How Should Teaching Modeling and Programming Intertwine?

Birgit Demuth
Presented by Sebastian Götz
Observation (Robert France, 2009)

There is a misguided notion among some students and practitioners that you can succeed in a software modeling course if your programming skills are weak:

You don’t have to program; you just draw pictures!
Hypothesis (Robert France, 2009)

Programmers with good modeling skills produce better quality programs than those with weaker abstraction skills.
Implication for Educators (Robert France, 2009)

Modeling should be developed alongside programming.
Teaching Software Development at the TU Dresden

<table>
<thead>
<tr>
<th>Semester</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>• Algorithms and data structures</td>
</tr>
</tbody>
</table>
| 2.       | • Programming  
|          |   • Introductory SE course |
| 3.       | • Software Project Course |
| 4. – 6.  | facultative advanced SE courses  
| (Bachelor)| • Software Engineering II  
| 4. - 10. | • Design Patterns and Frameworks  
| (Diploma)| • Component Based SE  
|          | • ... |
## Introductory SE Course

<table>
<thead>
<tr>
<th>SE</th>
<th>Software Engineering overview and software development processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OO</td>
<td>Object-Orientation (paradigm, thinking in objects, CRC card method, OO programming fundamentals)</td>
</tr>
<tr>
<td>OOA</td>
<td>Object-Oriented Analysis (static and dynamic domain modeling with UML)</td>
</tr>
<tr>
<td>OOD</td>
<td>Object-Oriented Design (software architecture, refined modeling, UML2Java)</td>
</tr>
<tr>
<td>OOP</td>
<td>Object-Oriented Programming (implementation of UML models with Java focusing on data structures (Collection framework, Generics))</td>
</tr>
<tr>
<td>DP/F</td>
<td>Reuse in OOD (design patterns (DP), frameworks (F) versus class libraries, testing with Junit)</td>
</tr>
<tr>
<td>Arch</td>
<td>Architecture of interactive systems, graphical user interfaces (introduction)</td>
</tr>
<tr>
<td>PP</td>
<td>Project Planning preparing the students for the subsequent software project (introduction)</td>
</tr>
</tbody>
</table>
Course Structure of two Didactic Approaches

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SE</td>
<td>OO</td>
</tr>
<tr>
<td>OO</td>
<td>OOP/DP/F</td>
</tr>
<tr>
<td>OOA</td>
<td>OOA</td>
</tr>
<tr>
<td>OOD</td>
<td>Arch</td>
</tr>
<tr>
<td>OOP</td>
<td>OOD</td>
</tr>
<tr>
<td>DP/F</td>
<td>SE</td>
</tr>
<tr>
<td>Arch</td>
<td>PP</td>
</tr>
<tr>
<td>PP</td>
<td></td>
</tr>
</tbody>
</table>
public class HelloLibrary {
    public static void main(String[] args) {
        Library myLib = new Library();
        Book b1 = new Book("UML");
        Book b2 = new Book("Java7");
        myLib.register(b1);
        myLib.register(b2);
    }
}
HelloLibrary Example (2)

```java
public class Book {
    private String title;

    public Book(String title) {
        this.title = title;
    }

    public String toString() {
        return title;
    }
}
```

```java
public class Library {
    private Book[] myBooks;
    private int number;

    public Library() {
        myBooks = new Book[10];
        number = 0;
        System.out.println("Hello Library");
    }

    public void register(Book book) {
        myBooks[number] = book;
        number += 1;
        System.out.println("New:" + book);
    }
}
```

HelloLibrary

```
public class Book {
    private String title;

    public Book(String title) {
        this.title = title;
    }

    public String toString() {
        return title;
    }
}
```

```
public class Library {
    private Book[] myBooks;
    private int number;

    public Library() {
        myBooks = new Book[10];
        number = 0;
        System.out.println("Hello Library");
    }

    public void register(Book book) {
        myBooks[number] = book;
        number += 1;
        System.out.println("New:" + book);
    }
}
```
Written Exams – Passed if ...

<table>
<thead>
<tr>
<th>Structure of the exam</th>
<th>All-up-rule</th>
<th>2-parts-rule</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Part I</strong> (45 points) <strong>OOA</strong>&lt;br&gt;(use cases, domain models, state machines, sequence and activity diagrams, CRC cards)</td>
<td>39,5/90 points</td>
<td>19,5/45 points</td>
</tr>
<tr>
<td><strong>Part II</strong> (45 points) <strong>OOD/DP/OOP</strong>&lt;br&gt;(interwined tasks about modeling, design patterns and programming including testing)</td>
<td>39,5/90 points</td>
<td>19,5/45 points</td>
</tr>
</tbody>
</table>
## Success Rates in Written Exams

<table>
<thead>
<tr>
<th>Approach</th>
<th>Success rate</th>
<th>Written exams (semester and year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall approach/All-up-rule</td>
<td>61.0%</td>
<td>WS 2001 to SS 2005</td>
</tr>
<tr>
<td>Waterfall approach/2-parts-rule</td>
<td>45.1%</td>
<td>WS 2005 to SS 2007</td>
</tr>
<tr>
<td>UMLByExample approach 2-parts-rule</td>
<td>59.6%</td>
<td>WS 2007 to SS 2012</td>
</tr>
</tbody>
</table>

**Number of students (average):**
- 230 students over all exams
- 370 students in first exams
## Average Number of Points

<table>
<thead>
<tr>
<th>Approach/Exam rule</th>
<th>Part I passed</th>
<th>Part II passed</th>
<th>Part I failed</th>
<th>Part II failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waterfall approach/All-up-rule</td>
<td>33.7</td>
<td>18.0</td>
<td>27.1</td>
<td>5.9</td>
</tr>
<tr>
<td>Waterfall approach/2-parts-rule</td>
<td>30.3</td>
<td>23.0</td>
<td>25.9</td>
<td>7.6</td>
</tr>
<tr>
<td>UMLByExample Approach/2-parts-rule</td>
<td>28.3</td>
<td>30.1</td>
<td><strong>18.4</strong></td>
<td>13.1</td>
</tr>
</tbody>
</table>
Lessons Learned

**Waterfall approach**
- Students learn modeling without a true understanding of the OO paradigm
- All-up-rule encourages students to concentrate on OOA -
  \[\text{You don't have to program; you just draw pictures!}\]
- Motivation for more attention to develop programming skills by the 2-parts-rule is not strong enough (decreased success rate)

**UMLByExample approach**
- intertwined modeling and programming in teaching OO software development
- helps to more effectively assist students in learning programming and understanding of modeling
- improves significantly programming skills
Conclusions

- Statistical evaluation of exam results over 11 years demonstrates the validity of Robert France's hypothesis.

- Limitation of our statistical observations:
  - Evaluation is based on **small** applications.

- Intuitive observation in the subsequent software project course:
  - Programmers with good modeling skills produce better quality programs than those with weaker abstraction skills.

- Empirical validation of this observation should be subject of future work.
Thank you for your attention and discussions!