AICA

DIDAttica 2013 informATICA

"Se ascolto dimentico, se vedo ricordo, se faccio capisco"

27ª DIDAMATICA 2013

Tecnologie e Metodi per la Didattica del Futuro

ATTI

Pisa, 7-8-9 Maggio 2013
Area della Ricerca CNR

Organizzato da

In collaborazione con

Atti del Convegno Didamatica 2013

978-88-98091-10-2

Using Web-CAT to improve the teaching of programming to large university classes

Mario G.C.A. Cimino, Giuseppe Lettieri, Giovanni Stea
Dipartimento di Ingegneria dell'Informazione, University of Pisa
Largo Lucio Lazzarino 1, 56122 Pisa, Italy
{m.cimino, g.lettieri, g.stea}@iet.unipi.it

Recent staff cuts for the Italian university system reduced the teaching manpower, making face-to-face student/teacher interaction all but impossible in large classes (e.g., 100-150 students). On the other hand, new generations of students call for evolved self-learning instruments, and teaching practices need to meet this demand by adopting systems that allow for short-loop feedback and scalable class management. This paper discusses the adoption of Web-CAT, an open-source computer-assisted teaching software, within the BSc in Computer Engineering at the university of Pisa. We present in detail the motivations for adopting it, the customization effort and the expected benefits.

1. Introduction

The recent (2010-) reformation of the Italian university system has had a negative impact on the teaching activity within universities. First of all, by cutting PhD scholarship funds and setting near-impossible pre-requisites for getting a research position, it has discouraged graduate and PhD students to pursue academic careers (at least careers in Italy). This has led, quite predictably, to a massive reduction in junior research personnel, normally doubling as teaching assistants and/or tutors [Repubblica 2012]. Furthermore, the new norms on academic career progression only evaluate research throughput and completely neglect the teaching (let alone the teaching quality), thus orienting researchers to the former in lieu of the latter. Last, but not least, the reduced turnover forbids universities to even balance the retirement rate with new recruitments. The net result is that fewer senior researcher and professors are left alone to face larger and larger classes (the most recent norm setting the limit for a first-year Engineering or Computer Science degree to 150 students per class [DM47]), unable to provide an effective level of interaction with individual students.

On the other hand, the trend for heterogeneous classes at the undergraduate level is still on the rise: while most freshmen can run basic PC applications, their background on programming and algorithmic reasoning is highly heterogeneous, and largely depends on their secondary school and/or personal interests. This, if anything, calls for individual, focused and customized learning programs, which the current, scarce teaching manpower is unable to provide at such large scales.

ISBN 978-88-98091-10-2

The effects are easily recognizable in the “Darwinian” bimodal distribution of student performance: those who start with a lead (good scientific secondary school, previous exposure to programming) get on well, whereas those who do not end up either dropping out or swelling the ranks of long-term underachievers who get their BSc very late and with poor marks. This last phenomenon, endemic in Italian universities in general and in ours in particular, is being actively punished by the government through further funding reductions, which acts as a fairly obvious incentive for the universities to lower the bar so as to boost the pass rates.

In this scenario, programs in Computer Engineering (as well as many others which require a solid background in Computer Science, e.g. other Engineering, Science and possibly Management programs) could partially compensate for the lack of manpower by using computer-assisted teaching software. Although a large stream of literature on these systems is already available (see, e.g., [Ihantola et al. 2010]), no trial on the field has been done in our university, and their importance and benefits are often underestimated.

This paper presents the work done by teachers of the Computer Engineering program of the University of Pisa on one such system, namely Web-CAT [Web-CAT], [Edwards and Pugh 2006], which is being adopted as a teaching aid to first-year undergraduates as of this year. We discuss the motivations that led to its adoption, which are possibly common to similar programs in other Italian universities. We then describe the modifications and extensions to the Web-CAT platform that we have already implemented, which enhance its effectiveness. Finally, we outline the roadmap for further development.

2. Web-CAT Overview

Web-CAT is an automated grading system for programming exercises licensed under the terms of the GNU General Public License. It is highly customizable and extensible, and supports virtually any model of program grading, assessment, and feedback generation. The system is implemented as a web application with a plug-in-style architecture, hence can be extended to provide additional services. At a high level, Web-CAT hosts several *courses* (e.g., programming, software engineering), and allows their teacher or teaching assistant to enroll students and administer programming homework to them. Furthermore, it automatically verifies and tests programs uploaded by the students, and it grades them based on pre-set grading schemes. Therefore, all a teacher has to do is to design assignments, tests and related grading schemes, while the system automates testing and grading. Web-CAT also tracks the submission epoch and the number of failed attempts, and allows for grading bonuses based on these. Obviously, automatically generated grades can be manually overridden, which grants teachers the maximum freedom.

Web-CAT already comes with two plugins for Java and C++ [Shah 2003], [Vastani 2004]. With the Java plugin, static analysis of source code is performed using PMD [PMD 2013] and Checkstyle. PMD is a code analyzer which finds unused variables, empty catch blocks, unnecessary object creation, dead code, and so on. Checkstyle ensures that code conforms to particular

conventions of documentation and style (coding standard), thus sparing the teacher a boring, though important task. Teachers can write test cases and code coverage via JUnit [JUnit 2013] and Clover [Clover 2013]. JUnit is a unit testing framework for test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit. Clover identifies the riskiest code in a project, to guide the coder in the testing. These tools ensure that the solution is valid (passes instructor tests), correct (passes student-written tests), and complete (all functionality is exercised by the tests).

Fig. 1 shows an example of static analysis performed by Web-CAT, by pointing out a duplicated import statement and an unused member, and a summary of test results. Note that the system reports memory leaks, memory usage statistics, and several other useful metrics. A grading report is instead shown in Fig. 2. The design/readability score is manually generated, while style/coding and correctness/testing scores are automatic. The bottom part reports detailed feedback on the execution of methods for each class.

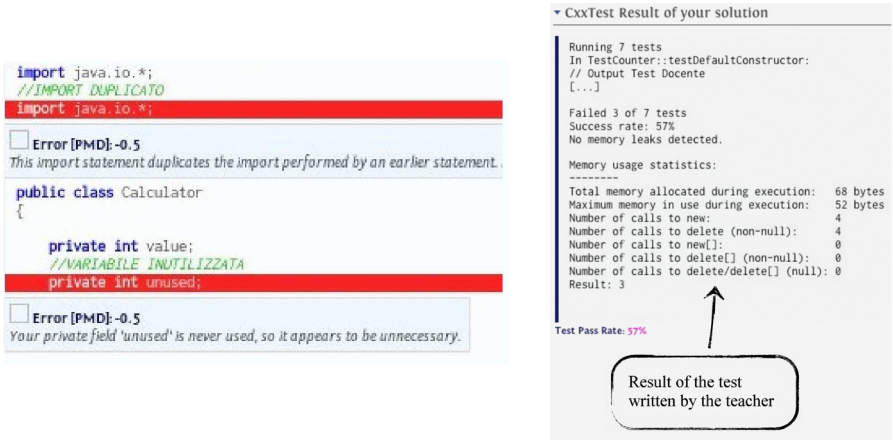

Fig.1 – Example of static analysis (left) and summary of test results (right)

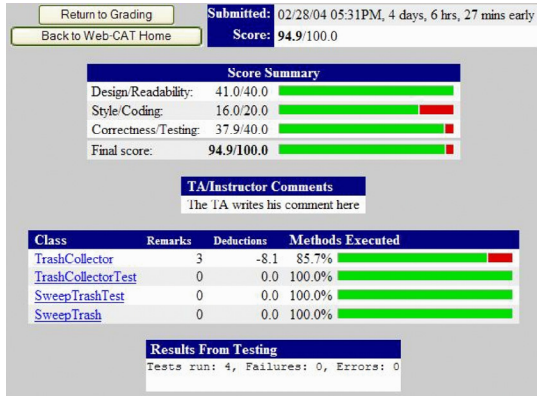

Fig. 2 – The grading report of Web-CAT

As far as the C++ plugin is concerned, testing is based on the Cxx framework, [Cxx 2013], which requires teachers to write a test class with the related methods. The plugin allows students to submit test plans as well as code, and it can evaluate test coverage besides code correctness.

Web-CAT is implemented in Java using the WebObjects framework [WebObjects, 2013]. From an architectural standpoint, it consists of three tiers: the client tier (i.e., the web browser); the server tier, where the logic resides, and java classes that support the generation of dynamic web pages are managed; the database tier, which manages information related to courses, students and assignments.

3. Contributions

We are about to experiment Web-CAT in the first year of the BSc in Computer Engineering at our university, after testing with it and finding it fit for our purposes [Del Vigna 2012][Salvini 2012][Formichelli 2012]. Web-CAT can be used for in-term homework assignment. In this role, its purpose is a) to present students with personalized self-assessment feedback, and b) provide instructors with aggregate, class-wise performance metrics in real time. Furthermore, Web-CAT acts as a database that can be mined by teachers to infer relationships between in-term performance (i.e., the learning process) and end-term exam results. We believe that this is the real enabler for long-term improvement of teaching practices. Finally Web-CAT can be used to manage end-term exams (whose grades actually matter), helping teachers to manage large classes. We now describe the motivations and expectations, the extensions that we have already developed, and those that are on the agenda.

3.1 Motivations, expectations and difficulties

For historical reasons, our teaching practice is based on the following philosophy: “ignore the **process**, grade the **results**”. Teachers do not normally intervene in the learning process of their students, which are expected to be autonomous, and they only grade exam papers. It is our belief that this philosophy is now counterproductive. Many students who fail (at least at first) might have succeeded if advised on their learning process early enough (i.e., during the first term). First of all, students should be presented with **clear requirements** right from the start. A widespread cause of failure is the misunderstanding that they will only be required to talk about programming principles (much like in a secondary school oral exam) rather than to get programs to work. A test-oriented approach to programming is part of the background of experienced programmers, but is completely new (and certainly not obvious) for first year undergraduates. Second, student lack instruments for self-assessment. Used as they are to constant, personalized tutoring by high school teachers, in their first university year they are abruptly confronted with impersonal teaching and long-term objectives, which they are expected to reach unsupervised. This implies that sometimes they just fail to see the problems they are facing until it is too late to solve them. We believe that administering

regular programming homework, with machine-generated feedback, can alleviate both the above problems.

On the other hand, the above teaching philosophy leaves teachers in the dark about their classes during the term, when corrective actions may have the highest impact. Mid-term exams are banned or discouraged due to cramped term schedules. Individual students can still go to weekly consultation meetings with teachers, of course, but very few actually do (partly because of the lack of feedback, as discussed above), and those who do hardly represent a statistically significant sample. Therefore, teachers lack the required feedback, and cannot direct more effort to those students for whom extra effort makes the difference (i.e., set up supplementary labs or classes for students who lack background, correct frequent programming mistakes etc.). By using Web-CAT during the term, teachers can identify deficiencies **in real time**, and devise specific and focused corrective actions.

Last, but not least, deans and department directors should be able to peer review the teaching activity. This may be required for several reasons: collective evaluation of department performance by national bodies (see again [DM47]), evaluating individuals for career advancement, coordinating and/or sharing best practices among teachers of similar subjects, identifying the causes of poor or unusual performance (on either side). As of today, the only reliable data gathered in our university are exam grades. Failures are not recorded, which makes it hard to infer pass rates, and we cannot correlate exam performance and in-term homework performance either (because there is no record of the latter). Web-CAT, by recording the performance of individual students, allows for extracting aggregate performance indexes.

The above-mentioned expected benefits cannot be reaped unless the majority of the students, possibly all, use Web-CAT routinely. Let us analyze possible reasons why they might not. The first one is that, unlike in American universities, here students cannot – and will not – be judged based on their in-term homework. This is because basing end-term marks on in-term homework grades would encourage cheating, since it is impossible to verify students' identity in a remote environment. Lacking a clear reinforcement, students may be discouraged to use the system, because this requires an immediate effort and pays off only in the long run. The second one is that today's students are privacy-conscious. They might perceive that, by using Web-CAT during their learning, they are exposing more of themselves than they really want to, and that their possible mistakes may negatively bias the teacher. The first issue can be mitigated by enriching the feedback that Web-CAT provides (which is already quite detailed) and by using it for the end-term exams as well. Anonymized student access (e.g., through untraceable credentials during the term) is an easy cure for the second issue.

Moreover, students are often too much result-oriented. Especially when lacking previous exposure to programming (e.g., in the first year), there is an actual risk that a positive machine-generated feedback (i.e., a 100% test compliance) will be mistaken for the true objective (i.e., writing good, correct programs). Worse yet, a *high* rate of compliance (e.g., 80%) might be mistaken for a positive feedback, whereas it may instead indicate that only the obvious tests were passed, and that the program does nothing significant in the general

case. We believe that this risk can only be mitigated by repeatedly stressing that (full) test compliance is a necessary, but by no means a sufficient condition for program correctness. This burden clearly rests on the teachers themselves.

3.2 Extensions of Web-CAT functionalities

Our first use of Web-CAT will be as a learning aid. Hence we have concentrated our first extension efforts in the direction of making it more useful as that. We plan to extend it to include the exam part later, and we will provide a roadmap for the planned extensions at the end of this section.

Our first extension is to integrate an interface to *tag* programming assignments. Tags are semantic contents (e.g., “pointers”, “list”, “recursion”, etc.) associated to individual assignments [Formichelli 2012][Salvini 2012]. The Tagging interface includes support for inserting, editing and removing tags by teachers. Tags are permanently stored in the database, with exercises and related assessments. This allows one to perform a-posteriori analyses by using tags as categories of filtering. To perform customizable analyses, we added an analytics subsystem by integrating with the Web-CAT database a JasperReports Server [JasperSoft 2013], a stand-alone and embeddable reporting server. JasperReports provides reporting and analytics that can be embedded into a web application, in a variety of file formats, to share reports and analytic views. The client application, iReport, supports a web-based, drag-and-drop report designer to create interactive reports for dashboards, as well as a Query designer to create customized report templates. Fig. 3 shows an overall picture of the tagging and analytics process. Teachers tag exercises via the Web-CAT teacher interface, as part of the preparation of exercises. When exams are graded, Second, the grading process is accomplished, once the exam is finished. Third, customized reports are made via JasperReports, to support the assessment process. The assessment process, in turn, may lead to new analyses/tags. Fig. 4 shows an excerpt of a customized analytics report made with iReport. Here, the tags assigned by the teacher are represented with different colors, defined in the legend on the bottom. The report shows the number of passed assignments for each student, and it is aimed at discovering possible problems related to specific topics.

As second extension we devised a language-independent plugin [Amadio 2013]. While the testing frameworks for Java and C++ is language-dependent and requires teachers to become familiar with specific testing frameworks, a test campaign may just rely on comparing the *actual* and *expected* program outputs obtained with a given input, whatever the language. To use this plugin, a teacher only needs to provide a link to the compiler and to a directory of matching input and output files. Obviously, input-output testing can be less accurate or insightful than the testing done using the Java and C++ plugins, but requires no extra effort on the part of the teachers, which makes it useful for specific purposes (e.g., a short Assembly language course, or learning the ropes of algorithms and functional programming before getting to use object-oriented languages). The new plugin has been tested with a C compiler. Part of the side benefits of this extension has been to acquire know-how on coding new plugins and getting an idea of the required effort.

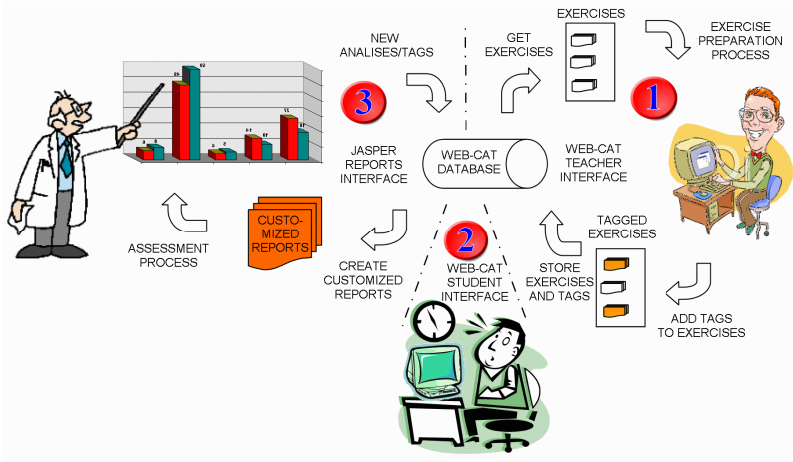

Fig. 3 – Tagging and analytics process via Web-CAT and JasperReports

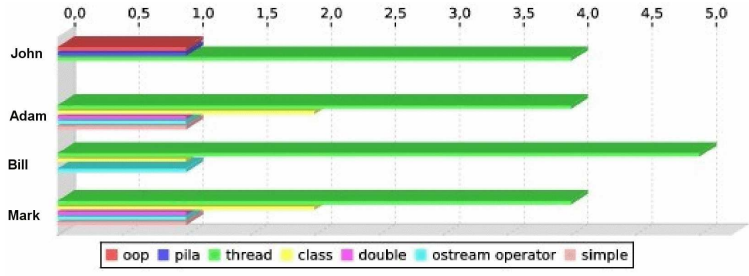

Fig. 4 – An excerpt of a customized analytics report made with iReport

3.3 Ongoing efforts

We have already argued that Web-CAT should be used for end-term exams as well. While automated grading (if taken with a grain of salt) is useful as an in-term feedback for both students and teachers, end-term grading should really be manually generated by the teachers themselves. On one hand, this poses problems of *scale*: grading 100+ complex assignments several times per year is time-consuming. Add to this the widespread student practice of "ten-percenting", i.e. carelessly tackling *all* the exam opportunities (as many as eight times per year), in the hope to compensate for a poor chance of success (note that our students get no career penalties for failing exams serially). A back-of-the envelope computation shows that a mere 20% of ten-percenters in a class of 150 succeeds in doubling the annual grading load of a teacher. On the other hand, and more importantly, when manual grading is required, the relevant question is: how do you grade a program that does *not* do what required? It is, in fact, far easier to grade a program which meets the specifications, e.g. based on efficiency, neatness or readability. The answer to the above question

depends pretty much on the teacher. We refuse to take the draconian shortcut and grade *every* bugged program as a no-pass, for the fairly obvious reason that a 200-line program may fail one test for very trivial reasons, which one more minute of debugging would have revealed to any student. On the other hand, our belief is that, whatever the choice a teacher makes, it should not rely on trying to divine the students' intention from their code, i.e., to figure out what the student **might have meant to do**, if only because it is a time-consuming exercise, whose results are almost inevitably wrong and biased by the teacher's mindset.

Both the above problems can be alleviated, if not entirely solved, by extending Web-CAT to incorporate *self-correction*, an alternative approach that has already been successfully experimented at our university (see [Lettieri 2013]). Students submit their assignment by uploading it on a server, and then are given a correct program to compare theirs against. Their code is still automatically tested using a number of predefined tests (which the students may or may not know beforehand), and students are given the opportunity to *correct* their assignment at home, until it passes all the tests. Corrections are recorded and visualized in the form of differential submission (i.e., lines inserted, deleted and modified), and the teacher can compare the initial and final code, see what and how many modifications were required to correct it, easily identify the bugs in the original submission and clearly follow (instead of trying to guess) the students' reasoning. Students *cannot* request a grade until their program passes all the tests. Obviously, they know their grade will be based on their initial submission (which takes place in a controlled environment), and it is in their best interest to show that only minimum modifications are required for their code to pass all the tests.

Scalability problems are alleviated because students are required to do most of the time-consuming work of bug fixing and correction, and teachers are left with grading. This, in turn, allows teachers to broaden the scope and/or increase the amount of code that students are required to submit for an exam test, thus making the exam more accurate: a typical case is exercises on lists-based data structures, which are often avoided by teachers (especially with large classes) because they require longer code and are generally harder to grade when wrong. Besides this, there are benefits for students as well: in fact, they are requested to correctly accomplish an assignment before they can even request a grade. Tough as it may seem, it is something with which aspiring computer engineers must learn to come to terms. Integrating self-correction in Web-CAT is in fact the next item on the agenda.

Finally, using a web-based system at the exams incurs the risk of cheating. While some precautions can be taken to prevent students from getting external aid (e.g., firewalling, diskless systems with limited permits), students' code should still be cross-checked pairwise to spot suspect similarities. This can easily be done by using one or more web-based services such as [Moss 2013] to which exam papers of a whole class can be submitted for cross-checking. At the time of writing, this service is being integrated into Web-CAT.

4. Conclusions and future work

This work has presented the motivations for adopting Computer Assisted Teaching software in Italian Universities. It has outlined the efforts in this sense made within the Computer Engineering program of the University of Pisa, which is adopting Web-CAT as of this term. In order to make the software more useful, we have customized it, adding functionalities that allow it to be used by teachers to manage large classes. As we write this paper, we are beginning on-field trials at the first year of the BSc. Our intention is to get a feedback from students as for usefulness, usability, possible modifications or extensions. Assuming positive feedback, our next step will be to increase the range of course offerings, extending it beyond the programming field. For instance, a logic circuits exam consisting in writing and simulating Verilog microcode can be ported on Web-CAT. So can an exam on database systems consisting in writing SQL queries. Our intention is also to use this system, and the large amount of data it provides, as a case study for research in several fields: artificial intelligence, for automatic grading of exams; security and privacy, to allow for anonymized student access during the course; data mining, to discover correlations between objective data (e.g., the number of exam tests solved before sitting for the exams) and exam performance.

Acknowledgements

We gratefully acknowledge the contribution of several people, without whom this work would not have been possible. Pericle Perazzo is experimenting Web-CAT within the “Algorithms” course at the BSc Degree in Computer Engineering, and has provided several assignments and the related tests. We are also indebted to Rudy Amadio, Gabriele Del Vigna, Daniele Formichelli, Tommaso Salvini, BScs in Computer Engineering at the University of Pisa, who have worked on coding/adapting plugins, and have installed, configured, documented and tested the system.

References

[Amadio, 2013], R. Amadio, “Progettazione e realizzazione di un plugin per la correzione assistita di esercizi in linguaggio C in ambiente Tomcat”, BSc Thesis in Computer Engineering, University of Pisa, 2013.

[Auffhart et al. 2008] B. Auffarth, M. López-Sánchez, J. Campos i Miralles, A. Puig, “System for Automated Assistance in Correction of Programming Exercises”, Proc. CIDUI 2008.

[Checkstyle 2013], Checkstyle, Java code checker, http://checkstyle.sourceforge.net

[Clover 2013], Java and Groovy Code Coverage, http://www.cenqua.com/clover

[Cxx 2013] Yet another C++ test framework, http://cxxtest.com

[Del Vigna 2012], G. Del Vigna, “Sistemi di ausilio per la correzione di esercizi di programmazione, C.d.L. Ing. Informatica”, BSc Thesis in Computer Engineering, University of Pisa, 2012.

[DM47] “Decreto autovalutazione, accreditamento iniziale e periodico delle sedi e dei corsi di studio e valutazione periodica”, Italian Ministry for University and Research (MIUR), DM47/2013, available online at http://tinyurl.com/buv4e8p

[Edwards and Pugh 2006] S. H. Edwards and W. Pugh, “Toward a common automated grading platform,” in SIGCSE ’06: Proceedings of the 37th SIGCSE technical symposium on Computer science education, (New York, NY, USA), ACM, 2006.

[Formichelli 2012], D. Formichelli, Integrazione di un modulo di reportistica in una piattaforma web per la correzione assistita di esercizi in linguaggio Java, BSc Thesis in Computer Engineering, University of Pisa, 2012.

[Ihantola et al. 2010] P. Ihantola, T. Ahoniemi, V. Karavirta, O. Seppälä, “Review of recent systems for automatic assessment of programming assignments”, Proc. 10th Koli Calling Int. Conf. on Computing Education Research, pp. 86-93, 2010

[JasperSoft 2013] JasperReports Server, Reporting and Analysis Server, http://community.jaspersoft.com

[JUnit 2013], Framework to write repeatable tests in java, http://junit.sourceforge.net

[Lettieri 2013] Giuseppe Lettieri, “Grading faulty programming assignments via student-submitted corrections”, proceedings of Didamatica 2013, Pisa, 2013.

[Moss 2013] Moss, a System for Detecting Software Plagiarism, Stanford University, CA, USA. http://theory.stanford.edu/~aiken/moss/

[PMD 2013] PMD, source code analyzer, http://pmd.sourceforge.net

[Repubblica 2012] M. Massimo, “All’università il precariato è strutturale: la ricerca senza borsa e senza futuro”, online article, http://www.repubblica.it, 28/3/2012

[Salvini 2012], T. Salvini, “Progettazione e integrazione di funzionalita' migliorative in una piattaforma web per la correzione assistita di esercizi in linguaggio C++” BSc Thesis in Computer Engineering, University of Pisa, 2012.

[Schwieren et al. 2006] J. Schwieren, G. Vossen, P. Westerkamp, “Using Software Testing Techniques for Efficient Handling of Programming Exercises in an e-Learning Platform”, University of Muenster, Germany, 2006

[Shah 2003] A. Shah, Web-CAT: A Web-based Center for Automated Testing, MSc Thesis in Computer Science and Applications, faculty of Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, 2003

[Vastani 2004] H. Vastani, Supporting Direct Markup and Evaluation of Students’ Projects On-line, MSc Thesis in Computer Science and Applications, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, 2004

[Web-CAT] Web-CAT, the Web based Center for Automated Testing, available online at: http://www.web-cat.org

[WebObjects 2004], Official WebObjects Community Website, http://www.wocommunity.org