The JavaFest: A Collaborative Learning Technique for Java Programming Courses

Matthias Hauswirth, Dmitrijs Zaparanuks, Amirhossein Malekpour, Mostafa Keikha
Faculty of Informatics, University of Lugano
Lugano, Switzerland
Matthias.Hauswirth@unisi.ch, {zaparand,malekpoa,keikham}@lu.unisi.ch

ABSTRACT

Learning to create well-designed and robust Java programs requires, besides a good understanding of the language, a significant amount of practice. In this paper we present the JavaFest, a collaborative learning technique for teaching Java to beginning programmers. A JavaFest is a group exercise that instructors can add to their repertoire of teaching techniques. It provides an opportunity for students to practice programming in a motivating but non-threatening environment, and to learn from the experience of their peers. Moreover, a JavaFest allows the instructor to gain insight into the current standing of the students in her class.

We describe the concept of a JavaFest and present three case studies in the form of three concrete JavaFests we developed and evaluated in our own object-oriented programming course. The general idea of a JavaFest, and the three specific examples we describe and evaluate, can easily be adopted to enhance any 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.

1. INTRODUCTION

This paper introduces *JavaFests*, a collaborative learning technique that instructors can included in their repertoire of techniques for teaching Java. JavaFests are not intended to replace traditional teaching techniques such as lectures or more interactive forms such as the Conversational Classroom [12]. They are, however, intended to enrich those techniques by helping students to gain practical experience in programming in a friendly and collaborative setting.

JavaFests differ from traditional techniques for practicing programming in two key aspects: (1) Unlike assignments and term projects, a JavaFest does not include a summative assessment. (2) Unlike individual assignments, projects, or exercises, a JavaFest is a collaborative endeavor. The absence of a grade and the collaborative nature of JavaFests encourage the exchange of knowledge and skills between students. To retain an element of competition, and thus to motivate students, a JavaFest lets teams of students compete against each other.

Besides providing a positive but competitive environment for practicing programming, a JavaFest also allows instructors and teaching assistants to gain a detailed insight into the state of the various students in the teams. Instructors can identify issues they have to cover or repeat in future lectures, and teaching assistants can see the concrete problems students have in developing in Java.

JavaFests also have an integrative aspect. First, they can be used to integrate students with different skill levels, by creating heterogeneous groups where each group consists of more and less advanced students. This encourages the good students to transfer some of their knowledge to their peers. Second, we designed our JavaFests to integrate topics from different computer science courses into our programming course, because we found that students generally appreciate when instructors help them to form connections between the diverse set of courses in the computer science curriculum.

Besides describing JavaFests as a teaching technique, we also present and evaluate the following three concrete JavaFests we developed for our first-year undergraduate Java programming course.

Web Analytics. This JavaFest asks students to develop a web server log file analyzer. It requires students to iterate over a collection of log entries and to aggregate and filter the data in various ways. This fest nicely relates to computer networking courses where students learn about DNS, IP addresses, the HTTP protocol, and related artifacts that show up in a web log file.

Turing Test. The goal of this JavaFest is to develop a program that passes the Turing test (resp. a scaled-down version of it). It allows instructors to make ample connections to the topic of artificial intelligence.

Break It!. In this JavaFest students are asked to implement a given algorithm, and, in a second phase, to develop unit tests that break the implementations of their peers. It helps to prepare students for subsequent courses on software engineering or testing.

In the remainder of this paper we describe the concept of a JavaFest in more detail (Section 2) and then describe our methodology for evaluating this new teaching technique (Section 3). Sections 4 to 6 present and evaluate the three concrete JavaFests mentioned above. Section 7 studies the lessons learned from conducting these JavaFests in our Java course. Section 8 presents related work, and Section 9 concludes.

2. WHAT IS A JAVAFEST?

A JavaFest is a collaborative design and development exercise to be conducted in the classroom over the course of several consecutive lectures (a JavaFest usually requires at least 120 minutes). JavaFests can be used in an introductory Java programming course to complement traditional lectures, homework assignments, and projects. Their goal is to provide an efficient, controlled, practical, collaborative programming experience for the students.

For a JavaFest the class is partitioned into small groups. Each group is asked to collaboratively solve a given programming problem. The problem is the same for each group. At some point during the JavaFest the groups are asked to present their current status to the class. Given that all groups work on the same problem, these presentations allow groups to adopt solution approaches that were successful in other groups.

To structure the collaboration within a group, each group member takes on one of three possible roles. Each group has a *moderator* and a *recorder* (two roles who do not contribute to the technical aspects of the solution), and a few *designers*.

Moderator. The moderator tries to keep the group on track, making sure they make progress. The moderator also ensures that the various designers contribute evenly to the group's design, and that no one is dominating or passive. Finally, the moderator will present the group's design at the presentation.

Recorder. The recorder keeps track of the artifacts produced and key decisions taken by the group. The recorder edits all code, takes notes, and draws diagrams.

Designer. The designers create the design. They propose and discuss solutions. They do not write code (code is written by the recorder).

To motivate groups to perform at their best, a JavaFest contains a competitive element. At the end of the development phase, the different solutions are pitched against each other, and the group with the best solution wins. This competition can focus on any one of many possible aspects. For example, groups may win if their code passes most test cases, their test cases break most code, they implement most of the requested features, their code is judged the most elegant by the other groups, or their code runs fastest.

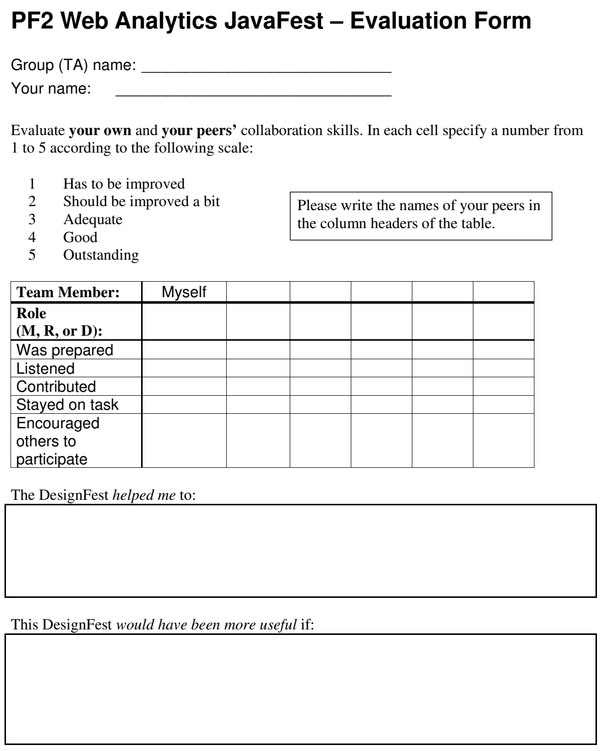
3. METHODOLOGY

We developed the idea of a JavaFest, and three concrete JavaFest instances, for the Spring 2008 "Programming Fundamentals II" (PF2) course at the University of Lugano (USI). The faculty of informatics at USI, established in 2004, has developed an innovative informatics curriculum [6] that includes an early focus on collaborative learning in the programming fundamentals courses [8]. Our PF2 course is based on the "Objects First with Java" textbook [2], and uses the BlueJ [7] development environment. While we developed the JavaFests presented in this paper to fit well with the given textbook, the general ideas apply to any Java programming course, and the particular JavaFests could easily be adapted for use in differently structured courses.

We use the Conversational Classroom [12] approach to teaching in the PF2 course. We have three lectures each week, over the entire course of the semester, where we discuss the readings assigned for each meeting. We conducted the three JavaFests described in this paper during the second quarter of the course. In the first quarter, the students' skills would not be sufficient for an effective JavaFest. During the second half of the course we conducted a group project in parallel to the lectures, giving students ample opportunity to practice.

Given the relatively small class size of 14 students, we partitioned the class into three groups. We used a heterogeneous grouping approach, mixing students with different skill levels. We conciously reassigned students to different groups for each new JavaFest to expose each student to a diverse set of opinions and approaches.

At the end of each JavaFest we had the students complete a questionnaire. The questionnaire, shown in **Figure 1**, consists of two parts. The first part contains a self and a peer evaluation form, modelled after the student self-evaluation and peer evaluation forms of Barkley et al. [1]. These forms require students to rate their own, and their peers', performance on a scale from 1 (has to be improved) to 5 (outstanding) according to five criteria: whether the student was prepared,


PF2 Web Analytics JavaFest – Evaluation Form

Group (TA) name: ___________________________

Your name: ______________________________

Evaluate **your own** and **your peers'** collaboration skills. In each cell specify a number from 1 to 5 according to the following scale:

1	Has to be improved
2	Should be improved a bit
3	Adequate
4	Good
5	Outstanding

Please write the names of your peers in the column headers of the table.

Team Member:	Myself					
Role (M, R, or D):						
Was prepared						
Listened						
Contributed						
Stayed on task						
Encouraged others to participate						

The DesignFest *helped me* to:

This DesignFest *would have been more useful* if:

Figure 1: Evaluation Form

listened, contributed, stayed on task, and encouraged others to participate. The main purpose of these forms is to help students improve their collaboration skills by requiring them to compare their behavior to their peers'. The second part of the questionnaire consists of two open questions: "The JavaFest helped me to:" and "This JavaFest would have been more useful if:". These questions serve to get students' feedback on the effectiveness of the given JavaFest. We combine that feedback with our own observations in the remainder of this paper.

Furthermore, we conducted a midterm course evaluation after the three JavaFests. We regularly conduct such evaluations for our courses to get feedback from the students in order to improve our teaching. They consist of a questionnaire to be completed by students and submitted anonymously. This time we included a few specific questions about JavaFests. In particular, we extracted proposals for improvement from the students' comments from the individual JavaFest questionnaires, and posed them as questions in this overall questionnaire. This allowed us to get the reaction from the entire class on the most promising improvements proposed by individual students. We discuss this overall evaluation in Section 7.

The following three sections introduce the JavaFests we developed for our course. All sections follow the same structure. They briefly outline the problem and then present the schedule, the scaffolding (in the form of code given to the students), and the tasks. Then they describe the integration with other courses in the computer science curriculum, and finally they present the results from the students' questionnaires.

4. WEB ANALYTICS

The first in our sequence of JavaFests focuses on relatively straightforward data processing tasks. Students receive a sequence of data items and need to compute aggregate statistics. In order to excite students about this rather mundane task, we did not pick some arbitrary data, but we chose the current week's log of the HTTP server of our department. This motivated students to unearth information about themselves, their peers, and about their teaching assistants and professors.

4.1 Schedule

We organized the JavaFest into several parts. After a brief presentation of the JavaFest idea and an introduction of the Web Analytics problem, groups started with the initial design of their application. After this first round of design, each group gave a brief presentation of their approach to the entire class. Groups then took the ideas they heard from other groups and improved their own designs accordingly. The competitive aspect of this JavaFest lied in completing as many analysis features as possible in the available time. We ended the JavaFest with the students completing the evaluation questionnaire.

Duration	Task
15 min	Introduction
60 min	Initial design
45 min	Presentations
45 min	Redesign
15 min	Self & Peer Evaluation

4.2 Scaffolding

We provided the students with a BlueJ project containing the classes shown in **Figure 2**.

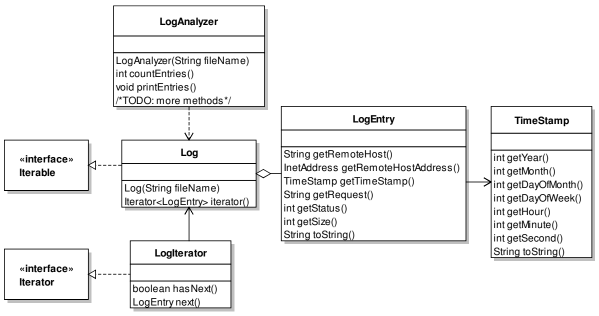


Figure 2: Web Analytics Scaffolding

The Log, LogIterator, LogEntry, and TimeStamp classes provide a convenient interface to accessing all entries in the web server log. This scaffolding shields the students from the complexity of the Java I/O classes. The LogAnalyzer class with its countEntries() and printEntries() methods provides an example of how to use the other classes and serves as a starting point for the students' solutions.

```
public class LogAnalyzer {

  private Log log;

  public LogAnalyzer(final String fileName) {
    log = new Log(fileName);
  }

  public int countEntries() {
    int count = 0;
    for (final LogEntry e : log) {
        count++;
    }
    return count;
  }

  public void printEntries() {
    for (final LogEntry e : log) {
      System.out.println(e);
    }
  }

  // TODO
}
```

Besides the above classes, we included a web server log file in the BlueJ project. We also pointed students to the location of the live log files to show them that they could continue using their analytics software to monitor and analyze future web accesses to our server.

4.3 Tasks

This JavaFest consists of the tasks to compute various kinds of statistics over the week-long web server log. These tasks build on top of the exercises provided in Chapter 4 of the textbook [2], allowing students to deepen their understanding of how to process collections of data items. We handed out the following tasks, and asked students to solve them in the order provided:

- Find the earliest request. Note that LogEntries are not sorted by time stamp!

- Print all requests for files about a specific person. Persons store their web pages in their public_html folder, and these pages are accessible using URLs such as "/∼username/. . . ".

- Print the number of requests (a) that were successful (status 200), (b) where access was forbidden (status 403), and (c) where the document was not found (status 404).

- Analyze the log file to determine how requests are distributed over the 24 hours of a day (how many requests were received in each of the 24 hours). Note that the provided log files contain requests from an entire week (not just one day).

  Produce a horizontal bar chart of the histogram, similar to:

```
 0 | ########
 1 | #######
 2 | ####
 3 | ##
 4 | #
 5 |
 6 |
 7 | #
 8 | #####
...
23 | ################
```

  Also produce a vertical bar chart.

- Provide statistics (in the form of a horizontal bar chart, one bar per person/homepage) about the size of the requested files. Which faculty member or student served the biggest amount of data?

- Provide statistics about the countries from which the requests came from. Use the InetAddress object returned by LogEntry.getRemoteHostAddress() to determine the hostname. Take the last part of the hostname, e.g. "ch", to determine the country.

Some of these tasks are simpler, while others, such as the printing of the vertical bar chart, are more challenging. Besides practicing loops and conditionals, students also have to process strings (at least use substring), and they get an opportunity to explore the API of InetAddress.

4.4 Curriculum Integration

Most students taking our programming course also take a networking course in the same semester. This web analytics JavaFest provides a nice bridge to that course, as students get a chance to study information relevant to the HTTP protocol, such as domain names, URLs, and status codes.

4.5 Evaluation Results

The responses to the question about the benefit of the Web Analytics JavaFest can be grouped into four categories.

- Students commented on having learned how to use specific *Java API* classes (such as HashMap or InetAddress). Some responses included elaborate details, like that they discovered that iterating over the keys of a hash table would not provide a sorted list.

- Students mentioned that it helped them to improve their *collaboration skills*. They listed points such as improvement in coordinating a team, in not forcing one's ideas upon others, in sharing ideas and knowledge between peers, in collaborating to find the best solutions, or in becoming aware of their own level of preparation.

- They listed *problem solving skills*. For example, they mentioned that they learned from their peers how to analyze a problem, how to think more "object-oriented", how to apply the theoretical parts from the lectures in a practical context, and how to efficiently develop a solution to a problem.
- Students also mentioned that they believed that this experience would give them more *confidence in programming something on their own*, or that it would help them to find interesting problems to play around with.

The fact that students learned about Java APIs is not surprising: we would expect the same benefits from normal homework assignments. However, the fact that they also worked on improving their collaboration skills and had a chance to see how their peers were solving a problem is a clear advantage over individual homework assignments. While many of these aspects could also be achieved in group assignments or projects, the JavaFest has the advantage that it takes place in a controlled environment, with clearly defined requirements, and in the presence of a mentor (teaching assistant and instructor).

The feedback about opportunities for improving the JavaFest included that some students felt *constrained by their role* and would have preferred to rotate roles throughout the fest. Moreover, one student found that the role of a moderator should better be taken on by a teaching assistant, and that this TA should provide active guidance to the team, pointing out the right path. Students found that the *competitive element was lacking*, or not strong enough. This may be because we did not explicitly point out the winning team at the end of the JavaFest. Several students would have preferred to have *more time* to solve all the tasks. The teams did not manage to complete all tasks, which clearly impacted their satisfaction. One student would have preferred a *bigger, "more useful" project*. We believe that at this stage of the course not using a large amount of scaffolding (to provide a big project) is beneficial, as it allows students to focus on writing code, rather than reading existing code. However, we think that a JavaFest that includes a significant amount of code reading (e.g. to identify design patterns) would be useful in itself. Some students would have preferred to *pick their own teammates*. Allowing this would, however, prevent the heterogeneous grouping we aimed at.

5. TURING TEST

In a Turing test [11] a human judge conducts a natural language conversation with a partner. The partner can be either a human or a machine. The judge does not see his partner. Both, the human partner and the machine try to appear human. If the judge cannot distinguish between the human and the machine, the machine passes the test, as it appears indistinguishable from a human, and thus is deemed intelligent.

The motivational goal of this JavaFest is to build a machine (a Java program) that passes the Turing test. It certainly won't be possible to build such an intelligent system in a few hours for a beginning programmer. However, constraining the conversation between the judge and the partner to a specific topic, and limiting it to a small number of questions and answers, makes the judge's task quite challenging.

5.1 Schedule

The Turing test JavaFest consists of four parts. First, we present the goal of the activity and introduce the scaffolding. Then the students have two hours for their design and implementation. After this, we conduct the competition to find the winning team. Finally we again ask the students to complete the evaluation questionnaire.

Duration	Task
15 min	Introduction
120 min	Design and Implementation
30 min	Competition
15 min	Self & Peer Evaluation

5.2 Scaffolding

In this JavaFest, students have to develop a class called Responder that has to have a public method respond() that takes the judge's question as an argument and returns the machine's response. We did not formalize the required interface using a Java interface or an abstract class, because at this stage of the course students are not yet familiar with the concept of inheritance.

```
public class Responder {
  public String respond(String question) {
    return "You␣really␣mean␣'"+question+"'?";
  }
}
```

We also provided the students with a library with very basic natural language processing functionality. This library can find the stem of a given word, and it can compute the similarity between two words. For example, students can use the stemmer on the words provided in the question and use the resulting word stems to look up answers in a hash table.

5.3 Tasks

This JavaFest is a continuation of some exercises (in particular Exercises 5.41 and 5.42) in the textbook [2]. The context of the conversation provided in the book is an automated tech support system.

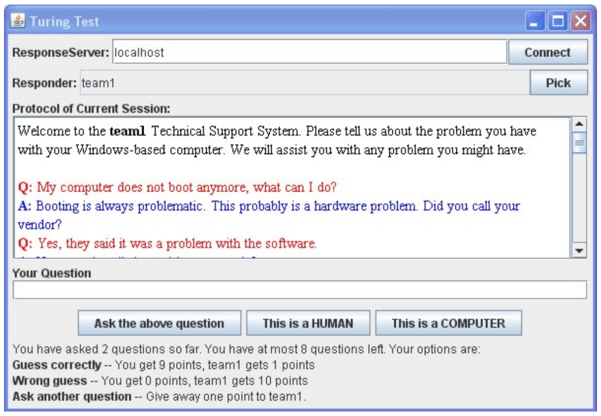
Each team has to develop a Responder class following the example given in the scaffolding. The goal is to improve the tech support system to make it appear more like a human tech support person (Exercises 5.41 and 5.42 give some initial ideas).

Given the feedback from the Web Analytics JavaFest, we strenghtened the competitive element of this JavaFest: we actually performed a Turing test using the various groups' Responders. At the beginning of the fest we explained to the teams that there would be a competition in the end, where their Responder and a teaching assistant would answer the same questions, and where the other teams had to guess whether the responses came from the assistant or the computer. Their goal thus was to develop a Responder that behaved as close to a human as possible.

5.4 Competition

The competition consists of several sessions. In each session, a team A is playing against a team B. Team A is asking questions which are answered by both the responder of team B and the teaching assistant. The students of team B only act as observers and do not actively participate in that session. At the beginning of the session the computer randomly decides whether the answers in this session are to be taken from team B's software responder or from the teaching assistant. Within a session, all answers come from the same source (either team B's responder or the TA). The goal of team A is to figure out, as quickly as possible (by asking as few questions as possible), whether the responses come from the TA or team B's responder.

Note that the TA needs to answer all questions, even when within that session his answers are ignored. This leads to the answers appearing with the same delay, no matter whether they were typed in by the TA or not. This way the TA can be in the same room as the students, as the only information team B has for taking their decision is the text of the answer appearing on the screen. In order to have a fair test it is important that the TA gives realistic answers and does not deliberatly try to appear unintelligent.



ResponseServer: localhost

Responder: team1

Protocol of Current Session:

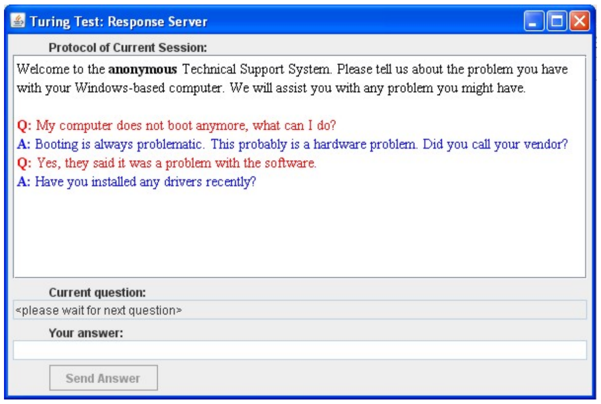
Welcome to the **team1** Technical Support System. Please tell us about the problem you have with your Windows-based computer. We will assist you with any problem you might have.

Q: My computer does not boot anymore, what can I do?
A: Booting is always problematic. This probably is a hardware problem. Did you call your vendor?
Q: Yes, they said it was a problem with the software.

You have asked 2 questions so far. You have at most 8 questions left. Your options are:
Guess correctly -- You get 9 points, team1 gets 1 points
Wrong guess -- You get 0 points, team1 gets 10 points
Ask another question -- Give away one point to team1.

Figure 3: Client

We implemented a distributed Java RMI application to enable this competition. The client (**Figure 3**) is a front-end used by the judge (team A) and runs on the instructor's computer. It is displayed on the classroom beamer for all students to see. Team A (or the instructor on their behalf) uses it to enter questions and to see the answers.



Protocol of Current Session:

Welcome to the **anonymous** Technical Support System. Please tell us about the problem you have with your Windows-based computer. We will assist you with any problem you might have.

Q: My computer does not boot anymore, what can I do?
A: Booting is always problematic. This probably is a hardware problem. Did you call your vendor?
Q: Yes, they said it was a problem with the software.
A: Have you installed any drivers recently?

Current question:
<please wait for next question>

Figure 4: Response Server

The server (**Figure 4**) runs on the teaching assistant's computer. It is invisible to the students. Before the competition all the Responder classes developed by the teams are installed on the response server (on the TA's computer). The response server uses dynamic class loading to load each responder with a separate class loader (they all have the same name). It uses reflection to invoke the Responder.respond() method (because, as described above, the Responders do not implement a common interface).

Given this system, the competition proceeds as follows: Team A can enter a question, which is then sent to the response server on the teaching assistant's computer. The teaching assistant takes on the role of the human partner. The response server takes the submitted question and prompts the teaching assistant to enter an answer. After the teaching assistant completes his answer, the response server sends either that human answer, or the answer produced by the currently chosen Responder class (team B's), back to the client.

Team A then has three choices: It can (1) ask another question, or it can guess that this session is answered (2) by a human, or by (3) team B's responder. In order to compute a score for the competition, team A gets 10 points at the beginning of the session. If team A decides to ask another question they have to pay with one point. If they decide to guess, the session is over and they either win (their remaining points) or lose (and get 0 points).

5.5 Curriculum Integration

Given the topic of the Turing test, this JavaFest is naturally related to artificial intelligence. Depending on the level of the students, it may also make sense to add pointers to information retrieval and natural language processing techniques (e.g. stemming, word similarity).

5.6 Evaluation Results

The positive comments about this second JavaFest were similar to the comments for the WebAnalytics fest. Students found that it helped them to work in groups, exchange points of view, and incorporate other's ideas. They also mentioned that they learned about some API methods, in particular for dealing with Strings.

There were few negative comments, mostly pointing out that the competition was not long enough. Because some groups asked for extensions to improve the strength of their Responders, we dedicated some of the competition time to improving their code.

From the point of view of instructor and teaching assistants, this JavaFest was a worthwhile experience for our students. For some of the Responders developed by the groups, we thought they would be human (based on the limited number of two or three questions teams A used before their decisions), but they were not. The competitive element certainly made this fest interesting for the students, and some of the resulting software responders were rather effective for students with less than half a semester of Java experience.

6. BREAK IT!

The main goal of this JavaFest is to learn to write good unit tests. The teams first need to implement a non-trivial Java class, and then they have to develop unit tests to test their class. To motivate students to write good tests, the JavaFest includes a competition where each team's tests are run against other teams' implementations. The team uncovering the largest number of bugs in the other teams' code wins the competition.

6.1 Schedule

This JavaFest begins with the introduction of the problem and a presentation of the class that needs to be implemented and tested. The teams then complete the implementation of the class and develop unit tests to test their implementation. These phases are followed by the competition, and the JavaFest ends with the evaluation questionnaire.

Duration	Task
15 min	Introduction
75 min	Implement the class
60 min	Develop JUnit tests
20 min	Break It! competition
15 min	Self & Peer Evaluation

6.2 Scaffolding

Before learning Java, our students had already taken a course in Scheme. They were thus familiar with functional programming and recursion. At this stage in the Java course, we were focusing on object references. We were thus looking for a problem that (1) includes objects referring to other objects, (2) includes recursion, (3) provides multiple opportunities for introducing bugs during the implementation, and (4) requires significant thought when writing test cases.

We came up with a class skeleton for a class Bit and the following requirements statement:

A Bit is an immutable object representing a bit. A Bit, however, has an additional interesting feature: it also knows its next more significant bit. This allows the use of Bits to represent arbitrary-length binary numbers. To create a number with value 0, use new Bit() or new Bit(false). To create a number with value 1, use new Bit(true).

Given a Bit b, representing a value N, use b.increment() to create a new Bit with a value N+1. To create a number with value 2, use new Bit(true).increment() or new Bit().increment().increment(). Given two Bits, b1 and b2, representing the values N1 and N2, use b1.add(b2) to create a new Bit with value N1+N2. Given a Bit b, use b.getAsString() to get a String with its binary representation (e.g. new Bit(true).increment() .increment().getAsString() returns "11"). Given two Bits, b1 and b2, b1.equals(b2) returns true only if they have the same value (e.g. new Bit().increment().equals(new Bit(true)) returns true).

```java
public class Bit {
  private final boolean value;
  private final Bit moreSignificant;

  private Bit(final boolean value,
              final Bit moreSignificant) {
    // TODO
  }
  public Bit(final boolean value) {/*TODO*/}
  public Bit() {/*TODO*/}
  public Bit increment() {/*TODO*/}
  public Bit add(final Bit other) {
    return add(other, false);
  }
  public Bit add(final Bit other,
                 final boolean carryIn) {
    // TODO
  }
  public boolean equals(final Bit other) {
    // TODO
  }
  public String getAsString() {
    String result;
    if (moreSignificant == null) {
      result = "";
    } else {
      result = moreSignificant.getAsString();
    }
    if (value) {
      result += "1";
    } else {
      result += "0";
    }
    return result;
  }
}
```

We left the body of most methods empty so that students could implement these methods during the JavaFest.

6.3 Tasks

Each group first had to implement the given Bit class. We asked students to start by implementing the constructors, to move on to the equals method, then tackle the increment method, and finish with the add method. This sequence of tasks exhibits increasing complexity. The constructors are straightforward to implement. The equals method already requires a recursive approach. Students could look at the getAsString method to get an idea for how to proceed. The increment method, and the add method required significant logical thinking skills. We stressed the fact that their implementation would have to be correct and pass the unit tests produced by other teams.

We asked students not to change the public interface of the Bit class, that is, they were not allowed to change the access modifiers of any of the given methods or to add another public method. We also disallowed the removal of the final modifiers of the instance variables, or the introduction of additional instance variables. Furthermore, we discouraged the use of loops. We had three reasons for these constraints. First, we wanted the other groups' unit tests to work with each group's class, and thus we needed a stable interface. As students had not learned about inheritance at this point in the course, we had to enforce the interface in this informal way. Second, we wanted to show the students that in order to test a class, they would need both mutator and accessor functions. Since the Bit class has no simple getter for the instance variables, they have to write test assertions using either the getAsString or the equals accessors. Third, because our students had taken a functional programming course in their prior semester, we wanted students to discover how one can write programs in Java in a functional style, using immutable classes and recursion.

The second task consisted of writing unit tests, and to run them on their own Bit class. We asked the teams to create test methods that cover all of Bit's features. While the students could use the JUnit support provided by the BlueJ environment, the focus of this task was on developing good tests and not on how to use the testing infrastructure, as students already familiarized themselves with using BlueJ's testing features as part of a prior homework assignment.

6.4 Competition

The competition for this JavaFest is straightforward. Each group's tests are run on each other group's code, and each group gets one point for each bug it uncovers in each other group's code.

6.5 Curriculum Integration

This JavaFest nicely connected to the computer architecture course the students had taken in the first semester (especially due to the concepts of binary numbers and full adders). It also allowed us to draw connections between object-oriented programming in Java and functional programming in Scheme (due to recursion and immutability).

6.6 Evaluation Results

In addition to answers similar to the ones from prior JavaFests, the students especially appreciated the connection to their prior courses and the juxtaposition of object-oriented and functional programming.

They stated that the JavaFest would have been more useful if they had had more time, for the competition (to explain or find the bugs) and the implementation (to complete all the methods). They also mentioned that they would have preferred smaller groups.

7. LESSONS LEARNED

In this section we discuss the lessons we learned from using JavaFests in teaching object-oriented programming in Java.

7.1 Collaboration

An important goal of a JavaFest is to encourage collaborative learning. As we described in Section 3, we had students evaluate their and their peers' performance.

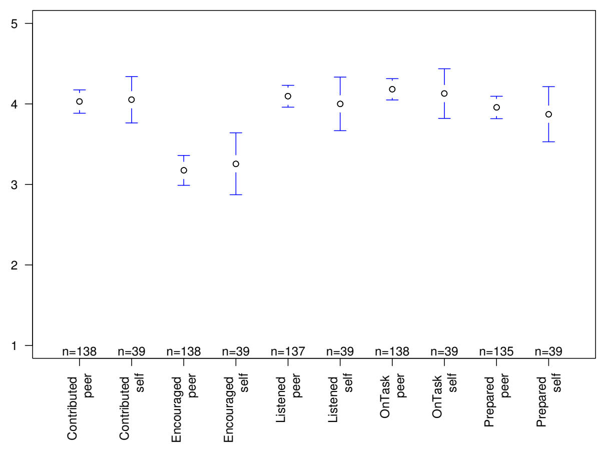


Figure 5: Self and Peer Evaluation Results

Figure 5 shows the means of the evaluation scores with 95% confidence intervals. The scores along the y-axis range from 1 (has to be improved), over 2 (should be improved a bit), 3 (adequate), 4 (good), to 5 (outstanding). Each of the five evaluated aspects (prepared, listened, contributed, stayed on task, and encouraged others to participate) is represented by a pair of data points along the x-axis. The left data point of each pair corresponds to the evaluation by the peers, while the right data point corresponds to the self evaluation. The luxury of having small class sizes in our course limits the statistical significance of our results: Most confidence intervals in Figure 5 overlap, except for the evaluation of the encouragement. The relatively weak evaluation of whether participants encouraged others to participate (an average of "adequate") provides a motivation to assign the role of the "moderator" to a (more experienced) teaching assistant.

7.2 Overall Evaluation

After the three JavaFests, as part of our regular course evaluation, we asked students to complete a questionnaire. We extended the standard questionnaire with questions specifically targeted at JavaFests. The following evaluation is based on the twelve completed questionnaires our students returned to us.

We asked the students how the JavaFests affected three skill sets relevant to our course. The questions we asked were:

From problem to program. JavaFests helped me to learn more about transforming a problem into a program.

Teamwork. JavaFests helped me improve my teamwork and collaboration skills.

Language and API. JavaFests helped me to learn more about the Java language and APIs.

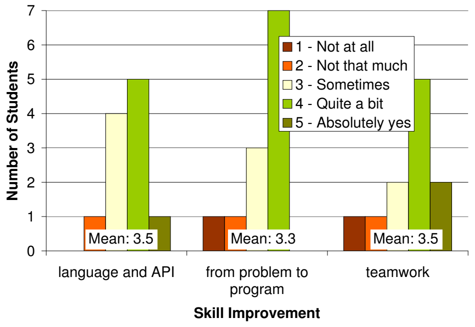


Figure 6: Skill Improvements due to JavaFests

Figure 6 shows the students' ratings of how much each of the three skill sets improved due to the JavaFests. For each skill set it shows the distribution of responses. We can see that the mean for each of the three skill sets is either 3.3 or 3.5. This means that the students found that the JavaFests helped them between "sometimes" (3) and "quite a bit" (4).

The following questions are based on the students' requests for improvement which we had gathered in the feedback forms of each JavaFest. We gathered the five most promising proposals for improvement and polled the students about their preference:

Students=Designers. All students should be designers.

TA active. TAs should take on an active role (guiding us down the best path).

Students pick team mates. I would prefer if we would pick team mates (instead of teams being predetermined).

Competition. I prefer JavaFests where there is some kind of competition.

Realistic tasks. I would like JavaFests with more realistic tasks, even if that meant that we would not start from scratch, but with some code provided to us.

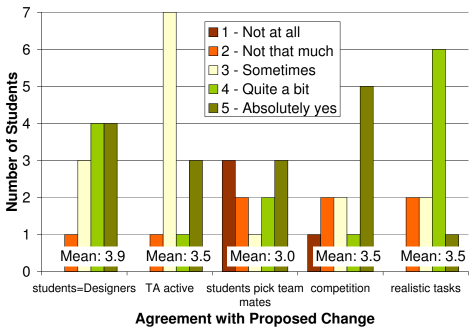


Figure 7: Agreement with Proposed Change

Figure 7 shows the results. It shows that the most promising change, from the students' perspective, is to eliminate the explicit assignment of roles and to have all students be designers. The figure also shows an interesting bi-modal distribution in the response to "students pick team mates": Having students pick teams and predetermining team membership are equally (dis)liked. The remaining three proposed changes all get the same slightly favorable rating (3.5).

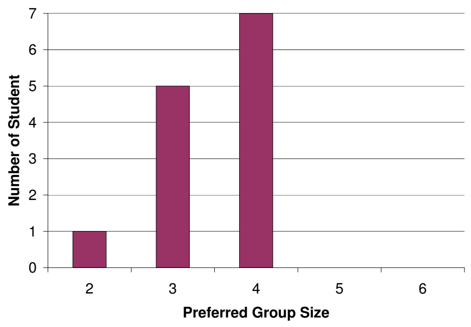


Figure 8: Preferred Group Size

We also asked students about their preferred group size. **Figure 8** shows that they clearly prefer small groups: Not a single student would prefer a group of more than four students.

We learn the following lessons from this evaluation. We will use group sizes of four students, given the decrease of diversity in even smaller groups and the students' preference against larger groups. In our future JavaFests we intend to always include a clearly delineated competition. Moreover, we plan to study the effect of eliminating explicit roles, and we will add some JavaFests with more realistic problems, where students have to extend a given system.

8. RELATED WORK

A rich body of prior work studied collaborate learning, some explicitly focusing on introductory computer science courses. LeJeune [9], for example, proposes a general collaborative learning model. On a high level, the main differences between that model and the approach we use in our JavaFests is that LeJeune's model explicitly fosters positive interdependence between group members (by having different members acquire different expertise prior to the collaboration), while our approach explicitly fosters competition between groups (by having groups compete for the best solution).

JavaFests are a variant of the DesignFest idea pioneered by the OOPSLA conferences [4]. DesignFests are workshops that give attendees the chance to learn more about design by participating in a design project with their peers during the conference. DesignFests were introduced at OOPSLA in 1995 and have taken place every year since then. JavaFests, unlike the OOPSLA conference DesignFests, are targeted at undergraduate students taking an introductory Java programming course. Instead of focusing exclusively on design, JavaFests also include other activities relevant to student Java developers, such as programming and testing. Moreover, JavaFests also include a competitive aspect, which we have strengthened based on student feedback. The fact that teams compete against each other, and that the result of the competition is clearly measurable, provides a form of closure to the activity and makes JavaFests particularly attractive for use in the classroom.

The team organization we used in the JavaFest conducted for this study is similar to the pair programming approach [13]. The recorder (in pair programming: the "driver") sits at the computer, while the designers (the "observer") sit around her. However, unlike in pair programming, during a JavaFest the designers are providing the input, while the recorder's role is less creative. Moreover, a group contains multiple designers, and thus a moderator is used to coordinate the interaction. We have found in our evaluation that students prefer teams consisting entirely of designers. We believe (and some students' comments indicate this) that having a moderator or recorder role is beneficial for developing team work skills. Finally, unlike pair programming, a JavaFest is a learning experience, not an approach to improve developer productivity or quality.

The goal of our Break It JavaFest is to help students learn to write unit tests. Goldwasser [5] introduces the idea of having students submit both code and unit tests for programming assignments, and of running students' unit tests against other students' code. We transform this idea into the context of a JavaFest, where the competition becomes an explicit and interactive activity: We have found that this phase of the Break It! JavaFest triggered valuable discussions in the classroom. Spacco and Pugh [10] advocate educating students in test-driven development. While we did not use a test-first approach in our Break It! JavaFest, one can do so by reordering the test development and implementation phases.

9. CONCLUSIONS

We introduce JavaFests, a new collaborative learning technique. We evaluate that technique in our own undergraduate Java course, and find it to be a useful tool for students to practice programming in a motivating, collaborative environment, and we identify several dimensions along which JavaFests can be improved in the future. We provide material for our JavaFests at `http://www.inf.unisi.ch/faculty/hauswirth/teaching/JavaFest/` for other instructors who would like to adopt them for their own courses.

10. REFERENCES

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

[2] D. J. Barnes and M. Kölling. *Objects First with Java: A Practical Introduction using BlueJ.* Prentice Hall / Pearson Education, 3 edition, 2006.

[3] J. D. Chase and E. G. Okie. Combining cooperative learning and peer instruction in introductory computer science. *SIGCSE Bull.*, 32(1), 2000.

[4] OOPSLA Community. Designfest. http://designfest.acm.org.

[5] M. H. Goldwasser. A gimmick to integrate software testing throughout the curriculum. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 2002.

[6] M. Jazayeri. The education of a software engineer. In *ASE '04: Proceedings of the 19th IEEE international conference on Automated software engineering*, 2004.

[7] M. Kölling, B.Quig, A. Patterson, and J. Rosenberg. The BlueJ system and its pedagogy. *Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology*, 13(4), December 2003.

[8] M. Lanza, A. L. Murphy, R. Robbes, M. Lungu, and P. Bonzini. A teamwork-based approach to programming fundamentals with scheme, smalltalk & java. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, New York, NY, USA, 2008.

[9] N. LeJeune. Critical components for successful collaborative learning in CS1. *J. Comput. Small Coll.*, 19(1), 2003.

[10] J. Spacco and W. Pugh. Helping students appreciate test-driven development (TDD). In *Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, 2006.

[11] A. Turing. Computing machinery and intelligence. *Mind*, LIX(236), October 1950.

[12] W. M. Waite, M. H. Jackson, and A. Diwan. The conversational classroom. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, 2003.

[13] L. Williams. Integrating pair programming into a software development process. In *Proceedings of the Conference on Software Engineering and Training*, 2001.