TDE: a method and tool to teach OOP

Maurizio Morisio
Politecnico di Torino
Italy
maurizio.morisio@polito.it

Marco Torchiano
Politecnico di Torino
Italy
marco.torchiano@polito.it

Abstract
This paper presents an innovative approach to teach OOP, based on automated tools. The novelty of the approach lies in the complete automation of assignment handling; in addition the use of automated testing allows an objective evaluation of student elaborates.
We applied the approach using Java and the Eclipse environment but it can easily be adapted to other languages and contexts.

1 Introduction

Courses of object oriented analysis, design, and programming are now mainstream in software engineering and computer science curricula.
In our institution usually the courses follow a constructivist approach: students attend lessons, receive assignments that develop in the labs, with assistance from senior students, or at home. Finally they sustain an exam, which consists in developing a program. The program is developed on paper in a traditional classroom (no PCs available) during a session lasting around 2 hours. After this session the students, using a carbon copy of the program handed to the teacher, develop the program on a PC, completing and debugging it as needed. Next, they have an individual session with the teacher, who grades the student considering several factors: the program developed in the classroom, the program developed on the PC, the differences between them in terms of functionality provided, design choices and defects.
Focusing on the Java/OOP course, we noticed a number of problems and issues that could hinder the effectiveness of the course. We consider them under the headings of Labs and Exams.

- Labs. On average every week the students develop a new assignment, partly at the labs and partly on their own. Senior students provide assistance during Lab hours, both on use of the tools and on programming and Java topics. The main problem is the limited number of senior students available, and therefore the level of assistance available to students. Ideally, each assignment developed by each student should be evaluated for programming style and functional correctness, but this is not possible with the resources available.
- Exams. The paper based procedure for the exam has several cons. From the point of view of the students, developing the program on paper is frustrating, especially because they are used to best class tool support in the lab sessions, including extensive Java documentation, syntax checks and suggestions, pretty printing etc. From the point of view of the teacher, but also of the students, grading the students in a consistent and fair way, especially when more than one teacher is involved (the course has around 100 students per year), is hard. And the textual description of the program to be developed is often subject to ambiguities or misunderstandings.
- Motivation. We collected a large anecdotal experience from the paper-based exams; for instance when explaining the students how to use Integrated Development Environments (IDE) some students asked: "Why do we have to learn how to use an IDE since the exam in on paper?" This was one of the main motivations to go for a computer based exam instead of the old
- Correct use of tools. We realized that the students thought a program to be done as they finished writing the code. They were used to no testing phase, except, in some cases, running it with a sample input and observe the results. In the lab session we observed that often they didn't go as

far as compiling the programs, therefore their programs were full of syntactic errors. This bad attitude derived from the habit of handing out programming assignment on paper.

In 2003 the teachers of the Java course introduced the test first practice [1] and the related supporting tools (JUnit [4] within Eclipse [2]) with the goal of exposing the students to a promising technique and experimenting its results [3].

With the availability of these tools and techniques developing a new process for the labs and the exams was straightforward. Defined an assignment, the teacher develops a wrapper class to support the required high level functions, and a corresponding set of acceptance test cases. The students develop the assignment. Both the students and the teacher can then run acceptance tests on the assignment. During the lab period, the students get a precise assessment of what does not work in their program. For the exam, both student and teacher get an objective evaluation of the functions developed.

2 The core process

The core process (see Figure 1 with corresponding UML activity diagram) has three basic steps.

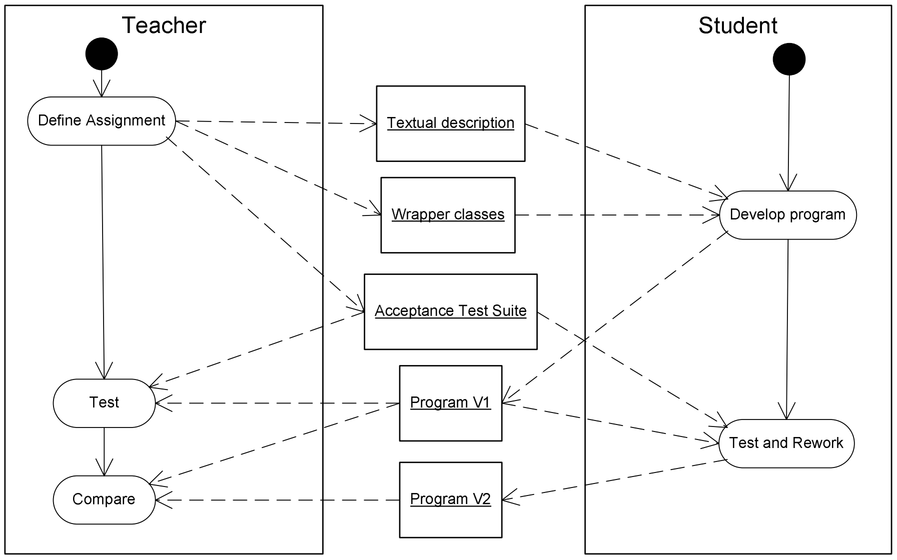

Figure 1. The core process

1 - The teacher defines the program to be developed, made of three parts.

- a textual description of the program functions and constraints (see Figure WW for an example)
- a wrapper class (when using Java, under the form of a Java class with prototypes of public methods, or as a Java interface)
- a suite of black box acceptance tests for the program (when using Java, under the form of one or more test classes in JUnit format).

Obviously the three parts of the program must be consistent. Acceptance tests call public methods of the wrapper class, and the textual description of the program must match the public functions of the wrapper class. In case of conflicts the formal part (wrapper class and acceptance tests) have precedence over the textual description.

2 - The student develops the program and delivers and initial version (V1).

3 - When complete she runs the acceptance tests against it using a web-based service. For each test the web service reports if passed or not, if not passed reports the error message from the compiler or the run time. The student is encouraged shall modify his program until all tests pass. The correct program is delivered as version V2

2.1 Metrics

In the above process we introduced some metrics based on the acceptance test suite developed by the teachers. The metrics are summarized in the following Table 1.

Table 1: Metrics used in the process.

Metric	**Definition**
NAT	Number of test methods defined in the Acceptance Test Suite
NPT_s	Number of tests passed by the program V1 developed by student *s*
$QUALITY_s$	$\frac{NPT_s}{NAT}$ Number of tests passed by program V1
MLOC*s*	Modified lines of code in Program V2 with respect to Program V1 developed by student *s*

The quality of a program (QUALITY) is evaluated in terms of passed acceptance tests, since the Acceptance Test Suite is developed independently by the teacher it represents a proxy of the field operation of the program.

The number of modified lines of code (MLOC) represents a proxy of the rework effort. Since the rework activity is performed discontinuously at home by the students it is very difficult to collect actual effort data.

2.2 Scenarios

We have adapted the process described above to two different scenarios.

Scenario 1. Weekly exercise. The teacher publishes a program description, and wrapper class. Students work on the development during a week. Then the teacher publishes the acceptance tests. Students check their programs against the test cases; modify and discuss them with the teacher. In our implementation delivery of V2 is not mandatory and students pass the assignment if QUALITY is above a predetermined threshold (currently we use 50%). This scenario can happen with students at the labs or elsewhere.

Scenario 2. Final exam. The scenario is similar to the one above for the first part, except all students are in the lab, a PC for each of them. The teacher publishes a program description and wrapper class. Students have a fixed amount of time (usually two hours) to develop and submit the program V1. Then the teacher publishes the tests, students have a few days (usually two) to fix all problems and submit again a fully working program V2.

The teacher grades each student considering two factors:

- the quality of program V1 measured by QUALITY,
- the effort to quality, that is the rework effort required to pass all acceptance tests, measured with the MLOC proxy.

3 Students and Metrics

With the proposed approach the students see the metrics (QUALITY in particular) as a measure of the goodness of their work.

In the exam scenario students essentially used the QUALITY metrics to track their progress towards a fully working version of their program.

We hope students could learn to use metrics to keep under control the minimal quality of their programs as they are developing them. For the time being the students seem to like the use of metrics since it provides them with an objective assessment of their work.

Currently we cannot foresee any specific lecture about metrics since we applied the approach in an OO programming course that is already over-packed with contents.

4 Issues

Using the approach in the last two academic years has pointed out the following problems and issues.

- White box evaluation of students elaborates.
  The acceptance tests developed by the teachers are, by definition, black box and must not make any assumption on the internals of what students produce. The white box evaluation of student's assignment (high level design issues such as choice of classes and functions, correct use of inheritance, use or misuse of patterns, simplicity; low level design issues such as choice of algorithms and data structures; formal issues such as names of classes and functions, indentation and documentation) has to be done manually.
- Design freedom, wrapper class.
  The teachers define a wrapper class (or sometimes an interface) in order to define a single set of acceptance tests for all programs developed by all students. Students must develop their program within the wrapper class and never modify it. In case of even a slight modification compilation will fail and no test will pass. This may be considered as a constraint on the design of the program by the students. We believe this is a minor issue as compared to the advantages of the approach. Besides, similar constraints are typical in commercial development. And finally students have full freedom of choices within the wrapper class.
- Effort to develop acceptance tests.
  Using this approach requires an upfront investment in developing the tool infrastructure. Besides, at every new exam session, the program assigned to students must be developed and the corresponding acceptance test cases defined. Writing the test cases requires usually some hours. Assuming 5 hours to write the test cases (for programs featuring 4-6 classes, to be developed in 2-4 hours) and assuming that an exam in the new form takes 20 minutes vs. 40 in the old form, break even lies at 15 students.

5 Conclusions – Future work

The ultimate goal we aim at reaching with this approach is automatic grading, with a clean procedure and public rules that can be understood by the students.
Since the proposed approach uses black-box testing, is not immediately suitable for programs heavily relying on algorithms or data intensive programs. For this kind of programs a different perspective should be adopted to assess efficiency and performance, and possibly adding white-box tests.

6 Acknowledgements

We wish to thank Giorgio Bruno, teacher at Politecnico di Torino for his cooperation and Hakan Erdogmus for the useful discussions that led to the definition of the measures and tool infrastructure of the course.

7 References

[1] K. Beck, *Test-Driven Development: by Example*: Addison Wesley, 2003.
[2] Eclipse Consortium, "Eclipse Platform Technical Overview" Object Technology International, February 2003 available at http://www.eclipse.org/whitepapers/eclipse-overview.pdf.
[3] H. Erdogmus, M. Morisio, and M. Torchiano, "On the Effectiveness of the Test-First Approach to Programming" *IEEE Transactions on Software Engineering*, 31 (3): 226-237, March 2005.
[4] JUnit.org, *www.junit.org*.