Experimental Comparison of the Comprehensibility of a Z Specification and its Implementation in Java

C.F. Snook
Dept. of Electronics and Computer Science
University of Southampton
Highfield, Southampton, SO17 1BJ, UK.
cfs@ecs.soton.ac.uk

R. Harrison
School of Computer Science, Cybernetics & Electronic Engineering,
University of Reading, Reading RG6 6AY, UK
Rachel.Harrison@reading.ac.uk

Key Words - Empirical Assessment, Formal Specification, Comprehension,

Abstract
Comprehensibility is often raised as a problem with formal notations, yet formal methods practitioners dispute this. In a survey, one interviewee said "formal specifications are no more difficult to understand than code". Measurement of comprehension is necessarily comparative and a useful comparison for a specification is against its implementation. Practitioners have an intuitive feel for the comprehension of code. A quantified comparison will transfer this feeling to formal specifications. We performed an experiment to compare the comprehension of a Z specification with that of its implementation in Java. The results indicate there is little difference in comprehensibility between the two.

1 Introduction
Formal methods have long held the promise of providing a much-needed solid engineering foundation for the ‘art’ of programming computers. Formal specifications can be used to provide an unambiguous and precise supplement to natural language descriptions and can be rigorously validated and verified leading to the early detection of specification errors. Experiential reports of their use have invariably been favourable and yet still the adoption of formal methods has been limited. Academic interest in formal methods has been lively with many active research groups throughout the world and plenty of conferences dedicated to their discussion. Despite this interest, uptake within industry has mainly been limited to safety critical applications (some due to mandate by regulatory authorities) and experimentation by a few pioneering market leaders. It seems that practitioners, in their constant search for an edge in productivity and quality are keeping an eye on formal methods but judge them to be insufficiently beneficial to outweigh pragmatic problems. However, proponents have countered popular myths that dubious practitioners have raised [9,2]. Formal specification is the first step to using formal methods and is, in itself, a useful activity even if the formal specifications are not subsequently used in a full formal development. However, even this first step is not being adopted to any great degree within the industry. Perhaps academia is not prioritising the problems it researches to the greatest effect. Targeting the pragmatic problems that practitioners initially face would lead to increased interest and funding from industry, and a more widespread take-up of formal specification would later lead to faster development of subsequent research in academically appealing areas of formal methods.

Since formal specification is the first step to using formal methods it is also the first barrier that must be overcome if the benefits of full formal methods including refinement and verification is to be achieved. Our research [13] explores some of the barriers to the widespread use of formal specification in industry. While we cannot hope to explain all such barriers, the aim is to make some progress in understanding what some of the barriers are and to evaluate them. Formal specification
bears many similarities with program design. It is convenient and useful when thinking about barriers to formal specification, to think about whether similar barriers exist in programming; and if so, how they have been overcome. The comparison with programming is useful because programming is a more developed and researched area. It is also the main activity and primary goal of the people that we would like to help overcome the barriers to formal specification. These people have a good intuitive ‘feel’ for attributes of programming, making comparisons meaningful in a practical sense. A more concrete, aim therefore is to compare the activity of formal specification with that of programming.

It is a common perception that one of the problems with formal notations is that they are difficult to understand and that highly trained mathematicians are needed to read them. In previous work [12] we surveyed the opinions of industrial experts and found that experienced interviewees thought that typical software engineers have no real difficulties with understanding formal notations. As one interviewee put it, formal specifications are no more difficult to understand than code. Here we design and conduct an experiment to test this, by writing a specification using Z and implementing it in a programming language. A close correspondence is maintained between the specification and the implementation, both in functionality and in structure. Subjects were given either the formal specification or the code and their understanding was tested using questionnaires. The results indicate that there is little if any difference in comprehensibility between the two. Further work is required (in the form of repeated experiments) to provide confidence in this result and even then we do not claim that it is the only barrier to be overcome or that formal specification will always be worthwhile in every case.

Apart from providing results, the experiment is interesting as an example of the difficulties of empirical assessment when measuring human behaviour in relation to software activities. One problem is that traditional ‘null hypothesis statistical testing’ techniques are not suited to showing that one treatment is ‘no worse’ than another. The problem is similar to that found in the pharmaceutical industry for bioequivalence studies when it is wished to show that a drug or treatment method is equivalent to an existing treatment [17]. Our analysis method is in line with current standards for bioequivalence.

2 Description of Experiment

The objective of the experiment was to investigate the theory that formal specifications are no more difficult to read than code. Since comprehensibility is a complex attribute for which we have no absolute measures we need to test this theory by measuring comprehension between two examples that are comparable in some sense. Many attributes could affect this comparison such as size, structure and inherent problem complexity. In order to make the link as tangible as possible we chose to compare a Z specification with its implementation. We do not expect to use this result to conclude whether formal specifications should be used. There are many other factors requiring empirical assessment before a conclusion can be reached. However the comparison with implementation is attractive because the community of potential formal specification users is likely to have extensive experience of code maintenance and hence a 'good feel' for comprehension of code. Having a comparative measure for a specification couched in terms of the comprehensibility of its implementation will transfer this 'good feel' to the realm of formal specification. Therefore the theory can be re-phrased as "a Z specification is (at least) as understandable as its implementation". To investigate this a Z specification of an example system was constructed. This was then implemented in the Java programming language. Subjects were asked to describe either the functionality represented by the specification or by the code. The mean level of understanding of each group (specification or code) was compared.

2.1 Hypothesis

The substantive hypothesis is that comprehension of formal specifications is not a problem. The alternative substantive hypothesis is that comprehension of formal specifications is a problem. Since this hypothesis contains subjective and difficult to measure attributes such as comprehension and
when to what degree it is a problem, we design an experiment of a more specific hypothesis. The degree to which the experimental hypothesis supports the subjective hypothesis is discussed in section 4. The experimental hypothesis is that ‘a Z specification is (at least) as understandable as its implementation in Java’. In terms of the measured variables of the experiment this is equivalent to: The average scores of the subjects reading the Z specification will be approximately the same as the average scores of the subjects reading the Java implementation.

2.2 Experimental Design

The experiment examines the effect of a single factor or independent variable, notation, on the measured dependent variable, comprehension. The factor has two levels, Z or Java resulting in two alternative treatments, comprehension of Z and comprehension of Java. It may have been possible to block the subjects according to ability, however all the subjects (students) had roughly similar backgrounds and experience. Blocking would therefore have given little benefit and merely complicated the statistical analysis. Hence, we decided merely to randomize the assignment of subjects to treatments.

The experiment was one-way [6]. This means that only one example problem was used and its description in each level of the factor was applied to different sets of subjects. The Subjects were split into 2 equal sized groups by random distribution of the experimental materials. A 2-way experiment (where 2 examples are used so that each subject attempts each of the treatment types) would have provided more statistical power but it was felt that doubling the effort involved would deter many of the volunteers. Another difficulty with 2 way experiments is that a second example is needed which is closely equivalent to the first but is also different enough to avoid significant learning effects. Although the one-way design has less statistical power (i.e. confidence levels) than a two-way design the method is just as valid. The statistical analysis performed gave good results and we feel the decision to use a one-way design was the correct one.

The subjects were given as much time as they required and were asked to record the time they had taken. (There was a 50 minute timetable slot, but all completed within this limit). They were then free to leave the room. It is hoped that this induced the subjects to work as efficiently as possible. The data are analysed below taking into account the time taken by each subject so that the effect of differing work rates can be accounted for.

2.3 Consideration of Influencing Attributes

Attributed that influence the comprehensibility of a Z specification have been investigated by Finney, Fenton and Fedorec [7] and Finney, Rennolls and Fedorec [8]. Here we attempt to eliminate such attributes in order to compare comprehensibility with that of an implementation.

Comprehensibility might be affected by structure [7]. The same system could be modelled in Z in many ways. Different specification structures could be adopted without changing the meaning of the model. Similarly the implementation could be structured in many ways and this might affect the comprehensibility of the implementation. To avoid the introduction of un-quantifiable influences on comprehensibility due to differing choices of structure, the specification and code were written with the same structure and naming of abstractions. There is a close correspondence between the schema and data entities in the Z specification and the component modules in the Java code. This may mean that to some readers the Z specification, or Java code appears to be unnaturally structured, however, eliminating this unknown quantity and concentrating on comparing the primary notations independently of structure increase the validity of the experiment in some sense. The effect of structure on the comprehensibility of Z specifications and Java code would be an interesting topic for subsequent work. Secondary notations play an important part in comprehension [10] and an alternative experiment might measure the difference in comprehension between items that employ the current accepted practice in style. For example the Java program could use an overloading of the equals method rather than the new method called sameas. Private variable scoping and get methods could be used instead of allowing direct inter-class access.
Similarly no commenting has been used in the Z specification or in the Java code. This is unnatural in both cases; one would not normally be expected to understand specification or code without a natural language explanation. However, if natural language commentary were provided in the experimental materials, the measure would no longer be of the comprehensibility of the notations. It would be severely and un-quantifiably influenced by the natural language descriptions. In general we do not expect experiments to reflect realistic scenarios. We construct experiments to eliminate influencing factors in order to measure the remaining effect of isolated variables.

2.4 Subjects

The 36 subjects were 2nd year computer science students who had been taught a course on formal methods and a similar length course on the Java programming language. The subjects were therefore familiar with the notations used, but were not very experienced. The experiment was voluntary, so there may be some self-selection effects, but since the allocation of either the Z specification or Java code was random and unknown to the subjects this should have no bias effect on the experiment. The entrance requirements for the degree course at the University of Southampton are relatively high and we would expect the students to be better than average in mathematical subjects. This should be borne in mind when considering the results.

One threat to validity may be that although the subjects have been taught to equivalent levels in these particular notations, they are likely to be more familiar with reading program code in general than reading formal notations. This would bias the results in favour of understanding the Java code. Similarly the subjects’ lecturers made several comments to the effect that the subjects did not like using formal methods. In order to confirm the lecturers’ comments we asked computer science students at The University of Southampton whether they liked using formal methods such as Z and B, and whether they liked using graphical design notations such as UML. Of the 118 students that responded, 67% preferred using graphical design notations and 15%, formal methods. There may be a self-fulfilling lack of confidence in the subjects’ abilities to read the Z specification leading to another bias towards the Java code.

2.5 Experimental Materials

A short specification was written in Z (Appendix A) to describe a road layout with vehicles moving along the roads and across the junctions. The specification was then implemented in Java (Appendix B). The Z specification was structured according to an abstract data type paradigm so that it was possible to maintain a close correspondence in terms of structure and allocation of functionality with the Java implementation. The Z specification and Java implementation are shown in the appendices.

2.6 Conduct

The subjects were allocated to one of the descriptions (Z or Java) at random. This was done by randomly distributing a set of envelopes (equal in number to that of the subjects) half containing Z specifications, the other half Java code. In order to ensure that the person marking the answer sheets did not introduce any bias, they were marked blind so that the marker was unaware to which representation (Z or Java) they related.

2.7 Data Collection Procedures

The subjects were given a questionnaire (Appendix C) to test their comprehension of the description they had been given. The questions asked were very open. The subjects were asked to describe the real-world objects and behaviour represented by the complete description and then asked what a particular named section of the description represented in real-world terms. The openness of the questions has the disadvantage that it allows a wider scope for interpretation by the subjects of what the required answer is. However, it was found to be impossible to construct more specific questions that would reflect comprehension without strongly suggesting the answer within the question. Since the results consisted of an English language description of the system, we were concerned to ensure
that the interpretation of the answers did not introduce experimental error. An answer sheet (Appendix D) was prepared which listed all the points that a subject might mention in describing the functionality of the system. A subject gained one mark for each point that was mentioned at some point in their answers. The answer sheet thus made the interpretation of answers as objective as possible.

Additional background questions were asked in case such qualitative information might aid understanding of results. This information provided support for the validity of the experiment.

3 Analysis of Results

Statistical analysis techniques assess the likelihood of the recorded sample against a known or assumed population distribution. The more powerful parametric methods assume that the underlying population is normal. They provide the most definitive results because they use all the available information in the data. If the normality of the parameter’s distribution is in doubt then more robust methods should be used. One such class of methods are non-parametric methods that reduce the data to an ordinal scale and make use of ranking properties. Rank statistics obey a normal distribution even when the parameter itself does not, however, because information has been discarded, the results are usually less powerful than parametric methods. A comparatively modern technique is ‘bootstrapping’ or ‘resampling’. These techniques utilise computer processing to take many samples from the original sample and calculate the statistic of interest for each of these resamples. Bootstrap techniques do not make assumptions about the underlying population distribution, but can be just as powerful as traditional parametric analysis techniques.

3.1 Variables

The independent variable is the notation (Z specificaiton or Java code) used for the description. Two dependent variables are analysed. Firstly, the score which is an integer value ranging from 0 to 22 representing the number of marks gained as a measure of comprehension. Secondly the rate of scoring was found by dividing the score by the time taken. This was used as an alternative measure of comprehension.

3.2 Method of Analysis

When performing comparative experiments we are usually interested in detecting a difference in some attribute under two treatments. Following the classical null hypothesis statistical testing process (NHSTP) we would construct a null hypothesis stating that there is no difference and attempt to reject this on the basis of the sample data being unlikely if it were so, leaving an alternative hypothesis that there is a difference. Here, our substantive hypothesis is that there will be no significant (in the practical sense) difference. Unfortunately not rejecting a null hypothesis is a much weaker result; all we may say is that this sample didn't cause us to reject the null hypothesis. It does not give us any basis for saying that the null hypothesis is likely to be true or any evaluation of its probability. A brief glance at the means and standard deviations in Table 1 is enough to confirm that, as expected, a standard test such as the t-test would not reject a null hypothesis.

One way round this problem would be to take the approach that a null hypothesis is a hypothesis that we wish to nullify (rather than one of no difference). Then we could formulate the null hypothesis that Z is more difficult to understand than Java and see if we can reject it. However this would require us to arbitrarily define what we mean by a difference [11]. Note that it would invalidate the NHSTP method if we were to choose this definition in the light of our sample data. Traditionally, when we reject a null hypothesis the meaning of 'different' is not discussed because it 'falls out' of the statistical analysis. A 'difference' is that magnitude such that a sample of differences greater than this magnitude would be unlikely to occur by chance if the 'no difference' hypothesis were true. Hence when we talk about statistically significant differences we are referring to the reliability of the evidence that there is a difference and not to the importance of the magnitude of the difference. Chow gives a good overview of criticisms of NHSTP (as well as making a case in its favour) in his book 'Statistical
Significance' [4]. Further criticism of the misuse of NHSTP is given by Bakan [3] and Rozeboom [11]. Many statistical authors (e.g. Wonnacott and Wonnacott [16]) recommend using confidence intervals to explain the results of experiments rather than NHSTP, and we take this approach partly due to our problem with the null hypothesis but also because it is more informative and less reliant on arbitrary choices of criteria.

Initially we constructed confidence intervals using parametric methods, which assume that the population distribution is a normal distribution. Examination of the sample data for score revealed that it is not obviously skewed, and roughly approximates a normal distribution, but this does not guarantee that the population distribution is normal. In fact the data is fundamentally non-normal because it is truncated at 0. We should therefore treat parametric analysis with some mistrust. For the sample data for rate the distribution appears even less normal. Therefore, we construct confidence intervals based on non-parametric bootstrap methods, which make no assumptions about the underlying population distribution other than the sample data is representative of it.

### 3.3 Examination of Data

The size of the data samples for the Z specification and the Java program were both 18. Each sample consisted of a score out of a maximum 22 marks and the time taken by the subject in minutes. A measure of the rate of scoring was obtained by dividing the score by the time taken. The experimental results are given in Appendix E.

An initial look at the medians, means and standard deviations (Table 1) of the data indicates that the Z and Java results appear to be very similar in both score, and rate of scoring. The most significant difference between the Z and Java results is in the standard deviation of the rate of scoring, which shows that the rate of scoring varies significantly more between subjects when reading a Z specification than when reading code. This is despite the fact that, when time is not taken into account, score varies less when reading a Z specification than when reading code.

We also examined histograms (using SPSS) showing the actual data and a superimposed normal distribution curve (Fig. 1). This showed a fairly good fit but with a slightly high proportion of readings around the mean, indicating a low standard error. For the rate of scoring data (Fig. 2) the histograms appear to be skewed towards the lower end indicating that this data is not a very good approximation to a normal distribution.

### 3.4 Bootstrap Confidence Intervals

In case the distributions are non-normal we used a robust bootstrap analysis [5]. This works upon the assumption that the data sample is representative of the real population. It does not make any assumptions about the nature (e.g. normality) of the real population distribution. Samples of the same size as the original sample are taken repeatedly from the sample data (it is permitted to select the same data point more than once within a sample). The statistic of interest is calculated for each sample and plotted to give a distribution that approximates its distribution in the real population. From this distribution a confidence interval can be deduced for any confidence level. We used MathSoft’s S-PLUS 2000 (Professional Release 2) statistics package to perform the bootstrap calculations. Despite the robust nature of the bootstrap analysis, the confidence interval gives a ‘better’ answer (i.e. tighter margin at the same confidence level) than the traditional parametric confidence interval.

**Score.** The bootstrap calculation for mean(java score)-mean(Z score) gives a difference in means of 2.22 at the 95% confidence level (25% expressed as a percentage of the mean for the Java sample). Hence we have a 95% confidence that the overall population would have a mean Z score no worse than 75% of the Java score.

```r
*** Bootstrap Results ***
bootstrap(data = just.the.data,
    statistic = mean(jscore) - mean(zscore),
    B = 20000, trace = F, assign.framel = F, save.indices = F)
```
Number of Replications: 20000
Summary Statistics:
<table>
<thead>
<tr>
<th></th>
<th>Observed</th>
<th>Bias</th>
<th>Mean</th>
<th>SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>0.5556</td>
<td>-0.006478</td>
<td>0.5491</td>
<td>1.053</td>
</tr>
</tbody>
</table>

Empirical Percentiles:
<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>5%</th>
<th>95%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>-1.5</td>
<td>-1.16667</td>
<td>2.277778</td>
<td>2.611111</td>
</tr>
</tbody>
</table>

BCa Percentiles:
<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>5%</th>
<th>95%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>-1.55556</td>
<td>-1.222222</td>
<td>2.222222</td>
<td>2.611111</td>
</tr>
</tbody>
</table>

*** Bootstrap Results ***
Call:
bootstrap(data = ADJUSTED.DATA1, statistic = mean(Z.score) - mean(Java.score),
B = 20000, trace = F, assign.frame1 = F, save.indices = F)

Number of Replications: 20000
Summary Statistics:
<table>
<thead>
<tr>
<th></th>
<th>Observed</th>
<th>Bias</th>
<th>Mean</th>
<th>SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>-0.3827</td>
<td>0.0006198</td>
<td>-0.3821</td>
<td>0.7068</td>
</tr>
</tbody>
</table>

Empirical Percentiles:
<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>5%</th>
<th>95%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>-1.771605</td>
<td>-1.54784</td>
<td>0.7808642</td>
<td>1.003086</td>
</tr>
</tbody>
</table>

BCa Percentiles:
<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>5%</th>
<th>95%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>-1.77095</td>
<td>-1.54784</td>
<td>0.7824074</td>
<td>1.003086</td>
</tr>
</tbody>
</table>

The bootstrap density distribution of mean Java score – mean Z score for the 20,000 bootstrap resamples was obtained from Splus (fig. 4).

**Rate.** The bootstrap calculation for mean(java rate)-mean(Z rate) gives a difference in means of 0.082 at the 95% confidence level (18% expressed as a percentage of the mean for the Java sample). Hence we have a 95% confidence that the overall population would have a mean Z score no worse than 82% of the Java rate.

*** Bootstrap Results ***
Call:
bootstrap(data = just.the.data,
statistic = mean(jrate) - mean(zrate),
B = 20000, trace = F, assign.frame1 = F, save.indices = F)

Number of Replications: 20000
Summary Statistics:
<table>
<thead>
<tr>
<th></th>
<th>Observed</th>
<th>Bias</th>
<th>Mean</th>
<th>SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>-0.02692</td>
<td>0.0003885</td>
<td>-0.02653</td>
<td>0.07046</td>
</tr>
</tbody>
</table>

Empirical Percentiles:
<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>5%</th>
<th>95%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>-0.1668326</td>
<td>-0.1430569</td>
<td>0.08754796</td>
<td>0.109355</td>
</tr>
</tbody>
</table>

BCa Percentiles:
<table>
<thead>
<tr>
<th></th>
<th>2.5%</th>
<th>5%</th>
<th>95%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Param</td>
<td>-0.1744356</td>
<td>-0.1489462</td>
<td>0.0821454</td>
<td>0.102708</td>
</tr>
</tbody>
</table>

The bootstrap density distribution of mean Java rate of score – mean Z rate of score for the 20,000 bootstrap resamples was obtained from Splus (fig. 5).

### 3.5 Analysis of Qualitative Data

The questionnaire included some questions to collect some subjective, qualitative data. (See questions 3 to 7 of Appendix C). (One subject in the Z group did not complete these questions). We are careful
not to draw firm conclusions from this data due to inevitable variations in interpretation of both the questions and the answers. The following summarises the responses to these questions.

In question 3 the subjects were asked how difficult they thought the specification or program was to understand compared to an English language equivalent. The answers were almost all positive (i.e. harder to understand than English) and there was very little difference between the answers for the Z spec and for the Java program. The means of the answers (interpreting the answers on a scale from -5 to +5) were +2.35 (Z) and +2.39 (Java).

In question 4 the subjects were asked how difficult they found mathematical subjects (i.e. to judge their mathematical abilities compared to their peers). Here there was more of a tendency towards 'easy' indicating that most subjects thought they had an aptitude towards mathematics. This was slightly more so in the Z group than the Java group (-1.65 versus -0.31), which may indicate a mathematical bias in favour of the Z group.

In question 5 the subjects were asked for their mathematical qualifications. All but 5 of the subjects had mathematics A-level. Three of the five without A-level mathematics were in the Java group, 2 in the Z group. This indicates a uniform mathematical ability throughout the two groups.

In question 6 the subjects were asked how much experience they had with the notation or language used in the specification or program. The form of the answers varied slightly, some referring to length of time in months and others referring to course modules or semesters. However, all the answers apart from two in the Java group indicate that they only have experience of the notation/language from a course module in the previous year. Two answers from the Java group indicated a frequent use of Java leading to more of a familiarity.

In question 7, subjects were asked for any other comments. Many left this blank but of those that offered comments seven (all from the Z group) said that Z or formal specification is difficult or more difficult than code, whereas only 3 (from the Java group) said that Java or code is hard to understand. In fact 4 (again from the Java group) said that programs are easy to understand. Hence there appears to be a tendency to believe that formal specifications are more difficult to understand than code. This has not been borne out by the results of this experiment but may be a bias towards understanding the Java. We believe that the most likely explanation for this is an unfounded perception that formal specifications are difficult to understand. This may have arisen because of less familiarity compared with programming languages in general.

4 Threats to Validity

The degree of credibility of any study depends on its validity. We have already discussed ‘Conclusion Validity’, the validity of the statistical analysis. In this section we consider other threats to the validity of the experiment and its conclusions [1,15].

4.1 Internal Validity

Defines the degree of confidence in a cause-effect relationship. Thus under this heading we must consider whether the subjects understanding of the specification and program could have been influenced by any factors other than the independent variable. There are 2 categories of factors that could be a threat here. The first category is attributes of the subject that might influence their understanding, such as ability or degree of training in relevant subjects. This was minimised by selecting the subjects from the same cohort of a course. There will still be differences in background and ability but the random allocation to groups should distribute such factors between the 2 groups. As with any sample method there is, however, always the chance that an unfortunate allocation has occurred. The second category is attributes of the materials other than the notational difference such as style. As discussed above, the structure, style, naming and font of the two descriptions were made consistent to eliminate these factors. A further threat to the internal validity was discovered after the experiment had been performed. The Java program had been tested in order to verify its correctness but the Z specification was only verified by inspection. Three errors were left undiscovered in the Z specification when it was used for the experiment. The errors are as follows:
1. The blank predicate part of the schema \textit{VehicleType} should either contain \textit{true}, or be omitted.

2. The identifier \textit{Destination already occupied}, used in the definition of \textit{Report}, should contain underscores instead of spaces,

3. The schema \textit{pickRoad} is incorrectly used as a function in the schemas \textit{moveNewRoad0} and \textit{destinationAlreadyOccupied}.

The first two errors are minor and unlikely to cause any misunderstanding or confusion to a reader. For these errors it is reasonable to assume that the subjects were able to easily identify the correction to the syntax if they noticed the error. The third error is more significant since a correction is not easily identified even if the intended meaning is recognised. If the errors made it more difficult for the subjects to understand the Z specification the support for our hypothesis is strengthened. However, since the subjects did not comment on the errors, and there does not appear to be a correlation between the errors and an area that was misunderstood, we assume that the subjects correctly deduced the intended meaning of the schemas. The limited experience of the subjects may have led to them assuming that there was no error, even if they did not recognise the syntax, and correctly guessing the meaning.

\section*{4.2 External Validity}

Defines the extent to which the conclusions from the experimental context can be generalised to the context specified in the research hypotheses. Having established the experimental hypothesis we must consider how well it supports the substantive hypothesis. There are several threats to the inductive process needed to assess the substantive hypothesis. Firstly, the notations used in the example are particular whereas the substantive hypothesis is general in terms of notations. However, both Z and Java are typical and representative of the majority of other notations. We feel that practitioners will accept that if the hypothesis is true for these notations then it is, to some extent, generally true. There may be notations that deviate one way or the other. For example, Java is an object-oriented language and procedural languages may be easier to understand (although, in the experiment, we have not used many object oriented concepts, such as inheritance, that are likely to affect understanding). However, similar experiments using alternative notations would clarify the generality in this respect.

Secondly, we must consider whether using students as subjects poses a threat to the validity of the experiment. The subjects were students who had undertaken an equivalent level of training in both notations. Lecturers reported that the students generally expressed a dislike of the formal notations. This is probably representative of the general population of practitioners in industry. We accept that students have less experience to rely on than practitioners. The extra experience of practitioners is likely to aid understanding of the program rather than the formal specification, but if our results reflect the situation without this bias in experience we view this as a desirable attribute. That is our results are more representative of the situation in the absence of a strong experiential bias (as might be found in industry) and therefore reflect the situation once an equivalent experience of formal specification has been obtained.

Thirdly, we should consider how the small size of the example problem affects the validity of the generality. This is a cause for concern, because the example problem is tiny compared with a real problem. Unfortunately it is impractical to use representative problems in this kind of experiment. We accept that scalability is an issue that could have a significant effect on the results. The experimental results therefore reflect the situation in the absence of scalability issues, which require further investigation.

\section*{4.3 Construct Validity}

Defines the extent to which the variables successfully measure the theoretical constructs in the hypotheses. The theoretical construct in the hypothesis is comprehensibility. Under construct validity we must therefore consider whether the dependent variable and its measure are valid measures of comprehensibility. The measure consists of 2 stages: an analogy between comprehensibility and being
able to describe the functionality of the represented system; and the validity of the scoring system used to measure the described functionality.

A threat to the first stage is that the subject may not have given a description that portrays their understanding. It seems reasonable to assume that the ability to describe something is proportional to the subject's understanding of it. This assumption is widespread in education via examination methods. The subject's written communication skills will affect their description as well as other factors such as their perception of what is relevant to the answer. However, these influences will not affect the validity of the results unless they affect one group significantly more than the other. We do not foresee any factors that could be influenced by the independent variable and hence might affect one group more than the other. (It may be that it is more difficult to describe the functionality of a program than a specification because of the difference in abstract level. However we consider this to be an essential part of what we are measuring rather than a source of bias. By 'comprehensibility' we mean ability to understand the functionality). The random assignment of subjects should therefore eliminate the effect of these factors, but as with any sample method there is always the chance that an unfortunate allocation has occurred.

The threat to the second stage is the method of scoring the written descriptions. The descriptions were marked according to a list of points (objects, properties or behaviour) and given one mark for each point mentioned. The answers were marked without knowledge of which group they belonged to so that no prejudice of the marker was introduced. Some points were easier to obtain than others and this means that the measure is non-linear affecting the scale validity. However, we feel that this will not be a significant problem as those who obtained harder marks generally obtained the easier marks. We considered weighting the points with differing amounts of marks but this would be a subjective judgement and in most cases it is not obvious what the weighting should be.

5 Possible Areas for Replication

Confidence in experimental results and further knowledge of influencing factors is gained by replication of experiments. Basili et.al. [1] discuss a framework for organising related sets of experiments with the aim of building up a complete picture of the results over a wide range of contexts. (The term 'replication' is generally taken to include variations in the experimental work as well as strict replications). An experiment (or other empirical assessment) using practitioners with varying degrees of experience would be useful to establish that the results may be generalised to industrial situations. The main limitation and threat to generalisability of the experiment was the small size of the example problem. Repetition using larger specifications and programs would be useful. The area of scalability and an evaluation of its importance to formal specification compared with program design would illustrate its effects on comprehensibility. Further work on the effects of different styles and structures on comprehensibility would also be an interesting and valuable area to explore. Existing work in this area includes that of Finney et.al. [7], who conducted an experiment that concluded that the degree of schema structuring in a Z specification affects its comprehensibility, schemas of approximately 20 lines being optimal. Vinter [14] conducted experiments that showed that subjects are likely to misinterpret certain forms of logical statements including disjunction, conjunction and quantification in the same way that people commonly misinterpret equivalent natural language descriptions. This implies that some forms will be more susceptible to misinterpretation than others, depending on context. In this experiment we have concentrated on eliminating the effect of secondary notation on comprehension by keeping the same structure and naming between the Z specification and Java program. An alternative and equally interesting experiment would measure the difference in comprehensibility when typical styles and practices are adhered to in both the Z specification and the Java program.

6 Conclusions

We set out with the intention of testing the substantive hypothesis that formal specifications are no more difficult to understand than code. Our experimental evidence strongly supports a hypothesis that subjects such as the ones we used could understand the Z version of the example approximately as
well as the Java version of the same example. The data recorded for the Z specification closely matches that for the Java. The means for both score and rate of scoring were very close. The variance for score was also closely matched but there does appear to be a slightly higher variance in the times taken for the Z specification. This may be due to a wider variation in mathematical background, familiarity and confidence.

We quantified the results in terms of confidence intervals for the usual 95% confidence level and found that we need to allow approximately a 25% margin, for score, and 18% margin for rate of scoring, to achieve this confidence (i.e. Z is within 25% as understandable as Java). Note that this does not mean that the data indicates that there is a 25% difference. (In fact, the data indicates that there is very little difference in comprehensibility).

We have chosen to adhere to the commonly used arbitrary confidence level of 95%. To give a guide to how the quantitative margin of the results would be improved by a looser choice of confidence level, we calculated alternative margins for the bootstrap result at the 80% and 75% levels. The corresponding results for scores were Z is within 18% and 14% as understandable as Java respectively. The corresponding results for rate of scoring were Z is within 7% and 4% as understandable as Java respectively.

In the previous section we discussed various threats to the validity of the results and in particular, threats to the generalisation of the experiment needed to support the substantive hypothesis. There are some areas that would benefit from further investigation, however, subject to these reservations, we conclude that formal specifications are no more difficult to understand than code. Consequently, industry should expect similar levels of effort in reading and understanding formal specifications as they already experience in reading and understanding programs provided they allocate similar resources to the task.

The threats to validity illustrate the difficulties involved in performing empirical assessments involving human performance. In particular the consideration of construct validity illustrates some of the difficulties of finding suitable and valid measures of complex attributes associated with human behaviour such as comprehension.

Acknowledgements

The authors wish to acknowledge the support of UK EPSRC, which has funded the Empirical assessment of formal Methods (EMPAF) project (GR/L87347) and a PhD studentship for this work. We would also like to thank the students who participated in the experiment and Professor Barbara Kitchenham for her helpful comments and suggestions.

References


Appendix A - Experimental Materials: Z specification

State

RoadType
- length : C
- length > 0

PositionType
- road : RoadType
- space : C
- space ∈ 1..road.length

VehicleType
- pos : PositionType

RoadsysType
- roads : P RoadType
goesto : RoadType \ RoadType
dom goesto = roads
ran goesto ⊆ roads

Traffic
- roadsys : RoadsysType
- vehicles : P VehicleType
- A v ∈ vehicles | v.pos.road ∈ roadsys.roads
- A v, w ∈ vehicles | v.pos.road = w.pos.road,
  v.pos.space ∈ w.pos.space

Initialisation

TrafficInit
- Traffic:
  - vehiclesinit? : P VehicleType
  - roadsysinit_roads? : P RoadType
  - roadsysinit_goesto? : RoadType \ RoadType
- vehicles = vehiclesinit?
- roads = roadsysinit_roads?
- goesto = roadsysinit_goesto?

Operations

Report ::= Okay | Destination already occupied

Success
- error! : Report
- error! = Okay

moveSameRoad
- : Traffic
- pos? : PositionType
- pos?.road ∈ roadsys.roads
- pos?.space < pos?.road.length
- v ∈ vehicles |
  v.pos.road = pos?.road
  v.pos.space = pos?.space+1
- w ∈ vehicles |
  w.pos.road = pos?.road
  w.pos.space = pos?.space+1
- vehicles = vehicles \ {v : VehicleType | v.pos.road = pos?.road |
  v.pos.space = pos?.space+1}
- roadsys = roadsys
pickRoad
roadset? : ∈ RoadType
road! : RoadType
roadset? ‰ 1
road! x roadset?

moveNewRoad₀

: Traffic
pos? : PositionType

pos?.road x roadsys.roads
pos?.space = pos?.road.length
; v : vehicles { v.pos = pos? }
$ ; w : vehicles { w.pos.road =
      pickRoad roadsys.goesto pos?.road | w.pos.space = 1 }
vehicles (= vehicles J { v : VehicleType | v.pos.road =
      pickRoad roadsys.goesto pos?.road | v.pos.space = 1 })
\ { v : vehicles | v.pos = pos? }
roadsys := roadsys

destinationAlreadyOccupied

\ Traffic
pos? : PositionType
error!: Report

pos?.road x roadsys.roads
; v : vehicles { v.pos = pos? }
((pos?.space < pos?.road.length |
; w : vehicles { w.pos.road = pos?.road |
  w.pos.space = pos?.space+1 })
\ (pos?.space = pos?.road.length |
; w : vehicles { w.pos.road = pickRoad roadsys.goesto pos?.road
  | w.pos.space = 1 })
error!=Destination already occupied

moveVehicle :
((moveSameRoad₀ ; moveNewRoad₀) | Success)
\ destinationAlreadyOccupied
Appendix B - Experimental Materials: Java Code

import java.lang.Exception;
class InvariantException extends Exception {
    public InvariantException (String msg) {super(msg);}
}

class RoadType {
    int roadlength;
    public RoadType(int inp_length) throws InvariantException {
        if (inp_length < 1) {
            InvariantException e = new InvariantException
                ("Invariant: road length must be >= 1");
            throw e;
        }
        roadlength=inp_length;
    }
}

class PositionType {
    RoadType road;
    int space;
    public PositionType (RoadType inp_road, int inp_space) throws InvariantException {
        if (inp_space < 1 || inp_space > inp_road.roadlength) {
            InvariantException e = new InvariantException
                ("Invariant: position must be within road");
            throw e;
        }
        road = inp_road;
        space = inp_space;
    }
    public boolean sameas (PositionType inp_pos) {
        boolean same = false;
        if (inp_pos.road==road & inp_pos.space==space) same =true;
        return same;
    }
}

class VehicleType {
    PositionType pos;
    public VehicleType(PositionType inp_pos) {pos = inp_pos;}
    public void moveto(PositionType inp_pos) {pos = inp_pos;}
}

class RoadsysType  {
    RoadType[] roads;
    RoadType[][] goesto;
    public RoadsysType(RoadType[] init_roads,RoadType[][] init_goesto) throws InvariantException {
        roads = init_roads;
        if (init_goesto.length < roads.length) {
            InvariantException e = new InvariantException
                ("Invariant: all roads must go somewhere");
            throw e;
        }
        for (int i=0; i<roads.length; i++) {
            if (init_goesto[i].length == 0) {
                InvariantException e = new InvariantException
                    ("Invariant: all roads must go somewhere");
                throw e;
            }
            for (int j=0; j<init_goesto[i].length; j++) {
                if (!isaroad(init_goesto[i][j])) {
                    InvariantException e = new InvariantException
                        ("Invariant: invalid goesto road");
                    throw e;
                }
            }
            goesto = init_goesto;
        }
    }
    public boolean isaroad(RoadType inp_road) {
        boolean r=false;
        for (int i=0; i<roads.length; i++) {
            if (roads[i] == inp_road) r=true;
        }
        return r;
    }
}
public RoadType[] allgoesto(RoadType inp_road) {
    int i=0;
    while (roads[i] != inp_road) i++;
    return goesto[i];}

import java.util.Random;
class Pick {
    static Random r = new Random();
    static public RoadType pickroad (RoadType[] array) {
        int n=Math.abs(r.nextInt() % array.length);
        return array[n];}
}

class Traffic {
    RoadsysType roadsys;
    VehicleType[] vehicles;
    public Traffic(RoadType[] init_roads,RoadType[][] init_goesto,VehicleType[] init_vehicles) throws InvariantException {
        roadsys = new RoadsysType(init_roads,init_goesto);
        for (int i=0; i<init_vehicles.length; i++) {
            if (!roadsys.isaroad(init_vehicles[i].pos.road)) {
                InvariantException e = new InvariantException
                    ("Invariant: Vehicle not in valid road");
                throw e; }
            for (int j=0; j<init_vehicles.length; j++) {
                if (init_vehicles[i].pos.sameas(init_vehicles[j].pos) && i!=j) {
                    InvariantException e = new InvariantException
                        ("Invariant: 2 vehicles at same position");
                    throw e; }
            }
        }
        vehicles = init_vehicles; }

    public void moveVehicle(PositionType inp_pos) throws Exception {
        PositionType destination;
        if (inp_pos.space < inp_pos.road.roadlength) {
            destination = new PositionType(inp_pos.road,inp_pos.space+1);}
        else {
            RoadType exit=Pick.pickroad(roadsys.allgoesto(inp_pos.road));
            destination = new PositionType(exit,1);    }
        if (isVehicleAt(destination)) {
            InvariantException e = new InvariantException
                ("Invariant: Destination already occupied");
            throw e; }
        getVehicleAt(inp_pos).moveto(destination);    }

    public boolean isVehicleAt(PositionType inp_pos) {
        boolean found = false;
        if (vehicles != null) {
            for (int i=0; i<vehicles.length; i++) {
                if (vehicles[i].pos.sameas(inp_pos)) found=true;}}
        return found;  }

    public VehicleType getVehicleAt(PositionType inp_pos) {
        int i=0;
        while (!vehicles[i].pos.sameas(inp_pos)) i++;
        return vehicles[i];     }
}
Appendix C - Experimental Materials: Questionnaire

Your email address:

Please record the time taken for each of the first 2 questions including all of the time you spend reading the specification/program).

Q1. Describe the physical objects represented in the system and their behaviour (i.e. the functionality of the specification/program)  Time taken for Q1.:  mins

Q2. 'PickRoad' represents an indeterministic or random choice. In real-world, functional, terms what is it used for?  Time taken for Q2.:  mins

Q3. How difficult did you find the specification/program to understand compared to how you think you would have found an English language equivalent?

Easy O O O O O  o O O O O  Hard  (replace an O with an X)

Q4. How difficult do you find mathematical subjects? (i.e. what is your subjective judgement of your own mathematical abilities compared to your peers)

Easy O O O O O o O O O O  Hard  (replace an O with an X)

Q5. What training/qualifications do you have in mathematical (and related) subjects (e.g. GCSE, A-level Maths/physics etc)?

Q6. How much experience have you had with the notation/language used in the specification/program?

Q7. Any other comments? (or things that might have affected your answers)
## Appendix D - Experimental Materials: Marking Sheet

<table>
<thead>
<tr>
<th>Q1. Roads</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Roads are directional</td>
<td></td>
</tr>
<tr>
<td>Roads have a length</td>
<td></td>
</tr>
<tr>
<td>… which is greater than zero</td>
<td></td>
</tr>
<tr>
<td>Roads are modelled as a sequence of discrete positions</td>
<td></td>
</tr>
<tr>
<td>The end of each road is connected…</td>
<td></td>
</tr>
<tr>
<td>…to one or ….</td>
<td></td>
</tr>
<tr>
<td>….more other roads</td>
<td></td>
</tr>
<tr>
<td>Vehicles</td>
<td></td>
</tr>
<tr>
<td>Vehicles exist on a particular road</td>
<td></td>
</tr>
<tr>
<td>…at a particular position on that road</td>
<td></td>
</tr>
<tr>
<td>2 vehicles cannot occupy the same position</td>
<td></td>
</tr>
<tr>
<td>Vehicles can move along roads…</td>
<td></td>
</tr>
<tr>
<td>…one position forward at a time</td>
<td></td>
</tr>
<tr>
<td>…but only if the destination position is unoccupied</td>
<td></td>
</tr>
<tr>
<td>A vehicle at the end of a road…</td>
<td></td>
</tr>
<tr>
<td>…can move to another road…</td>
<td></td>
</tr>
<tr>
<td>…that is connected to its road…</td>
<td></td>
</tr>
<tr>
<td>…in fact any of the connected roads…</td>
<td></td>
</tr>
<tr>
<td>…the choice is random/undefined</td>
<td></td>
</tr>
<tr>
<td>…but only if the destination is unoccupied</td>
<td></td>
</tr>
<tr>
<td>Q2. it represents the vehicles/drivers choice of …</td>
<td></td>
</tr>
<tr>
<td>… which new road to enter</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>Total time taken: (Q1+Q2 = ? + ?)</td>
<td></td>
</tr>
<tr>
<td>marks per minute</td>
<td></td>
</tr>
</tbody>
</table>
## Appendix E - Experimental Results

<table>
<thead>
<tr>
<th>Time (mins)</th>
<th>Q1+2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6 (answers to Q6 and Q7 are summarised)</th>
<th>Q7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Z</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>10</td>
<td>2</td>
<td>2</td>
<td>A &lt;1yr</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>8</td>
<td>3</td>
<td>-2</td>
<td>B 1course</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>7</td>
<td>-1</td>
<td>-2</td>
<td>A module</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>-1</td>
<td>C little</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td>1</td>
<td>-2</td>
<td>A fair amount in course</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>29</td>
<td>8</td>
<td>5</td>
<td>-4</td>
<td>A module</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>25</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>A semester</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>27</td>
<td>13</td>
<td>1</td>
<td>2</td>
<td>A semester</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>10</td>
<td>3</td>
<td>0</td>
<td>A course</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>9</td>
<td>0</td>
<td>-3</td>
<td>A course</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>15</td>
<td>3</td>
<td>-3</td>
<td>A module</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>23</td>
<td>6</td>
<td>2</td>
<td>-3</td>
<td>A 2modules</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>35</td>
<td>5</td>
<td>2</td>
<td>-5</td>
<td>A some last term</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>33</td>
<td>6</td>
<td>3</td>
<td>-2</td>
<td>A little</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>13</td>
<td>3</td>
<td>5</td>
<td>-2</td>
<td>A 12x45min lectures</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>7</td>
<td>1</td>
<td>-1</td>
<td>A 1 module</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>7</td>
<td>3</td>
<td>-2</td>
<td>A 2 modules</td>
<td></td>
</tr>
</tbody>
</table>

| Java       |      |    |    |    |                                         |    |
| 19         | 22   | 8  | 0  | -4 | A 1yr                                   |    |
| 20         | 22   | 9  | 2  | 3  | A 6months                               |    |
| 21         | 19   | 4  | 2  | -1 | A 6months                               |    |
| 22         | 15   | 7  | 3  | -3.5| A 6months                               |    |
| 23         | 14   | 10 | 2  | -1 | A 6months                               |    |
| 24         | 15   | 3  | 2  | 1  | A some                                  |    |
| 25         | 14   | 8  | 2  | -3 | A module                                |    |
| 26         | 15   | 16 | 3  | -2 | A 10months                              |    |
| 27         | 21   | 10 | 2  | 0  | A 12months in depth                     |    |
| 28         | 22   | 12 | 3  | -3 | A module                                |    |
| 29         | 35   | 7  | 4  | 2  | A 3 to 4 months                         |    |
| 30         | 12   | 6  | 3  | 3  | B 1st yr                                |    |
| 31         | 25   | 7  | 3  | -3 | A 6 months                              |    |
| 32         | 25   | 9  | 1  | 1  | A module                                |    |
| 33         | 15   | 4  | 3  | 1  | B not much                              |    |
| 34         | 25   | 8  | 3  | 5  | B semester                              |    |
| 35         | 23   | 17 | 3  | -4 | A lots of Java                          |    |
| 36         | 27   | 14 | 2  | 3  | A fair amount - module                  |    |

Key for Q5: A=Alevel maths, B=GCSE maths C=maths as part of french baccalaureat
Table 1 - Summary of Results

<table>
<thead>
<tr>
<th>Variable</th>
<th>Z (marks)</th>
<th>Java (marks)</th>
<th>(Z-J)/J (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCORE</td>
<td>median</td>
<td>7.50</td>
<td>8.00</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>8.28</td>
<td>8.83</td>
</tr>
<tr>
<td></td>
<td>std.dev</td>
<td>3.37</td>
<td>3.90</td>
</tr>
<tr>
<td>TIME</td>
<td>median</td>
<td>20.00</td>
<td>21.50</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>20.17</td>
<td>20.33</td>
</tr>
<tr>
<td></td>
<td>std.dev</td>
<td>7.76</td>
<td>5.97</td>
</tr>
<tr>
<td>RATE</td>
<td>median</td>
<td>0.37</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>mean</td>
<td>0.48</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>std.dev</td>
<td>0.32</td>
<td>0.22</td>
</tr>
</tbody>
</table>
Figure 1 - Histograms of score
Figure 2 - Histograms of rate of scoring
Fig. 3 – Bootstrap density distribution for mean Java score – mean Z score for 20,000 resamples

Fig. 4 – Bootstrap density distribution for mean Java rate of score – mean Z rate of score for 20,000 resamples