Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Athena, a large scale programming lab support tool

Anton Jansen, University of Groningen, The Netherlands

Abstract

Providing a programming lab to a large group of students requires a lot of effort. Often in these cases additional staffing is required to provide students with enough feedback on their work. However, due to resource constraints this is not always possible. This paper presents Athena, a lab support tool that reduces the effort required for programming labs and provides students with adequate and timely feedback. Athena reduces the need for additional staffing and provides students with a better learning experience. Furthermore, it allows for new educational forms for large student groups like programming lab exams.

Keywords: Programming labs, automated testing

1 Introduction

A programming cours is often supported by one or more programming labs. The exercise(s) of the labs are meant to teach students the skill of programming and the concepts of a certain programming paradigm. For large groups of students, the lab part of a programming course becomes laborious. This is mainly due to the fast amount of review work that needs to be done to provide students with feedback on their work.

In this paper, Athena is presented. Athena is a system that supports large scale programming labs and reduces the effort required to run such labs. Athena achieves this by automating certain laborious administrative tasks and by providing automated feedback towards the students on their work. The benefits of Athena are twofold: it reduces the need for additional staffing and provides a better learning experience for students.

The contribution of this paper consists of three parts. First, it introduces the Athena system and argues for the two aforementioned benefits of improved learning and reducing effort. Second, it provides considerations and guidelines how Athena can be integrated into programming courses. Third, it presents an in-depth evaluation of the pro's and con's of a support system like Athena.

The rest of this paper is organized as follows. First, a closer look is taken at programming courses, programming labs, and their laborious tasks. Next in section 3, Athena is introduced. Section 4 explains how Athena can be used in programming courses. After which the paper presents an overview of the lessons learned in section 5 and concludes in section 6 with conclusions.

2 Programming courses

Programming can be learned in many different ways. In this section, a look is taken at supervised learning in the form of programming courses. Two different strategies of teaching programming courses can be identified: the algorithm perspective and the component perspective.

In the algorithm perspective, the focus is on learning students to think in steps to build up an algorithm for a particular problem. On the other hand, in the component perspective the focus is on how predefined functionality (e.g. algorithms) in the form of components can be composed together to form an application.

However, both strategies try to achieve similar goals. Programming courses are not meant to learn a particular programming language to a student, but more the concepts behind a particular programming language paradigm. For example, in Object Oriented Programming (OOP) the concepts of methods and inheritance, in functional programming the concept of folding.

Programming courses additionally teach students to make transformations from the problem domain (the real world) to the solution domain (a programming paradigm). For example, analytic, problem solving, and divide & conquer techniques can be learned to ease this difference in worlds between

computers and the real world.

Furthermore, programming courses try to develop the student's sense for good and bad programming practices. This includes considerations of aesthetics, quality, and trade-offs made. Students need to become aware of these issues and have the ability to discuss and reason about them.

Learning how to program is a difficult and time-consuming process. The main reasons for this are: (1) the abstract nature of, (2) the required precision for, (3) and the skill required, to use concepts of programming languages. So how can we deal with these problems in a programming course?

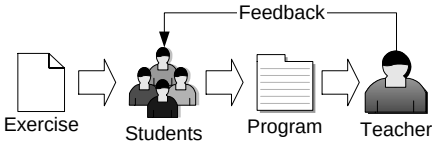

Figure 1: Programming labs

Many teachers use the solution to have a series of programming labs associated with their programming course. In these programming labs, programming exercises are used to train students in programming. The exercises are meant to train the students in putting the theoretical knowledge of programming concepts and combinations of them into practice. Often there is a need to have many small exercises to train concepts in isolation, before complex and particular combinations are tried.

In figure 1, the process of a programming lab is depicted. First, one or more exercises are defined by a teacher. They serve as the main input for the students. It is the student task to make a program as described in the exercise. This program is communicated to a teacher, which reviews and/or grades the program. The remarks of the teacher are communicated back to the students as feedback.

Reviewing and/or grading student programs for these exercises is a time-consuming job. The main reason for this is the sheer amount of code that needs to be examined. For example, 60 students are more than capable of writing over 7500 Lines Of Code (LOC) in a time span less than 3 hours for some small exercises. For large groups of students, the amount of review work can become so great that it prohibits a timely feedback from the teacher(s) towards the students. Consequently, the learning experience of the students becomes less than optimal in these cases.

For large groups of students, there is a clear need for more staffing to prevent such situations. However, due to resource constrains this is not always possible or economical feasible. Consequently, a different approach is needed for large groups, which reduces effort for a teacher, but does not compromise the learning experience of the student.

3 Athena

Athena is a computer system to support large scale programming labs. It reduces the required effort for a teacher to run these labs, while enriching the students learning experience. Figure 2 presents an overview of how programming labs with the support of Athena work. The main difference with the process of normal programming labs (see figure 1) is the additional feedback to students and filtering of information to the teacher.

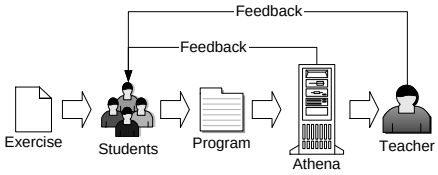

Figure 2: Programming labs with Athena

In the rest of this section, different aspects of Athena are presented. First, section 3.1 outlines the basic working process of Athena and it's supporting tools. Next, section 3.2 takes a closer look at the automated testing facilities of Athena, which creates the additional feedback towards the students.

3.1 Athena system

The basic working process of Athena (see figure 2) consists of students submitting their work to the Athena system using the *submission client* (see figure 3). The submitted work is automatically tested by the *arbiter*, which provides the additional feedback to the students using the *student web interface*. The teacher can examine the submitted work and configure the automated testing by the use of the Athena *management tool* (see figure 4). Following is a more detailed description of these supporting tools and their function:

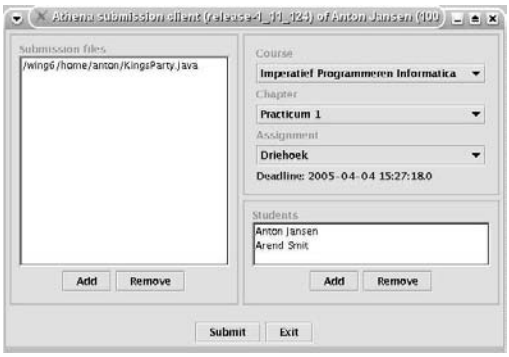

Figure 3: The submission client

Submission client The submission client (see figure 3) is a stand-alone Java application, which students use to submit their work to the Athena system. The student can select the files or directories containing their solution. To reduce garbage a filter can be defined, which filters out unwanted material (e.g. binaries). Optionally, in the case of group work, the student can select on or more fellow students that are responsible for the submitted work as well. This relieves the staff of the burden to maintain a proper group administration. For large groups of students this turns out to be a time-consuming and error-prone task, as groups tend to change quite often due to sickness, dropouts, and arguments among students.

Arbiter The arbiter is the heart of the Athena system, as it judges the submissions automatically. For each exercise that should be automatically tested, a testset consisting of one or more tests is defined in Athena. The arbiter executes these various tests and collects the result in a judgement about the submission. An elaborate description of the various means to test is presented in section 3.2.

Student webinterface Students receive the feedback from the Athena system through the *student webinterface* [Athena website]. This web-based interface provides a student with an overview of his or her submissions and their status in the Athena system. Futhermore, the judgement (created by the arbiter) can be examined. It includes detailed information about the executed tests, their results, expected results, and additional hints defined by the teacher.

Management tool The management and configuration of the various elements of Athena is done in the management tool (see figure 4). This tool allows teachers to configure the Athena system to their needs and manage their programming courses. For example, managing deadlines, creating and modifying courses and exercises, and printing and reporting are done using the management tool.

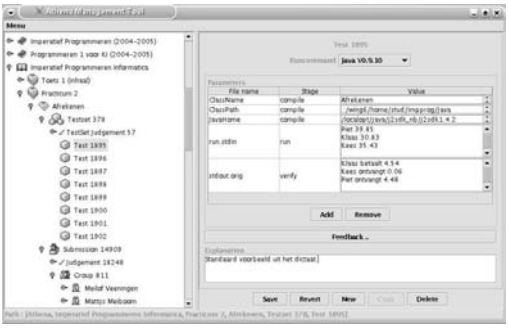

Figure 4: The management tool

3.2 Automated testing

The automated testing system of the arbiter is the heart of the Athena system. It generates the additional feedback for the students (see figure 2), thereby reducing the burden on the staff. Athena provides a testing *environment* as opposed to a single testing *framework*, as Athena needs to be pro-

gramming language and platform independent. In this testing environment, different testing frameworks are specified for the different combinations of programming languages and platforms.

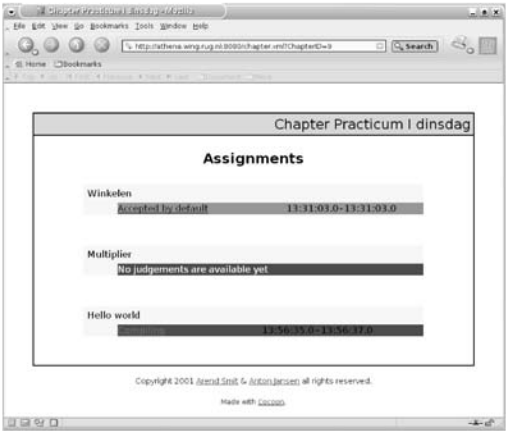

Figure 5: Student webinterface

Teachers configure a testing framework in the management tool to define an automated test for their particular exercise. Testing frameworks have been defined and used for multiple programming languages, which include frameworks for Modula-3, Pascal, Java, and C.

The test environment of Athena provides an infrastructure for the testing frameworks to report their findings to the system and optionally back to the student. Consequently, the Athena is completely programming language independent and therefore could be used for functional and logic programming languages as well.

Various testing techniques can be used in Athena. The following list gives an overview of the techniques, which have been used in Athena:

Don't do it From experience, we have learned that it is sometimes convenient not to test at all. Consequently, the Athena system then becomes an administration tool, which is specialized for programming courses.

Compile The work of the student is tested with a compiler to see whether it compiles or not. Optionally, warnings of the compiler can be allowed or disallowed. Surprisingly enough, many submissions fail to pass this test, due to small mistakes that are very hard to detect by a human.

Run The compiled work of the student is run to see whether it does not generate any run-time anomalies (e.g. segmentation faults, out-of-bounce exceptions etc.). Furthermore, it can be tested whether the program ends in a predefined time, to have an indication for end-less loop constructs.

Output The generated output of a student program on a given input can be automatically tested as well. The used options in Athena frameworks for output testing include:

- **Textual** The output of the student program is textual compared to the output of a reference program or a predefined output. Texts can be compared in numerous ways, the one used often in Athena is a context difference, which ignores whitespace (i.e. spaces and tabs).
- **Numerical** When the output consists of numbers, numerical tests can be made. The numbers of the student can be compared against predefined numbers or numbers generated by a reference solution. Depending on the absolute or relative difference the test can bet set to fail or succeed.
- **Dedicated** A special made program could be used to test the student program and interpreted the generated output in a domain specific way. For example, an application that automatically tests whether a circle is drawn on the screen by the application.

Performance The performance of the student program can be measured in time, memory usage, etc. In the use of Athena, this is primarily used to automatically test whether the student uses a particular efficient algorithm (e.g. quicksort).

Student The idea is here to let the students make the test themselves in a predefined testing framework. For example, Athena has a JUnit [JUnit] testing framework. This allows students to test their own Java application in a uniform way. The educational benefit is that students are forced to think

about testing, which is an inherent part of software development. The teacher now only needs to check the coverage of the testset of a student to ensure functional correctness.

4 Courses with Athena

In the previous section, the Athena system was introduced. This section presents how the Athena system can be integrated into programming courses and the consequences of this integration.

Although testing is regarded in software engineering as a crucial process to insure quality, most programming courses regard it as a separate topic, which lies often out of scope. If the automated testing facilities of Athena are used in a programming course, testing becomes an inherent part of the course. However, for automated testing to work in a course, a couple of issues need to be dealt with:

Interactions with the environment This is the most constraining, hard, and often an impossible aspect to achieve. For automated testing the interaction between the student program and its environment needs to be specified. Otherwise, the testing cannot be automated.

Transforming an idea to an exercise text that defines these required interactions proves far from trivial. The level of detail required prohibits exercises using many different types of interactions. Furthermore, the "catch" of some exercises lies in these interactions. Specifying them in the exercise text destroys the purpose of the exercise, as it gives the "catch" away.

Several strategies exist to ease interaction with the environment. The easiest and most often employed one is to provide examples of the intended interaction. A more specialized strategy is to provide the student with a framework, which already takes care of these issues. However, learning the framework takes time and requires knowledge sometimes not available to the students (e.g. in a starters programming course), therefore this strategy is not always a viable option. Another strategy is not to specify these interactions beforehand. This prevents testing of predefined tests, but still allows students to make their own testset (see 3.2).

Accuracy The texts of an exercise need a high degree of accuracy to prevent ambiguity. As ambiguous situations makes automated testing for these cases near to impossible. Furthermore, to deal with the issue of the previous point the interaction with the environment needs to described accurate and in detail.

Covering and incremental testset To benefit from the automated testing facilities of Athena a good testset is required. A good testset for exercise covers the various issues of an exercise. Furthermore, it is defined in an incremental way, such that first some trivial cases are tested, after which the more complex ones come. This is needed to make the relationship between the tests and the student program clear, thereby providing a smooth learning path. Often tests are made to check whether the wrong conceptual solution has been chosen. Feedback associated with these tests are defined to include instructions or remarks what the conceptual mistake likely would have been.

Reference solution Determining the coverage and suitability of a testset is difficult. Furthermore, experience learns that many mistakes are made in the definition of a test. A reference solution is therefore a crucial tool to test the testset and to provide validation of the suitability of the exercise.

The issues presented make the initial effort to define an exercise for automated testing significant higher than without. An automated testing environment like Athena pays-off in the correction phase. In courses using Athena, only submissions that have *passed* the testset are corrected and graded by the teacher. This significantly reduces the effort, as the teacher can assume the functional correctness of the student work.

Athena allows for new forms of benchmarking student performance. Programming exams can be held in which students get a fixed amount of time (e.g. 3 hours) to solve as many exercises as they can. The focus of the exam is to test the student's skill in *using* and applying a programming language, rather

than the theoretical knowledge about a language tested with traditional paper and pencil exams.

5 Lessons learned

Athena has been in use since 2001 for 3 or 4 different courses each year. Over 1000 different students from 4 different faculties have used the system so far. In this section, the experiences and lessons learned during this time are presented. First, the positive and negative sides are presented from the perspective of a student, after which the same is done for the teacher. From a student perspective, there are the following observations:

- **Hard, competitive** Submitted work of a student is only graded once it *passes* the automated tests. This makes the system very harsh towards students, as there is no consideration of (inadequate) work that has not passed the testset. This hardness also creates a competitive atmosphere, as students compare their results, which are instantly available.

- **Fraud** The hardness of the automated testing makes the temptation to fraud bigger. The large number of students for which Athena is used complicates the detection of such cases.

+ **Precise working attitude** The automated testing part of Athena encourages students to pick up a precise working attitude. The feedback of the testing systems confronts the student with mistakes of their program, independent of how small and trivial they might be.

+ **Objective** Athena's automated testing is objective, as opposed to the manual checking of functional correctness by a group of teachers. Each teacher has it's own way of finding flaws within a program. Consequently, this can leads to situations where a certain type of fault is detected by one, but not by the others. Hence, the determination of functional correctness becomes subjective. To fight this subjectivity, considerable effort needs to be spent in harmonizing and communicating functional correctness requirements, something that is not needed in Athena.

+ **Learning experience** The fast feedback of the system provides a better learning experience for students. No longer, they have to wait one or two weeks to get (detailed) feedback on their work. The provided feedback can be *directly* used in the same lab sessions. This prevents situations where a student makes the same systematic mistake during several lab sessions, as the grading process often fails to match the lab sessions pace. Furthermore, the quality of the student work has been *significant* higher than in cases where Athena is not used.

There are also some lessons learned from a teacher perspective:

- **Reduced exercise freedom** As already pointed out in section 4, the use of automated testing constrains potential exercises.

- **Dependability** The automated testing of Athena makes a course very dependent on a specific system environment. Ideally, the development environment of the student matches the testing environment, as a detected fault then can be replicated by the student. Our experience has learned there can arise many subtle differences due to a different mix of compiler and libraries versions, operating systems, access rights, and other system variables. Furthermore, these environments evolve, thereby sometimes requiring evolution of the testset of an exercise as well.

+ **Quality of grading** As the Athena system handles the functional aspect of the student work, the focus of grading shifts from functionality correctness to quality aspects of the work.

+ **Saves time** Although the initial investment is high to setup a course with Athena, the investment pays off for large groups. The automat-

ing of part of the administration and the automated testing saves time. The break-even point lies around groups of 40 students. If a course in can be reused for several years, this number can get as low as 25.

+ **Scalability** An advantage of Athena is its scalability. Courses with over 160 students participating use Athena. The staffing for the weekly labs of these courses consists of three teaching assistants working 8 hours a week. Only additional computers are needed to ensure fast feedback from the automated testing.

6 Conclusions

Programming labs are an essential part of programming courses. The exercises used in programming labs provide students with insight into the workings of the concepts of a programming paradigm. Furthermore, students learn to use a particular programming language, thereby training their programming skills.

An important part of the learning experience of students is the feedback they receive from the teacher on their work on the exercises. For large groups of students providing this feedback can become a problem. The review work becomes so great for a teacher that it no longer can be provided on time. Consequently, the learning experience for students becomes less than optimal.

This paper presented Athena, a support tool for large scale programming labs. Athena automates common administrative tasks like group lists, deadline management, result administration, and printing of student programs. Furthermore, Athena automatically determines the functional correctness of the work of students using automated testing.

The automated testing facilities of Athena improve the student learning experience, as more and timely feedback can be provided to the student. Furthermore, the automated testing significantly reduces the review time of a teacher. Athena takes care of the functional part of the student work. Consequently, a teacher can spend his or her precious time on reviewing the non-functional quality aspects (e.g. coding style and documentation).

Both automation of administrative tasks and the automated testing facilities significantly reduce the required effort from teacher(s) for large groups of students. As the initial effort required to define the necessary tests for automated testing is relative high, Athena starts paying off for groups larger than 40 students. This is especially the case if the course is used for more than a single occasion.

Further work on Athena includes the integration of a fraud detection system (e.g. Moss [Moss]). Especially for large groups of students this will fight the temptation for students to commit fraud. In addition to fraud detection, further work is needed at easing deployment of Athena. Currently, there is no nice installation procedure for the different user applications. This is something that needs to be developed to make Athena easier deployable for others.

References

[Athena website] http://athena.wing.rug.nl:8080

[JUnit] http://www.junit.org

[Moss] http://www.cs.berkeley.edu/˜aiken/moss.html