RESEARCH REPORT

INVESTIGATING THE USE OF JELIOT IN A COGNITIVE CONFLICT/VISUALISATION STRATEGY FOR TEACHING PROGRAMMING CONCEPTS

Linxiao Ma
John Ferguson
Marc Roper
Isla Ross
Murray Wood

The University of Strathclyde, 2008
1. INTRODUCTION
Given the rapidly increasing influence of Information Technology and its associated high demand for skilled programmers, programming education is of significant global concern. However, current programming education seems far from successful, with many first year programming students performing much more poorly than expected in programming tasks [11]. This poor performance is undoubtedly a major contributor to the relatively high dropout rates, of around 30-50% [6], associated with Computer Science courses. While lack of problem-solving ability is viewed as the main cause of failure in programming learning (e.g.[1]), previous studies (e.g. [3]) found students often hold inappropriate understanding of key programming concepts which may cause difficulties in solving programming problems. Unfortunately, traditional approaches to learning how to program are less than ideal when trying to ensure students develop appropriate mental models of key programming concepts. Early investigations by the authors [9] found that a large number of students still held non-viable mental models of basic programming concepts, even after experiencing a one year programming course under traditional learning approaches. To improve students’ mental models a constructivist-based learning model that integrates a cognitive conflict strategy and program visualisation was proposed. The early evaluation of this learning model reveals that a cognitive conflict driven learning approach is indeed able to encourage students to engage with the learning materials and motivate them to construct viable mental models [10].

Based on the proposed learning model, a web-based learning environment has been developed to help improve students’ mental models of key programming concepts. This learning environment is intended to support instructors when employing the proposed learning model in their classes and provide students with a practical tool to learn key programming concepts. To evaluate the performance of this learning environment, the authors have been conducting a series of studies covering a range of key programming concepts. This report gives an overview of the learning environment and also presents the results from the series of empirical studies.

2. RELATED WORK
A study was conducted by the authors [9] to investigate the viability of mental models held by novice programmers based on the concepts of simple value assignment and the more challenging concept of object reference assignment. The results identified a variety of mental models of value and object reference assignment held by participants. Many of these models were seen as non-viable, meaning that they could result in a flawed understanding of the programs using these concepts. A quantitative analysis revealed that, at the completion of the first year course, only two thirds of the students held viable mental models of value assignment, with only 17% of students holding viable mental models of object reference assignment. This result is of significant concern. Both assignment and object reference are key concepts in object-oriented programming. The high failure rates in programming courses are not surprising if students still do not understand these basic programming concepts at the end of their introductory courses. The results also show that students with viable mental models performed significantly better in the course examination and programming tasks than those with non-viable mental models. This underlines how important it is to help novice programmers develop appropriate mental models of key programming concepts.

To facilitate novice programmers constructing viable mental models it is proposed that an approach to teaching programming that emphasizes constructivism [4] rather than
Objectivism [12] might be helpful. Objectivism claims that there is one true and correct reality. The learning process is to transfer the objective knowledge to a learner’s mind [12]. Constructivism argues that traditional approaches to teaching based on objectivism are too passive and do not do enough to challenge pre-existing ideas or to help students create viable mental models. Instead constructivism argues that students actively construct knowledge by combining the experiential world with existing cognitive structures [4].

One of the key teaching strategies based on a constructive perspective is cognitive conflict, which challenges students’ pre-existing ideas, motivating them to adopt more appropriate understandings. However, it should be noted that cognitive conflict alone is unlikely to be sufficient to achieve a change in non-viable models. Students must be supported to create new viable models, and concepts must be presented in an order and fashion that allows the correct construction of inter-dependent models. This is not an easy task, especially for programming students. As Lui et al. [7] have highlighted, “Computer programming is all fabricated that finds few parallels in the physical world”. The novice programmer lacks the necessary base knowledge for constructing viable models of programming concepts. Hence they often misuse their previous knowledge or adopt intuitive models. To address this, Ben-Ari has suggested that program visualisation has the potential to create a suitable learning environment [5]. Visualisation techniques have been used for over 20 years and have, arguably, not been as successful as hoped for. A possible reason for this is that they have been used from a traditional, objectivist perspective, ignoring a student’s pre-existing models. It is therefore proposed that a potential way forward is to adopt an approach based on cognitive conflict that helps students realise that there is a problem with their current understanding, and to use a visualisation-oriented learning environment to support them in adopting viable models.

The proposed learning model integrates a cognitive conflict strategy together with program visualisation. There are four stages in the model:

- **Preliminary Stage**: Instructors investigate the pre-existing mental models held by programming students and identify typical inappropriate models;
- **Cognitive Conflict Stage**: Trigger a discrepant event to explicitly challenge students’ pre-existing mental models and push students into cognitive conflict status;
- **Model Construction Stage**: Help students construct viable mental models by using visualisation;
- **Application Stage**: Students go on to solve a programming problem using the newly constructed mental model.

To evaluate the effectiveness of this learning model, an initial study was carried out by the authors [10]. The results revealed that the model was effective in enhancing a student’s interest in, and engagement with, the learning materials and helped them to construct viable mental models. However, the technique was found to be less effective with a more complex concept (namely, reference assignment), in situations where students appeared to lack the necessary base knowledge to interpret the visualisation.

### 3. The Learning Environment

To support instructors using the proposed learning model in their teaching and to give students easy access to the model, a web-based learning environment has been developed at the University of Strathclyde, UK. This learning environment currently supports Java, but
other languages may be supported in the future. The system logs each student’s progress and provides an easy way for students to monitor the viability of their mental models. When a student logs into the system, a ‘concept roadmap’ (Figure 1) is displayed showing the key programming concepts students have to understand. The order of learning the programming concepts is presented to the students using the routes, shown as footprints, on the concept roadmap. The previous study [9] suggested that a student would not be able to understand a concept if she or he could not properly understand the base knowledge that supports that concept, e.g. understanding value assignment is essential before approaching reference assignment. It is therefore suggested that programming concepts have to be learned in an appropriate order. For example, students should first learn the concept of scope before learning the concept of parameter passing. Without a proper understanding of scope, students will have difficulty understanding parameter passing properly. Alongside the button for each concept on the roadmap a red cross indicates that the student has not yet demonstrated an understanding of that concept. When the student appears to have constructed the appropriate mental model, by passing the test associated with the concept, the red cross will turn to a green tick symbol.

![Figure 1: The Concept Roadmap](image1)

![Figure 2: The Cognitive Conflict Question](image2)

When a student enters each concept, the student’s history of exercises with this concept will be shown, covering the date of performing the exercise and the apparent viability of their mental models before and after the exercise. For each concept, there are three exercises. Each exercise contains a question to trigger cognitive conflict, the corresponding visualisation materials, and another question students need to answer using their constructed mental models, to check their understanding. Initially, the cognitive conflict question asks students to mentally execute a program fragment and predict the result (figure 2). The students’ answers are then mapped to a collection of pre-defined mental models that were identified in a previous study [8]. If the student’s answer maps to an inappropriate mental model the student is informed that their prediction is incorrect and there may therefore be some problems with their understanding of the concept. The student is then asked to run a visualisation of the program fragment that they mentally executed to try to identify potential problems in their current understanding and to help them construct an appropriate mental model. The Jeliot visualisation tool1, developed by the University of Joensuu, is currently employed as the visualisation mechanism within the teaching environment. Jeliot dynamically animates Java program execution. After students experience

the visualisation, their new understanding is tested by a new program fragment that demonstrates the same concept, typically using different data values.

4. **EMPIRICAL STUDIES**

To investigate the effectiveness of the suggested learning model and also students’ reactions to it, a series of empirical studies have been carried out, covering the programming concepts of conditional, loop, scope, parameter, and reference. In this section, the experiment methods, results, and also a discussion of the results for each study are presented.

4.1 **The Study for Conditional and Loop**

4.1.1 **Experiment Method**

44 students who were in the introductory Java programming class in Computer Science at the University of Strathclyde took part in this study. The course was based around the BlueJ teaching environment[^2] and used the associated textbook “Objects First with Java - A Practical Introduction using BlueJ” [2]. The test was conducted in week 4 in the second semester of the course when the participants had already covered the conditional and loop concepts.

Interviews were carried out to investigate the effectiveness of the proposed learning model. An interviewer sat side by side with the participant who was asked to talk aloud as they answered questions on their reactions to each stage of the learning environment. Five interviewers were involved in this study - one post-doctoral research assistant and four academic staff from the University of Strathclyde (the authors of this report). All of the interviewers were intimately familiar with the proposed learning model and the learning environment.

There were three exercises conducted in this study. Each exercise can be divided into three stages. In the first stage, the participants were asked to mentally execute a program fragment and describe their understanding to the interviewers. Prior to being informed by the learning environment as to whether their predicted result of execution was correct, each participant would inform the interviewers how confident they felt that their answer was correct. The interviewers then investigated the participants’ feelings and reactions when they discovered whether their prediction was correct or incorrect. At the second stage, the participants went on to use the visualisation materials. They were then asked to explain their understanding of the visualisation to the interviewers. In the third stage, the participants were asked to mentally execute another program fragment and describe it to the interviewers.

The teaching experiences of the lecturers reveal that students often have problems when the conditional concept appears together with the loop concept. Therefore, this study covered both concepts. In the first exercise, students were asked to work out a program fragment with a conditional block inside a loop block (figure 3a). Students had to predict the value of the ‘count’ variable and explain the program execution. In the second exercise, the program fragment has a loop block inside another loop block (figure 3b). The program fragment in the

[^2]: http://www.bluej.org/
The third exercise also has a loop block inside another loop block but the increment variable in the outer loop can affect the number of iterations of the inner loop (Figure 3c). For the details of the learning materials for conditional and loop concepts, please refer to Appendix A.

```java
int count = 0;
for (int i=0; i<4; i++) {
    if (i<3) {
        count++;
    }
    count++;
}
```

```java
int count = 0;
for (int i=0; i<2; i++) {
    for (int j=0; j<2; j++) {
        count++;
    }
    count++;
}
```

```java
int count = 0;
for (int i=0; i<3; i++) {
    for (int j=0; j<i; j++) {
        count++;
    }
}
```

Figure 3: The program fragments used in the study for conditional and loop concepts

4.1.2 Results

10 out of 44 (23%) participants were found to be holding appropriate mental model of both conditional and loop. The pre-tests revealed that the remaining 34 students held one or more inappropriate mental models of conditional and/or loop concepts. A total of 54 inappropriate mental models were found to be held by the 34 students. They can be categorized into 10 groups - see Figure 4.

<table>
<thead>
<tr>
<th>Model</th>
<th>Description of the Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>Treat 'if' statement as 'if...else' statement. When tracing the first program fragment (Figure 3a), students with this model only incremented the count inside the 'if' statement but ignored the one outside the 'if' statement. So the result is 3.</td>
</tr>
<tr>
<td>M2</td>
<td>The scope of the loop block only covers the 'if' block. When tracing the first program fragment (Figure 3a), students with this model thought that the 'count++' inside the 'if' statement executed 3 times. After that, the outer one executed once. So the result is 4.</td>
</tr>
<tr>
<td>M3</td>
<td>Have no idea of how loop works, so cannot make prediction of program execution.</td>
</tr>
<tr>
<td>M4</td>
<td>The loop block executes in one shot rather than executed line by line.</td>
</tr>
<tr>
<td>M5</td>
<td>The increment variable increments before the loop is entered, e.g. 'i' would increment to 1 when the 'for (int i=0; i&lt;2; i++)' executed.</td>
</tr>
<tr>
<td>M6</td>
<td>The increment variable in the outer loop increments simultaneously with the increment variable in the inner loop. Students with this model thought a nested loop works as 'i=0, j=0 -&gt; i=1, j=1', rather than 'i=0, j=0 -&gt; i=0, j=1 -&gt; i=1, j=0 -&gt; i=1, j=1'.</td>
</tr>
<tr>
<td>M7</td>
<td>The increment variable in the inner loop, i.e. 'j', does not re-initialize to 0 but rather keeps its current value when the outer loop has another go.</td>
</tr>
<tr>
<td>M8</td>
<td>When the inner loop finishes, the execution flow does not go back to the outer loop.</td>
</tr>
<tr>
<td>M9</td>
<td>The inner loop and outer loop cancels each other, i.e. only one 'for' statement works.</td>
</tr>
<tr>
<td>M10</td>
<td>The nest loop executes in an order as 'i=0, j=0 -&gt; i=1, j=0 -&gt; i=0, j=1 -&gt; i=1, j=1'.</td>
</tr>
</tbody>
</table>

Figure 4: The inappropriate mental models list for conditional and loop concepts
Figure 5: The distribution of inappropriate mental models

The post-tests reveal that 22 out of 34 (65%) students changed all their inappropriate mental models to appropriate ones. One student did not finish the post-test, so it is impossible to know if his or her mental model has been improved. The remaining 11 students did not change all their inappropriate mental models to appropriate ones.

Among all of the 54 inappropriate mental models, 41 (76%) changed to appropriate ones. Apart from one mental model held by the student who did not finish the post-test, the remaining 12 (22%) remained unchanged or changed into another inappropriate mental model. Figure 6 shows the distribution of the improved and unimproved mental models.
After the students who failed the pre-tests experienced the cognitive conflict event, they were asked to express their thoughts or feelings. Figure 7 categorises the results\(^3\). For the first exercise, 3 students felt surprised, 14 students was interested to find out where they went wrong, and 11 students felt not surprised. For the second exercise, 4 students felt surprised, one student felt disappointed, 7 students was interested to know where they went wrong, and 6 students did not feel surprised. For the last exercise, 4 students felt surprised, 3 students felt disappointed, 6 felt interested to know where they went wrong and 2 felt not surprised.

<table>
<thead>
<tr>
<th>Exercise</th>
<th>Surprised</th>
<th>Disappointed</th>
<th>Interested</th>
<th>Unsurprised</th>
</tr>
</thead>
<tbody>
<tr>
<td>1(^{st})</td>
<td>3</td>
<td>0</td>
<td>14</td>
<td>11</td>
</tr>
<tr>
<td>2(^{nd})</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>3(^{rd})</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 7: The distribution of students’ reactions to cognitive conflict event

In addition, to test how confident students felt about their answers, they were also asked how much they believed in the result provided by the learning environment that claimed their answers were incorrect. The results show that no student who gave clear response to this question doubted the result provided by the learning environment. For the first exercise, 18 students believed the result provided by the learning environment is correct and their answer was wrong. 3 students had no idea whether or not their answer was wrong but claimed to have less confidence in their answers. For the second exercise, 14 students believed in the learning environment and rejected their own answers, and one student had no idea. For the third exercise, 8 students rejected their own answer, and one student did not know.

\(^3\) There were some students who experienced the cognitive conflict event but could not give a clear explanation of their feeling or thoughts, and some other students have expressed more than one kind of feeling or thought.
The comments from students show that most of them have positive attitudes toward the learning environment. One student even complained that they were not provided with the learning materials at the beginning when they started to learn the conditional and loop concepts. However, there were some students who did not like the visualisation. They felt it was difficult to follow the visualisation because there were too many graphics and animations moving on-screen. The observation from interviewers also shows that some students had to watch the visualisation several times before understanding it. Some other students needed the interviewers to provide a textual explanation of visualisation before understanding it. In addition, some students seemed to understand the visualisation in different ways. They claimed they understood the visualisation very well and seemed very confident with that. However, they still failed the post-test, and their understanding was still not accordant with what the visualisation presented. Furthermore, interviewers also found that some students explained the program code rather than visualisation when they were asked to describe what happen in the visualisation. Their explanation of the code was different from what actually happened in the visualisation. The visualisation appeared not to affect their understanding. The observation from interviewers also found that some students seemed not to want to watch the visualisation when they thought their understanding was appropriate. They viewed watching of the visualisation as extra work for them.

4.2 The Study for Scope and Parameter

4.2.1 Experiment Method
Our teaching experiences reveal that some misunderstandings of the parameter concept are often caused by misunderstandings of the scope concept. Therefore, scope and the study for parameter studies were conducted in same session. 27 students who were in the introductory Java programming class took part in these studies. They did not have to be the same students with those who participated in the previous study for the conditional and loop concepts. The study was conducted in week 6 in the second semester of the course when the participants had already covered the scope concepts.

A similar study method was used to the one used in the previous study described in section 4.1.1. An interviewer sat side by side with the participant to note how he or she went through the exercises. The same interviewers were involved in this study. Unlike the previous study with three exercises, there were two exercises conducted in the scope study and another two conducted in the parameter study. However, the procedure for the exercises is same.

In the first exercise of the scope study, students were given a class with two fields ‘a’ and ‘b’ (figure 8a). In the ‘assignment’ method, there is a local variable ‘a’ with its value being set as the value of field ‘b’. In the second exercise (figure 8b), the ‘assignment’ method also has a local variable ‘a’ with initial value of 30. The field ‘b’ is set to be the value of local variable ‘a’ plus 5. In both exercises, students were asked to mentally trace the program fragment shown in figure 9.
public class Example{
    int a;
    int b;
    public Example(){
        a = 10;
        b = 20;
    }
    public void assignment(){
        int a;
        a=b;
    }
    public void show(){
        System.out.println("a is: "+a);
    }
}

public class Example{
    int a;
    int b;
    public Example(){
        a = 10;
        b = 20;
    }
    public void assignment(){
        int a;
        a=30;
        b=a+5;
    }
    public void show(){
        System.out.println("a is: "+a);
        System.out.println("b is: "+b);
    }
}

Figure 8a
Figure 8b

Figure 8: The class used in the study for scope concept

Example example=new Example();
example.assignment();
example.show();

Figure 9: Program fragment used in the study for scope concept

Figure 10a shows the program fragment used in the first exercise while figure 10b shows the one used in the second exercise.
public class Example {
    public Example() {
    }
    public int sum (int a, int b) {
        a = a + b;
        return a;
    }
    public static void main() {
        int a;
        int b;
        int result;
        a = 10;
        b = 20;
        Example example = new Example();
        result = example.sum(a, b);
        System.out.println("a is "+a);
        System.out.println("b is "+b);
    }
}

public class Example {
    public Example() {
    }
    public int sum (int result, int a, int b) {
        result = a + b;
        return result;
    }
    public static void main() {
        int a;
        int b;
        int result;
        a = 10;
        b = 20;
        Example example = new Example();
        result = 0;
        Example example = new Example();
        result = example.sum(a, b, result);
        System.out.println("result is "+result);
    }
}

Figure 10a          Figure 10b

Figure 10: The classes used in the study for scope concept

4.2.2 Results
The results of the pre-test show that only 5 out of 27 (19%) students held an appropriate mental model of scope. Apart from two students who held inappropriate mental models of variable and assignment, 20 out of 27 (74%) participants held a model that a local variable has same scope as a field (M1 in figure 11).

Apart from one student who did not finish the parameter study due to the lack of time, 9 of the remaining 26 (35%) students held an appropriate mental model of parameter passing. The remaining 17 students were found to hold one or more inappropriate mental models. 15 students held an inappropriate mental model (M3 in figure 11) that is associated with the scope concept. They thought that the scope of a formal parameter was actually the whole class rather than only the method. In addition, 9 students did not think the mapping between actual parameter and formal parameter has to be in order (M4 in figure 11). [Maybe this can be used to support the hypothesis that students' previous knowledge (e.g. language in this case) affects students learning of new knowledge].
<table>
<thead>
<tr>
<th>Model</th>
<th>Description of the Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>Treat a local variable as if it has the same scope as a field. When tracing the first scope program fragment (figure 8a), students with this model thought the local variable 'a' can be accessed outside of the 'assignment' method.</td>
</tr>
<tr>
<td>M2</td>
<td>Treat a field as a local variable. When tracing the second scope program fragment (figure 8b), students with this model treated the field 'b' as a local variable in the 'assignment' method.</td>
</tr>
<tr>
<td>M3</td>
<td>The scope of a formal parameter is the whole class rather than only the method. This model is associated with M1. When tracing the first parameter program fragment (figure 10a), students with this model thought the formal parameter 'a' can be accessed outside of the 'sum' method.</td>
</tr>
<tr>
<td>M4</td>
<td>The mapping between actual parameter and formal parameter does not have to be in order. When tracing the second parameter program fragment (figure 10b), students with this model thought the mapping between actual parameter and formal parameter was based on the value of names.</td>
</tr>
</tbody>
</table>

**Figure 11: the inappropriate mental models list for Scope and Parameter concepts**

![Pie charts showing the distribution of appropriate and inappropriate models before and after intervention.](chart.png)
<table>
<thead>
<tr>
<th>Group No.</th>
<th>Item No.</th>
<th>Scope Study</th>
<th>Parameter Study</th>
<th>No. of Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>√</td>
<td>√</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>√</td>
<td>√</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>M1 -&gt; √</td>
<td>√</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>M1 -&gt; √</td>
<td>Unfinished</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>(accidentally)</td>
<td>√</td>
<td>M1 -&gt; √</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>M1 -&gt; √</td>
<td>M3 -&gt; √</td>
<td>M4 -&gt; √</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>M1 -&gt; √</td>
<td>M3 -&gt; √</td>
<td>M2 -&gt; √</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>√</td>
<td>M1 -&gt; √</td>
<td>M3 -&gt; √</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>Other -&gt; √</td>
<td>M3 -&gt; √</td>
<td>M2 -&gt; √</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>Other -&gt; √</td>
<td>M3 -&gt; √</td>
<td>M2 -&gt; X</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>M1 -&gt; M1</td>
<td>M3 -&gt; √</td>
<td>M4 -&gt; √</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>M1 -&gt; √</td>
<td>M2 -&gt; √</td>
<td>M3 -&gt; √</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>M1 -&gt; √</td>
<td>M2 -&gt; √</td>
<td>M3 -&gt; √</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>M1 -&gt; √</td>
<td>M2 -&gt; √</td>
<td>M3 -&gt; Lost</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>M1 -&gt; M1</td>
<td>M2 -&gt; √</td>
<td>M3 -&gt; √</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>M1 -&gt; M1</td>
<td>M2 -&gt; √</td>
<td>M3 -&gt; √</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 12: The distribution of students with inappropriate models of scope and parameter

Figure 12 shows how students’ mental models changed. 17 out of 20 (85%) students who held the M1 model have changed their mental models into an appropriate model after experiencing the learning materials. 14 out of 15 (93%) students who held the M3 model have constructed an appropriate one. In addition, 7 out of 9 (78%) students have changed their mental models from model M4 to an appropriate one.

In group 1 of figure 12, all the 5 students had properly understood the scope of local variables and formal parameters, and consistently used the appropriate model throughout all exercises.

In group 2, the students held an inappropriate model of scope for local variables, i.e. M1 and successfully changed to an appropriate one after the exercise. When they went to do parameter exercises, they employed an appropriate mental model of scope for formal parameters. 7 students were categorised into this group. One of them did not finish the parameter study.

In group 3, all the 10 students have constructed an appropriate mental model of scope for local variables from an inappropriate one. However, they still used inappropriate mental model of scope for formal parameters in the parameter study.
In group 4, all the students showed model M1 in the first exercise, and no student held model M2 (i.e. treat a field as a local variable) which is the opposite of M1. However, all of them employed model M2 in the second exercise and all of them used an inappropriate model of scope for the formal parameter [evidence for a ‘surface’ change of mental model?]. In addition, the same pattern has been found in the parameter study. There were 6 students who held model M3 and successfully constructed an appropriate one in the first exercise in the parameter study. However, they have been found to be using M2 in the second exercise.

In these studies, students were also asked to express their thoughts or feelings after they failed the pre-tests and experienced the cognitive conflict event. Figure 13 categorises the results.

<table>
<thead>
<tr>
<th>Exercise</th>
<th>Shocked</th>
<th>Surprised</th>
<th>Disappointed</th>
<th>Interested</th>
<th>Unsurprised</th>
<th>Do not care</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope 1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Scope 2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Parameter 1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Parameter 2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 13: The distribution of students’ reactions to cognitive conflict event**

These two studies also investigated whether or not the students’ motivations to see the visualisation had changed after experiencing the cognitive conflict event. Before they were told whether their prediction was correct or not, they were asked to rank how keen they wanted to see the visualisation, from a range between 1 and 10. After knowing the results, they were asked to do the ranking again. The results (Figure 14) reveal that all the students’ motivations to see the visualisation had been improved or kept same. In addition, some students who were told their prediction was correct also gave ranking even though they did not have to do that. The results show that all these students’ motivation to see the visualisation decreased.

<table>
<thead>
<tr>
<th>Exercise</th>
<th>Experiencing Cognitive Conflict</th>
<th>Understanding was Appropriate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Increased</td>
<td>Same</td>
</tr>
<tr>
<td>Scope 1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>Scope 2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Parameter 1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Parameter 2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

**Figure 14: The distribution of students’ reactions to cognitive conflict event**

---

<sup>4</sup> There were some students who experienced the cognitive conflict event but could not give a clear explanation of their feeling or thoughts, and some other students expressed more than one kind of feeling or thoughts.
4.3 Object Reference Study

4.3.1 Experiment Method

The test was conducted in the last week of the course when students were busy preparing for their exams; hence only 11 students took part in this study. There was some overlap with this group of students and those who participated in the previous studies.

A similar study method was used to the one used in the previous studies. An interviewer sat side by side with the participant to note how he or she went through the exercises. The same interviewers were involved in this study. Unlike the previous studies which had two or three exercises, there was only one exercise conducted in this study. However, this exercise asked students to use two types of visualisation and take two post-tests. After the pre-test, students who passed the test and those who failed the test were randomly and equally separated into two groups: the Jeliot-first group in which students first experienced the Jeliot visualisation and then the home-made visualisation; the Home made-first group in which students first experienced the home-made visualisation and the Jeliot visualisation. Figure 15 shows the workflow of the exercise.

![Flowchart for the study of object reference](image)

Figure 15: The flowchart for the study of object reference

Figure 16 show the interface of the home-made visualisation materials. Compared to the Jeliot visualisation, this visualisation is concept-focused, i.e. it only shows the graphical elements and animations that are necessary for representing reference assignment. This design is expected to minimize the distraction from the unnecessary complications of the
visualisation. In addition, as Stasko et al. (1993) have suggested, animation might be more beneficial under two conditions. Firstly, it should be used in coordination with an additional explanation of the animation. Therefore, this tool provides students with textual explanation of the animation for each step of the program execution (see bottom pane of the window in figure 16). In addition, Stasko et al. (1993) suggested that animation systems need to have ‘rewind’ or ‘replay’ functions. This is provided by the left pointing arrow button in the animation tool (figure 16).

![Figure 16: The interface of the home-made visualisation](image)

Figure 16 and Figure 17 show the class and program fragment used in pre-test and post-test.
public class Student {
    String name;

    public Student(String n){
        name=n;
    }

    public int getName(){
        return name;
    }

    public void changeName(String newName){
        name=newName;
    }
}

public class Account {
    double balance;

    public Account(double initialBalance){
        balance = initialBalance;
    }

    public double getBalance(){
        return balance;
    }

    public void changeBalance(double newBalance){
        balance = newBalance;
    }
}

public class Car {
    int price;

    public Car(int p){
        price = p;
    }

    public int getPrice(){
        return price;
    }

    public void increasePrice(int increment){
        price=price+increment;
    }
}

Figure 17a

Figure 17b

Figure 17c

Figure 17: The classes used in the study of Object Reference

Student a;
Student b;
a = new Student("Ben");
b = new Student("Lucy");
a = b;
b.changeName("Tom")

Account a;
Account b;
a = new Account(15000);
b = new Account(20000);
a = b;
a.changeBalance(25000);

Car a;
Car b;
a = new Car(15000);
b = new Car(20000);
b = a;
a. increasePrice(2000);

Figure 18a

Figure 18b

Figure 18c

Figure 18: Program fragments used in the study of Object Reference

4.3.2 Results

The results of pre-test show only 1 out of 11 (9%) students passed this test. The remaining 10 students were found to be holding one or more inappropriate mental models of class, object, assignment or reference concepts. Two students were totally lost with the class and object concept (they could not give any explanation of the statements covering reference variable declaration and object creation) (M1). One student viewed assignment as from left hand side to the right hand side (M2). One student believed an assignment statement cannot work but rather would crash (M3). Another student thought a program would be executed in one shot rather than executed one statement at a time in order (M4). 5 students viewed the
declaration of a reference variable, such as `Staff a`, as the actual creation of an object, believing the statements to create an object, such as `a = new Staff(1)`, are used to initialize the object (M5). 8 students viewed an object as stored at a reference variable rather than referred by the reference variable (M6).

<table>
<thead>
<tr>
<th>Model</th>
<th>Description of the Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>Totally lost with class and object concept</td>
</tr>
<tr>
<td>M2</td>
<td>Viewed assignment as from left hand side to right hand side</td>
</tr>
<tr>
<td>M3</td>
<td>An assignment statement cannot work but rather would crash</td>
</tr>
<tr>
<td>M4</td>
<td>A program would be executed in one shot rather than executed one statement by one statement in order</td>
</tr>
<tr>
<td>M5</td>
<td>Viewed the declaration of a reference variable as the actual creation of an object</td>
</tr>
<tr>
<td>M6</td>
<td>An object was stored at a reference variable rather than referred to by the reference variable</td>
</tr>
</tbody>
</table>

**Figure 19: The inappropriate mental models identified in the Object Reference study**

After students experienced the first visualisation, no matter which version they used, all the students with the M2, M3 and M4 model changed their model to an appropriate one. The two students who were totally lost with the class and object concept (M1) also improved their understandings even though the understandings were still not appropriate. One of them claimed that “Ben is an object within Student class” while another one explained the statement of `a = new Staff(1)` as “create a new class cell, and place the new class into the cell”.

There was only one student who held the M5 model who constructed an appropriate one after using visualisation. This student was in the home made-first group and changed his or her model using the home-made visualisation. All the other four students kept their inappropriate models even though the home-made visualisation gave a textual description of object declaration and object creation along with the visualisation.

6 out of 8 students who held the M6 model changed their model to an appropriate one. All the 6 students constructed the appropriate model using the home-made one. 3 of them were from the home made-first group. They changed their models after using the first visualisation, i.e. the home-made one. The remaining 3 students were from the Jeliot-first group. They failed to construct an appropriate model using Jeliot but succeeded with the home-made one.

The observation from interviewers reveals that some students were using their existing mental models to interpret the visualisation rather than using the visualisation to improve their mental models. These students always claimed they had understood the visualisation but still explained the visualisation based on their original, inappropriate model. In addition, some students claimed that the textual explanation was important for understanding the visualisation. However, the observation from interviewers found that some students did not
engage with the textual explanation in the home-made visualisation. One student explained that he preferred to watch pictures and would only read the textual explanation when he did not understand the picture. Unfortunately, he was using his inappropriate model to interpret the visualisation but believed there was nothing wrong with his understanding of the visualisation.

At end of the exercise, students were asked to comment on the Jeliot visualisation and the home-made visualisation. The following lists the recognized advantages/disadvantages of both visualisations:

**Jeliot**

Advantages:
- A relatively professional tool with more functions
- Can support more concepts
- Can support custom code
- Can control the speed of the visualisation

Disadvantages:
- The visualisation is complicated. There are too many graphical elements and animations.
- Need verbal explanation

**Home-made**

Advantages:
- Visualisation is simple. Easy to follow
- Focuses on key parts
- Gives textual explanation to the visualisation

Disadvantage:
- Cannot support custom code

### 4.4 Discussion

The pre-tests identified a collection of non-viable mental models of conditional, loop, scope, parameter and reference concepts held by first year programming students. A surprisingly large percentage of students have been found to hold one or more non-viable mental models of these basic programming concepts after they had learned these concepts using traditional learning approaches and materials. The quantitative analysis revealed that⁵: 77% students held non-viable mental models of conditional and/or loop concepts; 81% students held non-viable mental models of the scope concept; 65% students held non-viable mental models of

---

⁵ Due to the limited number of participants, the quantitative data from object reference is not enough for a statistical analysis.
the parameter concept\textsuperscript{6}. The high rates of students with non-viable mental models of these very basic programming concepts reveal that the traditional teaching approaches and materials are not as effective as expected.

The results of the post-test reveal that a large percentage of students had improved their mental models after experiencing the learning materials based on the proposed learning model. This suggests that the proposed learning model which integrates cognitive conflict strategy and program visualisation may be more effective compared to the traditional learning models.

The investigation of students’ reactions to the cognitive conflict event found that explicitly exposing students’ problems with their mental models focuses their attention on the learning materials and motivates them to engage with the visualisation. The level of most students’ interest in viewing the visualisation improved when they knew their understanding of programming concepts was incorrect. This investigation also found that most students did not realise their understandings of the concepts were inappropriate before confronting the cognitive conflict event. According to cognitive conflict theory, students cannot construct an appropriate understanding when they are happy with their existing inappropriate understanding. That may explain why the traditional learning approaches and materials are less effective. On the other hand, the cognitive conflict-based practices break students’ cognitive balance and drive students to actively seek re-balance.

The visualisation was effective to help students construct appropriate mental models. However, some students complained the Jeliot visualisation was too complicated. It was hard to follow the animations because there was too much ‘stuff’ moving on-screen. The home-made visualisation with simplified graphics and animation seemed to perform better. However, the home-made visualisation seems impractical in a ‘real’ learning environment. It may take too much effort for instructors to design home-made visualisations for each concept. In addition, the textual explanation is important for students to understand the visualisation. It raises a question of how to use visualisation-based materials in programming education. For example, should visualisation be used as self-learning materials or should it used as a tool for lecturers to explain programming concepts? Also, some students were ‘misunderstanding’ the visualisation materials. The visualisation materials were expected to change students’ mental model to an appropriate one. However, some students used their inappropriate mental model to interpret the visualisation, and unfortunately they believed their understanding was appropriate.

Based on our teaching experiences, the main misunderstanding of the parameter concept is that the formal parameter is visible for the whole class. This misunderstanding seems related to the misunderstandings of the scope concept. In this case, the scope study and the parameter study have been arranged in same session, and the scope study was conducted before the parameter study. The results reveal that students who constructed viable mental models of the scope concept had no misunderstanding that the formal parameter is visible for the whole class. This may imply that a mental model could be constructed based on one or more other mental models. The construction of mental models has to be in order.

\textsuperscript{6} When the Parameter study was conducted, the Scope study has already help the students construct viable mental models of Scope concept.
5. CONCLUSION

Ongoing research has shown that many students still have non-viable mental models of key programming concepts at the end of introductory programming courses [9, 10]. This research confirms this finding, extending it to the key concepts of conditionals, loops, scope and parameter passing, and identifies a wide range of non-viable models of these concepts held by students.

To help address this problem a constructivist-based learning model that integrates cognitive conflict and program visualization has been suggested and demonstrated [10]. This research confirms the potential value of this model, showing significant (possibly short-term) improvements in students’ understanding of these concepts. The research has provided some evidence that cognitive conflict can help engage the student with the visualisation.

Finally, the key new finding is that Jeliot is a practical and cost effective visualisation component of the proposed model for many concepts including loops, conditionals, scope and parameter passing. Jeliot visualisations provide simple, effective explanations of these concepts along with additional functionality such as editing program code values and structure, and controlling animation speed and its starting point. However, for the more complex concept of object reference assignment the Jeliot animation appeared too complicated for our students to help them correct their misunderstandings.

6. REFERENCES


7. APPENDIX

7.1 Appendix A: Learning Materials for Conditional and Loop Concepts

Exercise 1 – Pre-test:
Please predict what would be printed on screen when the following statements are executed.

```java
public class Example {
    public static void main() {
        int count = 0;
        for (int i=0; i<4; i++){
            if (i<3){
                count++;
            }
            count++;
        }
        System.out.println("count is "+count);
    }
}
```

Please fill in your answer here:
The result of 'count' is = ___________

Exercise 1 – Post-test:
Please predict what would be printed on screen when the following statements are executed.

```java
public class Example {
    public static void main() {
        int count = 0;
        for (int i=0; i<4; i++){
            if (i>=1 && i<3){
                count++;
            }
            count++;
        }
        System.out.println("count is "+count);
    }
}
```

Please fill in your answer here:
The result of 'count' is = ___________
Exercise 2 – Pre-test:
Please predict what would be printed on screen when the following statements are executed.

```java
public class Example {
    public static void main() {
        int count = 0;
        for (int i=0; i<2; i++){
            for (int j=0; j<2; j++){
                count++;
            }
        } 
        System.out.println("count is "+count);
    }
}
```

Please fill in your answer here:

The result of 'count' is = ___________

Exercise 2 – Post-test:
Please predict what would be printed on screen when the following statements are executed.

```java
public class Example {
    public static void main() {
        int count = 0;
        for (int i=1; i<3; i++){
            for (int j=5; j<8; j++){
                count++;
            }
        } 
        System.out.println("count is "+count);
    }
}
```

Please fill in your answer here:

The result of 'count' is = ___________
Exercise 3 – Pre-test:
Please predict what would be printed on screen when the following statements are executed.

public class Example {

    public static void main() {

        int count = 0;
        for (int i=0; i<3; i++){
            for (int j=0; j<i; j++){
                count++;
            }
        }

        System.out.println("count is "+count);
    }
}

Please fill in your answer here:
The result of 'count' is = ___________

Exercise 3 – Post-test:
Please predict what would be printed on screen when the following statements are executed.

public class Example {

    public static void main() {

        int count = 0;
        for (int i=1; i<3; i++){
            for (int j=1; j<i; j++){
                count++;
            }
        }

        System.out.println("count is "+count);
    }
}

Please fill in your answer here:
The result of 'count' is = ___________
7.2 Appendix B: Learning Materials for Scope Concept

Exercise 1 – Pre-test:

An *Example* class has been defined

```java
public class Example{
    int a;
    int b;
	public Example(){
		a = 10;
		b = 20;
	}
	public void assignment(){
		int a;
		a=b;
	}
	public void show(){
		System.out.println("a is: \"+a);
	}
}
```

*Please predict what would be printed on screen when the following statements are executed.*

Example example=new Example();
example.assignment();
example.show();

*Please fill in you answer here:*

‘a’ is = ___________
Exercise 1 – Post-test:

An Example class has been defined

```java
public class Example{

    int a;
    int b;

    public Example(){
        a = 10;
        b = 20;
    }

    public void assignment(){
        int a;
        a=30;
    }

    public void show(){
        System.out.println("a is: "+a);
    }
}
```

Please predict what would be printed on screen when the following statements are executed.

Example example=new Example();
example.assignment();
example.show();

Please fill in you answer here:

‘a’ is = __________
Exercise 2 – Pre-test:

An Example class has been defined

```java
public class Example{

    int a;
    int b;

    public Example(){
        a = 10;
        b = 20;
    }

    public void assignment(){
        int a;
        a=30;
        b=a+5;
    }

    public void show(){
        System.out.println("a is: "+a);
        System.out.println("b is: "+b);
    }
}
```

Please predict what would be printed on screen when the following statements are executed.

Example example=new Example();
example.assignment();
example.show();

Please fill in you answer here:

’a’ is = ___________                      ‘b’ is = ___________
Exercise 2 – Post-test:

An Example class has been defined

```java
public class Example{
    int a;
    int b;

    public Example(){
        a = 10;
        b = 20;
    }

    public void assignment(){
        int b;
        b=30;
        a=b-5;
    }

    public void show(){
        System.out.print("a is: "+a);
        System.out.print("b is: "+b);
    }
}
```

Please predict what would be printed on screen when the following statements are executed.

Example example=new Example();
example.assignment();
example.show();

Please fill in you answer here:

‘a’ is = ___________                      ‘b’ is = ___________
7.3 Appendix C: Learning Materials for the Parameter Concept

Exercise 1 – Pre-test:

An Example class has been defined. Please predict what would be printed on screen when the following statements are executed.

```java
public class Example {
    // Default constructor used to initialise Example objects.
    public Example() {
    }

    public int sum(int a, int b) {
        a = a + b;
        return a;
    }

    public static void main() {
        int a;
        int b;
        int result;

        a = 10;
        b = 20;
        Example example = new Example();
        result = example.sum(a, b);

        System.out.println("a is "+a);
        System.out.println("b is "+b);
    }
}
```

Please fill in your answer here:

‘a’ is = ___________                      ‘b’ is = ___________
Exercise 1 – Post-test:

An `Example` class has been defined. Please predict what would be printed on screen when the following statements are executed.

```java
import jeliot.io.*;

public class Example {

    // Default constructor used to initialise Example objects.
    public Example() {

    }

    public int substract (int a, int b) {
        a = a - b;
        return a;
    }

    public static void main() {
        int a;
        int b;
        int result;

        a = 10;
        b = 5;
        Example example = new Example();
        result = example.substract(a, b);

        System.out.print("a is =\" + a);
        System.out.print("b is =\" + b);

    }
}
```

Please fill in your answer here:

‘a’ is = ___________                      ‘b’ is = ___________
Exercise 2 – Pre-test:

An Example class has been defined. Please predict what would be printed on screen when the following statements are executed.

```java
public class Example {

    // Default constructor used to initialise Example objects.
    public Example() {
    }

    public int sum(int result, int a, int b) {
        result = a + b;
        return result;
    }

    public static void main() {
        int a;
        int b;
        int result;

        a = 10;
        b = 20;
        result = 0;
        Example example = new Example();
        result = example.sum(a, b, result);

        System.out.println("result is "+result);
    }
}

Please fill in you answer here:

’a’ is = ___________  ‘b’ is = ___________
Exercise 2 – Post-test:

An Example class has been defined. Please predict what would be printed on screen when the following statements are executed.

```java
import jeliot.io.*;

public class Example {

    //Default constructor used to initialise Example objects.
    public Example(){
    }
    
    public int subtract(int result, int a, int b){
        result = a - b;
        return result;
    }
    
    public static void main() {
        int a;
        int b;
        int result;
        a = 10;
        b = 15;
        result = 0;
        Example example = new Example();
        result = example.subtract(a, b, result);
        System.out.println("result is ="+result);
    }
}
```

Please fill in you answer here:

‘a’ is = _________                      ‘b’ is = _________
7.4 Appendix C: Learning Materials for the Object Reference Concept

Pre-test:

A Student class has been defined

```java
public class Student {
    String name;

    public Student(String n){
        name=n;
    }

    public int getName(){
        return name;
    }

    public void changeName(String newName){
        name=newName;
    }
}
```

Please predict the results after the statements have been executed.

```java
Student a;
Student b;
a = new Student("Ben");
b = new Student("Lucy");
a = b;
b.changeName("Tom")
```

Please fill in your answer here:
```
a.getName() = ________
b.getName() = ________
```
Post-Test1

An Account class has been defined

```java
public class Account {
    double balance;

    public Account(double initialBalance) {
        balance = initialBalance;
    }

    public double getBalance() {
        return balance;
    }

    public void changeBalance(double newBalance) {
        balance = newBalance;
    }
}
```

Please predict the results after the statements have been executed.

```java
Account a;
Account b;
a = new Account(15000);
b = new Account(20000);
a = b;
a.changeBalance(25000);
```

Please fill in your answer here:

a.getBalance() = __________  b.getBalance() = __________
A `Car` class has been defined

```java
public class Car {
    int price;

    public Car(int p){
        price = p;
    }

    public int getPrice(){
        return price;
    }

    public void increasePrice(int increment){
        price = price + increment;
    }
}
```

Please predict the results after the statements have been executed.

```java
Car a;
Car b;
a = new Car(15000);
b = new Car(20000);
b = a;
b. increasePrice(2000);
```

Please fill in your answer here:

```
a.getPrice() = ____________ b.getPrice() = ____________
```