

7. Discussion and conclusions

Yet even the greatest theoreticians need data, and there is still much work to be done in musicology, psychology, and brain science. Then perhaps— just perhaps— music will become even more powerful than we have known it.

– Robert Jourdain: *Music, the Brain, and Ecstasy*

7.1. Introduction

The final chapter summarises the work described in this thesis and the results of the experimentation. It discusses some of the limitations of the work and how they might be overcome. It suggests areas for future investigation of the use of music in program auralisation and concludes by assessing the contribution of the thesis to the field of auditory display.

7.2. Summary of research carried out

The research described in this thesis addresses issues surrounding the musical representation of program events, and the way such techniques may be used to assist in the debugging process. The use of sound in the human-computer interface was given an historical perspective in chapter 1 in which several reasons were put forward for the relative paucity of sound-related research in HCI when compared with work in the visual field. Recent advances in sound-producing hardware and the wide availability of cheap electronic synthesisers have allowed the potential of using

sound in human-computer interactions to be explored. Some of the problems associated with the programming task were then outlined after which it was proposed that auditory feedback might assist in the programming process.

Chapter 2 discussed how sound has increasingly become a focus for HCI-related research and described in outline several exemplar systems and projects that illustrate the use of audio in different areas of HCI and computer science. Justification for the use of sound as an interaction medium in general was given, and particular emphasis was placed on showing how sound could be used in software visualisation tasks. The emergent research field of *auditory display* was described and a taxonomy of its member parts was suggested. Following this, summaries of the main avenues of past and current research in the field were made. The chapter concluded with a discussion of how sound has been applied in the programming domain using a technique known as *program auralisation*. Examples of existing systems were given and limitations of those systems were discussed.

Chapter 3 put forth a case for the use of music as a communication medium. The majority of auditory display applications rely on general non-speech audio and very little attention has been paid to the formal structures of music as a means of communicating information. We discussed the main features of music in relation to its ability to communicate highly-structured information. Arguments against the use of music were noted and responses given. Following this, we described some of the previous attempts to use music (both formally and informally structured) to communicate information in fields such as chemistry, biology, assistive technologies and chaos theory. The chapter concluded with a proposal for the use of music in the field of program auralisation and debugging.

In chapter 4 we described how a system to generate musical auralisations of Pascal programs was designed and constructed. The CAITLIN musical-program auralisation system was developed especially for this research. CAITLIN allows Pascal construct-related events to be mapped to the auditory domain via a consistent musical framework. This results in a run-time representation of program behaviour and, to a limited extent, program state. The notion of *points-of-interest*, or those features that are of interest to a programmer, was introduced. We then described how the points of interest of the Turbo Pascal constructs could be mapped to musical signature tunes, or motifs. The motifs for the various Pascal constructs were designed us-

ing hierarchic principles centred around a taxonomy of the language constructs. The rest of the chapter describes how CAITLIN's auralisation techniques were tested heuristically via a series of three small pilot experiments. CAITLIN was designed with the following principles in mind:

Usability: the system should not be difficult to learn, and so the interface mimicked that of the Borland Turbo Pascal I.D.E.

Musical framework: the auralisations should be designed around a consistent musical framework. Aspects of Pascal language features (such as construct hierarchy) were modelled in the musical representations.

The motifs discussed and evaluated in chapter 4 were semi-formal in terms of design and musical structure. In chapter 5 we reviewed issues surrounding the cognitive aspects of music and its use in the interface and discussed existing guidelines for the use of music in HCI and considered these in the light of work done in the music cognition and music-theoretic fields. Following this we proposed a set of organising principles that would allow the construction of auralisations designed within a formal musical framework. The chapter then described a listening-test experiment that evaluated how novice programmers perceived and recognised the auralisations. The results indicated how successful the musical mappings of program events were. It was found that after very little training the novices could achieve recognition rates of the construct auralisations that were significantly higher than could be obtained by guesswork alone. The experiment also showed how some of the cognitive principles that were set out in the organising principles played an important role in subjects' ability to comprehend the auralisations. We also discussed the role of context in listening to auralisations and how this experiment was testing subjects for whom no context of the auralisations was given.

Chapter 6 described an experiment that was designed to assess how the auralisations could be used by novice programmers to help locate bugs in pre-written programs. The context that was lacking in the previous experiment was provided in this study by the addition of full sets of program documentation. The results of the experiment showed that, within certain constraints (such as ability of the programmer, the complexity of the programs being debugged, and the output clues given by the faulty programs) the musical auralisations were useful.

7.3. Limitations of the research

We have shown that musical motifs can communicate information about program structure and that the auralisations can be interpreted at different hierarchical levels. Further, we have shown that the auralisations can, under certain circumstances, assist novice programmers in locating pre-planted bugs in short Turbo Pascal programs. The CAITLIN system was intended to be a complement to existing visualisation techniques. Ultimately one would wish to construct a programming and debugging environment in which the user has full control over the application of visualisation and auralisation techniques. Alty and Rigas [5] described this as an *equal-opportunity interface*, that is, an interface that makes no prior judgement about the capabilities of the user population as regards use of different input/output modalities. Such an interface would offer “... a variety of communication media, from which the user can select an appropriate mix to match their capabilities and limitations” [5]. Because this research was investigating the role of music as a communication medium and because there is so little prior research into this field, it was beyond the scope of this thesis to investigate multi-modal feedback for debugging. Further work would investigate ways in which music, other non-speech audio, and visual display techniques could be combined.

This research focused on the results of two experiments: a listening test and a bug location test.

7.3.1. Listening test

In the listening test (described in chapter 5) we aimed to assess how well novices could recognise the auralisation motifs. We showed that in the absence of the contextual information provided by program documentation, the motifs were recognisable. The research did not explore the cognitive latency of the musical structures. It was demonstrated that the motifs could be retained in memory without any noticeable degradation over a fifteen- to twenty-minute period (the length of the break between experimental tasks). Work in the music cognition field shows that tunes are memorable under certain circumstances and so we hypothesise that the motifs (which are tonal rather than atonal compositions) would be retainable after sufficient training. Once the motifs become as well-known to the listener as, say, a nursery rhyme, then we can envisage no reason why the motifs should not enter into

long-term memory. Naturally, longitudinal studies would be able to test this hypothesis.

Another feature that was not investigated was the tempo range in which the motifs remain recognisable. In the experiment, all motifs were played at the same tempo (140 beats-per-minute). Further work needs to be undertaken to assess the limits at which auralisations can be played. This is important, as program auralisations that take a long time to play may well cause frustration. Some subjects did comment that certain of the auralisations in the experiment took too long.

7.3.2. Bug location study

In the debugging experiment subjects were asked to locate a single bug in each of eight pre-written programs. The auralisations were designed using the same criteria as for the listening test. Consequently, only program information related to the language constructs could be communicated aurally. Future work needs to be done to extend the auralisations to the entire domain of program events and experimentation done to assess the usefulness of the auralisations in assisting in the removal of both program-flow-related bugs (as in this study) and **non**-program-flow-related bugs (beyond the scope of this research).

The debugging tasks themselves could be criticised for being unreal as subjects were not debugging their own programs. However, given that the majority of programming effort world-wide is in software maintenance, the results are still valid.

The debugging experiment aimed to test the general hypothesis: *The musical program auralisations generated by CAITLIN can assist novice programmers in locating bugs that manifest themselves either directly or indirectly in terms of program flow.* From this general statement five specific hypotheses were specified:

1. Do subjects locate more bugs with the additional auralisation information than without?
2. Do subjects locate bugs faster or slower with auralisations?
3. Does the musical experience of subjects affect their ability to make use of program auralisations?
4. What effect do the auralisations have on subjects' perceived workload?
5. Do subjects find the auralisations annoying?

7.3.2.1. Bug location

The experiment showed that bug location was improved by the auralisations for programs that exhibited one or both of the two characteristics:

- High complexity (relative to program size)
- Output that offered few clues as to nature of the bug

There was no evidence that those programs that did not meet these criteria benefited from the addition of auralisations. Furthermore, it became apparent that some subjects found the debugging task itself too challenging, in which case auralisation would lend no assistance, whilst others appeared to find all the problems too easy to be able to benefit from the auralisations. A future experiment would aim to address these limitations by:

- Ensuring that all subjects met a minimum standard for debugging ability
- Providing a series of harder problems (that would probably require longer to solve).

7.3.2.2. Debugging speed

We found no evidence to suggest that the auralisations either sped up or slowed down the bug location process. Further studies would be able to either confirm this to be the case, or else, show why this experiment did not demonstrate an effect.

7.3.2.3. Musical skill

No evidence was found to support the notion that the auralisations require users to be musically trained either in the listening test or the bug location experiment. This is encouraging as one of the principal motivations behind this research was to demonstrate that music could be used as a communication medium regardless of musical skill. Unfortunately, the experiment did not obtain subjective feedback from the subjects on how they perceived the auralisations.

7.3.2.4. Workload

The results of the study showed a highly significant increase in workload when working with the auralisations. Because the subjects were encountering musical auralisation for the first time, it is possible that their lack of familiarity led to the increased workload. A longitudinal study in which subjects are taught the principles and practice of program auralisation over several weeks would be a better measure

of whether workload is different when using auralisations in the programming and debugging processes.

7.3.2.5. Annoyance

The results of the annoyance assessment were ambiguous with an approximately bimodal distribution between those who found the auralisations annoying and those who did not. Again, it might be that those who found them annoying did so because of lack of familiarity, whilst those who did not were those for whom the auralisations had little impact or benefit. Again, a longitudinal study would yield more conclusive results.

7.3.3. Organising principles

In addition to the two experiments, we proposed a set of six organising principles for the incorporation of music in program auralisation. The experimental results suggest that these principles are valid. However, two issues remain. First, as a full program-event-domain mapping was not realised in this research, it is possible that the organising principles would need to be refined or extended in the light of the results of future studies (as outlined above). Secondly, the guidelines have been developed in the light of studies into auditory display, music-cognition and music-theory. This requires an auralisation designer to have some musical training to be able to make use of the principles. This problem could be mitigated by the development of a toolkit that future auralisation designers could use to ensure that their motifs adhered to sound musicological and music-cognitive principles.

7.4. Future work

This research has yielded some interesting and useful results about how music can be used as a communication medium in the field of computer programming. The experiments have led to the identification of further avenues of exploration.

7.4.1. Comprehensive auralisation system

First of all, a comprehensive musical program auralisation system needs to be constructed to allow the mapping of all relevant program domain events to musical events. The CAITLIN system serves as a useful starting point for such a system and

should be extended to allow the other program features (particularly sub-program calls) to be auralised. Of course, there are difficulties here with the mapping – how do you signal identity of a sub-program given that there is an unlimited number of possible procedures and functions? One way forward would be to create generic motifs, one for function calls and one for procedure calls and leave the task of exact identification to the contextual information provided by the program. The role of non-musical audio also needs to be investigated to find the best applications for each type of auditory display (music, non-speech audio, and speech).

It would be useful to be able to speed up and slow down the playback speed so that the programmer can ‘fast forward’ through certain sections of code and focus in detail on other areas (effectively changing the level of abstraction). DiGiano and Baecker believe “*that the capability to play back programs at different speeds...is key to deriving meaning from auralisations*” [53]. The facility for the tempo to be controlled in real-time by the user should be added to a future system. Bock’s ADSL system has a similar function [23].

Another technique that Bock uses is the acoustic analogue of the graphical zoom-in and zoom-out. Many visualisation methods allow a graphical representation to be seen as an overview by ‘zooming out’ of the picture. In essence, the data is highly quantised to give low resolution and thus a wider view. This is much like a map with a large scale. Likewise, zooming in (or using a map with a smaller scale) allows narrower sections of the data to be viewed by increasing the resolution and focusing on the details.

There are several ways of achieving this acoustically. Bock’s system uses ‘tracks’ that define which language constructs will be auralised. Jameson’s Sonnet system allows a threshold to be specified on loop iterations so that it is not necessary always to hear every repetition of the loop [86]. In both of these systems the programmer can also specify sections of the program code that will or will not be auralised.

A problem of scale arises when considering how best to auralise a program. Even for a short program more than a few iterations of a loop can generate a lot of program domain events. To auralise every occurrence of every event may not always be helpful. In their work on deriving music from chaotic systems Mayer-Kress et al. [112] commented that the “... *design challenge is to display the rate of change of the*

system such that the local detail does not prohibit the perception of larger dynamical structures”.

7.4.2. Longitudinal study

The motivation behind this research was to construct a system for musical program auralisation and to evaluate its outputs in terms of comprehensibility and usefulness in the debugging process. Because no prior evaluations of this kind existed, it was beyond the scope of this project to carry out full user studies of the CAITLIN tool. A major piece of work that could be carried out is a longitudinal study of a set of novice programmers. Such a study would involve training the novices in the use of the CAITLIN system alongside basic instruction in computer programming. Assuming relevant control samples were used, such a study would be able to find out what effects training and familiarity with the technique have on subjects' ability to program and debug. Such a study could be run concurrently with an introductory course in Pascal programming.

7.4.3. Program classes and users

The experiments in this research suggested that certain classes of program are more amenable to program auralisation than others. Studies should be undertaken to test whether programs of relative complexity or programs that produce output that contains few clues do benefit from auralisation. Additionally, such experiments should restrict themselves to subjects who can pass a rudimentary debugging aptitude test.

7.4.4. Equal opportunity interface

We have provided a complementary modality that goes some way to fulfilling the aim of an equal-opportunities interface [5] for the programmer. We do not propose that auralisation necessarily be used exclusively, but that it offers an additional tool to the large suite of visualisation aids presently available. The organising principles should allow musical frameworks to be developed for more general interface tasks and so future projects could use these principles to assist in construction of a true equal opportunity interface as envisaged by Alty and Rigas [5].

7.4.5. Program auralisation for the blind

King and Angus [89] stated: “*audification will never take over from visualisation*”. Whilst this may be true for sighted users it does not offer much hope for the blind community. It is imperative that further work be carried out to establish just what is and is not possible with regard to audio-visualisation for blind users. Although this research did not address the needs of blind and visually-impaired programmers, the results suggest that a musical program auralisation system could be applied to that branch of assistive technology. Full user studies would need to be undertaken, but given that existing sighted programmers have shown that auralisations can communicate program information in the absence of any context whatsoever, it is not unreasonable to hope that the system can be adapted and extended for use by the blind.

7.5. Conclusions and contribution of this thesis

Prior to this research there was little evidence to support or discount program auralisation as a useful tool. Previous auralisation systems had been published without empirical evidence to prove their efficacy. Bock [23] went some way towards measuring the contribution of auralisation to debugging, but failed to compare performance with a control group. As such his work provides anecdotal evidence that program auralisation is useful, but nothing concrete. Designers of other program auralisation systems further demonstrated the mechanics of integrating non-speech audio into the programming tasks. Limitations notwithstanding, this research is an important contribution to the fields of auditory display and human-computer interaction because it offers the first empirical evaluation of program auralisation.

Furthermore, although much has been made in auditory display of the use of non-speech audio, very little work has focused on using musical structures and frameworks as a basis for communication. Some recommendations for the incorporation of musical sounds have been put forth but without attention to music-cognitive and music-theoretic principles. In this thesis we have proposed a set of organising principles for musical auralisations that are based upon both auditory display research and formal musical studies. These organising principles provide guidelines for the construction of motifs which are based upon sound musicological foundations.

The development of musical program auralisation is a major contribution to the field. Prior to this research program auralisation had not been evaluated and relatively little exploration of musical frameworks for communication had been carried out. Following this work, we now have a foundation upon which to carry out further and more detailed investigations of how far the technique can be taken. The research showed that hierarchic motifs were useful as a communication medium and that the information conveyed by them could assist in the location of bugs. The organising principles can be followed to ensure that motifs are discriminable, even to those without formal musical training or specific musical knowledge and skills.

Prior auralisation systems allowed programmers to create ad-hoc auralisations of their code. The various systems allowed use to be made of musical pitches, but in an ad-hoc manner. If a musical framework is to be used then it cannot, by definition, be ad-hoc. There are complex cognitive and musicological interactions that affect our perception of the music we hear. This research offers the community a set of guidelines that allow the construction of musically coherent motifs. From these guidelines, it would be possible to create different motif sets for different environments. For example, one might envisage the need for motifs based on classical music styles, jazz, rock or even ethnic and world-music forms. Pre-defined motifs free the user from having to learn complex scripting languages and music theory in order to auralise a program.

The major strength of this work over prior program auralisation systems is that it is based on empirical results. The thesis set out to address the question of whether music can be used as a communication medium to assist novice programmers with debugging tasks. To answer this question we constructed a program auralisation system whose motifs were structured according to a formal musical framework. This resulted in a set of guidelines for the construction of further motifs. The experiments showed that the motifs can be comprehended at various levels of abstraction and that, within certain constraints, they could be used to help novice programmers locate bugs in short programs. No musical experience was necessary, indeed, those with musical training fared no better on the experimental tasks. The way the musically inexperienced reacted to the auralisations demonstrated that careful use needs to be made of surface and deep musical characteristics alike. Surface attributes help to reduce the learning curve of the system and provide immedi-

ate discrimination ability. The deep characteristics are necessary to provide differentiation for experienced users and the musically-skilled who tend to look beyond surface transformations of motifs for clues about what they are hearing.

In summary we have showed that music can convey information about program events. Secondly, we suggest that it can play a complementary role in the programming process, particularly in the location and diagnosis of bugs. Future work should attempt more precisely to define the relationship between auralised and non-auralised debugging with a view to creating a full auralised programming environment.