Full paper

Sound Globules: An Algorithmic Composition Environment

Andrew Eales

Wellington Institute of Technology

Andrew.Eales@Weltec.ac.nz

Alex Metcalfe

IBM New Zealand

Metcalfe@nz1.ibm.com

Abstract

This paper discusses the re-development of a legacy software system for algorithmic music composition. The software was developed using the Business Object Notation (BON) object-oriented development method and the IBM Structural Analysis for Java toolset. Sound Globules is an environment that is able to create music having a high degree of coherence, while avoiding the monotonous regularity inherent in music constructed from many mathematical processes. Sound Globules was implemented in Java, using the Matisse GUI builder, the Java Media Framework, and an embedded scripting language that allows real-time machine control of the compositional process. The BON model was derived from the specification provided by the user-manual. This paper also describes and evaluates the BON methodology, and compares the BON notation and process to the Unified Modelling Language (UML).

Keywords: Computer Music, Algorithmic Composition, Software Development, BON, UML.

1 Introduction

Regularity in musical structure promotes orderliness and coherence; while at the same time easily results in a work that is uninspiring in its predictability. Stravinsky noted the use of irregularity in the music of Haydn:

"Of all the musicians of his age Haydn was the most aware, I think, that to be perfectly symmetrical is to be perfectly dead" (Stravinsky, 1958).

The use of mathematical processes to create music has become commonplace with the development of multimedia computing. Fractals (Miranda, 2001), Cellular Automata (Eales, 1996) and genetic algorithms (Miranda, 2006) are popular algorithmic processes used to generate data that can be translated into various musical parameters.

Ariza (Ariza, 2005) provides an overview of the history of algorithmic composition using computers. The question of whether processes can create satisfying musical works is a non-trivial issue. Kugel (Kugel, 1990) drawing on the work of Myhill (Myhill, 1952) believes that computational procedures can never fully represent the creative musical process.

In the absence of any substantial quantitative or qualitative theory of musical style, structure or content, the authors of this paper are forced to take a subjective view of the results provided by various algorithmic compositional environments. Results obtained from these environments have often been disappointing to the authors. A novel approach to the problems faced when specifying musical structure and content is found in the Sound Globs system (Kozerski, 1989) for algorithmic music composition available during the early 1990's. This approach generates musical parameters according to a frequency distribution where the distribution can evolve at any time during performance. Parameters are used to generate sonic events, providing a soundscape that consists of a complex unfolding of musical ideas. Myhill (Myhill, 1979) discusses the control of non-deterministic processes in musical composition.

The following sections introduce the Sound Globs environment, discuss the analysis and design of the software, and describe the re-development of the software using the Java Media Framework (Metcalfe, 2006). The Business Object Notation (BON) object-oriented development method is also described, evaluated and compared to the Unified Modelling Language (UML).

Note that the developer of the original Sound Globs software, Russ Kozerski (Kozerski, 2007) who holds copyright to the software has allowed the development of the Sound Globules system.

2 The Sound Globules Environment

The sections below describe the Sound Globules software and introduce the concepts and controls encountered when interacting with the user-interface.

2.1 System Overview

In the Sound Globules environment, a "texture" is a part or layer within a musical composition. A texture consists of sonic events that are defined by a set of parameters that determine the duration, pitch, temporal density and vertical pitch density of each event. The musical results

of this compositional process can be auditioned and modified in real-time for each texture. Because the system uses MIDI (MIDI Manufacturers Association, 2001), specific timbres are determined by the audio hardware or software synthesis system used during performance. Transformation of digital audio is not supported.

By combining textures using the performance generator, a complete musical composition is created. Figure 1 shows the architecture of the Sound Globules environment.

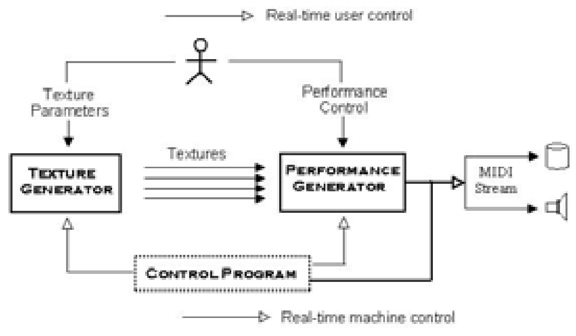
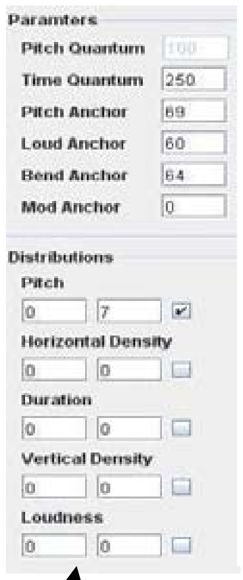


Figure 1. Architecture of the Sound Globules Environment

Users can create a musical work by interacting with the software, or, if a finer granularity of control is required, it may be achieved by using the embedded programming language. This interpreted "performance language" is able to combine and modify pre-defined textures in real-time. An example of the performance language is provided in section 2.4. The performance is realised by translating the combined textures into a MIDI data stream that can drive a sound card or be saved to a standard MIDI file for future playback.

2.2 The Texture Generator

The Texture Generator and an enlarged view of the available parameters are shown in figure 2. Musical parameters and their ranges are shown on the left-hand side of the user interface window. The quanta parameters define units of measurement for pitch and time. Pitch measurement is in cents, allowing textures having pitch relationships greater than or smaller than the traditional 100-cent chromatic scale of Western music. Anchors define a lower bound for each parameter, while the associated range defines a lower and upper bound relative to the anchor. The sample graph at the top of figure 2 provides a real-time graphical view of the events being generated by the software. Specific values for a given parameter can be emphasised using associated probabilities defined in the distribution editor shown in figure 2:


Paramters

Pitch Quantum 100

Time Quantum 250

Pitch Anchor 69

Loud Anchor 60

Bend Anchor 64

Mod Anchor 0

Distributions

Pitch 0 7 ✓

Horizontal Density 0 0

Duration 0 0

Vertical Density 0 0

Loudness 0 0

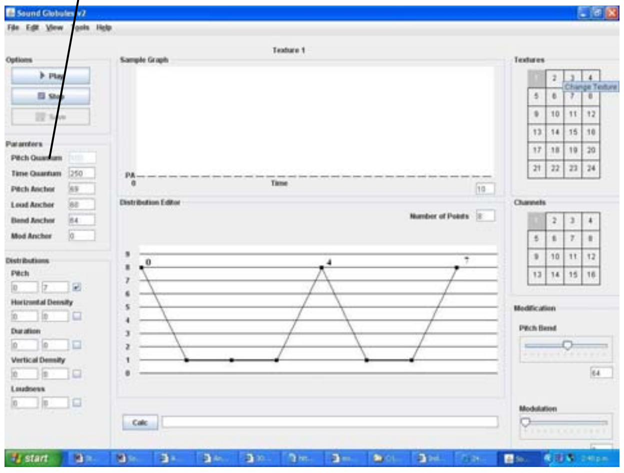


Figure 2. The Texture Generator

Dragging the control points with a mouse modifies probability weightings. The probability weightings within the distribution editor of figure 2 indicate that pitches within a range of eight pitches above the pitch anchor are to be generated with a significant probability weighting given to pitches 0, 4 and 7. A high probability weighting assigned to pitches 0, 4 and 7 above the pitch anchor emphasises an a-minor triad. This creates a traditional key centre, resulting in a musical work having a compositionally conservative pitch organization. The two matrixes on the right-hand side indicate the current active texture within the editor, and the MIDI channel currently assigned to the texture.

2.3 The Performance Generator

The Performance Generator that combines textures to produce a complete musical work is shown in figure 3. Textures can also be modified during performance by the performance generator. The sliders shown in figure 3 can modify the parameters for a specific texture, or can function as global controls allowing parameter changes to apply to all currently playing textures. User interaction with a specific texture allows for the placement of a texture into the musical whole, while global controls allow the entire work to be modified during performance.

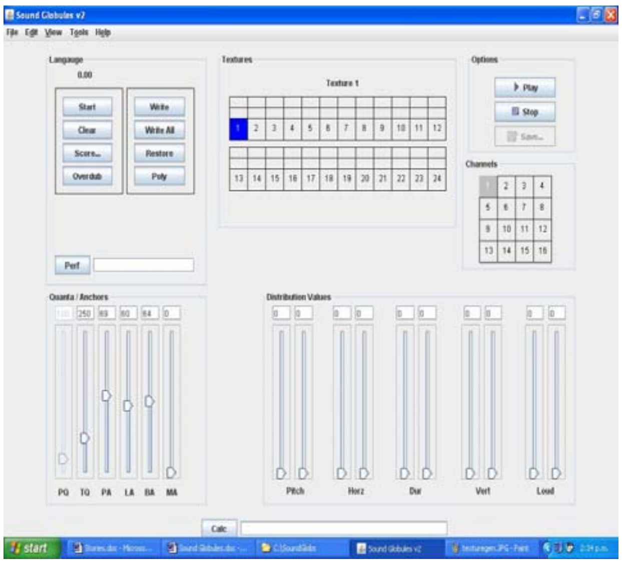


Figure 3. The Performance Generator

The control box (top, centre in figure 3) indicates which textures are currently being performed; texture one in this example. Buttons above each texture number select the texture that will receive modification commands that change musical parameters. Textures under performance control behave in a similar manner to value or reference parameters in programming languages. The performance generator receives copies of the textures created by the texture generator. These copies can be modified without changing the originals, or the performance generator can overwrite the originals as the user makes changes. Textures can also be routed to specific MIDI channels using the matrix placed on the right-hand side of the control window.

The GUI's shown in figure 2 and figure 3 were developed using the Matisse GUI builder that is part of the NetBeans distribution (NetBeans, 2008).

2.4 Real-Time control

The Generic Interpreter (Generic Interpreter, 2008) was used to develop a grammar for the embedded performance score. The Java class GlobsGrammer defines the symbols and associated semantics of the language that creates the embedded performance score. This language allows a Sound Globules performance to be controlled by a computer in real-time. An example performance score annotated by comments is shown below:

0	**1**	// make texture one current
0	**play**	
3.5	**2**	// add texture two and make it current
3.9	**on3**	// add texture three
2.0	**pa70**	// set pitch anchor for texture two to 70
0	**gla10**	// add 10 to the global loudness anchor
5.0	**stop**	

The leftmost column indicates delta timings representing elapsed seconds from the previous event. A delta time of zero specifies concurrent events; the example performance duration is thus 14.4 seconds. The Score… button shown in figure 3 opens an editor window, allowing the user to enter performance score commands. A performance score is stored as a text file, facilitating editing and modification within a wide variety of applications.

3 Software Analysis and Design

The systems analysis phase of the project proceeded by deriving the functional requirements of the system from the user manual. An extremely detailed user manual (Kozerski, 1989) was supplied with the original Sound Globs software. Analysis and development then followed the activities prescribed by the BON object-oriented development method (Waldén, 1994).

3.1 The BON Software Development Method

The authors selected BON for the Sound Globules project, as they were interested to compare their experiences using UML as a design notation to the notation and processes provided by BON. Development of a BON model starts with the static analysis and design of classes. The lifecycle of the complete method defines nine activities grouped into three parts. These are:

Gathering Analysis Information: determines the system border, identifies candidate classes, and groups related classes.

Describing The Gathered Structure: identifies class content, develops scenarios, and specifies contracts.

Designing A Computational Model: develops and refines the problem domain architecture, factors common behavior, and reviews the complete system. A detailed discussion of these activities is provided by (Waldén, 1994) and (Waldén, 1998).

BON provides two kinds of class diagrams, informal class charts, and formal class diagrams. Dynamic behaviour in BON is depicted using event and scenario charts, as well as object communication diagrams. Charts are informal, descriptive accounts of object interactions.

A detailed description of BON is beyond the scope of this paper. The authoritative reference to BON (Waldén, 1994) is freely available. A brief description of BON (Paige, 1999) is also available.

3.2 BON Compared to UML

It is important to note that BON, unlike UML, provides a development method covering the entire software development lifecycle. A notation for modelling static object relationships and dynamic object behaviour supports the method.

3.2.1 BON Object Relationships

The two fundamental object relationships in BON are client and inheritance relationships. Three types of client relationships are defined: simple client association, aggregation (whole/part relationships), and shared association. Aggregate relationships in BON are unambiguously defined as these relationships are derived from the Eiffel concept of the expanded type (Meyer, 1997). BON avoids the ambiguities encountered when specifying the semantics of object relationships using UML (Eales, 2005).

A class chart is a high-level English description of the interface to a class, containing queries (accessors), commands (mutators) and constraints. Class diagrams have methods and attributes that are typed and specify software contracts. Figure 4 shows a Sound Globules class diagram using the formal BON class notation where method preconditions and postconditions are denoted by question and exclamation marks.

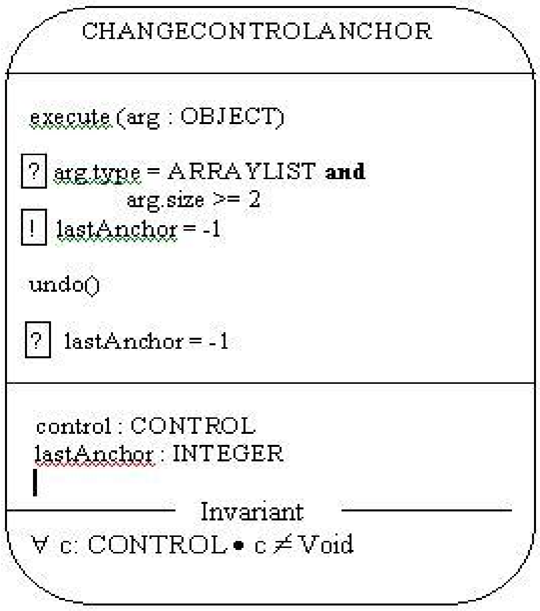


Figure 4. BON Class Notation

Identified constraints are refined in the class diagram using typed predicate logic. This class forms part of a command pattern and has two methods, execute(arg) and undo(). The precondition for method execute(arg) stipulates that the argument must be an ArrayList containing two or more elements. The postcondition stipulates that the lastAnchor attribute must contain the value of –1, which forms a precondition for the undo() method.

3.2.2 BON Dynamic Models

BON only provides two behavioural notations; the scenario charts and object communication diagrams mentioned in section 3.1. These correspond to collaboration and sequence diagrams in UML. Figure 5 shows a BON scenario chart annotated with a textual description:

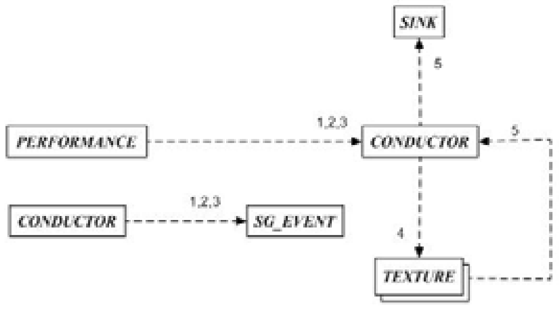


Figure 5. BON Scenario Chart

1. A Peformance object creates a Conductor.

2. A Performance object copies Textures and sends these Textures to the Conductor.

3. Performance Instructs Conductor to Conduct.

4. Conductor instructs Textures to start generating.

5. Conductor receives data from Textures and sends the data to a Sink.

Table 1 outlines the main differences between BON and UML 2.0.

Table 1: BON compared to UML

	UML2.0	BON
Method	No	Yes
Static Diagrams	6	2
Behavioural Diagrams	7	2
Constraints	OCL Extensions	Yes

It is interesting to note that constraints are an integral part of the BON methodology and do not require an external mechanism such as UML's Object Constraint Language (OCL). The resulting diagrams are thus more concise than their UML equivalents. Unlike OCL constraints that may or may not be inherited, BON constraints are always inherited, providing clarity and a direct mapping to the Eiffel programming language. The authors found that BON was easy to use, unambiguous, and allowed constraints to be clearly specified.

4 Software Implementation

Sound Globules was implemented within the NetBeans IDE using Matisse for GUI development, and the timing and MIDI capabilities of the Java Media Framework. The software consists of eighty-four classes arranged in the package structure shown in figures 6a and 6b. This view of the source code was generated by the IBM Structural Analysis for Java toolset (IBM, 2004) which provides software metrics and a visual representation of the

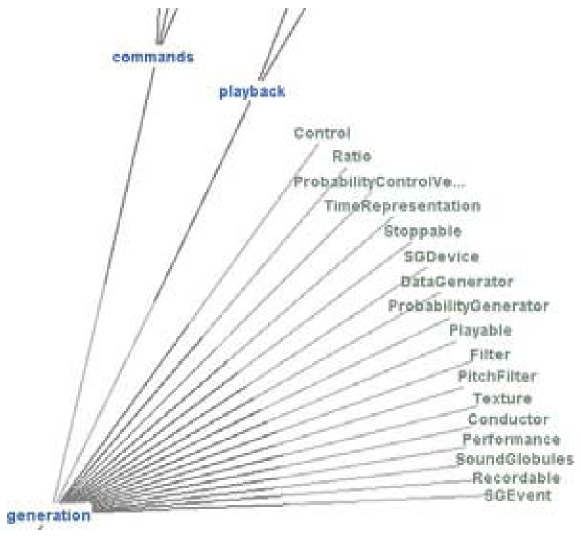


Figure 6b. Expanded Package View

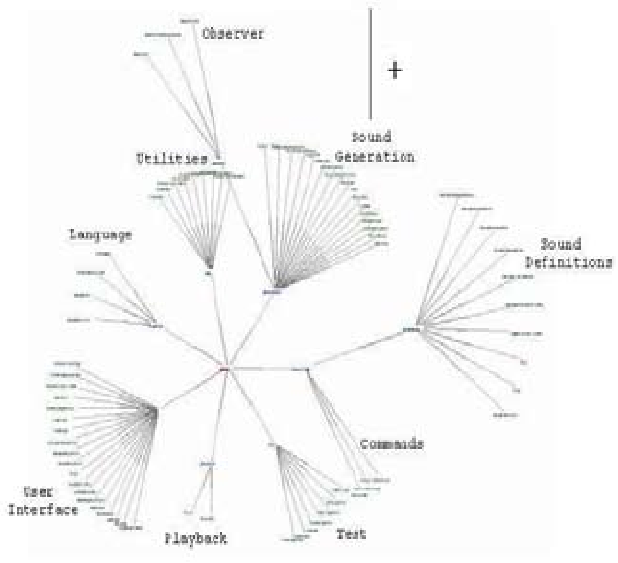


Figure 6a. Structural Analysis of the Sound Globules System

architecture of the software. Different levels of detail allow redundant relationships and relationships that can be simplified to be identified during development. Structural analysis also ensures that different concerns such as the core problem domain classes and user interface classes (packages SoundGeneration and UserInterface in figure 6a) are clearly separated. Additionally, implementation relationships between classes can be compared to the BON design, ensuring conformity between design and implementation.

5 Conclusion

The Sound Globules software provides a unique approach to algorithmic music composition. By allowing a wide range of musical parameters to be controlled in real-time, the monotony of self-similarity is avoided. The original Sound Globs software (developed prior to the release of Microsoft Windows) used a proprietary Windows-style interface. This interface, coupled to the sonic controls discussed in this paper, provided a level of sophistication that is impressive even when compared to contemporary developments in algorithmic music software.

The authors believe that the success of the Sound Globules project owes much to the brevity and clarity of the BON development method and its associated notation. The developers did not require any of the UML notations not provided by BON during the project. Textual description (achieved in UML by annotations) form an integral part of the BON method. The authors found that natural language clarifies and amplifies the diagrammatic notation, allowing BON designs to be terse, rigorous and expressive. In conclusion, the authors agree with the criticisms of UML provided by Paige and Ostroff (Paige, 1999):

“UML does not satisfy the single model principle. Information about a class need not be kept in one place; its contracts and invariants are written in OCL, and are not part of the diagram. Information about attributes that are not ‘simple’ is kept outside of the class. The semantics of a class may be given using a state machine. There is no single model for a class written in UML, and this may lead to consistency and communication problems as the class is reused or maintained, and as the system evolves.”

The author’s experiences question whether UML was a necessary development when further development of BON could have provided a more rigorous and expressive notation, coupled to a complete software development process.

The Sound Globules software is currently a functional prototype. Rigorous testing of existing code and completion of additional features is required to produce an alpha version of the software. The authors gratefully acknowledge the interest expressed by Dr. Russ Kozerski, the originator of Sound Globs. Thanks must also go to Hugh Anderson of the Wellington Institute of Technology for his assistance during the writing of this paper.

6 References

Ariza, C. (2005): *An Open Design for Computer-Aided Algorithmic Music Composition: athenaCL.* Boca Raton, Florida: Dissertation.com p.36

Eales, A. (1996): *Dances of Life:* Studies in Cellular Automated Music. M.Mus Thesis, Pretoria University.

Eales, A., Owen R (2005): Implementing Relationship Constraints in OO Programming Languages. *New Zealand Journal of Applied Computing* 9(1): 20 - 25

Generic Interpreter: Retrieved from http://www.csupomona.edu/~carich/gi/

IBM (2004) Retrieved from http://www.alphaworks.ibm.com/tech/sa4j

Kugel, P. (1990): Myhill’s Thesis: There’s More than Computing in Musical Thinking. *Computer Music Journal* 14(3): 12-25

Kozerski, R. (1989-91): Sound Globs 3.0 User Manual.

Kozerski, R. (2007): Private email.

Metcalfe, A. (2006): *Sound Globules Project*. Wellington Institute of Technology

MIDI Manufacturers Association (2001): *Complete MIDI 1.0 Specification v96.1* Los Angeles: MIDI Manufacturers Association

2001), Meyer, B. (1997): *Object-Oriented Software Construction*. Prentice-Hall.

Myhill, J. (1952): Some Philosophical Implications of Mathematical Logic: Three Classes of Ideas. *Review of Metaphysics* 6(2): 165-198.

Myhill, J. (1979): Controlled Indeterminacy: A First Step Toward a Semistochastic Music Language. *Computer Music Journal*, 3(3): 12-14

Miranda, E.R. (2001): *Composing Music with Computers*. Focal Press.

Miranda, E.R., Biles, J.A. (2006): *Evolutionary Computer Music*. Springer-Verlag.

NetBeans Matisse (2008): Retrieved from http://form.netbeans.org/

Paige, R. (1999): An Introduction to BON. Retrieved from http://www.cse.yorku.ca/~paige/Bon/bon.html

Paige, R. and Ostroff, J. (1999) A Comparison of BON and UML. *In Proc. UML'99, Lecture Notes in Computer Science, Springer-Verlag.*

Stravinsky, I. and Craft, R. (1958): *Conversations with Igor Stravinsky*. p.20. London : Faber and Faber.

Waldén, K., Nerson, J. (1994): *Seamless Object-Oriented Software Architecture – Analysis and Design of Reliable Systems.* Prentice Hall.

Waldén, K. (1998): Business Object Notation (BON): *Handbook of Object Technology,* ch10, CRC Press.

Full paper

An Educational Challenge: Ethnicity, Gender and Learning Styles

Dr Kay Fielden
United New Zealand
kfielden@unitec.ac.nz

Neil Comins
Unitec New Zealand
ncomins@unitec.ac.nz

Abstract

In this paper, results from a five-year study conducted with first year undergraduate computing students suggests that there is an educational challenge to be met in higher education. One hundred and eighty students from seven different ethnic backgrounds were tested for learning styles. The learning styles test was then considered against student final grades. Results from this study and some implications that can be drawn from these results are reported in this paper.

Keywords: Learning styles, computer education, ethnicity Introduction, gender

1 Introduction

A continuing educational challenge is to cater for students from a number of different ethnic backgrounds, and a mixed level of English achievement. Knowing how to teach effectively, knowing how to design effective curriculum, and knowing how to communicate effectively across this spectrum to meet a diversity of learning styles is an even greater challenge with the globalisation of computing education.

The five-year study reported in this paper is one attempt to identify what the factors, or combination of factors, are that need to be considered in facing this educational challenge.

The structure of this paper is as follows; firstly, the learning styles used in this study are defined; the seven cultural backgrounds are described; and the educational challenges presented are discussed.

In this small descriptive statistics sample, the research question posed, data gathering methods employed and data analysis techniques used are described. Next, findings from the data analysis are discussed. Implications for changes to curriculum design and teaching methods in first year computing education are presented and finally conclusions are drawn.

This quality assured paper appeared at the *21st Annual Conference of the National Advisory Committee on Computing Qualifications* (NACCQ 2008), Auckland, New Zealand. Samuel Mann and Mike Lopez (Eds). Reproduction for academic, not-for profit purposes permitted provided this text is included. www.naccq.nz

2 Definitions

The learning styles chosen for this study (visual, kinaesthetic and auditory) are defined below in Table 1. In this study there were seven different cultural backgrounds nominated by students. (This was a mandatory requirement for enrolment at the institution.)

2.1 Learning Styles

Learning styles tested in this study were visual, kinaesthetic and auditory (Table 1).

Table 1: Learning Styles - based on the test supplied by Wyman (2005)

Visual	Kinaesthetic	Auditory
Picture experience clearly - internal mental images	Feel the experience - internal feelings	Hear an experience - hear response internally
Sit at the front	Sit so they can get up & move	Sit where they can hear
Take detailed notes Like image-rich writing	Remember what is done- not recall what is said or seen	Acquire knowledge by reading aloud
Often close eyes to Visualise	Rely on what they can directly experience	Remember by verbalising to themselves
Watch something else if bored	Find reasons to tinker or move when bored	Hum or talk to themselves when bored
Can Visualise for long periods	Need to be active and take frequent breaks	Can listen for long periods
Benefit from illustrations and colour presentations	Enjoy field trips and tasks that involve manipulating materials	Benefit from audio presentations
Prefer Visuals to be isolated from Auditory and Kinaesthetic	Cooking, construction, art etc help them perceive & learn	Prefer to listen than see or feel
Communicate through writing	Communicate by touching and appreciate physically expressed encouragement – pat on the back	Communicate through talking
Have problems communicating to Auditories – they don't make eye contact	Believe that Auditory and Visual people are insensitive	Believe that Kinaesthetic people don't listen

2.2 Seven Cultural Backgrounds

All students are required to identify their ethnic background on enrolment. The more detailed responses have been grouped under the following: New Zealand European; Maori; Pacific Islands; Chinese; Indian; other Asian; and Other.

3 Educational Challenges

There are a number of educational challenges in introducing computing concepts at first year undergraduate level to a multicultural class. These include language skills, poor study skills, time limitations, conflict of interests, resources and lecturer's preferred learning style(s). Many of these can be directly, or indirectly related to the student's preferred learning style(s), with some examples described below.

The mix of ethnic background, new migrants, and international students means that there is a wide variety of English language skills in the classroom at any time. Those with poor skills are reluctant to participate in case they might be embarrassed by a mistake. The better skilled students are reluctant to "do all the work". This difficulty in fostering active discussions impacts upon auditory students, who need the talking while learning.

Poor study skills could be the result of not understanding what works best for them. This can result in a lot of time wasted with students attempting to learn using techniques suggested by others that are not appropriate. Frustration at poor success rates can often result in procrastination and poor preparation.

Resources and the lecturer's preferred learning style(s) can be related, in that the lecturer generally controls the forms of content delivery. An awareness of this resulted in attempts by the lecturers to expand the range of resources during the study, with minimal success. For instance, the non-auditory lecturers had difficulty in identifying and supplying auditory-type material.

4 Literature Review

Box (2003) considers the use of formative assessment and the feedback process as an approach to the learning of software development review and quality assurance skills. Box does not consider the forms of feedback required for students with different learning styles. As can be seen from Table 1 those with Visual learning styles need written feedback, students with auditory learning style require spoken feedback and kinaesthetic students benefit from a demonstration – especially when they have the opportunity to experience the feedback directly.

Butler & Morgan (2007) report on the learning challenges faced by novice programming students studying high level and low feedback concepts. Butler and Morgan suggest that concepts such as syntax are low-level cognitive tasks and that learning object-oriented concepts are high-level cognitive tasks. In their study in which student perceptions were measured, students believed that they received a high degree of feedback for low-level concepts and a low level of feedback for high-level concepts. Butler and Morgan did not discuss the implications for different learning styles. As can be seen by the comment about the Box (2003) study, there are major implications for the type of feedback given.

Choy & Delahaye (2003) suggest that youth learners are a neglected species in learning for an unknown future. Choy and Delahaye state that the generation of students considered in their research learn best by experience; they require support and feedback; want all work (including education) to be meaningful; prefer unstructured – but directed learning - and do not like being controlled. Choy and Delahaye also claim that this group of students were IT literate, and that the students have been conditioned through current social processes to expect immediate feedback. In their findings Choy and Delahaye state that youth-learning is complex and different for each student and that this provides a challenge for educators. Findings from this study suggest that more young people are using kinaesthetic learning styles and that this is different from the way in which educators are teaching and providing feedback.

In an in-depth case study of students' experiences of e-learning, (Conole, de Laat, Dillon, & Darby, 2006) changing patterns of learning were studied. Conole et al suggest that IT tools have changed the manner in which students gather information and communicate, and each student finds the best fit of technology for the way in which they learn. This study did not consider learning styles directly, but rather the findings suggest once again that kinaesthetic learning styles are being developed through students' desires to learn by experience. The study also implies that students with different learning styles adapt IT to suit their own needs for learning.

Gynnild (2003) studied student performance and needs in a learning situation, which was an empirical study. Gynnild researched the needs of students who perform well and those who perform poorly. Results of this study suggest that more differentiated teaching methods are required. Learning styles were not considered directly rather the assumptions made are that students with different learning styles require a variety of teaching methods.

Hawk & Hill (2000) in considering the transitions from one stage of schooling to the next, believe that feed forward is required for a seamless transition. It follows that first year undergraduate students in computing would also benefit from a variety of feed forward methods to cater for different learning styles.

One paper found in recent literature (Ho Teck & Leong-Wee, 2003) does consider directly learning styles. However, Ho Teck & Leong-Wee only consider one dominant style, they do not consider any mixed learning styles, nor do they consider the effect of gender and culture when combined with learning styles. Ho Teck & Leong-Wee (2003) consider only the effectiveness of case studies and suggest that case studies benefit all students regardless of learning style. This result may be flawed because of the simplistic nature of the way in which the analysis was performed.

In a case study of flexible delivery (Lee, Weerakoon, & Lingard, 2003) made a number of observations on student choice, approaches to learning and performance and