Musical Program Auralization: Empirical Studies

PAUL VICKERS
Liverpool John Moores University
and
JAMES L. ALTY
Loughborough University

Program auralization aims to communicate information about program state, data, and behavior using audio. We have argued that music offers many advantages as a communication medium [Alty 1995]. The CAITLIN system [Alty and Vickers 1997; Vickers 1999; Vickers and Alty 1996, 1998] was constructed to provide auralizations within a formal structured musical framework. Pilot studies [Alty and Vickers 1997; Vickers 1999] showed that programmers could infer program structure from auralizations alone. A study was conducted using 22 novice programmers to assess (i) whether novices could understand the musical auralizations and (ii) whether the musical experience and knowledge of subjects affected their performance. The results show that novices could interpret the auralizations (with accuracy varying across different levels of abstraction) and that musical knowledge had no significant effect on performance. A second experiment was conducted with another 22 novice programmers to study the effects of musical program auralization on debugging tasks. The experiment aimed to determine whether auralizations would lead to higher bug detection rates. The results indicate that, in certain circumstances, musical auralizations can be used to help locate bugs in programs and that musical skill does not affect the ability to make use of the auralizations. In addition, the experiment showed that subjective workload increased when the musical auralizations were used.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Auditory (non-speech) feedback*; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*Audio input/output*; H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing—*Methodologies and techniques*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging aids*; D.2.6 [**Software Engineering**]: Programming Environments

General Terms: Human Factors, Languages

Additional Key Words and Phrases: Music, auralization, debugging, Pascal

1. INTRODUCTION

Within the field of auditory display, program auralization, or the mapping of program data to sound, has attracted increasing levels of interest. Brown and Hershberger [1992] were among the first to suggest using sound to aid the visualization of software. Jameson's Sonnet [Jameson 1994], Bock's Auditory Domain Specification Language (ADSL) [Bock 1994] and the LISTEN Specification Language (LSL) [Boardman et al. 1995] developed the idea. Jameson developed the Sonnet visual programming language to add audio capabilities to a debugger, while ADSL and LSL added audio to programs at the preprocessing

Authors' addresses: P. Vickers, now at Northumbria University, School of Computing, Engineering, and Information Sciences, Pandon Building, Camden Street, Newcastle upon Tyne, NE2 1XE, UK +44 (0)191 243 7614; email: paul.vickers@unn.ac.uk; J. L. Alty, Loughborough University, Dept. of Computer Science, Loughborough, Leics, LE11 3TU, UK +44 (0)1509 222648; email: j.l.alty@lboro.ac.uk.

stage. However, formal evaluation has been sparse. We constructed the CAITLIN[1] system to study the effects of musical program auralizations on the debugging tasks of novice Pascal programmers.

The CAITLIN auralizations have a musical framework as music offers much as a communication medium [Alty 1995]. Pilot studies [Alty and Vickers 1997; Vickers 1999] suggested that music can convey information about program behavior. In this paper, we describe two experiments. The first aimed to discover whether musical auralizations could be understood and interpreted by novice programmers. The second study explored how the CAITLIN system could be used to assist bug location. Details of the CAITLIN musical auralization approach are given elsewhere [Alty and Vickers 1997; Vickers 1999; Vickers and Alty 1996, 1998]. In summary, auralizations are effected by mapping the constructs of a Pascal program (IF, CASE, REPEAT, WHILE, and FOR) to short musical tunes (motifs). The key aspects of a construct (points of interest), such as entry and exit and evaluation of Boolean expressions are assigned a motif whose content is consistent with the structure and harmony of the other motifs in the construct. A hierarchic approach was taken to the motif design to allow the taxonomy of the Pascal constructs to be maintained [Alty and Vickers 1997].

2. MOTIF RECOGNITION STUDY

This study investigated how novice programmers interpreted musical auralizations. In addition, as the system is intended for use by programmers regardless of musical expertise, it was hoped that the study would show no significant difference in performance across subjects with varying levels of musical knowledge and experience.

2.1 Subjects

Twenty-two undergraduate students on computing courses at Loughborough University took part. Twenty-one subjects were male, although this sex imbalance is typical of undergraduate computing courses in the UK. The mean age of the subjects was 21 (min. 19, max. 23). The musical experience and knowledge of the subjects were assessed by means of a questionnaire. Subjects had an average of 2 years' programming tuition and all had written programs in Pascal. On average, the subjects had experience of five programming languages (min. 2, max. 7).

2.2 Task

The study comprised two tests. Test 1 required subjects to listen to and identify forty construct auralizations. Test 2 presented pairs of auralizations that were either nested or sequential. In all, sixty construct auralizations were used in the two tests. One-half the auralizations were of iteration constructs, and one-half were selections. Auralizations were of differing length, depending on the construct type. They were generated by the CAITLIN system using a Boss DS-330 multitimbral synthesizer and were played to subjects over a stereo amplifier and speakers in a tiered lecture theatre. The volume was adjusted so that the music could be heard well at the back of the room. The speakers were placed close together to mask any stereo effects. Subjects recorded their response by ticking a box on a chart, each box representing a specific construct (see Figure 1). In test 2, there were two box charts per exercise and an additional box in which subjects could state whether the pair was sequential or nested. The auralization technique was explained and examples given. A practice test was run in which 20 auralizations were played, each three times, during which time subjects wrote down what construct they thought was being played. Immediate feedback was given so the subjects could check their answers. Following this, subjects completed the questionnaire that was used to gather information about their musical background and programming experience.

[1]See www.auralisation.org for more details.

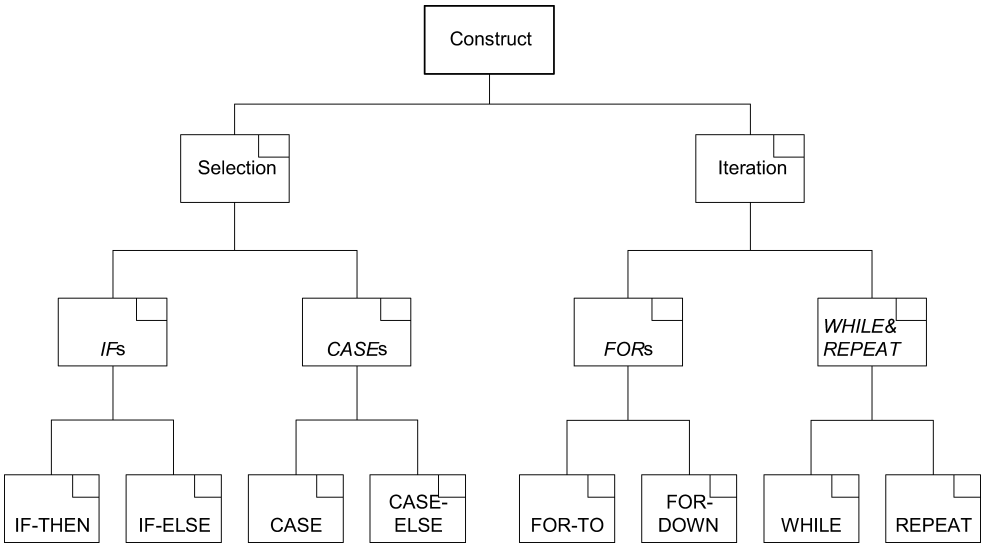

Fig. 1. Construct identification worksheet.

2.3 Scoring

Because of the hierarchic nature of the constructs and the corresponding auralizations, the worksheets allowed subjects to give different levels of answer. For example, if a subject thought an auralization represented a `FOR-TO` loop then they would tick the box labeled "`FOR-TO`." This is the most precise level of identification which we call *specific identity*, that is, precise construct identification is achieved. If the subject could identify the loop as a `FOR` but was unsure whether it was a `FOR-TO` or a `FOR-DOWNTO` then they could tick the higher level box labeled "`FOR`s." This is identifying the construct at its subclass level. The lowest level of identification (class level) is where, in this instance, the subject can identify the auralization as a loop as opposed to a selection, but cannot be more specific. This would involve the subject ticking the "Iteration" box on the worksheet.

In the first test, subjects had to tick the box on the worksheet that they thought matched the auralization being played. Each was played three times, and comprised ten unique selections and ten unique iterations. These were randomly arranged into two sets of twenty, each unique auralization occurring once per set. The second listening test was similar to the first: 20 auralizations were presented in ten pairs. The reason for playing pairs of auralizations was to see whether subjects could correctly discriminate between sequential and nested constructs. That is, would the auralizations allow subjects to hear the difference between say, an `IF` nested, within the body of a `WHILE` loop and a `WHILE` loop followed by an `IF`? Twenty unique auralizations were used, ten iterations and ten selections. Five pairs were nested, while the other five were sequential. Each pair of constructs was played three times.

2.4 Results

Scores could be correct at one of three levels. If a construct was correctly identified, then a specific identity score was given. A subclass score was given if the subject either ticked the correct subclass box (e.g., "`IF`s" for an `IF` or an `IF–ELSE`), or mistook a construct for its sibling (e.g., confusing an `IF` for an `IF–ELSE`). A class-level score was given if a subject ticked the correct class box (e.g., "Iterations" for a `WHILE`) or ticked the wrong subclass level box (e.g., "`FOR`s" for a `WHILE`—both are iterations), or ticked the wrong specific identity box (e.g., confusing a `FOR-TO` for a `WHILE`).

In the first listening test, the mean identification of specific identity by subjects was 46% A further 30% of subjects achieved identification at the subclass level (either deliberately or by identifying a construct as its sibling), and another 21% at the class level. This gives an mean absolute misidentification rate of 3%, that is, 97% of subjects managed to identify constructs at at least one of the three identity levels. Of the iterations (22 out of 440 iteration observations), 5% were misidentified as selections. None of the selections was incorrectly identified as an iteration.

In the second test, ten pairs of auralizations were played to 21 subjects (subject 15 did not participate). Five of the pairs were of nested constructs, while the other five of sequential constructs. The results are comparable with those for the first listening test, there being no significant difference between the two in terms of auralization identification rates ($specific\ identity = 49\%; subclass = 35\%; class = 14\%$). It is interesting that similar results were obtained even though subjects were required to identify two constructs-per-exercise rather than one. In one-half the cases, the constructs were nested, which meant that subjects were listening to interleaved auralizations. That this did not impair performance is encouraging. Furthermore, subjects scored high, in identifying the nestings and sequences ($mean = 97.5\%$).

2.5 Discussion

One objective was to see whether subjects' musical skill influenced their performance. The questionnaire measured four related (nonorthogonal) variables: level of interest in listening to/performing music, experience of playing an instrument, level of singing expertise, and score on a musical knowledge quiz. The data from the questionnaire were analyzed by multiple linear regression to see whether any of the factors influenced the scores. For each of the two listening tests, the music data were analyzed for their effect on the correct scores at the specific identity and subclass levels. The results indicate that musical knowledge and experience had no significant effect on subjects' ability to recognize constructs at the specific identity or subclass levels. In fact, the best predictor of a subject's score is a value approximating the mean of the sample.[2]

The mean specific identity scores were 46 and 49% in tests 1 and 2. If subjects merely guessed the specific identity of the constructs, then we could expect one out of every eight responses (12.5%) to be correct by chance alone, there being eight constructs from which to choose (Figure 1). Also, we would expect one-half the specific identity scores to be greater than 12.5% and one-half to be less. For test 1, three specific identity scores (no.'s 7, 19, and 31) were less than 12.5%; the remaining 37 were higher. This is a highly significant difference ($\chi^2 = 15.62$, 1df, $p < 0.01$). Considering the lack of confusion between iteration and selection classes, a more conservative approach would be to assume, for any given auralization, that subjects knew whether it was an iteration or selection. This reduces the choice to one among four. In this case, we observe that seven exercises had specific identity scores less than the 25% attainable by guessing. Again, we see that these scores are still significantly higher than would be obtained by guesswork alone ($\chi^2 = 8.05$, 1df, $p < 0.01$).

Separate analysis of the selections and iterations reveals where the main difficulties with identification lay. If we again assume that subjects could distinguish between iterations and selections (recall that identification at the class level was 97.5%), then for any of the twenty selection auralizations, subjects could get one in four of them correct by guesswork. Thus, we would expect one-half (ten) of the specific identity scores to be greater than and one-half to be less than 25%. In fact, six of the twenty selections had scores $\leq 25\%$ at the specific identity level and fourteen had scores higher. This difference is not significant ($\chi^2 = 0.94$, 1df, $p > 0.1$). For the iterations, there is no evidence for guesswork ($\chi^2 = 5.83$,

[2]For test 1, specific identity scores, the constant term (17.414) in the resultant regression model ($y = 17.414 + (0.225 \times musical\ score) – (2.531 \times play) + (1.063 \times interest)\ (.840 \times sing)$) is the best predictor of a subject's score ($p = 0.057$). None of the four factors is a significant predictor of correct score ($p = 0.740 \ldots 0.893$).

1df, $p < 0.02$). The six selection constructs with the low scores were the IF–ELSE and CASE–ELSE, suggesting that the motifs for the selection constructs were not sufficiently different. However, we must be cautious not to conclude that guesswork alone was involved in selection identification. A low score does not necessarily mean that subjects incorrectly identified a construct. For instance, in exercise 6, five subjects only attempted identification at the subclass or class levels.

Furthermore, subjects were not faced with just eight choices, but could select higher nodes in the hierarchy charts. Analysis of the subclass scores further suggests that more than guesswork was involved. A construct could be identified at its subclass level by one of two ways: either the subject deliberately selects the subclass box on the score sheet, or, the subject misidentifies the construct as its sibling, resulting in a subclass identification by default. In test 1, of those observations scored correct at the subclass level (261 in all), 59% were deliberate, 41% by default. This means that of those constructs identified at the subclass level, just under one-half were as a result of confusing a construct with its sibling. We may conclude that subjects generally understood what they were hearing, but that at the level of specific identity, construct motifs were still not distinct enough from their siblings. Where there is evidence of guesswork is in the disparity between mistaken identification of iterations as selections and vice versa. Recall that no selections were misidentified as iterations, while 5% of iterations were incorrectly identified as selections. This suggests that, when in doubt, subjects assumed the construct to be a selection. There is evidence of this practice. Subject 7 wrote: *"...The loops, especially nested, were easier to spot and so anything that wasn't, I assumed to be a selection."*

2.5.1 *The Role of Context.* The information presented to the subjects in this experiment was without context. That is, subjects had no domain information to help them understand the problem. Alty and Rigas [1998] identified the importance of context for blind users using a musical diagram reader. They identified three levels of perception that are important when using auditory interfaces: detectable mapping, perceptual context, and reasoning. These may be rephrased as uniqueness, metaphorical, and semantic level [Vickers and Alty 1998]. Uniqueness (or detectable musical mapping) refers to a construct auralization's ability to be uniquely identified and not confused with another. The metaphorical level (or perceptual context) is where, given a detectable mapping, the motif creates expectation of the part of the listener. The motif is interpreted in domain terms and meaning can be ascribed to it, although the listener may not necessarily be able to reason about the global interactions. The semantic level (or reasoning level) is where the listener develops high-level structures in the mind to understand the domain from a higher, or more abstract, viewpoint.

According to Alty and Rigas [1998] the production of unique mappings is necessary but insufficient for a successful auditory interface design; the contribution of the metaphoric and semantic levels is needed. In the program auralization domain, the construction of unique motifs provides mappings between the program domain and the auditory domain at the uniqueness level. At this level, little meaning is attached and all that is heard is a collection of different tunes. At the metaphoric level, the listener interprets the audio messages in the context of the other domain messages, thus setting up expectations. Here, users would recognize the tunes as constructs. Metaphors are built in the mind of the listener allowing recognition of the rising and falling tune as an IF statement, or the pleasing flute melody as a FOR loop. Once this level is achieved, expectations then arise. On recognizing a WHILE loop, the listener would expect to hear the tunes representing the various points of interest of the WHILE statement, that is, one or more major or minor chord devices followed by the tune signifying the end of the construct.

At the semantic level, listeners begin to form gestalts and start to reason about what the audio messages mean. If the construct is a WHILE statement, meaning that it is a loop, we would then expect to hear one or more evaluations of the controlling condition and the consequent execution of the

statement block when the condition is true. Further, within the statement block (and before the closing point of interest is sounded) we would anticipate hearing other statements (either simple statements represented by a single percussive sound, or further constructs with their own motif sequence). The listener is no longer simply hearing a set of different and recognizable tunes, nor even a collection of constructs, but a program and its various branches and interactions. Alty and Rigas claim that the level of mental activity required to perform at this level is likely to increase the memorability of the interface. In an experiment using the diagram reader [Alty and Rigas 1998], they found that subjects' accuracy greatly improved if a contextual clue regarding the nature of the picture being described was given. For instance, the outline of the letter E was recognized most by subjects who were told that the diagram was a letter; subjects who were given no clue about the picture scored much worse.

This experiment is, in effect, a test of subjects' short-term memory and their ability to discriminate between some subtle and some gross differences in motifs. The best level of perception that might reasonably be achieved by this experiment is the metaphoric level, that is, subjects would start to hear auralizations not as separate tunes, but as constructs. If subjects do not move beyond the uniqueness level, then the test is simply one of memory and the subject's ability to correctly repeatedly assign the various unique tunes to one of eight possibilities.

In a real programming and debugging situation, subjects would be listening to auralizations at the same time as having access to the program source (even blind programmers would have some textual/verbal representation of the code). The extra context provided by the presence of other constructs and understanding of the aims and structure of the program would allow users to move beyond the metaphoric level into the semantic level. Therefore, that specific identity scores are not high is not of concern as we would expect the added context of the program source to fill in the gaps. Of course, it would be preferable to specify the auralizations such that they are identifiable regardless of context.

2.6 Conclusions

In programming, we are not really trying to identify a construct's place on a taxonomy chart. Programmers are not interested in this. The programmer will already know from the source code what constructs have been used and whether or not the various selections have `ELSE` branches or not. The auralizations simply have to be recognizable within a given context. What is of interest is the structure of the program under consideration, not the categorization of the various language constructs. Of course, such a hierarchic design should allow a programmer to listen to an auralization at varying levels of abstraction. To understand program flow, it may be enough simply to hear that there is some form of selection here, some form of unbounded loop there—exact details can be gleaned from the listing. With training and continued use, users may become adept at distinguishing between all the construct auralizations even without contextual clues. It remains to study how the auralizations can be used in debugging tasks. The following sections describe such an experiment and discuss its results.

3. PROGRAM DEBUGGING STUDY

We showed above that the `CAITLIN` auralizations can communicate information about Pascal programs. The motivation behind this study was to explore the effects of musical program auralization on program debugging by novices. Auralizations offer a theoretical advantage over static program analysis techniques (flow graphs, etc.), because they are immediate and can be generated during program execution. Therefore, it could be hoped that immediate musical representations of program execution would assist novices with debugging. It was also hoped that no significant difference in performance across subjects with varying levels of musical knowledge and experience would be seen.

3.1 Subjects

Twenty-two undergraduate computing students at Loughborough University took part in the experiment These were not the same subjects used in the first experiment described above. Nineteen subjects were between 20 and 29 years old, three were 19 years old or younger. All but one reported a western-style cultural upbringing. Seventeen subjects had 2 or more years of programming tuition, while the other five had at least one year's tuition. On average, the subjects had experience of four programming languages (min. 2, max. 7). The subjects had no prior experience of program auralization and so a tutorial (with examples) explaining the philosophy of the technique was given.

3.2 Experiment

The two main questions addressed by this research were:

- Do subjects locate more bugs with the additional auralization information than without?
- Does the musical experience of subjects affect their ability to make use of program auralizations?

The experiment comprised eight different debugging exercises, numbered A1 to A8. For each, subjects were given a specification for the program, a pseudocode program description, sample input data, the expected output and actual output data. Each program was syntactically correct and contained a single logical error (bug).

Subjects were allowed as long as they needed to read the program documentation. When ready to locate the bug, they were given the program source code and had a maximum of 10 minutes to locate the bug. The eight tasks were performed in order beginning with A1. One-half the tasks had accompanying auralizations. The treatment (auralization) was given to alternate tasks. One-half the subjects had the treatment applied to tasks A1, A3, A5, and A7; the other one-half had the treatment applied to tasks A2, A4, A6, and A8. Thus, every program was tested in both the treatment and nontreatment states so that any differences between programs (such as complexity) would not influence the results. As each subject performed tasks in both states, any differences between the subject groups were balanced, although analysis of the data (see below) showed that the experiment still contained a bias despite the attempts to balance it.

The tests were administered by a specially written web application that collected timing information, questionnaire responses, NASA TLX scores (a measure of workload as perceived by the subject [Hart and Staveland 1988; NASA 1987]), and subject protocol data. All subject data were stored in data files on the host server.

The auralizations were prepared prior to the session and stored as MPEG-1 Layer 3 (MP3) audio files. The dynamic range of the music was not great, so a sampling rate of 22.05 KHz and a bit-rate of 40 Kbit/sec were used (near-CD quality, MP3 requires a 44.1 KHz sampling rate and a bit-rate of 128 Kbit/sec). This allowed the files to be compressed to about 10% of their original size (2.2 instead of 22 MB in the case of the largest file). The application used an embedded Windows Media Player control, which gave subjects facilities for playing, stopping, pausing, rewinding, and forward-winding the auralizations. `CAITLIN` allows auralizations to be played at different tempos. This feature was not available in the experiment as preprepared auralizations were used. Therefore, for each exercise, two auralizations were provided: one at 120 beats-per-minute (slow) and one at 140 beats-per-minute (normal). Either auralization could be accessed by pressing an appropriately labeled button.

3.3 Bug Types

The current mappings between program and musical event domains necessarily restrict bug types that can be investigated with the present `CAITLIN` system. As the system does not address assignments or

Table I. Raw Data Scores

Subject	A1	A3	A5	A7	A2	A4	A6	A8	Auralized	Normal
102	1	1	1	1	1	1	1	1	4	4
104	1	1	1	1	1	1	1	1	4	4
106	1	1	1	0	1	1	1	1	3	4
108	0	0	0	0	0	1	0	0	0	1
110	1	1	0	1	1	1	1	0	3	3
112	1	1	0	1	1	0	1	1	3	3
114	0	1	0	1	1	1	1	1	2	4
116	0	0	0	1	1	1	1	0	1	3
118	1	1	1	0	1	1	0	1	3	3
120	1	1	1	1	0	1	1	1	4	3
122	1	1	1	0	1	1	0	1	3	3
103	0	1	1	1	1	1	1	0	3	3
105	1	0	0	0	1	1	0	0	2	1
107	0	1	0	1	1	1	1	1	4	2
109	0	0	0	1	1	0	0	0	1	1
111	0	0	0	1	0	0	0	0	0	1
113	0	1	0	1	1	1	1	1	4	2
115	0	1	0	1	1	1	1	1	4	2
117	1	1	0	0	1	1	1	1	4	2
119	0	1	0	0	1	1	1	1	4	1
121	0	1	1	1	1	1	1	1	4	3
123	0	0	0	1	1	1	0	1	3	1
Auralized	8	9	6	7	10	9	7	7		
Normal	2	7	2	8	9	10	8	8		
Complexity	7	3	18	6	8	5	6	6	Avg. Complexity = 7.375	

subprogram calls other than by a percussive sound for each, errors that can be investigated experimentally are those that involve, or manifest themselves through, branches in program flow. That is, we are restricted to experiments that involve the class of programs in which the bugs affect program flow. A program whose bug involves an incorrect assignment (such as incorrect interest calculation in a mortgage application) would not be a member of this class unless the result of the assignment caused a perturbation in the program's flow (e.g., incorrect calculation of the mortgage's annual repayment results in a selection process involving the buyer's salary and the annual repayment giving a wrong decision).

Therefore, there are two classes of bug that we can investigate using the current `CAITLIN` system:

- Ill-formed simple Boolean expressions, or combinations of such predicates, directly causing perturbations in the program's flow.
- Incorrect assignments that manifest themselves indirectly through incorrect program flow.

[Meehan et al. 1991; Meehan 1993] classified bugs as either overt (where the program fails at compilation or run-time causing an error message to appear) or covert (where the program runs to completion but the output is not as expected). This experiment, then, was concerned with covert bugs, but also could find nonterminating loops.

3.4 Results

One mark was awarded for each bug that was correctly located. This gives a maximum possible score of 8 overall and 4 for each of the two states (auralized and normal). Table I shows the raw results with a 1 representing a correct response and zero an incorrect answer. Each row of the table holds the results for an individual subject, with the scores for each exercise held in the columns. Scores for exercises that were undertaken in the auralized mode are shown with a shaded background. Note, exercises were completed in order from A1 to A8 and the scores are arranged on the table to show the differences between auralized and normal exercises. The last row of the table shows the McCabe cyclomatic complexity measure for each program. (It is a function of the number of decision nodes in a

program [Pressman 1992].) The two rows above it show the number of subjects who correctly located the bug in the program in the auralized and normal states.

A paired two-tailed t-test performed on the raw data showed no evidence that the auralizations had an effect on performance overall [either positive or negative, $t(21) = 1.40$, $p = 0.175$]. To see whether subjects' musical skills affected their performance, the questionnaire from the motif recognition study (above) was used. As before, the data from the questionnaire were analyzed by multiple linear regression to see whether any or all of the musical factors had an influence on the scores. The regression model indicated that musical knowledge and experience had no significant effect on the debugging scores.

3.5 Discussion

The results indicate that the musical experience and knowledge of subjects did not appear to have an effect on either the correct or incorrect scores; that is, subjects with little musical knowledge did no better and no worse than subjects with more musical skill.

From the raw data, we do not yet find conclusive evidence that the auralizations led to any improvement in the bug location rate of the novice programmers. However, we need to return to the matter of program complexity. We stated earlier that each experimental program was tested in both auralized and nonauralized states in order that differences in individual program complexity would be balanced across the subjects. However, closer inspection of the results shows that program complexity may still have affected subjects' performance. From Table I, we can see that the mean complexity of the exercises that were auralized for the even-numbered subjects (A1, A3, A5, and A7) was 8.5 compared with 6.25 for the odd-numbered subjects' auralized exercises (an increase of 36%). Furthermore, this higher-complexity program group (A1, A3, A5, and A7) had an average bug location rate of 7.5 (68%) in the auralized state (subjects 102–122) compared with 4.75 (43%) for the nonauralized state (subjects 103–123).

Thus, despite the attempts to balance effects, because of complexity, we observe that a bias exists in experiment; that is, the even-numbered subjects had auralized programs of a higher mean complexity than the odd-numbered subjects. In addition, the performance of subjects on these programs was higher in the auralized state than in the nonauralized state. This effect is particularly noticeable for exercises, A1 and A5. By cross-tabulating the correct scores against whether or not the auralization was applied and looking for effects of the auralization on the individual exercises, we see that exercises A1 ($r = 6.6, 1\,df, p = 0.01$) and A5 ($r = 3.8, 1\,df, p = 0.049$) stand out as having significantly higher results in the auralized state compared with the nonauralized state.

The bug in program A1 was one with which, in our experience, novices repeatedly have difficulty [Rimmer et al. 1995]. The program had the following `WHILE` loop

```
WHILE (name <> zeds) AND (mark <> 0) DO
```

in which a logical `AND` was used instead of a logical `OR` (the specification for the program was that the loop would terminate upon finding a record with ten uppercase zeds in the name field and a zero in the mark field). Such an error is often made when learning to program. We attribute this to the tendency to ascribe imprecise natural language meanings to precise logical operators. Because this error is made so often by novices [Rimmer et al. 1995], we considered it hard to spot from the program documentation alone (after all, the Boolean expression sounds right when read aloud). It is pleasing that subjects seemed able to make use of the auralization to locate the bug. It is clearly audible from the auralization that the loop has not iterated sufficient times to process all the input data. Furthermore, the output generated by the program gave no direct clue as to the bug's location.

The very same category of bug was to be found in the second program, A2. This time there was no significant difference in locating the bug between those using and those not using the auralization. This

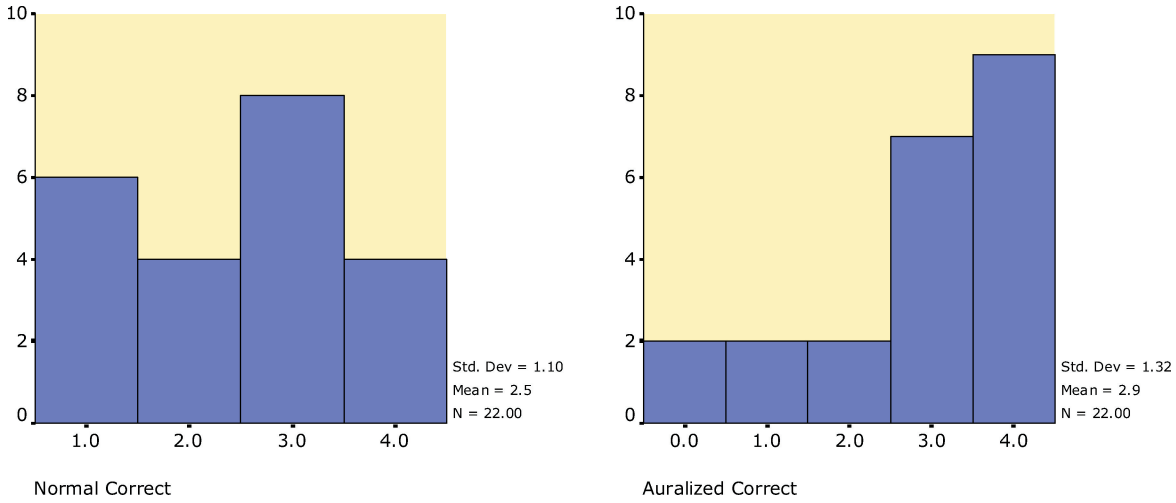

Fig. 2. Score distributions of raw data.

is not unreasonable as, having solved an identical class of problem in the previous exercise, the subjects how had a repertoire of case history against which to compare this problem [Nanja and Cook 1987]. However, there is one major difference between programs A1 and A2. In A2, the actual and expected outputs gave a clue as to the nature and location of the bug. The expected output showed all the input records, while the actual output showed only three of the records. This leads one to suspect a fault in the controlling `WHILE` loop. Program A1 had no such clues in its output. Other studies have shown that novices tend to search output for clues [e.g., Nanja and Cook 1987]. This strategy would not work for program A1 because of the lack of output clues. The auralization provides an alternative form of output as it is possible to hear the loop executing too few times.

The other program that had a significantly higher proportion of correct results in the auralized state was A5. This program had the highest complexity value of all (18 compared to a mean of 7.375) as it had four complex `IF` statements, which made liberal use of DeMorgan's rewrite rules.[3] As with A1, the output from this program gave no indication as to which of the four compound `IF` statements was in error.

Figure 2 shows how the correct scores were distributed in the normal and auralized states, respectively. The first histogram shows a relatively flat distribution for the scores on nonauralized exercises. When the auralizations were added, the score distribution changed to that shown in the second histogram. Although the original t-test did not show this to be significant, there appears to be a general trend toward improved scores when the auralizations are applied. Above, we discussed the effect of program complexity on the experiment and noted that there was an inadvertent bias in the grouping of the auralized and nonauralized exercises for the two subject groups. Therefore, it is reasonable to weight the bug location scores to take program complexity into account by multiplying a score (0 or 1) by its program's complexity and then dividing by the mean complexity of the eight exercises. The weighted scores are given as Table II.

The weighted scores were used to carry out a further t-test to see whether, when adjusted for complexity, the experiment shows an improvement in bug location scores in the auralized condition. With this adjustment, we observe a significant difference between the scores in the auralized and nonauralized states [$t(21) = 2.10$, $p < 0.05$]. What this suggests then, is that auralizations do help where the

[3](1) $\overline{P \wedge Q} = \overline{P} \vee \overline{Q}$, (2) $\overline{P \vee Q} = \overline{P} \wedge \overline{Q}$.

Table II. Weighted Bug Location

Subject	A1	A3	A5	A7	A2	A4	A6	A8	Auralized	Normal
102	0.95	0.41	2.44	0.81	1.08	0.68	0.81	0.81	4.61	3.39
104	0.95	0.41	2.44	0.81	1.08	0.68	0.81	0.81	4.61	3.39
106	0.95	0.41	2.44	0.00	1.08	0.68	0.81	0.81	3.80	3.39
108	0.00	0.00	0.00	0.00	0.00	0.68	0.00	0.00	0.00	0.68
110	0.95	0.41	0.00	0.81	1.08	0.68	0.81	0.00	2.17	2.58
112	0.95	0.41	0.00	0.81	1.08	0.00	0.81	0.81	2.17	2.71
114	0.00	0.41	0.00	0.81	1.08	0.68	0.81	0.81	1.22	3.39
116	0.00	0.00	0.00	0.81	1.08	0.68	0.81	0.00	0.81	2.58
118	0.95	0.41	2.44	0.00	1.08	0.68	0.00	0.81	3.80	2.58
120	0.95	0.41	2.44	0.81	0.00	0.68	0.81	0.81	4.61	2.31
122	0.95	0.41	2.44	0.00	1.08	0.68	0.00	0.81	3.80	2.58
103	0.00	0.41	2.44	0.81	1.08	0.68	0.81	0.00	2.58	3.66
105	0.95	0.00	0.00	0.00	1.08	0.68	0.00	0.00	1.76	0.95
107	0.00	0.41	0.00	0.81	1.08	0.68	0.81	0.81	3.39	1.22
109	0.00	0.00	0.00	0.81	1.08	0.00	0.00	0.00	1.08	0.81
111	0.00	0.00	0.00	0.81	0.00	0.00	0.00	0.00	0.00	0.81
113	0.00	0.41	0.00	0.81	1.08	0.68	0.81	0.81	3.39	1.22
115	0.00	0.41	0.00	0.81	1.08	0.68	0.81	0.81	3.39	1.22
117	0.95	0.41	0.00	0.00	1.08	0.68	0.81	0.81	3.39	1.36
119	0.00	0.41	0.00	0.00	1.08	0.68	0.81	0.81	3.39	0.41
121	0.00	0.41	2.44	0.81	1.08	0.68	0.81	0.81	3.39	3.66
123	0.00	0.00	0.00	0.81	1.08	0.68	0.00	0.81	2.58	0.81
Complexity	7	3	18	6	8	5	6	6	Avg. Complexity = 7.375	

complexity of the program makes more traditional debugging methods harder to apply. This analysis compares favorably with results obtained on Alty et al.'s PROMISE project [Alty et al. 1994] in which the use of sound was not found to be helpful for easy tasks, but became increasingly useful as task complexity increased. Further experimentation is necessary to explore this.

3.5.1 *Time Taken.* We found that the auralizations did not significantly affect the time taken to locate the bugs, although the result was a boundary case [$t(21) = 1.98$, $p = 0.06$]. The subject protocol data gathered during the experiment indicate that subjects were playing the auralizations (on average twice for both the normal-speed and slower-speed versions). If the auralizations were truly having no effect on bug location, then the implication is that despite playing the auralizations subjects were simply ignoring them. Alternatively, it may be that when faced with auralized exercises, subjects spent the time they would have spent examining the documentation for clues listening to the auralization for clues instead. In that case, it could be argued that the auralizations were useful.

If the auralizations were beneficial, how may the timing results be interpreted? Consider exercise A5 which scored considerably better in its auralized state. The problem in this exercise was to decide which of four very similar looking IF statements contained the bug. The output provided no clues as to which of the IFs was faulty and so the only way to solve the problem with the documentation alone was to evaluate each of the compound Boolean expressions in order to find the error. This is a time-consuming task. However, the auralization provides evaluations of the Boolean expressions automatically. Therefore, in theory at least, the auralizations are able to provide information that takes longer to find in the nonauralized state. As the auralizations take time to listen to, then it could simply be that the listening time balances out the extra reasoning time, thereby resulting in no overall change in the time taken to locate the bug.

3.5.2 *Subject Workload.* Subjects were asked to assess their workload after each exercise using the NASA task-load index (TLX) categories. At the end of the experiment, the TLX weighting factors were weighted by subjects to reflect their overall perceived influence. The ratings collected from each exercise were adjusted using the weight values to give an overall workload score per-exercise-per-subject. Subjects evaluated their workload for each auralized and nonauralized exercise and so the

means for the subjects were analyzed for differences using a two-tailed t-test. The t-test showed a significant increase in perceived workload when using the program auralizations [$t(21) = 2.12, p = 0.046$].

Prior to the study, the subjects had no experience in the theory and practice of program auralization. Therefore, it is reasonable to expect the workload to increase when this unfamiliar technique is presented. Further experimentation would be able to determine whether subjects who are practized in the technique still have higher workload scores when using auralizations in debugging tasks.

4. CONCLUSIONS

We conclude that there is a strong case for claiming that musical program auralization is helpful, although further verification is needed. For programs of relatively high complexity, where the output offers few clues about the nature of the bug, significant evidence was found for the auralizations having a beneficial effect. Several issues have been raised that indicate that further experimentation is warranted. In this study, programs with typical novice programming errors and higher cyclical complexity measures benefited from having the technique applied to them. Further studies will be able to confirm this and indicate what kinds of program lend themselves to being debugged with an auralization approach. It was also observed that some people appeared to have difficulty with the debugging process itself (see subjects 108 and 111 in Table I). It is possible that such individuals will not benefit from using auralizations until their programming and debugging skills have reached some particular standard.

Musical knowledge and expertise had no effect on subjects' performance. In addition, there is no evidence to suggest that lack of musical experience led to poorer performance in the auralized condition, and, therefore, we remain confident that the musicality of the CAITLIN system is not an obstacle. Further experimentation along the lines suggested above would help to determine whether this confidence is misplaced. The analysis of perceived workload shows that, in this study, the auralizations had a significant impact on the subjects. This is not, of itself, an undesirable result if the final outcome is that more bugs are located.

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

Although this research did not address the needs of blind and visually impaired programmers, the results suggest that a musical program auralization 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 auralizations 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.

REFERENCES

ALTY, J. L. 1995. Can we use music in computer-human communication? In *People and Computers X: Proceedings of HCI '95*. M. A. R. Kirby, A. J. Dix, and J. E. Finlay, Eds. Cambridge University Press, Cambridge. 409–423.

ALTY, J. L. AND RIGAS, D. I. 1998. Communicating graphical information to blind users using music: The role of context in design. In *CHI98 Conference on Human Factors in Computing Systems*. C. M. Karat, A. Lund, J. Coutaz, and J. Karat, Eds. ACM Press, Los Angeles, CA. 574–581.

ALTY, J. L. AND VICKERS, P. 1997. The CAITLIN auralization system: Hierarchical leitmotif design as a clue to program comprehension. In *Proceedings of the International Conference on Auditory Display (ICAD'97)*. Xerox PARC, Palo Alto, CA 94304, Palo Alto. 89–96.

ALTY, J., BERGAN, J., AND SCHEPENS, A. 1994. The design of the PROMISE multimedia system and its use in a chemical plant. Tech. Rep. 94/P/0112, Loughborough University.

BOARDMAN, D. B., GREENE, G., KHANDELWAL, V., AND MATHUR, A. P. 1995. Listen: A tool to investigate the use of sound for the analysis of program behavior. In *19th International Computer Software and Applications Conference*. IEEE, Dallas, TX.

BOCK, D. S. 1994. ADSL: An auditory domain specification language for program auralization. In *Proceedings of the International Conference on Auditory Display (ICAD '94)*, G. Kramer and S. Smith, Eds. Santa Fe Institute, Santa Fe, NM. 251–256.

BROWN, M. H. AND HERSHBERGER, J. 1992. Color and sound in algorithm animation. *Computer 25,* 12, 52–63.

HART, S. AND STAVELAND, L. 1988. Development of NASA-TLX (task load index): Results of empirical and theoretical research. In *Human Mental Workload*, P. Hancock and N. Meshkati, Eds. North Holland, Amsterdam. 139–183.

JAMESON, D. H. 1994. Sonnet: Audio-enhanced monitoring and debugging. In *Auditory Display*, G. Kramer, Ed. Santa Fe Institute, Studies in the Sciences of Complexity Proceedings. Addison-Wesley, Reading, MA. 253–265.

MEEHAN, D. 1993. Knowledge-Based Advising in the Domain of Scientific Programming. Ph.D. thesis, University of Liverpool.

MEEHAN, D., LEONARD, J., AND SCHONFELDER, J. L. 1991. An intelligent FORTRAN adviser for postgraduates and researchers. In *CALISCE 91 International Conference on Computer Aided Learning and Instruction in Science and Engineering*. Fred, Lausanne, Switzerland. 409–415.

NANJA, M. AND COOK, C. R. 1987. An analysis of the on-line debugging process. In *2nd Workshop on Empirical Studies of Programmers*. 172–184.

NASA HUMAN PERFORMANCE RESEARCH GROUP 1987. Task load index (NASA TLX) v1.0 computerised version.

PRESSMAN, R. S. 1992. *Software Engineering: A Practitioner's Approach*, 3rd ed. McGraw-Hill, London.

RIMMER, A., PARDOE, J., AND VICKERS, P. 1995. Interactive program assessment using SPROUT. In *3rd Annual Conference on the Teaching of Computing*. S. Alexander and P. Magee, Eds. CTI/Dublin City University, Dublin, Ireland. 285–294.

VICKERS, P. 1999. CAITLIN: Implementation of a Musical Program Auralisation System to Study the Effects on Debugging Tasks as Performed by Novice Pascal Programmers. Ph.D. thesis, Loughborough University.

VICKERS, P. AND ALTY, J. L. 1996. CAITLIN: A musical program auralization tool to assist novice programmers with debugging. In *Proceedings of the International Conference on Auditory Display (ICAD '96)*. S. P. Frysinger and G. Kramer, Eds. Xerox PARC, Palo Alto, CA. 17–24.

VICKERS, P. AND ALTY, J. L. 1998. Towards some organising principles for musical program auralization. In *Proceedings of the International Conference on Auditory Display (ICAD '98)*. S. A. Brewster and A. D. N. Edwards, Eds. Electronic Workshops in Computing. British Computer Society, Glasgow.

Received February 2005; revised June 2005; accepted July 2005