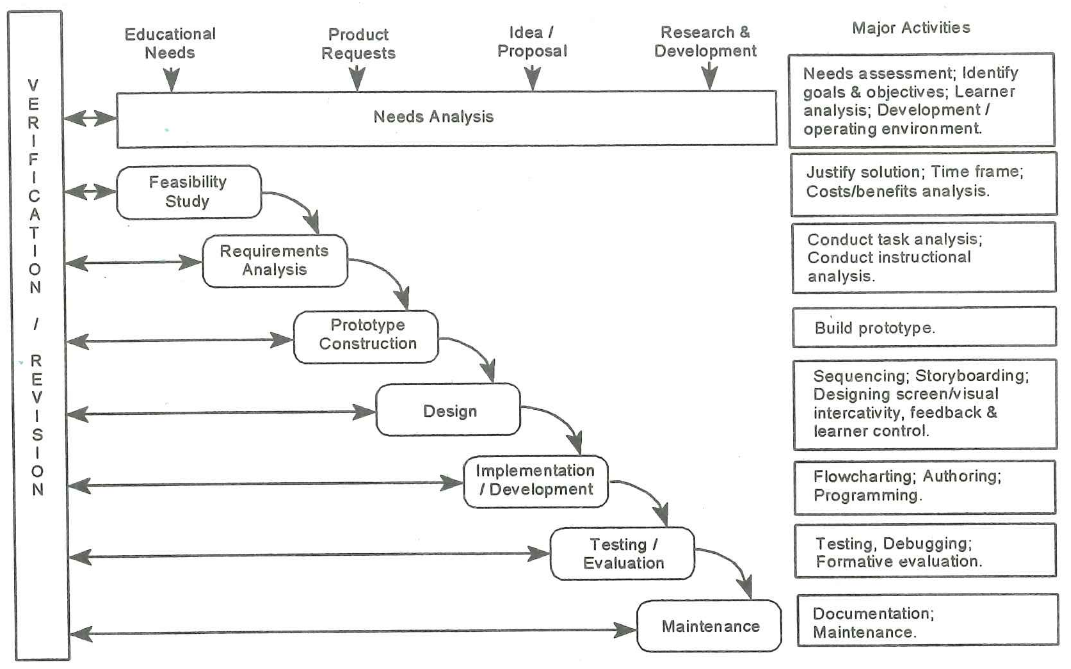
Software engineering (SE) is concerned with the development of software systems using sound engineering principles including both technical and non-technical aspects – over and above the use of specification, design and implementation techniques, human factors and software management should also be addressed. Well-engineered software provides the services required by its users. Such software should be produced in a cost-effective way and should be appropriately functional, maintainable, reliable, efficient and provide a relevant user interface [Shneiderman 1992, Sommerville 1992].

Computer-aided instruction (CAI) is concerned with the way in which computers can be used to support students engaged in particular educational activities and incorporates a variety of computer-aided instruction and learning modes, for example, formal courseware such as tutorials and drills, and more open-ended software such as simulations, concept maps and microworlds. Over the last decade such systems have proliferated, evolving from simple beginnings, merely in the educational territory, to the realm of complex software systems. As such, they should be designed and developed by applying the established principles of software engineering to instructional software development (ISD).

The general problem of ISD is therefore to develop appropriate methods for the specification, design and implementation of CAI software systems. The research reported in this paper aims to integrate software engineering approaches with instructional factors in the requirements, analysis, design and production phases of instructional software development. The integration has resulted in the evolution of a branch of software engineering called courseware engineering. Two SE methods are identified as being particularly useful and appropriate to ISD: prototyping and object-oriented (OO) design. Factors that support the use and applicability of these two approaches are discussed.

Software Engineering Models

One of the cornerstones of SE is the software life cycle, describing the activity stages that take place from the initial concept formation for a software system, up to its implementation and eventual phasing out and replacement. Complementing these life cycle models are a number of design and development models.



VERIFICATION / REVISION
Educational Needs
Product Requests
Idea / Proposal
Research & Development
Major Activities
Needs Analysis
Needs assessment; Identify goals & objectives; Learner analysis; Development / operating environment.
Feasibility Study
Justify solution; Time frame; Costs/benefits analysis.
Requirements Analysis
Conduct task analysis; Conduct instructional analysis.
Prototype Construction
Build prototype.
Design
Sequencing; Storyboarding; Designing screen/visual intercativity, feedback & learner control.
Implementation / Development
Flowcharting; Authoring; Programming.
Testing / Evaluation
Testing, Debugging; Formative evaluation.
Maintenance
Documentation; Maintenance.

Figure 1: Chen and Shen's life cycle model [Chen 1989, p 11]

Life cycle models

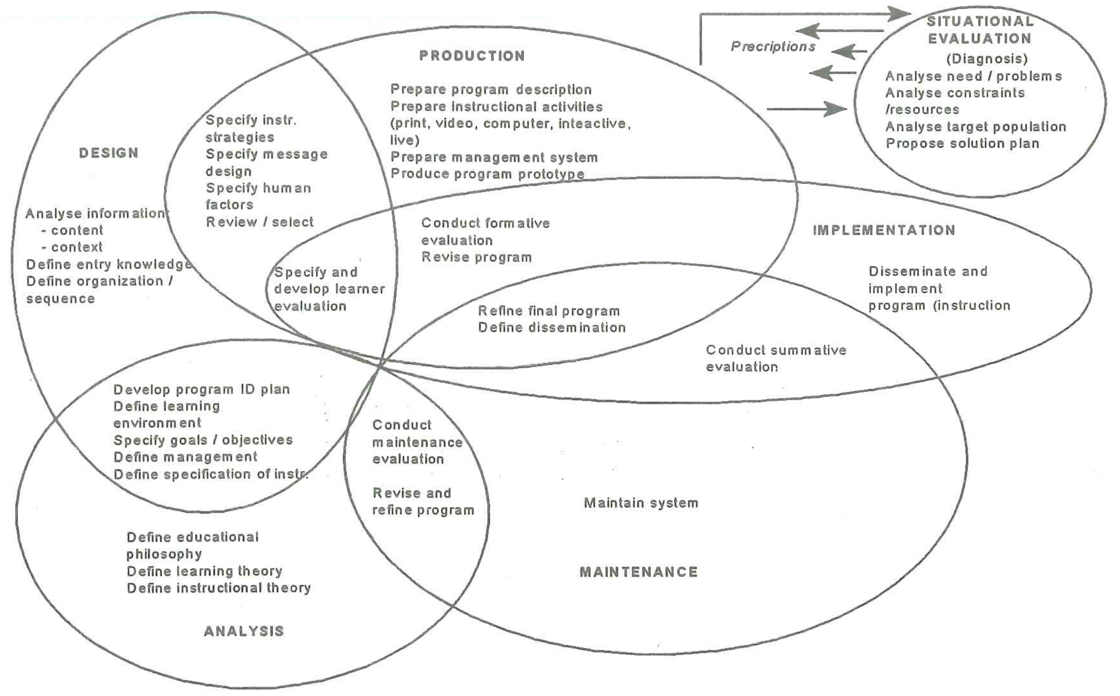
A multitude of general software engineering life cycle (or process) models exist, some being variations on others. Two examples are the so-called waterfall model, and the spiral model.

- *Waterfall model*: In the waterfall approach [Budgen 1994, Conger 1994, Sommerville 1992] the software process is viewed as made up of a number of stages or activities such as requirements specification, software design, software implementation, testing, operation and maintenance, etc. Each activity serves as input to the next. One disadvantage of this approach is that it suits a principled approach to design where all requirements for a system have to be known before system development is begun.

  The behaviour of a CAI system is highly dependent on the domain knowledge modelled within the system. The tasks a user will perform are often not known until the user is familiar with the system on which he performs them. A second drawback of this process model is that it does not promote the use of notations and techniques which support the user's perspective of the CAI system.

- *Spiral model*: An alternative to the waterfall approach is Boehm's spiral model [Boehm 1988] which is essentially an iterative model. Its key characteristic is an assessment of management risk items at various stages of the project and the initiation of actions to counteract these risks. Before each cycle, a review procedure judges whether to move on to the next cycle in the spiral. A cycle of the spiral commences by elaborating objectives such as performance, functionality, and so on. Alternative ways of achieving these objectives and constraints are then enumerated, followed by an assessment of each objective. This typically results in the identification of sources of project risk. The next step is to evaluate these risks by activities such as more detailed analysis, prototyping, simulation, etc. After risk evaluation a development model (or a combination of models) for the system is chosen.

Combining SE aspects with those related specifically to the design and development of instructional systems resulted in a number of specialised life cycle models specifically aimed at the development of computer-based instructional systems. Two examples are:



SITUATIONAL EVALUATION
(Diagnosis)
Analyse need / problems
Analyse constraints /resources
Analyse target population
Propose solution plan
Precriptions
PRODUCTION
Prepare program description
Prepare instructional activities (print, video, computer, inteactive, live)
Prepare management system
Produce program prototype
DESIGN
Specify instr. strategies
Specify message design
Specify human factors
Review / select
Analyse information
- content
- context
Define entry knowledge
Define organization / sequence
Conduct formative evaluation
Revise program
Specify and develop learner evaluation
IMPLEMENTATION
Disseminate and implement program (instruction
Refine final program
Define dissemination
Conduct summative evaluation
Develop program ID plan
Define learning environment
Specify goals / objectives
Define management
Define specification of instr.
Conduct maintenance evaluation
Revise and refine program
Maintain system
Define educational philosophy
Define learning theory
Define instructional theory
ANALYSIS
MAINTENANCE

Figure 2: ISD^4 [Tennyson 1994, p 4]

- *Chen and Shen's model*: Chen and Shen [Chen 1989] propose a life cycle model similar to the waterfall model but aimed specifically at the development of CAI with the joint objective producing high quality products and development effectiveness. Verification and revision occur after each phase resulting in an iterative, cyclic process, as illustrated in Figure 1.

- *Tennyson's ISD^4*: Tennyson's Fourth Generation Instructional Systems Design Model (ISD^4) [Tennyson 1994], as illustrated in Figure 2, advocates the employment of advancements from cognitive science and intelligent programming techniques to automate instructional system development. It focuses on explicit rules in terms of development activities rather than on the sequencing of phases.

Development models

Detailed software process models are still the subject of research, but a number of general models or paradigms of software development can be identified as supporting these process models [Dix 1993, Sommerville 1992]. Two of these approaches have been widely used:

- *Exploratory programming* (also known as evolutionary prototyping): A working system is developed as quickly as possible and then modified until it performs in an adequate way. This approach is used to a great extent in artificial intelligence systems development where a detailed requirements specification cannot be formulated, and where adequacy rather than correctness is the aim of the systems designers. The disadvantage of this approach relates to the encapsulation of design decisions. Firstly, some of the earlier decisions may have been wrong and may never be removed from the system. Secondly, because the behaviour of an interactive system is highly dependent on the knowledge modelled within the system, earlier versions of the system will not include all of the knowledge to be included in the completed system.

- *Throw-away prototyping*: The software process starts off in a similar way to exploratory programming in that the first phase of development involves the development of a program for user experiment. The objective of the development is, however, to establish system requirements. The prototyping process is followed by re-implementation of the software to produce the desired software system. Suffering from the same drawbacks as exploratory programming, it has a further disadvantage in that it may concentrate only on the surface features of the design, rather than on deeper issues and the functioning of the interface, and does not on its own guarantee that the software produced exhibits the required interaction qualities.

To compensate for the limitations of the prototyping approaches, and to support the waterfall and spiral models in the requirements and design activity phases, a number of researchers in recent years have advocated the use of two more approaches in the design process of interactive systems: formal transformation and system assembly from reusable components.

- *Formal transformation*: A formal or abstract specification of the software system is developed and then transformed, by means of correctness preserving transformations, to an implemented software system. The principal value of using formal specification techniques in the software development process is that it compels an analysis of the system requirements at an early stage. Correcting errors at this stage of development is much cheaper than modifying a delivered system. A range of abstract modelling approaches for interactive systems are reported in [Abowd 1990, Dearden 1995, Dix 1991, Harrison 1990, Palanque 1995, Paternó 1995, Sommerville 1992].

  The formality of these models is intended to assure the exploration of the consequences of the design without constructing prototypes or other working models of the design, as well as algorithmic manipulation of the design.

- *System assembly from reusable components*: The system development process is either a total reuse process using components which already exist in assembling the new system, or available components applicable to the envisaged system are reused while additional components are developed using any other development approach.

  A trait of an engineering discipline is that it is founded upon an approach to system design which makes maximum use of existing components. Design engineers base their designs on components which are common with, and tried and tested in other systems of the same, or similar, nature. A number of reuse approaches are reported in [Biggerstaff 1989, Johnson 1988, Tracz 1988].

Notwithstanding the disadvantages of the prototyping approaches, they are still the most viable options for the development of interactive CAI systems. The reason for this is firstly determined by the high interactivity that characterises instructional software in relation to the computer literacy levels of the primary developers of such software, as well as that of the end-users of the completed product. The intended end-users are frequently laymen with respect to computer usage and unless they are at ease with the human-computer interface, the software will do little to achieve its instructional and learning ends. Prototypes are instrumental in pilot-testing by such end-users. Prototyping also allow for early evaluation by instructors, trainers, teachers, subject-matter experts, peers, etc. Visual perception and hands-on experience of part of an operational system often results in the instructor-client modifying the objectives and strategies. Also, the developers of such system are, more often than not, novice computer users, with absolutely no background in formal specification methods. The use of formal transformation techniques would therefore be totally inappropriate.

The major purposes of conventional software are data processing and information processing, where defined activities occur in a predefined sequence. Instructional software, by contrast, comprises synthesis, presentation, practice and assistance facilities in the complex realm of human cognition, and has a high level of human-computer interactivity. Whether in a situation of program-control where the flow is branched deterministically according to user-response, or in a situation of user-control where the learner may branch or browse at will, the sequence of events and activities varies greatly. CAI prototypes can play a vital role in demonstrating proposals on-screen, thus clarifying actual requirements and identifying misconceptions and potential errors at an early stage. Not only should basic aspects such as screen layout and colours be scrutinised, but also the strategies for control and navigation through the material. Usability factors, such as learnability and consistency can be evaluated, also interface aspects such as coherence of textual and visual displays, and accessibility of facilities.

Particularly when a piece of instructional software breaks new ground, prototyping is required at the design

and programming stages in order to ensure feasibility of intentions, to refine requirements, to reduce excessive written descriptions, to determine the optimal navigation and control strategies hands-on, and also to ensure that an appropriate programming approach is used for implementation.

Development tools or authoring environments should offer modularity, thus facilitating the removal, addition or adaptation of a segment without affecting other segments or the unit as a whole, and plasticity, the ability to make changes easily. If exorbitant time and costs are incurred in developing a prototype, the process is not cost-effective. An ISD prototype may either be evolutionary, i.e. a limited version of the final product later developed through to full functionality, or else a throwaway. In the latter case, the software used to build the prototype may not be the same as that used for the final system.

Furthermore, although envisaged by many, there is still no common base of reusable components of CAI software which is widely documented and which can be used when developing a CAI system with similar functionality. It is also interesting to note that prototype production forms a central part of Chen and Shen's life cycle model, and is also listed as one of the activities in the production 'phase' of ISD[4]. Other researchers advocating the use of prototyping approaches for ISD include Black and Hinton [Black 1988, Black 1989], Gray and Black [Gray 1994], Lantz [Lantz, no date] Tripp and Bichelmeyer [Tripp 1990], and Wong [Wong 1993].

Design models

Software design is in essence a problem-solving task. It is more important to design a solution that will achieve its purpose in doing the required job properly, than to achieve elegance and efficiency at the expense of accuracy and reliability. A designer needs to abstract the critical features of a system, so as to concentrate initially on building a logical model of the system rather than becoming over involved with detailed design and physical implementation at an early stage.

Various methodologies and representations are available to facilitate the processes of analysis, design, and system modelling, in particular, the process-oriented, data-oriented, and object-oriented approaches:

- The *process-oriented paradigm* centres around the events, procedures and flows that comprise a traditional procedural software system. Such applications are characterised by conventional data flow and updating of data stores. Events trigger processes, and processes call other processes, sequentially or selectively. The process-oriented approach is epitomised by concepts and tools such as top-down design, functional decomposition, transaction analysis, data-flow diagrams, structure charts and input-output transformations. It tends to discount evolutionary changes.

- *Data-oriented approaches* are based on the philosophy that data is more stable and unchanging than processes. The underlying principles are enterprise analysis and relational database theory, and key concepts are entities, attributes, relationships and normalisation.

- The latest emerging methodology is the *object-oriented paradigm* [Bell 1992, Booch 1994, Budgen 1994, Coad 1995, Conger 1994, Schach 1996], which integrates aspects of, and uses formalisms from, both the other major methodologies, and uses certain concepts from object-oriented programming languages. It is based on objects, which encapsulate both data and operations (processes) on that data. An object is a real-world entity whose processes and attributes are modelled in a computerised application. In object-oriented programming languages, computation is achieved when messages are passed to the objects in the program, and a central aspect is the abstract data type (ADT), which permits operations to be performed on an object without being implementation-specific. Objects incorporating data are identified as data entities and not as specific data structures.

Conventional data-flow and process-linkage do not feature heavily in instructional and learning software and such software therefore does not lend itself to the process-oriented paradigm. Object-oriented development has been used in large, complex systems, compared to which CAI courseware and environments comprise relatively few objects and components. Nevertheless, the strategies outlined can be beneficially applied in the analysis, design and development of instructional and learning software.

Budgen [Budgen 1994] describes an object as an entity which possesses a state, exhibits behaviour, and has a distinct identity. Sommerville [Sommerville 1992, p. 194] proposes the following definition: "An object is an entity which has a state (whose representation is hidden) and a defined set of operations which operate on that state. The state is represented as a set of object attributes".

Analysis of classical CAI tutorials, simulations, drill-and-practice software, and state-of-the-art user-controlled interactive learning environments reveals distinct design objects, or components, which possess unique identities, certain attributes and relationships, and have operations performed on them, i.e. much CAI software is explicitly, or implicitly, consisting of instructional components [Merrill 1988]. Many CAI systems are comprised mainly of instructional presentations and exercise/question segments. The main processing activities are determination of which unit / segment / example / exercise to present or do next, and the assessment of student responses. The means of determination depends on the locus of end-user control – whether program-control, learner-control, or a combination thereof. The various and varied instructional activities and learning experiences – comprising learning segments, example presentations, practice exercises and assessment activities – whether in textual or graphic form, whether requiring active learner-participation or passive perusal, can readily be perceived as objects. The objects are separate, yet strongly interrelated and it is appropriate to implement them in an object-oriented design. Such component-based instructional systems can best be implemented by an object-oriented design. The OO paradigm thus appears to be the most appropriate software engineering development methodology for ISD.

Even in the traditional life cycle models such as the waterfall model, the distinction between the systems design (broad design) and the program design (detailed design/coding) can become blurred. In the OO paradigm, however, the boundary is even more indistinct, because both top-down analysis and bottom-up program development occur simultaneously or, at least, iteratively. The three traditional activities of analysis, design, and implementation are all present, but the joints between are seamless. The unifying factor is the prime role played by objects and their interrelationships. Modelling is prominent in object-oriented design, the basic architecture being assembled from models of the entities and the relationships between them. Reuse is a feature of OO design, since the prominent class and inheritance features lend themselves to code reuse.

CAI software incorporates well-defined objects, both concrete and abstract, and particularly in situations with an initial lack of precise specifications, its procurement can be expedited and facilitated by a development process incorporating evolutionary prototyping. Combining the essence of prototyping and the OO paradigm requires a life cycle model emphasizing overlap and evolution with explicit incorporation of a prototyping phase. Chen and Shen's model, ISD[4], as well as other similar life cycle models, for example the Wong prototyping model [Wong 1993], the Booch model[Booch 1994] and the Henderson-Sellers and Edward's fountain model [Henderson 1990], all incorporate these requirements.

Conclusion

This article overviewed general software engineering models, tools and techniques, and investigated their applicability to instructional systems development.

A life cycle model which includes evolutionary prototyping appears to be most appropriate for the development of instructional software, so that initially fuzzy requirements can be refined and the initial working version can be modified and expanded towards a final operational CAI product.

The object-oriented methodology proves itself to be, in the terms of Korson & McGregor [Korson 1990], "a unifying paradigm", which is appropriate for the analysis and representation of CAI. Viewing a system as object-based provides a more versatile foundation than a view based fundamentally on data modelling or on its functions and procedures. Although user-input plays a major role in determining the path through instructional software, there is little conventional data flow. The concept of an object is a utilitarian approach, which brings together such varied items as concrete objects, abstract objects, data, processes, and environmental entities external to the software (yet vital components of the system), such as the human user. Incorporation of the user as an object is particularly beneficial in CAI, due to its highly interactive and individualised nature. The tools and representations of the OO paradigm can be of great value in the analysis, design and documentation of instructional software.

It is hoped that object-based control structures developed for specific applications can eventually be used

as *generic, content-free shells* to present formal instruction or practice exercises in different instructional modes in varying subjects and courses. This would capitalise on the modularity and reuse potential inherent in an object-oriented design.

References

[Abowd 1991] Abowd G D. 1991. *Formal Aspects of Human-Computer Interaction*. DPhil. Thesis, Oxford University, Programming Research Group.

[Bell 1992] Bell D, Morrey I & Pugh J. 1992. *Software Engineering: A Programming Approach* (2nd ed.). Hemel Hempstead: Prentice Hall International (UK) Ltd.

[Biggerstaff 1989] Biggerstaff T J & Perlis A J (Eds). *Software Reusability*. Volumes 1 & 2. Reading MA: Addison-Wesley.

[Black 1988] Black T R. 1988. Prototyping CAL courseware: a role for computer-shy subject experts. In: *Aspects of Educational Technology Vol XXI, Designing New Systems and Technologies for Learning*, edited by H Mathias, H Rushby & R Budgett. London: Kogan Page.

[Black 1989] Black T R & Hinton T. 1989. Courseware design methodology: the message from software engineering. In: *Aspects of Educational Technology Vol XXII, Promoting Learning*, edited by C Bell, J Davies & R Winders. London: Kogan Page.

[Boehm 1988] Boehm B W. 1988. A spiral model of software development and enhancement. *IEEE Computer*, **21**(5), 61 – 72..

[Booch 1994] Booch G. 1994. *Object-Oriented Analysis and Design: with Applications* (2nd ed.) Redwood City, CA: Benjamin/Cummings Publishing Company, Inc.

[Budgen 1994] Budgen D. 1994. *Software Design*. Wokingham: Addison-Wesley.

[Chen 1989] Chen J W & Shen C. 1989. Software engineering: a new component for instructional software development. *Educational Technology*, **29**(9), 9 – 15.

[Coad 1995] Coad P, North D & Mayfield M. 1995. *Object Models: Strategies, Patterns & Applications*. Englewood Cliffs, NJ: Prentice Hall.

[Conger 1994] Conger S A. 1994. *The New Software Engineering*. Belmont, CA: Wadsworth Publishing Company.

[Dearden 1995] Dearden A M. 1995. *The use of Formal Models in the Design of Interactive Case Memory Systems*. DPhil Thesis, University of York (UK).

[Dix 1991] Dix A. 1991. *Formal Methods for Interactive Systems*. London: Academic Press.

[Dix 1993] Dix A, Finlay J, Abowd G & Beale R. *Human-Computer Interaction*. Hemel Hempstead: Prentice-Hall.

[Gray 1993] Gray D E & Black T R. 1994. Prototyping of computer-based training materials. *Computers in Education*, **22**(3), 251 – 256.

[Harrison 1990] Harrison M D & Thimbleby H (Eds). 1990. Formal Methods in Human-Computer Interaction. Cambridge: Cambridge University Press.

[Henderson 1990] Henderson-Sellers B & Edwards J M. 1990. The Object-Oriented Systems Life Cycle. *Communications of the ACM*, **33** (9), 142 – 159.

[Johnson 1988] Johnson R & Foote B. 1988. Designing reusable classes. *Object-Oriented Programming*, **1**(2), 22 – 35.

[Korson 1990] Korson T & McGregor J D. 1990. Understanding object-oriented: a unifying paradigm. *Communications of the ACM*, **33** (9), 40 – 60.

[Lantz no date] Lantz K E. (no date). *The Prototyping Methodology*. Englewood Cliffs, NJ: Prentice-Hall.

[Merrill 1988] Merrill M D. 1988. Applying component display theory to the design of courseware. In: *Instructional Designs for Microcomputer Courseware*, edited by D H Jonassen. Hillsdale, NJ: Lawrence Erlbaum Associates.

[Palanque 1995] Palanque P & Bastide R (Eds). 1995. *Proceedings of the Eurographics Workshop in Toulouse France, June 1995*. Wien: Springer-Verlag.

[Paternó 1995] Paternó F (Ed). 1995. *Interactive Systems: Design, Specification and Verification*. Berlin: Springer-Verlag.

[Schach 1996] Schach S R. 1996. *Classical and Object-Oriented Software Engineering* (3rd ed.). Boston, MA: Aksen Associates Inc. Publishers.

[Shneiderman 1992] Shneiderman B. 1992. *Designing the User Interface*. Reading, MA: Addison-Wesley.

[Sommerville 1992] Sommerville I. 1992. *Software Engineering* (4th ed.). Wokingham: Addison-Wesley Publishing Company.

[Tennyson 1994] Tennyson R D. 1994. Knowledge base for automated instructional system development. In: *Automating Instructional Design, Development, and Delivery*, edited by R D Tennyson. Berlin: Springer Verlag.

[Tracz 1988] Tracz W (Ed). 1988. *Software Reuse: Emerging Technology*. Washington DC: IEEE Computer Society Press.

[Tripp 1990] Tripp S D & Bichelmeyer B. 1990. Rapid prototyping: an alternative instructional design strategy. *Educational Technology, Research and Development*, **38**(1), 31 – 44.

[Wong 1993] Wong S C. 1993. Quick prototyping of educational software: an object-oriented approach. *Journal of Educational Technology Systems*, **22**(2), 155 – 172.

COBIE: AN INTEGRATED COBOL ENVIRONMENT

N.Pillay
Department of Financial Studies
Technikon Natal - Pietermaritzburg

Abstract

Development software is one of the courses that first year students studying towards a Diploma in Financial Studies enrol for. This course contains an appreciable amount of RMCOBOL programming. Past experience is indicative of the fact that students seem to have difficulty programming in RMCOBOL. This paper looks at some of the problems experienced by these students and describes COBIE, a **Cobol Integrate Environment**, developed to assist students overcome these problems. Furthermore the paper provides a description of the students experience with COBIE. Attention is also given to the effect that a student's learning style may have on the difficulty experienced by a student in using COBIE. Finally ways in which COBIE can be extended to meet the needs of novice computer programmers is examined.

Introduction

As part of the course on software development first year Financial Studies Students are required to complete a course on COBOL programming. These students, majority of whom are novice computer users, experience much difficulty in programming in RMCOBOL. This paper firstly provides an account of the problems experienced by these students. It then goes on to illustrate how a phased development of a programming environment can possibly provide a means of overcoming the problems experienced by students.

On outline of each of the phases involved in the development of this programming environment is then provided. A detailed description of the first phase of the development of COBIE is given. This description firstly examines the facilities provided by COBIE and then gives an account of the students' experiences with COBIE.

Finally further extensions to the system in order to meet the needs of these novice programmers is reviewed.

Problems Encountered

The problems experienced by students are two-fold:

Problem 1 :

Students experienced difficulty in using the RMCOBOL system. In order to create, run and compile COBOL programs using RMCOBOL the following procedure had to be followed:

1. The user had to create the source and data files in an editor or wordprocessor of his or her choice.
2. The user had to then evoke the RMCOBOL Compiler at the command line.
3. Step 2 resulted in a compiler listing either being scrolled across the screen or being written to a list file if the user specified this option.
4. To view the list file the user would have to again use an editor or word processor of his or her choice.
5. Based on the listing described in steps 3 and 4 the user would then have to correct any errors listed by the compiler in the source code.
6. Finally the user would call the program RUNCOBOL at the DOS prompt to run the COBOL program.

As a result these students, being novice computer users experienced much difficulty in just creating, editing, compiling and running RMCOBOL programs. Students found the procedure that had to be followed to complete these tasks difficult to use and remember.

Problem 2:

Students experienced much difficulty and frustration in programming in COBOL. Barstow et al [BARS84] define this as a "complexity barrier" which hampers the productivity of novice programmers. According to Barstow et al this barrier can best be broken by building large programming support systems called **programming environments**.

A possible solution to **Problem 1** is for students to use **RMCOSTAR** which is a RMCOBOL project manager and editor from which COBOL programs can be edited, compiled, run and debugged. Reasons for not using **RMCOSTAR** include the following:

- RMCOSTAR is generally not sufficiently user friendly for novice computer users. For example files are stored according to projects which can lead to these confusion.
- It would not be possible to extend the RMCOSTAR system in order to provide a solution to the difficulties specified in **Problem 2**.

Proposed Solution

According to Teitelman et al [BARS84] a programming environment that is "cooperative and helpful" will prevent a novice programmer from spending most of his or her "time and energy fighting a system that at times seems bent on frustrating his best efforts". Hence based on the literature surveyed it was decided that a possible solution to the **Problem 2** defined above would be to develop a programming environment which these novice programmers would be "comfortable" using. Examples of such programming environments already in existence include Cedar which was developed from the Mesa environment, Pecan, Pict, and the Cornell Program Synthesiser just to name a few.

The foundation of the programming environment to be developed would be an integrated environment which would help users overcome the difficulties specified in **Problem 1**.

According to Teitelman et al [BARS84] the contribution made by users in terms of suggestions of improvements that should be made to Interlisp and facilities that should be provided by the Interlisp Programming Environment have proven invaluable. Hence throughout this study much emphasis will be placed on user feedback.

The development of COBIE will be undertaken in three phases:

Phase 1:

1. The development of an integrated environment that enables the user to edit, compile, and run COBOL programs in an attempt to provide a solution to **Problem 1**.
2. Evaluation of this environment.

Phase 2:

1. During this phase improvements will be made to COBIE based on the evaluation in Phase 1.
2. According to Barstow et al [BARS84] a programming environment must combine the powers of

a compiler, an editor, a debugging system, a documentation system and a problem solver. COBIE will be extended to provided a debugger and documentation. During the process of debugging users will be able to make necessary changes to the source code.

3. Evaluation of the extended system.

Phase 3:

1. Improvements made to the system based on the evaluation in **Phase 2**.
2. Additional mechanisms need to be added to COBIE to assist students to program in COBOL. The following mechanisms will be examined in order to extend COBIE to a environment in which these novice programmers are "comfortable" programming in:

- In their description of an integrated Prolog Programming Environment Schreweis et al [SCHR93] emphasise that an "UNDO" option should be available to users. The importance of an "UNDO" option is still further emphasised in Teitelman et al's description of the Interlisp programming environment [BARS84].
- An automatic error correction facility:

  Teitelman et al in their paper " Automated Programmering: The Programmer's Assistant" [BARS84] describe DWIM a mechanism for automatically correcting trivial errors made by programmers such as spelling mistakes and punctuation errors. The importance of such a facility is further stressed by the developers of the Interlisp environment.
- Barstow et al [BARS84] emphasis that "In writing a program, the use should not need to be continually concerned with the exact form of the structures being used."

  This idea forms the basis of the Cornell Program Synthesizer (CPS). CPS is a programming environments that provides users with templates of the grammars of each simplified statement of a programming language. All the user has to do is provide the arguments or expressions at a cursor position in these templates. The use of such COBOL templates in COBIE will be looked at.
- Question and answering:

  Barstow et al [BARS94] describe a question and answering technique to help uses identify the effects of making certain changes in their programs prior to making these changes.
- An interactive browser to display various view of the current state of computation of a program.
- According to Schreweis [SCHR93] " The human mind is strongly visual oriented and acquires information at a significantly higher rate by discovery of graphical relationships in complex pictures than by reading text". The significance of the use of visual environments is further stressed by Shu [GLIN90] who states that pictures are more powerful than words as a means of communication.

  An example of a visual environment is PECAN. Shu describes PECAN as a system which supports multiple views of a user's program. These views include a syntax directed editor, a Nassi-Schneiderman flowchart, a module interconnection diagram of how the program is organised, and stack data view showing the current state of the data stack.

  Other examples of visual environments include Pict, PIGS and Xerox Star System. The effect of COBIE being extended to a visual environment based on the above examples will be examined.

Various monitoring mechanisms similar to those described by Barstow et al [BARS84] to keep track of statistics regarding the utilization of the facilities provided by COBIE at each stage in this phase will be built into the system.

Cobol Integrated Environment (COBIE)

The integrated environment developed to meet the objectives of Phase 1 described above was created using Turbo Vision and is similar to the Integrated Development Environment (IDE) provided by Turbo Pascal.

System Specifications

COBIE, like RMCOBOL, is a DOS based system which can be run on any IBM compatible microcomputer.

COBIE Interface

The COBIE interface consists of three components namely a pull down menu, a status bar and a desktop.

According to Kay [GLIN90] multiple windows that have the following characteristics:

- displays associated with several user tasks which could be viewed simultaneously;
- switching between tasks must be done with the press of a button;
- no information will be lost in the process of switching; and
- screen space would be used economically;

formed the basis of an "integrated environment. COBIE provides users with this multiple window facility in which to implement the necessary tasks required.

Facilities Offered by COBIE

The pull down menu which forms part of the COBIE interface is comprised of three submenus namely **File**, **Cobol**, and **View**.

The **File** submenu provides the user with the option of creating new source and data files or editing existing files, and saving these files.

The **Cobol** submenu provides the user with a **Compile** and **Run** option. Upon choosing the **Compile** option the user is prompted to confirm the name of the file to be compiled. The RMCOBOL compiler is then evoked. Thereafter the user's screen is divided into two windows:

- one containing the source program; this window contains a line number indicator to assist the user find any errors indicated in the compiler listing;
- one containing the listing produced by the compiler.

The user can switch between the two windows and hence correct errors specified in the compiler listing in the source code. The **Run** option results in the RMCOBOL program **RUNCOBOL** being implemented. Upon choosing the **Run** option the user is required to confirm the name of the COBOL file to be run.

The **View** submenu provides the user with the options of viewing a particular compiler listing or a report file that has been created as the result of running a specific COBOL program.

Evaluation of the COBIE System

"Hands-on" sessions with COBIE were held during which students were required to firstly make minor changes to two COBOL programs and then compile and run these programs. In order to evaluate the COBIE environment at the end of these sessions students were issued with questionnaire. Eighteen students of the twenty two enrolled for the course completed and returned the questionnaire. This questionnaire was designed based on that used by Glinert et al [GLIN90] in their student evaluation of the Pict system. A list of the questions contained in the questionnaire are listed in **Table 1**.

Table 1 : Evaluation Questionnaire

1.	Have you found the program easy to use? Explain
2.	Could you easily remember the steps that were required to carry the necessary tasks?
3.	Did you find the program fun to use?
4.	Would you like to use the program again?
5.	What improvements do you think should be made to the program?
6.	List any problems that you have experienced.
7.	Describe **your** experience of using the program.
8.	Any other comments

To obtain a measure of the effect of students' learning styles prior to the use of the system a survey was conducted using Kolb's Inventory to determine the learning styles of these students. Nineteen of the twenty two students enrolled for the course completed and returned the questionnaires.

In order to obtain a correlation between a student's learning style and his or her experience with COBIE only the feedback of those students that responded to both questionnaires was examined. Sixteen students responded to both questionnaires of which nine were female and seven male.

Responses to questionnaire

Table 2 lists the responses to the first four questions:

Table 2: Responses to questionnaire

QUESTION NUMBER	YES	NO	UNSURE	NO RESPONSE
1	9	7	-	-
2	12	2	2	-
3	10	5	-	1
4	15	-	-	1

From **Table 2** it is clear that a majority of the students found the COBIE system easy to use. Comments made by students included:

"The program eliminated time wasting and irritating procedures."

"Steps were not difficult."

"Easy to grasp, understand and remember."

However seven students did not find the program easy to use. Comments made by these students included "as time goes on it will be easier"; "Not easy - because it was my first time to use it".
This is consistent with responses obtained by Glinert et al [GLIN90] in the evaluation of the Pict system. Students attributed the lack of ease of use of the Pict system to the fact that it was the first time that they were using the system.

It is evident from **Table 2** that again a majority of students found that they could easily remember the steps that were required to edit, compile and run COBOL programs using COBIE.

All students with the exception of one student who did not respond to this question indicated that they definitely wanted to use the COBIE system again.

Two problems were listed by students:

1. Firstly due to COBIE being used in a network environment students were prompted to "Insert a disk in drive A:" four times during the compilation process. This made compiling of a program rather tedious and frustrating.

2. Secondly if a compiler listing of a source program could not be created a blank window was displayed on the screen by COBIE and this window could not be "closed".

Improvements mentioned by students were based on these two problems experienced.

Comments describing the students experience with COBIE included:

"It was most educational."

" I found it interesting and fun to use. The steps were easy to remember, however I experienced some difficulties when trying to compile my program."

"Challenging"

"At the moment the program is fine because I understand what we've done at this point in time."

Only five students provided any additional comments. This comments emphasised that students required more practical sessions with COBIE.

The Effect of Learning Styles on the ease of use of COBIE

Finnie [FINN90] defines four learning styles namely, Converger, Diverger, Assimilator and Accommodator as derived by Kolb. From the study conducted it was found that students who responded positively to the questions "Have you found the program easy to use?" and "Could you easily remember the steps that were required to carry out the necessary tasks?" predominately possessed a the Converger or Accommodator learning style. Those students that responded negatively to these questions predominately possessed either a Diverger or Assimilator learning style. Hence it is evident that students possessing a Converger or Accomodator learning style experience less difficulties in using the integrated COBOL environment compared to students possessing a Diverger or Assimilator learning style.

Conclusion

From the study conducted it is evident that the COBIE system has helped novice programmers overcome some of the difficulties experienced with using the RMCOBOL system. Certain improvements have been made to COBIE based on the feedback obtained from. students. Currently the COBIE system is being used by first year students to write COBOL programs.

COBIE will be extended further to provide a debugging and documentation system. Finally a number of

mechanisms (as outlined above) such as the use of syntactical templates and visual programming environments will be examined in order to obtain a means by which these novice programmers can be assisted to program in COBOL.

References

[BARS84]	-	Interactive Programming Environments, David R. Barston, Howard E. Shrobe, Erik Sandewill, Mcgraw Hill, 1984.
[DONA85]	-	Integration Techniques in Cedar, James Donahue, Sigplan Notices, Vol 20, No 7, July 1985, Pages 265 - 286.
[FINN90]	-	On Learning Styles and Novice Computer Use, G.R. Finnie, Quaestiones Informaticae, Vol 5., No. 1, 1990, Pages 1- 10.
[GLIN90]	-	Visual programming Environments - Paradigms and Systems, E.P. Glinert, IEEE Computer Society Press, 1990.
[SCHR93]	-	An Integrated Prolog Programming Environment, U. Schreiweis, A Keune, H. Langendorfer, ACM Sigplan Notices, Vol. 28, Part 2, 1993.
[SWEE85]	-	The Mesa Programming Environment, R.E. Sweet, Sigplan Notices, Vol. 20, No.7, July 1985, Pages 216-229.

THE DESIGN AND USAGE OF A NEW SOUTHERN AFRICAN INFORMATION SYSTEMS TEXTBOOK.

G.J.Erwin[1]
and
C.N. Blewett[2]

[1] Business Information Systems, Faculty of Commerce and Administration, Department of Accountancy, University of Durban-Westville, Private Bag X54001, Durban, 4000. Phone: (W) (031) 820-2435 Fax: (W) (031) 820-2429 e-mail: erwin@is.udw.ac.za

[2] Business Information Systems, Faculty of Economics and Management, Department of Accounting and Finance, King George V Ave., Durban, 4001. Phone: (W) (031) 260-2161 Fax: (031) 260-3292 e-mail: blewett@bis.und.ac.za

Abstract

Changes in South African society have brought about changes in the background of student audiences and these have stimulated a review of the areas of governance, syllabus, curriculum, teaching methods and research in educational institutions. In this paper we describe our revised approach to teaching of (Business) Information Systems/ Computer Studies / Business Computing, and the first-year textbook which we created to support this new approach. We discuss the "product-driven" and "problem-driven" arrangements of current textbooks and report on a survey of Business Information Systems students using the new textbook at the University of Durban-Westville. We discuss problems with foreign textbooks and current teaching methods and describe the structure and content of an Information Systems course designed locally. We conclude with recommendations on teaching methods, textbook design and support material which are appropriate to the new South Africa.

Keywords: textbook design, education, Information Systems

Computing review categories: K.3.2

Introduction and Background

"There are ever-fewer white students and the trickle of Africans has become a flood. Universities (*et al*, added by authors) must adapt to the needs of a new and very different clientele." (Moulder, 1994).

The student audiences at, for example, University and Technikon, in South Africa are rapidly changing. Moulder discusses how the change in student audiences requires us to:

1. alter the "composition of student, academic and ..... administrative bodies"; the *governance* structures of our institutions.
2. change the *syllabus* by moving away from "geriatric, northern hemisphere cultures".
3. change the *curriculum*, "the whole way in which teaching and learning are organised".

and

4. change the "criteria (for) *research*".

In other words, Moulder is summarising the areas of attention for the transformation of our educational efforts.

This paper describes work we have done to address the *syllabus* issue, with our new textbook (Erwin and Blewett, 1996), and the *curriculum* issue, by reshaping the presentation of Information Systems to be *problem-driven* , items 2 and 3 above. We begin by describing the design of current Information Systems textbooks, and the problems which students face using foreign textbooks. We then present the features of our new Information Systems textbook which address those problems, discuss the problem-driven approach to teaching and learning, compared with the product-driven approach, and, after reporting on a survey of Information Systems students using our new textbook, draw some conclusions about the success of the textbook and the problem-driven approach. These issues also apply in business, and at primary/ secondary schools, but, this paper does not address those areas, except in passing.

Many new students will be and are educationally disadvantaged, under-prepared and unprepared. These students come from very different backgrounds to previous entrants to our tertiary institutions. The growing penetration of sectors of our society by Information Technology (IT) means that students will encounter courses in subjects such as Business Information Systems (BIS) and Computer Studies. Many of these students "are studying in a foreign language" (Moulder, 1994). Teaching methods and materials are under review by many educational institutions.

Some Notes on Current Textbooks and Student Learning

Currently prescribed textbooks for Information Systems / Computer Studies courses at Universities, Technikons, Technical Colleges and other educational institutions often bear little specific relevance to South African society. They are illustrated with foreign-based examples, lack reinforcement of the material suited to educationally disadvantaged readers, and have inappropriate "chunking" and sequencing of material. Such textbooks can create learning problems for many students. Our teaching experience shows that current American/British textbooks assume certain experiences and institutions in society, leaving our students confused between fundamental material and the mechanisms of a particular society's structure. Furthermore, American/British textbooks present material in a sequence which is much more for the author's and course leader's convenience, than for the student's. Textbooks are often designed for the approval of other academics, rather than to improve the learning process of students. Academics / course instructors are already familiar with the material in a textbook. Their main concern when assessing a textbook is often the *organisation* of the material therein, rather than the *impact of the material* on the learning student. This approach does not recognise the difficulties students have with material on first meeting it.

The success of a student's learning process depends, *inter alia*, on the method of *initial exposure* to the material and progress through it without jolts or obstacles. Subsequently, after first-level comfort is achieved, there will be an (overlapping) *formal learning* stage in which a student returns to the material, for example, for examination purposes. There is a two-stage process, with much overlapping of stages, as new topics are encountered. Every topic has an initial stage, followed by a more formal stage. The initial learning stage significantly influences the outcome of the more formal learning stage. Existing textbooks have been arranged largely to suit the more formal type of student learning which occurs when preparing for exams / tests, not the *initial learning* that occurs when a student first encounters a topic.

Production of a New Textbook

We believe serious learning problems emanate from the use of inappropriate textbooks, (and other media, methods and material) and that these problems will grow substantially, unless addressed. As in any situation, there are three major options available (Erwin and Blewett, 1996): namely, *do-nothing*, *improve existing methods*, or *adopt new methods*. In the early 1990s, we observed the mounting difficulties our students were having with IS material and teaching methods, and we decided we could not do-nothing. We could attempt to improve our existing methods and material, by piecemeal attention to many different areas, or we could go "back to the drawing board" and address the problems we were facing by building new materials and methods. Therefore, after some months of planning and discussion with colleagues and others, we wrote and published a *new type of first-year Business Computing / Information Systems textbook* (Erwin and Blewett, 1996) to reduce learning problems associated with foreign textbooks. We incorporated our new textbook into an Information Systems course using problem-driven methods. In these two ways we hoped to introduce material better suited to our student audience, and offer better teaching support as our new student audience emerges. In (Moulder, 1994)'s phrase, we are "Riding the Wave", not being swamped by it.

Target Audience

We aim our textbook at 1st. Year University, Technikon, Technical College students of Business Information Systems (BIS), Information Systems (IS), Computer Studies, Information Technology, Computer Management, Business Computer Science, Business Data Processing (BDP), Commercial Computer Science, Management Information Systems, and so on. The book also caters for business people, with little or no computer experience, who wish to improve their computer literacy and want to understand the process of computerisation in a business.

Objectives of a Textbook

A (first-year) textbook can be constructed for one or more of a variety of objectives. These objectives include:

* to cover the field
* to be up-to-date with latest research findings
* to expose principles
* to provide drill and practice
* to encourage interest in the subject
* to cover a specific set of topics, often as a basis for professional work
* to introduce core topics in a comfortable style
* to provide a variety of (edited) sources for student growth

and so on.

We believe that, at first year (and novice) level, the appropriate objective of an IS textbook is a mixture of "expose principles", "encourage interest in the subject", "cover a set of topics, ... as a basis for professional work" and "introduce core topics in a comfortable style".

Let's look at some of the problems we recognised with existing available textbooks and teaching methods, and the approach we took to address those problems.

Summary of Existing Problems

Here is a summary of major problems we encountered with existing textbooks. We discuss these problems and our "solutions" in the sections that follow.

Existing Business Computing / Information Systems textbooks are:

*	foreign:	aimed at "literate", non-South African, audience
*	product-driven:	describe how? Often to the exclusion of why and when?
*	glossy:	distracting and unreal photo galleries; sophisticated applications
*	superficial:	not useful for the practice of business; stop at first level, even for core topics
*	boring/dense:	not read and learnt; use passive voice; assume good literacy; back-end Appendices.
*	comprehensive:	distracting wide coverage of most-recent software/hardware versions
*	up-to-date:	un-necessarily so, to "prove" usefulness to a literate audience as an up-to-date text
*	targeted:	at the instructor (his preferred chunks of knowledge), not the student

Existing textbooks have:

* good coverage of required material
* emphasis on PCs without ignoring other types of computers
* well-organised exercises and summary material

However, in our South African situation, existing textbooks suffer from :

* **inappropriate context**
* **inappropriate examples**
* **lack of reinforcement**
* **inappropriate "chunking" and sequencing of material**
* **inappropriate use of language**
* **poor placement of Appendices**
and
* **an unrealistic element.**

We now consider each of those problems in turn, with the use of examples, and, after each discussion, immediately discuss our approach to each problem.

Problem: Inappropriate Context

Existing textbooks describe the society and events of a first world society, rather than our mixed first/third world society. For example, an (American) textbook on Computer Security starts off by claiming "You seldom hear of bank robberies these days!" (Pfleeger, 1989, p.1). American society has largely moved to the use of credit cards and other forms of cashless transactions. This reduces the availability of cash as a target for robbers. In South Africa, bank and building society robberies are so common that we hardly notice them anymore!

Almost all American textbooks refer to USA laws, such as The Freedom of Information Act: a law giving citizens the right to have access to data about them gathered by federal agencies; Fair Credit Reporting Act; Federal Privacy Act and so on. For example, "The major piece of legislation on privacy is the **Privacy Act of 1974 (PA74)**, enacted by President Gerald Ford." (Stair, 1992, p. 632). (Stair, 1992) is an American textbook, so, our relatively young, local, novice reader can make little sense of this sentence in South Africa, where the 'President' is currently Nelson Mandela. This example from Stair confusingly refers our students to a *foreign history* and *foreign legislation* at an *early stage* in the learning of Information Systems "Principles". South African students using Stair will learn about the issues of privacy in the confusing context of unfamiliar and (largely) irrelevant Government institutions and regulations. Material of this type is important in order to expose legal aspects of the issues of privacy and security, but the context in which such textbooks do this makes the topic very difficult for novice students. See (O'Leary and Williams, 1985, p. 543) for a similar example. Our textbook, on the other hand, as one example, refers to South African issues, such as "The government may introduce new laws which require strict control of the business, such as the calculation and claiming of **Value Added Tax (VAT)**." (Erwin and Blewett, 1996, p. 21).

Another example of inappropriate context (for South African students) is "... a new type of robot developed at the MIT Mobile Robotics Laboratory in Cambridge, Massachusets." (Stair, 1992, p. 354). There are many blockages for our students here, including the "MIT", the "Cambridge", and the "Massachusetts". A similar inappropriate (for South African students) example occurs in (Stair, 1992, p. 217), namely, "Researchers at AT & T Bell Labs ...". Our textbook, on the other hand, uses an example such as "For example, ... rainfall figures in Cape Town help to plan the construction of a new dam." (Erwin and Blewett, 1996, p. 212).

Existing textbooks also assume wide availability and accessibility of computer equipment. For example, one textbook offers a "Teaching Tip : Take off the enclosure on a microcomputer processor to expose the circuit boards. Point out the motherboard, memory, add-on boards, disk drives, and so on." (Long and Long, 1990, p. 60). This is not only unrealistic with Information Systems class sizes of hundreds, but is normally *forbidden* by institutions, and is impossible for distance teaching.

In general, the level of computer literacy, availability and personal, computer comfort in South Africa seems far lower than in the USA and UK. The content and style of a textbook needs to recognise the characteristics of its target readership, usage and society.

Inappropriate Examples

Our experience at teaching in University, Technikon, Damelin Management and Computer Schools, industry and commerce, and other institutions has shown us problems when students learn with inappropriate textbooks and examples. In textbooks, such as (Capron, 1990), the beautifully printed picture galleries, glossy print and USA examples immediately present credibility problems for our students. The examples in books such as Capron, become a distant, almost dream-like fantasy collection when read by many of our students. Capron-type examples simply convince many students of the unmanageable sophistication and complexity of computerised applications, reducing their own confidence in being successful with computers. A confident, identifiable *start to learning* is needed. Further texts and/or study can subsequently introduce complexity and breadth.

Another example of an inappropriate example appears in (O'Brien, 1988, p.21). The illustration (photo) for an optical scanning "wand" is of a mysterious product called "Dr Pepper". Only people who have travelled to the USA (or been very observant while watching American TV situation comedies) will know that "Dr Pepper" is a soft drink, and not a medicine or a brand of Mexican food.

A similar example appears in (Stair, 1992, p. 152), namely, "... which leads to the manager's SSN (098-40-1370) in the Department table." Many of our students have no idea what "SSN" is, so this example is much more a blockage to learning than an aid. An equivalent example in our textbook is "Another way to think of a byte is to consider *eight soccer referees*." (Erwin and Blewett, 1996, p. 256).

The examples used in our textbook are *local examples*. There are frequent references to Durban, Cape Town and other Southern African locations and issues. This removes the problem of students firstly having to comprehend an *example* before even attempting to understand the *material*. The examples in a first year broad-coverage textbook need to be *familiar* and *incremental* in nature. Examples should begin at a comfortable (but correct) level, then expand to incorporate other aspects, such as problems of size and geographic dispersion. Fundamental concepts should transfer from small scale operations (such as a retail store) into medium (such as a sporting club) and then into larger manufacturing and financial services businesses (such as a bank, or a motor car manufacturer).

Problem: Lack of Reinforcement

A common student complaint is that the course instructor delivers the course material too quickly. Other complaints include little reinforcement (reminders) of the material and examples are usually *big* business, first world orientated. Many textbooks are tightly edited to remove any repetition. Such removal of repetition causes difficulty for students who do not read the book sequentially. In our experience, *no student reads a book sequentially.* In fact, even *courses* are not usually presented in the same sequence as the textbook. We recognise this "non-sequential" attribute of our courses and repeat important material in several places. For example, we show the components of a transfer (Source, Destination, Channel and Protocol) in different parts of the book (Erwin and Blewett, 1996, p. 238, 268, 579, 588, 607). Each repetition is in a different situation and encourages students to see the application of fundamental material in different areas, by reinforcing the presence of a major principle or framework.

We saw that students often forgot the meaning of abbreviations. So, throughout the book, we explode abbreviations into their full wording. This means that a student will no longer confuse the Bank for International Settlements (BIS) with Business Information Systems (BIS). We also use acronyms liberally to assist memorisation of key points. For example, a student remembers the seven **Critical Computerisation Questions (CCQs)** (to assess whether a computerisation investigation should proceed) by using the acronym **CRITICAL**.

A textbook should reinforce other course material as well as *internally* reinforce the book itself. Ideally, the student also has his/her textbook material reinforced by events and institutions in his/her society *outside the class*, as well as within the educational institution. In existing textbooks, this aspect is very well handled for American (and other first world) students by reference to well-known institutions and practices in their society. Such books provide no reinforcement support of that type for our students in Africa. Our textbook refers to local institutions, such as Eskom and Telkom. In addition, our textbook provides "Interactions". Many Interactions require the student to visit different parts of society, such as a bank, soccer club, pharmacy, supermarket and so on. These visits include a questionnaire which helps the student to observe the meaning of words from the textbook *in the real world.*

Problem: Inappropriate "chunking" and Sequencing of Material

"Chunking" is the choice of which material goes into various chapters, episodes and sections. For example, Capron (1990) chunks a chapter on "Database Management Systems". This Chapter first describes what Database Management Systems are, followed by uses for a Database Management System. This sequence of exposure of the material is *product-driven.* A product-driven approach describes the *content of a product; what* it is, and *how* it works. Although product-driven sequence can be useful at a *later, more formal learning stage* for students, it does not properly assist them through the *initial*, *formative* learning stage. Our textbook recognises and caters for *both* stages of learning.

Our textbook has a *problem-driven* approach. A problem-driven approach describes a situation which, as it develops, requires the use of a technology, product, method or design which appropriately services the problem situation. The problem, once recognised, identified and measured, drives the use of the product for solving the problem. A problem-driven approach is also a *requirements-driven* approach. In our textbook, we introduce problems a business faces, such as stock control and payment of suppliers, then, may draw an analogy between existing manual methods and a computerised approach. Database Management Systems are shown to be an appropriate technology (product) for a certain class of problems, rather than a product to be described *per se*. The problem-driven approach is then supported by Structured Material to assist with learning core topics and the required detail for examinations and tests. Our textbook provides the proper chunking and sequencing of material to assist the student when learning concepts, and structured material for later use, such as in examinations. This approach addresses a common question from students, namely, "I know *what* it is and *how* to use it, but *why or when* do I use it?". Later, we discuss how the problem-driven approach drives the design of an Information Systems *course*, not just the textbook material.

Consider another example of the product-driven chunking compared with the problem-driven chunking. (Stair, 1992) includes "Chapter 3: Hardware: Input, Processing and Output Devices". This Chapter covers as a "chunk", the topics of Computer System Components, CPU, Primary and Secondary Storage, such as Tapes and Disks, Classifying Computers, and Input and Output Devices. Chapter 18 of Stair includes a section on "backup procedures" associated with "Information and Technology Management". This chunking and sequence of presentation is the *product-driven* approach, because it describes components and not applications within a business. Our *problem-driven* approach, covering the same material as Chapter 3 of (Stair, 1992), begins with a Story in "Episode C2: Computer Disaster Planning." This Story continues the events in the computerisation of a supermarket, owned and managed by Mr Makhathini, in Africa. As a result of a disaster which destroys Mr Makhathini's computer, the issue of backup procedures, and copies of data and software arises. Discussion of methods of backup follows, and then a description of magnetic tape principles occurs in the context of a problem to be solved. Now, in this Story, and because of the business problem, for the student, the calculation of the capacity of a magnetic tape becomes important and relevant, not just an exercise in arithmetic. Our textbook has *no single section* which describes hardware and computer components. The relevant descriptions occur in contexts in which such component descriptions are relevant and needed. A student learns about components and products at the time the information is needed. Descriptions also occur at several different levels of detail. There is a first level description, for assimilating the overall concept. Then, there is a more detailed description in a separate sub-section. This style recognises that a student needs to be comfortable at an overall level first, then, as problems arise the more detailed description is more comfortable and meaningful.

Our chunking arrangement and sequencing mean that students do not regard concepts as strictly compartmentalised. Concepts are introduced in an appropriate problem situation and often reappear in later Episodes. For example, the topics covered in (Stair, 1992, Chapter 3: Hardware: Input, Processing and Output Devices) need approx. 30 pages. (Erwin and Blewett, 1996) covers the same material as (Stair, 1992, Chapter 3) in over 100 pages, namely, Storage Devices, pp. 105-112; Principles of Disk Operation, pp. 211-228; Data Representation and the Machine Cycle, pp. 245-298; Input Devices, pp. 355-364; Output Devices, pp. 473-484, and Magnetic Tape Principles, pp. 563-574. The Stair chunking suits the purpose of arranging like-items together. The Erwin and Blewett chunking arranges material as a flowing, natural story, then, as problems and situations arise which need computer approaches, the textbook describes and explains the material to apply that technology. This chunking means that the introduction of hardware is part of problem solutions. For example, students learn about Output Devices in the context of Office Automation and Word Processing. They learn about Magnetic Tape Principles in the context of disaster and backup issues.

There is so much material available in the IS field that a student can become confused about his/her position within the material. To reduce this problem, we introduced a *uniform framework for every Episode* in the textbook. No other textbook known to us (in this field) has such a uniform framework in every Chapter / Episode. All other textbooks known to us in this field present each Chapter in a separate framework, or arrangement. Each Chapter's material is arranged in its own special way. Course instructors have no difficulty adjusting to different styles of presentation in each Chapter. Novice students struggle with such a scheme. Every **Episode** in our textbook has the following format / framework:

Story -> Transition -> Structured Material -> Infobyte

Every **Story** unfolds in the STAIR framework, namely:

Stimulus:	events or circumstances which lead to discussion of a business problem
Trouble:	exploration and identification of business problem(s) as a result of the Stimulus
Approach:	discussion of various approaches for resolving the identified business problem(s)
Implementation:	actions to install the chosen Approach
Review:	a look back at the Implementation and its success/failure.

The **Transition** after each Story is a summary of the Business Computing principles and events in the Story, and an anticipation of the formal treatment of material to follow in the Structured Material.

The Structured Material follows the Transition. **Structured Material** is in the **KAIR** format. namely:

Knowledge:	facts and techniques
Awareness:	importance, role, potential and widespread usage
Interaction:	practical exercises, direct touch, visits to the real-world
Reality:	business implementation issues.

The **KAIR** elements make up the four parts of **Computer Literacy and Competency**.

To further provide the student/reader with a 'road map' through the material in the book, the **Knowledge** part of **KAIR** in Structured Material is split into **COURSE** elements, namely:

Complaint:	opportunities and/or difficulties which lead to a business investigating a specific topic
Overview:	the mainstream content of the topic
Usage:	how to use this technology
Resources:	resources associated with the use of this technology
Strategy:	options and approaches
Examples:	illustrations of the use of this technology in various levels of business

After each Structured Material section is an **Infobyte**. An Infobyte provides background, and further detail on Business Computing topics. We aim Infobytes at students requiring more than a first-level understanding of a Business Computing topic. Typically, this will be an Information Systems major student, an Information Systems support person and, of course, an Information Systems developer. For example, Infobyte I3 appears at the end of Episode A3 and covers Data Representation in a Computer and the Machine Cycle.

The uniform framework in every Episode means that a student *can learn about various topics in a standard way*, and begin to see the common aspects of topics such as Database and Spreadsheets. To further assist the student, we also devised **a framework for learning about the usage of various packages**, components, and so on. This is the **SKRAP** framework, namely:

Setup:	actions to establish the initial use of a package, data and data definitions for an application
Keep:	actions to preserve /save / keep work / data / software established in the application
Retrieve:	actions to retrieve / find / call back for use, the data / definitions in an application
Alter:	actions to alter / amend / change data and definitions in the application
Print:	actions to list / inspect / print / display / query data and definitions in the application.

This SKRAP framework considerably assists students when approaching a new software package or application. *The SKRAP actions have to be supplied by every package / application*. No application can exist without providing the SKRAP actions. The student can now search a way to perform each SKRAP action, and, when (s)she has conquered SKRAP, (s)he will be able to move on to more complex variations of those basic actions. Initial comfortable usage comes from pursuing SKRAP.

The book has three major **Levels** arranged in Sections: A, B and C.

Level A contains **Fundamentals:** computers, hardware, software, internal computer principles, business requirements, computer selection, and so on. Some detailed Fundamental material appears elsewhere. For example, Magnetic Tape Principles appear in Infobyte I8 at the end of Episode C2 in Level C.
Level B covers major **Applications** areas; *Database*, *Spreadsheet* (DSS) and *Office Automation*.
Level C covers **Implications** of using computerised Information Systems, such as backup, disaster recovery planning, networks, Information Systems Development (ISD), package software, and so on.

In conjunction with Infobytes, a course instructor or a novice reader can choose a selection of material to suit various depths and lengths of courses of study. The repetition of some material at different Levels and within Levels assists such a customised choice.

Problem: Inappropriate Use of Language

Every subject has its own jargon and specific terminology. Any course in the subject includes the acquisition of this terminology. However, when a textbook uses difficult, hard-to-read language a student has understanding problems. Many textbooks use *passive voice* to expose material. Passive voice is an attribute of "verbs in sentences in which the grammatical subject is the recipient of the action described by the verb" (Hanks, 1988, p. 833), as in 'Total revenue from soccer tickets is calculated by the computer application.'. *Active voice* is an attribute of "verbs used to indicate that the subject of a sentence is performing the action or causing the event or process" (Hanks, 1988, p. 11), as in 'The computer application calculates total revenue from soccer tickets.' Our textbook almost exclusively uses *active* voice, because active voice presents concepts in a more 'natural' way and contributes to the easy-to-read, soft language, attribute of a textbook. This style can sometimes appear to make the content elementary, because the wording is so simple. However, our treatment of material is usually deeper than American and Briish textbooks.

Material in existing textbooks is often dense and boring with long passages of unrelieved text. Many students find that a textbook presents material in a particular way, and then moves on to the next topic. Our experience is that many students understand material better when we present it to them *in several different ways*. That seems to be one of the main objectives of lectures. However, in our textbook, we introduced a unique concept, known as the *Wise Guru*. This 'person' is a device to break the text into smaller sections by looking back on material and describing its contents *in a different writing style, different font,* and with *different words*. The *Wise Guru* also points out important material and common topics for examination questions to students.

Because we live and teach in Africa, we were concerned to produce an African-flavoured textbook which recognised both our strengths and weaknesses. We wanted to present the student with a 'comfortable, but correct' introduction to fundamental concepts. To do this we divided the book into Episodes, rather than Chapters, and wrote a series of Stories. Every Episode begins with a Story. We wrote the stories in conversational, novel-style English. Read as a whole throughout the book, the Stories cover the first year of computerisation for Mr Makhathini's supermarket business. The Story format, and language, enable us to lead the student into important issues in an interesting way. The Stories contain drama, action, discussion, human difficulties and humour. These Stories allow comfortable assimilation of concepts at *initial* learning stage and are excellent pre-reading for lectures, tutorials and discussion groups. Plans are underway to offer these Stories, and the Structured Material, as a series of video Episodes for presentation to classes, on educational TV, and so on. This medium can assist course instructors with presentation of material.

Problem: The Placement of Appendices

Information Systems textbooks often contain Appendices at the end of the book with detailed, off-the-mainstream topics. Our experience is that students often regard such Appendices as unimportant. In our textbook, we introduced the concept of an **Infobyte**. An Infobyte is a section at the end of each Episode which contains the next level of detail after the material in the main part of the Episode. For example, in Episode C2: Disaster Planning, Infobyte I8 contains two Nybbles. The first Nybble is about Backup Principles, and, the second Nybble is about Magnetic Tape Principles. The Infobyte approach keeps detail out of the mainstream topics of the book, and places the detailed material adjacent to its (commonly-applied) area of application.

Problem: Unrealistic element

Much of the material presented in existing texts conveys the idea to students that computerisation is an almost magic process which can improve any given situation with little effort or mistake.

Existing textbooks describe *smooth success* for computer implementations following ideal sequences of activities. Real world experience is different, and our textbook recognises this. Our book discusses plans, options and issues, and recognises that, in reality, not all will go as planned. Important messages in the book are "It's not easy to computerise", and "Not everyone is keen about computerisation". But, it is manageable and achievable. The novice Mr Makhathini has some computerisation problems, such as staff resistance, and in discussion in our book, various approaches are covered. Mr Makhathini fails at some of his attempts, but the student learns from such setbacks.

We retain a realistic treatment of computers and computerisation throughout our entire book. Our characters raise real life issues, both as people going through the computerisation process, and from the *Wise Guru*. The *Wise Guru* is a character (outside Mr Makhathini's family and business) who talks directly to the student reader of our book. The *Wise Guru* helps the student reader, by offering advice, and comments on issues as Mr Makhathini makes decisions or mistakes. The *Wise Guru* advises about important issues such as examiners' approaches to topics. The *Wise Guru* has empathy for the student in the learning situation. Students reading our book will not feel that they are a third party reading dry facts presented to them by some well-learned authors. Rather, through the use of Mr Makhathini's relatives, staff and friends, and other aids such as the *Wise Guru*, the students reading our book should relate to the development and learning experiences of Mr Makhathini.

Content of our Textbook

There is no new information in our book. The level of our material is first level, novice level; assuming no previous contact with computers, or business. Our *frameworks and approach* are new in order to address the problems discussed above. The textbook contains a full syllabus to support a one year, or shorter, Informatiuon Systems course.

Implementation of problem-driven approach at University of Durban-Westville (UDW)

The textbook (Erwin and Blewett, 1996) is prescribed for Business Information Systems I (BIS1) in the Faculty of Commerce and Administration (Department of Accountancy) at the University of Durban-Westville (UDW) in 1996. Previously, (Stair, 1992) was prescribed, except in 1995 when *no textbook* was prescribed. Stair had become so inappropriate that a decision was taken to issue notes, rather than use Stair. Some of the material from our textbook was used in BIS1 in 1995 as test exposure. BIS1 is a second-year course for most students, since BIS at UDW is currently a *two-year major*. BIS1 is a *year* course, and is not yet modularised/semesterised. UDW expects to begin a *three-year IS major* in 1997. Almost all BIS1 students are registered for a B. Comm. or B. Acc. (towards becoming a Chartered Accountant) degree, with the rest being B.Sc. students. BIS1 (wholly or partially) will eventually form a *service course* for most UDW students.

Adopting a problem-driven textbook means that large parts of formerly product-driven material has to be adjusted. At UDW, BIS1 was reorganised in 1996 to match the problem-driven nature of the textbook with a problem-driven approach to the whole course. Using a product-driven textbook in a problem-driven course is awkward. And vice-versa. It is not a successful strategy to change the textbook to be problem-driven, and then continue to lecture and organise the rest of the course in a product-driven way.

Here are some of the aspects of BIS1/UDW course delivery which supported the problem-driven textbook:

1. Before lectures commenced students had to register on a workstation in the BIS Information Centre (IC) with their own personal details, and some other data re courses taken, and so on. This forced (most) students to view and use the IC. (BIS does not have a Lab! Labs are where chemicals mix to form explosions. BIS has an Information Centre, with the traditional workstations, printers, cables, magazines, and so on.) The students' personal data was then used, by the students, as a base for appending more data, and then as a means of discussing data, information, integrity of data, and so on. Students had access to overall class data as well as their own individual data. Students saw the problems of integrity, confidentiality, and so on, *before* they realised they had seen a database system or the software for database (dBASE IV), or had any lectures on database systems. Real-life problems showed them the issues. The lectures and textbook helped students to make sense of their experience. Students were encouraged to use the SKRAP framework described above.

2. All student work has to be submitted in a personal "bin" on the BIS LAN (Novell). No handwriting is accepted. As part of the problem-driven approach, BIS students must use the technology that they are being taught.

3. No *tutorials* were held, partly due to lack of staff. In lieu of tutorials, BIS students attend Personal Development Programs (PDPs) where the practical work and issues from lectures are discussed in groups of about 20, supervised by an academic. "Tutorial" is a poor term to describe the process of "comfort-generation" that we sought early in the course, so we sought a new term. The use of words to describe course items is critical to the formation of student perceptions about the course. PDPs were optional.

4. "Interaction"s from the textbook were used for practical work to send students out to the real-world.

5. Students also had practical work in the BIS IC (Information Centre). To begin with this work used the students' personal data from each individual in the class. Issues of confidentiality, accuracy, integrity, lies, errors in software/network, non-cooperation and so on, soon arose for discussion at PDPs and in lectures. Novice students find it difficult to relate to business organisation and control issues. Dealing with their own data, seeing the problems there, and moving on to more general approaches and concept-learning, was a comfortable starting approach. Our approach is to offer "Work Performance Improvement Courses", rather than "Computer Courses".

6. The style of test and examination questions altered. Students were coached in the meaning of "Describe" and "Discuss", amid fierce resistance from many who wanted the BIS1 course to be run according to their perception of a "computer course", namely, questions on keyboard activities. Tests and examinations do not examine keyboard activities. BIS1 test and examination questions tend to begin with a scenario, such as "You are the owner of a supermarket, and .... Outline the steps you would take to implement a stock control system, including the use of a computer. Justify your steps at every stage." Or, "During your BIS1 course, you were asked to visit a bank, or a car manufacturer, or a building society, or ..., and report back on their use of spreadsheets (or networks, or ...). Describe your visit, in the context of the spreadsheet

(network, ...) usage you found, and discuss what you found using material from your BIS1 course. Extra marks are awarded for a systematic answer." Students *do not like* these types of questions, but, we believe that this aspect of the problem-driven approach is an excellent way to build up confidence and competence, and to prepare students for the workplace. IT issues are always related to, in fact, flow from, problems.

7. No multiple-choice tests or examination material are used. We believe such material is more a language and semantics exercise rather than a test of knowledge. Such material encourages piecemeal learning, and remembered phrases, at the cost of students' inability to describe or discuss an issue/problem.

8. All practical work and Interactions were DP requirements.

The problem-driven approach for the whole course has been partly diluted/distorted by the large number of disruptions to the academic programme at UDW in 1996. However, we feel that the problem-driven paradigm will be a much more successful one than the previous product-driven style.

Survey of BIS1 Students at University of Durban-Westville (UDW)

The survey was conducted during a scheduled BIS1 lecture period in a lecture hall on 18 June, 1996. Approx. 200 students are registered for BIS1, and 131 respondents completed the "Questionnaire" form (Appendix A). At the time of the survey, UDW had only just re-opened after a closure for almost 6 weeks due to unrest on campus. Lectures were still returning to normal rhythm at the time of the survey.

A special section of the Questionnaire form referred to the textbook. Other aspects of the course were also covered, but, only results from the textbook section are reported here. Not all respondents replied to all items on the questionnaire. Each respondent assigned a RATING to attributes of the textbook on a scale of 0 to 10, where:

0: TOTALLY UNSATISFACTORY
5: SATISFACTORY
10: EXCELLENT

Some respondents, apparently about 10, interpreted this RATING range to mean that they could only use *exactly* 0, 5 or 10. The remainder appeared to award ratings throughout the range 0 to 10. Several answer forms were excluded due to obviously invalid answers. For example, answer 1 = 10, answer 2 = 9, answer 3 = 8, ... down to -10 then start over at 10.

Table 1: "UDW Survey Results for the Textbook 18/6/96" below, shows the results of the textbook survey.

The *first column* in Table 1 shows the "Attribute" rated. For example, "Easy-to-read". The *second column* shows the *Total Number of Ratings (Respondents)* for that attribute. The *third column*, headed "<5 ", shows the *percentage* of Ratings less than 5. In other words, the % of respondents for this attribute who rated the attribute as "less than satisfactory". The *fourth column*, headed ">=5", shows the *percentage* of Ratings greater than or equal to 5. In other words, the % of respondents for this attribute who rated the attribute as "satisfactory or better". The *fifth column*, headed ">=7", shows the *percentage* of Ratings greater than or equal to 7. In other words, the % of respondents for this attribute who rated the attribute as "very good or better". The *sixth column*, headed ">=9", shows the *percentage* of Ratings greater than or equal to 9. In other words, the % of respondents for this attribute who rated the attribute as "almost excellent or excellent".

Table 1: UDW Survey Results for the Textbook 18/6/96

Attribute	Total Number of Ratings (Respondents)	<5 % "less than satisfactory"	>=5 % "satisfactory or better"	>=7 % "very good or better"	>=9 % "almost excellent or excellent"
Useful	128	9	91	63	41
Informative	128	12	88	62	41
Interesting	128	17	83	60	36
Understandable	128	10	90	71	44
Wise Guru	128	14	86	65	37
Easy-to-read	128	8	92	74	47
Well-organised content	126	11	89	63	37
Ease of finding material	126	13	87	55	25
Content arranged for effective learning	126	10	90	60	29
The "Story" at the start of each Episode	128	20	80	61	38
The style of language used	126	11	89	67	44
Examples used in the textbook	126	8	92	60	40
Level of Material (0 if elementary level, 10 if much too complex)	113	18	82	22	7
OVERALL RATING FOR THE TEXTBOOK	122	9	91	57	27

Except for the Attribute "Level of Material (0 if elementary level, 10 if much too complex)", the higher the Rating (the closer to 10) the better the acceptance of and reaction to the textbook by the respondents.

Evaluation of Survey Results

Overall acceptance level for the textbook was very high. 91% of respondents for "OVERALL RATING FOR THE TEXTBOOK" rated it as "satisfactory or better", and 57% as "very good or better".

Below, in Fig. 1 is the graphical, frequency distribution of the OVERALL RATING FOR THE TEXTBOOK replies. The Y-Axis is the NUMBER OF RESPONDENTS. The X-Axis is the RATING given.

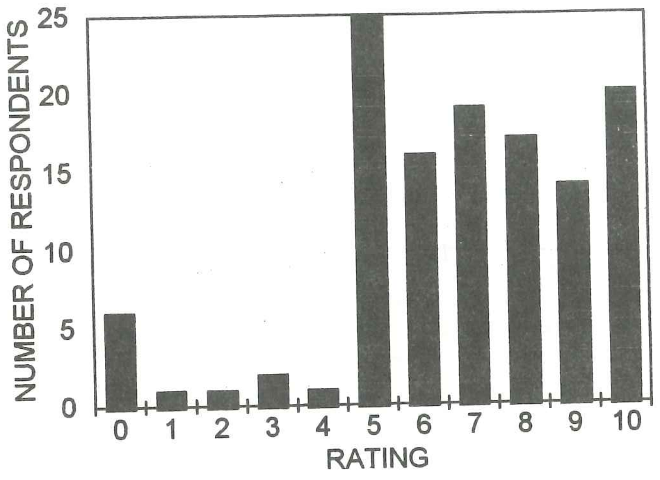


Fig. 1: OVERALL RATING

The **specific new features** of the book also obtained high ratings. The surveyed *new features* were "Wise Guru", "Easy-to-read", "The 'Story' at the start of each Episode", "The style of language used" and "Examples used in the textbook". Below, in Figs. 2 to 6, are the graphical, frequency distributions of the RATINGs for these new features. The Y-Axis, in each case, is the NUMBER OF RESPONDENTS. The X-Axis is the RATING given.

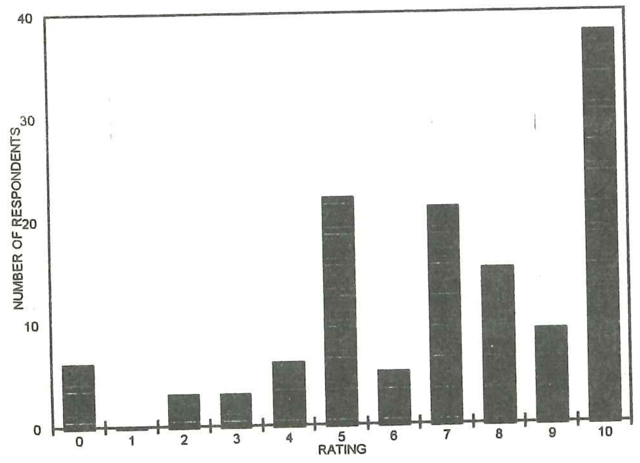


Fig. 2: Wise Guru

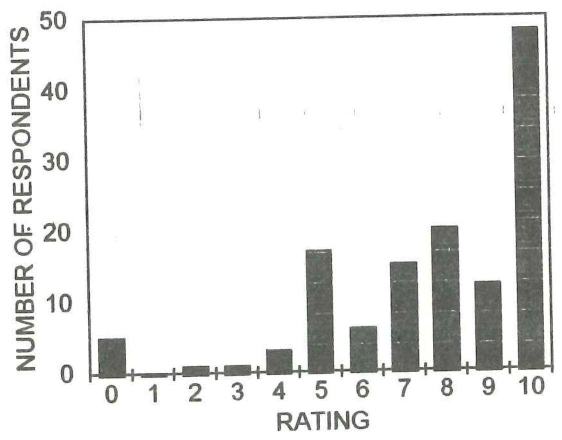


Fig. 3: Easy-to-read

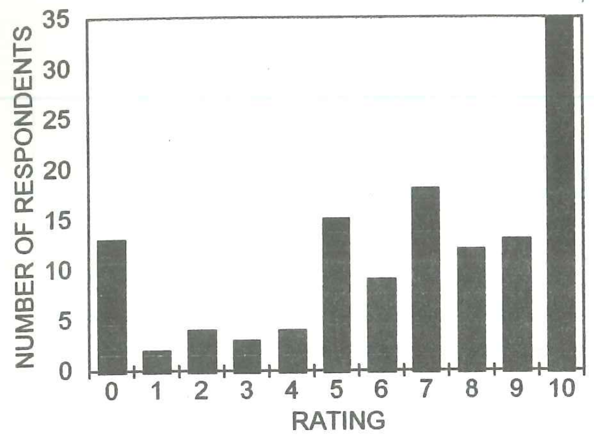


Fig. 4: The "Story"

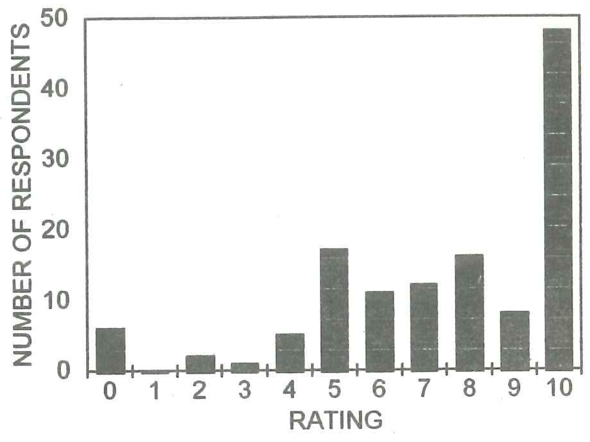


Fig. 5: Style of language

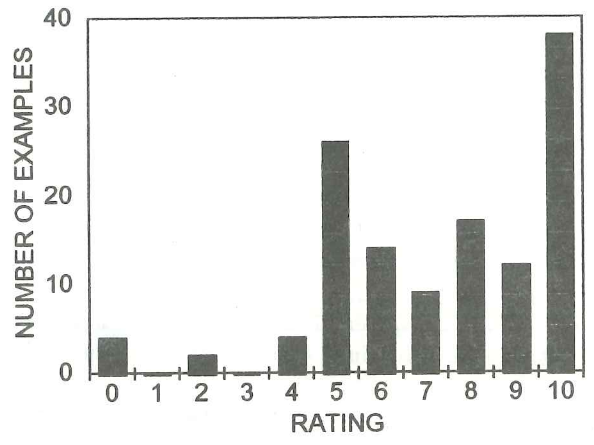


Fig. 6: Examples

A textbook is one medium, amongst many, which can contribute to course success/failure. Examination results, and students' level of understanding and knowledge, can be other indicators of course improvement compared with previous material and methods. For the authors, the results are pleasing, although preliminary. There was no indication of significant rejection of any feature of the textbook, or the textbook as a whole. Several other aspects of the course received poor RATINGs, showing that respondents did not seem to rate all questionnaire items highly regardless of the item.

Conclusions

The problem-driven approach appears to be a satisfactory approach for a first-year Information Systems course, as well as suited for education of business people.

The provision of a full instructors' support pack by the end of 1996 (as scheduled), and possibly video Episodes, could further assist course results. The Overhead Projector slides should be available under Microsoft Powerpoint Version 7.0 (currently), via Internet for prescribers. Course instructors can then select individual slides for printing on Transparencies, or retain the electronic slides for display within a classroom with a notebook/laptop computer. The course instructor can customise the overhead slide material, and other, to suit a local course.

A textbook is not the only medium of teaching in a course. Tutorials, lectures, PDPs, assignments, practicals, and so on, all make a contribution. But, for many students, the textbook sets the tone and level of the course, and becomes a safe source of useful knowledge.

Course instructors need to revisit course objectives, course methods and course perspective to determine an appropriate conceptual model for teaching. Driven by changing student population, changing employer requirements and changing society, there is a need to reevaluate existing paradigms. Courses should be designed in the best possible way, and then, the best possible textbook, and other material, used to support the course. We believe that the problem-driven conceptual model is the appropriate one for the South African future in Information Systems education. (Moulder, 1994) says we have to "ride the wave" that is rolling over South African society, and education in particular. Otherwise, we will drown!

Notes

Comments, criticism, suggestions, experiences on any aspect of the textbook, courses, students, and so on, are welcome. E-mail addresses for the authors appear at the start of this paper and on the Title Page of our textbook. The authors invite other institutions and colleagues to share e-mail correspondence, with survey results, course methods, textbook assessment, and student matters. Even the *Wise Guru* has an e-mail address. The front pages of the textbook contain further detail about the design of the textbook.

Acknowledgements

Juta & Co. Ltd. of Cape Town put together a great group of individuals, making up a great *team* of publishers. Thanks for your expert assistance and happy production days!

References

Capron H. (1990), *Computers - Tools for an Information Age*, Second Edition, The Benjamin/Cummings Publishing Company, Inc., Redwood City, California.

Erwin G.J. and Blewett C.N. (1996), *Business Computing: An African Perspective*, Juta & Co Ltd., Cape Town, South Africa. ISBN 0 7021 3340 X, approx. 700 pages.

Hanks P. (Ed.) (1988), *The Collins Concise Dictionary of the English Language*, 2nd. Edition, Collins, London.

Long L. and Long N. (1990), *Computers*, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, p. 60.

Moulder J. (1994), Riding the Wave, in *FOCUS*, Transformation Section, University of Natal, Durban, South Africa, Winter 1994, pp. 21-22.

O'Brien J. A. (1988), *Information Systems in Business Management*, Fifth Edition, Richard D. Irwin, Inc., Homewood, Illinois.

O'Leary T.J. and Williams B.K. (1985), *Computers and Information Processing*, Benjamin/Cummings Publishing Company, Inc., Menlo Park, California.

Pfleeger C.P. (1989), *Security in Computers*, Prentice-Hall Inc., International Edition, New Jersey.

Stair R.M. (1992), *Principles of Information Systems: A Managerial Approach*, boyd and fraser publishing company, Boston, MA.

****

University of Durban-Westville
Faculty of Commerce and Administration
Questionnaire re Business Information Systems 1 (BIS1)
To a BIS1 student in 1996:

Hi! Here is an opportunity for you to comment on the BIS1 course and its components. Please answer the following questions as fully and honestly as you can so that UDW can assess the quality of your BIS education. **You do not need to supply your name**. You must be a registered BIS1 student in 1996. **Return your completed Questionnaire to Professor Erwin.**

Assign a RATING for each component as:

0:	*TOTALLY UNSATISFACTORY*
5:	*SATISFACTORY*
10 :	*EXCELLENT.*

OVERALL BIS1 COURSE	***RATING***
Overall *RATING* for BIS1 course:	------
Usefulness to you	-------
Interesting	-------
Organised (0 means very badly organised)	-------
Difficult (0 means very difficult)	-------
Relevant to job/work future	-------
Practical Computer Assignments (PCAs)	-------
Hard to understand (0 means very hard to understand)	-------

What would you like to see **different** in BIS1? ---

What is **missing** from the BIS1 course? ---

What did you like MOST about the BIS1 course? ---

What did you DISLIKE about the BIS1 course? ---

BIS STAFF	***RATING***
Overall rating for Mr Webb:	-------
Overall rating for Ms Oldman (PDPs):	-------
Overall rating for Rajiv Lutchman (IC):	-------
Overall rating for Prof. Erwin:	-------
Overall rating for BIS Staff attitudes:	-------

INFORMATION CENTRE (IC)	***RATING***
Availability of computer(s)	-------
Help from demonstrator(s)	-------
Hours of opening	-------
Access to information about IC usage	-------
Overall rating for the IC:	-------

Appendix A: BIS1/UDW Questionnaire. 18 June 1996. **Page 1 of 2**

LECTURES	***RATING***
Useful	-------
Interesting	-------
Start and end on time	-------
Suitable venue	-------
Overall rating for Lectures:	-------

TEXTBOOK: "Business Computing: An African Perspective by Erwin and Blewett."

	RATING
Useful	-------
Informative	-------
Interesting	-------
Understandable	-------
Wise Guru	-------
Easy-to-read	-------
Well-organised content	-------
Ease of finding material	-------
Content arranged for effective learning	-------
The "Story" at the start of each Episode	-------
The style of language used	-------
Examples used in the textbook	-------

Level of Material (0 if elementary level, 10 if much too complex) -------
What is a CCQ? --
What is KAIR? --
What is COURSE? --
What is STAIR? --
What is MOLEST? --
What did you **like** about the textbook? (Use overleaf if needed) --------------------------
What did you **dislike** about the textbook? (Use overleaf if needed) ----------------------
Overall rating for the textbook: -------
Other comments on the textbook: --

PERSONAL DEVELOPMENT PROGRAMS (PDPs)

	RATING
Useful	-------
Interesting	-------
Start and end on time	-------
Suitable venue	-------
Overall rating for PDPs:	-------
Number of PDPs you attended:	------
Want more/less/about the same PDPs:	------

OTHER COMMENTS on any aspect of BIS1: (write on back of this page):
Are you taking BIS1 towards becoming a CA? --------
If you answered NO, why are you taking BIS1? --
****
END OF QUESTIONNAIRE. Thank you for your cooperation. Date completed: -------------
Professor Geoff Erwin: Head: Business Information Systems.

Appendix A: BIS1/UDW Questionnaire. 18 June 1996. **Page 2 of 2**

TEACHING A FIRST COURSE IN COMPILERS WITH A SIMPLE COMPILER CONSTRUCTION TOOLKIT

Dr. G. F. Ganchev
Computer Science Dept., University of Botswana
P. Bag 0022, Gaborone, Botswana
Phone: (267) 308221, E-mail: ganchev@noka.ub.bw

Abstract

We describe the use of a toolkit designed to support the Compiler Construction course in the University of Botswana. The toolkit is based upon the principles of simplicity, modularity and flexibility. Its educational goal is to maintain a balance between theoretical material and the practical presentation of concepts. We view the students as active participants in Computer Aided Learning . They actively explore and control the interactions and monitor the data and control flow in the compilers they build. The feedback provided by the toolkit's interface helps the students understand where they are in the compilation process.

Keywords

Computer Aided Learning, Compiler Construction, Scanning, Parsing, Grammar, Context Analysis, Code Generation, Stack Machine

1. Introduction

In many universities Compiler Construction is no longer a compulsory undergraduate Computer Science course. Perhaps one of the reasons is the general perception that learning compilers can be complex, while only a small percentage of graduates will be involved in compiler writing in their careers. Nevertheless the subject area "Programming Languages" of the ACM Computing Curricula [ACM-91] includes at least four main knowledge units that are closely related to understanding compilers. These knowledge units are Representation of Data Types, Sequence Control, Run-Time Storage Management, and Language Translation Systems.

Our experience indicates that the difficulty in learning compilers does not necessarily lie in the complexity of the topic itself. The cause is really two-fold: one can be characterized as a problem of foundation, the other as a problem of presentation:

- Students often lack real understanding of some fundamental concepts that are prerequisite to the topic. They are capable of obtaining good test scores, but fall apart when asked to apply their knowledge in practice. Modern compilers are syntax driven. Learning compilers requires some ease with formal languages and automata. Students should have some initial experience in the theory in order to understand the current syntax analysis methods, and their impact on the other compiler components.

- The available introductory textbooks either concentrate on one particular compilation model and even one example programming language, and then explain their model in a great detail but miss the general state of the art picture, or have an encyclopedia-like approach covering a range of methods and techniques, but lacking concrete guidelines for their integration and implementation.

As a first step in addressing these issues, in the UB curriculum we have a separate course in Languages and Automata. Still, in the Compilers course we try to rely on as simple a set of concepts as possible. We organized our course with a focus on the concepts that differentiate one compilation model from the other, and included a closed laboratory component intended to allow the students to rapidly build components of compilers for languages that they define. We encourage the students to explore and choose particular models as described below for each of their compiler components.

2. The Course Contents

The course is designed for undergraduate single major Computer Science students. Our goal was to maintain a balance between theoretical material and the practical presentation of concepts.
The students taking the course have passed a course in Formal Languages and Automata and a course in Programming Languages.

After surveying a number of textbooks, analyzing the advanced material in sources as [Hol-90], and considering the algorithms of several models, we developed our course from the themes listed below. We were convinced that a set of tools designed to support these themes would provide the students a meaningful and lasting learning experience not only in compiler construction, but also more generally in building large software systems.

Our course starts with an introduction to language translation systems encompassing the range from assemblers to compilers and interpreters with emphasis on syntax directed translation. Then we present the main parsing, translation, and code generation techniques in use today with many examples. We recommend two texts for the course - [Wat-93] and [Aho-86], however none of them is strictly followed, and the lectures contain a lot of material that binds the concepts being presented.

During the first half-semester we cover scanning (case-type and with finite automata) and parsing. Two bottom-up and two top-down parsing techniques are discussed: mixed precedence, LR(k), Recursive Descent and LL(k). All are supported by examples and exercises. The second half of the course concentrates on semantic analysis, run-time storage management (including routines and data types representation), and code generation (including expression evaluation and sequence control). The modern emphasis on compiler construction tools is underlined. The horarium is three lecture hours and one two-hour guided laboratory per week. A laboratory assistant helps the students learn the toolkit in closed laboratory sessions.

3. The UB Compiler Construction Toolkit

The University of Botswana toolkit differs from other similar systems [Ben-90,Hol-90] by the intentional stress on simpler concepts. In [Mar-95] M. Maredi and H.J. Oosthuizen point out the problems associated with the poor mathematical background of a large group of South African students. We have to overcome a similar obstacle. Many of our students experience difficulties in understanding automata based models, attribute-value flow, semantic specifications. In contrast, more mechanistic concepts are easily understood. For example, the students understand better the construction of a case-type scanner than building a finite automaton from a regular expression. They understand more easily the construction of precedence-based shift-reduce parsers than LR(k) parsers. They prefer building translation schemes to defining semantic functions. Fig. 1 shows the model of a compiler adopted in our course.

We made a fundamental decision that our tool will be used primarily by the individual students rather than by the instructor in the classroom (although it could certainly be used for classroom demonstrations). Everyone knows the importance of the student role in the learning process. It has been stressed by several authors [Cou-93,Sch-93] that one of the most important ingredients of a successful learning tool in science and engineering education is its flexibility and its ability to be adapted according to the student's needs and personal ideas. This is why our toolkit offers the students differing alternative models to probe their understanding of different parts of compilers they build. We have included the following:

- two scanner generators. One of them generates case-type scanners from a BNF description of the source language syntax. Some conventions must be observed for the generated scanner to be immediately ready for use. Its main advantage is its very simple structure which students are able

to understand and modify without difficulties. In the closed laboratories this scanner generator is used in all cases which are not primarily concerned with scanning. The second generator generates scanners from a description of the lexicon by regular expressions. As mentioned above, we prefer the first, more mechanistic approach. In the closed exercises the second generator is used only to illustrate a more general and modern approach. Because of the standard interface used by both generators, in their individual work the students are able to choose the approach that they prefer.

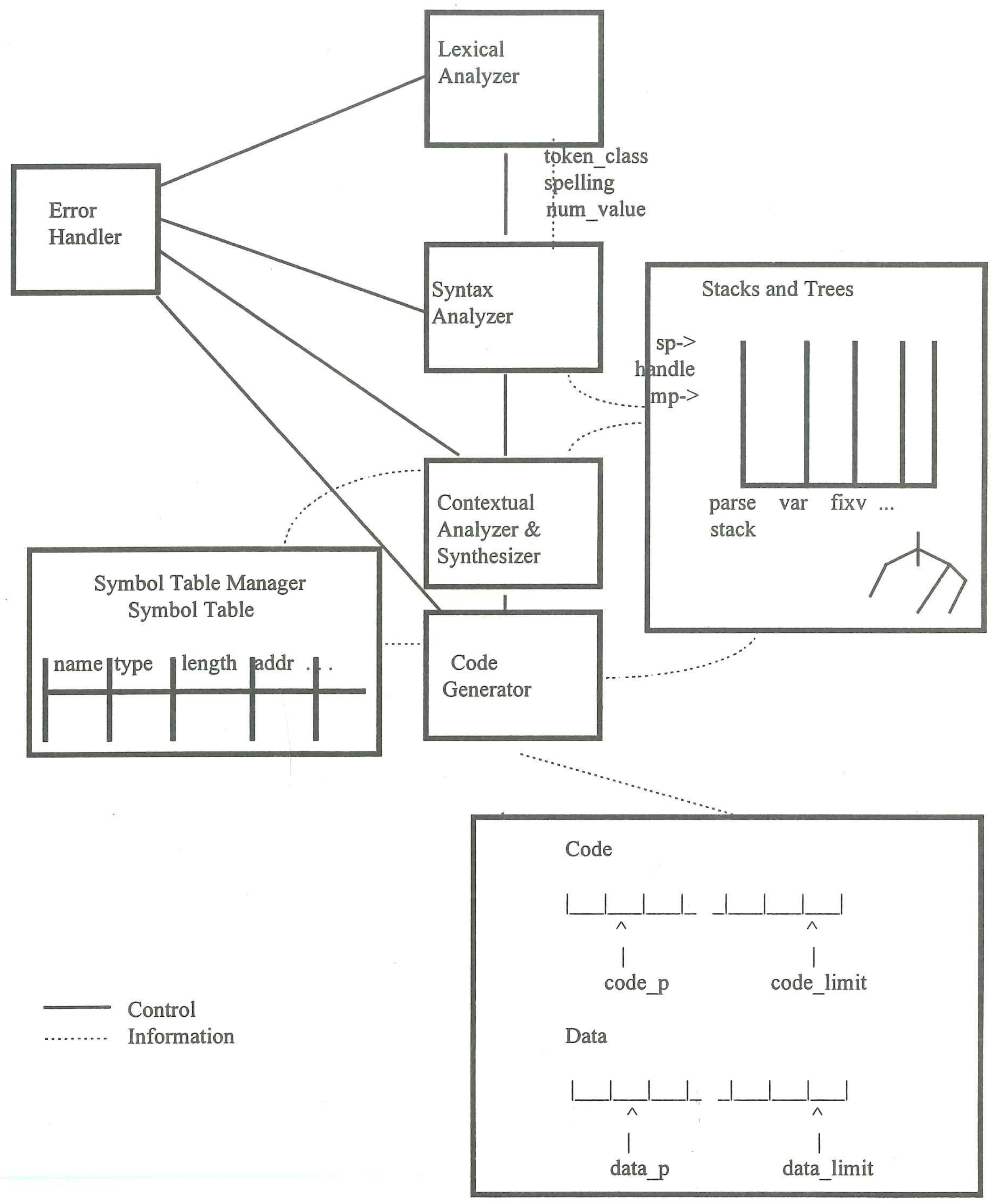


Figure 1. The adopted structure of a compiler

- two parser generators - a top-down (LL(1)) and a bottom-up (mixed precedence). Only the bottom-up parser generator was operational in the last academic year, however the students practiced top-down parsing by building hand-written recursive descent parsers from extended BNF descriptions of the source languages.

- an almost empty "skeleton" routine for semantic analysis and synthesis. The students try their semantic actions interactively before supplying the code to fill in the "skeleton" routine. The different options that they could explore originate from the different possibilities for defining the concrete syntax and linking it with the semantics of the source language.

- a set of primitive code generation routines that generate assembly code for the TAM virtual machine (see below). The students call these routines from their own semantic programs.

- an error handler, a major part of which is an adjustable syntax error recovery routine implementing "panic" mode of recovery [1]. In addition students are encouraged to implement and explore recovery by replacement and error productions [1].

- a stack-machine emulator. We have adopted the TAM virtual machine described in [9], and have developed an interactive user interface and an assembler for it.

- utility programs. At the time being these include symbol-table, stack and tree manipulating routines.

- a standard interactive user interface that allows prototyping and debugging of compilers being built by the students.

Below we shortly discuss the last three components.

3.1. Defining Symbol Tables

Understanding symbol-table manipulation is a key point in learning contextual analysis and synthesis. Our goal was to implement a reusable and easily adjustable component that would conveniently allow visualization. We chose to implement the symbol table as a heterogeneous C++ class having objects of another class as an attribute. Each symbol table entry consists of a scope level number, a name and an attribute pointer. In this way all symbol-table operations were implemented in advance, while the structure of the attribute was left open. In a minimalist scenario, the students are only expected to specify the structure of their attribute. This was the case in our closed laboratories. In order to make the exercises simpler and to provide a standard visual image of the table, we used attributes with a fixed structure, however the heterogeneous class approach allows much more flexibility. Not only can the attribute be different for different compilation exercises, not only can it be arbitrarily complex (for example, a tree structure), but also each symbol-table entry can have a different structure for its attribute. This could be useful for more advanced cases or optimal utilization of memory.

An interesting side effect of the flexible implementation of the symbol table was the active interest of several of our students in object-oriented programming.

3.2. Defining and Manipulating Trees

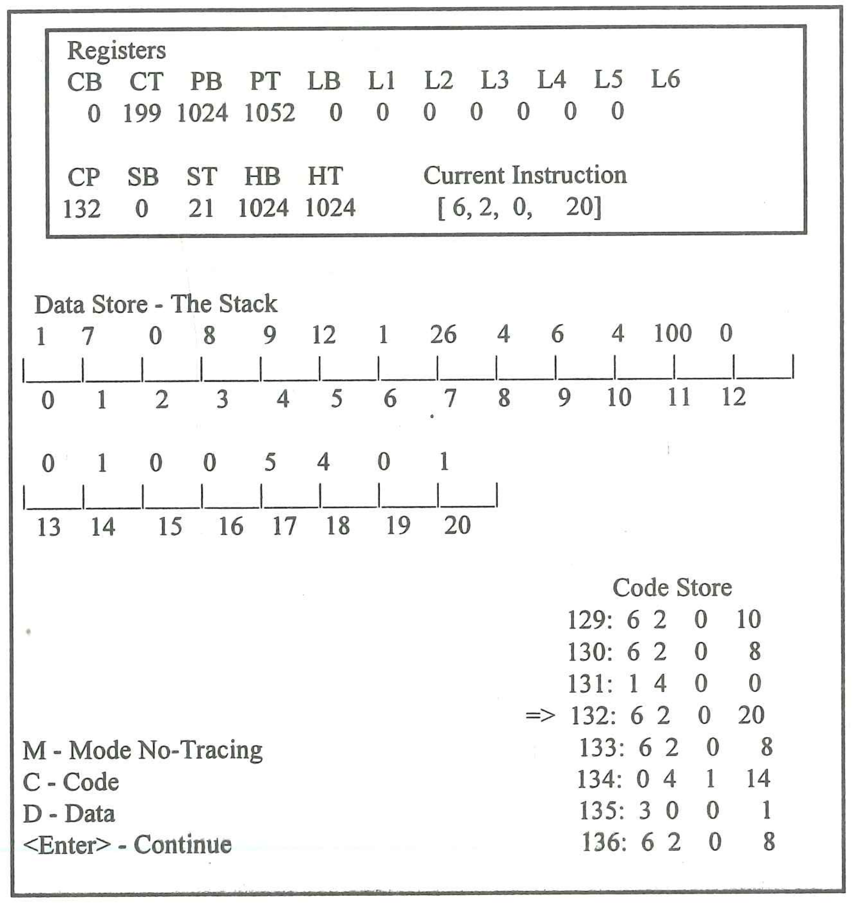
Parse trees and abstract syntax trees are popular and well understood intermediate representations, but it would be unrealistic to expect that the students will have the time to implement tree structures in the time scheduled for our course. Instead we implemented a C++ class that allows flexible tree manipulation compatible with the one described in [Wat-93]. We eliminated the limitation of [Wat-93] on the arity of nodes. For visualization we chose a two-dimensional representation of trees as more appropriate for our educational goals, instead of the widely used linear representation.

3.3. The Stack Machine

The code generation part of a compiler is very much dependent on the target machine. Different machines have different addressing conventions and register configurations. A virtual stack target machine has many advantages for a first course in compilers, like ours. Firstly, using a stack target machine eliminates the problem of intermediate storage and register allocation for expression evaluation. Secondly, the stack is the natural run-time storage organization for languages with nested scopes and recursive routines. Both texts [Wat-93] and [Aho-86] refer to virtual stack machines, but the presentation in [Wat-93] is tightly bound to such a machine. We found this presentation very useful and easily understandable by the students.

The TAM stack machine [9] has two separate stores for code and data. The data store accommodates a stack and a heap growing in opposite directions. All evaluation takes place on the stack. Primitive arithmetic, logical and other operations are treated uniformly with pre-programmed functions and procedures. An important advantage for our course is the fact that the procedure call and return conventions are implemented at the TAM machine level. This simplifies code generation, while still allowing students to observe the adopted run-time storage organization. A number of registers are dedicated to specific purposes.

Our implementation of the TAM stack machine includes a loader and an interpreter. Each of them illustrates a topic in our course. The interpreter can be run in a step-by step mode, thus allowing students to observe the execution of their compiled programs. Fig. 2 is a snapshot of a screen showing our user interface to the TAM interpreter. The current instruction is at address 132 of the code store (pointed by the register CP). The program is stored from location 0 (register CB) to location 199 (register CT). Next available location in the stack is 21 (register ST), the stack base is 0



```
Registers
CB   CT   PB   PT   LB  L1  L2  L3  L4  L5  L6
 0  199 1024 1052   0   0   0   0   0   0   0

CP   SB  ST   HB   HT        Current Instruction
132  0   21  1024 1024        [ 6, 2,  0,    20]


Data Store - The Stack
1  7    0   8    9   12   1   26   4   6    4  100  0
|___|___|___|___|___|___|___|___|___|___|___|___|___|
 0   1   2   3   4   5   6   7   8   9   10  11  12

 0   1   0   0   5   4   0   1
|___|___|___|___|___|___|___|___|
 13  14  15  16  17  18  19  20

                                       Code Store
                                   129:  6 2   0   10
                                   130:  6 2   0    8
                                   131:  1 4   0    0
                                => 132:  6 2   0   20
M - Mode No-Tracing                133:  6 2   0    8
C - Code                           134:  0 4   1   14
D - Data                           135:  3 0   0    1
<Enter> - Continue                 136:  6 2   0    8
```

Fig. 2. A snapshot showing the TAM interpreter interface