A FLEXIBLE AND MODULAR SOFTWARE ARCHITECTURE FOR COMPUTER AIDED ASSESSMENTS AND AUTOMATED MARKING

Michael Striewe, Moritz Balz and Michael Goedicke

Specification of Software Systems, University of Duisburg-Essen, Essen, Germany

Keywords: Computer-aided assessment, Self-training, Assessment framework, Automated grading.

Abstract: In this paper we describe and discuss a flexible and modular software architecture for computer aided assessments and automated marking of exercise solutions. General requirements and problems are outlined with respect to assessment and self-training scenarios. As the main research result based on these requirements, an appropriate complete solution is presented by depicting a distributed, extendable, server-based assessment environment. Both the requirements and the architecture are evaluated by means of an actual system that is in use for assessments in Java programming for three years.

1 INTRODUCTION

Computer based exercises, e-learning, computer aided assessments and automated marking of solutions became important topics of research and discussion in recent years. Both increasing numbers of students and steady progress in computer infrastructures made it desirable to offer computer based exercises and examinations. Main goals are increased efficiency, reduced manpower needed for corrections and possibilities to apply various media and modern teaching techniques. Computer aided learning and computer aided assessments can be treated as very closely related in this context, because we are convinced that a tool used for marking solutions should be accurate enough to explain its marking decisions in a way that improves the students' learning process. If tools can assure the necessary quality in automated grading, they are useful for both formative and summative assessments. This applies to virtually any learning scenario, independent from topics of an actual course or the methods typically used for assessments on these topics.

Thus it is desirable to develop a software architecture that is flexible enough to serve for self-training and exercises as well as for assessments and exams and that can be used for various types of tasks in different topics. We especially don't want to limit ourselves to restricted question types like multiple-choice questionnaires, where the number of possible wrong and right solutions is known beforehand. Thus we try to cope with the problem that cognitive skills and application of methods cannot be assessed via multiple-choice tests (Chalmers and McAusland, 2002).

We are going to explain some general requirements for a flexible software architecture in section 2 and discuss an actual use case in section 3. A solution which is based on this use case and which fulfills the requirements follows in section 4. This solution is evaluated in section 5, before we give some bibliographic remarks in section 1.1 and conclude the paper with section 6.

1.1 Related Work

Most similar to the use case and our system we are going to described below is the web based tool PRAKTOMAT (Krinke et al., 2002), which offers almost the same features. It is capable of checking code in Java, C++ and Haskell with static checks and dynamic checks from an external test driver. The system is available as an open source project and under continuous development since 2001, but written in Python and thus limited to UNIX server environments. In addition, its architecture is not explicitly designed to be extendable by other marking components or to be used with completely different types of exercises.

Another almost similar project is DUESIE (Hoffmann et al., 2008), which is capable of checking Java and SML exercises as well as UML diagrams. The user interface is more limited without plugins for ECLIPSE and the core system is realized in PHP5, using external compilers, interpreters and

build tools. The technological gap between the scripting language PHP5 and external tools written in other languages is obvious and might cause problems regarding interoperability, security and comfort that are solved in a much more convenient way in the architecture proposed in this paper.

A platform for automated testing and assessment without limitation to certain subjects is offered by the web-based learning management system ILIAS (ILIAS, 2008). Several tools for multiple-choice questions in various forms are well discussed (e.g. (Hendrich and von der Heide, 2005) and (Glowalla et al., 2005)), but they are all not appropriate for complex use cases like the one depicted in section 3.

A lot of more specialised tools for limited domains exist as well. Different functional programming languages can be checked with the LLsChecker (Rösner et al., 2005) by comparing results from students solutions with results from a sample solution for given inputs. The tool EXORCISER (Tscherter, 2004) can be used for self-training without a server and offers exercises and automated verification for various topics in theoretic computer science, i.e. languages, grammars and Markov algorithms. Also in weakly structured domains like law, intelligent systems for detecting weaknesses in answers from students are used (Pinkwart et al., 2006). Nevertheless, these systems are not designed to be used both for self-training and exams and thus do not fulfill all of the requirements named in section 2.

2 REQUIREMENTS

A general assessment and training environment must support the whole process of assessments, which embraces the manual or automated management of examinees, manual creation of tasks, automated delivery of tasks, automated collection of solutions, automated marking solutions and management of manual reviews. Only slight variations of this process are used in self-training scenarios, where in most cases management of participants is less rigid and manual reviews by teachers should be reduced to a minimum in order to realize the expected increased efficiency.

Since a general system should be used with different kinds of tasks, we must expect users with knowledge in different subjects, different complexity of tasks and also different experiences regarding computer aided assessment services. This leads in every part of the overall process to advanced requirements in terms of flexibility and usability.

Additionally, since assessment systems are used in potentially distributed environments and manage data of possible judicial relevance, security issues are of concern during the whole requirement analysis.

2.1 Examinee Management

The security issues to expect require first an integrated user management and strict permission policies. For this purpose, examinees must be represented in the system at all stages of the overall process. We expect these user data to be specific: (1) Since the system will be prepared for mass assessments, we must handle large amounts of user accounts for each exam which may stay in the system for just one exam; (2) We cannot assume that we have an integrated access to a user/student data base. So one of the requirements for the system is that user/student data is easy to import and afterwards easy to identify in the system to allow a precise assignment of exercises. Additionally, if user data remains in active use in the system for a more than one exam, the personal data must be re-usable and also changeable by administrators.

2.2 Task Creation

Tasks in the context of assessments can be defined at different levels of abstraction. We define an *exercise* to be one task of a certain type, for example a multiple-choice question, a programming task or writing a short essay. An exercise consists of a task description and – according to the type – attached fragments. We distinguish *internal fragments* and *external fragments*: Internal fragments are created by the teacher and cannot be changed by the examinees. They may be visible to the examinees like questions in a multiple-choice exam or invisible like sample solutions the submitted solutions are compared with. In contrary, external fragments are created or at least modified by the examinees. They can be provided by the teacher as a starting point for a solution that has to be modified, for example source code fragments for programming solutions. They may also be created by the students themselves. We require the system to be able to handle any kind of fragments by offering uniform methods for attaching them to exercises.

A set of single exercises constitute an *exam* representing an academic record. Hence the definition of an exam must include an arrangement of exercises including premises, dependencies and possible repetitions. It should especially be possible to use a single exercise in different exams.

An important requirement for mass assessments is the definition of possible exercise or exam variants

that computer aided assessment systems can manage easily. Since examinees are expected to be managed by the same system and be unambiguously identified during exams, the assignment of variants to certain examinees can take place before the actual exam begins. Based on the students' data and the available exam variants different strategies to assign a student to a specific exam variant can be applied. In contrary, in self-training scenarios no fixed assignments are required, so the system should also be able to offer different exercises or exercise variants at once and allow students to choose one of them.

2.3 Exercise Delivery and Solution Submission

The requirement to cover different types of exercises leads to another requirement: a high flexibility of tools for submitting solutions. The simplest case of solutions types is multiple-choice, which requires at least a web site that can run on the assessment system's server. However, restricted input methods of web applications may not satisfy the needs of complex input types such as programming texts, graphics or the text of an essay. In such cases we need rich clients that connect to the assessment system in a secure way. Since special input methods like programming tasks may be integrated into other tools like integrated development environments (IDEs), we also expect a wide range of task-specific tools and thus need a flexible interface at the server side.

This stage of the process is most likely the target of fraudulent attempts by examinees and thus has high security requirements. To provide consistent security for exam situations, we identified the following issues so far:

- We cannot expect to be able to use existing login data as mentioned in subsection 2.1. Hence it must be possible to generate login data for imported user data.

- Login data must be easy to understand for students, even if they are nervous in an exam situation. Hence, we must define a minimum input required to identify students and their assigned exams.

- The login data usage must be restricted to avoid fraud. This restriction should be made based on time frames or locations by e.g. using one time passwords.

- Each commit of solutions must be logged and comprehensible afterwards to detect possible defrauds. Since data loss is not acceptable, solutions must be stored immediately and together

with meta-information about account, time and location from where the solution was submitted.

- Traceability of student's activities during the exam can be an important issue, both for judicial reasons and to enable recovery in the case of a failed client. Data related to the students' activities has to be collected and stored with the related solution(s).

In terms of usability, each possible client is required to provide a simple user interface to ask for login data in a simple way, provide input for single exercises, allow the creation or modification of solutions and commit (submit) the solutions to the assessment system. When an exam is composed of more than one exercise or one exercise contains more than one external fragment, the client must ensure that a student is given notice of all of them. In addition, the client should inform the student if external fragments are missing or have not been modified yet.

2.4 Marking Solutions

Different types of exercises need appropriate manual or (semi) automated marking mechanisms. Additionally, we require the system to be able to analyze one exercise regarding multiple aspects. For example, if students are asked to solve a task by submitting a diagram, one mechanism can check for the presence or absence of required elements inside the diagram, while another mechanism checks for conformance to styling guidelines that were taught in the according course.

Technically, our requirement is that the needed type-specific marking components are easy to develop, integrate and manage at run time. They also must be able to run outside the actual assessment system to allow load balancing: since the students' access to the assessment system must be as robust as possible and the system resources needed for marking are type-specific and not predictable, it is necessary to loosely couple these tasks. Thus we also need a run time system that executes arbitrary marking components and connects to the assessment system for this purpose. The assessment system is responsible for delivering solutions on request of the marking process until every single exam has been assessed and marked.

Running marking components in a separate run time system that connects to the assessment system raises additional security issues. To prevent fraudulent attempts, the assessment system has to make sure that only valid marking run time systems may access stored solution data and may deliver results for them.

2.5 Review Management

Teachers must be able to manage examinations by manual marking of single exercises as well as by review of automated results. Since automated marking may not be trusted in all cases, teachers may be required to validate automated results and override them if necessary. To avoid fraudulent attempts, all result changes have to be logged and be easily (re)viewable afterwards and may not be undone.

In exam scenarios it should also be possible to let examinees review their own solutions afterwards and see a justification of the results, both for educational and judicial purposes. At last, the result data must be exportable in different formats to be used as public notices, certificates and input data for other assessment management systems, for example those of examination authorities.

In self-training scenarios result reviews by students are even more important. If automated marking systems are used to reduce the need for manpower, students have to learn just by studying the results from the marking components. Thus we require each result to be visible for review and to be clear enough to be understandable without further explanation.

3 A SPECIAL USE CASE: PROGRAMMING EXERCISES

The requirements depicted above exceed by far those of average existing assessment systems. The reason is that with the advancing use of assessment systems, the need increases to cover more types of tasks than just multiple-choice tasks. For example, in basic lectures on computer science there is a strong need for systems able to provide and check programming exercises. We are using a system for automated marking of programming exercises since three years (Striewe et al., 2008). More than 8.500 exams and an equivalent number of self-training sessions have been conducted that shaped the profile of an existing programming course and clearly decreased failed final examinations according to our statistics. We will describe this special use case in the following to justify the requirements.

The user management is related to nearly all requirements for user managements mentioned above: The tool is used in a basic studies course with about 600 students per year conducting each up to 6 small exams during winter term. Before each exam, login data is imported from another existing tool managing registration and room assignment. The assessment system is responsible to assure student identity across several imports based on unique attributes like the matriculation number. During exams, no external network access is permitted to protect the exam against network attacks, so that all data must be completely contained inside the system. We chose the approach to identify students and their assigned exams by transaction numbers (TANs) that are generated before the exams begin. Since the user data import provides attributes from the registration tool like time slot and room number, we can assign exercise variants based on groups. In self-training exercises, no variants are offered and no assignments are made. The server used for self-training has full network access, so students can use their personal universal accounts. Without this flexibility in user management, the use of the same assessment environment both for exams and self-training would not be possible.

The exercise creation requires three kinds of resources for programming exercises: First, external fragments of source code are delivered that must be filled by the students. Second, graph transformation rules identifying instances of patterns contained in the solution are provided as internal fragments. Third, internal fragments of source code have to be provided for black-box testing to generate input to the submitted solutions and compare the output to the expected values. Thus our system has to handle fragments of completely different type inside one exercise. In addition, it is independent from these types, whether they are used as internal or external fragments, as there are external code fragments as well as internal code fragments.

The exercise delivery can happen in two ways. The simple way is a web interface that allows to download external fragments as far as they are provided and upload source files and compiled files afterwards. This first approach is used without problems in the self-training scenario, but has proven to be inconvenient for exam situations. Hence we built a plug-in for the ECLIPSE IDE (Eclipse,) which embraces the platform's functionality regarding code editors, project management and compilation in order to display and upload all parts of the solution automatically. Nevertheless, the web interface can be used in exams as well, again stressing the benefits of our flexibility requirements.

Independent from the interface being used, the TAN is the only login data that must be supplied by the student in exam situations. Since the TAN is printed on a paper sheet handed out to each student, all necessary data is available to the students without requiring external knowledge about access data. This

meets the requirements to make the login as fast and as riskless as possible. Additionally, since time and room information are available, we validate this way that no access happens at invalid times. On the contrary, in self-training students can use the web interface and are hence not forced to use ECLIPSE.

The programming-specific automated marking of solutions also relates to all the aforementioned requirements. Each exercise is checked via dynamic black-box tests as well as static source code analysis. Since the black-box tests execute arbitrary code defined by the students we must be especially careful to avoid malicious software attacks. The resource-demanding nature of static source code analysis required us to run multiple marking component instances in parallel while ensuring in the assessment system that marking of a specific solution happens only once. Details about the techniques used for dynamic and static checks are far beyond the scope of this paper.

The source code analysis is based on heuristics and cannot detect any possible correct solution, so that a review by the teacher is necessary if black-box test and source code analysis deliver contradicting results. We also offer students the possibility to review their own code and ask exercise-specific questions after the exam. Especially in this case it is of great benefit if the same system is used for exams and self-training, so students already know how to interpret the results presented to them during the review.

4 ARCHITECTURE

As specified in the requirements, the assessment system is inherently distributed to separate concerns and allow multiple access from a multitude of parallel users. Because of this, one central component – referred to as *core system* in the following – is responsible for the coordination of the single parts of the systems. To this core system we connect different clients used by examinees, an administration console for teachers and one or more marking components, while the core system connects itself to a database. Figure 1 gives an overview about this general architecture. The shown ECLIPSE front end for exams is only one sample rich client and in itself extendable for different purposes.

In general, the whole assessment system is designed and deployed as packages run on Java Enterprise Edition application servers. The package containing the core system is obligatory. Separate packages exist for the server-side parts of web access

front ends used by examinees and teachers as well as for web services communicating with rich clients. Another package contains the marking run time system. Each package can be deployed on the same physical server or run distributed over the network. The database server location is also independent and can be connected via network access.

Our productive system is configured in a way we consider the default design: All packages except the marking run time system are hosted on the same server as the database. The marking run time system is separated for security reasons. Several security requirements mentioned in section 2 are implemented by using elaborated network and firewall settings instead of developing additional software solutions.

4.1 Core System

The core system itself serves as a broker for all information used in the assessment process, the main tasks being: (1) Management of authentication; (2) import and management of according student data; (3) management of exercise definitions, creation of exams and assignment to examinees; (4) delivery of exams to students; (5) collection of results; (6) delivery of solutions to marking components depending on the type of the tasks and abilities of available components; (7) management of reviews and manual corrections if necessary. User activities are recorded during the whole process using logging mechanisms of the application server to make any interaction with the system traceable.

All persistent data is stored in a single relational database to avoid different storage locations like separate files in the file system. By using a relational database we can rely on database transaction mechanisms to prevent critical data loss during examinations. Additionally, it is easy to backup all system data from this single storage location. Each submitted solution to an exercise is stored with time stamp, unique identification number of the submitting account and network address of the computer used for submitting the solution, making them traceable even without using the server logs. The business logic itself uses object-relational mappings to represent data and thus facilitates a structured development approach regarding the data model.

4.2 Web Access for Students

The web-based front end for students provides two perspectives on the system. The default perspective is the *self-training mode* that can be accessed by students with a personal account.
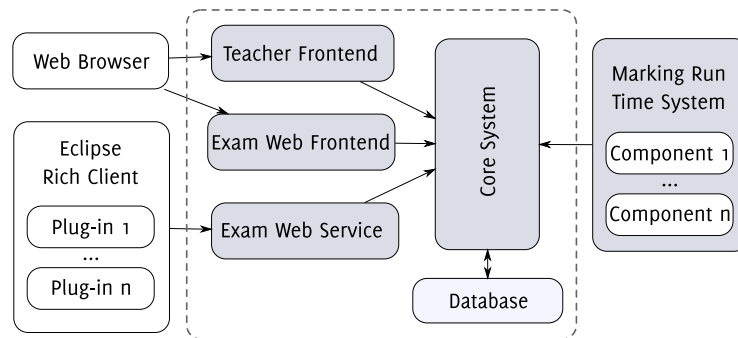
Figure 1: Architectural overview with possible clients (left), core system (center) and marking components (right).

This perspective utilizes login data defined inside the assessment system or connects to external authentication services. Once logged in, students can work with available self-training exercises and are for this purpose free to submit multiple solutions without any time restrictions. The personal account also allows a review of existing results of self-training exercises as well as exams. Since this user interface is not appropriate for exam situations, we provide a second, simplified perspective. This *exam mode* uses the TANs to identify students and allows to attend only one assigned exam. Students can thus neither review results nor choose to attend different exercises. Nevertheless they are allowed to submit multiple solutions.

The general handling of exercises is similar in both perspectives. Multiple-choice questions as a simple type of exercises can be handled directly inside the web browser, so students can directly tick their answers and submit them to the server. More complex types of exercises are handled by offering downloads for external fragments. Students have to know how to handle them properly, e.g. opening files in an appropriate editor. Similarly, fragments have to be uploaded again to the server in order to submit a solution. Since all expected fragments of the solutions are known beforehand, the system guides the user through the upload process by specifying all expected resources. This ensures that a solution can only be submitted if all expected fragments are present.

The result review allows students to examine their submitted data as well as all output from the marking run time and possibly manual teacher comments. All files attached to a solution are offered as downloads. Solutions cannot be changed in any way in this view.

The tracking and tracing of student activities is very limited when using web access since web browsers submit data only if requested by the user. If it is desirable to log mouse movements, clicks or cancelled submission attempts it is currently necessary to use a rich client with this capabilities.

In the future, we will also explore interactive web sites with JavaScript to fulfill such requirements, but do not expect the related tracking and tracing to be reliably recorded since web browsers allow end users to disable such interaction.

4.3 Rich Client Access for Students

Our solution of a plug-in for the widely used ECLIPSE IDE is an example how arbitrary client software for certain purposes can be integrated in the overall system with lean communication layers. Rich clients can be used to offer extended features to the students that assist them in solving their task, to enable the use of more complex exercises at all and to implement additional tracing capabilities for student activities. The communication between core system and rich clients is realized with SOAP web services. The according server-side communication layer allows to deploy customized adapter components for different types of clients that can be independently enabled and disabled.

The client plug-in itself uses the provided ECLIPSE platform, especially the Java Development Tools (JDT, 2008), to accomplish programming exercises. A dialog guides the student through the login process by requesting a TAN, downloading the external fragments and opening a Java project as well as all resources the student is expected to edit. Additionally, the user interface is simplified by closing all elements except a navigational view and a view for console output. To identify source code files and the according compiled binaries our plug-in relies on unambiguous information provided by the platform and avoids asking the user for additional information.

It is possible and planned for the future to develop other plug-ins for ECLIPSE in order to cover other types of exercises than Java programming. Virtually any kind of exercise can be handled this way, be it text input or even graphical tasks that need specialised editors. For simple tasks like multiple-

choice questions that do not make use of particular ECLIPSE features we also plan to develop plug-ins in order to enable tracking and tracing of student activities in exam situations.

4.4 Web Access for Teachers

In contrast to the student user interfaces that are as simple as possible, the administrative access for teachers must provide comprehensive and flexible tools to create, edit and analyze assessment data. User accounts can be created by importing existing user data via network services, spreadsheets or text files with comma-separated values. Exams can be assembled by using a repository of single exercises. Exercises have for this purpose been defined independently by specification of a description, internal and external fragments and an assignment to a marking component. Questions and answers in multiple-choice exercises can directly be edited in the web-browser, while source code fragments for programming exercises have to be provided as uploaded files.

The TAN creation process which joins login and exam data relies on standard techniques for generating random strings and explicit checks for duplicates to produce unique values. TANs can be grouped afterwards to allow manual or time-based activation. Reviewing results includes the opportunity to view or download the submitted solution fragments and to override automated results from the marking components. Detailed statistics for every exercise can be exported for further processing in external tools.

4.5 Marking Components

Type-specific solution input tools lead to an architecture enabling the integration of different, type-specific marking tools that can be dynamically configured regarding activation for exams, order of activation and testing criteria input. This modular architecture prevents any predictions of the capabilities the marking process might need. In addition to the security requirement to persist submitted solutions immediately, this is a reason to decouple the marking of solutions from their submission. Hence the marking components are executed in a run time system that can operate independently from all other parts of the system forming a master-worker architecture. In this way different marking components may be distributed over multiple physical servers for security or performance reasons and thus perform their work in parallel. Since marking and result submission is subject to security concerns, we set up strict network access rules using firewalls to ensure that access to the core server is only possible from valid marking components.

The marking run time systems themselves are able to execute multiple marking components on one physical server by providing only an environment and a network connection to the core system which is dynamically configurable. The core system is contacted regularly to access upcoming marking tasks which are then passed to an appropriate component. The result of the marking is submitted to the core system including all error messages and hints like the console output from black-box tests for programming exercises. This architecture is to some degree fault tolerant because the core system is not affected by the marking process. Thus marking components or the related servers may be disabled, disconnected or even crash without any consequences for the exam situation. Hence it would also be possible to design marking components connecting to different core systems, but running on one fixed physical machine, for example because of specialized hardware resources not available on all servers.

When checking programming exercises by executing code submitted by examinees, this code is started in a sandbox environment and not inside the marking run time system. This allows to apply security constraints to the sandbox, for example to prevent file and network access, and to catch easily any kind of runtime exception without affecting the marking component itself.

5 EVALUATION

The general requirements listed in section 2 have already been discussed and justified in section 3 by mapping them to the actual requirements of our actual assessment system. This system integrated well into an existing environment regarding server and client infrastructure, student registration systems and examination authorities. During the three years of use the chosen architecture has proven to be very robust. For example, extending the system for using multiple-choice exercises via web access in the second year was possible without any complications. Updates of the graph transformation component used for static code checks could be deployed easily.

In the case that crashes or unhandled infinite loops made marking components unusable in the first, experimental year of use, a switchoff or restart of the related faulty processes or physical servers was possible without any affect on the exam. Hence, the architecture fulfilled our requirements of undisturbed operations even with the use of potential unstable

marking components.

Deployment and network setup was as easy as expected and no serious crashes were reported. Two virtual machines that were copies of our servers have successfully been ported to another network environment and have been used in other real-world applications for university courses without any problems. Only slight effort had to be made for reconfiguration of firewall and network settings. Hence the teachers could concentrate on the development of tasks and marking setup for high quality exams and self-training offers.

The general approach to offer an architecture that can be used both for formative and summative assessments could be evaluated as well. In a survey among 62 students, 48 students named the system to be "useful" or "largely useful" for self-training and 40 students stated the same for the exams. Hence we can state that automated grading is not only useful for teachers in summative assessments to reduce the amount of time needed for corrections, but the same system can also be useful for formative assessments with feedback directly to the students.

6 CONCLUSIONS

In this paper we depicted and evaluated an architecture for computer aided assessments and automated marking of solutions in a server-based environment. We discussed general requirements as well as an actual assessment system. The presented system is not only a domain-specific implementation of the requirements limited to certain topics or kinds of exercises, but an extendable and flexible environment that can be used in various cases. The shown use case and the evaluation pointed out that the system is powerful enough to handle the complex scenario of automated checks for Java programming exercises both for exams and self-training scenarios.

More marking components for entirely different types of exercises will be developed in the future. Additionally, the concepts for access via web browsers and rich clients will be reviewed in order to make them even more flexible for different ways of submitting solutions.

REFERENCES

Chalmers, D. and McAusland, W. D. M. (2002). Computer-assisted assessment. Technical report, Glasgow Caledonian University.

Eclipse. ECLIPSE website. http://www.eclipse.org/.

Glowalla, U., Schneider, S., Siegert, M., Gotthardt, M., and Koolman, J. (2005). Einsatz wissensdiagnostischer Module in elektronischen Prüfungen. In (Haake et al., 2005), pages 283–294.

Haake, J. M., Lucke, U., and Tavangarian, D., editors (2005). *DeLFI 2005: 3. Deutsche e-Learning Fachtagung Informatik, der Gesellschaft für Informatik e.V. (GI) 13.-16. September 2005 in Rostock*, volume 66 of *LNI*. GI.

Hendrich, N. and von der Heide, K. (2005). Automatische Überprüfung von Übungsaufgaben. In (Haake et al., 2005), pages 295–305.

Hoffmann, A., Quast, A., and Wismüller, R. (2008). Online-Übungssystem für die Programmierausbildung zur Einführung in die Informatik. In Seehusen, S., Lucke, U., and Fischer, S., editors, *DeLFI 2008, 6. e-Learning Fachtagung Informatik*, volume 132 of *LNI*, pages 173–184. GI.

ILIAS (2008). ILIAS website. http://www.ilias.de/.

JDT (2008). Eclipse java development tools. http://www.eclipse.org/jdt/.

Krinke, J., Störzer, M., and Zeller, A. (2002). Web-basierte Programmierpraktika mit Praktomat. In *Workshop Neue Medien in der Informatik-Lehre*, pages 48–56, Dortmund, Germany.

Pinkwart, N., Aleven, V., Ashley, K. D., and Lynch, C. (2006). Schwachstellenermittlung und Rückmeldungsprinzipen in einem intelligenten Tutorensystem für juristische Argumentation. In Mühlhäuser, M., Rößling, G., and Steinmetz, R., editors, *DeLFI 2006, 4. e-Learning Fachtagung Informatik*, volume 87 of *LNI*, pages 75–86. GI.

Rösner, D., Amelung, M., and Piotrowski, M. (2005). LlsChecker, ein CAA-System für die Lehre im Bereich Programmiersprachen. In (Haake et al., 2005), pages 307–318.

Striewe, M., Goedicke, M., and Balz, M. (2008). Computer Aided Assessments and Programming Exercises with JACK. Technical Report 28, ICB, University of Duisburg-Essen.

Tscherter, V. (2004). *Exorciser: Automatic Generation and Interactive Grading of Structured Excercises in the Theory of Computation.* PhD thesis, Swiss Federal Institute of Technology Zurich, Switzerland. Dissertation Nr. 15654.