Solve & Evaluate with Informa: A Java-based Classroom Response System for Teaching Java

Matthias Hauswirth, Andrea Adamoli
Faculty of Informatics, University of Lugano
Lugano, Switzerland
Matthias.Hauswirth@unisi.ch, Andrea.Adamoli@lu.unisi.ch

ABSTRACT

This paper describes the use of clickers in a Java programming course. However, instead of using ordinary hardware clickers, we use software clickers, implemented in Java, that allow for much richer problem types than the traditional multiple-choice question. The problem types we introduce in this paper give students a much higher degree of freedom in solving a problem, and thus more opportunities for making mistakes. We look at mistakes as learning opportunities, and we introduce a pedagogical approach that allows students to learn from mistakes of their peers. We finish with a case study and evaluation of our implementation of these ideas in an undergraduate Java programming course.

Categories and Subject Descriptors: K.3.2 [Computers and Education]: Computer and Information Science Education – *Computer science education*

General Terms: Design, Languages.

Keywords: Collaborative learning. Classroom clickers.

1. INTRODUCTION

Group response systems, also called classroom clickers, are technological interventions to improve teaching. A clicker is a small device, similar to a remote control, that allows a student to electronically submit his answer to a multiple choice question. An instructor can use clickers during ordinary lectures to quickly gain insight into the level of understanding of her students.

In this paper we describe the use of clickers for teaching Java programming. However, instead of using ordinary hardware clickers, we use software clickers, implemented in Java, that allow for much richer problem types than the traditional multiple-choice question. The problem types we introduce in this paper give students a much higher degree of freedom in solving a problem, and thus more opportunities for making mistakes. We look at mistakes as learning opportunities, and we introduce a pedagogical approach that allows students to learn from mistakes of their peers.

This paper makes the following contributions: It introduces (1) a set of clicker problem types to support teaching programming; (2) a new pedagogical approach (Solve & Evaluate) to allow students to learn from the mistakes of their peers, and to productively use the waiting times in clicker-based teaching; and (3) a case study and evaluation of our implementation of these ideas in an undergraduate Java programming course.

The remainder of this paper is structured as follows: Section 2 provides the background on the Informa software clicker framework necessary for understanding this paper. Section 3 introduces four programming skills and the clicker problem types we devised to support teaching them. Section 4 presents the Solve & Evaluate pedagogical approach. Section 5 describes our implementation. Section 6 presents a case study using our system, and Section 7 discusses the results. Section 8 surveys related work, and Section 9 concludes.

2. BACKGROUND

Informa [7] is an extensible framework for group response systems. Such a system can quickly gather feedback from all students in a classroom. The normal use of such a system is for an instructor, during a lecture, to project a multiple-choice question on the classroom projector, and for students to use clickers, special purpose remote controls, to submit their answers, by pressing one of the buttons on their clicker. The instructor can immediately see the responses aggregated in the form of a histogram, and can adopt her lecture to the level of understanding of the students.

Informa is a software implementation of a clicker system. Informa is written in Java, which allows Informa plugins to use the extensive class libraries available for that language. Informa does not require special hardware, but it requires Java-enabled computers for the instructor and all students. Currently this limits Informa's applicability to institutions where students bring laptops to class, but in principle the clicker software could run on any Java-enabled device, such as a cell phone.

The main idea behind Informa is that a software-based clicker is able to support much richer interactions than traditional hardware clickers. In Informa, multiple-choice questions are just one of an extensible number of problem types. For example, Informa provides a plugin supporting so-called text-highlighting problems, where students are presented with a question and a body of text, and then have to highlight all the relevant sections in the text. This type of problem is richer than a multiple-choice question, because students have to "construct" the solution (an arbitrary set of highlights) themselves, instead of picking among a set of solutions predetermined by the instructor.

Multiple-choice problems have the advantage that their solutions can easily be summarized in a histogram. More complex questions require different forms of aggregation and visualization. Informa presents the list of all individual solutions to the instructor. To scale to larger numbers of students, plugins for new problem types can provide aggregate visualizations of solutions. So far we have implemented such aggregations for multiple-choice problems (using histograms) and for text-highlighting problems (by overlaying semi-transparent highlights of all submitted solutions).

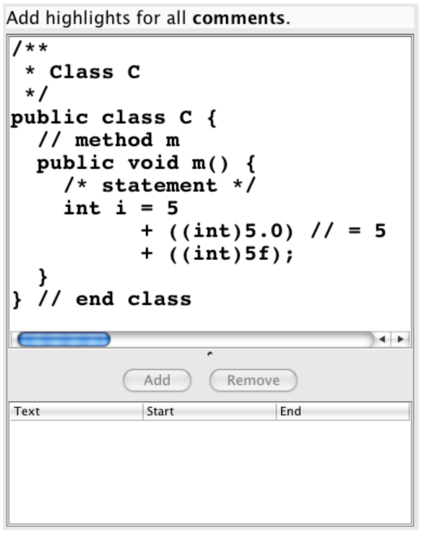
3. PROGRAMMING SKILLS

Learning to program in a language like Java means learning a number of interrelated skills. In this section we describe four such skills, and show how we extended Informa to add support for teaching them. We picked these four skills driven by the needs in our own classroom and the significant differences between them. A student learning to program needs to acquire many other skills, and Informa allows adding support for them in the future.

3.1 Syntax

Syntactic constructs and their names are examples of abstraction, a fundamental skill in computer science. The ability to talk about syntactic constructs, naming the concepts, and recognizing them, is one of the first examples of abstraction that students see, and it is a good first step in honing their abstraction skills.

The text highlighting problem type, which is already available in Informa [7], provides a straightforward way for checking the students' understanding of the language syntax. Figure 1 shows a text highlighting question where the text corresponds to a piece of Java source code, and the question corresponds to a request to highlight a specific syntactic construct. For example, the instructor could ask students to highlight all comments (as in the figure), all String literals, all method bodies,


Add highlights for all **comments**.

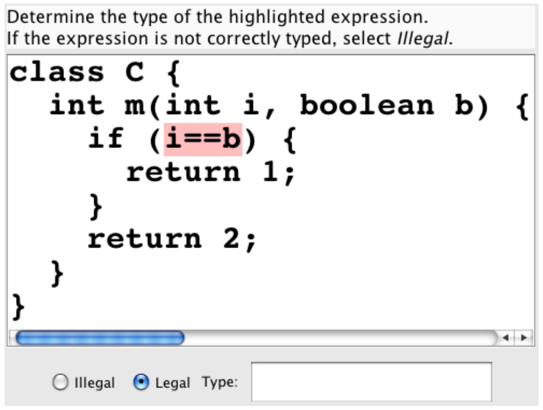
```
/**
 * Class C
 */
public class C {
  // method m
  public void m() {
    /* statement */
    int i = 5
          + ((int)5.0) // = 5
          + ((int)5f);
  }
} // end class
```

Figure 1: Syntax Problem

all conditions in conditional statements, or the dots corresponding to method invocations. Further questions, not necessarily limited to language syntax, include the highlighting of all read accesses to fields, all uses of final variables, or all interface method invocations.

3.2 Types

The type system is an integral part of a language like Java. An important skill for a Java programmer is to be able to infer the type of an arbitrary expression.


Determine the type of the highlighted expression.
If the expression is not correctly typed, select *Illegal*.

```
class C {
  int m(int i, boolean b) {
    if (i==b) {
      return 1;
    }
    return 2;
  }
}
```

Illegal [✓ Legal] Type:

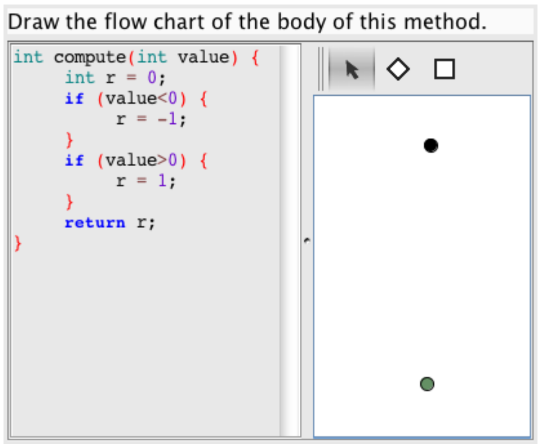
Figure 2: Typing Problem

We have developed a new problem type for Informa that allows us to practice and test that skill. Figure 2 shows such a typing problem. The instructor writes a snippet of Java code and highlights an expression. The student then needs to decide whether the highlighted expression is legal, and if it is legal, he needs to determine its type.

3.3 Control Flow

One of the challenges for beginning programmers is to understand the flow of control. In languages based on the principle of structured programming [3], students need to understand the fundamental concepts of sequence, selection, and iteration. Languages like Java provide if and switch statements for selection and for, while, and do-while loops for iteration.

A useful means to explaining a concept is the use of multiple representations. In the case of control structures, we use two such representations: the source code involving the control structures, and a lower-level representation, control flow graphs (flow charts), as a graphical way to explain the semantics of the control structures. To assess whether students understand a specific control structure, we ask them to translate source code containing that structure into a control flow graph.


Draw the flow chart of the body of this method.

```
int compute(int value) {
      int r = 0;
      if (value<0) {
            r = -1;
      }
      if (value>0) {
            r = 1;
      }
      return r;
}
```

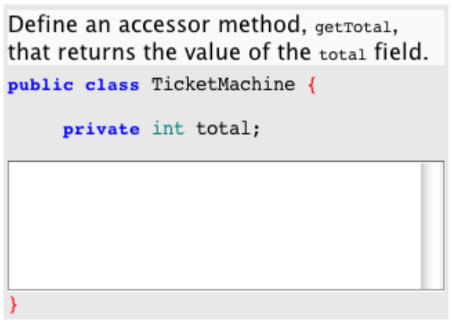
Figure 3: Control Flow Graph Problem

Figure 3 shows how we extended Informa to present such a problem on a students' clicker. We show the Java source code side-by-side with a control flow graph. Initially the graph is incomplete (it only contains the entry and exit nodes). The student has to complete the graph by adding the necessary nodes and edges.

3.4 Coding

The previous three skills, understanding the language's syntax, type system, and control structures, are basic components necessary for being able to program. The ultimate goal of teaching programming, however, is to enable students to translate a requirements specification into an implementation.

The direct way to assess these skills is to ask the students to perform exactly this task. Figure 4 shows how we do this with a new Informa problem type. The instructor writes the requirements and provides a skeleton of the code. The student solves the problem by completing the code.


Define an accessor method, getTotal, that returns the value of the total field.

```
public class TicketMachine {

    private int total;

}
```

Figure 4: Coding Problem

4. SOLVE & EVALUATE APPROACH

The pedagogical script of the original Informa system [7] was straightforward. During a lecture, an instructor may pose several Informa problems. Each problem constitutes one step. A step consists of three phases: *solve*, *discuss*, and *reveal*. In the *solve* phase, the students individually solve the problem on their clicker and submit their solutions. In the *discuss* phase, the instructor shows an aggregate visualization of the submitted solutions and leads a discussion of the problem and the different solution approaches. In the *reveal* phase, the instructor finally reveals the correct solution (or one of the correct solutions).

The key problem in this approach was the uneven time it took students to solve a problem. While some students often finished within a minute, others took an order of magnitude longer. Because the fast students had to wait for the slower ones, they shifted their attention from the course topic to unrelated activities.

Given that some of our new types of problems can take a significant amount of time to solve, introducing these problem types required a pedagogical script that would eliminate wait times. To use the wait time of the efficient students productively, and to turn that time into a valuable educational experience, we thus developed a new pedagogical script.

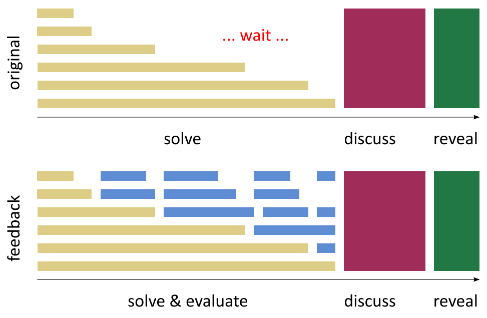


Figure 5: Pedagogical Scripts

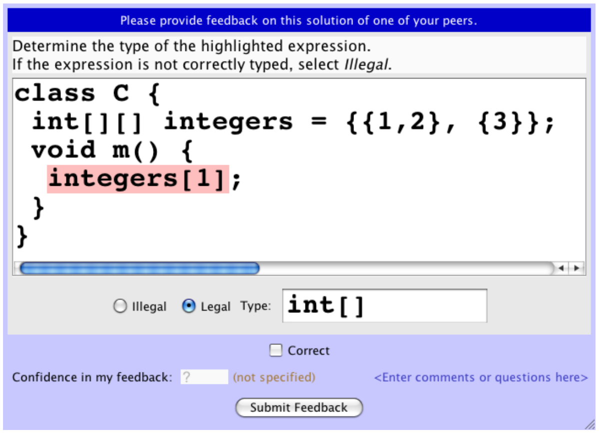
The new script, shown at the bottom of Figure 5, adds a new phase, *evaluate*, right after the *solve* phase. After submitting his problem, a student receives a solution submitted by one of his peers and needs to evaluate that solution and to submit his evaluation. During the evaluation phase, the student continues evaluating his peers' solutions until the instructor moves on to the discussion phase.

Note that the goal of the evaluation phase is not at all to relieve the instructor from evaluating the students' solutions. For many problem types, Informa can automatically evaluate the correctness of a solution (e.g. by comparing it to the correct solution provided by the author of a problem, or by automatically checking certain properties, such as whether a submitted piece of Java code compiles). The goal of having students evaluate the solutions of their peers, besides keeping them focused while waiting for the slower students, is to expose them to the various types of mistakes one could make in order to strengthen their understanding of the problem.

5. IMPLEMENTATION IN INFORMA

Informa is an extensible framework written in Java. We extended Informa by developing plugins for the three new problem types: types, control flow, and coding. These plugins extend the set of available problem types beyond the multiple choice and text highlighting available in the original version of Informa [7].

We also extended Informa to support the new pedagogical script based on the Solve & Evaluate approach. In the clicker application running on the students' computers, we have provided a new view (Figure 6) that allows them to evaluate a peer's solution. The view presents the solution, and asks the evaluator to specify three things: (1) whether the solution is correct or not, (2) comments about the solution, and (3) confidence in his evaluation.


Please provide feedback on this solution of one of your peers.

Determine the type of the highlighted expression.
If the expression is not correctly typed, select *Illegal*.

```
class C {
 int[][] integers = {{1,2}, {3}};
 void m() {
  integers[1];
 }
}
```

Illegal [✓ Legal] Type: int[]

☐ Correct

Confidence in my feedback: ? (not specified) <Enter comments or questions here>

Figure 6: A Student evaluating a peer's solution

During the solve & evaluate phase, the instructor can display the progress of the phase on the classroom projector in the form of a matrix (Figure 7). Each row and each column corresponds to a student. The rows represent a student's submitted solution, and the columns represent the evaluations he submitted. Each cell in the matrix represents one specific evaluation (one solution being evaluated by one peer): large circles correspond to an evaluation stating that the solution is "correct", small circles mean "incorrect". The shade of each circle corresponds to the confidence of the evaluator (the darker the shade the more confident the evaluator). Row and column headers show a color coding of the status of that student (red means the student is still in the solve phase, blue means the student is already in the evaluate phase). Students cannot evaluate their own solution, and thus the diagonal of the matrix is blank.

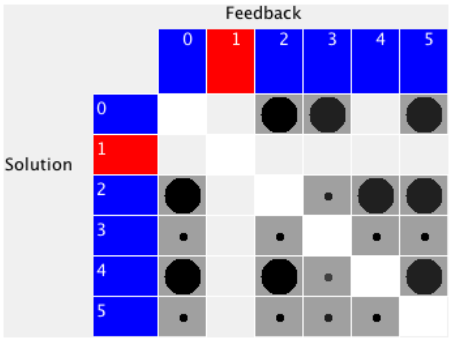


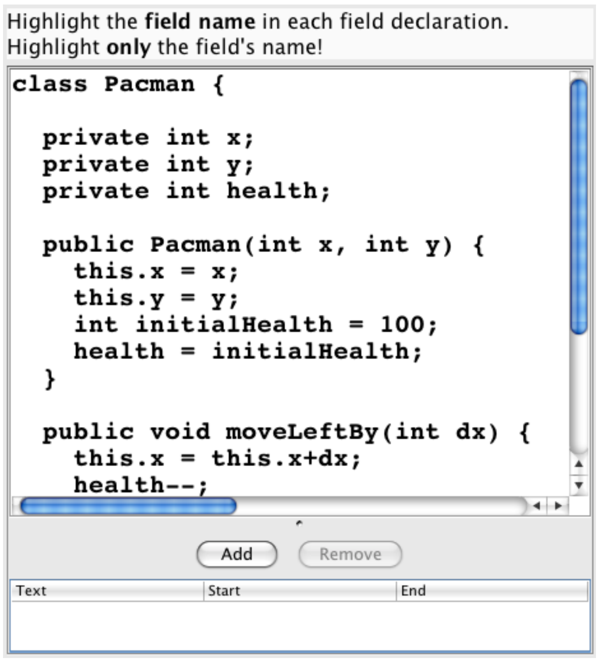
Figure 7: Progress during solve & evaluate phase

Figure 7 shows that one student (number 1) has not yet submitted his solution, while all other students have submitted their solutions, and most students have already evaluated the solutions of all of their peers (student 4 still needs to evaluate the solution of student 0). The figure shows that the solutions of students 3 and 5 seem to be incorrect (all other students evaluated them as incorrect). This progress visualization allows instructor and students to quickly see whether a solution is likely correct (a row with mostly large circles), or whether a problem was too difficult for the current skill level of the class (a matrix filled with mostly small circles) solely based on the feedback provided by the students.

6. CASE STUDY

We have been using Informa's Solve & Evaluate approach in our Java programming course (Programming Fundamentals II) at the University of Lugano. In this section we discuss four problems (one of each problem type) we used in our course and we analyze the behavior of the students during the solve & evaluate phase. We used Informa in anonymous mode: Students were aware that we were unable to identify their solutions, and even the students did not know their own numeric identifier (as used in the matrix of Figure 7). For this case study we instrumented Informa to capture a trace of each session. Besides containing all the posted problems and submitted solutions and feedbacks, the traces also contain time stamps of all activities. This allows us to analyze the temporal progression for each session.

6.1 Syntax


Highlight the **field name** in each field declaration.
Highlight **only** the field's name!

```
class Pacman {

  private int x;
  private int y;
  private int health;

  public Pacman(int x, int y) {
    this.x = x;
    this.y = y;
    int initialHealth = 100;
    health = initialHealth;
  }

  public void moveLeftBy(int dx) {
    this.x = this.x+dx;
    health--;
```

Figure 8: Syntax: Problem

Figure 8 shows a syntax problem we used to check whether students understand the difference between declaration and use of a variable and the difference between instance variables, local variables, and parameters.

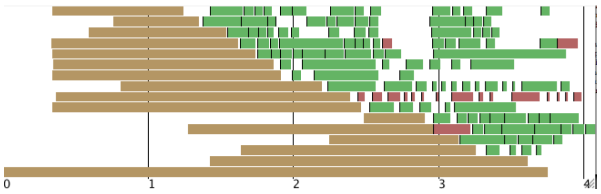
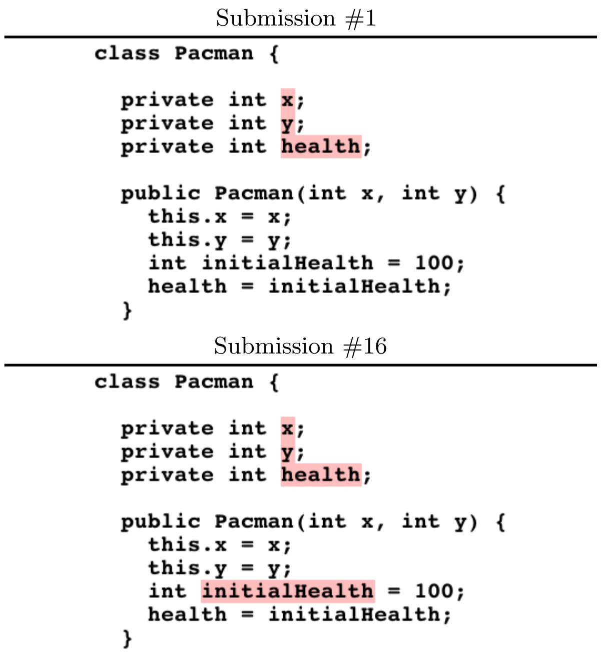


Figure 9: Syntax: Timeline

Timeline. We posed this problem to our class during a session in our course. Figure 9 shows the timeline of the Solve & Evaluate phase (following the notation of Figure 5). It is based on the data gathered by our instrumentation during the session. The x-axis represents time (0 to 4 minutes). On the y-axis we have one row for each student (16 students submitted solutions for this problem). The row for a student shows a sequence of non-overlapping bars. The first bar corresponds to the time interval in which the student solved the problem (from the time he received the problem until he submitted the solution). The top row shows the student who first submitted his solution. The subsequent bars on each row correspond to the time intervals when the student evaluated a peer solution. The colors of those bars correspond to whether he evaluated the peer's solution as correct (green) or not (red).

The timeline shows that our Solve & Evaluate approach was successful in that the students indeed spent most of their waiting time evaluating their peers' solutions.


Submission #1

```
class Pacman {

  private int x;
  private int y;
  private int health;

  public Pacman(int x, int y) {
    this.x = x;
    this.y = y;
    int initialHealth = 100;
    health = initialHealth;
  }
```

Submission #16

```
class Pacman {

  private int x;
  private int y;
  private int health;

  public Pacman(int x, int y) {
    this.x = x;
    this.y = y;
    int initialHealth = 100;
    health = initialHealth;
  }
```

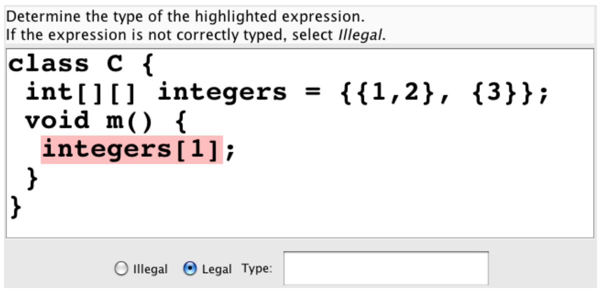
Figure 10: Syntax: Solutions

Solutions. Figure 10 shows the two kinds of solutions submitted. Submission #1 is correct and identical to all other submissions, except for the last submission, #16. Submission #16 contains one mistake, the inclusion of the declaration of local variable `initialHealth`. This points out that the student did not understand the difference between a local variable declaration and a field declaration.

Given that this incorrect solution was the last submission, none of the students got to evaluate its correctness. This is a potential weakness of our approach. However, the bigger issue with this problem was that it was not difficult enough. Formative assessment is based on the idea that students will learn from mistakes. If the problems are too easy, students will not make mistakes and will thus not learn anything new. If we had picked a more difficult problem, the submitted solutions would have included several interesting mistakes, which students would have analyzed during their evaluations, and the instructor could have clarified in the discussion.

6.2 Types

Figure 11 shows a typing problem with which we intended to check the students' understanding of multidimensional arrays.


Determine the type of the highlighted expression.
If the expression is not correctly typed, select *Illegal.*

```
class C {
 int[][] integers = {{1,2}, {3}};
 void m() {
  integers[1];
 }
}
```

Illegal [✓ Legal] Type:

Figure 11: Typing: Problem

Timeline. The timeline in Figure 12 shows the progress of the 21 participating students. While the instructor moved on to the discussion phase after roughly 3.5 minutes, some students continued with their evaluations even past the 7 minute mark. The third to last student is not visible in the figure. The reason for this is that the clock of his computer was not set correctly and thus the timing information about his submissions was off by a large amount[1].

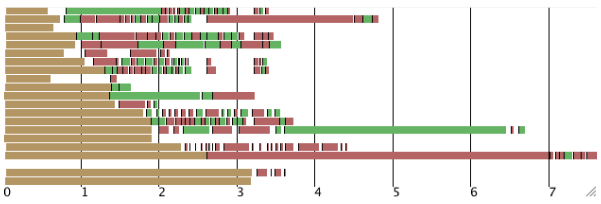


Figure 12: Typing: Timeline

Solutions. Table 1 shows a summary of the submitted solutions. While the majority of the 21 students submitted the expected answer (`int[]`), a sizable minority found that the expression was not legal. With their answers the students uncovered a mistake in the problem: in Java, an ExpressionStatement cannot contain an arbitrary expression (it only allows assignments, pre and post increment/decrement expressions, method invocations, and object instantiations). While technically the highlighted expression is a legal expression of type `int[]`, the class in which the expression is embedded would not compile. We intentionally used an ExpressionStatement for placing the expression to avoid giving away the expression's type: e.g. using an assignment statement like `int[] a = integers[1]` would have directly pointed the students to the solution. However, we did not validate that the resulting source code indeed was correct.

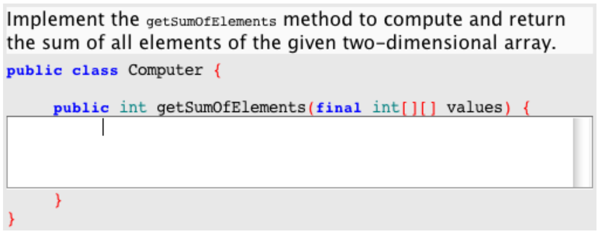
[1]Our current implementation gathers timing information on the students' computers and thus depends on their time being set correctly.

Count	Type
10	int[]
8	*illegal*
2	int
1	int[][]

Table 1: Typing: Solutions

6.3 Coding

Figure 13 shows a coding problem testing the understanding of arrays of arrays, nested loops, and local variables.


Implement the `getSumOfElements` method to compute and return the sum of all elements of the given two-dimensional array.

```
public class Computer {

    public int getSumOfElements(final int[][] values) {

    }
}
```

Figure 13: Coding: Problem

Timeline. The timeline in Figure 14 shows an interesting phenomenon. The top few rows contain a one minute gap between 5 and 6 minutes. The reason for this gap is that nobody submitted new solutions during this time interval. Thus the students who had already evaluated all previously submitted solutions had to wait. Moreover, it is interesting to see that the top two students took much longer than normal for the first evaluation after that gap. This is most probably because they focused their attention on something else while waiting, and did not immediately get back to evaluating once the new solution arrived.

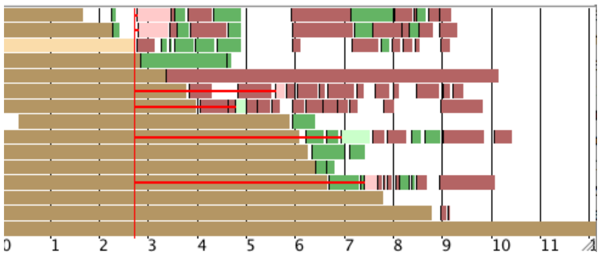


Figure 14: Coding: Timeline

Solutions. Figure 15 shows three of the 15 submitted solutions. Solution #1 uses new-style for-each loops. While students had previously seen such loops for iterating over collections, they had not been used for iterating over arrays of numbers. This solution thus required the evaluators to think about the applicability of this type of loop to arrays, something they might not have considered if all they had to do was submitting their own solution. Solution #3 contains a simple mistake (the use of `j` as an array index). The fastest two students immediately flagged it as incorrect. This solution provided a good motivation for discussing the benefits of for-each loops: it would have been impossible to make that mistake with a for-each loop. Submission #6 shows the solution of a relatively confident student ("most probably correct") who made two mistakes: he forgot to initialize the loop variables and he redeclared the sum. Moreover, this student did not consistently indent the code, making the evaluation of the solution's correctness by a human slightly more difficult. We hope that having to evaluate inconsistently formatted solutions by peer students will help to convince students to improve the formatting of their own code. Finally, this student's use of temporary variables (`x` and `y`) for holding the array length can raise questions about advantages and disadvantages of introducing such temporaries and about the use of meaningful names.

Submission #1

```
int sum = 0;
for (int[] val : values) {
  for(int v : val) {
    sum += v;
  }
}
return sum;
```

Submission #3

```
int sum = 0;
for(int i = 0; i< values.length; i++){
  for(int j = 0; j < values[j].length; j++){
    sum = sum + values[i][j];
  }
}
return sum;
```

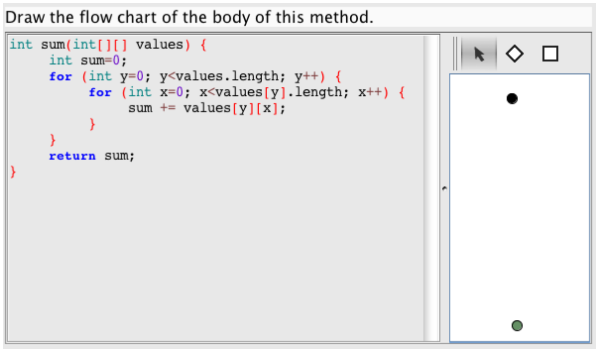
Submission #6

```
int x = values.length;
int sum=0;
for(int i;i< x;i++){
int y = values[i].length;
  for(int j;j< y;j++){
  int sum = sum + values[i][j];
  }
  }
  return sum;
```

Figure 15: Coding: Selected Solutions

6.4 Control Flow

Figure 16 shows a control flow problem testing the understanding of nested for loops. We posted this problem directly after the above coding problem. Students had thus already tried to write nested loops themselves, and now they had to show that they understood the semantics of such loops.


Draw the flow chart of the body of this method.

```
int sum(int[][] values) {
    int sum=0;
    for (int y=0; y<values.length; y++) {
        for (int x=0; x<values[y].length; x++) {
            sum += values[y][x];
        }
    }
    return sum;
}
```

Figure 16: Control Flow: Problem

Timeline. Figure 17 shows the timeline of the Solve & Evaluate phase with the 19 participating students. We can see that the fastest student submitted a solution after roughly 4.5 minutes, while the slowest students took over 16 minutes. Some students would have taken even longer, if the instructor had not moved on to the discussion phase after about 15.5 minutes, causing the six slowest students to quickly submit whatever partial solutions they had. Note that Informa does not force students to submit anything, and it does not prevent late submissions (the figure shows one student continuing to submit evaluations even after 22 minutes).

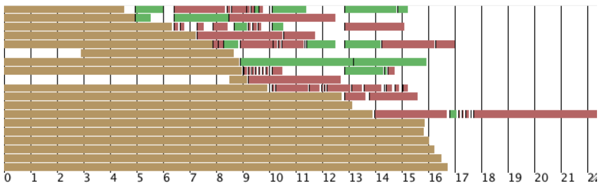


Figure 17: Control Flow: Timeline

The timeline shows that two students started late working on their solution (after roughly 3 and 8.5 minutes). Given that Informa posts a given problem to all students at the same time, these two students must have connected to the Informa session after the session had already started. The student connecting at 8.5 minutes took less than a minute to submit his solution. It turns out, however, that he did not actually solve the problem but submitted an empty solution. The other latecomer submitted a complete and almost correct solution.

Solutions. Figure 18 shows three of the 19 solutions. Submission #1, the first submitted solution, is wrong in many respects. It confirms that the quickest student is not necessarily always the best student. The submission shows a fundamental lack of understanding: while all the nodes in the control flow graph are connected and all paths from the entry node lead to the exit node, the graph does not contain all the necessary nodes. Moreover, the graph does not contain any cycles, and some paths go straight to the exit node without going through the return statement. Looking at this graph, an instructor can immediately come up with points she needs to discuss and clarify in class. It is not clear whether the student was confused by the graphical notation used (and e.g. was reluctant to draw an upwards arrow to introduce a loop back edge), or whether he indeed did not understand the principle of iteration. A discussion in class can immediately clarify this point. As submission #5 shows, there is more than one student who is confused about back edges. Submission #5 is already closer to a correct solution: it contains all the required nodes, and all paths go through the return statement. Submission #10 is one of the correct solutions.

7. LESSONS LEARNED

This section discusses the lessons we learned from our experience in using Informa's Solve & Evaluate approach in our course, from the instructor's as well as from the students' perspective.

7.1 Instructor's Perspective

What problems to post? Like any teaching activity, an Informa session requires a significant amount of preparation prior to a lecture. The instructor has to select or create the problems to be used. While this activity is supported by our interactive problem editors, the main challenge is not the actual editing of a problem, but the pedagogical issues that should motivate the use of a specific Informa problem. The main purpose of Informa is not *summative* assessment (evaluating the students) but *formative* assessment (supporting the learning process). Thus, the goal of a specific problem should be to uncover misunderstandings and to trigger discussions and clarifications.

How many problems to post? It is tempting to try to post too many problems in a session, or to make the problems too complex. We have found it most useful to only use a small number of problems, about two to three, and to keep the complexity of each problem to a minimum. As our case study shows, the *solve & evaluate* phase of a problem can take over ten minutes. Including the *discuss* and *reveal* phases, a single problem can thus take in the order of 5 to 30 minutes. Note, however, that this time can be a very effective period of learning that allows the instructor to focus the teaching on exactly the issues the students still need help with. Using our approach takes some extra time compared to traditional teaching, but we found this cost to be outweighed by the benefit of being able to better address the needs of the students.

How to manage time? Our case study has shown that time required for solving a problem can vary up to an order of magnitude between the fastest and the slowest students. How long should an instructor wait before moving on to the *discuss* phase? We have found the progress view (Figure 7) to be very valuable in this decision. During the *solve & evaluate* phase, we continuously monitor that view, and we tend to move on once a significant majority of the students have submitted their solution. We also use the time during the *solve & evaluate* phase to study the submitted solutions to identify the issues we want to point out in the *discuss* phase.

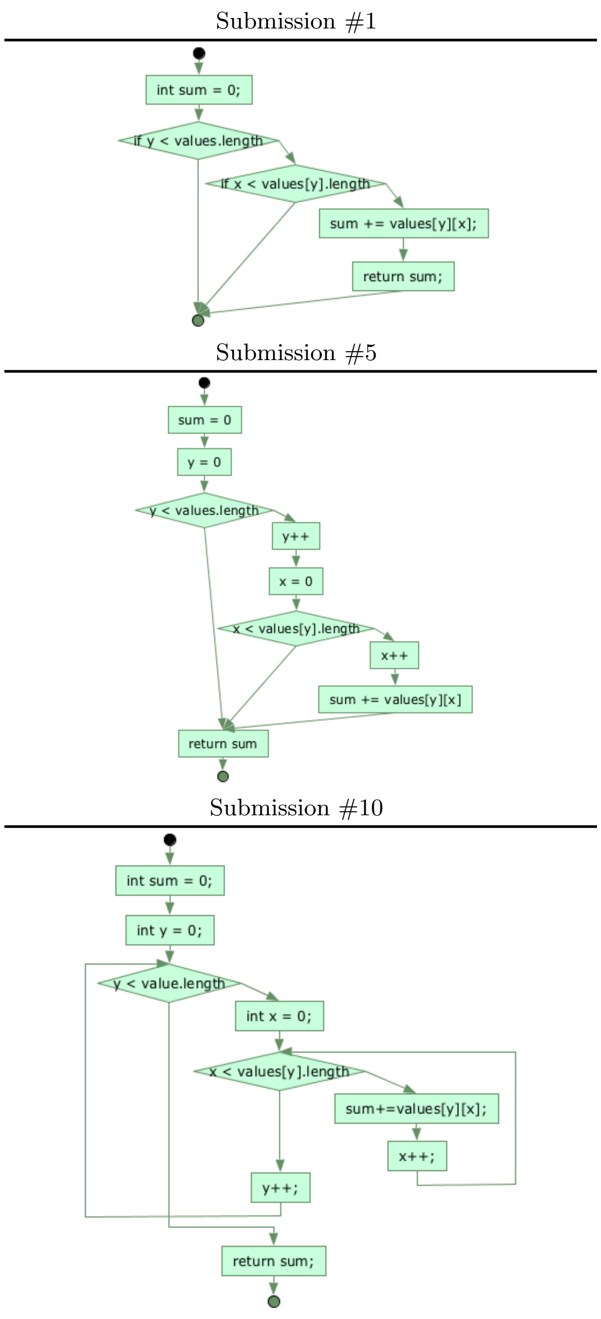


Figure 18: Control Flow: Selected Solutions

Technical issues. Informa uses Java RMI to communicate between the instructor and student applications. The correct functioning of the system thus requires a sufficiently sized network (particularly in the case of large classes using wireless connections). Moreover, students may have to configure their personal firewall to allow incoming and outgoing network connections for Java programs.

Benefits. We felt that Informa provided us with an immediate view of the level of understanding of our students, turning traditional lectures into effective two-way communications in the classroom. Our approach represents a form of problem-based learning, helping us to naturally introduce new topics, triggered by the students' needs. Finally, as a teacher, being able to follow the students' gradual improvements over the course of multiple sessions provided us with certain degree of personal satisfaction.

7.2 Students' Perspective

We distributed a questionnaire to all students registered in our programming course, asking for their opinion about the Solve & Evaluate approach for Informa. The questionnaire was optional but not anonymous, allowing us to correlate the responses with the grades of the students. Students had to provide a numerical answer (0=strongly disagree to 5=strongly agree) to a set of 10 questions, and they could provide their requests for improvements. Figure 19 shows the means and 95% confidence intervals[2] for each of our 10 questions, based on the 14 responses we received.

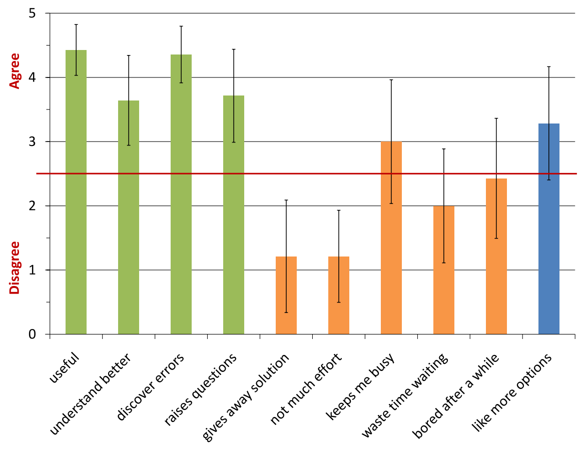


Figure 19: Questionnaire Results

Is it useful? The students agreed with the statement "Generally, I think that the evaluation phase is useful" with a mean of 4.43 ± 0.40.

For whom is it useful? We were wondering whether the students' perception of usefulness depended on the skill level of the students? To answer this question we correlated the response to "useful" with the numerical grade of the students, and we found a Pearson's correlation coefficient of -0.51 (with a 95% confidence interval from -0.82 to +0.022). While the correlation coefficient is negative, indicating that weaker students found our new approach more useful than their stronger peers, the 95% confidence interval straddles 0, and thus that negative correlation is not statistically significant.

Why is it useful? We asked students whether the evaluation phase helped them to better understand the problem (agreement of 3.64 ± 0.70), to discover errors in their own solution (agreement of 4.36 ± 0.44), or to raise new questions they would like to discuss in class (agreement of 3.71 ± 0.72). The students' agreement to all three statements was thus statistically significant.

Potential issues. We further asked the students whether evaluating their peers' submissions gave away the correct solution and thus eliminated their interest in discussing the solution with the instructor. The respondents clearly disagreed (1.21 ± 0.88). Moreover, we were wondering whether the students really took the evaluation phase seriously. We thus asked whether they would usually not put much effort into evaluating their peers solutions. They also significantly disagreed (1.21 ± 0.72).

The answers to the remaining three questions, while positive from our perspective, were not statistically significant (the confidence interval straddled 2.5). We asked whether the evaluation phase was keeping them busy while waiting for their classmates' answers, and whether doing these evaluations was better than just waiting. Their agreement was 3 ± 0.96. In a related question we wanted to know whether even with the evaluation phase, they still felt like they wasted time waiting for new submissions to evaluate. Their disagreement was 2 ± 0.89. Finally, we asked whether, after evaluating several of their peers' solutions, they got bored. They disagreed with 2.43 ± 0.94.

Improvements. We were wondering whether the students would appreciate a more detailed evaluation, where, instead of just labelling a solution as correct or incorrect, they would have to evaluate different aspects of that solution (e.g. whether it compiles, or whether it causes runtime errors). Their response to this proposal was measured, with a mean of 3.29 ± 0.88, they slightly favored performing more involved evaluations. Besides their numerical responses to our questions, the students also provided valuable comments. They included general comments such as "I think that correcting the solution is a good way for learning", or comments about the usability of the user interface. One key comment was that they found the *reveal* phase very important, even after going through the *evaluate* phase, in order to see a solution which definitely is correct. Another request was to add more flexibility to the *evaluate* phase, allowing a student to go back and forward between submissions in order to contrast and compare them.

[2]These intervals are just approximations of statistical confidence intervals, given that we did not pick a random sample of the overall student population but only a self-selected group of students in our own course.

8. RELATED WORK

Group response systems have been used for the continuous gathering and evaluation of feedback about student learning [1, 5] for over a decade. While traditional clicker devices are special purpose remote controls with a limited user interface consisting of only a small number of buttons, some educators have used laptops [4] or programmable calculators [11] instead of proprietary clicker devices. Fies and Marshall [6] and Roschelle et al. [12] survey clicker-related research. They point out the logistical difficulties to instructors and the cost of purchasing clickers for students, two issues that our software-based clicker approach can overcome. They also find that next generation clickers should focus on *formative* assessment, which is exactly what we do with our approach.

Most prior work has studied group response systems with a focus on the use of multiple-choice questions [1, 4, 10, 13]. However, two related research efforts support more advanced types of problems. A project at SRI's CTL uses graphing calculators for the submission of polynomial equations in mathematics courses [11], and the "Classroom Learning Partner" (CLP) project at MIT uses pen-based Tablet-PCs to allow students to submit answers to questions [8, 9]. The CLP allows free-form graphical input by the students and thus first has to correctly interpret the electronic ink before being able to automatically compare or evaluate them. The CLP might thus significantly benefit from using our pedagogical script, which would allow the system to make use of humans for the evaluation.

To the best of our knowledge, no existing group response system has explored pedagogical scripts like ours, where evaluating the solutions submitted by peers turns the wait time of faster students into a valuable part of the learning process. However, traditional collaborative learning approaches, such as the "Think-Pair-Share" and the "Send-A-Problem" techniques [2], have similar goals. In "Send-A-Problem", in a first stage multiple groups solve a given problem, and in a second stage the groups analyze, evaluate, and synthesize the solutions. In "Think-Pair-Share" students first think about a problem individually, then form groups and share their ideas with a partner. Both techniques first proceed towards a solution and then use peers to analyze the result. This analysis phase is similar to our evaluation phase: it allows students to compare and discriminate among multiple solutions.

9. CONCLUSIONS

Have you ever been wondering whether a concept you just explained was truly understood by all your students? This paper presents an approach that can provide you with immediate feedback on the level of understanding of your students, that provides specific support for learning and evaluating skills relevant for programming, and that turns mistakes made by students into learning opportunities for their peers.

We describe how we used an implementation of this approach in an undergraduate Java programming course and discuss the lessons we learned. We found that this approach significantly improved our teaching, and we plan to make our system available[3] open source, so others can use and extend it in the future.

10. ACKNOWLEDGEMENTS

We would like to thank our students at the University of Lugano for their active participation, enthusiasm, and feedback, and Mehdi Jazayeri and our anonymous reviewers for their valuable comments.

11. REFERENCES

[1] A. L. Abrahamson. An overview of teaching and learning research with classroom communication systems (CCSs). In *Proceedings of the International Conference of the Teaching of Mathematics*, June 1998.

[2] E. F. Barkley, K. P. Cross, and C. Howell Major. *Collaborative Learning Techniques*. Jossey-Bass, 2005.

[3] E. Dijkstra. Structured programming. pages 41–48. Yourdon Press, Upper Saddle River, NJ, USA, 1979.

[4] S. W. Draper, J. Cargill, and Q. Cutts. Electronically enhanced classroom interaction. *Australian journal of educational technology*, 18(1):13–23, 2002.

[5] D. Duncan. *Clickers in the Classroom*. Pearson Education, 2005.

[6] C. Fies and J. Marshall. Classroom response systems: A review of the literature. 15(1):101–109, March 2006.

[7] Matthias Hauswirth. Informa: An extensible framework for group response systems. In *Proceedings of the 4th International Conference on Collaborative Computing (CollaborateCom'08)*, November 2008.

[8] K. Koile and D. Singer. Development of a tablet-pc-based system to increase instructor-student classroom interactions and student learning. In *Workshop on the Impact of Pen-based Technology on Education*, April 2006.

[9] K. Koile and D. Singer. Improving learning in cs1 with tablet-pc-based in-class assessment. In *Submitted to Second International Computing Education Research Workshop*, September 2006.

[10] R. Pargas and D. Shah. Things are clicking in CS4. In *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE)*, March 2006.

[11] J. Roschelle, P. Vahey, D. Tatar, J. Kaput, and S. Hegedus. Five key considerations for networking in a handheld-based mathematics classroom. In *Proceedings of the 27th Conference of the International Group for the Psychology of Mathematics Education*, July 2003.

[12] Jeremy Roschelle, William R. Penuel, and Louis Abrahamson. Classroom response and communication systems: Research review and theory. In *Annual Meeting of the American Educational Research Association*, April 2004.

[13] A. R. Trees and M. H. Jackson. The learning environment in clicker classrooms: Student processes of learning and involvement in large courses using student response systems. *Learning, Media and Technology*, 32(1):21–40, March 2007.

[3] `http://www.informaclicker.org/`