

Jussi Nikander

**Managing Automatically Assessed
Exercises in TRAKLA2**

November 29th, 2005

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietojenkäsittelyopin laboratorio

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory of Information Processing Science

Author:	Jussi Nikander	
Name of the thesis:	Managing Automatically Assessed Exercises in TRAKLA2	
Date:	November 29th, 2005	Number of pages: viii + 73
Department:	Department of Computer Science	Professorship: T-106
Supervisor:	Professor Lauri Malmi	
Instructor:	D.Sc.(tech) Ari Korhonen	
<p>The large class sizes in many universities have made automatic assessment an attractive solution to the increasing amount of work. An automatic assessment system is a software product that takes a submission to an exercise as an input, assesses the submission without need for human intervention, and produces feedback as an output. The output usually includes numerical grade for the submission, and can include other textual feedback, pictures and animations.</p> <p>Until now, most research on automatic assessment has focused either on the development of assessment methods, on the students' point of view, or on the learning results obtained by using an assessment system on a course. In this thesis, however, automatic assessment systems are examined from the instructor's point of view.</p> <p>Automatic assessment systems used for varied purposes in computer science teaching are surveyed, and several instructors who have experience with automatic assessment are interviewed. From the survey and the interviews, a list of functionalities required by automatic assessment systems is gathered and represented. In addition, a new instructor's user interface to the TRAKLA2 automatic assessment system is implemented. The results of the survey and the interviews are taken into account in the implementation. The new instructor's user interface is compared to previous versions in order to see the strengths and weaknesses of the new system. Finally, some future directions for the development of TRAKLA2 are suggested.</p>		
<p>Keywords: Automatic assessment, instructor's point of view, interviewing, TRAKLA2</p>		

Tekijä:	Jussi Nikander	
Työn nimi:	Automaattisesti tarkastettavien tehtävien ylläpito TRAKLA2:lla	
Päivämäärä:	29.11.2005	Sivuja: viii + 73
Osasto:	Tietotekniikan osasto	Professuuri: T-106
Työn valvoja:	Professori Lauri Malmi	
Työn ohjaaja:	TkT Ari Korhonen	
<p>Lukuisissa yliopistoissa kurssien suuret koot ovat lisänneet kiinnostusta automaattiseen tarkastukseen. Automaattinen tarkistin on ohjelma, joka ottaa syötteenä opiskelijan palautuksen johonkin tehtävään, arvioi palautuksen automaattisesti ja antaa tulosteena palautetta tehtävään. Palaute yleensä sisältää tehtävästä annetut pisteet, mutta voi sisältää myös muuta tekstipalautetta, kuvia tai animaatioita.</p> <p>Automaattisen tarkastuksen tutkimus on keskittynyt lähinnä tarkistustapojen kehittämiseen, järjestelmän opiskelijakäytön kuvaamiseen tai järjestelmän avulla saatujen oppimistulosten analysointiin. Tässä työssä tarkastellaan automaattisia tarkastusjärjestelmiä opettajan näkökulmasta.</p> <p>Tässä työssä tarkastellaan useita opetuskäyttöön tehtyjä automaattisia tarkastusjärjestelmiä ja haastatellaan järjestelmiä käyttäneitä opettajia. Haastattelujen ja järjestelmätutkimuksen pohjalta tehdään lista automaattisten tarkastusjärjestelmien tarvitsemista opettajan toiminnallisuuksista. Toiminnallisuuslistaa käytetään hyväksi tehtäessä TRAKLA2 automaattiseen tarkastusjärjestelmään uusi opettajan käyttöliittymä. Uutta käyttöliittymää verrataan vanhan TRAKLA-järjestelmän opettajan käyttöliittymään. Lisäksi ehdotetaan joitain tulevan tutkimuksen aiheita.</p>		
Avainsanat: Automaattinen tarkastus, haastattelu, opettajan näkökulma, TRAKLA2		

Acknowledgements

First and foremost, I'd like to thank my supervisor, Professor Lauri Malmi, and my instructor, D.Sc. Ari Korhonen, for their guidance, support, and especially patience, as this project just kept getting longer and longer. Without their help I never would have gotten to this point.

I also want to thank Ville Karavirta for the tremendous effort he has put into the core and student's user interface of the TRAKLA2 environment and for the implementation of HTTPS support. Otto Seppälä I'd like to thank for the implementation of the submission viewer applet. I also wish to thank Harri Salonen for his contributions to TRAKLA2.

My greatest gratitude, however, goes to my fiancée Satu, who always loved and supported me, no matter how tired, grouchy, or moody I got. Thank you, love.

Finally, I would like to apologize to the rest of my family, who barely heard or saw anything from me, as I was absorbed in getting this thing finished. I'll try and rejoin the rest of mankind now, I promise.

Fnord.

Otaniemi, November 29th, 2005

Jussi Nikander

Contents

Terminology	viii
1 Introduction	1
2 Objectives and Context	5
2.1 User Groups	6
2.2 The TRAKLA2 Instructor’s User Interface	6
2.3 Evaluating Automatic Assessment Systems	7
3 Related Work	8
3.1 Exercise Types	8
3.2 Multiple Choice	8
3.2.1 Älypää	9
3.3 Programming Assignments	11
3.3.1 ASSYST	12
3.3.2 Ceilidh	13
3.3.3 Style++	14
3.3.4 Scheme-robo	16
3.4 Visual Answers	17
3.4.1 TRAKLA	18
3.4.2 TRAKLA2 Prototype	19
3.4.3 Stratum	21
3.4.4 Exorciser	22
3.5 Text Answers	23
3.6 Peer Assessment	23
3.7 Conclusion	23

3.7.1	Instructor’s User Interfaces	25
4	Interviews	26
4.1	Interviewees	28
4.2	Conducting and Analysing the Interviews	28
4.3	Results	29
4.3.1	Examination and Modification of Assessment Results	30
4.3.2	Searching	30
4.3.3	System maintenance	31
4.3.4	Result viewing	32
4.3.5	Statistics	32
4.3.6	Results Beyond Functionality	33
5	The TRAKLA2 System	34
5.1	Overview	35
5.1.1	Student’s User Interface	35
5.2	Instructor’s User Interface Design	37
5.2.1	Design Principles	37
5.2.2	Use Cases for Instructor’s User Interface	38
5.2.3	Development Process	40
5.3	Architecture	41
5.3.1	Tools and Frameworks	42
5.3.2	Implementation	42
5.4	The Prototype Instructor User Interface	44
5.4.1	User Reaction to the First Version	45
5.5	Implementation of the Instructor’s User Interface	46
5.5.1	Code Level Details	46
5.5.2	User Views In Instructor’s User Interface	48
6	Evaluation	56
6.1	Experiences on Using TRAKLA2	56
6.2	Using ISO 9126 To Evaluate Automatic Assessment	57
6.2.1	Functionality	57
6.2.2	Usability	63

6.2.3 Reliability	64
6.3 Conclusion	66
7 Summary	67
7.1 Future Work	68

Terminology

This section defines some terminology used in this thesis.

- **Computer Assisted Assessment (CAA)** covers the use of computers in assessment, including assignment delivery, marking and reporting. It also encompasses, for example, the use of computers in optical mark reading and similar activities.
- **Computer Based Assessment (CBA)** covers the use of computers for the entire assessment procedure including delivery of the exercises and feedback.
- **Automatic Assessment (AA)** is a procedure that takes an answer to an exercise as input and produces a mark and optionally some feedback as an output.
- **Automatic Assessment System** is a software system capable of automatic assessment.

Chapter 1

Introduction

The amount of students on introductory programming courses can be staggering. For example, on some courses at the Helsinki University of Technology the number of attendees is well over four hundred per year [27]. Such a huge number of students is a great challenge to the instructors, as there will be dozens of students per instructor. Therefore, the course staff does not have a lot of time to spend on instructing a single student or assessing his exercises.

However, for most students, it is not enough to just read books or passively listen to lectures. In order to learn the student must actively participate in the learning process by doing exercises. Moreover, just doing exercises is not enough, and the student must be given feedback on his work. Otherwise he cannot know how he fared, and cannot learn from his mistakes.

Unfortunately, if a course has hundreds of students, all of whom do a large number of exercises, the teaching staff does not have time to manually grade all the exercises. *Automatic assessment*, which is becoming ever more common in computer science education, is therefore needed [8]. With automatic assessment it is possible to cope with a large number of students without requiring an unrealistic number of instructors. Also, AA allows the course staff to concentrate on instructing the students during lectures, exercise sessions and meetings, instead of having to spend all their time assessing the exercises. Furthermore, if the instructors do not need to use their time assessing simple exercises, it is possible to give more challenging exercises to the students. Such exercises, for example programming projects, cannot be assessed automatically.

In addition to saving time, the use of automatic assessment has several other benefits compared to manual grading. Students can submit their answers for grading whenever they wish and receive immediate feedback on the answers [49]. It is also possible to allow the students to submit their exercises multiple times. By reviewing their old answers and the feedback given by the automatic system, the students can correct their submissions and learn from their mistakes [26]. Using an automatic system it is also easier to compare student answers to one other and spot possible plagiarism [38]. An automatic assessment system also eliminates the differences in evaluation and assessment between different people checking the same exercise [35, 39].

Several down-sides to automatic assessment have also been reported. A survey made by Carter et al. [8] lists reliance on technology, security, and plagiarism issues. An automatic system must run on a computer, usually one connected to the internet. All computers are susceptible to system failures which may, for example, prevent a student from submitting his exercises or, in the worst case, cause a submitted answer to be lost. Also, when answers are submitted through the internet, it is hard to reliably identify a student and make sure that he submits his own work. He can just as easily copy his answers from a friend, or from the Internet. Furthermore, if security isn't given enough consideration, it is possible to gain unauthorized access to the system and modify the results.

Even if AA is not used, however, there will be analogous problems on courses with hundreds of students. For example, when hundreds of submissions are handled without automatic systems, there is a chance that a submission may be lost, or that a grader forgets to assess one exercise. Also, even if the submission is returned on paper, it is not possible to reliably check whether a student has done the exercises by himself. The student can still have a friend do the exercise in his stead, copy his answers from the Internet or from another student. Discovering plagiarism is actually easier when automatic systems are used. When there are hundreds of students returning the same exercise, a human instructor needs a lot of time to check all the answers. The huge amount of work and the human inability to remember all the small details makes it impossible for the instructor to spot all suspicious answers. Therefore, using a well-made automatic plagiarism checker is a great help in spotting plagiarism, since most of the work can be done automatically. The human instructor needs only to check the submissions found by the automatic system.

Considering the many advantages automatic systems have and their huge potential for time-saving, it is not surprising that a great number of automatic assessment systems have been developed during the last decade and a half. They are a very attractive, and in some cases, the only workable solution to the workload caused by the large number of students.

However, most of the research in the area of automatic assessment systems is focused on either describing implemented systems and frameworks [11, 14, 21, 33, 44], or analysing their effectiveness [5, 27, 43]. Typically, the main focus in the development of an AA system is the implementation of the assessment module, exercises and the student's user interface. However, the system must also have some sort of interface for the instructors. Unfortunately this interface is almost never described in detail, and typically is thought to be less important than other aspects of the assessment system. This will easily lead to a situation where the system is very hard for the instructors to use, and therefore will not be adopted outside the group that originally created it.

The amount of work that an automatic assessment system can save is usually cited as the main reason for using such a system. There are, however, several other possible benefits in using AA. The system can log massive amounts of data concerning student activity. For example, the students' exercise submissions, answers, and assessment results can be stored. Typically, this information is used only to monitor student progress on the course, and to compile the final grades for the students. The information could, however, be used for several other purposes. It is, for example, possible to compare the exercises and see which of them are hard and which easy, by examining the students' results. If the system gathers

additional data, like some information on the students' problem solving process, a more detailed analysis is also possible [40, 41].

An automated system can, however, do only what is programmed to do. Therefore automatic assessment systems must be well-designed, and flexible enough to support several different use cases. If, for example, a system does not adequately show whether it received a submission, many students will probably find the system frustrating to use. Or, if it fails a submission because of a simple typographical error, the students are unlikely to think the assessment to be fair. Similarly, if the feedback consists only of the amount of points gained from an exercise, is not very useful for the student. He knows that something was wrong in his submission, but does not have any hints on where the mistake is.

From the point of view of the instructor, the system must be flexible enough to support the way he wishes to organize his course, and to handle the occasions where students need some special considerations. If, for example, an automatic assessment system has no support for deadlines, the instructor cannot use it to structure his by having the exercises have different deadlines. As another example, on large courses, students often need extra submissions, or extensions to deadlines. If the system does not have support for such modifications, the instructor may need to do a lot of extra work manually. This, in turn, will undermine the main reason for adopting automatic assessment: reducing the amount of work required for course management. Also, if the system has no support for handling special cases, the instructor may be forced to modify the data in it manually. This, in turn, is error-prone, and can make the system behave incorrectly.

To summarize, from the instructor's point of view, an automatic assessment system has a large role besides just helping the students to learn the topics of the course and reducing the number of exercises that need to be graded by hand. An automatic assessment system is often a course management tool, which is used to track students' progress and store their results. A good automatic assessment system also allows the instructor to run the course as he wishes, and to modify it for students who require some individual arrangements. A good system can also be used to further develop the course and support research.

Unfortunately, in many automatic assessment systems, the instructor's user interface has only a very limited functionality or may be completely missing. Especially if the system is new and still under development, the instructor's user interface typically has a much lower priority than the student's interface, assessment modules or new exercises. This can easily lead to a situation where the instructor's user interface to the system is a number of shell-scripts or a database terminal program. For the developers of the system, this can be enough, since they know how the software works, how the data is stored, and where to find required pieces of information. Therefore they can rather easily create new functionality on-the-fly. Such an interface is, however, completely inadequate for people who are not familiar with the code-level operation of the system. Therefore, without a good user interface for the instructor, it will be hard for other institutions to adopt the system.

It is clear that instructor's user interfaces to automatic assessment systems need more research. A good user interface makes the system easier to use and further reduces the amount of effort required to manage the exercises and courses. It also prevents simple mistakes, like incorrectly formulated SQL commands or typographical errors, from disturbing the normal

flow of the course. A good interface makes it possible for people who have not participated in its development to effortlessly take the system into use, and therefore makes it easier for other institutions to adopt it.

In this thesis, I will study *the instructor's user interfaces to automatic assessment systems*. By examining existing systems, and interviewing instructors who have experience on using automatic assessment, I will try to find out the required features for an useful instructor's user interface. I will research the automatic assessment systems in the context of *computer science education*, with special emphasis on courses with hundreds of students.

The rest of this thesis is organized as follows: in Chapter 2, I introduce the objectives of this thesis in more detail. Chapter 3 gives an overview of previous research on automatic assessment systems as I briefly introduce several existing tools. In chapter 4, I describe the results of expert interviews I have conducted. Chapter 5 concentrates on the TRAKLA2 system, and there I describe the new instructor's user interface I have implemented to the system. Chapter 6 contains the evaluation of the new TRAKLA2 features and the experiences gained from using the system. The final Chapter 7 holds a summary of the work.

Chapter 2

Objectives and Context

In this work, I will examine automatic assessment systems from the point of view of the course staff. The emphasis is in trying to find the instructor's user interface functionality required for such a system. As a case example, I implement an instructor's user interface to the TRAKLA2 system[22]. TRAKLA2 is an automatic assessment system for data structures and algorithms, which is currently used at Helsinki University of Technology, Tampere University of Technology, University of Turku and at Helsinki Polytechnic Stadia.

The research problems for this work are:

- To find the required functionality for a useful course staff interface to an automatic assessment system
- As a case example, to construct a staff interface to the TRAKLA2 system

The functionality required for course staff interface is gathered by examining existing systems, literature on previous research, and by interviewing instructors who have experience with automatic assessment systems. The functionality is gathered into two lists of functionality: functionality in existing systems and functionality mentioned by experts.

In this work an *automatic assessment system* is defined as a software product which takes a *submissions* to an *exercise* as input, assesses the submission without need for human intervention, and produces useful feedback based on the assessment. Typically the feedback takes the form of points (out of some maximum) and text (typically comments on the answer). Other forms of feedback, such as visualizations, animations or even speech, are also possible.

While the assessment feature is the most important aspect of the system, many AA systems include also several other features. A system can include database for storing results, user information, or course definitions, one or more interfaces for different types of *users*, authentication and plagiarism checking tools, etc.

The *context* of this thesis is in the field of *computer science education* (CSE) with special emphasis on large courses. A large course in this context is any course where the number

of students is large enough that the course instructor has no way of manually checking all the answers the students produce. Typically at Helsinki University of Technology, large courses can have hundreds of students. The Data structures and algorithms course, for example, typically has over 400 students.

2.1 User Groups

An automatic assessment system typically has several different kinds of users. Pardo [33] lists three (student, professor and grader), while Dalziel [9] lists two (designer and student), Perston and Russel [34] have three (instructor, rater and student) and Higgins, et al. [14] use five (student, tutor, teacher, administrator and developer).

In essence, there are three different kinds of user groups: *students*, *instructors* and system *administrators*. The same conclusion has been drawn in [48]. Students are the most populous user group, who use the system to submit exercises and view feedback. The instructor user group contains all course staff. In several systems, course staff can be divided into different user groups. The division is, however, completely system-dependant. Therefore, in general discussion it is better to include only one user group for course staff. Administrators are the people who are responsible for maintaining the whole system. Therefore, they are a separate user group from instructors, who are responsible for maintaining one course. It should be noted, however, that instructors and administrators are in many cases the same people.

Sometimes system developers are included as a separate user group. A system developer's view of an automatic assessment system is, however, fundamentally different from the other users. Students, instructors and administrators are the end users of the system, and interact with it through an user interface. Developers, on the other hand, implement new functionality and fix faults in the system. They interact with the system on a code-level. In this work, therefore, the point of view of the system developers is mostly left out. Only when describing the new instructor's user interface to TRAKLA2, is the point of view of a system developer touched upon, as the description delves to the code-level details of the system.

In this work, I will concentrate on the point of view of *instructors* and *administrators*. The main focus is in the point of view of an instructor using the system on a large course. The instructor has two goals when using the system. First, he is interested in reducing the amount of work required to administer a very large number of students, who all submit dozens of exercises during the course. Second, he is interested in the data gathered by the system which can be used in course development or other research.

2.2 The TRAKLA2 Instructor's User Interface

The TRAKLA2 automatic assessment system [22] is the successor of the highly successful TRAKLA system [15]. TRAKLA was used at the Helsinki University of Technology between 1991 and 2003, and TRAKLA2 replaced it on spring 2004. When TRAKLA2 was

taken into use, only the students user interface was implemented. Instructors had to use the command-line database interface for system administration. In this work, an instructor's user interface to the TRAKLA2 system is implemented. The interface described here is based on the experiences with a rapid prototype, and the list of functionality gathered from related work and expert opinions.

The development of the instructor's user interface used the *rapid prototyping* software development model. In this model, a prototype of the new system is first implemented in the very beginning of the development process. This prototype is then used to gather experiences for later parts of the system development.

2.3 Evaluating Automatic Assessment Systems

Evaluating an automatic assessment system is problematic, and there is very little literature available on the subject. Valenti et al. [48], however, have suggested adapting the ISO9126 standard for "Software Engineering – Product Quality" [1] for evaluating computer based assessment systems.

Quality in The ISO9126 standard is identified with six characteristics: *functionality, reliability, usability, efficiency, maintainability, and portability*. Each characteristic is further divided into several subcharacteristics. Valenti et al. suggest using the three domain specific quality characteristics, functionality, usability, and reliability, for CBA system evaluation. The rest of the characteristics are left unused, since "they are either domain independent or un-assessable by the end users" [48].

The evaluation of quality cannot be done only based on the quality characteristics. The characteristics must be defined in the context of the evaluation, that is the description of the target systems and the environment where it will be utilized. The systems considered in this work must serve three different user types: students, instructors and administrators. The context is the *instructor's* and *administrator's* point of view to an *Automatic assessment system* used for *computer science education*. The system is assumed to be deployed in a web environment and accessed through a standard web browser.

Chapter 3

Related Work

Today, automatic assessment is used with numerous different types of exercises and there have been numerous assessment systems, ranging from very simple tools used on just one course at a single university to large commercialized products used in dozens of institutions. In this chapter, I will discuss several existing AA systems in more detail. For each system, I will describe the general functionality and purpose of the system and introduce the user interfaces of both students and instructors, if they are present. I will also discuss how the system handles automatic assessment and give an overview of the types of exercises and the way data is stored.

3.1 Exercise Types

The field of automatic assessment is huge, and therefore it would be helpful to categorize the different systems. Carter et al. [8] divide exercises into five categories: *multiple choice questions*, *text answers*, *programming assignments*, *visual answers* and *peer assessment*. In the following these categories are discussed in more detail.

It should be noted, however, that the categorization is by no means comprehensive. The Carter et al. paper is a survey on the use of assessment, and not a taxonomy of exercise types. Furthermore, the paper was about computer aided assessment, and therefore some of the fields described there are not relevant when discussing automatic assessment.

3.2 Multiple Choice

Multiple choice questions are the simplest, and therefore perhaps the most common form of AA. It is easy to create multiple choice questions by using, for example, HTML and Javascript. In such a system, the assessment procedure can be embedded into the questions themselves, and there is no need for a dedicated AA server to check the answers. However, if the questions are to be used in summative assessment, a server to store the students' results

for later use is required. Multiple choice questions can also be used in several situations that have little to do with teaching. They are common in surveys, polls, and trivia contests.

The most common form of a multiple choice question has four alternatives, of which only one is correct. There are, however, several simple variations on this theme. For example, the number of incorrect choices can vary, usually from two up to any number. Also, it is possible to include more than one correct answer, and perhaps a correct one and several semi-correct ones.

Typically a student is rewarded points for the correct answer, while an incorrect choice gives either zero or a negative number of points. If negative points are possible, the student usually has the option of not answering a question. The approach, where there are *almost correct answers*, would require changing this formula to include the semi-correct choices. One solution is to give several points for the correct answer, one for a semi-correct choice and zero or negative points for an incorrect answer.

It is also possible to expand the concept of multiple choice questions beyond the simple approach of having the user select one choice from a number of alternatives. One option is to list a number of terms, concepts, etc., and have the student connect those that are associated to each other. An example of this type of exercise is: “Connect each given algorithm to the data structure it *must* use”.

Assessing multiple choice questions is technically very easy. The assessment procedure takes the student’s answer, the key of the correct answer and the grading formula as input. It compares the student’s answer to the correct one and gives points accordingly.

The simplicity of implementing multiple choice questions has made them a very popular feature in course management systems, as well as other educational software that otherwise have no AA capability. For example, the WebCT¹ system includes this feature.

3.2.1 Älypää

The Älypää² (“egghead”) quiz system is an edutainment system using multiple choice questions. The system was created by Älypää Oy in Finland and has been in use since 1998. Despite being an edutainment system, it is an excellent example of what a good multiple choice question system should be like. In educational context, the system is used by several companies to quiz their employees of company practices, policies, etc. The system is also used at Helsinki University of Technology as an optional part of a data structures and algorithms course.

Since Älypää is an edutainment system, the user who answers the multiple choice questions is referred to as the *player*, not the student. Therefore, in order to be consistent with the terminology used in Älypää, the term player will be used here, as well. Also, using Älypää is referred to as playing, and taking one quiz is one game of Älypää.

Another user group for Älypää are administrators, who manage the system’s question bank.

¹<http://www.webct.com/>

²<http://alypaa.com/>

Since Älypää is an edutainment system, the administrator handles responsibilities that in most systems belong to the instructor in addition to “normal” system administration. There is no separate instructor user group. The player’s user interface to Älypää is based on Macromedia’s Flash framework while administrators use a web browser.

Älypää uses a format similar to the “Who Wants to Be a Millionaire?” game. In the millionaire game, the player is asked a number of increasingly difficult questions, which have four alternatives. Each correct answer increases the amount of money the player earns from the game, and at any time he can quit and walk away with the amount of money he has. Wrong answer ends the game, and the player loses his money. The player also has three “lifelines” during the game that he can use to make a question easier.

In Älypää the player is given a series of 25 multiple choice questions that have four alternatives. One of the choices is the correct one. A correct answer will lead to the next question, until all 25 questions have been answered. A wrong answer ends the game. Each question must be answered within a given time limit, or the game ends.

During one game the player has three “lifelines” which make answering a question easier: he can remove two wrong choices, ask for a hint and skip a question. Each function can be used once per game. During the game, the player is given points for correct answers. The amount of points awarded is based on how many correct answers the player has already given, how quickly the player answers the question, and how difficult the question is. A wrong answer deducts points from the final score. The player can also decide to end the game voluntarily, keeping the current score.

The framework behind Älypää includes several noteworthy features. First, while four alternatives are shown for each question, there is no set limit to the number of wrong alternatives in the database. The system randomly selects three wrong choices to accompany the correct one each time a question is asked. Also, all questions in the system can be ranked according to how difficult they are, and are divided into categories according to the topic they cover. In each game the 25 questions are selected randomly. If desired, questions covering certain topics or of certain difficulty can be selected. For example, first questions are typically easier, and the questions can be limited to certain topic or set of topics. With a sufficiently large bank of questions, these features make Älypää a good system for e.g. informal quizzes.

The questions are maintained by an administrator, who has access to the system through a web interface. Through the administrator interface, it is possible to add, remove and modify questions. Each question contains the correct answer, a number of incorrect answers, difficulty, and category where it belongs. All of these can be modified from the web interface. It is not, however, possible to add, remove or modify difficulty levels or question categories from the web. Also, each question must be modified separately through the UI, and there is no way of doing batch runs (i.e. to remove several questions, or add a number of questions with one command). Furthermore, the administrator interface does not contain a way of accessing any information on results gained by players of Älypää. The system is capable of showing score lists, but it shows only the scores of the top players in decreasing order, instead of giving a comprehensive list that includes all players.

The Älypää system has been successfully used on a Data Structures and Algorithms course

at the Helsinki University of Technology. It was voluntary supplemental material for the course. The pedagogical aim was to make the students interested on the topic at hand, and perhaps make them learn some basic facts. Student response to the system was very positive, and many used it quite extensively. It has not been studied, however, if using Älypää affects the students' overall course results.

3.3 Programming Assignments

In the field of computer science, the automatic assessment of *programming assignments* is perhaps the most classic example of AA. Over the years, there have been dozens of different systems created. The systems vary from simple ones used just in the university where they were created, to widely used professional products. The automatic assessment of programming exercises has been a subject of research for decades. “Grading programs”, as Forsythe and Wirth [13] call their programs that grade students' assignments, were in use as early as in 1961. Also others, including Naur [31], did early work on automatic assessment. The language of choice, in those days, was Algol.

The *programming assignments* category of AA includes all systems that automatically assess some aspects of student-written computer programs. Naturally, there are several different aspects of a programming assignment that can be assessed, and many analysis techniques can be used to assess the same aspect. The earliest assessment systems were simple: the student program was ran with several (possibly randomized) inputs, and its output was compared to the output created by an instructor-provided model program. These *output matching* methods are still used in the assessment of programming assignments, although often combined with other methods [4, 5, 16].

A more advanced method for program assessment is analyzing the internal structure, or the run-time behaviour of the student's answer. Robinson and Trosun [37], for example, did early work in this area by estimating the execution time spent in each program block. More recent examples include Scheme- robo [38] where abstract syntax trees are used in analyzing whether some desired structures were present in the answer, and CourseMaster [14], which has the ability to search submissions for specific features, such as desired type of loops.

Yet another important aspect is assessing programming style, a field that has been researched at least from the early eighties [36]. Style analysis is not concerned with the functionality of the program, but measures how well – meaning in this context how understandable the code is to another human reader – the student is able to write program code and follow widely accepted coding conventions. Today, several automatic systems include style assessment features [2, 4, 16, 30].

The abovementioned three approaches, output matching, structure analysis, and style analysis, are the most common types of AA for programming exercises. Most systems use at least one of the approaches in assessing student performance. Another important aspect to consider is the application area of the programs to be assessed. There are many systems for assessing short exercises on introductory programming courses [38, 4, 16] without specifying any particular problem domain. There are also program assessment systems that are

tied to a specific application domain, like graphical user interface programming [11], or database programming [25, 29]. The following systems encompass the three most common approaches to automatic assessment of programming and, at the same time, highlight a few different approaches to how an instructor can use a system. Because of this, the user interfaces of the systems vary from nearly non-existent to highly complex. The described systems are not tied to any particular problem domain.

3.3.1 ASSYST

ASSYST [16] is an automatic assessment system for Ada programs developed at the University of Liverpool. The system has graphical user interfaces for both *students* and *instructors*, and the instructor has a very large role in the assessment procedure. The system is capable of independent, completely automatic assessment, but is generally used only under instructor supervision.

The student's user interface to ASSYST is very simple, and merely allows the student to submit his solution for assessment. It is assumed that the student submits his own test data with the program. Feedback for the assessment is not immediate, since the instructor has to explicitly start the assessment procedure, and it is assumed that the instructor is present when the system assesses submissions. Feedback for the submission is sent to the student through email.

ASSYST uses measurements in five different areas to assess the submissions: correctness, efficiency, complexity, style, and test data adequacy. In correctness analysis ASSYST checks that the submitted program can be compiled and that it contains the required functionality. Functionality is assessed using pattern matching to examine the output of the student's program. The instructor provides a specification for the expected output and the system uses the Unix Lex and Yacc tools to create a program that checks whether the output of the submission conforms to the specification.

There are two methods for checking the program efficiency in ASSYST. First, the system can observe the CPU time used by the program, and compare it to the execution time of a model program. However, since many submitted programs on basic courses are tiny, the execution times can be so small that observing the CPU time might not give an accurate picture on how efficient a student's answer is. Therefore, ASSYST is able to count the number of statements executed by the program. By comparing the statement count of a student's answer to the count provided by the model answer, it is possible to estimate the efficiency of small programs.

Program complexity analysis in ASSYST is based on the size of the flow control graph of the program, using McCabe's metric [28]. The metric defines the cyclomatic complexity of a program as $V(G) = e - n + 2$, where e is the number of edges and n the number of nodes in the flow control graph of program G .

Style analysis in ASSYST is based on a metric originally created for the C language and adapted to Ada by the authors of the system. The metric measures several aspects of style, including module length, amount of comments, indentation, etc. Each measured aspect has

a weight value assigned to it and the total style value is combination of the values.

Finally, the student has to submit test data along with his program. ASSYST runs the submitted program with the accompanying test data, and measures the statement coverage given by each test. The adequacy of the test data is measured by dividing the number of statements executed with the total number of executable statements.

The assessment process is started and controlled by an instructor. The instructor can access ASSYST through a mouse-driven graphical user interface. Through the interface, the instructor may view courses, student information, and the students' submissions. He can also grade each submission using ASSYST's assessment modules. After automatic assessment the instructor may modify the grade, add his own comments and send the results of the assessment to the student through email. The available literature did not specify how new courses are added to the system, how the instructor can modify existing courses, or how the submissions and results are saved on disk.

3.3.2 Ceilidh

Ceilidh [39] is an automatic assessment system for C programming exercises, originally created at University of Nottingham. Ceilidh was later adopted and modified in several institutions around the world. In Nottingham, the system was developed further and eventually renamed CourseMaster [14].

Originally, Ceilidh was a collection of C, C++, and awk programs for UNIX. These programs were called through an user interface developed for VT-100 terminals, which were common in Nottingham those days. Later, other user interfaces, including graphical ones, were created for other platforms.

The system has four different categories of users: students, tutors, teachers and administrators (called "course developers") where students have the most restricted and administrators the least restricted access to the system [4]. Each user category has all the features of the more restricted categories available to them. Therefore tutors have all the student-level features available for them, plus some new functionality.

Through Ceilidh a student can view course information, including deadlines for exercises, exercise specifications, and outlines for program exercises. He can also use the system to submit his exercise for assessment, and view the results. Typically, a student can submit a single exercise for multiple times. Finally, the model answer is available once the deadline for the exercise has passed.

The system is capable of several different types of evaluation to student programs. The different types of assessment included in the system are *dynamic correctness*, *dynamic efficiency*, *typographical analysis*, *complexity analysis* and *structural weakness*. Any subset of these can be used to evaluate the student submissions.

The dynamic correctness and dynamic efficiency evaluate the run-time properties of the program. Correctness evaluation uses output matching. The submitted program is run with a set of test data, and output is compared to specified model output. The model output

is not an exact model that strictly states what is required. Instead, a regular expression is used to specify the model output. This makes the system more flexible, and gives the students more freedom in constructing their programs. Also, using regular expressions allows the students to focus more on the functionality of programs instead of worrying about the correctness of the output. Efficiency evaluation is based on *program profiling*, where unexecuted code is spotted, number of loop iterations counted, etc., and results are reported after the program has terminated.

Typographical analysis is used to rate how understandable to program is to a human reader, and evaluate the programming style used. Typographical analysis measures how the program source code is presented to a human reader, including consistent and understandable indentation, identifier name lengths, amount of comments, average length of program modules, etc.

Complexity analysis evaluates the static complexity of the program by counting the frequency of certain program structures, like conditionals, loops, library functions, etc. The complexity of the student solution is compared against the model solution. Structural weakness analysis searches for unused variables, return values and other unused program parts. It uses the Unix C lint utility to find the problems in the students' code.

The instructors, tutors and teachers in Ceilidh terminology, can view the students' submissions through the user interface, observe the current course situation, including information on which exercises students have submitted, see grading, and the attendance to live meetings. In addition, teachers can create new exercises, modify grades given by automatic assessment, get some statistics for the course, and run plagiarism checks. The ceilidh user interface calls underlying Unix scripts and programs in order to perform the required functionality. This makes it also possible for the instructors to browse the system from command-line, instead of using a specific user interface. Furthermore, it is possible to use Unix scripting languages to produce new functionality.

All data in Ceilidh is stored straight to the file system. Each course in the system resides in its own directory, and each round has its own subdirectory. Exercises, in turn, reside in the subdirectories of rounds. Configuration files are used to create exercises and stored in the same subdirectory as the exercises. Student submissions to an exercise are stored in the filesystem, in a subdirectory of the exercise directory. Typically only one submission for each student is stored by the system, and it is overwritten when the student submits his work anew.

3.3.3 Style++

The Style++ [2] tool automatically measures programming style for C++ programs. It was created at Tampere University of Technology, and is currently used there on several C++ programming courses. Unlike many other systems, Style++ is just an assessment tool and does not include any functionality related to course management. Instead, it can be included into other automatic assessment systems that already contain this functionality. For example, on some courses at Tampere University of Technology Style++ is incorporated

to the Ceilidh system. On these courses Style++ is used to assess the programming style, while Ceilidh assesses other aspects of the program.

The Style++ tool is implemented for the UNIX environment, and can be used either from command line or through a graphical user interface. The output of the tool intentionally resembles the output of a typical compiler in order to make the program more attractive to new users. This is based on the assumption that compiler is the basic tool of programmers, and a system that gives output similar to compiler output is therefore easier to learn and understand. The output given by Style++ can be configured to include several different types of feedback. The output can include warnings, or instructions for the stylistic changes the student should make. It can also show the grade for the submission, compare the given score to the maximum, or include information on how submissions are scored.

At Tampere University of Technology the tool is used in two ways. First, the students can use the tool during program development to see if their programs fulfill the style requirements of the course. The tool is installed on all UNIX machines at the university, and the Style++ configuration file defining the requirements for the course is available. Second, the tool is used as a part of program assessment, either in cooperation with other automatic assessment tools, or as “pre-assessment”, where the submission has to pass Style++ analysis before being given to a human tutor.

Assessment in Style++ covers 64 different measurements of programming style. These measures include several modularity, typography, clarity, simplicity, independence, effectiveness and reliability aspects of the program. Not all measurements need to be used. Using a configuration file, it is possible to tell Style++ which measurements are used and how the submission should be measured (for example, how long variable names should be, or how large percentage of comments the code should contain). It is also possible to use the configuration file to specify the feedback messages given to the user. If no special configuration is given, the tool uses the default values meant for the introductory programming course.

The Style++ tool is based on the C++ Front End program that is a C++ compiler created by Edison Design Group. C++ Front End compiles the given program to a machine-independent intermediate language. It can also check the program syntax and semantics, and can give versatile feedback. The compiled program also contains a large amount of information of the original code, making it possible to trace errors in the intermediate language back to the original source.

Since Style++ is not designed as a stand-alone system for summative assessment, there is no instructor’s user interface as such. The job of the instructor is to decide how the tool is used to assess students’ programs (that is, to write the configuration file) and, if used in cooperation with other assessment systems, integrate Style++ into the automatic assessment procedure.

3.3.4 Scheme-robo

Scheme-robo [38] is a system for automatic assessment for programming exercises using the Scheme programming language. The system was created at Helsinki University of Technology for use on the first course on programming. In the strictest sense, Scheme-robo is only an automatic assessment system and does not contain any course management functionality. At Helsinki University of Technology the management core of the TRAKLA [15] system was used for course management tasks. Here, however, Scheme-robo system is described as it was used at Helsinki University of Technology, course management features included.

There are two user groups in Scheme-robo: students and instructors. In Scheme-robo the students' user interface is completely email-based, and students interact with the system by sending emails in pre-defined format. Exercises, which are grouped into rounds, are sent to the students in an email once the round opens. Students solve the exercises, and send their submissions to the system by email. The system responds by sending the feedback and points to the student. For each exercise, there is a limited number of allowed submissions, and the highest score attained will be the student's final score for the exercise. The students can also request Scheme-robo to resend a certain round, or ask for their current point totals.

In a typical Scheme-robo exercise, the student must develop one Scheme function instead of a complete program. The main method of assessing the submissions is by executing the student-made functions in a Scheme interpreter, and examining the return values. The submissions are executed in a safe sandbox specifically created for this purpose. The sandbox prevents the assessed function from running infinitely by monitoring the executing time, and excludes file I/O and other features that might compromise the security of the assessment system. The assessment is very similar to output matching done by e.g. Ceilidh. However, instead of matching printed output, the assessment procedure compares function return values. This moves the focus of the exercises from producing program output, which is not pedagogically very interesting, to creating correct program functionality.

Most of the test runs in Scheme-robo are pre-defined. The system executes a student's answer and compares the result to a given model output. Random tests, where the output of the student's submission is compared to the output of an instructor-provided model answer, can also be used. The features of Scheme and the Scheme interpreter used by Scheme-robo enable exercises where the student is required to submit a function that is part of a larger program. The assessment procedure can then load the student's function along with the rest of the instructor-provided program, and test the whole.

In addition to test runs, Scheme-robo is also capable of examining the structure of the submissions. The structure analyser checks the submission for certain keywords, in order to make sure that the student's submission does not include Scheme commands that are not allowed for the exercise. For example, if the exercise is to reimplement the Scheme `reverse` primitive, the primitive itself must not be used in a submission. Scheme-robo is also capable of examining the structure of a submission by converting it into an abstract syntax tree, which is a very simple procedure in a Lisp-like language. The tree can be then examined for certain subpatterns. This is usually used to make sure that the student has used the exercise

skeleton provided by Scheme- robo. In addition, the system includes plagiarism detection, which also uses abstract syntax trees. The plagiarism detector compares all exercises, instead of just one exercise. The code to a single exercise is so small that similar solutions are common, and therefore trying to use single exercises for plagiarism detection would not work. If a pair of students have many similar submissions the system alerts the instructors, who manually check the submissions and have the final say in the matter.

There is no instructor's user interface, as such, in Scheme- robo. In the position of an user interface, there is a number of Unix shell scripts and awk programs, that are used from standard Unix shell. Using the scripts, it is possible to create new courses, generate reports from the system, see student results, modify grading and handle special cases. Further functionality can be achieved by creating new scripts.

The state of the assessment system is contained in the file system, and if the system halts for some reason (i.e. power failure), it can continue the assessment from the same place. Only those operations, for which the results were not stored on the disk, must be redone. Exercises in the system are defined using configuration files, which include exercise description, possible skeleton for submissions, assessment principles, test runs, etc. Since many exercises can share similar assessment principles or test runs, a configuration file can include other configuration files, in a style similar to the include -functionality of the C language. This will remove the need for copying configurations, and the problem of maintaining the same information in several places. Student submissions, results of assessment, etc. are also stored in the file system.

3.4 Visual Answers

In automatic assessment systems supporting *Visual answers*, a student manipulates visualizations in order to create an answer. This category covers a wide range of visualizations including automata and other formal constructs [46], data structures and algorithms [22], UML diagrams and flowcharts [14], etc. Unlike other categories in the classification, *visual answers* are defined by the medium (visualization) through which the student solves the problem instead of the type of exercise to be solved. Therefore, for most subjects, it is possible to create AA systems that test the same topic without using visualizations.

Data structures and algorithms (DSA) is one of the most central topics in computer science, and the basics of DSA are taught to all CS students. However, it seems that in most institutions DSA is taught as a part of a programming course (typically the second course on basic programming). Therefore, only a few AA systems that are designed to assess DSA on conceptual level instead of implementation level exist [6, 12, 15, 22]. If DSA is taught on a programming course, the exercises are typically small programs, and their assessment falls into the *program assessment* category.

When DSA is taught on the conceptual level, the focus is not in the implementation of the data structure, but on how it works on a more abstract level. Therefore, the student might be given a visualization of an empty binary tree, and tasked with showing how keys are added to and removed from the tree. In essence, the student simulates the algorithm by placing

the keys to the correct positions in the tree visualization.

Another aspect of computer science education where visualization has been used extensively is theoretical computer science. Through visualization exercises on different theoretical entities like *formal languages* and corresponding *automata*, *semantic/analytic tableaux trees*, etc., can be made much easier for student to comprehend [3, 10, 17, 45]. When teaching such systems, the visualizations are typically used to show the structure of the entity at hand, and the student is tasked with demonstrating how it works. The student might also be given one entity (i.e. a language) and tasked with constructing a corresponding other entity (i.e. an automaton). In systems like these, visualization is typically used to make it easier for the student to grasp the subject matter. Visualization has the additional upside that typographical errors are not possible when manipulating a visualization. Therefore, the use of visualization makes the problem easier to understand, and may also prevent student from doing trivial mistakes.

Visualization can also be used in solving exercises related to formalisms that can be shown as diagrams. UML diagrams are an example of this. For example, the CourseMaster system [14] is capable of automatically assessing UML class diagrams, flowcharts, and logical circuits. Closely related to CourseMaster is also the DATsys system, which is a framework for automatic diagram assessment [47]. Important in automatic assessment of diagrams is to be able to separate the visualization (the diagram itself) from the data it represents, since typically the instructor is concerned with the correctness of the model and not the clarity of the visualization.

3.4.1 TRAKLA

TRAKLA [15, 21] is an automatic assessment system for data structure and algorithm exercises, created at Helsinki University of Technology. There, the system was used as a compulsory part of the basic DSA course between 1991 and 2003. After spring 2003 the TRAKLA system was replaced by a newer, more flexible one.

In TRAKLA, students can solve the exercises either through email-based user interface or by using visualizations in the World Wide Web. A text notation is used to portray the required data structures in the email UI. In the exercises, the students have to manipulate the given data structures and return their results for evaluation. Typically, only the final stage of the data structure is returned for evaluation, but sometimes one or two intermediate stages are also required. The exercises are tailored with different input for each student in order to prevent plagiarism.

Originally, there was only the email-based user interface, until the graphical UI, called WWW-TRAKLA, was created in 1997 [20]. WWW-TRAKLA is a graphical front end to TRAKLA that accepts the input data structures and exercise descriptions for a TRAKLA exercises, and visualizes them for the student using a Java applet. Using the applet the students can solve and submit their exercises through the Web. In addition to visualization, WWW-TRAKLA allows the student to manipulate the data structures by dragging and dropping elements. The WWW-TRAKLA front end does not change the assessment in

any way, and the students are still free to use the original email-based user interface. The WWW-TRAKLA web environment is simple. On the main page the student selects the exercise and gives his student id number to a web form. The system parses this data and produces the required applet. In order to solve another exercise, the student has to refill the form. There is no session control.

The students return their answers through email or the Web for automatic assessment. The answers are assessed by comparing the student's answer to a model answer produced by a real algorithm using text comparison techniques; each exercise has its own assessment algorithm. The assessment takes the exercise and the student's answer to it as input. It solves the exercise, and compares the model answer to the student's submission using various heuristics. The heuristics can spot some simple errors, and can therefore recognize some correct answers that include, for example, a typographical error.

Originally, the exercises were assessed only once, when the deadline had passed. After assessment, points, possibly some textual feedback on mistakes, and the correct answer were sent to each student. Later, the possibility of multiple submissions was introduced. In the current version the submissions are assessed almost immediately (the assessment script is run ever fifteen minutes), and feedback consisting of a grade and possibly additional textual feedback is sent to the student. Submissions are accepted until the deadline, or until the maximum number of submissions is reached. The best submission is chosen as the final points for each student. After the deadline has passed, the correct answer is sent to the student.

In TRAKLA, courses are divided into rounds that contain one or more exercises. Each round has its own deadline, and the number of rounds per course, and exercises per round, is unlimited. The exercises are selected from an exercise bank. There is no real user interface for instructors and administrators, and they use a number of command-line scripts and configuration files to set up courses, receive student results and solve special cases. All data is stored in text files that are processed by scripts in order to create course results, statistics, etc. For example, a course is set up by creating a new directory for it, creating a configuration file which defines the course, and adding this new course to the list of courses in the system.

3.4.2 TRAKLA2 Prototype

The TRAKLA system was succeeded by TRAKLA2 [26] in spring 2003. By that time the limits of the TRAKLA system had become obvious. For example, in TRAKLA a student's submission usually consisted of *only* the final stage of the data structure. The assessment therefore just checked if the student had reached the correct final configuration, and could not check if the student had used the correct algorithm to reach this configuration. However, when trying to learn data structures and algorithms, it is important to learn how the algorithm works, and not just be able to produce the same final results.

Also, it was a hard and time-consuming task to create new exercises for TRAKLA. The system was implemented in C and no system-specific development tools existed. In addi-

tion, there was no common assessment logic, forcing the developer to create the assessment for each exercise from ground-up. Moreover, the TRAKLA namespace could contain only 100 exercises.

TRAKLA2 includes several improvements over TRAKLA, but main idea is the same: the student is given a data structure and has to simulate an algorithm on the structure. TRAKLA2 exercises, however, are only available through an applet, and there is no email-based user interface. The student uses graphical manipulation to modify the data structures and to simulate the functionality of the required algorithm.

Upon completion, the student submits the whole simulation sequence for evaluation and receives immediate feedback from the system. Exercises in TRAKLA2 are assessed using a *generalized assessment procedure*, which takes as input the student's submission and the model answer. The procedure compares these two and tries to find identical states.

TRAKLA2 exercises are developed using the Matrix framework [23], which contains several ready-made data structures that can be automatically animated and visualized. The framework also includes extensive support for the development of exercises. It contains, for example, a number of interfaces that can be used to implement user interface functionality to the exercises.

The TRAKLA2 prototype was used only on the spring 2003, when the new exercises were run in parallel with exercises of the old TRAKLA system. Therefore, the same web environment was used for both systems. As typical for the Data structures and algorithms course, the exercises were split into several rounds. On most of the rounds the students used the old system, but there were two rounds on the course where the new TRAKLA2 exercises were used. The WWW user interface to the two systems was identical. The only difference between the old and new exercises was the applet used for solving the exercise.

The TRAKLA2 exercises used the generalized assessment procedure, but the course management was done using the TRAKLA server. The server, however, was never meant to work with the new exercises, which led to a number of problems. The new exercises were added to the TRAKLA server using a number of ad-hoc solutions because of the incompatibility of the two systems. For example, the new applet had to be fitted into TRAKLA web pages.

The instructor's user interface to the TRAKLA2 prototype was identical to old system, since the prototype used TRAKLA's course management features. The old system's student's web user interface, however, was not meant to be a stand-alone system. While the student could solve exercises through the web, he could not view his points there, and had to ask for results list through the e-mail interface. Therefore the TRAKLA system was not ideal for new exercises, which did not need the email interface. The TRAKLA server was replaced by a new web environment created specifically for TRAKLA2. This new environment was used for the first time in spring 2004.

3.4.3 Stratum

STRATUM [17] is an automatic assessment system for teaching logic, automata, grammar and other formal systems in computer science. The system was developed at Helsinki University of Technology for use on several theoretical computer science courses. Later, it has been adapted for teaching other subjects, most notably compilers. The system is currently used for Smullyan's analytic tableaux [42], finite automata, regular expressions, and compiler assignments.

The STRATUM-system was created for distributing personalized home assignments through WWW and automatically assessing the student submissions. Submissions are returned through email, but typically STRATUM is used with a separate tool for solving the exercises. The tool contains functionality for submitting the exercise with a single command and thus the students do not need to know which email address to use, what kind of mail to write or other specifics of the submission mechanism.

The student's user interface to STRATUM is divided into two separate parts: WWW interface through which to fetch the exercises and check the results, and a tool for solving and submitting the exercises. Each student has their own password-protected page under STRATUM. User id and password for accessing the page are sent to each student at the start of the course. From the web page the student can fetch his exercises, view submitted answers and, if his submission was incorrect, view feedback from the assessment system. In order to prevent plagiarism the exercises are typically tailored for each student.

Smullyan's tableaux are solved using a tool called TABEDIT. It is a stand-alone tool that provides a graphical environment where the student can load an exercise, and use the various rules of Smullyan's method to either prove or disprove the given logical tableau. Rules are selected from a list of tableau rules. If a correct rule is chosen, the system applies it to (selected part of) the tableau. Incorrect rules cause the program to wait for a while. The wait time increases exponentially with each mistake in order to prevent the student from using the trial-and-error problem solving method. TABEDIT also contains functionality to save and load the assignments, making it feasible to leave an exercise unfinished and continue it later.

In the case of tableaux, assessing the students' submissions is rather straightforward. Solving an exercise is computationally challenging (it is a **coNP**-complete problem), but checking whether a given proof is valid is straightforward. A submission can be verified by visiting all nodes in the proof in order to see whether tableau rules were followed and checking that each terminal node is marked as either contradictory or non-contradictory. In case of incorrect proof, the system sends feedback to the student.

In STRATUM the instructor's user interface is a number of Unix shellscripts. Using the scripts it is possible, for example, to add and administer students, gather statistical data and results. Some administrative tasks are also done by modifying configuration files. The instructors access the system through UNIX shell, and access control is done by the operating system. Automated tasks, like assessing exercises and assembling results lists are run periodically. Each student's information is stored in a separate directory that contains his exercises, answers, results of the automatic assessment and a log of all actions. Similarly, different

course instances reside in their own directories.

3.4.4 Exorciser

EXORCISER is an interactive learning environment for teaching the basics of theoretical computer science, created at the Swiss Federal Institute of Technology [45]. The focus of the system is on making it possible for the student to receive a large amount of fine-grained feedback.

In EXORCISER, the student can specify how much help he wants the system to give, and select the difficulty of the exercises. The system is used and exercises solved through a graphical user interface. Exercises are solved by graphical manipulation of the required entities, which may be automata, strings, etc. The system can, depending on the level on feedback selected, either notify the student each time he makes a mistake, give feedback only when the student thinks he has solved the exercise, or upon request. The student can change this level of feedback whenever he likes. The student can also ask for assessment whenever he likes, and may continue to solve the exercise or start anew after viewing the feedback given by the system.

Assessment in EXORCISER is mostly formative and designed to help the student correct problems in his solutions. It does not assign points to the solutions, like many other automatic assessment systems do. Assessment in EXORCISER is based on the idea of *solution space*, which contains all the solutions of a single exercise, including the numerous incorrect ones. For each exercise there is a grading mechanism that tries to catch all errors occurring in a submission. Each error caught raises a condition, which can then be used in the creation of feedback. The exact assessment method is different for each exercise in the system. The system has “grading language”, which uses a number of *if* · · *then* rules to specify the feedback. The conditions used by the grading language are raised by the assessment mechanism.

Since the EXORCISER system is not used for summative feedback and student grading, there is no need for the instructor to monitor the course progress or see student submissions. Instead, the instructor needs to group the exercises in the system into a set that can be used as a part of the course, typically by tying them to a certain lecture or a set of lectures. The exercises are implemented in Java and can be configured using XML configuration files. The configuration of the exercises includes writing the feedback generation schemes and grouping exercises into larger groups, called *exercise chains*. In an exercise chain, the student is given a new exercise based on his performance in the previous one. For example, if it is clear that the student has not yet grasped how to solve the exercise, he is given a new instance of the same problem. Otherwise, the student is given a new exercise, which may be selected from a number of options based on the student’s past performance.

3.5 Text Answers

Text answers category according to Carter et al. includes both short text answers, where the student is required to fill in blank spaces in text, and longer essays. The automatic assessment of textual answers is a very complex problem that has been studied for decades. (According to Williams [50] there was research on the subject as early as in middle sixties [32].)

Even simple typographical errors, for example, can be a challenge to automatic systems. Furthermore, assessing natural languages introduces a lot of challenges that are much more difficult than simple typographical errors. Even if the student is required just to fill in single words in the text, things like different dialects (in the English language, for example, many words are written differently in British English and American English), synonyms, etc. make the analysis of the answers complicated. Essays, where the text to be assessed may be hundreds of words long, again introduces new challenges and is a topic of much active research [7, 19]. However, the automatic assessment of natural languages is a very wide topic and therefore detailed consideration of *textual answers* is out of the scope of this work.

3.6 Peer Assessment

In *peer assessment*, the student's submission is given to his peers for assessment, and the feedback is given back to the student. With large class sizes, this task would take a lot of time if done manually. Therefore automated systems have been developed, and this feature is included in many course management systems. However, peer assessment is not AA, since the assessment is not done by a computer. Therefore this category is out of the scope of this work and will not be discussed further.

3.7 Conclusion

There are numerous automatic assessment systems that are capable of evaluating exercises in very different fields. The systems vary from single-purpose tools like Style++, to commercial systems like CourseMaster that support several kinds of automatic assessment.

The functionality offered by the different systems vary, depending on the purpose of the system and the choices made by the developers. Unfortunately, most of the literature concerning automatic assessment systems describe the teacher's functionality only briefly, or not at all. In the following, I will list some of the functionality found in the systems. If possible, I will try to include functionality that the systems are explicitly stated to include. However, if it is obvious from the description of the system that it includes some functionality, it is also mentioned here.

Creation and modification of courses and exercises. On some systems the instructors can create or modify courses. This allows the instructors to adapt to situations where the

course does not go as planned by e.g. modifying the deadlines. Alternatives to creation and modification through instructor UI are separate administrator UI, or the use of configuration files to represent courses. Some way of creating and modifying courses is present in all systems where exercises can be grouped into courses.

Configuring the assessment. In some systems, the assessment procedure can be configured, and in some systems this configuration may be modified through the instructor UI. Many systems, however, use only configuration files and do not include a feature to change the configuration through instructor UI. Assessment configuration through an UI is available in Exorciser. Also, in Ceilidh, Scheme- robo and Style++, it is possible to configure assessment through configuration files.

Viewing a student's answer. With this functionality the instructor may call up a list of a single student's answers and view them in more detail. Depending on the system, the instructor may also be able to access the model answer to the exercise, feedback given to the student, or the list of errors the student's answer includes. If students can submit their exercises multiple times, the system typically shows all submissions. This functionality is included in ASSYST, Ceilidh, Scheme- robo, TRAKLA and Stratum.

Modifying assessment results and feedback. Some systems allow the instructor to modify the results of automatic assessment, or write their own notes to the feedback before it is sent to the student. Modifying the assessment can also be done after the results have been sent to the student if, for example, the student and instructor agree that assessment has been unfair. Modifications to the feedback typically must be done before it is sent, and this feature is more often implemented in systems where the assessment is computer assisted, or where the automatic system assesses only some aspects of the answer. This functionality is included in ASSYST and Ceilidh.

Viewing student list and course results. With this functionality, the instructor can call up a list of students registered to the course. Some systems are capable of showing only the list of all students on the course, while others may filter the list in order to see e.g. only those students who are in a certain exercise group. The list may show only the student's identities, or can include for example a summary of their current status on the course. The summary can show exercises returned or assessed, current points or grade, etc. This functionality is included in ASSYST, Ceilidh, Scheme- robo, Stratum and TRAKLA. Älypää includes similar ability to see the players' results.

Results maintenance. In automatic systems most of the students' answers are never viewed by a human instructor. Therefore the systems typically are capable of various results maintenance tasks beyond automatic assessment, such as checking that all answers have been sent before deadline. Perhaps the most often mentioned such task is running plagiarism checks, which are included in several systems. Running the check is, however, a heavy process that drains a lot of the system resources. Therefore, in many systems, plagiarism checks need to be initiated manually. Some functionality for results maintenance is included in TRAKLA, Assyst, Ceilidh and Scheme- robo. Ceilidh and Scheme- robo include also plagiarism checking. In TRAKLA and Stratum plagiarism checking was not needed, since each student is given a unique problem instance.

Generating statistics and reports. Often an automated system is capable of generating some kind of statistics or reports. The most common report gives the results of all the students on the course and is typically used to compile course results to final grades. Other possible reports can be as simple as log files on system use, or as sophisticated as comparing the results of several courses. Statistics are available in Scheme-robo, Stratum and TRAKLA.

Viewing student lists, and students' answers, modifying assessment and feedback, results maintenance, and generating statistics are useful for *course and system management*, while creating courses and modifying the assessment are more needed for *system configuration*. If the system is used for summative assessment, it usually includes at least some course management functionality. Systems like Exorciser, which are used for formative assessment and self-learning do not require these functions and may instead include more sophisticated facilities for course configuration.

3.7.1 Instructor's User Interfaces

The sophistication of the instructor's user interface can vary a lot. The UI may be a simple command line interface, a dedicated text-based user interface or a graphical one. Also, it may be a superset of the student UI, or have a completely different set of functions. The UI can include course management functions, be purely aimed for system and course configuration, or include functionality required for both. It is also possible to construct a system where the instructor needs no UI, as proven by Style++.

Most of the reviewed systems are designed to be used to assess exercises that will affect the final course grade. Thus, the systems must be capable of summative assessment and need to store student's results. Typically such systems also include some additional *course management* functionality, like checking current course results, or viewing students' answers. In systems designed for self-study (i.e. Exorciser) or systems that can be included in a separate course management system (i.e. Style++), such functionality is not present.

UIs can also be divided into graphical and text-based interfaces. In the surveyed systems, Ceilidh, Scheme-robo, Stratum and TRAKLA have text-based instructor's interfaces while Älypää, ASSYST and Exorciser have graphical UI's. Style++ has no instructor's user interface. From this very limited selection of systems, it seems that both text-based and graphical UIs are common in automatic assessment systems. Also, it is possible that the different user groups have different user interfaces. In TRAKLA and Stratum, for example, students have a graphical web-based UI while instructor access it through Unix command line.

Chapter 4

Interviews

In addition to research on other automatic assessment systems, a number of experts were interviewed on the subject of teacher's point of view to automatic assessment. There are two main reasons why these interviews were conducted. First, the amount of information on the instructor's user interface in the literature was limited. Second, the literature seldom describes the practical side of administering an automatic assessment system.

Most of the material for the related work section was gathered through literature research. However, most automatic assessment papers give much more space to describing the assessment features and the student's point of view, than telling how the instructor interacts with the system. Therefore, the description of instructor's interface is typically very compact and contains only the description of those features that the authors think are the most interesting in a *scientific publication*. These features may, in fact, not be those that are most commonly required in practice. For example, the functionality for searching for students, which is very important on large courses, is not mentioned in the literature. Perhaps it is considered too obvious or trivial and therefore not included in system descriptions.

In interviews, it was possible to concentrate on the practical use of automatic assessment on courses, and try to find out the exact features and functionality which instructors think to be most important in their daily work. The interviews were patterned after the standardized open-ended interviews in Patton's classification, as described on page 181 of [18]. This means that all interviews were conducted similarly, and each interviewee was asked the exact same questions. A total of 10 questions were asked in each interview. The questions are as follows:

1. What automatic assessment systems have you used?
2. Did the systems have other instructors of the same course using them?
3. What functionality of automatic assessment systems have you used in your work?
4. Was there some functionality missing from the systems you used?
5. Which functionality were the most important in your opinion?

6. Which functionality were the least important in your opinion?
7. What problems did you encounter while using the system?
8. Are you aware of other automatic assessment systems?
9. Which if the following functionality are most important in your opinion?
10. Is the list of functionality comprehensive?

For the last two questions the interviewer showed the list of functionality described in Section 3.7. The lists contains the following categories:

- Creation and modification of courses
- Configuring the assessment
- Viewing a student's answer
- Modifying assessment results and feedback
- Viewing student list and course results
- Results maintenance
- Generating statistics and reports

In the context of this work, the third, fourth, fifth, and sixth questions were the most important, as they were about the functionality of the systems the interviewees used in their work. In addition to the questions listed above, the interviewer also asked some additional questions when he felt it was required for example, to clarify something the interviewee said.

The conducted interviews differ from the standardized interviews in two ways. First, the wording of the questions was not *exactly* the same in each occasion. Second, while most of the interviews were conducted in person, a few were done over the phone. The first difference was due to the inexperience of the interviewer, who was not able to keep the wording constant in all interviews. The second difference was due to practical considerations as the interviewer did not have the resources to do all the interviews in person.

These inconsistencies in conducting the interviews are, however, minor, and do not invalidate the results. The main purpose of the interviews was to find out what kind of functionality instructors have used in their work, what functionality they consider important, and what functionality have they found the systems lacking. The exact wording of the questions or the medium through which the interviews were conducted are unlikely to alter these results.

4.1 Interviewees

A total of 12 people were interviewed for this research. The interviewees were all either instructors using automatic assessment, or developers of automatic assessment systems. Several people had developed the system used on their own course. All the interviewees worked on the field of computer science. Six interviewees were members of the Laboratory of Information Processing Science at Helsinki University of Technology (TKK). The rest worked at different TKK laboratories, or at other Finnish universities.

The selection of interviewees brings at least two different bias to the results. First, all the interviewees were computer science professionals working as teachers in Finnish universities. This bias does not distort the results, as the aim of the study is to find features required by computer science teachers. Second, half of the interviewees are from the Laboratory of Information Processing Science at Helsinki University of Technology. The people from the same laboratory are more likely to have similar functionality lists than people from different institutions. Each laboratory has its own culture and traditions how to teach, which will inevitably reflect in the way automatic assessment is used in teaching, and therefore affect the functionality considered important by the instructors. Also, the people in the same laboratory are more likely to use the same automatic assessment systems.

In qualitative analysis of the interview results, where we are more interested in the number of the different answers given, the TKK bias is less important than in quantitative analysis, where the interest is in seeing how many times each answer was given.

4.2 Conducting and Analysing the Interviews

Most of the interviews were conducted personally, with the interviewer and interviewee sitting in the same room. Most of the people from the Laboratory of Information Processing Science were interviewed in the interviewer's study, while interviews of the personnel in other TKK laboratories were done in the interviewee's study. Interviewees who lived outside the Helsinki Metropolitan Area were interviewed by phone, since the interviewer did not have resources to travel to make the interviews in person.

The in-person interviews were recorded using a laptop computer and a microphone, and stored in the mp3 format. The interviewer also made notes during the interviews. The telephone interviews were not recorded due to technical difficulties. During the phone interviews, the interviewer made notes, which were expanded after the interview in order to preserve the interviewee's original meaning as faithfully as possible.

The interviews were analysed from the notes made by the interviewer, and from the recording of the interviews, when available. In the analysis the goal was to find all the functionality the interviewee used in his work and the functionality he had wanted, but did not have.

The interviews were first analysed by noting each functionality mentioned by the interviewees. Instances where different interviewees mentioned the same functionality were marked, and a list of different functionalities was made. The functionalities in this list were

Table 4.1: Interview Results

Functionality
Examination and modification of assessment results
Viewing a student's answer
Viewing the results of automatic assessment
Viewing the model answer
Giving and adjusting points manually
Manually starting automatic assessment
Searching
Searching for a student
Searching for a specific answer to an exercise
System maintenance
Adding students to the system
Controlling user access
Adding and modifying exercises
Adding and modifying courses
Adding and modifying exercise rounds
Modifying and configuring the assessment procedures
Result viewing
Viewing the course results list
Viewing the results of a single student
Exporting the course results
Exporting the answers
statistics
Logs
Statistics

then combined into larger categories by the author, where functionalities that were closely related (i.e. viewing a single student's answer and the results of the assessment of the same answer) were put into the same category.

4.3 Results

The results of the interviews are summarized in Table 4.1. The functionalities are grouped into categories, which are shown with boldface in the table. The number of times each functionality was mentioned and the relative importance given to each functionality by the interviewees was also recorded. However, these results are not shown in the table because of the bias caused by the selection of the interviewees. In addition, many interviewees did not mention a specific functionality by name, but indicated that they had used it. For example, one interviewee mentioned that it was important to be able to modify rounds. He probably thought that modifying courses was also important, even though he did not say

this.

A large number of different functionality was mentioned in the interviews, and the features ranged from very specific, like being able to give a single student more submissions, to the very broad, like having a better user interface. In analysing the interviews, the functionalities were grouped into five loose categories. The categories are roughly equivalent to the functionality discovered in related work and described in Section 3.7.

All of the categories in the Table 4.1 are divided into several different functionalities. The different functionalities, in turn, may contain several individual actions. The *adding and modifying exercise rounds* functionality, for example, includes adding exercise rounds, opening and closing rounds and changing round attributes.

4.3.1 Examination and Modification of Assessment Results

The category examination and modification of assessment results contains all functionality that the instructor requires for viewing the students' answers and the system's assessment of each answer. It also includes functionality for manually adjusting the results of the automatic assessment.

Typically, the functionalities in this category are used when a student has questions about the assessment of his submission. In order to answer student's questions, and check the validity of the assessment procedure, the instructor requires access to the **student's answer**, the **results of the the automatic assessment** and, if present, the **model answer**. It should be noted that in many systems a student can submit several answers to a single exercise, and therefore the system should enable the instructor to differentiate between the answers and pick the correct one for analysis.

the instructor can answer the student's questions by comparing the student's answer to the results of the automatic assessment and the model answer to the exercise. If the instructor finds a fault in the automatic assessment, he must be able to either **adjust the points** given to the student, or modify the automatic assessment procedure and **restart the assessment manually**.

The functionality in this category was mentioned by several interviewees. A few did not go into the details, but mentioned a broader category of "handling special cases", with strong inclination that functionality mentioned above is required to handle such cases. Also, a large number of interviewees felt that the functionality in this category is perhaps the most important and most often used functionality in an automatic assessment system.

4.3.2 Searching

The functionality required for searching a particular piece of data in the system was not mentioned in the literature. However, many interviewees felt that a search functionality is very important for their work. Typically, the search function is used to find either a **specific student** in the system, or a **specific answer to an exercise**.

The searching category was not mentioned in Section 3.7, since the author did not find references to it in the literature. This is rather surprising, considering how this category was mentioned in many interviews. Several interviewees also pointed out that it must be possible to use several different search keys. For example, when searching for a single student, it should be possible to search using the student's name, the student number, user name, and email address, since students often fail to mention their student number or user name. These are typically used as the main key for identifying students.

4.3.3 System maintenance

The system maintenance category includes all functionality that the instructor needs to modify the data in the system, with one exception. The manual adjustment of points is included in the examination and modification of assessment results category, since it is so closely related to other functionality in that category.

The system maintenance category contains functionality used to add users, exercises and courses to the system, as well as configuring how the system works. Much of the functionality included here is used for general *course management* and is not specific to automatic assessment systems. However, most automatic assessment systems require this functionality, since they contain abstractions for courses and exercises in the course.

The functionalities in system maintenance category are used to maintain three logically separate types of data: user data, course data and assessment configuration. Therefore the category could be divided into subcategories, but that was considered to be unnecessary in this case. The system maintenance category contains so few different functionalities that dividing them into smaller categories would not improve the clarity of the analysis.

The maintenance of user data includes two functions: **adding users** and **controlling user access**. In many interviews it was mentioned that students could not register to the system themselves, but had to be added by the instructors. This caused additional work, and a few instructors stated that they wanted a system where the students were able to register without instructor assistance. It should be noted that the interviewees discussed only students, and did not mention anything about other user groups. Furthermore, access control functionality was mentioned only in one interview. The interviewee stated that since students, assistants and instructors all use the automatic assessment system, it is important to be able to manage the access level of the users. It seems, however, that access control is not considered a very important feature by most instructors.

Maintenance of course and exercise data contains a lot of functionality, and was considered by many interviewees as one of the most important aspects of the system. Indeed, in a system where a student solves exercises on a certain course, the functionality for **adding, removing and modifying exercises and courses** must be present. Also, since courses are often divided into a number of rounds, rounds must also be present in the system. Some interviewees mentioned the specific aspects of courses, rounds or exercises they had to modify. These aspects included modifying the deadline of an exercise or round, modifying the exercise description, adding additional information to exercise descriptions (i.e. links

to third-party material), selecting exercises from an exercise pool, generating personalized input for exercises and generating new exercise instances.

Some interviewees, however, did not consider course or exercise maintenance to be important. One person described course creation as task that is done once per course, and therefore less important than, for example, having access to student answers. The implication was that accessing student answers has to be easier than course maintenance.

In contrast to maintenance of user or course data, **modifying and configuring automatic assessment** often gets a lot of attention in the literature, since it is closely related to how the system performs automatic assessment. To give an example, the configuration of the automatic assessment is extensively discussed in [16].

In the interviews, several other functionalities related to system maintenance were also mentioned. These functionalities were typically mentioned only by one interviewee, and the author did not think them important enough to be included in Table 4.1. These miscellaneous maintenance functionality included, for example, the ability to add notifications to the system (for students to view) and modifying the number of times a single student may submit a certain exercise.

4.3.4 Result viewing

All the interviewees used automatic assessment systems for summative evaluation. In such systems, the instructor needs to see how the students performed on the course in order to decide the final grades. The most common way to accomplish this is to have the system to compile a **list of course results**, where the results of all students are shown. Rather surprisingly, this functionality was seldom mentioned in the interviews, although it is included in most systems. It seems that the majority of the interviewees thought it self-evident that the system could produce a result list.

The results of the automatic assessment system are often combined with the results of other parts of a course (i.e. the examination). It is therefore important to be able to **export the results** in a format that other systems can use. One possibility is using tabulator-separated text files, which can be manipulated using scripting languages, and are understood by many spreadsheet programs.

In some cases the instructor may, instead of a list of all results, wish to **view the results of a single student**. This can be, for example, in order to decide whether a student is given permission to take the course exam in a special situation. In addition, the instructor may wish to **export the students' answers**. He can, for example, use a separate plagiarism checking software and needs to import the answers to it.

4.3.5 Statistics

A majority of the people interviewed in this study mentioned that the system they used could generate statistics. Nobody thought this to be especially important functionality and two interviewees mentioned it as a non-important aspect of the system. Also, one interviewee

did not mention statistics directly, but implicated that the system he used generated them.

There are two different types of statistics that systems often generate: **operation logs**, which record the operations done by the system as they are done, and **statistics**, which are calculated from the data available in the system.

4.3.6 Results Beyond Functionality

The most important aspect of the interviews was to find out the functionality used by the interviewees. However, the definition of what is a functionality was left open, and the interviewer encouraged a broad perspective. This was done to gain more insight on the subject matter. The interviewees mentioned several aspects of the systems that did not fall under functionality, in the opinion of the author. Many of the insights gained are, however, relevant to this work.

Perhaps the most important insight from the interviews was that several interviewees either mentioned that the instructor's user interface was lacking, or explicitly wished for a better UI. Some technical considerations also emerged. Several systems either did not support parallelism, or the support was not very well implemented. One interviewee also mentioned the importance of backups and storage of old data.

Also, a few interviewees wished to develop more general *course management systems*, where the same system that handled automatically assessed exercises would also have support for larger project exercises, group work and semi-automatic or manual assessment of exercises.

Another technical problem mentioned occasionally was security, both privacy and data integrity. It seems, however, that most instructors do not consider security to be a large problem.

Chapter 5

The TRAKLA2 System

TRAKLA2 [26] is an automatic assessment system designed for visual algorithm simulation exercises. The system has been developed at Helsinki University of Technology and a production version was first used on spring 2004. The system is the successor of the TRAKLA system [15] that was used at the university between 1991 and 2003. The different versions of TRAKLA and TRAKLA2 as well as their main differences can be seen in Figure 5.1

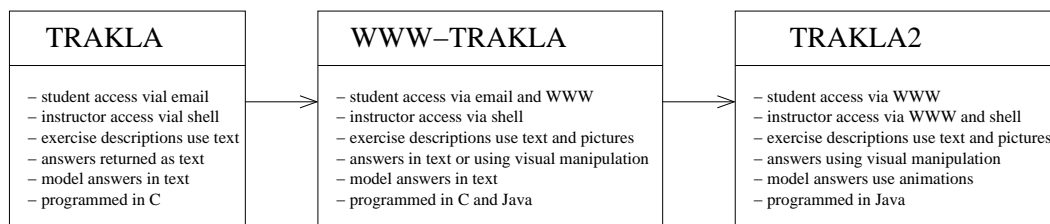


Figure 5.1: Versions of the TRAKLA system.

In Figure 5.1, TRAKLA is the first version of the system. Students interacted with it through email, and solved the exercises using pen and paper. Submissions were sent to the system using a pre-defined mail format and assessment results were sent back the same way. The first major revision to the system was when the web-based WWW-TRAKLA [20] user interface was added for the students. In WWW-TRAKLA the students solved the exercises by manipulating visualizations, and submitted them through the web. The exercises and the server-side functionality remained almost identical to the original version, however. TRAKLA2, the current version of the system, is a complete reimplementaion of the whole concept.

TRAKLA2 consists of an *applet* for solving and submitting the exercises, and a dynamic *web environment* for serving the applets to the users. The new system allows for more exercise types than TRAKLA, has a better way of assessing student answers, and includes several other improvements. The system is implemented in Java, and therefore platform independent. Exception to this is the database, which is implemented in C. The system

also includes a completely new web interface that contains both functionality for solving exercises and functionality for monitoring own progress and current situation.

While the first version of the student's user interface was completed early in the year 2004, the system lacked any sort of instructor's user interface. The only way for instructors to access the system was to manipulate the SQL database by hand. Therefore, there was a definite need for a new instructor's user interface for most common tasks.

The emphasis on the development of the instructor's interface was placed on monitoring students' progress and maintaining the system during a course. Creation of new courses and adding new exercises, which are typically done only once per course, were left for later development.

5.1 Overview

TRAKLA2 is meant for distributing and assessing exercises. The dynamic part of an exercise – which the student manipulates in order to create an answer – is implemented as a Java applet. The applet must be embedded to a web page which contains the static part of the exercise, mainly the exercise description. The web page can be a normal static webpage, but in order to use the system on a course, a *learning environment* centered on the exercises has been implemented [26]. From now on I will refer to the whole learning environment when using the term TRAKLA2.

There are three user groups in TRAKLA2: students, instructors and administrators. Currently students and instructors use the system through a web-based user interface. The two groups' user interfaces share the same look and feel, but have a completely different functionality. Administrators do not have a dedicated user interface and they use the student and instructor's interfaces as well as other tools as required.

A central concept for TRAKLA2 are the courses that are in the system. A course in TRAKLA2 consists of one or more exercises typically grouped together into rounds. Each exercise has a maximum number of points that can be gained from it, and optionally a maximum number of times it can be submitted. Each round has a deadline and optionally a minimum number of points that a student must gain from the round. The course can also have its own minimum points, which need not be the same as the sum of the round minimums.

5.1.1 Student's User Interface

The student's user interface has been designed to allow for two different use cases. The students can *solve exercises* and they can *review their results*.

For the first use case, the student must be able to select the exercise he wants to solve. There must also be a mechanism he can use to submit the answer. Finally, the system must show feedback and results of the submission to the student. In TRAKLA2 the exercises are solved and submitted using a Java applet.

For the second use case, the system must show the student an overview of the current results and the number of points gained from each exercise. The point total and grade must also be shown.

The two use cases are satisfied by two views in student's user interface: the *exercise view* allows the student to solve exercises and *main view* shows a summary of the student's results. The main view can also be used to select the exercise to be solved.

In order to use the system, the student must log in by supplying a valid user name and password. Currently, at Helsinki University of Technology, the student's unique student number is used as user name. Other institutions using the TRAKLA2 system can have their own rules for user names. The student chooses the password when he logs into the system for the first time. The password can later be changed.

After login, the student is shown the main view of the system where he sees an overview of the course and his results. From the main view, the student can choose an exercise to be solved. The exercise is shown in the exercise view. If the student is registered to several courses, he can also change the course shown in the main view. The main view is shown in Figure 5.2.

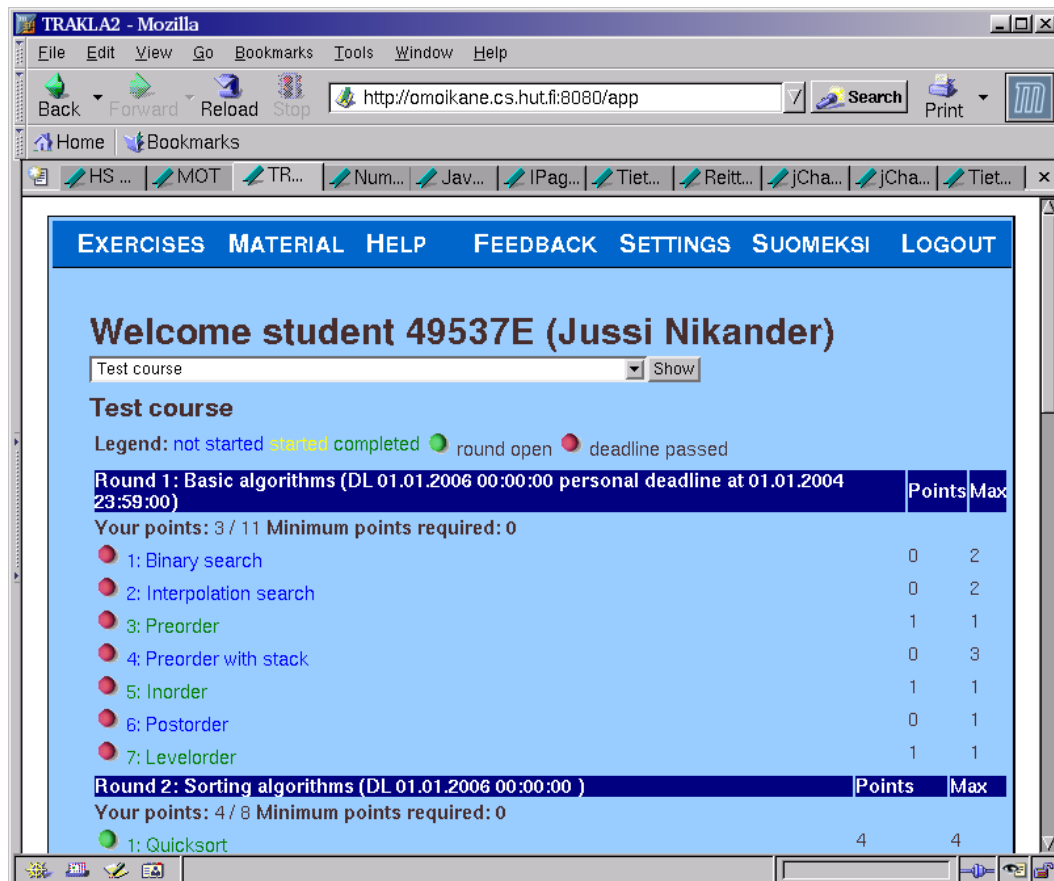


Figure 5.2: Main view of the student's user interface.

The courses in TRAKLA2 are divided into rounds, where each round contains one or more exercises. The main view of the system shows the status of each exercise and each round for the currently selected course. The information for a single exercise includes the state of the exercise, maximum points gained and information on whether the deadline for the exercise has passed. There are three possible states for each exercise: *not started*, *started* and *finished*. An exercise is in the state not started if the student has not yet submitted it. The state started means that the student has submitted the exercise at least once, but has not gained the maximum points for it. State finished means the student has submitted the exercise and received the maximum possible points. It is possible to submit an exercise that is in state finished. This won't affect the score, since the system selects the submission with the highest score for each exercise.

From the main view, the student can also select an exercise, which will lead to the *exercise view*. The exercise view gives a verbal description of the task the student must solve and shows the Java applet for the exercise. There may also be a link to a separate hint page or some third-party pages on the topic at hand. There are also links back to the main page, as well as the previous and next exercise, if applicable.

In addition to the two important views, the system includes a view where the student can change his user information (except the user name), a form which can be used to send feedback to the administrators, a help page, and a page that contains links to third-party material.

To summarize, there are two important views in the student's user interface: the *main view* and the *exercise view*. These implement the two important use cases required from the system: *solving exercises* and *reviewing results*.

5.2 Instructor's User Interface Design

From the instructor's user interface the instructor can view the progress of a course and handle the most common special cases that take place during the progress of a course. The instructor's user interface is aimed at computer science teachers who use TRAKLA2 on their course. Currently, tasks related to the creation of a course are not implemented in the instructor's user interface. Only administrators can create courses in the system.

5.2.1 Design Principles

There are several important considerations that were taken into account in the design of the instructor's user interface. First, and the most important from the point of view of the end user, is simplicity and intuitivity of the user interface. The view shown to the user upon entering the system must be compact enough to fit into the screen of a normal web browser and the most commonly used functionality must be easily available in the first view.

The three most commonly used functions in the instructor UI are 1) finding a single student's results, 2) generating a result list of all students on the course and 3) accessing

statistics on the progress of the course. These correspond roughly to **viewing a student's answer**, **viewing student list** and **generating statistics** functionality introduced in Section 3.7. The interview results in Section 4.3 also include all of the three abovementioned functions. All three of the functions must be available straight from the first view the instructor gets when logging into the system. Other views, such as seeing a specific type of statistics (i.e. the amount of answers during the last 24 hours) or viewing a specific student's submission can be implemented in views that are reachable from the main view through one or more links.

The system holds a very large amount of data that can be used to generate different statistics for the instructor. Some of this data, like grade or point distributions, can be viewed either using graphics or as text. In most cases, however, graphics are easier for humans to grasp and should be used in places where exact values are not vitally important. Therefore most statistics should be shown using graphics as default and text, if available, optionally. The file format for the graphics must be understood by most web browsers.

Another important aspect is data security. Through the instructor UI it is possible to view the results and submissions of all students, as well as change data, including student results. Therefore it is important that only authenticated users – instructors of the course – can access the information. If students could use the instructor's interface and give themselves points, the whole point of the system would be undermined. Therefore access to the instructor's user interface must be protected using user names and password and the communication between TRAKLA2 and the instructor's client interface (web browser) must be encrypted to prevent third parties from gaining access to the system.

In order to make the system available on as many web browsers as possible and to avoid security holes and other pitfalls of having a lot of functionality in the client program, the system will use minimal amount of JavaScript and other client-side scripting languages. Since most web browsers still have differences in how they interpret scripting languages, minimizing the amount of client-side script will make the system more stable and usable on several different browsers. Multi-browser support is not, however, considered very important for the instructor's user interface. The number of people using the interface is rather small, at least for now. Therefore there is not much need to make the system run smoothly on more exotic browsers.

5.2.2 Use Cases for Instructor's User Interface

The use cases the Instructor's user interface is designed to fulfill are *Viewing and fetching course results*, *handling special cases*, *viewing the results of a student*, *testing exercises*, and *monitoring course progress and statistics*. These come from previous experience I have had with automatic assessment systems, mainly with the old TRAKLA and the TRAKLA2 prototype. In this section the use cases are compared to the functionalities introduced in Section 3.7, which described the conclusions of the literature research, and the functionalities described in Section 4.3, which contains the results of the interviews. For each use case, it is first compared to the results of the literature research and then to interview results. For clarity the source for the functionality it is compared to is shown parentheses.

Fetching the course results is roughly equivalent to **viewing student list and course results** (literature) and **viewing the course results list** (interviews). It is perhaps the easiest and most basic use case. Since TRAKLA2 is used for summative evaluation and is a compulsory part of the basic data structures and algorithms course at Helsinki University of Technology, the instructor must be able to fetch the students' results from the system. The results must include at least each student's point total and current grade. Typically the student's point total for each round is also included. The final results are, of course, available once the course is over, but the instructor may also fetch partial results during the course in order to see how the course progresses.

Handling special cases is roughly equivalent to **modifying assessment results and feedback** (literature) as well as the category **examination and modification of assessment results** (interviews). Handling special cases is, however, broader and can include aspects of **system maintenance** (interviews). It is another important use case that typically crops up a few times on every course, and more often on large courses. Handling special cases includes functions other than merely modifying assessment results. Sometimes the instructor may have to, for example, modify the number of times a student can submit an exercise, or change the deadline for an exercise round. However, in TRAKLA2 system it is not possible to modify the feedback given by the automatic assessment.

Special cases come up when a student is unable to use the system for some reason. For example, he might have too much other work close to a deadline or he feels that his submission has been assessed incorrectly. He can then request either a change in the deadline of some or all exercises, points awarded manually, points corrected for some exercise, or a combination of the above. The instructor must therefore be able to change every deadline on the course and award students points manually. Since the system always chooses the highest score for a single exercise as the score given, there is no need to update already given scores.

Viewing the results of a student is equivalent to **viewing a student's answer** (literature) and **viewing the results of a single student** (interviews). It is required when a student is unhappy with the assessment of an exercise. The instructor must be able to view the student's answers in order to see if there has been an error in the automatic assessment or if the assessment procedure is somehow unfair. This is closely related to *handling special cases*. If the instructor cannot see a student's results, he cannot handle some of the special cases described.

Testing exercises has no equivalent functionality in the results of the literature research or the interview results table. Some interviewees did, however, state that they wanted to be able to see the system from the point of view of the student, or be able to test the exercises. Testing exercises is an activity where the instructor solves an exercise in the system in order to see if it works correctly or is useful for the course. It is more closely related to system development than course administration. In this use case, an instructor may test how an exercise works and see the fine details that may not be clear in general exercise descriptions. For example, a list of exercises typically does not tell how duplicates are handled in a binary tree exercise or what version of B-tree is used in the exercise. This becomes more and more important as the use of the system spreads and more institutions become interested in it.

Monitoring course progress and statistics is perhaps the broadest use case. It includes all views that show the students' results as well as server activity logs, statistics on exercises and rounds, and other views showing how the course progresses. The **generating statistics and reports** functionality (literature) and the **statistics** category (interviews) typically implement part of this use case. However, in TRAKLA2 we wish not only to compare courses with each other but also see detailed information on exercises and rounds. This is more detailed and broader than statistics or reports available most surveyed systems. Most surveyed systems provide only list of students and their results. Other statistics information can be limited or non-existent. In TRAKLA2 the statistics will include details of each round and exercise (i.e. how many answers have been submitted, average score, etc.) as well as ability to compare the results of different courses with each other.

5.2.3 Development Process

The development of the TRAKLA2 instructor's user interface loosely used the *rapid prototyping* model of software development. The project was started by the construction of a rapid prototype. This prototype, described in Section 5.5 included an initial guess of the functionality required for the instructor's UI. During the construction of the prototype, the author started background research on related work.

After the prototype was completed, the author started working on the specification and design of the UI, while still continuing research on related work. After the background research was concluded, it was noticed that the amount of information on instructor's user interfaces in the literature was rather scarce, and therefore the related work section was to be complemented with expert interviews. However, even before the conducting the first interview, the author had started implementing the new interface. Figure 5.3 illustrates the development process.

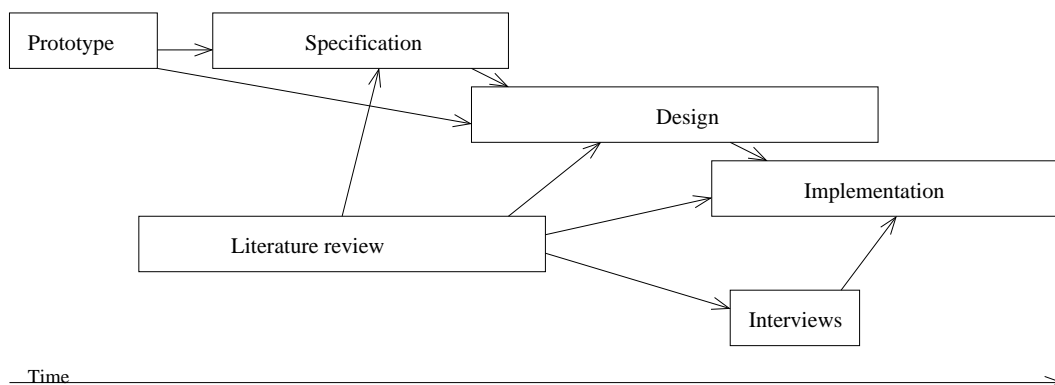


Figure 5.3: The development process of the instructor's user interface.

In Figure 5.3 the x-axis shows the passage of time. The different boxes visualize the different steps of the development process. The size of the boxes does not accurately describe how long some part of the development progress took, but are designed to show which parts were done simultaneously. The arrows describe which parts directly affected some other

part of the progress. The implementation of the new UI is shown in a box that is open from the right, since the system will be developed further even after the completion of this particular project.

5.3 Architecture

The TRAKLA2 environment consists of three large parts: the *WWW-server*, the *RMI-server*, and the *database*. The system also writes several log files. An overview of the whole system is shown in Figure 5.4. This architecture is shared by student and instructor's user interfaces. Normal use of the system is handled through dynamic web pages and the user navigates the system using a web browser. The pages are created by the *WWW-server*. When the user solves an exercise or views a submission, a Java applet is embedded in a web page. The applet communicates with the *RMI-server*. The *RMI-server*, in turn, writes log files and, when an exercise is submitted, sends the submission data to the *WWW-server*. The database is used to store all persistent data, and all database communication goes through the *WWW-server*.

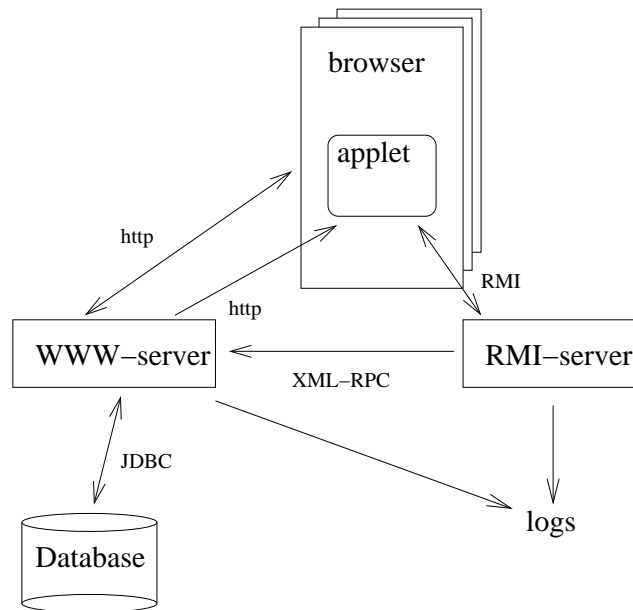


Figure 5.4: Architecture of the TRAKLA2 learning environment [26].

The *WWW-server* is the heart of the TRAKLA2 environment. It handles session control, creates web pages, embeds applets to web pages when appropriate, and stores submissions and assessment results to the database. In addition, the *WWW-server* handles all communication between different modules of the system. For example, when a student submits an exercise, the submission is received by the *RMI-server*, which then sends the results of the assessment to the *WWW-server*. The *WWW-server* stores results to the database.

The *RMI-server* handles communication with the applet. It sends exercise input to the

applet, receives and stores submissions, and forwards the submission information to the WWW-server. The RMI-server is in constant communication with the exercise applet, and stores information on the student's interaction with the applet. The details of the server-applet interaction are described in [40] and [41]. Communication between the RMI-server and the exercise applet is handled using Java RMI, and communication between the RMI-server and the WWW-server using XML-RPC¹.

5.3.1 Tools and Frameworks

The whole TRAKLA2 environment, with the exception of the database, has been implemented in the Java programming language. Several freely available third-party frameworks and tools were used in the implementation. Aside from the Java programming environment, the most important third-party programs used are Jetty web server/servlet engine and Tapestry web application framework. The server also uses the jCharts chart creation package, log4j logging package, and the Apache implementation of the XML-RPC protocol. For communication with the database, the PostgreSQL Java interface is used.

Tapestry² is the tool on which the TRAKLA2 environment has been built. It is a framework for creating dynamic web applications. The dynamic web content in Tapestry uses the Java servlet API, and the framework handles many aspects of servlet creation automatically. For example, construction of link URLs, session control and localization are handled by the framework and the system developer is free to concentrate on creating the content. The developers of Tapestry claim that since the system uses standard Java servlet API, it works over any servlet engine. However, most such engines, like Apache Tomcat, require a separate web server software where the engine is run. The Jetty web server has a built-in servlet engine and was therefore selected as the web server used in the implementation of the TRAKLA2 environment. With Jetty there was no need to use separate web-server and servlet engine.

jCharts³ was selected for chart creation, since it was known to work with Tapestry. In the Tapestry benchmark application shipped with the framework, jCharts was used to create a simple, dynamic pie chart. Similarly, log4j was recommended by the developers of Tapestry and therefore adopted as the logging utility for TRAKLA2. The Apache implementation of XML-RPC was used since there was previous good experience with it.

5.3.2 Implementation

The WWW-server implementation can be divided into two more or less independent parts. The first is dynamic web page and session state implementation and the second data model and database interface implementation.

The page and session state implementation extensively uses the Tapestry framework in order

¹<http://www.xmlrpc.com/>

²<http://jakarta.apache.org/tapestry/>

³<http://jcharts.sourceforge.net/>

to handle the creation of the required dynamic content, session control, and security. The data model and database interface, on the other hand, are not dependent on Tapestry.

Web Pages

Typically an instance of Tapestry contains one web application that, in turn, contains one or more web servlets. A servlet contains, for example, the student's user interface to TRAKLA2. If a web application contains several servlets, these can be logically separate, but will share the same namespace on the server.

For each dynamic web page, Tapestry requires three different files. First is the *page template*, a HTML template file used to create the actual page shown to the user. The second is the *Java implementation* that creates the dynamic content to the page when it is rendered. The third is an optional *page specification*, which is an XML descriptor file that includes data on the variables used on the page, and can greatly reduce the amount of required Java code.

The templates are created using static HTML code combined with dynamic *Tapestry components*. The components are dynamically rendered to HTML when the page is requested by an user, and data fetched using the Java implementation is used in the creation of the rendered page. The Tapestry framework includes a number of ready-made components, including several different types of links, loop structures for rendering variable-length data, components for creating web forms, etc. It is also possible to create new components that, for example, render structures used on several pages. When a page is rendered, the Tapestry engine examines each component on the page and creates HTML code from it, usually by calling a number of Java methods to retrieve the required data, and creating new HTML code from it. All dynamic links are also created at this point and hooks to handle the event of a user clicking on a link are set up.

The Java code for a Tapestry page typically includes a number of get and set methods that control properties either on the currently viewed web page, or in the HTTP session. These methods are called by Tapestry framework when encountering a component that either needs the property, or when an action (such as sending a web form or clicking a link) modifies it. In many cases, the required methods are very simple: `getSomething` method returns a property, and `setSomething` gives it a new value. Since there may be a large number of such methods on a single page, Tapestry includes *page specification* files that can be used to automatically generate these methods. However, if a get or set method does something more complicated, like fetching information from database, it must be implemented manually.

Data Model and Database Interface

The in-memory data model used in TRAKLA2 includes a lot of functionality. For example, there is no need to store student grades to the database, since the system is capable of calculating them in runtime. Using the object-oriented model, each student, course, round

and exercise in the system is represented by an object. For example, a Student object, which includes the student's identity (typically student number), the list of courses he is registered to, and the results for each course, represents a student.

A TRAKLA2 course is represented by a Course object, which includes basic information of the course, such as name, code, start and end times, as well as a list of rounds on the course. Each round, in turn, is represented by a Round object, which includes a number of Exercise objects, each representing one exercise on the round. The Course, Round or Exercise objects do not hold any information about the students on a course. Instead, each student's results for a particular course are stored to the Student object.

The objects representing students, courses, and other data entities in the system are created by a DataStorage object. The DataStorage class implements the *singleton* design pattern, meaning that there is never more than one instance of the object in the system. DataStorage is the system's interface to the database. The rest of the system uses DataStorage to store and retrieve persistent data, and do not see the database directly.

The database is deliberately designed to hold only raw data, which is processed by TRAKLA2 in order to create information such as total points for a course, grades, etc. This makes the database design as simple as possible and ensures that all processed data is always up to date. This also makes the total size of the database smaller, and simplifies the database management code.

A PostgreSQL database is used to store the persistent data in the system. Other database systems, including a couple of object-oriented databases and MySQL were considered in early stages of TRAKLA2 development. PostgreSQL was considered the best choice because the tested object-oriented alternatives did not work well with Tapestry, and PostgreSQL implemented the SQL standard much better than MySQL.

5.4 The Prototype Instructor User Interface

Before the instructor's user interface was implemented, a prototype was constructed. The purpose of the prototype was to gather opinions on what is really required from the instructor's user interface. Therefore the prototype worked as a *rapid prototype* in the development of the instructor's user interface.

The main design principle of the prototype was that there was no need to include any functionality for course management in the system. It was assumed that once a course had been created there was no need to change anything on it, and the instructor's interface would be required only for monitoring how the course progresses. From this came the original name of the instructor's user interface "monitor interface". It should be noted, that this decision was made before starting the literature study.

The prototype contained only a fraction of the functionality of the final version. Using the prototype it was possible to fetch course result list, as well as see the results of a single student, and some statistics on the rounds and exercises. The system had six views: *main view*, *course results view*, *student results view*, *statistics view*, *course details view*, and

exercise testing view.

In the *main view*, which was reached after login, the instructor was shown a list of courses he could access. For each course the instructor could select *course results view*, *statistics view* and *course details view*.

The *course results view* and *student results view* were identical to the corresponding views of the final version, described in Section 5.5.2. In the course results view the instructor was presented with a table of results for all students on the course. The instructor could also select one student and see his results in more detail. In the prototype, however, the student's submissions were not shown.

The *statistics view* and *course details view* both contained statistics information. Most of the statistics in the two views were included in the final interface, but the views were combined and the information rearranged.

5.4.1 User Reaction to the First Version

After the prototype was taken into use, it was rather quickly realised that during a course there would be occasions when an instructor should be able to modify the contents of a course. A bit surprisingly, it was the exercise descriptions that most often required modifications, instead of some course or student specific data items. It was noted, however, that the instructor's user interface should also include functionality to change deadlines and student results.

Moreover, it became clear that the monitor interface was not very useful if the instructor could only see the results of the assessment (i.e. points awarded) but could not access the answers. If a student asked about the assessment of an exercise, his answers had to be retrieved from the file system by hand before they could be reviewed. This is a time-consuming process and made it difficult to check the student's answers.

Another large problem was that the system had no search function and therefore it was rather time-consuming to find a single student's results. It was especially hard if there were several parallel courses in the system simultaneously, and the instructor was not sure which course the student was enrolled to.

Furthermore, it was realised that the views in the interface were rather poorly designed. Especially the main view of the system was criticized, since it contained no data, and had no real purpose in the system aside from being a point where different information could be requested. It was also felt that the statistics and course information views did not serve any purpose. It was, for example, impossible to go and view an exercise from the course information view as the link to the exercise test view was available only in the statistics view. It was suggested that the two views should be merged and the information that was considered relevant was retained in the new view.

In general, it seemed that the first version was a good starting point for the instructor's user interface but lacked the functionality that would have made the system truly useful. Furthermore, the different views in the system had some useful information and included

good functionality, but the contents of the different views and the navigation between them should be rethought. In the new version special emphasis should be in ease of use: the most common functionality should be easily accessible immediately after logging into the system.

The most useful feature of the original system was the ability to create course result lists. These lists were, for example, used in publishing official course results. Furthermore, together with the results of the other compulsory parts of the course, the result list was used to calculate final course grades for the students.

Using the feedback gained from the first version of the instructor's user interface, several modifications were designed and implemented. The user interface described in Section 5.5.2 is the result of the redesign.

5.5 Implementation of the Instructor's User Interface

When the work on the instructor's user interface (IUI) was started, TRAKLA2 had been in development for a year. Therefore, the system already had a lot of features that already existed for the student's user interface (SUI). A lot of these features were reused in the IUI.

In Tapestry, the two user interfaces to TRAKLA2 are located as separate servlets in same web application, and they share a lot of functionality in the server side. They share the same data model and database interface, but use separate web pages. Since the IUI contains functionality that the students must not be able to access, a lot of attention has been given to making it impossible to access the instructor's functionality without a permission. For example, the web pages created by Tapestry are normally sent to the client browser using unencrypted HTTP-protocol. The IUI, however, uses the secure HTTPS protocol.

5.5.1 Code Level Details

Since both student and instructor's user interfaces reside in the same Tapestry web application, they share a namespace in Tapestry. The upside of this was that it was easy to reuse parts of the SUI. The downside was that unless it is explicitly forbidden, the pages of the IUI could be viewed from the student's interface.

Web Pages

In a Tapestry application, the web page templates reside in subdirectory `context/` and the page specifications are in subdirectory `context/WEB-INF/`. Therefore, it is not possible to separate pages by putting them into different subdirectories. In order to be able to separate the SUI and IUI templates, the name of each template that belongs to the IUI starts with "Monitor". For example the main page of the instructor's user interface is in the file `context/MonitorMain.html`.

The Java classes corresponding to the web pages were named after the name of the template followed by “Page”. Therefore, the java class that handles the dynamic information to MonitorMain.html was named `MonitorMainPage.java`. This led to some rather long names for both web page templates and Java class files, but allowed for easy identification of all pages and corresponding Java classes.

In the creation of web pages, only very little functionality originally written for the SUI could be reused. Each web page typically requires different functionality, and the aims of the student and instructor’s user interfaces were very different. Therefore there was no code available that could be reused.

In the student’s user interface, there was no need for creating graphics dynamically. However, when the IUI was designed, it was decided that dynamic graphs for representing course results and server activity were required. After some consideration, the jCharts package was selected for this. In order for the jCharts to be used, a new Tapestry service was implemented to serve the charts, and a Tapestry asset was created to encapsulate a single chart. The implementation for these was heavily based on their implementation in the Tapestry benchmark application, which included a simple dynamic chart. The JPEG picture format was used for the charts since it was supported by all common web browsers.

Encryption and Data Security

Tapestry does not directly support encrypted web communication. Therefore, in order to enable encrypted communication, HTTPS support had to be added to the whole instructor’s user interface. Fortunately, the student’s user interface also requires HTTPS upon login, in order to guard the user’s password. The addition of HTTPS support required approximately 20 lines of additional code to the Java class of each Tapestry page that needed encrypted communication.

The inclusion was handled through object inheritance: the Tapestry framework includes a class `BasePage`, which contains all basic functionality required by a Tapestry page. In a typical Tapestry application, each Java class corresponding to a dynamic web page inherits `BasePage`. For HTTPS to work, `BasePage` was extended and one method was overridden to create `SecurePage`, that has all the functionality of `BasePage`, but forces encrypted communication. All Java classes that handle page functionality in the IUI were made to inherit `SecurePage` instead of `BasePage`.

Data Model

The data model originally created for student’s user interface was good enough to be used in the instructor’s user interface with some modifications. The original data model was written to gather one student’s information for all courses, and work with this information. For the IUI it was more important to work with the information of all students on a particular *course*. It was possible to reuse the student’s data model in this: instead of collecting the information on all courses for the student, the information on a single course was used. The

information for the whole course was created by combining the students' information.

In addition, statistical information was also needed in the IUI. This was handled by separate functionality created specifically for it. The two data models, student data and statistical data, have some overlap. However, it is easier to keep the data separate, since they are used rather independently. The student data is used to create result lists and show a single student's data upon request, while statistical data is used in course comparison and when course statistics are viewed.

Since the data had to be ordered in a new way, the `DataStorage` class, which is used to access the database, had to be modified quite extensively. In addition to the functionality used to gather data for a single student, `DataStorage` now includes functionality for generating result lists and statistics for whole courses. Since these data had to be arranged differently from what a single student required, the old data retrieval functionality in `DataStorage` could not be used.

Modifications to the Applets and RMI-Server

The RMI-server is used by TRAKLA2 to communicate with the exercise applets and to relay assessment results to the WWW-server and the database. Originally, the information relayed by the RMI-server included student, course, round, and exercise identification, as well as the amount of points awarded. Notably, no information about the submission itself was passed to the WWW-server. However, in order to make it possible for the instructor to review the students' submissions, its details must be forwarded as well. Since the WWW-server communicates with the database, the answer data must go through it in order to be stored.

The answers are stored to the filesystem by the RMI-server. It was briefly considered to change this and make the RMI-server forward the answer data to the database for storage. However, this idea was discarded for two reasons: first, the XML-RPC protocol used for communication between the RMI-server and WWW-server does not support transmission of arbitrary Objects. Second, the current storage system works just fine, so there was no need to go and change it. Therefore, the RMI-server was modified to send the filename of each stored answer, with the full path included, to the WWW-server.

Using the data stored to the WWW-server the instructor can use a simple Java applet to view the students submissions. The viewer applet is very simple: given a file path, it can request the RMI-server to send it the given file. The applet, in turn, is capable of visualizing the answer contained in the given file.

5.5.2 User Views In Instructor's User Interface

The instructor's user interface has several views and is, in general, more complex than the student's user interface. The *main view* of the IUI serves as a central point of navigation through which the instructor may access the different functionality. The main view can also be enhanced by maximizing hidden sections. The different views implement the use cases

described in section 5.2.2.

Login and First View

As in the SUI, one needs to log in to the system. Users are recognized by user name and password, both of which can be selected freely. In order to access the UI, the user must have instructor access to at least one course in the system.

After login, the instructor is shown the *main view*. The main view includes functionality for accessing student results for the whole course, the results list, detailed results for a single student, details of the course, statistics for rounds, and list of all exercises in the system. Also, if the instructor has access to more than one course, it is possible to change the active course. In Figure 5.5, the *main view* is shown as it is seen when viewed after login. The upper part of the view shows the name of the course, the course changing and searching functionality. Under the separator is a number of *tabs* that can be used to view the details of the course, the statistics of each round, and the student list. By default, the course details tab is selected. The contents of the main view change according to the selected tab.

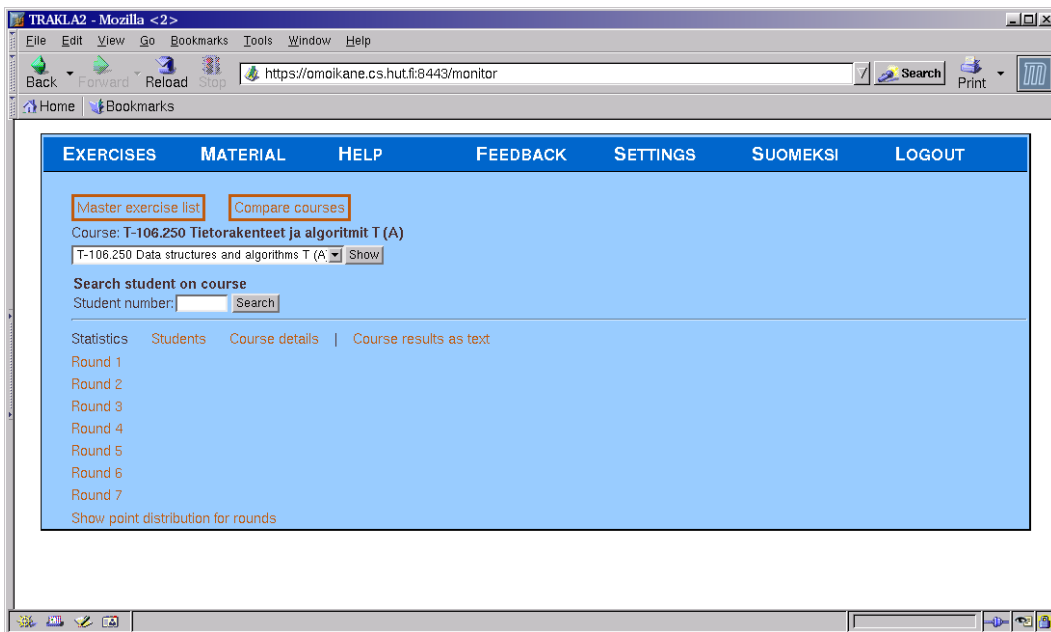


Figure 5.5: Main view of the instructor's user interface.

Viewing Course Results

The course results are shown in two different views, which both contain essentially the same information. The difference between the views is in the available functionality. One view contains tabulator-separated text fields. The result list in this view is therefore easily exported to other systems (i.e. spreadsheet programs), or manipulated using languages such

as Python or AWK. The other view is a HTML table from which the instructor may select a student and view his results in more detail. In both views the result information has a summary for each student, containing the amount of points he has gained from each round, point total and the grade awarded. The text list can be viewed by selecting “course results as text” link from the *main view*, and the HTML table by selecting the “Students” tab.

The detailed results for a single student can also be found using the *search for student* functionality in the main view. If the student is found, the main view is changed to *student results view*. This view is shown also when a student is selected from the results list.

Viewing the Results of One Student

The student results view contains the following information: the points the student has gained for each exercise, point totals for each round, point total for the whole course and the grade gained. The number of submissions to each exercise is also shown. The *student results view* is shown in Figure 5.6.

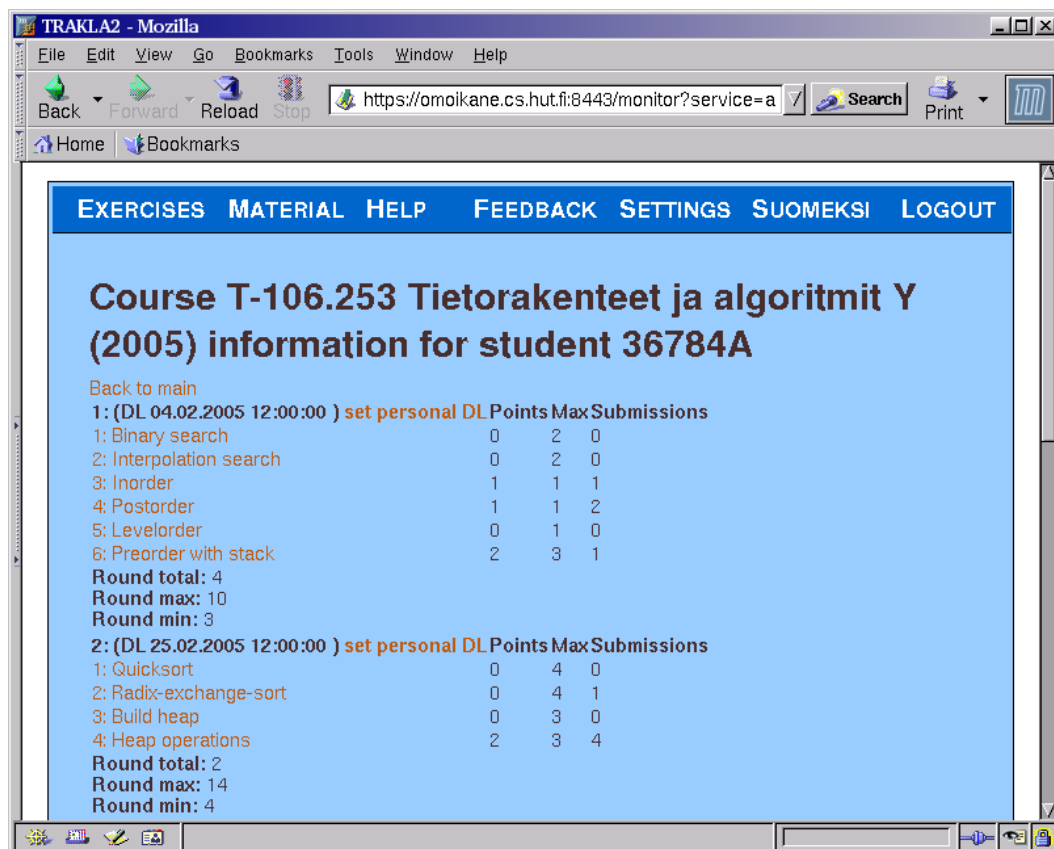


Figure 5.6: Partial view of one student’s results. First two rounds are shown.

In the student results view, it is possible to select the student’s submissions to a single exercise and see them in the *submission view*. In this view, the instructor is first shown

information on each submission the student has made for the exercise. The points gained, time of submission and possible notes (telling, for example, that the submission was done after the deadline) are shown. The instructor can then select a submission to be shown in a Java applet.

Round Statistics

The statistics tab on the *main view* lists all the rounds of a course. From this list it is possible to expand one or more rounds in order to view their details. Figure 5.7 shows the main view with the details of one round expanded.

The details of a round include round-specific information and details for each exercise. The round-specific details include the times for opening and closing the round as well as the *makeup factor*. The round is available for the students after the opening time has passed and submissions received after the closing time are handled as late submissions. Points for such submissions are multiplied by the makeup factor, which is between 0 and 1. If makeup factor is 0, it is not possible to gain points from late submissions, and if it is 1, the deadline has no effect on assessment and late submissions are given points normally. The closer to 0 the factor is the more late submissions are penalized.

The maximum points for the round and the minimum points a student needs in order to pass the round are visible as well. If a student has not gained the minimum points for each round, he cannot gain pass the course. Finally, the number of submissions for the round (the sum of submissions for all exercises of the round) and the average number of points gained by students are also shown.

For each exercise on the round the following information is shown. First is the name of the exercise, which is also a link to the *exercise testing and modification view*. Next is the maximum points for the exercise, followed by the number of submissions allowed. If a student submits an exercise more times than is allowed, the extra submissions are not taken into account when selecting the final points for the exercise.

After the number of submissions, there are the available languages for the exercise description. After that comes the total number of submissions for the exercise, followed by the percentage of students that have submitted it at least once. If all students registered to the course have submitted the exercise, this number is 100.

Next is the average points for the exercise in two columns. First one is the average of all submissions, and second the average of all student results. That is, the second is the average of the maximum points gained by each student for this exercise. These two averages do not include those students that have not submitted the exercise.

Finally there is the average of student points as a percentage of maximum points, again in two columns. The first percentage includes only those students that have submitted the exercise, while the second includes all students on the course. If all students have submitted the exercise, the two percentages are identical, otherwise the second is lower of the two.

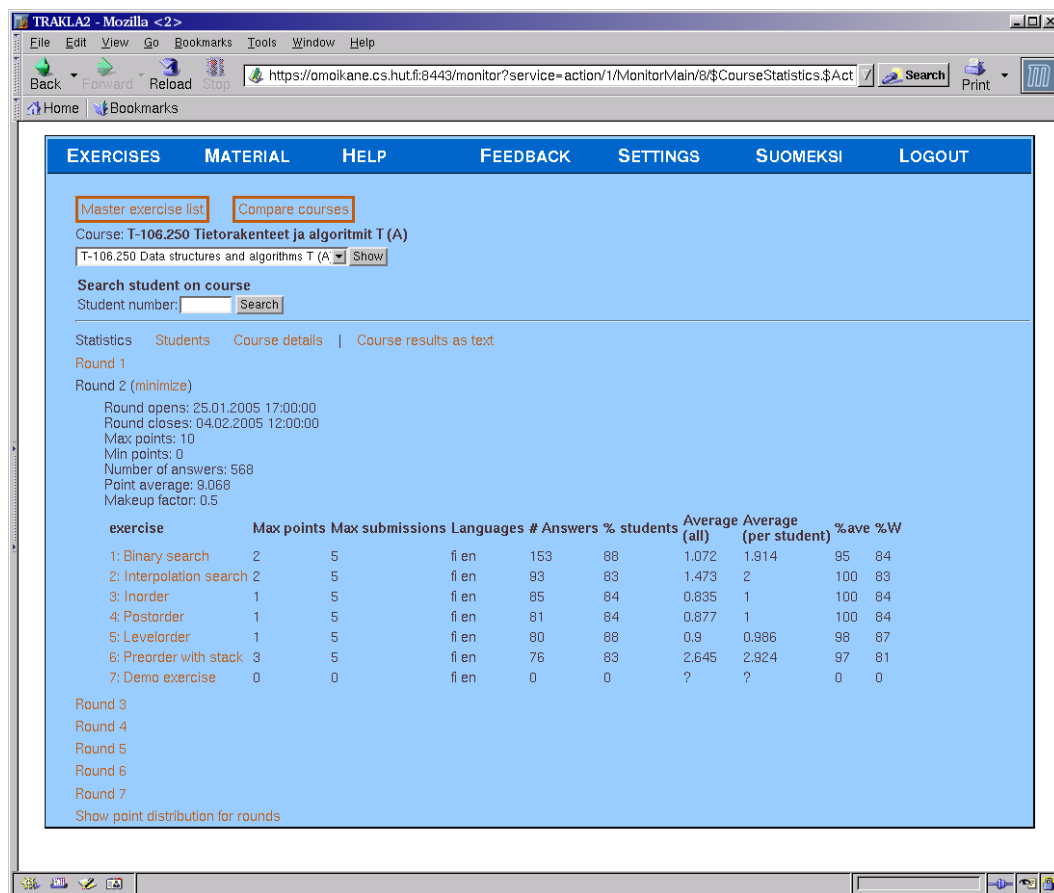


Figure 5.7: Main view with statistics for a round expanded.

Course Details

The main view contains a “course details” tab. Opening this tab shows the basic details for the course. The basic details include the code and name of the course. The code is the University-specific code used to identify the course in that institution; name is self-explanatory. The number of students registered to the course, the number of answers students have submitted, and the opening and closing time for the course are also shown. The opening time tells when students can start registering to the course. After the course has been closed, exercises cannot be submitted any more and no new students may register to the course. When the *course details* are open, the *course grading* and *server activity* can also be expanded.

The course grading simply shows the point limits for each grade. When the grades are shown, it is possible to open a view to show details of the point and grade distributions for the course. The distributions are shown as charts, where the x-axis contains the points (or grades) and y-axis the amount of students who have gained the given number of points or the given grade. The charts for a course are shown in Figure 5.8.

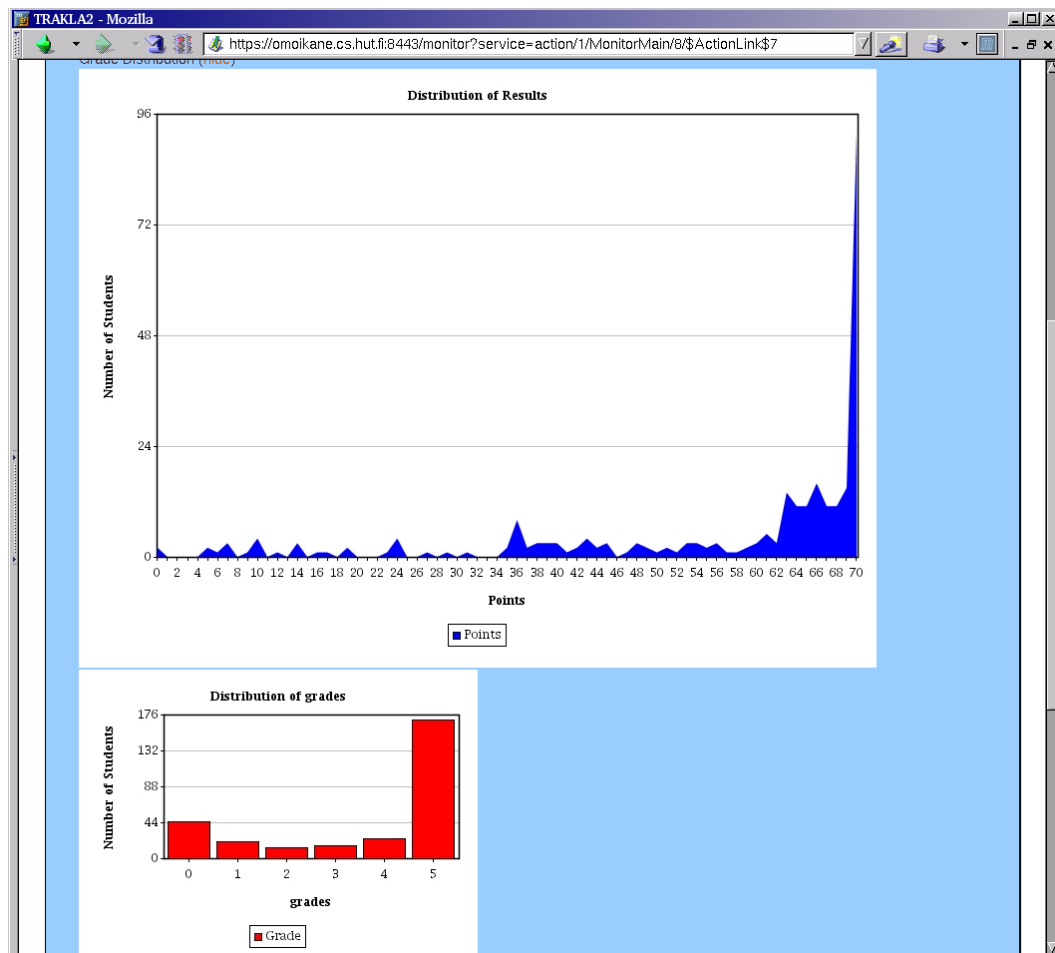


Figure 5.8: Graphs showing the point and grade distribution for a course.

The *activity* view shows a chart where the number of exercises submitted to the course during a certain time period is shown. The chart can show the activity either for a single day, week, month, the whole course, or the overall distribution between hours of a day. The day view lists the number of submissions for each hour of the day, week view divides each day to six portions of four hours each, month and course views show each day separately. The distribution view combines the submissions done during the whole course and shows the results by hour.

Course Comparison

The *course comparison* view can be selected from the main view. In this view, the instructor may compare the results of courses, or see the results of one course by selecting only one item for comparison. The results may be compared either using the points or the grades gained by the students, and the results can be viewed either as charts or text. The initial

view before courses are selected for comparison is shown in Figure 5.9.

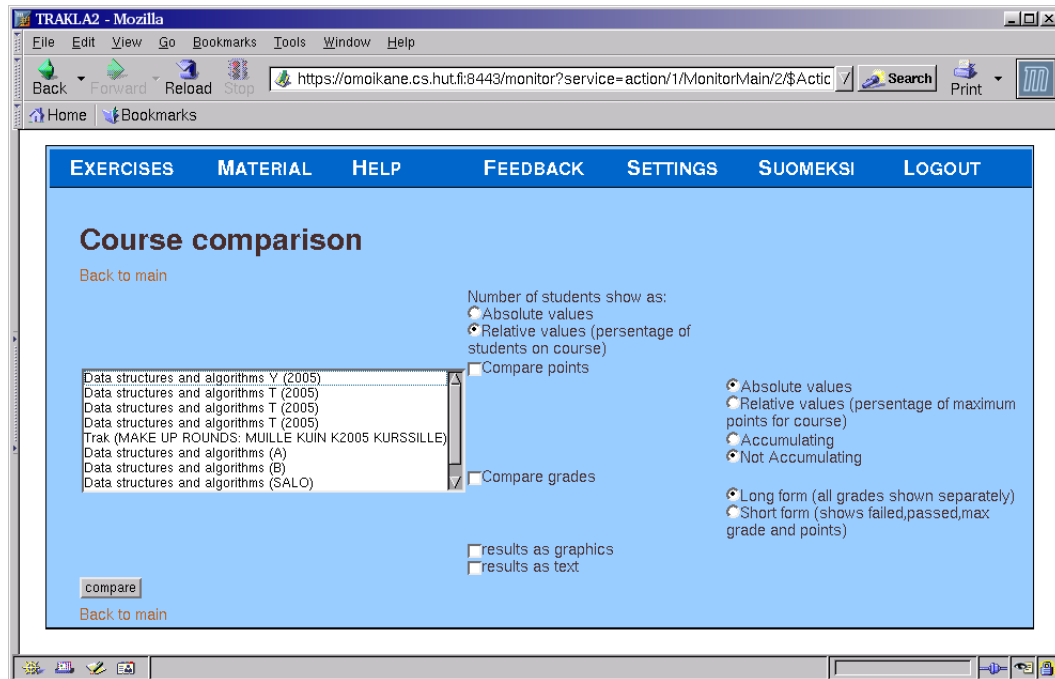


Figure 5.9: Course comparison view.

The comparison options are the following. First, the instructor needs to select whether the number of students on the course are handled as absolute or relative values. If absolute values is selected, the number of students gaining a given result is handled as is. When relative values are used, the number of students is handled as a percentage of the total amount of students on the course. By default, relative values are used to make the comparison between courses with different number of students easier.

Second, the instructor must select whether he compares the points or the grades. For points, he can select whether the points are handled as absolute values or values relative to the maximum points for the course. Further, he can select whether point amounts are shown as cumulative amount or not. For grades, the instructor can select either “long form”, where number of students gaining each grade is shown separately, or “short form”, where the students are grouped into four categories. The first category are those students who failed to pass the course, the second those who passed the course but did not gain the maximum grade, third those who gained the maximum grade and fourth those who gained the maximum grade with maximum points. The short form is included since it can be used to compare courses that have different grade levels and since it is used in previous research [27]. Finally, the instructor can select to view the results either as graphics, in text or both.

Using graphics the results are shown as a line chart for points and a bar chart for grades. If grades are shown in short form, the points of each course are shown as a stacked bar. In the long form, normal bar chart is used and in text format each course’s results are listed

separately.

Exercise Testing and Modification

There are two ways an instructor can enter the *exercise testing and modification view*. First, he can click the *list of all exercises* in the *main view* and select an exercise from this list. Second, the maximized statistics for a round include links to all exercises of the round. Both ways lead to the same view.

In the *testing and modification view*, the instructor is shown the exercise description in all supported languages (typically English and Finnish), as well as the Java applet containing the exercise. From the view, the instructor may modify the descriptions or solve the exercise himself. The description modification is included since exercise descriptions often contain mistakes, inconsistencies or simple typographical errors, that can be easily fixed through the web. Other errors in exercises, however, would require modifications to the code. Therefore only exercise descriptions can be fixed from the web.

Chapter 6

Evaluation

In this chapter, the TRAKLA2 system is evaluated with the ISO 9126 product quality standard. The system is also compared to its predecessor, the TRAKLA system. In the evaluation, the instructor's point of view is given most consideration. Some parts of the evaluation will, however, touch the student's point view.

6.1 Experiences on Using TRAKLA2

The TRAKLA2 web environment was used for the first in the spring 2004 on the Data structures and algorithms course at Helsinki University of Technology. At the same time, the system was taken into use on the corresponding course at University of Turku. On the spring 2004 course, only the student's user interface had been implemented. The instructors had no dedicated user interface, but had to access the database directly. The implementation of the instructor's user interface started in August 2004, and a prototype version was ready before the spring 2005 course started. Based on the literature study, experiences on the spring 2005 course and the expert interviews, the instructor's user interface was implemented on the summer of 2005. The new user interface was taken into use on the beginning of September 2005. On fall 2005, the TRAKLA2 system is used at Tampere University of Technology and Helsinki Polytechnic Stadia.

The student response to the system has been very positive. At both Helsinki and Turku the course staff gave the students a questionnaire where, among other questions, they were asked to give an opinion on the TRAKLA2 system. The results of the survey at Turku are reported in [24] and some results of the survey at Helsinki can be found at [26]. No surveys on the instructor's user interface have been made. However, the number of people who have used the instructor's user interface is much too small that any statistical results could be gained from a survey.

6.2 Using ISO 9126 To Evaluate Automatic Assessment

ISO 9126 [1] is a product quality standard for software systems that uses six characteristics to evaluate the quality of a product. Valenti et al. [48] describe how the domain specific characteristics of the standard, *functionality* (see Section 6.2.1), *usability* (see Section 6.2.2), and *reliability* (see Section 6.2.3), can be used to evaluate computer based assessment systems.

For evaluating AA systems, the characteristics must be defined in the context used. In this case, the context is automatic assessment systems used on large university courses. The system discussed in this thesis is designed for the evaluation of data structure and algorithm simulation exercises. The evaluation will be done for this particular purpose by comparing the new TRAKLA2 system to its predecessor, the TRAKLA system.

In [48] a computer based assessment system is composed of two large subsystems, the *test management system* and the *test delivery system*. Test management system is used to manage the questions and tests (question groups, similar to rounds and courses in TRAKLA2) while test delivery system is used to deliver the tests to students. These are rather similar to the instructor and student's user interfaces in TRAKLA2. Therefore, in this work, I will only consider those criteria that measure the test management system.

6.2.1 Functionality

Functionality is defined as “The capability of the software product to provide functions which meet stated and implied needs when the software is used under specific conditions”[1]. This characteristic is concerned with what the software does, not when or how, like the other characteristics. This characteristic tells whether the required functionality is present in the program. The subcharacteristics of functionality include *suitability*, *interoperability*, *security*, and *accuracy*. The subcharacteristics and the evaluation criteria associated with them are shown in Table 6.1.

The subcharacteristics suitability and accuracy are closely related. Suitability measures the software product's capability to “*provide an appropriate set of functions* for the specified tasks and user objectives”. Accuracy measures the software product's capability to “*provide the right or agreed results or effects* with the needed degree of precision”. (Emphasis is mine, in both quotations) In other words, suitability measures how well the software can do what it is supposed to do, and accuracy measures how well the results conform to what is expected.

Interoperability measures a system's capability to interact with other systems. This includes the ability to use services provided by other systems, the ability to provide such services, as well as the capability to export and import data. Security measures the system's capability to protect the information and data stored in it from unauthorised access.

Several criteria can be used to evaluate the functionality of assessment systems. Each criterion is linked to one of the subcharacteristics of functionality. Furthermore, the criteria can be divided to those that concern the management of individual questions and those that

<ul style="list-style-type: none"> • <i>Suitability</i>, which is divided into two subcharacteristics. <ul style="list-style-type: none"> – <i>Question management</i>, which is concerned with the development and management of single questions in the system. <ul style="list-style-type: none"> * Types of questions * Question structure – <i>Test Management</i>, which is concerned with the development and management of tests. <ul style="list-style-type: none"> * Test preparation * Test banks * Test Evaluation Tools * Response Analysis • <i>Security</i> • <i>Interoperability</i> • <i>Accuracy</i>

Table 6.1: Criteria for assessing functionality

concern the management of tests. Tests, in this context, can mean any group of questions combined to a larger whole. The list of subcharacteristics and criteria is shown in Table 6.1.

Suitability

The evaluation of suitability of an automatic assessment system contains more criteria for evaluation than the rest of the subcharacteristics put together. In the following, the different criteria are explained, and the TRAKLA2 system is evaluated using each criterion.

Types of questions measures the kinds of questions the system supports. The more different question types the system supports, the better. It is, for example, better if the system supports both multiple choice and text answers, than if it supported just text answers. It is also deemed important that the types of questions available can be used to measure the type right type of knowledge. If the questions cannot be used to measure the subject matter of a given course, they are useless for the assessment of the course.

The suitable types of questions depend on what the system will be used for. If the system will be employed in teaching (natural) languages, it must be able to analyse textual answers, whereas if the topic at hand is computer programming, a system that can assess computer programs can be more suitable. Of course, a system that analyses text can be used to assess student's knowledge of computer programming through essays. However, a system that "knows" computer programs is more suitable for ensuring the students have the practical

knowledge required.

Both TRAKLA and TRAKLA2 are designed to provide data structure and algorithm simulation exercises through the world wide web. This is the only type of exercise that the old system can automatically assess. Originally, TRAKLA2 had only simulation exercises, similar to those of the old system. Later, new types of exercises have been added. Currently the system includes, for example, exercises where the student is required to rank functions according to their order of growth, and code analysis exercises.

All exercises in TRAKLA2 are administered using Java applets. In the system's web environment, the data structure and algorithm applets could therefore be replaced by applets that tested some other problem domain. This way it would be possible, at least in theory, to use the web environment to assess any types of questions. TRAKLA2 has, however, been designed for data structure and algorithm simulation exercises in mind, and no other question types have been used. Still, TRAKLA2 supports more question types than TRAKLA.

Question structure measures how well the questions in the system match the educational needs to be assessed. An aspect of this is that the system should be able to store question metadata related to, for example, the topic and the cognitive level the question covers, the question's author, or its source. Another important aspect is that the system should be flexible enough to, for example, support different scoring schemes.

Question metadata is not supported in TRAKLA or TRAKLA2. Neither system was designed to include information on the source of questions, the cognitive levels they are aimed at, or the creator of each question. In both systems the instructor is free to select scoring from zero to any reasonable limit, as well as give the number of times the exercise can be submitted. However, in TRAKLA a maximum number of allowed submissions needs to be given, whereas in TRAKLA2 it is optional. If no limit is given, the system allows for unlimited number of submissions.

The flexibility of the system can be measured by looking at the question creation process and dividing the available tasks into two categories: those that *must* be done, and those that *can* be done. Generally, the less one must do in order to create a question, the better since this makes the creation easier. Also, the more one can do the better since this gives flexibility.

In TRAKLA, each exercise was implemented completely independently from others, and there was no common interface or a generalized assessment procedure. TRAKLA2 includes a generalized assessment procedure common to all exercises, and the underlying framework includes several tools for constructing new exercises. The TRAKLA2 exercises need to be able to produce only the data structures required for the student, and a model answer. The model answer is used by the generalized assessment procedure to evaluate student's answer.

Test preparation measures the tools available for gathering questions into tests, and the different test types that can be done with the system. Among the different test types mentioned in [48] are, for example, tests with random question selection, and adaptive tests where next question is selected based on student's past performance. Also important is the ability to customize the tests given to different students, in order to discourage plagiarism.

Related to test preparation, but mentioned as a criterion for the test delivery system, is the ability to manage how and when tests are showed to students. Typically a test (group of questions) can be solved only during a given time period. This period can be relatively long (i.e. a whole semester-long course) or brief (few hours for an examination). A test can also be restricted to a certain set of computers (for a closed lab session or an examination) or be freely available through the internet.

Test preparation is quite similar in both systems. A number of questions is combined into a round, which has a given deadline, and rounds are combined to a course. The same exercise can be on multiple rounds, and the rounds can have different deadlines. All exercises of a round come available when the round has opened, and are available until deadline. In TRAKLA2 it is also possible to return exercises after the deadline has passed, but the student can be penalized for doing this. In TRAKLA, late submissions had to be handled by hand or by using a separate course for make-up exercises.

Both systems automatically give each student a different instance of the exercise, so there is no need to customise the questions manually. In TRAKLA2, each student can be given a new instance of the exercise each time he tries to solve it. Both systems support customization, where each student is given different instance of the exercise, and in TRAKLA2 is even possible to give all students the same instance of the exercise.

In TRAKLA, courses are prepared by creating a new directory for the course and copying a configuration file, which defined the exercises and rounds of the course, to this directory. The round is then included manually to the web pages. Currently in TRAKLA2, the only way of creating new courses is by inserting them to the database by hand. There is, however, no need to manually include the new course on any web pages, as the system generates them automatically. An easier way of creating new courses or modifying existing one will be implemented in the future.

The **Test banks** criterion is used to measure how well the different questions can be assembled into *test banks*. A test bank is a group of questions related to, for example, the same subject, from which questions for a single test can easily be selected. A bank typically should be large enough that multiple tests can be easily created from it, and several tests can share the same bank of questions.

The TRAKLA namespace supported a maximum of 99 exercises. TRAKLA2 supports arbitrary number of exercises. Neither system has means of automatically selecting some subset of questions. Typically questions pertaining to the same subject, for example sorting, are grouped together into a round manually.

Test evaluation tools are tools for evaluating tests both before and after administration. Before administering a test, the focus is in ensuring that the test to be given is appropriate, covers the topic, that the questions are accurate and that the test is soundly formatted. Post-test evaluation is focused on whether the test worked, and whether the questions in the test were indeed adequate for the task. Post-test evaluation is very closely related to the next criterion, response analysis.

For pre-test evaluation both systems allow the instructor to do the test himself using the student's user interface. Through the student's interface the instructor can see the questions

from the student's point of view and see whether the constructed test is adequate in his opinion. For post-test evaluation, both systems offer statistics, which are further discussed below.

Response analysis measures tools for analysing both individual answers and whole tests. Responses are analyzed by using statistics and must include at least the ability to fetch the results of each student. Other possibilities are comparison of different exercises, test performance with means and distribution, lists of different correct and incorrect answers, etc.

For post-test evaluation the instructor has a number of response analysis tools available for him. In both TRAKLA and TRAKLA2, the instructor can view the student result list, which includes scores for each round, the course total and the grade awarded. Both systems also include statistics on exercises, rounds and the whole course.

In TRAKLA all statistics are gathered into one text file, which is updated at certain intervals. The statistics of the course include the number of students on a course, total number of answers produced, point distribution, as well as daily and hourly student activity. Statistics for a round include point distribution. Statistics for an exercise include number of submissions, percentage of students who have tried to solve the exercise, average points and average percentage compared to the maximum points.

In TRAKLA2 the statistics are divided into several different views the instructor can see. All statistics of the old system are also included in the new one. In addition, TRAKLA2 calculates some statistics not supported by the old system and shows, for example, the total number of submissions and the students' point average for each round. Furthermore, the system can show student point distribution and grade distribution using graphics, and is capable of comparing the results of several courses. Neither system has functionality to calculate statistical distribution, mean or deviation of student results. Overall, TRAKLA2 has more statistical information available.

In conclusion, all suitability aspects of TRAKLA have been carried over to TRAKLA2, and many have been expanded in the new system. However, the creation of new courses in TRAKLA2 is not yet as simple as it was in the old system. Also, the systems have similar tools for evaluating tests both before and after administration.

Security

Security is not divided further in [48]. The most important aspect of security is the ability to prevent cheating. Both systems include several features for preventing cheating in its various forms.

Users in TRAKLA2 are always identified with a user name and a password. Authorization to view the instructor's UI can be attained only by manipulating the user data in the database. Each user can access only his own user account, and only instructors can access the instructor's user interface. The user names and passwords are protected using the secure HTTPS protocol, and security is handled by the web server. TRAKLA2 also logs all actions where some aspect of a course or student results is modified. Therefore, if someone gained

unauthorized access to the instructor's user interface, his actions can later be spotted.

In TRAKLA, the users are identified only with a user name in the student's user interface. It is therefore possible to try and solve another user's exercises. However, during the decade of use there were no reports of anyone trying to abuse the system. The instructor's user interface is available only locally on the server machine. The instructor uses SSH protocol to log into the server machine and has various scripts and other tools for accessing the instructor's user interface. The security is handled by the operating system.

In both systems, each student gets an unique instance of each exercise, making it impossible for the students to copy the answers from each other. In TRAKLA, the exercises are unique for each student, but in TRAKLA2 each time a student tries to solve an exercise, he gets a new instance of it.

TRAKLA system uses CGI-scripts, which are sometimes considered to be a security risk. However, during the many years of operation, there was no indication that the script (which was used to create the web page showing an exercise) caused any security problems.

TRAKLA2 uses the Tapestry framework and a separate Java-based server for communicating with the user. So far, there has been one security hole discovered by a student. Using the hole it was possible to gain points greater than the maximum from an exercise, and to substitute one exercise for another. The hole was quickly found and fixed, and no further problems have been discovered. Both in TRAKLA and TRAKLA2, all student answers are stored and can later on be checked to see if there are any cheating.

In over a decade of use, there were no serious security issues with the TRAKLA system. TRAKLA2 has now been used for two years, and there has been only one security problem with it. Both systems therefore seem to be rather secure.

Interoperability

For interoperability, the ability to use and provide services is deemed to be more important than exporting or importing data [48]. However, Valenti et al. seem to focus more on systems that are to be integrated into larger course management systems. In contrast, most of the systems described in this thesis are designed to be stand-alone.

Neither system has functionality to automatically import data from another system, use services provided by another system, or provide such services. Both systems are capable of exporting student results in a tabulator-separated text format, which is understood by a large number of different programs. There has been discussion on modifying TRAKLA2 to use the centralized user identity service provided by Helsinki University of Technology. So far no action has been done to implement this.

Accuracy

It should be noted that Valenti et al. do not use *accuracy* to evaluate assessment systems. For an automatic assessment system, a possible measure of accuracy would be the output given

by the assessment process. For summative assessment, the output grade should correlate with the student's level of understanding concerning the subject matter. The better the student knows the topic, the better grades he should get. For formative assessment, the feedback given should reflect the possible mistakes and the overall level of knowledge the student has.

6.2.2 Usability

Usability is defined as "The capability of the software product to be understood, learned, used and attractive to the user when used under specified conditions"[1]. Usability measures how easy the system is to learn, to use for the purpose it was built for and how "good" the users will see the software.

The subcharacteristics of usability include *understandability*, *learnability*, *operability* and *attractiveness*. Of these, two subcharacteristics, understandability and operability, can be combined together. For the combined subcharacteristic, the *ease of editing* can be measured. For this criterion, the number of features in the exercise editor (i.e. WYSIWYG capability, clipboard, etc.), the ability to include pictures or symbols, etc. are measured. The amount of programming knowledge required is also taken as a measure. For learnability, the existence of tutorial, help and training facilities are cited as important.

The instructor uses the TRAKLA system through the command line. The system has no dedicated instructor's user interface, but a number of scripts that can be used to, for example, compile student results. For tasks that do not have a dedicated script, standard UNIX tools like cat, grep, and less can be used.

In TRAKLA2, the instructor has a dedicated, graphical user interface available. However, the current version of the UI does not include all functionality an instructor needs for creating and managing courses. New courses, for example, have to be created by manually modifying the database.

TRAKLA2 includes documentation for people who wish to create new exercises to the system. Neither system has documentation for the use of the instructor's user interface.

In both systems, the exercise descriptions are stored as HTML code. In TRAKLA, each description is stored in a different file on the hard disk, and can be edited using a text editor. In the new system, the descriptions are stored in a database, and can be edited through the web interface. In both systems, the exercise descriptions can include pictures, figures, and links to other material. Currently, for all other editing tasks in TRAKLA2, the instructor needs to possess knowledge of the SQL language and needs access to the TRAKLA2 server machine.

Attractiveness, which measures the impact the UI design has on the user, is not discussed in [48]. In the case of an automatic assessment system, attractiveness could be divided into two categories: attractiveness to instructors and attractiveness to students. The instructors are the user group who make the decision whether the system is taken into use. It is, however, students, who are by far the most populous user group in the system. Therefore, the instructor should consider the system's attractiveness to students when making the decision

whether to take a system into use. Attractiveness of the TRAKLA2 instructor's user interface was not measured for this thesis, as it is hard to measure the attractiveness of a system you have implemented yourself. However, the TRAKLA2 system has been taken into use in several institutions besides Helsinki University of Technology, and there have been no major complaints about the user interface. This would indicate that the system is considered at least somewhat attractive by several instructors.

6.2.3 Reliability

Reliability is defined as “the capability of the software product to maintain a specified level of performance when used under specified conditions”[1]. It measures the stability of the software, its ability to withstand software faults (bugs), and recover from fault situations.

Reliability is divided into three important subcharacteristics: *maturity*, *fault tolerance* and *recoverability*. Valenti et al. do not discuss the subcharacteristics independently. They state that reliability is very important, and that special care should be done to ensure that data is not lost because of a fault. This is deemed to be especially important in examinations and closed labs.

The three subcharacteristics are very closely linked. Maturity measures the system's ability to avoid faults, while fault tolerance is the system's capability to continue operation in the case of a fault. Recoverability measures the system's ability to recover from failures. There are two types of faults that are specific to an assessment system: faults in the assessment procedures and faults in exercises.

Faults in assessment procedure introduce systematic errors in all assessment. This, in turn, can lead to incorrect marking of exercises or bad feedback. Therefore, faults in the assessment procedure must not be tolerated. Faults in single exercise may lead to similar behaviour in that exercise, and must not be tolerated either. Other faults, like imprecise exercise description, are also possible.

One of the most important aspects of reliability in automatic assessment systems is the electronic trail left by the student as he solves an exercise. In a well-designed system, information about the student's exercise is stored in several places, and in the case of fault, the evaluation of the exercise can be restarted from this information.

In the original TRAKLA, the student received his exercises through email, solved them by hand, and submitted his answers through email. The submission was then assessed and feedback was sent to the student. In this system, there was a permanent electronic trail from each operation. The student could save the original email containing the exercise descriptions, the emails containing his submissions and the feedback messages received from the system. If the system experienced a fault, the student could easily resubmit his exercises.

With the introduction of WWW-TRAKLA, the complete electronic trail disappeared. When the student submitted his answer through WWW-TRAKLA, the answer was sent to the server, but not immediately stored locally. WWW-TRAKLA sent a copy of the submission to the student, but if this mail disappeared, the submission could be lost. However, from

the moment the submission was received and stored at the server, the trail became reliable. It should be noted that using WWW-TRAKLA, the student submitted each exercise separately, while in the email-based system it was possible to submit multiple exercises simultaneously. Therefore, a fault would lose only one submission.

In TRAKLA2 all exercises are done through web-based applets. The exercise is assessed and submitted to TRAKLA2 server immediately when the student pushes the submit-button. The system informs the student whether the submission was successfully stored or not. Unfortunately, if the system crashes before the student submits his answer, the student has to wait, without closing or modifying the applet window, for the system to come back on-line, or the work is lost. On the other hand, the new system assesses the exercises immediately after they have been submitted, instead of running the assessment procedure at certain intervals. Therefore if the data is received, the system stores the assessment and the given result immediately to the database. Furthermore, the student can lose a maximum of one submission for a server fault, since each applet contains only one exercise.

Systematic faults in the assessment have not been found in either system. However, when the TRAKLA system was moved from one platform to another in the year 2001, a systematic fault was found in the assessment procedure for the quicksort exercise. The fault was due to an array that had been left uninitialized in the program code. In the old platform the array had been automatically initialized, but this did not happen on the new platform.

During the development of TRAKLA2, there were several faults found in the exercise descriptions. This was the main reason for adding the functionality for modifying the exercise descriptions. Also, in one algorithm analysis exercise a systematic fault was found. In this exercise, one correct solution was missing from the possible solutions. Other AA-system specific faults have not been discovered.

The TRAKLA2 system is still under development and therefore is not yet completely fault-free. For example, in the current version of TRAKLA2, if the student uses the reload-button of his web browser, he can be moved from one page in the dynamic web environment to another, or be redirected to an error page. This is due to the way the Tapestry framework creates URLs leading to other pages in the system.

In the case of power failure, TRAKLA automatically comes back up when the computer is restarted, since system is run by a UNIX cron script. TRAKLA2, in contrast, will need its own start-up script added to the list of automatically started services, if the instructor wants TRAKLA2 to automatically restart. Such script is currently being developed but has not yet been taken into use in the production version.

Currently the old TRAKLA system is still more reliable than TRAKLA2. The new system is, however, more complicated than the old one, since it includes a dynamic web environment and a database, and is still under active development.

6.3 Conclusion

The new TRAKLA2 system contains several improvements over its predecessor. It, for example, allows for more types of exercises, has some additional options when creating new courses, and contains more types of statistics than the old system. It is, however, still under development and is not as mature as TRAKLA. There are still several tasks in TRAKLA2 for which there is no user interface, forcing the instructor to manually access and modify the database. There are also several aspects of the system that could be improved. For example, the statistics given by TRAKLA2 could include the statistical distribution, mean or deviation of user results for a course.

The future work on the TRAKLA2 system should include, at least the functionality an instructor needs to create and manage courses in the system without having to modify the contents of the database by hand.

Chapter 7

Summary

In this work I have described the instructor's user interface to the TRAKLA2 automatic assessment system. In the construction of the interface, I have taken into account the solutions employed in other automatic assessment systems, and the insight I gained by interviewing experts. In the design of TRAKLA2, I have tried to focus on features that have proven to be useful in other systems, and features that the instructor clearly requires.

In the related work in Section 3, I discovered a number of important functionalities found in other systems. These functions are described in Section 3.7. However, since the literature contained only very brief descriptions of these features, I made a number of expert interviews, in order to gain more insight on what is needed for automatic assessment systems. The interviews are described in Section 4. In the design of the TRAKLA2 system, described in Section 5.2, I have taken into account the results of my study.

In Section 5.2.2, I show that the use cases from my previous experience with automatic assessment systems are very closely related to the results of the literature study and the interviews. Therefore it is clear that both the functionality discovered, and the use cases considered, contain a rather large amount of the functionality that an instructor's user interface to an automatic assessment system needs.

In Section 5.5, I describe the implementation of the new instructor's user interface. The code in the implementation is divided into two groups: code for the web environment and code for the data model. The data model is independent of the web pages, but the web environment uses the data model rather extensively. The instructor's user interface, described in Section 5.5.2, contains a number of views through which the instructor can access the available functionality.

I have also evaluated the new system, both in practice by having it used in a Data structures and algorithms course, and by using the ISO 9126 model for software quality. I have compared the system to its predecessor, TRAKLA, and found that while TRAKLA2 includes several improvements it is not yet as mature as the old system, and still lacks some important functionality.

7.1 Future Work

The TRAKLA2 environment is far from ready yet. As mentioned in Section 6.2.3 the system still has some bugs that need to be fixed. Also, the lack of functionality for creating new courses and managing the existing ones effectively needs to be fixed. Only when it is possible to use the system without having to manually modify the database, can the system be considered ready to be used in other institutions without constant help and assistance by the developers.

Bibliography

- [1] Software engineering – product quality. Standard, International Standardization Organization, 2003.
- [2] Kirsti Ala-Mutka, Toni Uimonen, and Hannu-Matti Järvinen. Supporting students in C++ programming courses with automatic program style assessment. *Journal of Information Technology Education*, 3:245–262, 2004.
- [3] Jon Barwise and John Etchemendy. Computers, visualization and the nature of reasoning. In *The Digital Phoenix: How Computers are Changing Philosophy*, pages 93 – 116. Blackwell Publishing, 1998.
- [4] S. Benford, E. Burke, E. Foxley, N. Gutteridge, and A. Mohd Zin. Ceilidh: A course administration and marking system. In *Proceedings of the 1st International Conference of Computer Based Learning*, Vienna, Austria, 1993.
- [5] Michael Blumenstein, Steven Green, Ann Nguyen, and Vallipuram Muthukumarasamy. An experimental analysis of GAME: a generic automated marking environment. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pages 67–71. ACM Press, 2004.
- [6] S. Bridgeman, M. T. Goodrich, S. G. Kobourov, and R. Tamassia. PILOT: An interactive tool for learning and grading. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*, pages 139–143. ACM Press, New York, 2000.
- [7] Jill Burstein, Claudia Leacock, and Richard Swartz. Automated evaluation of essays and short answers. In *Proceedings of 5th Annual CAA Conference*, Loughborough, 2001.
- [8] J. Carter, J. English, K. Ala-Mutka, M. Dick, W. Fone, U. Fuller, and J. Sheard. ITICSE working group report: How shall we assess this? *SIGCSE Bulletin*, 35(4):107–123, 2003.
- [9] James Dalziel. Enhancing web-based learning with computer assisted assessment: Pedagogical and technical considerations. In *Proceedings of the Fifth International Computer Assisted Assessment Conference*, Loughborough, 2001. Loughborough University.

- [10] Ulrich Endriss. An interactive theorem proving assistant. In *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference, TABLEAUX'99*, pages 308 – 313, 1999.
- [11] John English. Automated assessment of gui programs using JEWEL. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pages 137–141. ACM Press, 2004.
- [12] Nils Faltin. Structure and constraints in interactive exploratory algorithm learning. In Stephan Diehl, editor, *Software Visualization: International Seminar*, pages 213–226, Dagstuhl, Germany, 2002. Springer.
- [13] George E. Forsythe and Niklaus Wirth. Automatic grading programs. *Communications of the ACM*, 8(5):275–278, 1965.
- [14] Colin Higgins, Pavlos Symeonidis, and Athanasios Tsintsifas. The marking system for CourseMaster. In *Proceedings of the 7th annual conference on Innovation and Technology in Computer Science Education*, pages 46–50. ACM Press, 2002.
- [15] Juha Hyvönen and Lauri Malmi. TRAKLA – a system for teaching algorithms using email and a graphical editor. In *Proceedings of HYPERMEDIA in Vaasa*, pages 141–147, 1993.
- [16] D. Jackson and M. Usher. Grading student programs using ASSYST. In *Proceedings of 28th ACM SIGCSE Tech. Symposium on Computer Science Education*, pages 335–339, San Jose, California, USA, 1997. ACM Press, New York.
- [17] Tomi Janhunen, Toni Jussia, Matti Järvisalo, and Emilia Oikarinen. Teaching smullyan’s analytic tableaux in a scalable learning environment. In Ari Korhonen and Lauri Malmi, editors, *Kolin Kolistelut/Koli Calling. Proceedings of the Fourth Finnish/Baltic Sea Conference on Computer Science Education*, pages 85 – 94. Helsinki University of Technology, 2004.
- [18] Burke Johnson and Larry Christensen. *Educational Research Quantative, Qualitative and Mixed Approaches*. Pearson Education Inc., 2nd edition edition, 2004.
- [19] Tuomo Kakkonen and Erkki Sutinen. Automatic assessment of the content of essays based on course materials. In *Proceedings of the International Conference on Information Technology: Research and Education 2004 (ITRE 2004)*, pages 126–130, London, UK, 2004.
- [20] Ari Korhonen. World wide web (www) tietorakenteiden ja algoritmien tietokoneavusteisessa opetuksessa. Master’s thesis, Helsinki University of Technology, 1997.
- [21] Ari Korhonen and Lauri Malmi. Algorithm simulation with automatic assessment. In *Proceedings of The 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'00*, pages 160–163, Helsinki, Finland, 2000. ACM Press, New York.

- [22] Ari Korhonen, Lauri Malmi, and Panu Silvasti. TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. In *Proceedings of Kolin Kolistelut / Koli Calling – Third Annual Baltic Conference on Computer Science Education*, pages 48–56, Joensuu, Finland, 2003.
- [23] Ari Korhonen, Lauri Malmi, Panu Silvasti, Ville Karavirta, Jan Lönnberg, Jussi Nikander, Kimmo Stålnacke, and Petri Iiantola. Matrix - a framework for interactive software visualization. Research Report TKO-B 154/04, Laboratory of Information Processing Science, Department of Computer Science and Engineering, Helsinki University of Technology, Finland, 2004.
- [24] Mikko-Jussi Laakso, Tapio Salakoski, Ari Korhonen, and Lauri Malmi. Automatic assessment of exercises for algorithms and data structures – a case study with TRAKLA2. In *Proceedings of Kolin Kolistelut / Koli Calling – Fourth Finnish/Baltic Sea Conference on Computer Science Education*, pages 28–36. Helsinki University of Technology, 2004.
- [25] H. Laine. SQL-trainer. In *Proceedings of Kolin Kolistelut / Koli Calling – First Annual Baltic Conference on Computer Science Education, Report A-2002-1*, pages 13–17, Joensuu, Finland, 2002. University of Joensuu.
- [26] Lauri Malmi, Ville Karavirta, Ari Korhonen, Jussi Nikander, Otto Seppälä, and Panu Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267 – 288, 2004.
- [27] Lauri Malmi, Ari Korhonen, and Riku Saikkonen. Experiences in automatic assessment on mass courses and issues for designing virtual courses. In *Proceedings of The 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, pages 55–59, Aarhus, Denmark, 2002. ACM Press, New York.
- [28] Thomas J. McCabe. A complexity measure. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*, page 407, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.
- [29] A. Mitrovic. Learning SQL with a computerized tutor. In *Proceedings of the 29th SIGCSE Technical Symposium of Computer Science Education*, pages 307–311. ACM Press, New York, 1998.
- [30] Sami Mäkelä and Ville Leppänen. Japroch: A tool for checking programming style. In *Proceedings of Kolin Kolistelut / Koli Calling – Fourth Finnish/Baltic Sea Conference on Computer Science Education*, pages 151–155. Helsinki University of Technology, 2004.
- [31] Peter Naur. Automatic grading of students' ALGOL programming. *BIT* 4, pages 177–188, 1964.
- [32] E. B. Page. The imminence of grading essays by computer. *Phi Delta Kappan*, pages 283–243, January 1966.

- [33] Abelardo Pardo. A multi-agent platform for automatic assessment management. In *Proceedings of The 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, pages 60–64, Aarhus, Denmark, 2002. ACM Press, New York.
- [34] Jon A. Preston and Russell Shackelford. Improving on-line assessment: an investigation of existing marking methodologies. In *Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, pages 29 – 32. ACM Press, New York, 1999.
- [35] Jon Anderson Preston. Evaluation software: improving consistency and reliability of performance rating. In *ITiCSE-WGR '97: The supplemental proceedings of the conference on Integrating technology into computer science education: working group reports and supplemental proceedings*, pages 132–134, New York, NY, USA, 1997. ACM Press.
- [36] Michael J. Rees. Automatic assessment aids for pascal programs. *SIGPLAN Notices*, 17(10):33–42, 1982.
- [37] S. K. Robinson and I. S. Torsun. The automatic measurement of the relative merits of student programs. *ACM SIGPLAN Notices*, 12(4):80–93, 1977.
- [38] Riku Saikkonen, Lauri Malmi, and Ari Korhonen. Fully automatic assessment of programming exercises. In *Proceedings of The 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'01*, pages 133–136, Canterbury, UK, 2001. ACM Press, New York.
- [39] S.D.Benford, E.K.Burke, and E.Foxley. Courseware to support the teaching of programming. In *Proceedings of the Conference on Developments in the Teaching of Computer Science*, pages 158–166. University of Kent, April 1992.
- [40] Panu Silvasti. Tilastollisen datan kerääminen algoritmisten harjoitustehtäväsovelmiin käytöstä (Collecting statistical data of the usage of algorithmic exercise applets). Master's thesis, Department of Computer Science and Engineering, Helsinki University of Technology. (in Finnish), 2003.
- [41] Panu Silvasti, Lauri Malmi, and Petteri Torvinen. Collecting statistical data of the usage of a web-based educational software. In *Proceedings of the IASTED International Conference on WEB-BASED EDUCATION*, pages 107–110, Innsbruck, Austria, 2004. IASTED.
- [42] Raymond M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.
- [43] Pete Thomas. The evaluation of electronic marking of examinations. In *The Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education ITiCSE'03*, pages 50 – 54, Thessaloniki, Greece, 2003. SIGCSE, ACM Press.

- [44] Nghi Truong, Paul Roe, and Peter Bancroft. Static analysis of students' java programs. In *Proceedings of the sixth conference on Australian computing education*, pages 317–325. Australian Computer Society, Inc., 2004.
- [45] Vincent Tscherter. *Exorciser: Automated Generation and Interactive Grading of Structured Exercises in the Theory of Computation*. PhD thesis, Swiss Federal Institute of Technology, 2004.
- [46] Vincent Tscherter, Reto Lamprecht, and Jürg Nievergelt. Exorciser: Automatic generation and interactive grading of exercises in the theory of computation. In *Fourth International Conference on New Educational Environments*, pages 47–50, 2002.
- [47] Athanasios Tsintsifas. *A Framework for the Computer Based Assessment of Diagram Based Coursework*. PhD thesis, University of Nottingham, 2002.
- [48] Salvatore Valenti, Alessandro Cucchiarelli, and Murizio Panti. Computer based assessment systems evaluation via the iso9126 quality model. *Journal of Information Technology Education*, 1(3):157–176, 2002.
- [49] Anne Venables and Liz Haywood. Programming students need instant feedback! In *CRPITS '20: Proceedings of the fifth Australasian conference on Computing education*, pages 267–272, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.
- [50] Robert Williams. Automated essay grading: An evaluation of four conceptual models. In A. Herrmann and M. M. Kulski, editors, *Expanding Horizons in Teaching and Learning. Proceedings of the 10th Annual Teaching Learning Forum*, 2001.