Algorithms **2010**, *3*, 168-182; doi:10.3390/a3020168

OPEN ACCESS

algorithms

ISSN 1999-4893

www.mdpi.com/journal/algorithms

Article

A Family of Tools for Supporting the Learning of Programming

Guido Rößling

Department of Computer Science, TU Darmstadt, Hochschulstr. 10, D-64289 Darmstadt, Germany;
E-Mail: roessling@acm.org; Tel.: +49-6151-163510; Fax: +49-6151-163052

Received: 25 February 2010; in revised form: 16 March 2010 / Accepted: 22 March 2010 / Published: 15 April 2010

Abstract: Both learning how to program and understanding algorithms or data structures are often difficult. This paper presents three complementary approaches that we employ to help our students in learning to program, especially during the first term of their study. We use a web-based programming task database as an easy and risk-free environment for taking the first steps in programming Java. The Animal algorithm visualization system is used to visualize the dynamic behavior of algorithms and data structures. We complement both approaches with tutorial videos on using the Eclipse IDE. We also report on the experiences with this combined approach.

Keywords: algorithm animation; Animal; programming support; WebTasks

1. Introduction

Programming is a fundamental part of Computer Science. Educators therefore typically expect that graduates of a CS course, or courses close to CS, will be able to program. However, many studies and the experiences of a large number of teachers agree that "programming is not easy", and that learning to program is also not easy, see, e.g., [1].

Some effort has been put into investigating what factors contribute to this difficulty. For example, the problems seem not to depend on gender, but are correlated with problem solving skills and the first language of the student [2].

Over the last two decades, several tools for supporting the process of teaching or learning programming have appeared. Tools like *BlueJ* [3–5], *Alice* [6,7], *DrScheme* [8–10], *Greenfoot* [11–13] or *Academic Java* [14] address programming in their own unique way. For example, while both BlueJ and

Alice target novices in object-oriented programming, BlueJ is based on UML and Java, while Alice uses a built-in drag-and-drop interface to program a 3D world with a high degree of interaction. DrScheme is used for teaching the fundamentals of how to design (good) programs in a set of pedagogically motivated teaching languages based on the functional language Scheme, while Greenfoot offers a framework for two-dimensional grid assignments in Java together with an IDE usable by novice programmers. Academic Java finally provides an interface to a large number of small programming examples that previous research had indicated covered both important and often misunderstood concepts in Java.

On the other hand, there have been a number of approaches to help students better grasp programs by visualizing the program itself ("program visualization"), or to present the dynamic behavior of algorithms of data structures ("algorithm visualization"). These systems include for example *Jeliot 3* [15], which can visualize Java programs stepwise, or *Leonardo* [16], which can execute and also reverse the execution of C programs. The tools for visualizing or animating algorithms and data structures include ANIMAL [17] and JHAVÉ [18].

In this paper, we present a small family of tools that are used for teaching programming, algorithms, and data structures at the TU Darmstadt. These tools consist of a large database with easy to moderately difficult Java programming tasks, a system for visualizing algorithms and data structures, and recordings of tutorials on using the Eclipse IDE. Additionally, we outline our plans to integrate these features into the *Moodle* [19] open source distributed learning environment.

In the following Sections, we will present the current state of these components and outline the integration of the tools and learning materials that we are working on.

2. WebTasks: Programming Tasks Database

Testing the programs written by students for syntactic and semantic correctness, functionality or style is common today. Some of the established systems for accomplishing this include Boss [20], CourseMarker [21], and Web-CAT [22].

In 2006, we were looking for a system that supports students with little to no previous programming experience in learning to solve simple Java programming tasks. At the same time, we wanted to have a large pool of possible tasks covering the spectrum from "extremely easy" to "moderately difficult", in order to address both novice programmers and those with some previous exposure, but perhaps in a different programming language. At this stage, we did not yet expect students to be able to write their own tests, so that the test-driven development approach supported by Web-CAT did not apply to our students. In fact, we do not expect our students be able to write a full-fledged Java class including a *main* method: most tasks only require students to fill in the body of a prepared method.

At the same time, the system was supposed to assist, not assess, students. For example, the results of a student's submission are not taken into account for measuring the same student's accomplishments in the CS 1 course. While we encourage our students to use the system to get some hands-on practice in solving simple Java tasks, their submissions are not monitored and the use of the system is completely voluntary. Plagiarism or "cheating" are thus also not seen as a concern, as the students cannot benefit from this, but only succeed in cheating themselves.

To further lower the barrier in using the system, the system was supposed to run completely in the web browser, without requiring any software installation or even a Java SDK on the students' computers.

For these reasons, the established systems such as Boss [20], CourseMarker [21], and Web-CAT [22] did not perfectly match our expectation. We instead used a competitive programming lab to develop our own solution, called *WebTasks*, based on the best of four competing systems.

WebTasks [23] runs as a set of JSP pages on an Apache Tomcat server. As the contents mainly cover comparatively simple tasks, the target audience typically consists of students attending courses such as CS 1. No special algorithmic knowledge is required by most tasks, only a basic understanding of the Java programming language.

WebTasks currently contains 118 Java programming tasks. Only six of these tasks require the user to upload a complete class file, while the others require only that the body of a prepared method is implemented correctly. In this way, the student's workload is reduced, and many potentially problematic issues, such as the correct use of the Java `main` method, can be avoided. This also makes using WebTasks significantly easier for novices to Java, while at the same time improving the control we have over the submissions.

Figure 1 shows a typical example of a programming task. Note that parts of this web page are in German, as the system was developed to support our local students; however, some tasks including the one shown here have been translated into English. Additionally, an internationalization of the user interface would not be difficult to do. The task shown requires the student to implement a variation on the Pascal triangle, where two initial values are passed in as an array of size 2. The student has to compute the n^{th} step of a Pascal triangle based on this initial input, and return it as an array of int values.

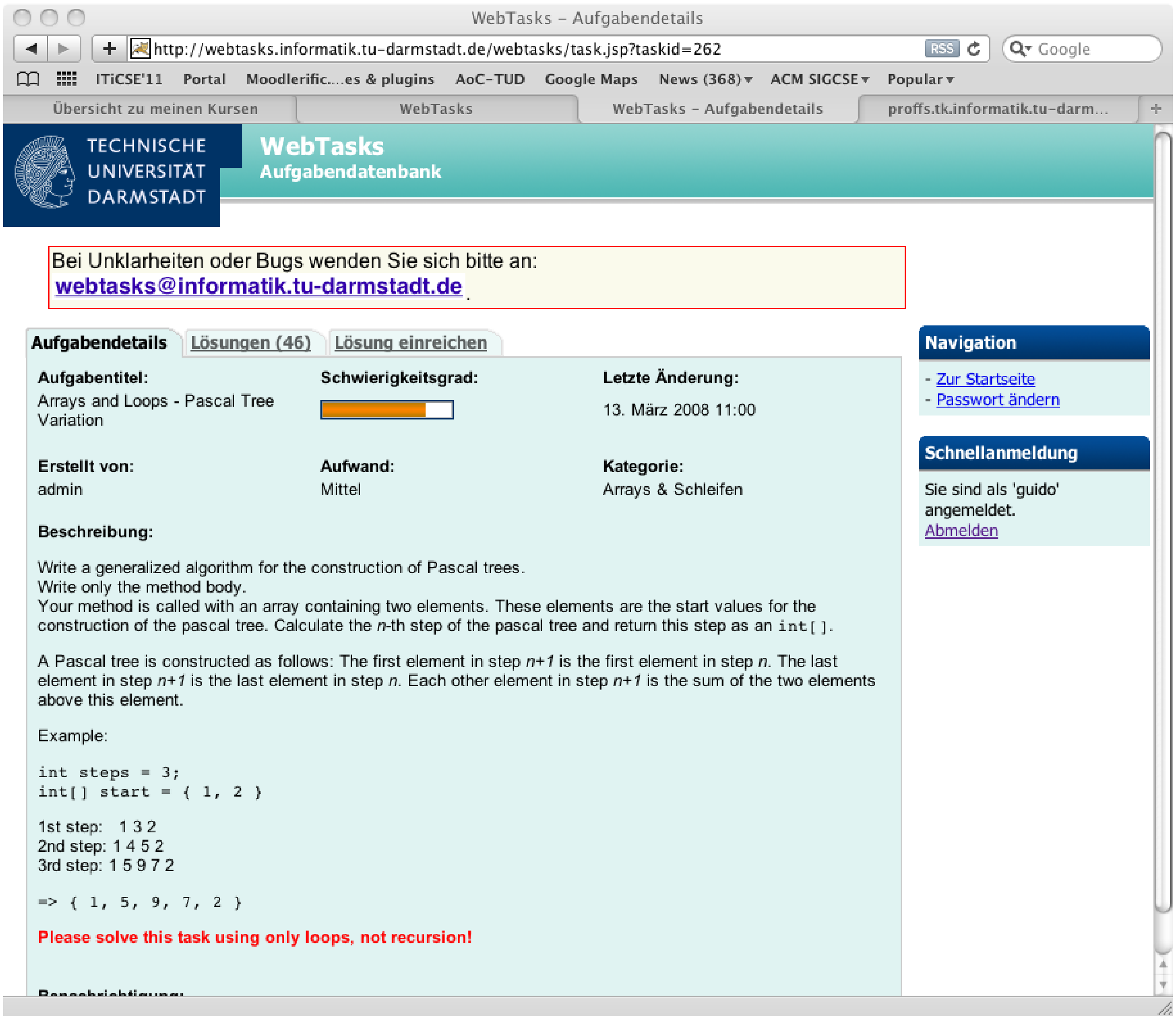
The difficulty level of the task is indicated by a colored difficulty bar. The task in Figure 1 is thus ranked as rather difficult, based on the assumed programming skills of an average student in the first year of CS.

Figure 2 presents the basic work flow for a student working with WebTasks. The student will first pick an assignment and submit a solution proposal. This will be compiled by the server, and, if the compilation was successful, it will automatically be tested for correctness. A correct solution will be submitted to the internal forum. We will now take a closer look at these steps.

If the student decides on implementing the task, he will be taken to an input field as shown in Figure 3. Here, the student sees the predefined method header and `return` statement, and only needs to provide the correct implementation of the method body. In the Figure, a (slightly incorrect) solution proposal has already been filled in.

Once students think that their code should solve the task, they can press the "Abschicken" (Submit) button to submit their solution. The solution will then be automatically validated using JUnit [24,25]. If the tests are not successful, the result will be shown to the student, as indicated in Figure 4. Here, the system informs the user that test 2 out of 7 has failed, because the last value in the result was 2, not 1. Note that the tests are aborted once one test has failed. Thus, in Figure 4, only the tests *Test1* and *Test2* were executed. This was done to avoid overwhelming programming novices with a large number of failing tests in the initial submissions: novices shall be able to tackle one problem after the other.

Figure 1. Example task from the WebTasks database.


WebTasks – Aufgabendetails
http://webtasks.informatik.tu-darmstadt.de/webtasks/task.jsp?taskid=262
Google
ITiCSE'11 Portal Moodlerific....es & plugins AoC-TUD Google Maps News (368) ACM SIGCSE Popular
Übersicht zu meinen Kursen WebTasks WebTasks – Aufgabendetails proffs.tk.informatik.tu-darm...
TECHNISCHE UNIVERSITÄT DARMSTADT
WebTasks
Aufgabendatenbank
Aufgabendetails Lösungen (46) Lösung einreichen
Navigation
- Zur Startseite
- Passwort ändern
Schnellanmeldung
Abmelden

Bei Unklarheiten oder Bugs wenden Sie sich bitte an:
webtasks@informatik.tu-darmstadt.de

Aufgabentitel: Arrays and Loops - Pascal Tree Variation	**Schwierigkeitsgrad:**	**Letzte Änderung:** 13. März 2008 11:00
Erstellt von: admin	**Aufwand:** Mittel	**Kategorie:** Arrays & Schleifen

Beschreibung:

Write a generalized algorithm for the construction of Pascal trees.
Write only the method body.
Your method is called with an array containing two elements. These elements are the start values for the construction of the pascal tree. Calculate the *n*-th step of the pascal tree and return this step as an `int[]`.

A Pascal tree is constructed as follows: The first element in step *n+1* is the first element in step *n*. The last element in step *n+1* is the last element in step *n*. Each other element in step *n+1* is the sum of the two elements above this element.

Example:

```
int steps = 3;
int[] start = { 1, 2 }
```

1st step: 1 3 2
2nd step: 1 4 5 2
3rd step: 1 5 9 7 2

```
=> { 1, 5, 9, 7, 2 }
```

Please solve this task using only loops, not recursion!

Sie sind als 'guido' angemeldet.

The most popular programming assignment asks students to determine the average value for an array of int values. This task currently has 627 valid solutions. 617 students solved a second tasks that asked them to "append" an int to an array of int values, requiring them to create a new array of the proper size and copy all "old" values accordingly. In total, the 118 programming tasks have received more than 10,000 solutions so far. Note that this number ignores the failed submissions—only successful submissions that passed all internal JUnit tests are counted.

Once a task has been solved by a student, he or she is able to view the solutions of other students, and may comment on them. The underlying rationale for this feature is that students may learn much from peer solutions, especially if those use clever "tricks" that the student had not thought of before. These "tricks" can for example include the use of `System.arraycopy` to copy the elements in an array, the use of iterators or the modified "for each" version of the Java `for` loop.

Figure 2. Schematic workflow for a student submission to WebTasks.

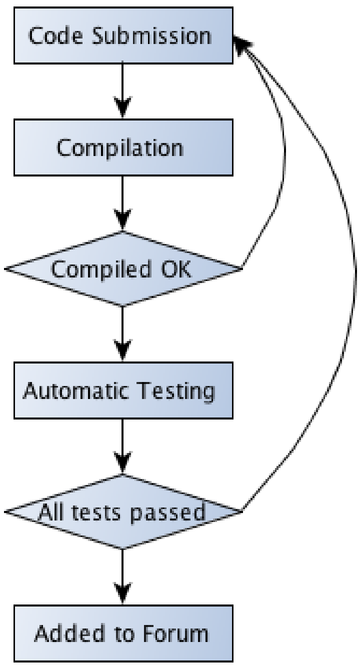
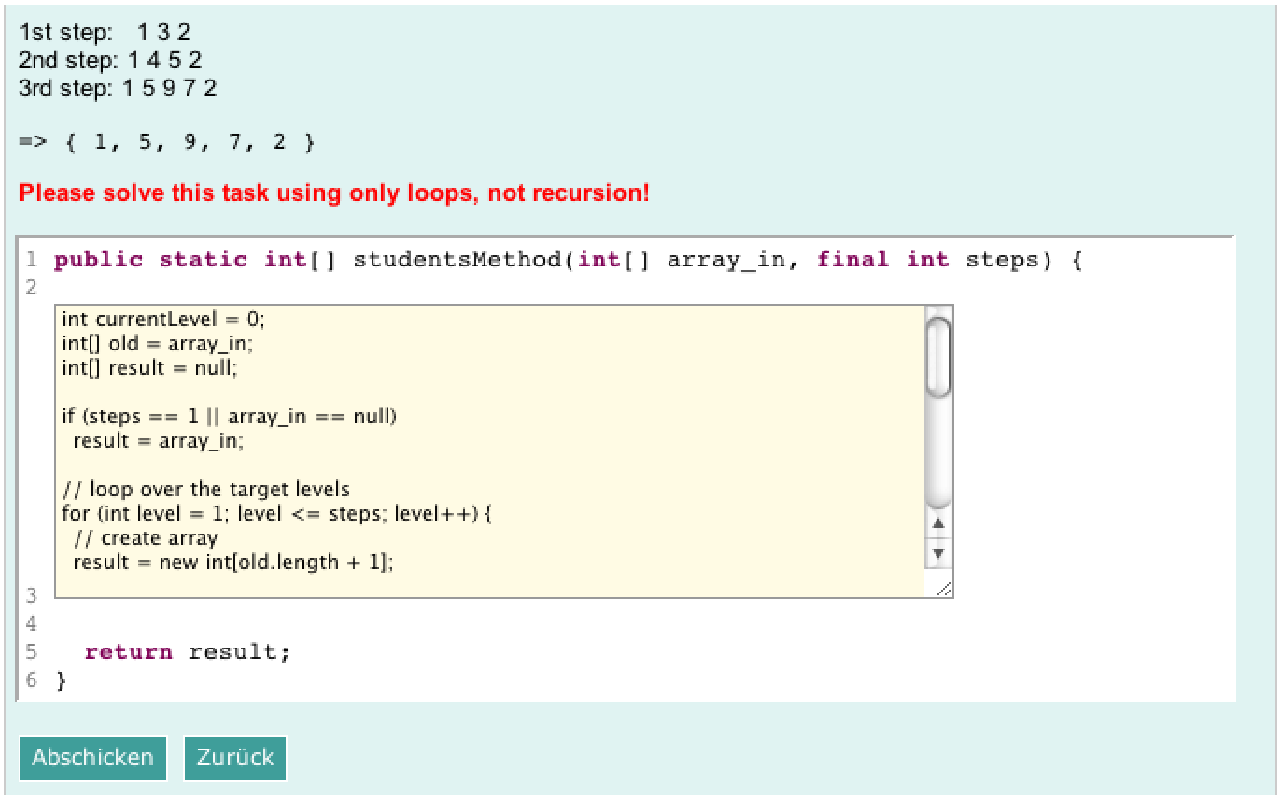


Figure 3. Example student code for a given WebTasks task.



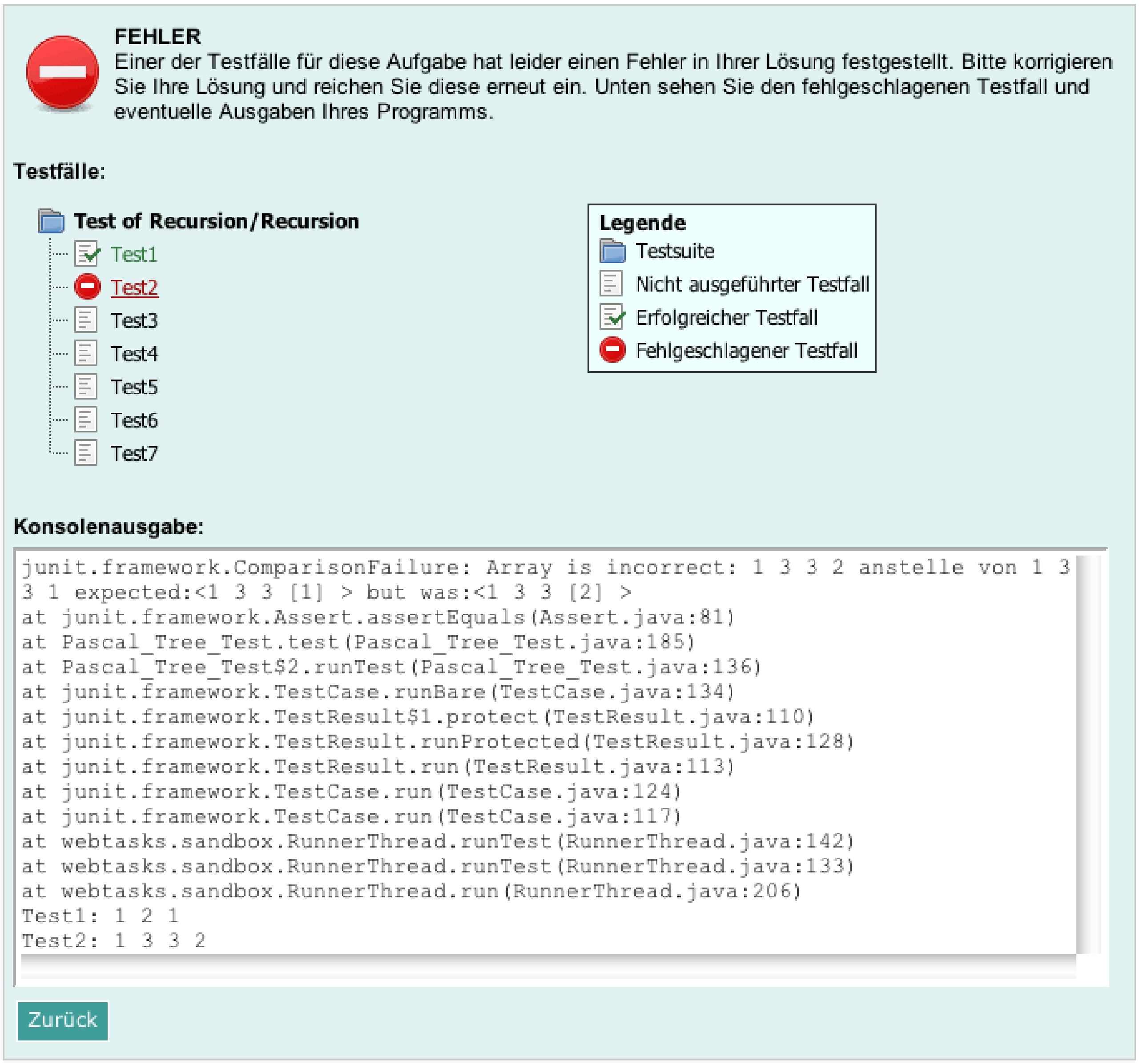
1st step: 1 3 2
2nd step: 1 4 5 2
3rd step: 1 5 9 7 2

```
=> { 1, 5, 9, 7, 2 }
```

Please solve this task using only loops, not recursion!

```
public static int[] studentsMethod(int[] array_in, final int steps) {

  int currentLevel = 0;
  int[] old = array_in;
  int[] result = null;

  if (steps == 1 || array_in == null)
    result = array_in;

  // loop over the target levels
  for (int level = 1; level <= steps; level++) {
    // create array
    result = new int[old.length + 1];


  return result;
}
```

Figure 4. Example validation failure output for code submitted to the WebTasks database.



FEHLER
Einer der Testfälle für diese Aufgabe hat leider einen Fehler in Ihrer Lösung festgestellt. Bitte korrigieren Sie Ihre Lösung und reichen Sie diese erneut ein. Unten sehen Sie den fehlgeschlagenen Testfall und eventuelle Ausgaben Ihres Programms.

Testfälle:

Test of Recursion/Recursion
- ✓ Test1
- ● Test2
- Test3
- Test4
- Test5
- Test6
- Test7

Legende

🗀	Testsuite
▭	Nicht ausgeführter Testfall
✓	Erfolgreicher Testfall
●	Fehlgeschlagener Testfall

Konsolenausgabe:

```
junit.framework.ComparisonFailure: Array is incorrect: 1 3 3 2 anstelle von 1 3
3 1 expected:<1 3 3 [1] > but was:<1 3 3 [2] >
at junit.framework.Assert.assertEquals(Assert.java:81)
at Pascal_Tree_Test.test(Pascal_Tree_Test.java:185)
at Pascal_Tree_Test$2.runTest(Pascal_Tree_Test.java:136)
at junit.framework.TestCase.runBare(TestCase.java:134)
at junit.framework.TestResult$1.protect(TestResult.java:110)
at junit.framework.TestResult.runProtected(TestResult.java:128)
at junit.framework.TestResult.run(TestResult.java:113)
at junit.framework.TestCase.run(TestCase.java:124)
at junit.framework.TestCase.run(TestCase.java:117)
at webtasks.sandbox.RunnerThread.runTest(RunnerThread.java:142)
at webtasks.sandbox.RunnerThread.runTest(RunnerThread.java:133)
at webtasks.sandbox.RunnerThread.run(RunnerThread.java:206)
Test1: 1 2 1
Test2: 1 3 3 2
```

To use the system, a user first has to log in with the credentials of the CS Department of the TU Darmstadt, or register inside the system. While the first option is only available to students who take courses from our department, the system-internal registration is available to all interested users. The use of the system is encouraged within our CS 1 course as an easy way to get fast feedback on the correctness of programs. Compared to regularly submitted programming tasks which are corrected by a teaching assistant, the delay between submission and access to the result when using WebTasks is only a few seconds: the time needed for the server to compile and test the submission and then provide the output of the compiler and test system.

The database also contains 50 multiple choice tests. The topics covered in the tests include questions about various tree structures (Heaps, spanning trees, B- and AVL trees), complexity, maximum flow problems, hashing, shortest path problems, boolean and binary operators and sorting algorithms. All these topics belong to the first two *Introduction to Computer Science* courses taught at the TU Darmstadt, the core courses for teaching programming concepts, programming languages, object orientation, algorithms and data structures.

It should be pointed out that the system as it is is not meant for official grading of student submissions. Although students have to log in, no authentication beyond the login and password is done, so that no valid proof of identity can be guaranteed. WebTasks also offers an "exam" mode in which the access to the system is limited to the computers from our central CS computer pool (with about 100 terminals). Additionally, the pool network is manually reconfigured so that each terminal can only access the WebTasks server, effectively preventing access to the home directory (as this might contain solutions), other computers inside the pool, or the Internet. Thus, given that the students in the pool will also be supervised by at least one person, we can be reasonably sure that the solution submitted by a given student will actually come from that student.

The CS Department also considered using the system to grade students on their programming skills. The idea was to assign a number of "exam" dates for which students could register. Each such date would present them with three to five tasks chosen either randomly or by the educator, to be solved within two hours. As all tasks in the database are essentially easy, one would expect that all students with a certain basic skill in programming and a basic understanding of Java would be able to solve this task. WebTasks would have been able to support this testing in a special testing mode.

However, after some internal discussions, the CS Department dropped these considerations in favor of a mandatory CS 1 programming project and a mentoring system. This was done to place more emphasis on supporting students than on using exams to distinguish between "sufficiently qualified" students and those who were not allowed to continue. As the goal of WebTasks is to assist, not monitor or evaluate, students in taking the first steps in programming Java in a threat-free environment, we were glad that the plans were ultimately dropped.

We initially also included an automatic and rigorous checking of the code quality with the use of *CheckStyle* [26]. Since our target users are programming novices who already felt challenged with the tasks, we decided not to follow up on this.

3. Visualizing and Animating Algorithms and Data Structures

The WebTasks database described in Section 2 is meant to provide students with little programming experience in Java with an easy and fault-tolerant environment for working on simple programming tasks. Another problematic area in programming is understanding common algorithms and data structures. Topics such as searching and sorting algorithms are usually taught in a CS 2 course, together with data structures such as binary trees, graphs, or more complex data structures such as AVL trees.

The main problem here is often not that the students have to program these algorithms or data structures. Rather, the students first have to be able to understand the behavior of these structures. For this purpose, the field of algorithm and program visualization has been active in implementing tools and providing guidelines for supporting the understanding of algorithms and data structures.

Our basic credo is that the highly dynamic nature of the underlying data structures or algorithms should also be presented dynamically, as getting a firm grasp of the "inner workings" based solely on the (static) code is difficult for many students. Besides supporting students by showing them how a given data structure or algorithm behaves, it has increasingly been recognized that it is also important to actively engage the learners. A seminal report by a set of experts in the field [27] has defined a taxonomy of "levels of engagement", which has been referenced in more than 100 publications and has caused some follow-up research with similar results including [18,28]. The report was further supported by a survey of successful evaluations [29].

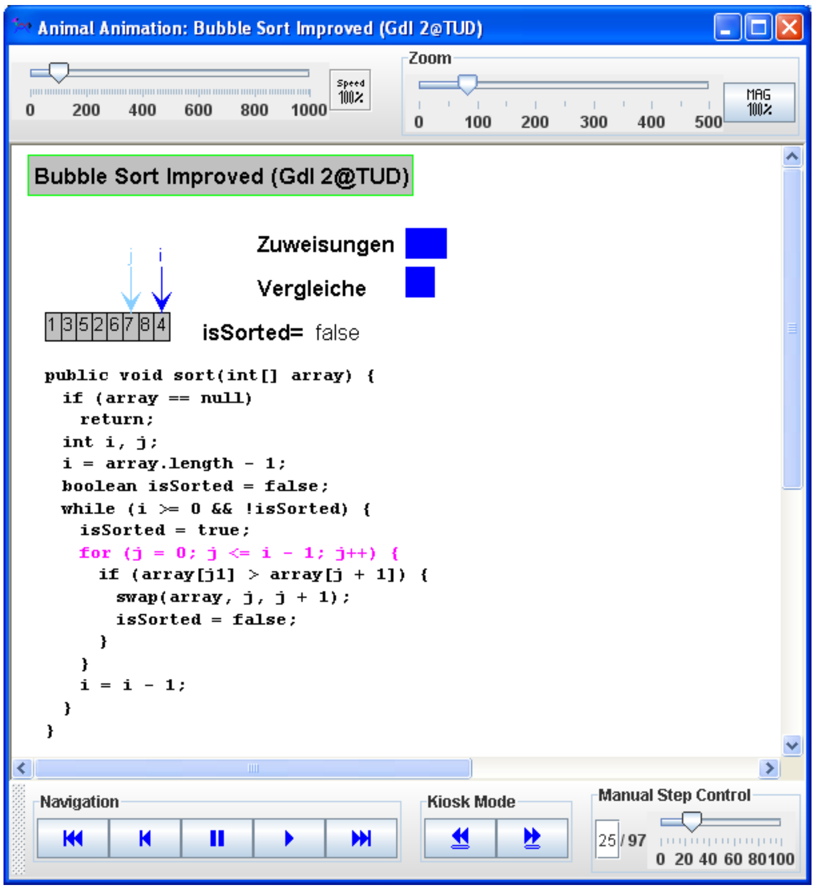
One of the most elaborate systems for visualizing (almost) arbitrary content is ANIMAL [17,30]. ANIMAL presents contents based on a set of built-in flexible graphical primitives and animation effects. It does not "understand" the underlying "code", but usually processes programs written in the ANIMALSCRIPT visualization scripting language [31]. This notation allows the placement of objects in relation to each other, e.g., at an offset from another object's top left corner. It also supports a number of computer science-specific data structures, such as arrays, lists or code blocks including indentation and highlighting.

What sets ANIMAL apart from other related systems is the flexibility of providing content to the user. The presentation controls allow easy access in both directions and optionally also offer a "table of contents" for each animation—provided that the animation author has populated this. More importantly, there are several ways how animation content can be generated:

- Even novices can generate contents manually using drag and drop. While this may initially seem to be almost as time-consuming as doing it in programs such as PowerPoint, the support for data structures makes the process faster. Additionally, since each animation frame builds on the previous frame, the user does not have to copy all current elements to the next slide, as would be needed in presentation software such as PowerPoint or Keynote.
- Content can be written directly in ANIMALSCRIPT. ANIMAL provides an online reference to the language, so that writing a syntactically correct program is not difficult.
- Animations can be generated from a program by adding appropriate statements that will create ANIMALSCRIPT output at "interesting places".
- Programs can also be enriched by using the *AlgoAnim API*. This API will also create ANIMALSCRIPT output, but provides a far cleaner interface than incorporating the associated output statements into existing code [32].
- By placing output-producing programs into the *generator framework*, the associated programs can be run directly inside ANIMAL [33]. Additionally, this allows the end-user to adjust parameters, such as the concrete values to be sorted or inserted, as well as visual properties, such as the color chosen for individual elements.

Figure 5 shows an example screenshot of ANIMAL visualizing a version of Bubble Sort. The animation shows the array and the current values for the array index variables i and j. It also includes the complete source code with indentation below the array. The currently executing code line, in this case the inner "for" loop, is highlighted to make it easier for the user to link the code with the visualized contents. Additionally, the value for the boolean variable "isSorted" is shown and the number of assignments and comparisons ("Zuweisungen" and "Vergleiche" in German, respectively) are visualized at the top.

Figure 5. Example screenshot from the ANIMAL user front-end animating BubbleSort.



```
public void sort(int[] array) {
  if (array == null)
    return;
  int i, j;
  i = array.length - 1;
  boolean isSorted = false;
  while (i >= 0 && !isSorted) {
    isSorted = true;
    for (j = 0; j <= i - 1; j++) {
      if (array[j1] > array[j + 1]) {
        swap(array, j, j + 1);
        isSorted = false;
      }
    }
    i = i - 1;
  }
}
```

The user can navigate the animation using the basic navigation controls at the bottom left. Note that this also includes jumping to the start of the animation or taking one step backwards. ANIMAL places no limit on the number of steps to be taken backwards; users could easily jump to the end of the animation and then walk through it backwards if they wanted to do so.

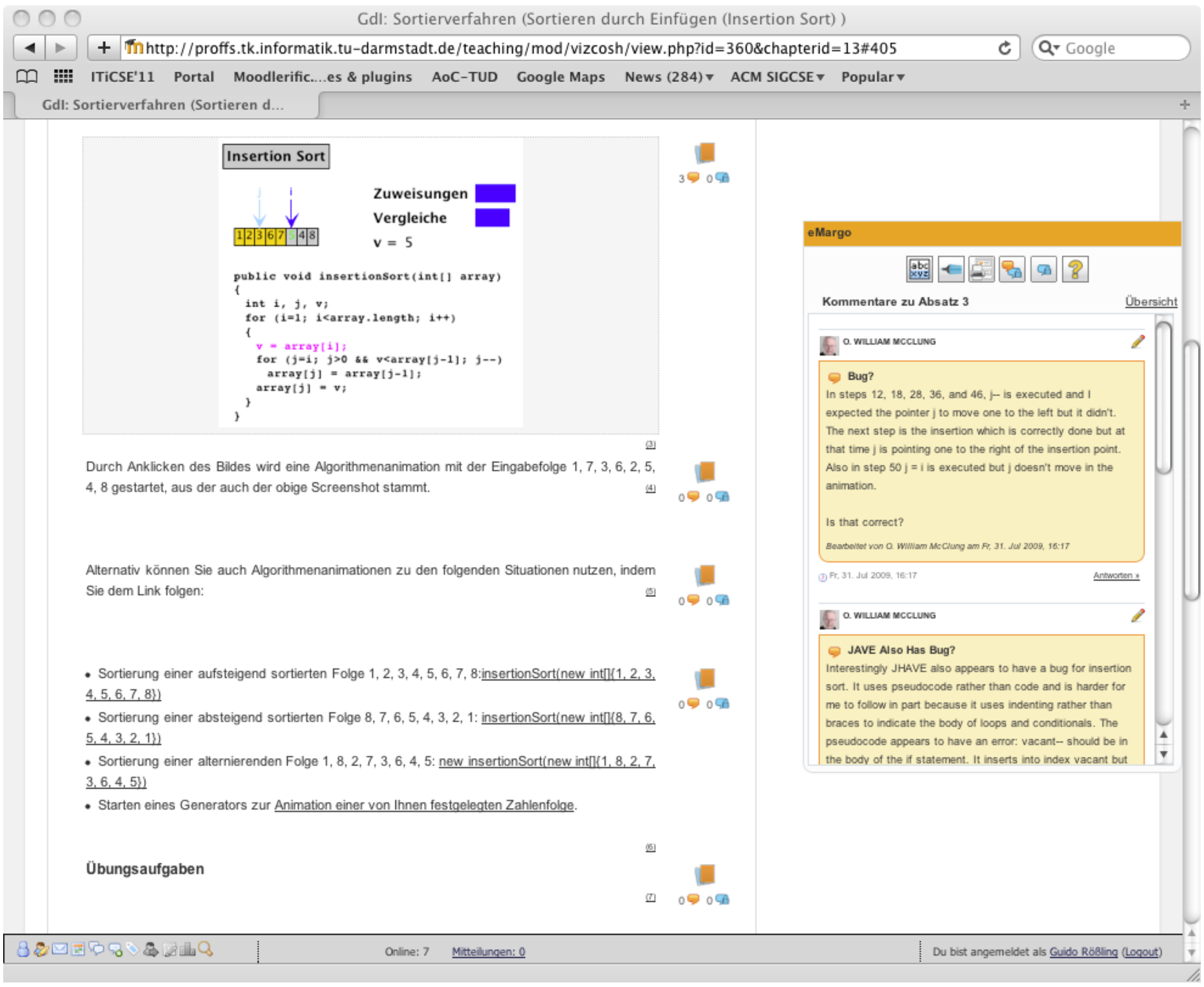
The "kiosk mode" buttons put the animation on continuous play until the end, or in reverse mode the beginning, of the animation has been reached. Users can also type in the number of the step they want to jump to, or drag the ruler to some point in the animation. Note that the ruler is always normalized to the percentage of the animation shown, and thus will always stay between 0 and 100 (for 100%). Dragging the ruler from left to right gives a "fast fly-by overview" of the animation.

One key aspect for the acceptance of algorithm visualization in a course is the integration of the visualization with the other lecture materials [34,35]. The report of an international expert group recommended what they termed as a *visualization-based computer science hypertextbook (VizCoSH)*, which would have a structure similar to a textbook but would tightly integrate the teaching and learning materials with algorithm or program visualizations [34].

We have developed a VizCoSH prototype as an activity for the Moodle distributed learning environment [19]. This activity emulates a regular "textbook", separated into chapters and paragraphs. However, it also provides several additional features. For example, each paragraph can be discussed in a separate "thread" [35].

Figure 6 shows a screenshot of this activity. The top of the screen shows a screenshot of the animation. One click on this image will lead to an information page about the animation, and a second click will cause the ANIMAL system to start up, load the animation from the web server, and display to the end user. Below, links to other animations are included, illustrating the way the algorithm–here, Insertion Sort—will behave on data placed in ascending, descending or alternating order. Additionally, users can start the built-in generator to generate an animation portraying how "their" custom input data will be sorted using Insertion Sort. The icons to the right of the main text provide access to the discussion about the current paragraph. The 3 next to the orange bubble indicates that three comments have been posted about this paragraph (visible to the right), while the 0 next to the blue icon shows that there are no private posts for this element.

Figure 6. Example screenshot of a hypertextbook with an included ANIMAL animation.


GdI: Sortierverfahren (Sortieren durch Einfügen (Insertion Sort))
http://proffs.tk.informatik.tu-darmstadt.de/teaching/mod/vizcosh/view.php?id=360&chapterid=13#405
Google
ITiCSE'11
Portal
Moodlerific....es & plugins
AoC-TUD
Google Maps
News (284)
ACM SIGCSE
Popular
GdI: Sortierverfahren (Sortieren d...
Insertion Sort
j
i
1 2 3 6 7 5 4 8
Zuweisungen
Vergleiche
v = 5
3 0
eMargo
abc xyz
?

```
public void insertionSort(int[] array)
{
  int i, j, v;
  for (i=1; i<array.length; i++)
  {
    v = array[i];
    for (j=i; j>0 && v<array[j-1]; j--)
      array[j] = array[j-1];
    array[j] = v;
  }
}
```

[3]

Durch Anklicken des Bildes wird eine Algorithmenanimation mit der Eingabefolge 1, 7, 3, 6, 2, 5, 4, 8 gestartet, aus der auch der obige Screenshot stammt. [4] 0 0

Alternativ können Sie auch Algorithmenanimationen zu den folgenden Situationen nutzen, indem Sie dem Link folgen: [5] 0 0

- Sortierung einer aufsteigend sortierten Folge 1, 2, 3, 4, 5, 6, 7, 8: insertionSort(new int[]{1, 2, 3, 4, 5, 6, 7, 8})
- Sortierung einer absteigend sortierten Folge 8, 7, 6, 5, 4, 3, 2, 1: insertionSort(new int[]{8, 7, 6, 5, 4, 3, 2, 1})
- Sortierung einer alternierenden Folge 1, 8, 2, 7, 3, 6, 4, 5: new insertionSort(new int[]{1, 8, 2, 7, 3, 6, 4, 5})
- Starten eines Generators zur Animation einer von Ihnen festgelegten Zahlenfolge.

0 0

[6]

Übungsaufgaben

[7] 0 0

Kommentare zu Absatz 3 Übersicht

O. WILLIAM MCCLUNG

Bug?
In steps 12, 18, 28, 36, and 46, j-- is executed and I expected the pointer j to move one to the left but it didn't. The next step is the insertion which is correctly done but at that time j is pointing one to the right of the insertion point. Also in step 50 j = i is executed but j doesn't move in the animation.

Is that correct?

Bearbeitet von O. William McClung am Fr, 31. Jul 2009, 16:17

Fr, 31. Jul 2009, 16:17 Antworten »

O. WILLIAM MCCLUNG

JAVE Also Has Bug?
Interestingly JHAVE also appears to have a bug for insertion sort. It uses pseudocode rather than code and is harder for me to follow in part because it uses indenting rather than braces to indicate the body of loops and conditionals. The pseudocode appears to have an error: vacant-- should be in the body of the if statement. It inserts into index vacant but

Online: 7 Mitteilungen: 0 Du bist angemeldet als Guido Rößling (Logout)

In addition to commenting on paragraphs, users can also use a text marker, included above the comment threads on the right, to highlight individual words or passages in the text. The navigation between chapters is accomplished using the table of contents shown at the top of each page (missing on the screenshot), as well as the pair of back/forward buttons at the top and bottom of each page. Finally, users can print the complete "book" or a single chapter comfortably.

The large benefit of the integration into Moodle is that the animations of algorithms and data structures can now be placed together with the other learning materials, such as slides, exercise sheets, or homework submission and correction facilities. All these activities are placed in the same Moodle course, allowing students to access them whenever needed without having to "switch context".

4. Tutorials for Teaching the Use of Eclipse

To further enable our students to work with the comfortable but also complex *Eclipse* Java IDE, we have produced a small number of (German) screen recordings as tutorials. The videos were produced using the excellent *Camtasia* [36] software, partially under Windows (Camtasia Studio 6) and under Mac OS X (Camtasia 1.0.1).

Since the software is very easy to use and records the complete desktop or a part thereof, we first wrote a script of things we wanted to show. We defined five videos to be provided as tutorials:

Eclipse Basics presents the basics of using Eclipse: starting the IDE, explaining the initially visible icons, and taking a walk through the "Hello World" tutorial. During the programming, we have intentionally introduced a bug, in order to illustrate how Eclipse will highlight errors and how they can be fixed using the "Quick Fix" option.

Implementing a new class shows the implementation of a class, based on a basic piece of code. We show how code can be indented and formatted, and how mistakes can be addressed (in this case, by renaming a mistyped variable). We also implement the `toString()` method and use code completion as well as the generation of get- and set-methods including comments. The use of the code outline and the "Problems" and "JavaDoc" view is explained. The tutorial ends by discussing the look-up of definitions using the F3 key, using quick fixes, formatting source code, adding and organizing imports, and sorting members.

Implementing and testing a sample calculator presents a simple implementation of a basic calculator that is developed "live". The tutorial starts by creating a new project, package, and class, and implementing several simple calculator methods, complete with some intentional bugs. We then show how a JUnit test can be implemented in a separate package, and how test methods are annotated with the `@Test` notation. The development of tests is done in parallel with re-runs of the existing and new tests. We then add an initialization method and show how methods not yet ready for testing can be ignored, how expected exceptions can be checked, and how infinite loops can be prevented during testing.

Extending classes illustrates how a new class can be defined to extend an existing class. We show one example each using an abstract and a concrete base class. We then repeat this by having a class implement an interface.

Debugging is illustrated in a separate video. Here, we use JUnit tests to pinpoint the offending method and code line, and then use breakpoints to narrow down the possible lines of code. The tutorial also shows how variables can be inspected and how (changed) method code can be re-run.

5. Summary and Conclusions

We have presented our tools for assisting students to learn programming Java. These consist of a web-based database of easy to moderately difficult programming tasks, a system for visualizing the behavior of algorithm and data structures, and tutorials on how to use the Eclipse IDE.

Our *WebTasks* system [23] has been in use since 2006. The system provides a set of programming tasks that mostly require the student to provide a method body. In this way, we can prevent a number of typical problems that novices to Java face, such as the correct use of the `main` method. Submissions are tested on upload, providing feedback within a few seconds of submitting a solution. Students who have solved a given task are granted access to all solutions for this task, so that they can look at what their peers did. They can also comment on other students' solutions. The system has seen much use since its inception, with currently more than 10,000 (correct) submitted solutions to the more than 100 programming tasks.

To make understanding the dynamic behavior of algorithms and data structures easier, we provide the ANIMAL system [17,30]. ANIMAL offers full reversibility of the contents and thus makes following the display easier. ANIMAL animations typically also show the underlying source code and highlight the currently executing line, making it easier to connect the code with the visualized representation.

Finally, the difficulties that especially novice programmers face when working with a full-fledged IDE like Eclipse are addressed by tutorial videos that illustrate the use of Eclipse.

Our experiences with the tool support so far are very encouraging. Student feedback for the CS 1 lecture in which the tools are used has been very good, including winning the prize for the best lecture in the summer term 2009 based on student evaluation results. Students also stated that using the tools had been a great help in getting ready to program Java and understand the presented algorithms. However, some students also stated that they would have preferred to have a single base system, rather than a set of independent elements.

To provide better access to our materials, we are working on integrating both the WebTasks database, the algorithm visualizations and the tutorial materials into the Moodle Learning Management System [19]. This will allow students to use a single login to access all course materials, exercise their programming skills, and watch and interact with visualizations of algorithms and data structures. Currently, the visualization of ANIMAL content has already been implemented, and we are working on integrating the WebTasks database into Moodle. Once this integration has been accomplished, we will be glad to share the materials with interested educators.

Acknowledgements

We gratefully acknowledge the assistance of our students who implemented parts of the system described in this paper, in alphabetical order: Tobias Ackermann, Sebastian Hartte, Stephan Mehlhase, Jens Pfau, Anselmo Stelzer, and Teena Vellaramkalayil.

References

1. McCracken, M.; Almstrum, V.; Diaz, D.; Guzdial, M.; Hagan, D.; Kolikant, Y.B.D.; Laxer, C.; Thomas, L.; Utting, I.; Wilusz, T. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bull.* **2001**, *33*, 125–180.
2. Pillay, N.; Jugoo, V.R. An investigation into student characteristics affecting novice programming performance. *SIGCSE Bull.* **2005**, *37*, 107–110.
3. Kouznetsova, S. Using BlueJ and Blackjack to teach object-oriented design concepts in CS1. *J. Comput. Small Coll.* **2007**, *22*, 49–55.
4. Kölling, M., Using BlueJ to introduce programming. In *Reflections on the Teaching of Programming: Methods and Implementations*; Springer-Verlag: Berlin, Heidelberg, Germany, 2008; pp. 98–115.
5. Barnes, D.J.; Kölling, M. *Objects First with Java: A Practical Introduction Using BlueJ*, 4th ed.; Prentice Hall: Upper Saddle River, NJ, USA, 2008.
6. Dann, W.P.; Cooper, S.; Pausch, R. *Learning to Program with Alice*; Prentice Hall: Upper Saddle River, NJ, USA, 2006.
7. Rodger, S.H.; Hayes, J.; Lezin, G.; Qin, H.; Nelson, D.; Tucker, R.; Lopez, M.; Cooper, S.; Dann, W.; Slater, D. Engaging middle school teachers and students with Alice in a diverse set of subjects. In *SIGCSE '09: Proceedings of the 40th ACM Technical Symposium on Computer Science Education*; ACM: New York, NY, USA, 2009; pp. 271–275.
8. Page, R.; Eastlund, C.; Felleisen, M. Functional programming and theorem proving for undergraduates: a progress report. In *FDPE '08: Proceedings of the 2008 International Workshop on Functional and Declarative Programming in Education*; ACM: New York, NY, USA, 2008; pp. 21–30.
9. Bieniusa, A.; Degen, M.; Heidegger, P.; Thiemann, P.; Wehr, S.; Gasbichler, M.; Sperber, M.; Crestani, M.; Klaeren, H.; Knauel, E. HtDP and DMDA in the battlefield: A case study in first-year programming instruction. In *FDPE '08: Proceedings of the 2008 International Workshop on Functional and Declarative Programming in Education*; ACM: New York, NY, USA, 2008; pp. 1–12.
10. Felleisen, M.; Findler, R.B.; Flatt, M.; Krishnamurthi, S. *How to Design Programs—An Introduction to Programming and Computing*; MIT Press: Cambridge, MA, USA, 2001.
11. Gallant, R.J.; Mahmoud, Q.H. Using Greenfoot and a Moon Scenario to teach Java programming in CS1. In *ACM-SE 46: Proceedings of the 46th Annual Southeast Regional Conference*; ACM: New York, NY, USA, 2008; pp. 118–121.
12. Al-Bow, M.; Austin, D.; Edgington, J.; Fajardo, R.; Fishburn, J.; Lara, C.; Leutenegger, S.; Meyer, S. Using Greenfoot and games to teach rising 9th and 10th grade novice programmers. In *Sandbox '08: Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games*; ACM: New York, NY, USA, 2008; pp. 55–59.
13. Kölling, M. Greenfoot: a highly graphical IDE for learning object-oriented programming. In *ITiCSE '08: Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*; ACM: New York, NY, USA, 2008; pp. 327–327.

14. Academic Java. Availible online: http://academicjava.com/ (accessed on 25 March 2010).
15. Moreno, A.; Myller, N.; Sutinen, E.; Ben-Ari, M. Visualizing Programs with Jeliot 3. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI 2004)*; ACM Press: New York, NY, USA, 2004; pp. 373–380.
16. Demetrescu, C.; Finocchi, I. A portable virtual machine for program debugging and directing. In *SAC '04: Proceedings of the 2004 ACM Symposium on Applied Computing*; ACM: New York, NY, USA, 2004; pp. 1524–1530.
17. Rößling, G.; Freisleben, B. ANIMAL: A System for Supporting Multiple Roles in Algorithm Animation. *J. Visual Lang. Computing* **2002**, *13*, 341–354.
18. Naps, T. JHAVÉ – Addressing the Need to Support Algorithm Visualization with Tools for Active Engagement. *IEEE Comput. Graph. Appl.* **2005**, *25*, 49–55.
19. Rice IV, W.H.; Nash, S.S. *Moodle 1.9 Teaching Techniques - Creative ways to build powerful and effective online courses*; Packt Publishing: Birmingham, UK, 2010.
20. Joy, M.; Griffiths, N.; Boyatt, R. The Boss online submission and assessment system. *J. Educ. Resour. Comput.* **2005**, *5*, 2.
21. Higgins, C.A.; Gray, G.; Symeonidis, P.; Tsintsifas, A. Automated assessment and experiences of teaching programming. *J. Educ. Resour. Comput.* **2005**, *5*, 5.
22. Edwards, S.H.; Pérez-Quiñones, M.A. Experiences using test-driven development with an automated grader. *J. Comput. Small Coll.* **2007**, *22*, 44–50.
23. Rößling, G.; Hartte, S. WebTasks: Online Programming Exercises Made Easy. In *Proceedings of the 13th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 2008)*; ACM Press: New York, NY, USA, 2008; p. 363.
24. Wick, M.; Stevenson, D.; Wagner, P. Using testing and JUnit across the curriculum. In *SIGCSE '05: Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*; ACM: New York, NY, USA, 2005; pp. 236–240.
25. Object Mentor. JUnit.org Resources for Test Driven Development. Availible online: http://www.junit.org (accessed on 25 March 2010).
26. Burn, O. Checkstyle 5.0. Availible online: http://checkstyle.sourceforge.net/ (accessed on 25 March 2010).
27. Naps, T.L.; Rößling, G.; Almstrum, V.; Dann, W.; Fleischer, R.; Hundhausen, C.; Korhonen, A.; Malmi, L.; McNally, M.; Rodger, S.; Velázquez-Iturbide, J.Á. Exploring the Role of Visualization and Engagement in Computer Science Education. *ACM SIGCSE Bullet.* **2003**, *35*, 131–152.
28. Grissom, S.; McNally, M.; Naps, T.L. Algorithm Visualization in Computer Science Education: Comparing Levels of Student Engagement. In *Proceedings of the First ACM Symposium on Software Visualization*; ACM Press: New York, NY, USA, 2003; pp. 87–94.
29. Urquiza-Fuentes, J.; Velázquez-Iturbide, J.A. A survey of successful evaluations of program visualization and algorithm animation systems. *Trans. Comput. Educ.* **2009**, *9*, 1–21.
30. Rößling, G. *Animal-Farm: An Extensible Framework for Algorithm Visualization*; VDM Verlag Dr. Müller: Saarbrücken, Germany, 2008.

31. Rößling, G.; Gliesche, F.; Jajeh, T.; Widjaja, T. Enhanced Expressiveness in Scripting Using ANIMALSCRIPT V2. In Proceedings of the Third Program Visualization Workshop, Warwick, UK, July 2004; pp. 15–19.
32. Rößling, G.; Mehlhase, S.; Pfau, J. A Java API for Creating (not only) ANIMALSCRIPT. *Electron. Note Theor. Comput. Sci.* **2009**, *224*, 15–25.
33. Rößling, G.; Ackermann, T. A Framework for Generating AV Content on-the-fly. *Electr. Note Theor. Comput. Sci.* **2007**, *178*, 23–31.
34. Rößling, G.; Naps, T.; Hall, M.S.; Karavirta, V.; Kerren, A.; Leska, C.; Moreno, A.; Oechsle, R.; Rodger, S.H.; Urquiza-Fuentes, J.; Velázquez-Iturbide, J.Á. Merging Interactive Visualizations with Hypertextbooks and Course Management. *SIGCSE Bullet. inroad* **2006**, *38*, 166–181.
35. Rößling, G.; Vellaramkalayil, T. A Visualization-Based Computer Science Hypertextbook Prototype. *ACM Trans. Comput. Educat.* **2009**, *9*, 1–13.
36. Aman, J.; Wilson, B.; Shirvani, S. Maintaining lecture context in a blended course. *J. Comput. Small Coll.* **2007**, *23*, 56–63.