When bugs sing

This item was submitted to Loughborough University's Institutional Repository by the/an author.


Additional Information:

- This is a journal article. It was published in the journal, Interacting with computers [© Elsevier]. The definitive version is available from: ttp://www.sciencedirect.com/science//journal/09535438 or at: doi:10.1016/S0953-5438(02)00026-7

Metadata Record: https://dspace.lboro.ac.uk/2134/3357

Publisher: © Elsevier

Please cite the published version.
This item was submitted to Loughborough’s Institutional Repository by the author and is made available under the following Creative Commons Licence conditions.

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/
When Bugs Sing

Paul Vickers¹ & James L Alty²

¹Liverpool John Moores University
School of Computing and Mathematical Sciences
²Loughborough University
Department of Computer Science


Abstract

In The Songs of Insects, Pierce (1949) described the striped ground cricket, nemobius fasciatus-fasciatus, which chirps at a rate proportional to ambient air temperature. Twenty chirps-per-second tell us it is 31.4°C; sixteen chirps and it’s 27°C. This is a natural example of an auditory display, a mechanism for communicating data with sound. By applying auditory display techniques to computer programming we have attempted to give the bugs that live in software programs their own songs. We have developed the CAITLIN musical program auralisation system (Vickers & Alty, 2002b) to allow structured musical mappings to be made of the constructs in Pascal programs. Initial experimental evaluation (Vickers & Alty, 2002a, 2002b) showed that subjects could interpret the musical motifs used to represent the various Pascal language constructs.

In this paper we describe how the CAITLIN system was used to study the effects of musical program auralisation on debugging tasks performed by novice Pascal programmers. The results of the experiment indicate that a formal musical framework can act as a medium for communicating information about program behaviour, and that the information communicated could be used to assist with the task of locating bugs in faulty programs.

Author Keywords: Auditory Display; Music; HCI; Auralisation; Debugging; Pascal
1. Introduction

Program auralisation (the mapping of program data to sound) has attracted a modicum of interest. Brown and Hershberger (1992) were among the first to suggest using sound to aid in the visualisation of software. Jameson’s Sonnet (Jameson, 1994), Bock’s ADSL—Auditory Domain Specification Language (Bock, 1994), and LSL, or, the LISTEN Specification Language (Boardman et al., 1995) all developed the idea. Jameson built a visual programming language to add audio capabilities to a debugger whilst the ADSL and LSL systems added audio to programs at the pre-processing stage. However, to-date very little formal evaluation of program auralisation has been done. We built the CAITLIN system to study the effects of musical program auralisations on debugging tasks.

The CAITLIN system uses a musical framework as music offers much as a communication medium (Alty, 1995, Alty et al., 1997). Earlier experiments showed that music can convey computing and program-related information (Vickers & Alty, 2002a, 2002b). In this paper we describe an experiment to determine if the CAITLIN approach could assist with bug location. Technical details of the CAITLIN system are given elsewhere (for example, see Vickers & Alty, 2002b). In summary, CAITLIN’s auralisations are effected by mapping the constructs of a Pascal program (IF, CASE, WHILE, REPEAT, and FOR) to short musical tunes (motifs). The key aspects of a construct (what we call points-of-interest), such as entry, exit, and Boolean evaluations are assigned motifs consistent with the structure and harmony of the other motifs in the construct. Motifs were constructed using an hierarchic approach (Vickers & Alty, 2002a) to allow the taxonomy of Pascal constructs to be maintained.

2. Objective

The motivation behind this research is to explore what contribution musical program auralisation has on the task of debugging programs (particularly by novice programmers). In theory, auralisations offer an advantage over traditional static program analysis techniques (flow graphs etc.) because an auralisation is immediate and can be generated during program execution. Because sound can carry information in parallel streams, there is much potential bandwidth available for communicating information about program state. We also know that sound (especially music) has a good persistence in the short-term memory (Watkins & Dyson, 1985). The short-term auditory image store has a longer decay rate than the short-term visual image store\(^1\), and so it could be expected that immediate

\(^1\) Typically, the auditory image store has a half-life of between 3.5 and 13 times longer than that of the visual image store. The half-life of an auditory or visual image is defined as the time after which the probability of retrieval is less than 50%. On average this is 200 (70~1000) ms for the
musical representations of program execution would assist novices with debugging.

Bock (1994) found that when using his ADSL system, on average, subjects located 68% of bugs in short C programs. However, no attempt was made to discover how much the audio cues contributed to the task since tests were not performed in the absence of audio cues. Therefore, we need to explore whether or not auralisation leads to any significant difference in bug location rates when compared with tasks in which no auralisations are available.

Novice programmers are often poor at discovering bugs from a program source listing (Bonar & Soloway, 1985, Nanja & Cook, 1987). Because novices are unfamiliar (inexpert) with programming languages and concepts, it is harder for them to identify sources of error in their program logic. For example, to an expert, a controlling condition on a Pascal WHILE loop such as:

\[ \text{WHILE (name} \neq \text{‘ZZZZZZZZZZ’)} \text{ AND (mark} \neq 0) \]

in a block that is to read names and marks from a file until a record containing a name of zeds and a mark of zero is reached, the error is quite obvious (the AND should be an OR). However, to novices who has not formalised their reasoning about the logical operators and still tends to assign inexact natural language meanings to Boolean expressions (Bonar & Soloway, 1985, Moan, 1989) the error is not at all obvious.

So, we wished to find out whether immediate musical representations of program execution would assist novices with bug location tasks. In an experiment on interpretation of CAITLIN’s musical auralisations (Vickers & Alty, 2002a) no significant difference in performance was found across subjects with varying levels of musical knowledge and experience. We would hope that in bug location tasks, the musical experience of subjects would continue to have no effect on performance.

### 3. Imposed constraints

To keep the experimental design straightforward, several constraints were imposed on the research. Up till now no good evidence has been put forward to show whether program auralisation assists in the programming process. Therefore, because the basis of this research is to make progress in answering that fundamental question it is reasonable to restrict the domain of problems to which experimentation is to be applied.

#### 3.1. Points of Interest

We have described elsewhere (Vickers & Alty, 1996, 2001b) how the CAITLIN system auralises Pascal programs by mapping points of interest within the program to musical events. At present these points of interest are drawn only from the set of selection and iteration constructs visual image store and 1500 (900–3500) ms for the auditory store (Card et al., 1983).
available in Turbo Pascal. This means that the domain of possible program events is larger than the
corresponding musical event domain (see Figure 1). Although this restricts the information that can
be communicated about a running program, it makes experimental design simpler whilst still
allowing a range of debugging problems to be tackled.

![Figure 1 Mapping between Program & Auralisation Domains](image)

As only the Pascal constructs have been auralised, the CAITLIN system does not provide a complete mapping between
program features and musical motifs.

### 3.2. Bug types

The current mappings between program and musical event domains necessarily restrict the
number of bug types that can be investigated with the present CAITLIN system. As the system does
not address representations of assignments or sub-program 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. For instance, the errant WHILE loop condition described above
is of this type. A bug that 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. the 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
auralisations:

- 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. (1993, 1991) classified bugs as belonging to one of two classes:

- **overt/direct** — where the program fails at compilation or run-time causing an error message
to appear.
- **covert/indirect** — the program runs to completion but the output is not as expected.
This experiment, then, is primarily concerned with covert bugs, but could also address non-terminating loops.

4. Experimental hypotheses

From the above it can be seen that using the CAITLIN system will allow the following general hypothesis to be investigated: 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.

This hypothesis can be tested by comparing the number of bugs identified in programs under two experimental conditions: with and without the aid of the auralisations. We will not investigate whether subjects can correctly specify the bugs for two reasons. First, that is not a function of the direct information carried by the auralisations. Secondly, we are not trying to assess subjects’ skill in debugging. All we are interested in for the purposes of this research is whether subjects can locate the position of a bug within a given program text.

This question is wide in scope and embraces several other related questions that will help us to find out how auralisation can affect novice programmers. From the above reasoning, four questions are identified that may be explored by this research.

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?

The first question may be answered by having subjects locate bugs in a series of programs using a mixture of auralised and non-auralised assistance. Thus our first test may be expressed in terms of the null hypothesis:

- Program auralisations have no effect on the level of bug location. \((H_{10})\)

By capturing the time in which subjects complete the exercises we have the means to answer question 2. Given that extra information is presented by the auralisations and that the auralisations take time to listen to, it might reasonably be expected that the time taken to locate bugs will be longer when using the auralisations. The null hypothesis for question 2 would then be:

- Program auralisations have no effect on the time taken to locate bugs. \((H_{20})\)

The third question requires surveying the subjects to discover their musical backgrounds. Results from this survey can then be compared with the scores attained in the debugging exercises to look for
any interactions. An aim of this research was to develop a system that could be used by musical and non-musical people alike without the need for any special musical training. Therefore, we expect the musical experience and knowledge of subjects to have no significant effect on the results, which leads to the third null hypothesis:

- Musical knowledge and experience do not significantly affect the ability of subjects to use the system to locate bugs. \( (H3_0) \)

The assessment of workload is interesting. Debugging programs using traditional techniques will involve a level of mental, physical, and cognitive activity that may be considered together as workload. By supplementing the information available to subjects by means of the auralisations, it is helpful to know whether this impacts on the level of workload experienced. Because the auralisations carry much of the same information as a program listing, input data and output data, just in a different medium, then the bandwidth increases; this could mean that more effort is expended in dealing with the increased bandwidth. Conversely, it might be found that workload decreases as the auralisations present certain information more clearly.

Using the NASA task load index (or TLX, Hart & Staveland, 1988, NASA Human Performance Research Group, 1987) it is possible to assess the level of workload experienced by subjects when carrying out tasks. By comparing the workload scores of subjects for auralised and non-auralised tasks, a judgement can be made as to whether the auralisations affected the workload experienced. Because we suspect there might be an effect on workload, but we do not know in which direction it will lie, we formulate the fourth null hypothesis thus:

- The auralisations have no significant effect on the perceived workload of subjects. \( (H4_0) \)

If the subjects perform as hoped, then we would be looking to accept or reject the four null hypotheses as follows:

<table>
<thead>
<tr>
<th>Accept/Reject</th>
<th>1 or 2 tail testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1&lt;sub&gt;0&lt;/sub&gt;</td>
<td>Reject – an effect is required</td>
</tr>
<tr>
<td>H2&lt;sub&gt;0&lt;/sub&gt;</td>
<td>Reject—a difference in times is anticipated</td>
</tr>
<tr>
<td>H3&lt;sub&gt;0&lt;/sub&gt;</td>
<td>Accept—a difference would show musical expertise does help</td>
</tr>
<tr>
<td>H4&lt;sub&gt;0&lt;/sub&gt;</td>
<td>Reject—we reason that some effect will be evident</td>
</tr>
</tbody>
</table>

Table 1 Evaluation of hypotheses

5. Bug location study

An experiment was designed that would allow exploration of the five questions raised above.
5.1. Subjects

Twenty-two subjects took part in the experiment. All were undergraduate second-year and final-year computing students at Loughborough University. The experiment was carried out as part of a course in human-computer interaction. Twenty of the subjects were male. Nineteen subjects were between 20 and 29 years old, the remaining three were 19 years old or younger. All but one subject reported a western-style cultural upbringing. Seventeen subjects had two or more years of programming tuition. The other five had at least one year’s tuition. On average, the subjects had written programs in four programming languages (min. 2, max 7).

5.2. Musical background

A problem with trying to objectively measure musical ability is that there is currently no accepted standard for doing so. Edwards et al (2000) report an attempt to devise such a test. However, the test is still in its early stages and has not yet been validated. For the purposes of this study we used a questionnaire to measure musical knowledge and experience. The questionnaire used a number of questions to gather values for four variables: interest, play, sing and musical score. The interest variable attempts to measure a subject’s interest in music. The responses were scored using the following scale:

0. No interest in music at all
1. Enjoy listening to music
2. Enjoy performing music (alone, with friends or professionally)
3. Enjoy listening and performing

The play (question 8) variable is a simple Boolean flag stating whether or not a subject plays a musical instrument.

Sing (question 9) is a measure of how much subjects participate in singing. The possible values are:

0. I do not sing
1. I sing in the bath
2. I sing informally to others
3. I sing in a choir
4. I sing semi-professionally
5. I sing professionally

Musical score represents the score attained by subjects on a musical knowledge test. One mark was given for each correct answer resulting in a range 0 to 15.
5.3. **Descriptive statistics**

All subjects reported having some interest in music, with the majority (16) stating that they enjoy listening to music. The remaining six subjects enjoyed performing music. The *play* variable showed that six subjects played an instrument, a result consistent with the responses to the *interest* question. Most of the subjects (16) claimed not to be singers, the remainder singing at informal or amateur levels.

![Chart 1 Musical knowledge scores](image)

**Chart 1 Musical knowledge scores**
The ordinate on the bar chart shows how many subjects achieved the scores in the musical knowledge test shown on the abscissa. The maximum possible score was 15. We see that 11 subjects scored zero, and none scored more than 9.

From Chart 1 we observe that all subjects scored 9 or less on the music knowledge test (maximum possible score=15). The scores of those who played instruments were no higher than the group as a whole (mean 3.17, min. 0, max. 6).

5.4. **Task**

Eight Pascal programs were presented to the subjects (see Appendix A for some samples). The program source code was formatted using a standard layout. Each program contained a single bug in one of its constructs. For each program subjects were asked to identify the line of code that contained the bug. For half the exercises auralisations generated by the CAITLIN system were also given. Following each exercise subjects were asked to assess their workload using the NASA TLX ratings. The auralisations were derived from the motifs described in Vickers & Alty (2002a). The programs were designed so as not to result in auralisations of excessive length, the longest auralisation lasting for approximately two minutes.

5.5. **Procedure**

The experiment comprised two sessions:

- Introduction and tutorial
• Debugging test

There was a short break between the two sessions to allow the subjects to move to a computer laboratory in which the debugging tests were run. At the start of the experiment each subject was given a workbook that contained:

• The full written text of the introduction and tutorial session.

• Instructions on how to complete the exercises (including how to use the experiment software).

• Program descriptions\(^{2}\) for the eight exercises, comprising:
  
  ❖ A textual description of the program’s requirements in terms of what it must do and what format the input and output data should take
  
  ❖ A program design in pseudo-code
  
  ❖ The input and output data that were used
  
  ❖ The expected output, that is, what a bug-free version of the program should produce.

This set of documentation provided the sort of information that novice programmers use when debugging. Nanja and Cook (1987) showed that novices tend to look immediately for candidate bug locations by searching the output for clues, recalling similar bugs, and testing program states.

5.6. Introduction and tutorial

The subjects had no prior experience of program auralisation and so a briefing session was given. The tutorial explained the technique and gave examples of the various auralisations used. An explanation of how the auralisations are built and how they represent the various components of the constructs was given.

5.7. Debugging tests

The debugging tests were administered using a specially written web application that allowed collection of timing information, questionnaire responses, NASA TLX scores, and subject protocol data (see Vickers, 2001 for a technical description of the application). All subject data was stored in data files on the host server.

The auralisations were prepared prior to the experiment and stored on the web application as MPEG-1 Layer 3 (MP3) audio files. The MPEG format was chosen as it meant that the audio files were not so large that long downloading delays would occur. Because the dynamic range of the

\(^{2}\) An example is given in Appendix B
music was not large, and because subjects were using low-end stereo headsets to listen to the auralisations additional compression was possible by using a lower bit-rate than would be used for CD quality sound. The application used an embedded Windows Media Player control that gave subjects facilities for playing, stopping, pausing, rewinding, and spooling the auralisations.

CAITLIN allows auralisations to be played at different tempi. This feature was not available in the experiment as pre-prepared auralisations were used. Therefore, for each exercise two auralisations were provided: one at 120 beats-per-minute (slow) and one at 140 beats-per-minute (normal). Subjects could access either auralisation by pressing an appropriately labelled button.

The debugging test comprised five stages: logging in, on-screen instructions, completion of questionnaire, the eight debugging exercises and collection of NASA TLX ratings. Each of these stages was administered by the web application. A description of each stage follows:

**Logging on.** Each subject was assigned an identification code that was needed to log on to the experiment. This identification code was used later on by the application to determine which exercises should be auralised. If the identification number was even then the subject would take exercises 1, 3, 5 and 7 in the auralised mode and 2, 4, 6 and 8 in the non-auralised mode. This was swapped for subjects with odd identification codes. An equal number of odd and even codes was assigned to the subjects.

**On-screen instructions.** Working examples of the various form fields, buttons, Windows Media Player controls, and dynamic HTML (DHTML) widgets used in the application were presented at this point. Subjects could spend as much time as necessary in this section.

**Completion of questionnaire.** The same questionnaire that was given to the subjects in the earlier listening test study (Vickers & Alty, 2002a) was given to subjects on this experiment. The questionnaire was completed on-line.

**The eight debugging exercises.** Each exercise was conducted in three parts: reading of program documentation, location of the bug, and collection of TLX scores. The bug location part was timed and subject to a ten-minute maximum. If the exercise was not completed within ten minutes limit then the application automatically moved on to the next one. This ensured that the experiment did not take too long. The application presented the program documentation (description, pseudo-code, input, output, and expected output) in scrollable DHTML frames. The on-screen documentation was an exact copy of that given in the workbook allowing subjects the choice of reading from paper or from the monitor screen. There was no time constraint imposed on reading the documentation. When ready to try and locate the bug in the program, subjects pressed the button labelled “Start Debugging”. Pressing the ‘start debugging’ button caused a frame containing the actual program source code and a timer to appear on the screen. The timer showed how much time remained for the
exercise. If the exercise required auralisations to be presented then a set of controls to play the two auralisations also appeared.

Subjects indicated where they thought the bug lay by clicking on the relevant line of code in the source code frame. This action would highlight the line with a red background. Moving the mouse pointer over a selectable line (anything after the first BEGIN) caused the line to turn blue as long as the mouse pointer remained over it. Subjects could change their minds by selecting a different line. To submit their answer, subjects pressed the button labelled ‘Next exercise’.

If the exercise had associated auralisations then the Windows Media Player control would appear. Pressing any of the transport buttons (e.g. play, stop etc.) would affect the currently-loaded auralisation. By default, an auralised exercise begins with the normal speed auralisation ready to play. To change between the normal and slower speed versions of the auralisations subjects pressed one of the two buttons below the Media Player control.

Collection of NASA TLX ratings. Once the timer reached zero, or the ‘Next exercise’ button was pushed, the program information would be cleared from the screen and a form would be presented from which subjects could choose appropriate values for the various task load index factors.

5.8. Experimental design

This study was designed to allow a comparison to be made between tasks performed in both auralised and non-auralised states, thus addressing a major limitation in Bock’s (1995) experiment. We used a repeated-measures within-subjects design. The allocation of exercises and auralisations by subject is shown in Table 2.

<table>
<thead>
<tr>
<th>Auralised?</th>
<th>Subject I.D. Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Even</td>
</tr>
<tr>
<td>Yes</td>
<td>1, 3, 5, 7</td>
</tr>
<tr>
<td>No</td>
<td>2, 4, 6, 8</td>
</tr>
</tbody>
</table>

Table 2 Experimental design
Subjects with even identification numbers had auralisations applied to exercises A1, A3, A5, & A7, whilst the odd-numbered subjects had the auralisations applied to A2, A4, A6, & A8.

Subjects performed each of the eight tasks (named A1 to A8) in order beginning with task A1. Half of the tasks were tackled with accompanying auralisations and half without. The treatment (auralisation) was given to alternate tasks. Subjects with even-numbered identification codes had the treatment applied to tasks A1, A3, A5 and A7 whilst the odd-numbered subjects had the treatment applied to tasks A2, A4, A6 and A8. In this way every program was tested in both the treatment and non-treatment states meaning that any differences between programs should not influence the results. Also, as each subject performed tasks in both states, any differences between the odd- and even-numbered groups were counterbalanced.
However, one final factor must be taken into account. The eight programs used in the experiment differed in complexity (as measured by McCabe’s cyclometric complexity metric (McCabe, 1976)). Whilst the experimental design ensures that all programs are used under both experimental conditions, examination of the mean cyclomatic complexity of the two groups of programs reveals a residual experimental bias. The mean complexity of the eight programs was 7.375. Six programs had complexity values of between 5 and 8, whilst programs A3 and A5 had values of 3 and 18 respectively (A5 involved multiple compound Boolean expressions). This affects the mean complexity of the two groups. The mean complexity of the exercises that were auralised for the even-numbered subjects (A1, A3, A5, & A7) was 8.5 compared with 6.25 for the odd-numbered subjects’ auralised exercises (an increase of 36%). If no bias existed, then one mark would be awarded for each correctly located bug giving a maximum possible score of 8 for the experiment as whole. To account for the different complexities, each program score should be weighted by dividing its complexity measure by the mean complexity of the eight exercises. This would give, for example, a score of 0.95 available for program A1 (7 ÷7.375), 0.41 for program A3 (3 ÷7.375), and 2.44 for program A5 (18 ÷7.375). The maximum available mark remains 8.

5.9. Results

The results from the experiment were collated and tests applied to examine the five hypotheses.

5.9.1. Hypothesis 1: Bug location

The first test was to see whether the auralisations had any effect on the location of bugs. A summary of results is given in Error! Reference source not found. A paired t-test (t(21)=2.10, p<0.05) shows a significant difference between the results of the auralised and non-auralised sets. From this we conclude that the auralisations did affect bug location performance.

5.9.2. Hypothesis 2: debugging time

The times taken (in seconds) by subjects to locate their candidate bug (that is, press the ‘Next exercise’ button) were logged. A time limit was placed on each exercise resulting in some subjects failing to submit an answer in time. Such cases were given a value of 99999 seconds, to represent infinity. These infinities make calculation of mean values impossible, and so median values for each subject were used instead. The median response times (Table 4) were analysed using a paired t-test which showed that the response times in the auralised state were not significantly different from those of the non-auralised state (t(21)=1.98, p=0.06). Consequently we cannot reject the null hypothesis and must conclude that there is insufficient evidence to show that the auralisations significantly altered the time taken to complete the exercises.
5.9.3. **Hypothesis 3: musical experience**

As in the earlier listening test (Vickers & Alty, 2002a), it was hoped that the musical knowledge and experience of subjects would not affect their ability to make use of the musical auralisations. The musical variables on the questionnaire could not be considered independent of each other. For example, in the previous study a significant difference was found between the scores obtained on the musical knowledge test by those who played a musical instrument and those who did not.

Table 3 Weighted debugging test scores. The scores are arranged so as to show the auralised scores and normal scores in groups. Auralised scores are shown shaded. Note, exercises were completed in order from A1 to A8 and the scores are merely arranged on the table to show the difference between auralised and normal exercises. Th last row of the table shows the McCabe complexity measure for each program. The two rows above it show the weighted total score of subjects who correctly located the bug in the program in the auralised and normal states. The two rows above them show the total number of subjects who correctly located the bug in both states.

<table>
<thead>
<tr>
<th>Subject</th>
<th>A1</th>
<th>A3</th>
<th>A5</th>
<th>A7</th>
<th>A2</th>
<th>A4</th>
<th>A6</th>
<th>A8</th>
<th>Auralised</th>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>0.95</td>
<td>0.41</td>
<td>2.44</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.81</td>
<td>4.61</td>
<td>3.39</td>
</tr>
<tr>
<td>104</td>
<td>0.95</td>
<td>0.41</td>
<td>2.44</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.81</td>
<td>4.61</td>
<td>3.39</td>
</tr>
<tr>
<td>106</td>
<td>0.95</td>
<td>0.41</td>
<td>2.44</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.81</td>
<td>3.80</td>
<td>3.39</td>
</tr>
<tr>
<td>108</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.68</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.68</td>
</tr>
<tr>
<td>110</td>
<td>0.95</td>
<td>0.41</td>
<td>0.00</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.00</td>
<td>2.17</td>
<td>2.58</td>
</tr>
<tr>
<td>112</td>
<td>0.95</td>
<td>0.41</td>
<td>0.00</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.00</td>
<td>2.17</td>
<td>2.71</td>
</tr>
<tr>
<td>114</td>
<td>0.00</td>
<td>0.41</td>
<td>0.00</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.00</td>
<td>1.22</td>
<td>3.39</td>
</tr>
<tr>
<td>116</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.00</td>
<td>0.81</td>
<td>2.58</td>
</tr>
<tr>
<td>118</td>
<td>0.95</td>
<td>0.41</td>
<td>2.44</td>
<td>0.00</td>
<td>1.08</td>
<td>0.68</td>
<td>0.00</td>
<td>0.00</td>
<td>3.80</td>
<td>2.58</td>
</tr>
<tr>
<td>120</td>
<td>0.95</td>
<td>0.41</td>
<td>2.44</td>
<td>0.81</td>
<td>0.00</td>
<td>0.68</td>
<td>0.81</td>
<td>0.81</td>
<td>4.61</td>
<td>2.31</td>
</tr>
<tr>
<td>122</td>
<td>0.95</td>
<td>0.41</td>
<td>2.44</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.00</td>
<td>0.00</td>
<td>3.80</td>
<td>2.58</td>
</tr>
<tr>
<td>103</td>
<td>0.00</td>
<td>0.41</td>
<td>2.44</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.00</td>
<td>2.58</td>
<td>3.66</td>
</tr>
<tr>
<td>105</td>
<td>0.95</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.08</td>
<td>0.68</td>
<td>0.00</td>
<td>0.00</td>
<td>1.76</td>
<td>0.95</td>
</tr>
<tr>
<td>107</td>
<td>0.00</td>
<td>0.41</td>
<td>0.00</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.00</td>
<td>3.39</td>
<td>1.22</td>
</tr>
<tr>
<td>109</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.08</td>
<td>0.68</td>
<td>0.00</td>
<td>0.00</td>
<td>1.08</td>
<td>0.81</td>
</tr>
<tr>
<td>111</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.68</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.81</td>
</tr>
<tr>
<td>113</td>
<td>0.00</td>
<td>0.41</td>
<td>0.00</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.00</td>
<td>3.39</td>
<td>1.22</td>
</tr>
<tr>
<td>115</td>
<td>0.00</td>
<td>0.41</td>
<td>0.00</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.00</td>
<td>3.39</td>
<td>1.36</td>
</tr>
<tr>
<td>117</td>
<td>0.95</td>
<td>0.41</td>
<td>0.00</td>
<td>0.00</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.81</td>
<td>3.39</td>
<td>3.49</td>
</tr>
<tr>
<td>119</td>
<td>0.00</td>
<td>0.41</td>
<td>0.00</td>
<td>0.00</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.00</td>
<td>3.39</td>
<td>1.22</td>
</tr>
<tr>
<td>121</td>
<td>0.00</td>
<td>0.41</td>
<td>2.44</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.81</td>
<td>0.81</td>
<td>3.39</td>
<td>3.66</td>
</tr>
<tr>
<td>123</td>
<td>0.00</td>
<td>0.41</td>
<td>2.44</td>
<td>0.81</td>
<td>1.08</td>
<td>0.68</td>
<td>0.00</td>
<td>0.00</td>
<td>2.58</td>
<td>0.81</td>
</tr>
</tbody>
</table>

Therefore, the four musical factors (interest, musical score, sings, plays instrument) were analysed together via a multiple linear regression model to see if they had any significant effect on the number of bugs correctly located in the auralised state. The resultant regression model

\[
y = 4.537 + (0.160 \text{ % musical score}) - (1.711 \text{ % play}) - (0.601 \text{ % interest}) + (0.61 \text{ % sing})
\]

shows that there is no significant relationship between the musical factors and subject scores on the auralised exercises (0.16 <= p <= 1.711). Therefore, we accept the null hypothesis.
press the button within the 10 minute interval. The two rightmost columns show the median bug location times of the auralised and normal states for each subject. Medians were used as the infinities make calculation of means impossible.

<table>
<thead>
<tr>
<th>Subject</th>
<th>A1</th>
<th>A3</th>
<th>A5</th>
<th>A7</th>
<th>A2</th>
<th>A4</th>
<th>A6</th>
<th>A8</th>
<th>Auralised</th>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>344</td>
<td>164</td>
<td>293</td>
<td>257</td>
<td>188</td>
<td>43</td>
<td>90</td>
<td>30</td>
<td>275.0</td>
<td>66.5</td>
</tr>
<tr>
<td>04</td>
<td>338</td>
<td>321</td>
<td>190</td>
<td>199</td>
<td>92</td>
<td>36</td>
<td>135</td>
<td>260</td>
<td>260.0</td>
<td>113.5</td>
</tr>
<tr>
<td>06</td>
<td>226</td>
<td>217</td>
<td>366</td>
<td>530</td>
<td>175</td>
<td>91</td>
<td>112</td>
<td>114</td>
<td>296.0</td>
<td>113.5</td>
</tr>
<tr>
<td>08</td>
<td>249</td>
<td>270</td>
<td>167</td>
<td>200</td>
<td>137</td>
<td>159</td>
<td>253</td>
<td>65</td>
<td>224.5</td>
<td>148.0</td>
</tr>
<tr>
<td>10</td>
<td>536</td>
<td>422</td>
<td>356</td>
<td>469</td>
<td>185</td>
<td>39</td>
<td>132</td>
<td>306</td>
<td>445.5</td>
<td>158.5</td>
</tr>
<tr>
<td>12</td>
<td>234</td>
<td>347</td>
<td>9999</td>
<td>230</td>
<td>171</td>
<td>235</td>
<td>25</td>
<td>191</td>
<td>290.5</td>
<td>181.0</td>
</tr>
<tr>
<td>14</td>
<td>195</td>
<td>557</td>
<td>157</td>
<td>198</td>
<td>194</td>
<td>64</td>
<td>398</td>
<td>89</td>
<td>195.5</td>
<td>141.5</td>
</tr>
<tr>
<td>16</td>
<td>457</td>
<td>273</td>
<td>212</td>
<td>414</td>
<td>342</td>
<td>78</td>
<td>287</td>
<td>72</td>
<td>343.5</td>
<td>182.5</td>
</tr>
<tr>
<td>18</td>
<td>395</td>
<td>501</td>
<td>228</td>
<td>411</td>
<td>56</td>
<td>73</td>
<td>499</td>
<td>141</td>
<td>403.0</td>
<td>107.0</td>
</tr>
<tr>
<td>20</td>
<td>130</td>
<td>341</td>
<td>193</td>
<td>99</td>
<td>341</td>
<td>74</td>
<td>94</td>
<td>264</td>
<td>161.5</td>
<td>179.0</td>
</tr>
<tr>
<td>22</td>
<td>322</td>
<td>218</td>
<td>174</td>
<td>297</td>
<td>109</td>
<td>38</td>
<td>300</td>
<td>300</td>
<td>257.5</td>
<td>204.5</td>
</tr>
<tr>
<td>03</td>
<td>210</td>
<td>436</td>
<td>446</td>
<td>75</td>
<td>331</td>
<td>89</td>
<td>150</td>
<td>580</td>
<td>240.5</td>
<td>323.0</td>
</tr>
<tr>
<td>05</td>
<td>278</td>
<td>9999</td>
<td>115</td>
<td>310</td>
<td>437</td>
<td>28</td>
<td>9999</td>
<td>161</td>
<td>299.0</td>
<td>294.0</td>
</tr>
<tr>
<td>07</td>
<td>271</td>
<td>366</td>
<td>300</td>
<td>152</td>
<td>271</td>
<td>88</td>
<td>112</td>
<td>94</td>
<td>103.0</td>
<td>285.5</td>
</tr>
<tr>
<td>09</td>
<td>136</td>
<td>190</td>
<td>105</td>
<td>230</td>
<td>233</td>
<td>171</td>
<td>81</td>
<td>187</td>
<td>179.0</td>
<td>163.0</td>
</tr>
<tr>
<td>11</td>
<td>237</td>
<td>188</td>
<td>168</td>
<td>351</td>
<td>224</td>
<td>273</td>
<td>171</td>
<td>93</td>
<td>197.5</td>
<td>212.5</td>
</tr>
<tr>
<td>13</td>
<td>188</td>
<td>434</td>
<td>246</td>
<td>181</td>
<td>370</td>
<td>138</td>
<td>359</td>
<td>79</td>
<td>248.5</td>
<td>217.0</td>
</tr>
<tr>
<td>15</td>
<td>301</td>
<td>211</td>
<td>492</td>
<td>89</td>
<td>443</td>
<td>58</td>
<td>114</td>
<td>154</td>
<td>134.0</td>
<td>256.0</td>
</tr>
<tr>
<td>17</td>
<td>196</td>
<td>255</td>
<td>228</td>
<td>409</td>
<td>282</td>
<td>145</td>
<td>296</td>
<td>359</td>
<td>289.0</td>
<td>241.5</td>
</tr>
<tr>
<td>19</td>
<td>188</td>
<td>125</td>
<td>193</td>
<td>178</td>
<td>335</td>
<td>101</td>
<td>250</td>
<td>144</td>
<td>197.0</td>
<td>183.0</td>
</tr>
<tr>
<td>21</td>
<td>441</td>
<td>207</td>
<td>153</td>
<td>230</td>
<td>128</td>
<td>51</td>
<td>85</td>
<td>96</td>
<td>90.5</td>
<td>218.5</td>
</tr>
<tr>
<td>23</td>
<td>348</td>
<td>9999</td>
<td>360</td>
<td>215</td>
<td>219</td>
<td>85</td>
<td>9999</td>
<td>539</td>
<td>379.0</td>
<td>354.0</td>
</tr>
</tbody>
</table>

5.9.4. Hypothesis 4: 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 auralised and non-auralised exercise and so the means for the subjects were analysed for differences with a t-test. The t-test indicates a significant increase in perceived workload when using the program auralisations ($t(21)=2.12, p<0.05$).

6. Discussion

The results of this experiment tell us several things. First, the data indicate that the CAITLIN auralisations did lead to an improvement in the bug location rate of the novice programmers under investigation, and using the auralisations approach did not give rise to a significant increase in the time taken to locate the bugs. Furthermore, it was found that the workload of subjects increased when auralisation information was added to the tasks. The musical experience and knowledge of the subjects did not appear to have any 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 and experience.

6.1. Bug location

Overall, the results suggest that the auralisations did assist with bug location. A closer inspection
of the individual programs is revealing. By cross-tabulating the number of correct exercises against whether or not the auralisation was applied we can look for effects of the auralisation on individual exercises. From this we observe that programs A1 ($r=6.6$, 1df, $p=0.01$) and A5 ($r=3.8$, 1df, $p=0.049$) have significantly higher results in the auralised state than in the non-auralised 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

\[
\text{WHILE (name} \neq \text{zeds) AND (mark} \neq \text{0) DO}
\]

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 and a zero in the mark). 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, 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 auralisation to locate the bug. It is clearly audible from the auralisation that the loop did not iterate 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 auralisation. 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 whilst the actual output showed only three of the records. This may lead 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 & Cook, 1987). This strategy would not work for program A1 because of the lack of output clues. The auralisation 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 auralised 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. As with A1, the output from this program gave no indication as to which of the four compound IF statements was in error.

What this suggests then, is that auralisations do help, especially where the complexity of the program makes more traditional debugging methods harder to apply. This compares favourably with results obtained on the PROMISE project (Alty et al., 1994) in which the use of sound was found to
become increasingly useful as task complexity increased. Further experimentation is necessary to explore this.

6.2. Time Taken

We found that the auralisations did not significantly affect the time taken to locate the bugs ($t(21)=1.98, p=0.06$). Given that subjects were playing the auralisations (on average twice for both the normal-speed and slower-speed versions, according to protocol data gathered during the experiment) why did the time taken to locate the bugs not increase? It may be that when faced with auralised exercises subjects spent the time they would have spent examining the documentation or output for clues listening to the auralisation for clues instead.

Consider exercise A5 which scored considerably better in its auralised 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 auralisation provides evaluations of the Boolean expressions automatically. This indicates that auralisations are able to provide some information quicker than can be found in the non-auralised state. As the auralisations 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.

6.3. Subject Workload

A significant increase in perceived workload when using the auralisations was observed amongst the subjects. Prior to the study the subjects had no experience of the theory and practice of program auralisation. Therefore, it is reasonable to expect the workload to increase when this unfamiliar technique is presented. Further experimentation could determine whether subjects who are practised in the technique still have higher workload scores when using auralisations in debugging tasks.

6.3.1. Musical experience

The musical experience and knowledge of the subjects was shown to have no discernible effect on the bug location results. This is encouraging as it demonstrates that the musical framework of our auralisation approach is widely usable. This finding also confirms and supports the results of earlier studies (see Vickers & Alty, 2002a, 2002b).

7. Conclusions and further work

We conclude that there is a case for claiming that musical program auralisation is helpful. For programs of relatively high complexity, where the output offers few clues about the nature of the bug, the auralisations appeared to have an especially beneficial effect. Further studies will be able to
confirm this and indicate what kinds of program lend themselves to being debugged with an auralisation approach.

Musical knowledge and expertise had no effect on subjects’ performance. Additionally, there is no evidence to suggest that lack of musical experience led to poorer performance in the auralised condition, and, therefore, we remain confident that the musicality of the CAITLIN system is not an obstacle. The analysis of perceived workload shows that, in this study, the auralisations 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. Secondly, 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 auralised and non-auralised debugging with a view to creating a full auralised programming environment.

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 (see Vickers & Alty, 202a), it is not unreasonable to hope that the system can be adapted and extended for use by the blind.

By restricting the system’s use to novice programmers, we are also limiting the size of programs to which the CAITLIN system may be applied. There arises then the question of scalability, that is, whether or not the technique can be usefully applied to large-scale programs. Unfortunately there is not yet any evidence to suggest that auralisation can or cannot be scaled. Francioni et al (1992) state that the “scalability of auralisation, specifically for representing program behaviour, is essentially an un-known and remains to be empirically determined”.

It has been identified that novice and expert programmers approach the task of program analysis and debugging in different ways (Gellenbeck & Cook, 1991, Nanja & Cook, 1987, Riecken et al., 1991). Hence, it does not necessarily follow that an auralisation-based debugging system that can be used by novices will be considered useful by experts.

This study has opened up further avenues of exploration for the development of future program auralisation systems. 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. CAITLIN serves as a useful starting point for such a system and could 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 can we 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 may be useful 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 (1992) believe “that the capability to play back programs
at different speeds...is key to deriving meaning from auralisations”. The facility for the tempo to be
controlled in real-time by the user should be added to a future system.

A technique that Bock (1995) used 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 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 what
language elements will be auralised. Jameson’s Sonnet system (Jameson, 1994) allows a threshold to
be set for loops so that it is not necessary always to hear every repetition. In both systems the
programmer can also specify sections of the program code that will or will not be auralised. In their
work on deriving music from chaotic systems Mayer-Kress et al. (1994) summed up this issue with
their conclusion 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”.

In summary, our results indicate that program auralisation is worth further study and
development. Alty (1995) said experiments are needed to “…determine what is possible and
practicable” regarding the use of music as a communication medium and this paper makes a
contribution to that agenda.

8. References

and J.E. Finlay (Editors), People and Computers X: Proceedings of HCI ’95. Cambridge

J.L. Alty, J. Bergan, & A. Schepens, (1994). The design of the PROMISE multimedia system and its
use in a chemical plant. Loughborough, Loughborough University.


D.S. Bock, Auditory software fault diagnosis using a sound domain specification language, Ph.D. Thesis, Syracuse University, Syracuse.


Appendix A — Sample program source code

Exercise A2

PROGRAM refunds;

VAR
  name: STRING [10];
  value, total, counter: Integer;
  datafile: Text;

BEGIN
  Assign (datafile, 'refunds.dat');
  Reset (datafile);
  total := 0;
  counter := 0;
  Readln (datafile, name, value);
  WHILE (name <> 'ZYGOTE') AND (value <> 0) DO
    BEGIN
      IF value > 100 THEN
        Writeln (name, ' Excellent (' , value , ')')
      ELSE IF value > 50 THEN
        Writeln (name, ' Average (' , value , ')')
      ELSE IF value > 0 THEN
        Writeln (name, ' Poor (' , value , ')')
      ELSE
        Writeln (name, ' Refund (' , value , ')');
      IF NOT ((name = 'SMITH') AND (value = -40)) THEN
        total := total + value;
        Inc (counter);
        Readln (datafile, name, value);
    END;
  Writeln ('The number of transactions made was ' , counter);
  Writeln ('The total value of transactions was ' , total);
  Close (datafile);
END.
Exercise A5

PROGRAM DeMorgan;
USES Crt;
VAR a, b: Integer;
    input_file: Text;
BEGIN
    ClrScr;
    Assign (input_file, 'DeMorgan.dat');
    Reset (input_file);
    REPEAT
        Readln (input_file, a, b);
        IF NOT ((a < 0 OR (a > 20)) AND NOT ((b < 0 OR (b > 20))) THEN
            Write (' Ok ');
        IF NOT (NOT (NOT ((a < 0 OR (a > 20))) OR NOT (NOT ((b < 0 OR (b > 20))))) THEN
            Write (' Ok ');
        IF NOT (NOT (a <= 0) OR (a >= 21)) AND NOT ((b <0) OR (b >=21)) THEN
            Write (' Ok ');
        IF NOT ((a <= -1) OR (a > 20)) AND NOT ((b < 0) OR (b > 20)) THEN
            Write (' Ok ');
        Writeln ;
        Writeln ('-----------------');
    UNTIL Eof (input_file);
    Close (input_file);
END.
Exercise A8

PROGRAM illness;

VAR
    name: STRING [4];
    no_days, total, suspect_total: Real;
    datafile: Text;

BEGIN
    Assign (datafile, 'illness.dat');
    Reset (datafile);
    total := 0;
    suspect_total := 0;
    WHILE NOT Eof (datafile) DO
        BEGIN
            Readln (datafile, name, no_days);
            IF (name = 'fred') OR (name = 'jim') OR (name = 'paul') THEN
                suspect_total := suspect_total + no_days;
                total := total + no_days;
            END;
        END;
    Writeln ('Total employee absence = ', total:6:2, ' days.');
    Writeln ('Suspects total absence = ', suspect_total:6:2, ' days.');
    IF total > 0 THEN
        Writeln ('Suspects accounted for ', suspect_total / total:4:2, ' of the total absence');
    Close (datafile);
END.
Appendix B — Sample program documentation set

Exercise A2

Description

A program is required that will read a file of sales data and for each record report whether the transaction was excellent, average or poor. A total of all transactions is to be produced. However, it is known that SMITH made some refunds for £40 (which had been agreed with head office in advance) and so it is not required to take these particular refunds into account. All other refunds are to be counted when calculating the total. The input file has one record per-line of the form:

Name (10 characters) followed by Sales_value (integer)

The file is terminated by a record with a name of ZYGOTE and a sales value of zero. Note, there may be valid sales/refunds (i.e., non-zero values) by ZYGOTE within the file.

Notice in the expected output that the refund of £40 given by SMITH (see input data) has not been subtracted from the total value of transactions (remember the refund policy described above). The program design, in the form of pseudo-code is given below:

BEGIN
Open input file
Initialise variables as required
Read first record
WHILE not end of data DO
BEGIN
  IF sales value > £100 THEN
    Report 'Excellent'
  ELSE IF sales value > £50 THEN
    Report 'Average'
  ELSE IF sales value > £0 THEN
    Report 'Poor'
  ELSE
    Report 'Refund';
  IF record not a SMITH authorised refund THEN
    Accumulate sales total
    Accumulate counter
  Read next record
END
Display count of transactions
Display total transaction value
Close input file
END.

The input and output data are given on the next page.

Input Data

SMITH 101
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>JONES</td>
<td>51</td>
</tr>
<tr>
<td>DAVIES</td>
<td>50</td>
</tr>
<tr>
<td>ZYGOTE</td>
<td>200</td>
</tr>
<tr>
<td>SMITH</td>
<td>-40</td>
</tr>
<tr>
<td>JONES</td>
<td>-40</td>
</tr>
<tr>
<td>SMITH</td>
<td>-50</td>
</tr>
<tr>
<td>JOHNSON</td>
<td>-10</td>
</tr>
<tr>
<td>ZYGOTE</td>
<td>0</td>
</tr>
</tbody>
</table>

### Expected Output

<table>
<thead>
<tr>
<th>Name</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>Excellent (101)</td>
</tr>
<tr>
<td>JONES</td>
<td>Average (51)</td>
</tr>
<tr>
<td>DAVIES</td>
<td>Poor (50)</td>
</tr>
<tr>
<td>ZYGOTE</td>
<td>Excellent (200)</td>
</tr>
<tr>
<td>SMITH</td>
<td>Refund (-40)</td>
</tr>
<tr>
<td>JONES</td>
<td>Refund (-40)</td>
</tr>
<tr>
<td>SMITH</td>
<td>Refund (-50)</td>
</tr>
<tr>
<td>JOHNSON</td>
<td>Refund (-10)</td>
</tr>
</tbody>
</table>

The number of transactions made was 8
The total value of transactions was 302

### Actual Output

<table>
<thead>
<tr>
<th>Name</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>Excellent (101)</td>
</tr>
<tr>
<td>JONES</td>
<td>Average (51)</td>
</tr>
<tr>
<td>DAVIES</td>
<td>Poor (50)</td>
</tr>
</tbody>
</table>

The number of transactions made was 3
The total value of transactions was 202