Difficulties in Teaching and Learning the Java Programming Language

D. Adair, M. Jaeger
Australian College of Kuwait, Mishref, Kuwait, d.adair@ack.edu.kw;
Australian College of Kuwait, Mishref, Kuwait, m.jaeger@ack.edu.kw

Abstract
This paper is part of an ongoing effort to improve teaching and learning of the object-orientated Java programming language for novice programmers at undergraduate level. An examination of what makes the skill of programming in Java difficult to acquire is conducted using a method that integrates triangular fuzzy numbers and the analytic hierarchy process to develop a fuzzy evaluation model, which prioritizes the relative weights of difficulties found in the teaching and learning of the Java programming language. The results derived by the evaluation model can provide a reference for course designers and teachers seeking to reduce the difficulties of delivering Java programming courses.

Keywords: Object-orientated programming, learning effectiveness

1. Introduction
Java is an object-orientated, concurrent, distributed, platform-independent programming language and the object-orientated approach facilitates structuring programs through composition and inheritance. A design objective for the Java programming language [1] was to combine a complete, but minimal set of existing features, and it was specifically designed to be a safe and simple language with obscure or dangerous language features, such as pointer arithmetic and arbitrary pointer casts [2] purposely left out.

Programming using the Java language is a combination of knowledge of the programming tools and language, problem solving skills, and effective strategies for program design and implementation. Students however, find the skill of programming difficult to acquire in general [3, 4]. Teaching to program using the Java programming language has also proved to be problematic [5, 6]. This may be due to many factors ranging from difficulties within the subject and methods used to teach it through to expectations and attitudes of the teaching staff and their students. Problems within the subject could be lack of aptitude, cognitive factors such as learning styles and motivation, the difficulty of programming using multiple skills and processes, and the choice of the initial programming language [7]. It has also been found that unless instructors take special care to introduce the teaching material in a way that limits the complexity of the Java programming language, details can easily overwhelm introductory students [8].

Other factors to cause students not to progress well and quickly [7], may be that programming is a ‘novel’ subject, quite unlike other university subjects; programming can be perceived as boring; programming may have a bad reputation with the image of a "programmer" perceived as poor; and quite often, programming is taught in an unreasonable compressed time-scale.

A number of papers have been written which address some of the problems that often occur when teaching the Java programming language. Experiences have been related about teaching Java to first and second year undergraduate courses [9]. The findings of Clark et al. (1998) and others [10, 11] have identified object-orientated concepts as the fundamental knowledge which
students must know to perform well. Importantly, they state that introducing students to programming using the simpler procedural concepts only compounds this difficulty, as the need to always use some aspect of object-based code in Java cannot be avoided and that attempting to disguise this eventually causes frustration and confusion, even for good students. Recent surveys of the views of those working in the field of teaching Java programming have been provided by [12, 13].

Building on previous surveys of the perceived difficulties of Java programming language topics [14] and, taking into account the points made in the previous paragraphs, student surveys were carried out here to investigate difficulties of teaching and learning Java using a nine-point rating scale to express the perceived relative importance of any two criteria at the same level. The first survey concerned the general difficulties of teaching and learning the Java programming language while the second concentrated on the difficulties students perceive concerning the object-orientated aspects of Java. The results were analysed using the fuzzy analytic hierarchy process (AHP). This is a useful method for solving complex decision-making problems involving subjective judgment [15] (but does not fully represent human perceptions and judgments. Fuzzy set theory [16] does resemble human reasoning, and, by incorporating fuzzy set theory into AHP, the resulting fuzzy AHP enables a more accurate description of the multiple-attribute decision-making process [17]. The students surveyed here were novice programmers taught in a more conventional manner. Conventional here, means an approach to learning Java programming where the basics of the programming language are first taught and the students are then guided towards effective strategies for the more advanced programming skills. Direct referral to object-orientated programming comes about half way through the course. The survey was carried out at the end of a semester-long programming module, while they were in the second-year of an engineering undergraduate degree course.

The remainder of this paper is structured as follows: Section 2 briefly describes the fuzzy AHP model, Section 3 giving the empirical analyses of the surveys and Section 4 presents the research findings with discussion.

2. The Fuzzy AHP Approach

2.1 Hierarchical Structure and Uncertainty

The essential steps in the application of AHP is first to decompose a general decision problem in a hierarchical fashion into sub-problems that can be easily comprehended and evaluated, secondly to determine the priorities of the elements at each level of the decision hierarchy, and, thirdly to synthesize the priorities to determine the overall priorities of the decision alternatives.

The hierarchy of problems associated with teaching and learning Java programming needs to be established before performing the pair-wise comparison of AHP. After constructing a hierarchy of problems, each evaluator is asked to compare the elements at a given level on a pair-wise basis to estimate the relative importance. To compare, the scale used here is a nine-point scale which shows the participants’ judgments among the options as equally, moderately, strongly, very strongly or extremely preferred. The discrete scale of 1-9 has the advantages of simplicity and easiness of use, but does take into account the uncertainty associated with the evaluator’s judgment.

The AHP method [18] indicates that the eigenvector corresponding to the largest eigenvalue of the pair-wise comparisons matrix provides the relative priorities of the factors and preserves ordinal preferences among the alternatives. This means that if an alternative is preferred to another, its eigenvector component is larger than that of the other. A vector of weights obtained from the pair-wise comparisons matrix reflects the relative importance of the various factors. So assuming there are \( N \) number of decision elements, denoted as \( (E_1, \ldots, E_n, \ldots E_N) \), the judgment matrix would be \( A = [a_{ij}] \), in which \( a_{ij} \) represents the relative importance of \( E_i \) and \( E_j \).
Then, by using the row vector average normalization, the weight of $E_i$ is calculated as,

$$w_i = \frac{\left(\prod_{j=1}^{n} a_{ij}\right)^{1/n}}{\sum_{i=1}^{n} \left(\prod_{j=1}^{n} a_{ij}\right)^{1/n}}, \quad i, j = 1, 2, \ldots, n \tag{1}$$

where $w_i$ denote the weight of the $i^{th}$ decision element, and weight vector $\mathbf{w} = (w_i), i = 1, \ldots, n$.

The uncertainty found in human perceptions and judgments is captured using triangular fuzzy numbers, $\tilde{1}$ to $\tilde{9}$ to represent subjective pair-wise student perceptions of the problems of programming. A fuzzy number is a special fuzzy set $\tilde{F} = \{x, \mu_F(x) : x \in \mathbb{R}\}$ where $\mu_F(x)$ is a continuous mapping from the real line $\mathbb{R}$ to the closed interval $[0, 1]$. A triangular fuzzy number denoted as $\tilde{M} = (a, b, c)$ or $(a, \frac{b}{c})$, where $a \leq b \leq c$ has the following triangular-type membership function,

$$\mu_M(x) = \begin{cases} 
0 & x < a \\
\frac{x-a}{b-a} & a \leq x \leq b \\
\frac{c-x}{c-b} & b \leq x \leq c \\
0 & x > c 
\end{cases} \tag{2}$$

Here $a$ and $c$ stand for the smallest possible value and the largest possible value of the support of $\tilde{M}$, respectively, and $b$ is the most promising value of $\tilde{M}$ that describe a fuzzy event [19]. The triangular fuzzy number $\tilde{M}$ is shown on Figure 1.

![Figure 1: A triangular fuzzy number](image-url)
In fuzzy AHP, triangular fuzzy numbers are utilized to improve the scaling scheme in the judgment matrices [20], and interval arithmetic is used to solve the fuzzy eigenvector [21].

2.2 Fuzzy AHP Algorithm

A computer algorithm for the above methodology was written following the suggestions of [22]. The following briefly describes the components of the algorithm.

1. The scores of pair-wise comparison returned on the questionnaires are used to form pair-wise comparison matrices for each of the evaluators.

2. Fuzzy judgment matrices are constructed according [16] as,

\[ \tilde{A}^k = \begin{bmatrix} \tilde{a}^k_{ij} \end{bmatrix} \]  \hspace{1cm} (3)

where \( \tilde{A}^k \) is the fuzzy judgment matrix of evaluator \( k \) and \( \tilde{a}^k_{ij} \) are the fuzzy assessments between the criteria \( i \) and \( j \) of evaluator \( k \) and \( \tilde{a}^k_{ij} = (a^k_{ij}, b^k_{ij}, c^k_{ij}) \), where,

\[ \tilde{a}^k_{ij} = (1,1,1) \quad \forall \ i = j \quad \text{and} \quad \tilde{a}^k_{ji} = 1/\tilde{a}^k_{ij} \quad \forall \ i, j = 1, 2, ..., n \]

Here \( n \) is the number of the related criteria at that level.

3. A consistency test is then carried out according to [23] to check for any inconsistency within the judgments in each pair-wise comparison as well as for the entire hierarchy. If the consistency is not passed, the original values in the pair-wise comparison matrix are revised by the evaluator asking the student involved to carry out the pair-wise comparison again.

4. The next stage is to convert the fuzzy data into crisp scores [24]. This method can clearly express fuzzy perception, which is based on the procedure of determining the lower and upper scores by fuzzy minimum and fuzzy maximum, and the total score is determined as a weighted average according to the membership functions.

5. The aggregate crisp judgment matrix is now established. After undertaking the defuzzification method of (4) in order to perform the fuzzy aggregation procedure, the aggregate crisp judgment matrix for \( k \) evaluators, \( A^* \), is established as,

\[ A^* = \begin{bmatrix} \tilde{a}^*_{ij} \end{bmatrix} \]  \hspace{1cm} (4)

where,

\[ \tilde{a}^*_{ij} = \{1 \times 1 \times ... \times \tilde{a}^*_{ij} \} \]  \hspace{1cm} (5)

6. Equation 1 is now used to calculate the weight vertex for the aggregate crisp judgment matrix,

\[ W^* = \begin{bmatrix} \tilde{w}^*_{i} \end{bmatrix} \quad i = 1, ..., n \]  \hspace{1cm} (6)

for the aggregate crisp judgment matrix \( A^* \). The ranking order of the criteria is determined by the weight vertex \( W^* \).
3. Empirical analyses of the surveys

3.1 Building the Hierarchy Frameworks

A hierarchical structure, derived from reviewing relevant literature, of the general difficulties found in the teaching and learning the Java programming language is constructed as shown on Figure 2, with the definitions of evaluating sub-criteria or attributes given as Level 4. The overall objective is to evaluate the relative weights associated with the general difficulties.

A second hierarchical structure, derived from a previous study [14] and a review of the literature, was established for the difficulties students perceived concerning the object-orientated aspects of Java. The structure is shown on Figure 3 and the overall objective is to evaluate the relative weights associated with these difficulties.

Figure 2: Hierarchical structure of general difficulties in Java programming evaluation
3.2 Results of the fuzzy AHP approach for difficulties in teaching and learning Java

Based on the hierarchical structures described above (Figures 1 and 2), questionnaires were designed in the form of a pair-wise comparison. The conventional AHP questionnaire format of a nine-point rating scale was used to indicate the relative importance of each criterion or sub-criterion in the same hierarchy.

The students surveyed using the questionnaire were 39 undergraduates who were novice programmers, with an average age of 21.45 years, 19% female, 78% mechanical engineering students, 22% civil engineering students who had successfully completed a one semester
course on the Java programming language whose curriculum is reported in [14]. The survey was conducted at the end of a course conducted during the second semester of the second year. It was computer based, and of the thirty-nine returns for the ‘General difficulties evaluation’, two had to be eliminated due to multiple responses per comparison. The thirty-nine returns for ‘Difficulties with Java Language Topics’ were all used.

Next the fuzzy judgment matrices (FJM) of the evaluators were constructed with a typical matrix shown below for Level 2 of the general difficulties evaluation.

\[
FJM = \begin{bmatrix}
\text{Level 2} & C_1 & C_2 & C_3 \\
C_1 & 7 & 7 & 3^{-1} \\
C_2 & 7^{-1} & 7 & 5^{-1} \\
C_3 & 3 & 5 & 1
\end{bmatrix}
\]

A consistency test was carried out for each of the fuzzy judgment matrices using the method of [23], and, if the subjective judgments of an evaluator was found to be inconsistent, the student was asked to repair the pair-wise comparison until the consistency index was less than 0.1.

Defuzzification followed according to the recommendations of [24] to obtain crisp judgment matrices for the total evaluators. A typical crisp matrix (CM) for one evaluator for Level 2 of the general evaluation is shown below, as is the aggregate crisp judgment (ACJ) for the total evaluators, obtained using Eqs. (5) and (6).

\[
CM = \begin{bmatrix}
\text{Level 2} & C_1 & C_2 & C_3 \\
C_1 & 1.000 & 0.305 & 0.471 \\
C_2 & 3.258 & 1.000 & 4.522 \\
C_3 & 2.123 & 0.211 & 1.000
\end{bmatrix}
\]

\[
ACJ = \begin{bmatrix}
\text{Level 2} & C_1 & C_2 & C_3 \\
C_1 & 1.000 & 0.876 & 1.299 \\
C_2 & 1.094 & 1.000 & 1.378 \\
C_3 & 0.7576 & 0.6778 & 1.000
\end{bmatrix}
\]

The above was repeated for Level 3 of the general difficulties evaluation and Level 2 of the difficulties with Java language topics.

Finally the overall criteria weights (Equation 1) for both surveys were determined, and in order to compare all the quality factors at the same level of the hierarchical structures, the priority weights (including local weights and global weights) and ranking are also shown. The final priority weights and ranking of the two surveys are summarized in Tables 1 and 2.
Table 1: Weighted criteria and sub-criteria of 'general difficulties evaluation'.

<table>
<thead>
<tr>
<th>Weight</th>
<th>Criteria/sub-criteria</th>
<th>Weights of criteria level</th>
<th>Local weights of sub-criteria level (ranking)</th>
<th>Global weights of sub-criteria level (ranking)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cognitive difficulties (C₁)</td>
<td>0.349</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Teaching strategy (C₁₁)</td>
<td>0.411 (1)</td>
<td>0.143 (2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learning style (C₁₂)</td>
<td>0.403 (2)</td>
<td>0.141 (3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Motivation (C₁₃)</td>
<td>0.186 (3)</td>
<td>0.065 (9)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Programming difficulties (C₂)</td>
<td>0.383</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multiple skills (C₂₁)</td>
<td>0.348 (2)</td>
<td>0.133 (4)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Educational novelty (C₂₂)</td>
<td>0.224 (3)</td>
<td>0.086 (7)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multiple processes (C₂₃)</td>
<td>0.428 (1)</td>
<td>0.211 (1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Image difficulties (C₃)</td>
<td>0.267</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interest (C₃₁)</td>
<td>0.345 (2)</td>
<td>0.092 (6)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reputation (C₃₂)</td>
<td>0.270 (3)</td>
<td>0.072 (8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pace of instruction (C₃₃)</td>
<td>0.385 (1)</td>
<td>0.103 (5)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Weighted criteria of 'difficulties with Java language topics'.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Weights of criteria level (ranking)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods (J₁)</td>
<td>0.0776 (6)</td>
</tr>
<tr>
<td>Syntax (J₂)</td>
<td>0.0656 (13)</td>
</tr>
<tr>
<td>Applets(J₃)</td>
<td>0.0771 (7)</td>
</tr>
<tr>
<td>Streams (J₄)</td>
<td>0.0764 (9)</td>
</tr>
<tr>
<td>Threads (J₅)</td>
<td>0.0811 (2)</td>
</tr>
<tr>
<td>OOP (J₆)</td>
<td>0.0784 (4)</td>
</tr>
<tr>
<td>Exceptions (J₇)</td>
<td>0.0824 (1)</td>
</tr>
<tr>
<td>GUIs (J₈)</td>
<td>0.0780 (5)</td>
</tr>
<tr>
<td>Objects (J₉)</td>
<td>0.0749 (12)</td>
</tr>
<tr>
<td>JDK (J₁₀)</td>
<td>0.0759 (11)</td>
</tr>
<tr>
<td>Arrays (J₁₁)</td>
<td>0.0769 (8)</td>
</tr>
<tr>
<td>Strings (J₁₂)</td>
<td>0.0760 (10)</td>
</tr>
<tr>
<td>Files (J₁₃)</td>
<td>0.0797 (3)</td>
</tr>
</tbody>
</table>

In the above table the local weight is derived from judgment with respect to a single criterion.
4. Research Findings

In the pair-wise comparisons of AHP, triangular numbers were introduced to improve the scaling scheme of Saaty’s method. For this study, the resulting fuzzy AHP was used to evaluate the general difficulties of teaching and learning an object-orientated programming language, and to rank the relative difficulties of topics involved in teaching and learning the Java programming language.

4.1 General Difficulties

As can be seen from Table 1, the criterion ranked by the students of the survey as providing the most difficulty was Programming difficulties (C_2). This criterion covered the areas of Multiple skills (C_{21}), Educational novelty (C_{22}) and Multiple processes (C_{23}).

Of these three, Multiple processes (C_{23}) was ranked as the most difficult, both locally and globally. This would indicate that the students have a distinct difficulty in separating and prioritizing the two main processes involved in programming, i.e. translating the specifications of the problem into an algorithm and then translating the algorithm into the program code. When teaching it is clear that a strategy must be used to ensure both these processes are given at least equal weight time-wise for students to become successful programmers. It was observed during the course that students would concentrate (use a disproportionately amount of time) on the lower levels of say, ‘getting the syntax right’. It has been suggested [7, 25] that there is an intermediate stage, where the algorithm is first mapped to a pseudo-code, which comes with experience, and this stage should be formally part of the teaching strategy.

Multiple skills (C_{21}) was ranked second for difficulty under the criterion of Programming difficulties (C_2) and 4^{th} globally. Traditionally when learning to program, a student will learn lower level skills first, and then progress upwards [26]. The situation when learning object-orientated programming however is not so straightforward in that students are asked to comprehend some of the more difficult basics of object-orientated programming fairly early on, or right from the start of their course.

The third ranking under the criterion of Programming difficulties (C_2) was Educational novelty (C_{22}) which was also ranked in 7^{th} place overall. This finding is surprising as learning to programme in the Java language does require precision and problem solving. Also it has been suggested in the literature [12] that programming is indeed an educational novelty. However here the students did not recognize this as a difficulty. This could mean that they coped well with the intense effort of being very precise and the close scrutiny when problem solving, simply found other categories as more difficult, or there was a tendency towards denial and suppression of the difficulties [27].

The criterion ranked second in difficulty was Cognitive difficulties (C_1) which included the sub-criteria of Teaching strategy (C_{11}), Learning style (C_{12}) and Motivation (C_{13}).

Teaching strategy (C_{11}) was perceived as the most difficult sub-criterion under this criterion and the 2^{nd} most difficult globally. The teaching strategy used for these students consisted of a series of lectures, tutorials and laboratories. It is clear that this approach needs major revision, and a course consisting of seminars and laboratories is proposed. This may help take the tedium out of lectures on details of programming without the immediate opportunity to explore and create. The seminars would be less formal than lectures with their lengths being much less than the standard hour lectures but much more frequent. Assignments also could be revised with the tendency to use text-book exercises being replaced by those linked to other subjects in their engineering course.
Ranked 2nd under this criterion was Learning style (C.12) which also came a close 3rd globally. It could be that the tendency of the computer programming instructor, as was the case here, is to try to give a student a 'deep' understanding [28] of the subject. This may in fact be wrong for some students in that perhaps only a 'surface' approach is necessary as computer programming is more of a skill with underpinning knowledge.

The criterion ranked lowest was that of Image difficulties (C.3). This included the areas of Interest (C.31), Reputation (C.32) and Pace of instruction (C.33), with the main difficulty of these three expressed as the Pace of instruction (C.33).

However Pace of instruction (C.33) ranks in the middle of the global concerns expressed by the students. There does not seem much problem with the fact that the course was taught to a set time-scale, with a distinct ending and with set dates for assessment. Several factors may have contributed to this relative lack of concern, one being that the students were well established at the education institute without distractions of transition from secondary to tertiary education, and a second being that there was an abundance of tutorial help, both formal and informal provided by teaching assistants to alleviate the well known 'learned helplessness', where a student may feel or express that they "can't do programming!".

The other two sub-criteria, Interest (C.31) and Reputation (C.32) of the criteria image difficulties ranked very low globally as difficulties faced by the students. The reputation of computer programming as being difficult to this group of students, who had just completed their first computer programming course, seemed unimportant. Also there did not seem to be any concern with the image of computer programming, either positively or negatively, in general. There was evidence however that many of the students found computer programming boring and did not regard it as very creative, based on anecdotal comments during the course.

4.2 Difficulties with Java Language Topics

In the second evaluation the weightings of the perceived relative difficulties of the topics involved in the teaching and learning of the Java programming language were found. As will be seen from Table 2, Syntax (J.9), Objects (J.6) and JDK (J.10) were found to be the easier of the topics. Topics that were found to be the most difficult included Exceptions (J.7), Threads (J.5) and Files (J.13). Those found as the most difficult correspond to the study carried out by [29], but the topics perceived to be easy by this group of students differ with the same study, except for syntax.

It is important to note that the students did not find the topic Objects (J.6) as being particularly difficult. This has an implication for teaching programming in that an 'objects early approach' may be acceptable to the students. However the topic OOP (J.8), which included concepts such as inheritance and polymorphism, is considered difficult by the students. Inheritance and polymorphism are linchpins of the OO paradigm, and their proper use is necessary in order to obtain the maximum benefits of OO programming. It may be concluded from this that object-orientated programming may not be particularly difficult to teach but certain topics within OOP need great care, and proper time should be allocated to teaching and support for the proper use of inheritance and polymorphism.

The topic perceived as most difficult, Exceptions (J.7) may underscore the belief that a certain amount of programming maturity is necessary to fully understand the necessity for exceptions. Normally, to build confidence and allow students into the subject quickly, exercises and assignments are chosen that are 'safe' and not particularly prone to errors. When the time comes however, to introduce exceptions, to help make codes more robust by catching real-time errors, students often fail to see the seriousness of such a suggestion. They continue to write programs, which importantly still work, without any possibility of catching and dealing with runtime problems.
Unsurprisingly perhaps the topic which was perceived as the second most difficult by the students was Threads (J7), where multi-threaded applications deliver their potent power by running many threads concurrently within a single program. This is part of the area of class definitions and it is felt that more emphasis must be placed on deciding on class definitions as part of the analysis and design phase of a Java software project. It was noted during the course that students spent little time at the beginning of a project or assignment deciding on what threads to include and the urge to start writing code without proper planning was prevalent.

4.3 Validity

It should be noted that the ratings given by the students were subjective. Therefore, throughout this paper, whenever it has been reported that when the students find a feature easy or difficult in comparison to another, it should be remembered that what is reported is what the students perceived as easy or difficult when comparing.

There are some limitations in this work which would need further study and research. This study was conducted with reasonable sample size but the students were all from the same educational institute and background. More work could be done with a study using students from different educational institutes, countries and cultures. This would add to the evaluation power together with a more sophisticated evaluation process.

This study uses fuzzy AHP to develop an evaluation model to help educationalists when designing computer programming courses to understand the difficulties involved. It would be possible to adopt fuzzy multi-attribute approaches such as fuzzy TOPSIS [30] to calculate the relative weights of the difficulties of teaching and learning object-orientated programming.

The evaluation criteria were selected from a review of the literature, and it is possible that further criteria could be included. Further studies could use different methodologies, such as forming focus groups, interviews and longitudinal studies to identify a more complete list of criteria important to the teaching and learning of object-orientated computer languages.

Finally, having identified the difficulties and hence where to place emphasis, a newly designed course would need to be tested using a pre-test post-test experiment with a suitable control group to evaluate the practically of the fuzzy AHP evaluation procedure.

5. Conclusions

The acquisition of computer programming skills is complicated and there are many complicating factors within these skills. This paper has analysed some of these features and ranked them in degree of difficulty as perceived by a group of novice undergraduate students.

It is clear from the study that Multiple processes (C23) ranks as of most concern to the students and this would indicate a difficulty in being able to prioritize the two main processes involved in programming, i.e. translating the specifications of the problem into an algorithm and then translating the algorithm into the program code. A strategy must be used to ensure both these processes are given at least equal weight time-wise for students to become successful programmers.

Certainly, care should be taken when introducing the topics Exceptions (J7) and Threads (J5) with proper time allocation and support given. It must be made clear to students when and why Exceptions (J7) are used to encourage their inclusion in programming exercises. A strategy should also be in place to ‘slow down’ students who wish to get on and write code and to make them plan and incorporate Threads (J5) properly.
References


