# Learning Programming With Ruby

**Wolfgang Müller**, mueller@md-phw.de
University of Education Weingarten, Media Education and Visualization Group, Leibnizstr. 3, 88250 Weingarten, Germany

**Ulrich Kortenkamp**, kortenkamp@cermat.org
University of Education Karlsruhe, Centre for Educational Research in Mathematics and Technology (CERMAT), Bismarckstr. 10, 76133 Karlsruhe, Germany

## Abstract

Introducing students into the fundamentals of programming can still be considered as a real challenge. The choice of the right programming language seems to play a major role. In this paper we present our experiences with the programming language Ruby in introductory programming classes. Ruby is a relatively young programming language, which provides some very interesting aspects and seems like a very good candidate as a beginner's programming language, integrating the advantages of other languages with respect to learning programming, while still being professional enough to support open-ended learning. We discuss several aspects of Ruby that distinguish it from other languages and which make it a good choice for a beginner's course in programming. In addition, we discuss the approaches applied at two different universities in teaching programming with Ruby. We present in some detail the individual programming exercises and present student results. Finally, we discuss the pros and cons for applying Ruby as a first programming language.

## Keywords

Computer Science Education, Programming, Object-oriented Design, Ruby, Shoes, Rails

## INTRODUCTION

Introducing students into the fundamentals of programming can still be considered as a real challenge. Learning to program is generally considered hard, and programming classes often have high dropout rates. Reasons for this have been analyzed in some detail. A major problem is that programming requires competencies in different fields, and students do not only have to learn the syntax of a programming language and the semantics of language commands, but also at the same time fundamentals in algorithmic thinking and fundamental algorithms, program design, and program comprehension. Modern object-oriented languages in general add complexity in terms of concepts and language structures, and discussions are ongoing whether object-oriented concepts should be taught in introductory programming classes or not (Kortenkamp et al. 2009).

There is also an ongoing debate on the right choice of programming language for introducing students into programming. Languages such as Logo, Modula, or Eiffel have been designed with educational aspects in mind, but the lack of real-world relevance resulted often in a lack of motivation for students learning these languages. As a result, these programming languages play a minor role in programming classes today. C++, C#, and Java are examples for languages with wide application in the field of programming, also being applied frequently for introductory programming classes. However, the inherent complexity of these languages makes it difficult for novices to meet with success, and therefore there are quite a few approaches to utilize simplified variants of these languages or certain forms of filters to ease learning. The steep learning curve for such programming languages and the little attraction of corresponding classes to students have lead to the development of languages of programming environments that integrate graphics, animation, and multimedia components and that facilitate the development of interesting and interactive multimedia products with low effort (see Kelleher and Pausch, 2005, for a comprehensive overview). These languages and environments are designed to increase students motivation to learn programming and some of them are especially designed to attract artist, children, and women. Examples for such approaches are Processing (Raes and Fry 2007) Alice3D (Cooper et. al. 2000), Scratch (Maloney et. al., 2005), Squeak (Guzdial 2000, Guzdial and Rose 2002) and Croquet (Smith et. al. 2003). However, these languages in general also lack relevance in software development, and novices in programming are required to learn an additional programming language thereafter.

Ruby is a relatively young programming language, which provides some very interesting aspects and seems like a very good candidate integrating the advantages of other languages with respect to learning programming. It is also a programming language with increasing application in professional software development, especially in the area of Web projects. Ruby has been designed to provide an enhanced readability, providing command structures resembling more similarity to English language. Similar to languages such as Python, Ruby is an interpreted language, and the interactive interpreter provides simplified access to the programming language. Moreover, Ruby also integrates aspects for functional and object-oriented languages. The Ruby package Shoes make it very easy to create graphical user interfaces with multimedia components, thus addressing more complex programming exercises with higher motivational value from very early on. Finally, using the web application framework Rails it is possible to get to the level of complex web projects with database integration and web interfaces even in an introductory course on programming.

In the following, we will discuss certain aspects in some more detail, which make Ruby an interesting candidate for a programming language for beginners' classes in programming. In addition, we discuss the approaches applied at two different universities in teaching programming with Ruby. We present in some detail the individual programming exercises and present student results. Finally, we discuss the pros and cons for applying Ruby as a first programming language.

## ASPECTS OF THE RUBY PROGRAMMING LANGUAGE

Ruby is a dynamic, reflective, general-purpose object-oriented programming language that combines syntax inspired by C and Perl with Lisp and Smalltalk-like features. Ruby was initially developed and designed by Yukihiro Matsumoto during the mid-1990s. Ruby supports multiple programming paradigms, including imperative, functional, object-oriented, and reflective programming. It supports dynamic typing and automatic memory management utilizing a garbage collector. Similar to many other modern languages, Ruby is in interpreted language, providing an interactive interpreter for both the execution of scripts and development and testing.

A major design goal of Ruby was from the very beginning an enhanced readability of the code, emphasizing the needs of a programmer and understanding a programming language as a user interface with its requirements regarding usability. In addition, Ruby is often attributed with the principle of least surprise (POLS), meaning that the language tries to behave in such a way as to minimize confusion for experienced users. Last not least, Ruby was from the very beginning intended to be programming language for programming education, teaching students that programming is fun (Matsumoto 2000).

An example for the integration of functional elements into Ruby and also the enhanced readability of its expressions is depicted in figure 1. In fact, the iterator expression in figure 1a closely corresponds to its expression in English language.

```
[1,2,3].each {|i| puts i * i}
  # 1
  # 2
  # 3
```

```
[1,3,5,7].inject(:+) # => 16
[1,3,5,7].inject(:*) # => 105
```

Fig. 1: a) Iterator over a list with a parameterized code block,
b) Inject iterator for accumulating list elements with a specified operator
(Thomas 2009)

**Code Blocks** In fact, many of the concepts presented above already base on the concept of code blocks, which are the most visually distinctive aspect of Ruby Code (see Carlson and Richardson, 2006, for a detailed discussion). Code blocks are related to the concept of closures in other languages. They can be considered objects that contain some Ruby code, and the context necessary to execute it. Essentially, a Ruby code block is a method that has no name, represented in terms of an object. Ruby code blocks are related to closures in functional programming, and represent another example how functional concepts have been mixed with procedural and object-oriented concepts in Ruby. In fact, Ruby programs can hardly be written without code blocks.

Most other languages have something like a Ruby code block: Lisp's and Python's lambdas, C's function pointers, C++'s function objects, Perl's anonymous functions, Java's anonymous inner classes. In general, these language elements are usually considered as advanced concepts. In fact, Java first excluded closures deliberately in the first design to keep the language simple. It was only introduced later, but in a more restricted way based on anonymous classes. Concepts related to closures are usually not covered in introductory programming classes, and, instead, postponed and passed around. However, this often leads to confusion for novices in programming, since solutions to problems cannot be expressed simply and elegantly, and such solutions no longer resemble equivalence to typical design patterns to be found in textbooks, documentations, and forums. Furthermore, for specific problem areas such as user interface development, computer communication, and parallel computing, the corresponding API typically requires the utilization of closure-like structures. As a consequence, touching problems and examples in such areas usually require the introduction of this so far not covered aspect.

Unlike most other languages, Ruby makes code blocks easy to create and imposes few restrictions on them. Their primary purpose was the abstraction of loop constructs in Ruby. However, now they represent a central element of the Ruby language, and many language constructs in Ruby accept code blocks as parameters. In their simplest form they are defined as a block, with block delimiters either being curly braces or a `do ... end` construct, where the first version is preferred for simple code blocks in a single line, while more complex ones have to be defined in the latter form. Code blocks may also accept parameters given in "pipes", similar the lambda construct in Lisp. Figure 1a depicts an example for such a code block.

Like many other interpreted languages, Ruby does not enforce strict typing, but checks the type of objects at runtime. Even further, it is not necessary to assign types to variables or parameters, but a concept called "**Duck Typing"** is used. Actually, in object oriented programming the type of a variable corresponds to a class or interface. In Ruby, objects also belong to a certain class, but in order to be used as a given type it is not necessary for the object to belong to a class or implement an interface, but it is sufficient to implement the methods that will be used. In other words: If an object "behaves" like objects in a certain class, then we can use it as such in generic code that is written for objects in that class.

While it may feel uncomfortable for programmers that it is not mandatory to write code that includes explicit checks typing, it is very convenient for beginners who have enough other concepts that they have to learn. Learning the concept of a "type" or "class" can be postponed until it is indeed necessary. We will discuss this later when we explain the "classes on demand" paradigm.

## RUBY FRAMEWORKS

**Testing** So-called "agile" approaches (Cockburn 2007) currently represent a major trend in software development, and one central element of these is the application of a test-driven development paradigm. Following this paradigm, software development starts with the definition of unit tests, which are supposed to verify and validate the source code to be developed thereafter. Corresponding test cases are designed and implemented for the smallest testable parts of the application. Test frameworks simplify the implementation of such test cases and their automated execution.

A number of test frameworks are available for Ruby. As of version 1.9 the MiniTest framework replaces the Test::Unit framework as the standard testing framework in Ruby.

**Shoes** is a cross-platform approach to provide a simple and easy way to understand User Interface framework in Ruby (why 2009). Shoes is primarily intended for applications in education, targeting to ease UI development in Ruby for beginners and novices in programming. It is connected to the Hackety Hack environment, a free Ruby-based environment aiming to make programming easily available for beginners (why 2003). While Shoes is based on the standard Ruby interpreter and libraries, it is typically distributed also as an own, closed framework, supplementing the runtime system with a GUI and a documentation browser.

Shoes provides elements from standard UI toolkits in a simplified form. As such, Shoes also relies on the standard concept of stackable widgets with content. However, in Shoes layout mechanisms are directly bound to specific widget types, stacks and flows, thus, simplifying the definition of layouts. At the same time, Shoes utilizes Web concepts for the definition of content element level. The instantiation of elements such as paragraphs, borders, and images very much resembles the definition of the corresponding elements in HTML. As such, students with a basic understanding of web technologies and HTML can easily grasp the concepts of the

GUI API and are instantly able to design and implement simple user interfaces. Moreover, Shoes brings URLs and links to the GUI and makes it easy to implemtn actions. Again, code blocks represent the central language element to define these. Code blocks are being applied ubiquitously in Ruby programs, and learners will work with them in programming from the first moment on. As a consequence, students do not have to learn a new language construct when working with Shoes for the first time, like they often have to in other languages.

Shoes also supports the creation of graphics with primitives such as lines, ovals, or paths, and provides operations for transformation, such as scaling or rotation. In addition, means to define animations in an easy way are provided. Based on these functionalities, it is easy to develop interactive multimedia applications with Shoes, such as visualizations or small computer games.

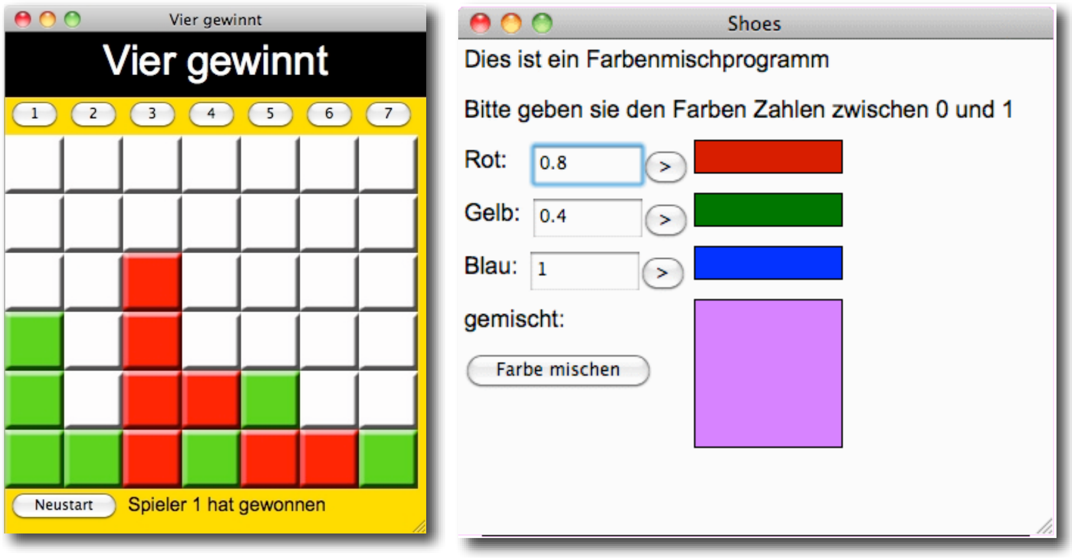
**Ruby on Rails** or “Rails” for short (http://rubyonrails.org/) is an open-source framework for creating web-based applications following a model-view-controller approach. It is particularly easy to write robust applications. It uses a concept of “convention over configuration”, that is best explained using an example: Usually, web applications use some kind of database (e.g., MySQL) that is used to store data in tables. These tables correspond to models that are usually represented by classes. In most languages this means that it is necessary to either use special tools to model this relationship between the database models and the classes, or to write the corresponding code twice with small variations. Rails circumvents this repetitive and error prone task by using conventions – every model (a subclass of a special API class) automatically corresponds to a table in database that is identified by the name of the class. The columns of this table in turn correspond to member variables of the class. The accessors for these members are created automatically using the metaprogramming facilities of Ruby (see Perrotta 2010 for this and more metaprogramming examples in Ruby).

## USING RUBY IN INTRODUCTORY PROGRAMMING CLASSES

Ruby has been applied as the programming language in a 6 ECTS introductory class in programming over one semester for students in the bachelor program on Media Education and Management at the University of Education in Weingarten in 2008 and 2009. In this study program, aspects of media design and production do represent a percentage of about 30%, only. Students in this program are expected to get a fundamental understanding in the principles of programming in this class. Typical class sizes were around 35 students, around 90% of them female. The vast majority of students does not have any experience in the field of programming at all.

Ruby was applied to ease the comprehension of fundamental programming concepts, while utilizing a “real” programming language with possible relevance to later work-life projects at the same time. Before, Java was being applied in similar as the introductory programming language. However, the inherent initial complexity of Java programs due to the strict application of OO concepts from the very beginning proved to be difficult for students. As a result, interesting levels of programmings, such as developing graphical-interactive applications, could hardly be reached.

Classes started with a general introduction into the Ruby's interactive shell irb, fundamental language elements, imperative programming concepts, functions and procedures, advanced language concepts were discussed. These included unit testing, I/O, exceptions, code-blocks, fundamental object-oriented concepts in Ruby, and GUI programming with Shoes. Object-oriented concepts were covered to the level of classes, interfaces and duck typing, and inheritance, only. More advanced concepts, such as design patterns, were omitted. In addition, students were introduced to data structures and fundamental algorithms, such as sorting. During this class, focus was put on performing projects in project teams, applying selected agile software development techniques. Examples for such projects includes the development of a simple graphical interface, a graphical calculator or a color editor, and a simple computer game. In the context of this last project, student developed independently in small teams small interactive games such as Tic-Tac-Toe, 4 in a Row, or Memory. Individual students developed further sophisticated applications, such as online shops with payment system and a simple HTML editor. During this development, they applied the test-driven development paradigm. Figure 2 depicts examples from these developments.



Spieler 1 hat gewonnen

Dies ist ein Farbenmischprogramm

Bitte geben sie den Farben Zahlen zwischen 0 und 1

Rot: 0.8

Gelb: 0.4

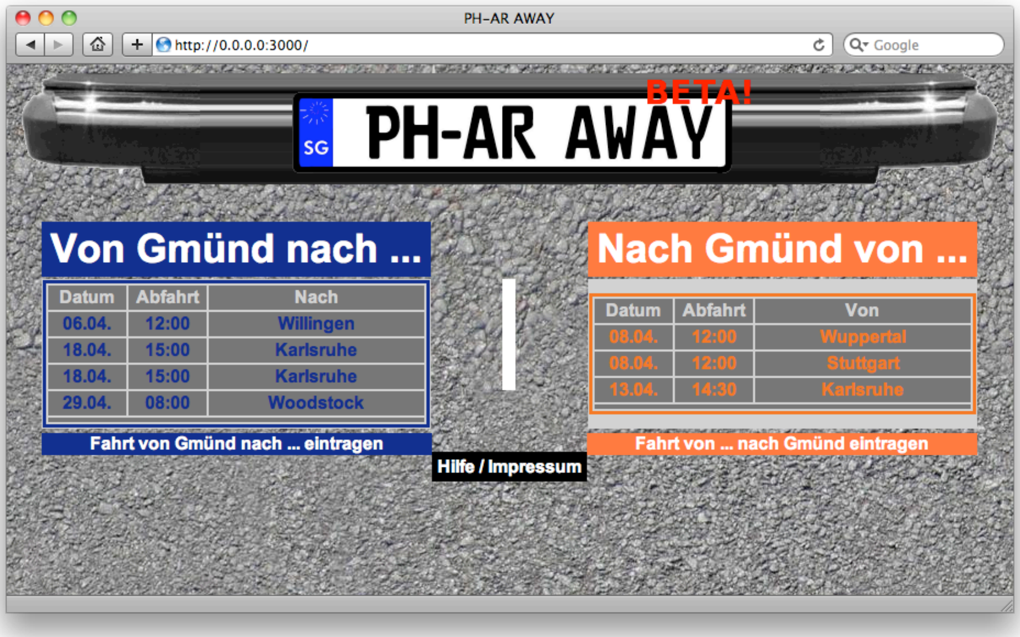
Blau: 1

gemischt:

Fig. 2: Example student projects based on Shoes.
Left: Four in a Row; right: Color Editor

At the University of Education Schwäbisch Gmünd[1] Ruby was integrated into the curriculum over the course of three semesters. All students had introductory courses in HTML in their first year. After that, the first true computer science course (2 ECTS) focuses on elementary algorithms (like sorting and searching), the second one, introduction to object-oriented programming (2 ECTS), uses Shoes, and the third course (6 ECTS) uses the Rails framework to create a web-based application. Typical course size is only 10-20 students. As these three courses are the only true computer science courses for them, it is challenging to bring them to a level high enough for teaching computer science in schools. However, in the last course which is organized in project form, they usually succeed to create a working non-trivial web application, for example a web-based ride sharing system that can be used by all students.

1 Ulrich Kortenkamp moved from the University of Education Schwäbisch Gmünd to the University of Education Karlsruhe recently. The curriculum structure in Karlsruhe is currently being adapted to the one described here.



Von Gmünd nach ...

| Datum | Abfahrt | Nach |
|---|---|---|
| 06.04. | 12:00 | Willingen |
| 18.04. | 15:00 | Karlsruhe |
| 18.04. | 15:00 | Karlsruhe |
| 29.04. | 08:00 | Woodstock |

Nach Gmünd von ...

| Datum | Abfahrt | Von |
|---|---|---|
| 08.04. | 12:00 | Wuppertal |
| 08.04. | 12:00 | Stuttgart |
| 13.04. | 14:30 | Karlsruhe |

Fig. 3: Example student project based on Rails. A ride-sharing web-application that was used later by students of the whole university

## DISCUSSION

The above presentation of language concepts depicts that Ruby provides a number of interesting features for teaching and learning programming. However, a simple exchange of programming languages alone without any modifications of teaching concepts clearly can hardly provide a very new access to programming. Yet, with the introduction of Ruby in beginner's classes in programming we also introduced a number of additional approaches connected to modern software development and targeted to ease learning programming. Clearly, most of these new approaches are not clearly bound to the programming language Ruby. Still, Ruby much simplified the introduction of these innovative concepts into the classroom.

In our classes, we urge students to apply agile software development approaches, such as test-driven development and pair programming. Agile techniques represent currently the state-of-the-art in software development, and the application of them in the classroom provides a high potential for making learning of programming skills more effective. Agile techniques not only introduce guidelines on how to solve problems in programming on an organizational level, some of them are also directly targeted to information exchange and learning in the context of software development.
From an educational point of view, test-driven software development provides a number of advantages. Taking tests as the starting point of the development of a solution forces students to analyze the problem in substantial detail, to describe the expected behaviour of classes and methods, and to classify and document problematic and exceptional use cases. Once developed, it also supports students in debugging own code by providing on the one hand a guideline on how to proceed in testing the software, on the other hand it helped students to avoid introducing new errors in the course of the development. Here, the possibility to use Ruby with the interactive interpreter and to utilize a state-of-the-art test framework provided in the standard library provided the flexibility to introduce test-driven development patterns

with simple means and to scale up to using the test framework at a later phase in the middle of the class.

The introduction into test-driven development was done using *micro-tests* as in the following example. Students were asked to write a function that calculates the maximum of numbers in a given array. They started with an empty definition of the function max: **`def max(l) end`**. Before they wrote the rest of the code –of course, they were not allowed to use the built-ins for this functionality– they should formulate some simple Boolean expressions that can verify the correctness of their code for some special cases, for example **`print max([5,3,6,3,7,2])==7`**. This line of code will print "false" if executed. Next, they had to fill in the implementation of the max function. If they did it correctly, all their tests should evaluate to "true", providing a means for self-evaluation of their exercises. As an extension of this concept, students could also work collaboratively on tests, sharing them with the class. Not only the tests gave them confidence in their own abilities, but they also served as a discussion stimulating element. Clearly, the test-driven approach in our classes proved not only applicable, but it supported students in particular in the project phases.

Another technique we applied in classes was the "I-am-an-algorithm" scenario. Instead of analyzing algorithms from the outside, students took over the role of the computer and went through the *algorithms in a role-playing game*. Again, we illustrate this with an example: The **`inject`**-method for arrays that takes a block as argument is far from being trivial to understand for beginners. Here, we asked three students to take over the role of the (1) the array, (2) the inject control structure, and (3) the code block. Now the inject-student asked the array-student for each of its elements, passing them via the pipe to the block-student, who works with this element and passes it back to the inject-student, who is responsible to ask for the next element, etc., etc. This didactical technique proved to be very effective. Students experience the shared responsibilities of the different code elements and become used to the abstraction patterns used in programming. It is particularly effective with the code blocks and pipes in Ruby.

A third example shall illustrate the advantage of Ruby with respect to the "lazy introduction" of classes of objects, or *classes on demand*. Since all data types in Ruby are objects internally, and since Ruby supports open definition of classes, i.e. the extension of class definitions at any time, it is possible to ignore the concept of classes until it is necessary or helpful to introduce it. In beginners' courses on object-orientation it is usually a problem that there is no real need for this higher abstraction (Kortenkamp et. al 2009). The artificial introduction of this concept is neither motivating nor justifiable. In Ruby, it is possible to introduce classes when it comes with a real benefit. The example given in (Kortenkamp et. al 2009) illustrates this with the introduction of a coding length method for numbers, strings and arrays. By "opening" the Fixnum class it is possible to add a method for the codelength (we use a compact notation for the definitions here to save space):

```
class Fixnum; def cl;(Math.log(abs)/Math.log(2)).floor+1 end; end
```

The same is true for String:

```
class String; def cl; 8*length end; end
```

This method cl can then –thanks to duck-typing– be used in generic code for arrays:

```
class Array; def cl; inject {|s, e| s += e.cl }; end; end
```

Using these three definitions, it is possible to find the coding length of an array of integers and Strings using the cl method. The same code using classic procedural concepts would be much more evolved and harder to understand. The students are able to appreciate this approach and are open to object-orientation for their further projects.

Not all aspects of Ruby proved to be completely unproblematic, though. Being a relatively young programming language, especially the Ruby language documentation as well as the error messages shown turned out to be unsatisfactory for learners in programming. Compared to languages such as Java, documentation of the standard API documentation proved to be rather brief, and provided often only limited examples. Also, students considered it often very difficult to find relevant descriptions in the API documentation due to Ruby's highly granular module structure and the corresponding fragmentation. On the other hand, there exists a large amount of freely available learning material, books and even comics on web, representing a valuable source for learners of this language.

Moreover, the Shoes framework in its current version can be considered a beta version only. The GUI functionality provided by the Shoes API is quite limited, and suited to develop simple graphical, interactive applications, only. Some bugs in the
Unfortunately, the further development of Shoes appears unclear, since the anonymous main developer, only known under his pseudonym “why the lucky stiff”, disappeared and completely shut down his web presence on August 19, 2009.

## CONCLUSION

Out first results from the application of Ruby as first programming language can be considered very promising. Not only does it support all programming paradigms we need for our students, but it also enables us to introduce them deliberately and flexibly.

Ruby proved to be easy to install and use, so there is not much overhead for the set-up of a learning environment. Even in the most basic installation the built-in interactive ruby interpreter irb can be used, if there is a need for more then it is possible to use an IDE like Netbeans, which comes with both JRuby and standard Ruby. Moreover, it is possible to migrate seamlessly into more complex project scenarios, as Ruby is robust, efficient and advanced enough to be used for commercial-grade programming.

The three didactical approaches of *micro-testing*, *algorithm-role-playing* and *classes on demand* are very well supported by Ruby and proved to be effective in our classes. A formal evaluation of upcoming Ruby classes based on self-efficacy (Bandura 1997) is currently in preparation. *Micro-Testing* fits very well into the general test-driven standard of Ruby-based development. *Algorithm-role-playing* highlights the importance of blocks, closures in functional programming and responsibilities of objects in object-oriented design. *Classes on demand* are only possible due to the open class definitions in Ruby and the general object-nature of all types in this language.

The lack of introductory textbooks for Ruby in German language was a bit problematic. However, we expect that the range of available books will increase in the near future due to the high suitability of Ruby for introductory programming courses.

## REFERENCES

Bandura, A. (1997) Self-efficacy. The exercise of control. New York. Freeman.

Carlson, Lucas and Richardson, L. (2006)  Ruby Cookbook. O'Reilly Media. 2006.

Cockburn, Alistair (2007) Agile Software Development: The Cooperative Game. 2nd Edition. Pearson Education Inc.

Cooper, S., Dann, W. and Pausch, R. (2000).  Alice: a 3-D tool for introductory programming concepts, Proceedings of the fifth annual CCSC northeastern conference on The journal of computing in small colleges. Ramapo College of New Jersey, Mahwah, New Jersey, United States.

Guzdial, Mark  (2000) Squeak: Object-Oriented Design with Multimedia Applications. Prentice Hall.

Guzdial, Mark and Kim Rose, Kim (Eds.) (2002) Squeak: Open Personal Computing and Multimedia. Prentice Hall.

Kelleher, Caitlin and Pausch, R. (2005)  Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers, ACM Computing Surveys, 37(2), 83–137.

Kortenkamp, U., Modrow, E., Oldenburg, R., Poloczek, J. and Rabel, M. (2009) Objektorientierte Modellierung – aber wann und wie?, LOG IN Heft Nr. 160/161, 22–28.

Matsumoto, Yukihiro (2000) The Ruby Programming Language. http://www.informit.com/articles/article.aspx?p=18225 (last checked 30.3.2010).

Maloney, J., L. Burd, et al. (2005) Scratch: A Sneak Preview. International Conference on Creating, Connecting, and Collaborating through Computing., Kyoto, Japan.

Perrotta, P. (2010) Metaprogramming Ruby. Pragmatic Programmers

Reas, C. and Fry, B. (2007)  Processing: A Programming Handbook for Visual Designers and Artists. MIT Press.

Smith, D.A., Kay, A., Raab, A. & Reed, D. (2003) Croquet – a collaboration system architecture. First Conference on Creating, Connecting and Collaborating through Computing: 2.

Thomas, D. (2009). Programming Ruby - The Programmatic Programmer's Guide. Pragmatic Programmers.

why the lucky stiff (2003)  The Little Coder's Predicament. Online: http://viewsourcecode.org/why/hacking/theLittleCodersPredicament.html (last visited: 22.03.2010).

why the lucky stiff (2009). Shoes. Online: http://shoes.heroku.com/ (last visited: 30.03.2010)

## BIOGRAPHY

**Wolfgang Müller** holds a doctorate in Computer Science and is a professor for Media Education and Visualization with the University of Education in Weingarten. He is a German delegate to IFIP TC 3, WG 3.3 „Research on Educational Applications of IT“. In addition, he is a co-editor in chief of both the Springer Journal Transactions on Edutainment and the Journal “Notes on Educational Informatics” (NEI).

**Ulrich Kortenkamp** received its PhD from the Swiss Federal Institute of Technology Zurich in theoretical computer science. In 2006 he became a Professor for Computer Science and Media Education at the University of Education Schwäbisch Gmünd. Currently he holds a position as Professor for Mathematics and Education in Karlsruhe and director of the Centre for Educational Research in Mathematics and Education, CERMAT.

## Copyright