Delft University of Technology

TI3800 - Bachelor Project

Final Report

2013-07-10

# WebLab project

*Authors:*
Marieke van der Tuin
Bastiaan Reijm
Tim de Jong
Jeff Smits

*Thesis Committee:*
Dr. Eelco Visser
Vlad Vergu MSc.
Dr. Andy Zaidman
Dr. ir. Felienne Hermans

Delft University of Technology,
Faculty of Electrical Engineering, Mathematics, and Computer Science

# Summary

WebLab is an online academic tool used to improve education by providing a framework for teachers to supply a higher quantity and quality of assignments to students. Currently this system is being used in a variety of courses including the Concepts of Programming Languages course taught to bachelor students at the Delft University of Technology. As a tool is used more and more functionality must be added in order to meet the ever increasing demands. The goal of this project is just that, expand on the current system to provide support for a new set of features. Specifically two new courses want to start using WebLab. MySQL support is added to support the database part of the Web & Database Technology course so students can execute queries and test their code against the correct queries without seeing those queries. Java support is added to provide extra practice material for the Object-oriented Programming in Java course; students who are new to programming have a chance to practice with the material at their own pace without having to install a myriad of software packages. Aside from these main features other features including group support and random assignment collections are also included in this project. Finally, as with any other software engineering project we include our requirements analysis, system analysis, project process, and take an in depth look at the testing of such a diverse and complex system.

# Preface

This report is the final report that concludes the project in which the WebLab/ LabBack system was extended. The project was executed as the final project of the Bachelor Computer Science at the Delft University of Technology in the Netherlands (course TI3800 - Bachelor Project). The assignment itself was also issued by the university. The course load per student is 15 ECTS, 420 hours.

The first meeting with our company supervisors, Dr. Eelco Visser and Vlad Vergu MSc., was on the 19th of March, 2013. The project was officially started on the 22nd of March and was concluded with the final presentation on the 10th of July. All research and development took place at the university, on the floor of our supervisors.

We would like the thank the following people by name:

- Eelco Visser, for the assignment and for the support and advice.
- Vlad Vergu, for the support, encouragements and the stern 'you *will* learn how to use this' approach; it really made a difference.
- Andy Zaidman, for the supervision of the project and the great support, advice and tips.
- Danny Groenewegen, for patiently helping out with all those new tools and languages.
- Jan Hidders and Bas van Sambeek, for the information on the course Web & Database Technology and for giving access to the course assignments.
- Felienne Hermans, for getting up early in the morning just to attend our presentation.

# Contents

**Summary** i

**Preface** ii

**Contents** iii

**1 Introduction** 1

**2 Assignment** 2

**3 Current System** 3
3.1 WebLab . . . 3
3.2 LabBack . . . 4
3.2.1 Communication . . . 4
3.2.2 Implementation . . . 6

**4 Requirements** 7
4.1 Requirement gathering . . . 7
4.1.1 Interview with Dr. Andy Zaidman . . . 7
4.1.2 Interview with Dr. Jan Hidders . . . 8
4.1.3 Input from our supervisors . . . 9
4.2 MySQL . . . 9
4.2.1 MySQL assignment . . . 9
4.2.2 Testing . . . 9
4.2.3 Databases . . . 10
4.2.4 Safety constraints . . . 10
4.3 Java . . . 10
4.4 Groups . . . 10
4.5 Project Assignment . . . 11
4.6 Random Assignment Collection . . . 11
4.7 Other . . . 11

**5 Implementation** **12**
5.1 MySQL . . . 12
5.1.1 Options . . . 12
5.1.2 Choices . . . 15
5.1.3 Technical details . . . 17
5.2 Java . . . 22
5.2.1 Options . . . 22
5.2.2 Choices . . . 24
5.2.3 Technical Details . . . 25
5.3 Group support . . . 26
5.3.1 Options . . . 27
5.3.2 Choices . . . 28
5.3.3 Technical details . . . 29
5.4 Random Assignment . . . 33
5.4.1 Options . . . 34
5.4.2 Choices . . . 35
5.4.3 Technical details . . . 36
5.5 Unimplemented Features . . . 38
5.5.1 Multiple Tries . . . 39
5.5.2 TA Tests . . . 39
5.5.3 Project Assignment . . . 40

**6 Testing** **44**
6.1 Current System Tests . . . 44
6.2 Overview of Testing . . . 45
6.3 WebDSL Testing . . . 46
6.4 Testing Labback with XML files . . . 46
6.4.1 Structure of the tests . . . 47
6.4.2 Usage of the tests . . . 47
6.5 Scala Testing . . . 48
6.6 Java Security Testing . . . 48
6.7 Selenium Testing . . . 49
6.8 SDF Testing . . . 50

**7 Process** **51**
7.1 Planned process . . . 51
7.1.1 Code evaluation . . . 52
7.2 Actual process . . . 52
7.2.1 Code evaluation and reaction . . . 53
7.3 Personal reflections . . . 55
7.3.1 Reflection Marieke . . . 55
7.3.2 Reflection Bastiaan . . . 55
7.3.3 Reflection Tim . . . 56
7.3.4 Reflection Jeff . . . 57

**8 Conclusion** **58**

**Bibliography** **59**

**A Project Proposal** **59**

**B SIG Feedback** **61**

**C Deployment Strategy** **63**

**D Java Security Tests** **70**

**E WebDSL Testing** **77**

# 1 Introduction

In the school year 2010–2011 all first year students of the Bachelor Computer Science at the TU Delft followed the course 'Concepts of Programming Languages'. This course taught students the general concepts of programming languages, using three different programming languages: Scala, C and JavaScript. The course included lab assignments. All students had to install the different compilers and editors needed to perform the assignments. This was not always easy due to different operating systems and configurations. The assignments had to be handed in on paper. The teaching assistants had a lot of work on their hands: how to check if the code is correct by only looking at the printed code? How can you be sure everything works correctly?

Therefore Eelco Visser —the course lecturer— created a tool named WebLab, which enabled students to perform their practical assignments online and also let the TAs grade the students' work online. WebLab could easily run automated tests to check the students' code. The test results formed a part of the grading. The rest of the grading could be done by the TAs by looking at the code of the students and giving points according to predefined checklists. To conclude: WebLab made a big difference in effort for both students and teachers to set up, and for TAs to grade lab assignments.

This tool would be perfect to use in other programming courses. Our goal is to make WebLab available for other first year courses to improve the learning process of the students and give the teachers and TAs access to its ease of use. To reach this goal we will make WebLab available to two main programming courses in the first year of the Bachelor: Object-oriented Programming (OOP) and Web & Database Technology (WDT). After implementing the features needed for these two, WebLab can also be used by two other programming courses of the first year because they both use Java, the same language used by OOP. So when our goal is reached, one third of the first year courses use WebLab.

In this final report we will first describe the assignment for the Bachelor project. Next we will describe the current WebLab system and the separate grading system it uses (LabBack). After that we will discuss the requirements of the assignment. In chapter 5 we will describe the implementation, followed by the description of the testing of the system in the next chapter. Chapter 7 will describe the process during the project, including personal reflections of the team members.

# 2 Assignment

WebLab gives students access to a cloud programming interface in their web browser, allowing them to make tutorials, assignments and even take exams without having to install or configure any software on their own computers. WebLab itself does not compile the student code or even run the tests that students can make. Instead it interfaces with the back end system called LabBack. LabBack supports automatic assessment of programs, which provides possibilities for larger amounts of exercises without significantly increasing grading workload.

WebLab is currently used in the Bachelor of Computer Science course 'Concepts of Programming Languages'. This programming course is currently given in the first year and uses WebLab for lab assignments and exams. The programming languages Scala, C, and JavaScript are currently supported.

For this Bachelor project, WebLab has to be made available for two other first year programming courses: Web & Database Technology and Object Oriented Programming (OOP). To realise this, several parts have to be completed:

**MySQL.** Compilation, tests, syntax highlighting in the editor

**Java.** Compilation, tests, syntax highlighting in the editor

**Groups.** Performing an assignment with a group of students

**Random Assignments.** Each student gets a random set of assignments

**Multi-File.** The possibility to have multiple code files per assignment

The original assignment document that we submitted as part of the Bachelor Project can be found in appendix A. Further specification of the features will be done by interviewing the teachers of the two courses. Also the input of the creator of WebLab (Eelco Visser) and the creator of LabBack (Vlad Vergu) will be used. Finally both Eelco Visser and Vlad Vergu will be the product supervisors.

# 3 Current System

The current system has two parts, the web front-end (WebLab) and the server-side back-end (LabBack). The front-end provides the complete online environment including persistent content. The back-end provides a safe, contained environment to execute foreign code written by the students. The two independent systems communicate via the file system by reading and writing XML files. A schematic overview of WebLab, LabBack and the user is shown in Figure 3.1.

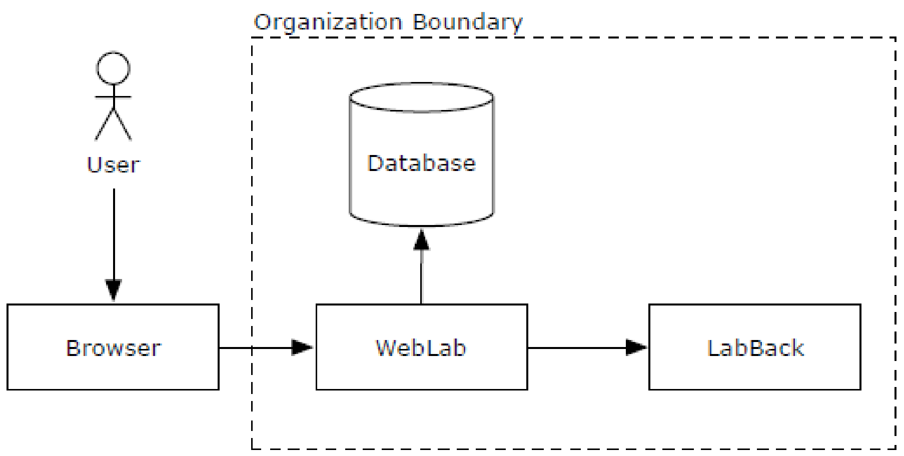


Figure 3.1: Hierarchy WebLab and LabBack, source: [? ]

## 3.1 WebLab

WebLab provides the front-end to the users. The code is written in the WebDSL programming language. WebDSL compiles to Java and is compressed into a WAR file (Web application ARchive). It uses a MySQL database for automatic persistent storage of content. The WAR file itself is deployed on a Tomcat server.

WebLab consists of three main parts: persons, courses and assignments, as can be seen in Figure 3.2. One course can have several course editions. Each

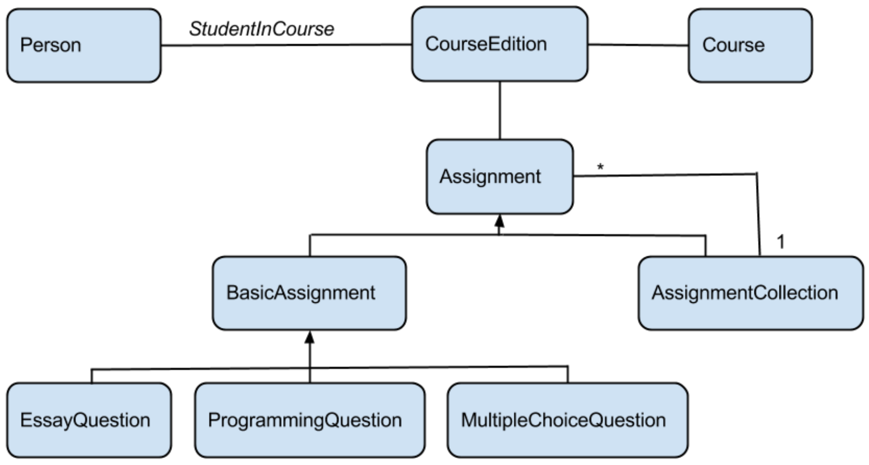


Figure 3.2: Class overview of WebLab

course edition has one assignment (an `AssignmentCollection`). Within the `AssignmentCollection`, multiple assignments can be included. There are two types of assignments: an `AssignmentCollection` and a `BasicAssignment`. A `BasicAssignment` can be an essay question, a multiple-choice question or a programming question.

If a student enrols to a certain `CourseEdition`, a special profile called `StudentInCourse` is created, where all data for a user especially about the `CourseEdition` is saved.

## 3.2 LabBack

LabBack is a system that can execute foreign code in a safe, contained environment. This makes it capable of executing student code without being susceptible to fraud. Apart from executing normal code, it can run tests. This capability is used in WebLab for automated grading.
LabBack 2.0 is designed to be scalable to multiple instances on multiple servers [**?** ]. But currently version 1 is in use.

### 3.2.1 Communication

LabBack v1 communicates with WebLab using the file system. LabBack monitors a shared directory for new XML files that WebLab writes. These describe what kind of job is needed from LabBack and where WebLab has written the (source code) files which may be necessary for the job.

Once a new job file is written by WebLab, LabBack reads the job, executes it, and meanwhile updates how the job fares by changing the job file. When LabBack is done, WebLab is in charge of removing the files again.

In Figure 3.3 a process diagram is given [**?** ].

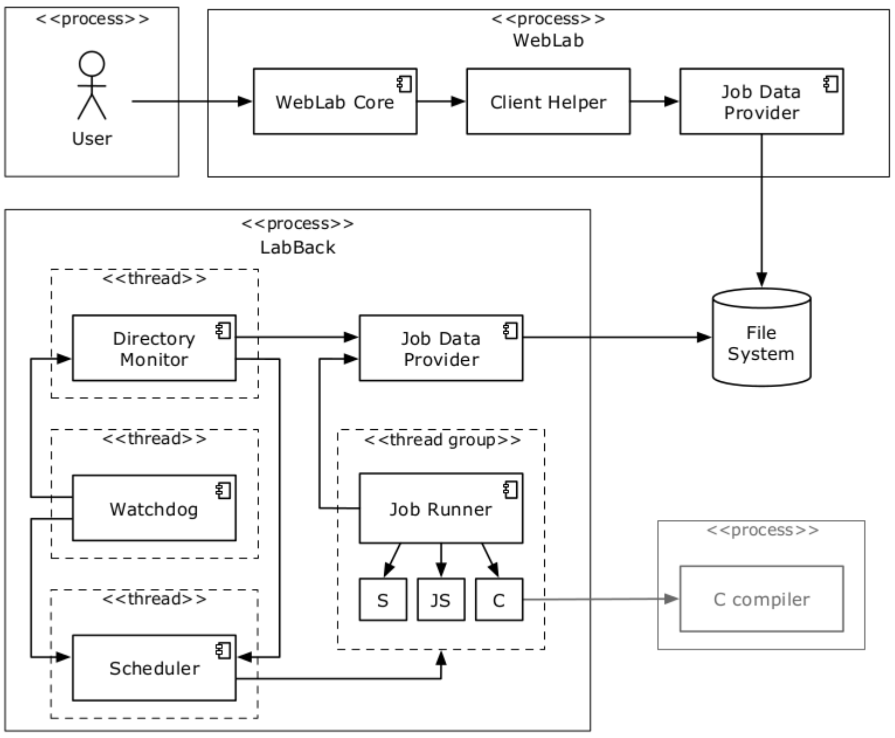


Figure 3.3: LabBack 1.0 processes and threads, source: [**?** ]

### 3.2.2 Implementation

LabBack is mainly implemented in the programming language Scala. It uses different libraries and programs to compile, execute and test Scala, JavaScript and C programs:

**Scala** Compiled with the Scala compiler, which is callable from Scala.

**JavaScript** Compiled and interpreted by the library Mozilla Rhino[1].

**C** Compiled to LLVM bytecode using the program Clang[2]. Then compiled to JavaScript using the program Emscripten[3], and the JavaScript is executed using Mozilla Rhino again.

[1] `https://developer.mozilla.org/en-US/docs/Rhino`
[2] `http://clang.llvm.org/`
[3] `https://github.com/kripken/emscripten/wiki`

# 4 Requirements

In order to gather the requirements for the project, we have interviewed the teachers of the two courses that we would support in Weblab. The requirement gathering resulted in two kinds of requirements: those that were really required and those that would be nice to have, if we would have time left after the required ones. We will use the MoSCoW method[1] to describe this distinction, where we have 'Must have' and 'Could have' requirements.
After the Requirement gathering follow the requirements for MySQL, Java, Groups, Project Assignment and Random Assignment Collection, each in their own section. The final section of this chapter lists the Other requirements, which fit in none of the above categories.

## 4.1 Requirement gathering

Because the goal of the project was to add support for two additional courses to WebLab, we decided to start the gathering of requirements by interviewing the teachers of those courses. This section starts with an Interview with Dr. Andy Zaidman. It then continues with an Interview with Dr. Jan Hidders. Finally, it is concluded with Input from our supervisors.

### 4.1.1 Interview with Dr. Andy Zaidman

In order to know what supporting Object-oriented Programming in Java would entail we interviewed Dr. Andy Zaidman, the course lecturer. Although, he had never used the system before he did have some prior knowledge about what WebLab can and cannot do. Currently WebLab is not a full fledged IDE and thus for the graded assignments Dr. Zaidman would still prefer to use Eclipse or IntelliJ because it prepares students better to be able to use these work environments.

However, there is room for WebLab in the course and it can be used to save a lot of time. Every week a 2 hour instruction session is given for students who want extra practice with the basic material. These instruction sessions are led

[1]Prioritisation technique described in [? ]

by Dr. Zaidman, where he hands out an assignment on paper and the students attempt to work out the problem. Usually this is writing a method or function that accomplishes something like sorting an array or computing the distance between two points. After 20 minutes or so Dr. Zaidman writes a solution on the board and the students answer the next question.

This can be automated using WebLab. Students get a basic assignment, write Java code, compile it, test it, and know immediately if their solution is correct. Students can login into and use WebLab at any time of day and as Dr. Zaidman noted, this will save everyone time. With that time he can focus on improving other parts of the course.

### 4.1.2 Interview with Dr. Jan Hidders

To gather the requirements for MySQL support in WebLab, we had an interview with Dr. A.J.H. Hidders, the teacher of Web- & Database Technology. We asked him about the current form of the MySQL lab in his course and about what features he needs added to WebLab in order to start using it for the course.

**The MySQL lab** Currently, the lab is done by pairs of students. Each pair has to do two sets of assignments, each of which consist of a random selection from an assignment database. When a TA checks a set of assignments, he marks each assignment as either correct or incorrect. After the set has been checked, the students get one chance to fix the incorrect assignments. After that they get the second (harder) set of assignments, for which they also get two tries. However, if the students have too many incorrect assignments left after their second chance on a set, they fail the whole lab.

The assignments themselves consist of writing read-only queries for a predefined database, though these assignments also allow —and sometimes encourage— the use of temporary views.

**Features needed from WebLab** Dr. Hidders would like to be able to conduct the MySQL lab in WebLab in the same way as the present. This would require a lot of new (preferably generalised) WebLab features: student groups, randomized assignment sets, allowing multiple tries for assignments. Of course, this also requires that the MySQL language is supported and that the system is connected to the database that is used in the assignments. However, Dr. Hidders has agreed that these are a lot of features to implement and that it would be acceptable to prioritize; possibly leaving less important features out entirely.

Dr. Hidders mentioned he would like to extend the current assignments to include the `INSERT`, `DELETE` and `UPDATE` MySQL commands. He would also like to try unit testing, including giving students the possibility to write their own tests. The testing should at least support checking if the students solution returns the same result as the teachers solution. An important feature for the testing would be that TAs can add more tests, to be used next year and grow the amount of automated tests. Use of `VIEW`s is a 'must-have' because not everything in MySQL can be expressed without them.

### 4.1.3 Input from our supervisors

In addition to the interviews with the course teachers, our supervisors have also had their say in the requirements. They have a certain vision for the future of WebLab as a highly scalable platform that can automate and facilitate the making and grading of assignments, allowing teachers to increase the workload of assignments and thus provide extra practice to improving the quality of education. From this standpoint, they have added their ideas and priorities to those of the teachers.

For both Java and MySQL, they gave priority to a safe system and automated tests for the student code. We already knew we would need multi-file support for the bigger java assignments (because each Java class needs its own file). They asked us to expand on this idea, to design a project structure for WebLab assignments similar to projects in an IDE like Eclipse or Netbeans. The design of such a structure has priority over the actual implementation, because this would be a difficult project on its own. As for the plethora of additional features for the MySQL lab, our supervisors gave priority to the random assignment sets and student groups.

As an extra optional feature of their own that they added, they asked us to make the assignment editor screen in WebLab resizeable.

## 4.2 MySQL

The following is a list of required features for adding MySQL support to Weblab. These requirements follow from the interview with Dr. Hidders and from our supervisors. Each requirement is labeled Must or Could, according to its priority using the MoSCoW method.

### 4.2.1 MySQL assignment

1. **Must.** MySQL code is highlighted in the assignment editor.
2. **Must.** Students can write read-only query assignments.
3. **Must.** Views are allowed in assignments.
4. **Must.** Write commands are allowed (e.g. `insert`, `update`, `delete`).
5. **Could.** `create/drop table` are allowed.
6. **Could.** Transaction control commands are allowed.

### 4.2.2 Testing

7. **Must.** MySQL code can be tested.
   (a) Unit testing framework within MySQL itself.
   (b) Some other language is used to set up unit testing.
8. **Must.** Students can add their own unit tests.
9. **Could.** The default specification test is a check if the student query gives the same result as the teacher's answer query on the database.

### 4.2.3 Databases

10. **Must.** Allow teachers to set up a database for the assignments from within WebLab.
11. **Must.** Allow teachers (or an administrator) to remove databases from within WebLab.
12. **Could.** More than one database per course, the database to be used can be set per assignment.

### 4.2.4 Safety constraints

13. **Must.** A student cannot see any result of a query executed by another student. This includes the result of `INSERT`, `UPDATE`, `DELETE`, `CREATE/DROP VIEW` and `CREATE/DROP TABLE` statements.
14. **Must.** A student query can not permanently harm or change the database that other students still have to use.
15. **Must.** It must be absolutely impossible that a student query can read or alter the Weblab database.

## 4.3 Java

The following is a list of features for adding Java support to Weblab. These requirements follow from the interview with Dr. Zaidman and from our supervisors.

16. **Must.** A student cannot see any result of a query executed by another student
17. **Must.** The system has support for the Java programming language
18. **Must.** The system has support for a single class and multiple methods
19. **Must.** The system has support for the JUnit Testing Framework
20. **Must.** The system runs securely
21. **Must.** The course manager can make, edit, and delete Java assignments
22. **Must.** The system has support for multiple Java classes in one assignment
23. **Could.** The system has support for multiple files such that students can do assignments with multiple classes

## 4.4 Groups

The following is a list of required features for adding Group support to Weblab. These requirements follow from the interview with Dr. Hidders and from our supervisors.

24. **Must.** Groups can have a flexible size

25. **Must.** Students can create their own groups or the teacher can create them
26. **Must.** Both individual and group assignments should be supported for a course edition
27. **Could.** Dealing with concurrent editing

## 4.5 Project Assignment

The following is a list of features for adding Project Assignment support to Weblab. These requirements follow from the interview with Dr. Zaidman and from our supervisors.

28. **Must.** Has support for files and directories
29. **Must.** Users with the right access can lock files and directories
30. **Must.** Users with the right access can change the visibility of a file or directory
31. **Must.** Users can files and directories
32. **Must.** Users can remove files and directories if the parent directory is not locked
33. **Could.** The system tries to determine an automatic entry point
34. **Could.** Graders can see multiple solutions in the same view

## 4.6 Random Assignment Collection

The following is a list of required features for adding support for random assignments to Weblab. These requirements follow from the interview with Dr. Hidders and from our supervisors.

35. **Must.** Students can receive a personal random selection of a certain size out of a complete set of assignments.
36. **Must.** The random assignment should also work with groups
37. **Could.** More than one random assignment collection per course edition

## 4.7 Other

38. **Could.** Teacher's Assistants can add their own tests.
39. **Could.** The tests added by each TA can be automatically merged and added to the specification tests.
40. **Could.** The screen elements, most importantly the editor, resize automatically to fit the screen size.
41. **Could.** The system allows multiple tries for certain assignments

# 5 Implementation

The implementation of the specified requirements consists of different parts. First of all, the different options have to be researched. Second, there have to be made certain choices or design decisions. After that the features could be actually implemented.

In this chapter the three different steps of this implementation process can be found per feature: options, choices and the technical details, including screenshots and descriptions of the implemented features. Finally to run the full system see the Deployment Strategy in appendix C.

## 5.1 MySQL

MySQL is a powerful query language that is used to search, modify and create databases. A MySQL query is sent to a MySQL server, which can contain multiple databases. Most queries require that a specific database is selected to run the query on. Because MySQL is such a powerful language —an entire database can be deleted with one short statement— the challenge was to make it safe to run on the WebLab/Labback system. To give an example, without any safety restrictions it would be easy for a student to change all of his grades into a 10. This section explains the decisions we have made and the design of our implementation for MySQL support in WebLab.
First, the Options are listed for each decision. Next, the Choices that we made are explained. Finally, the Technical details of the implementation are shown.

### 5.1.1 Options

To add MySQL support to WebLab, some decisions had to be made as to how the requirements would be realised. The subsections that follow describe these decisions in detail.

#### Permanent changes or reverting changes

Since one of the requirement is supporting changes in the database, we have to do this in a secure way. Keeping the safety restrictions in mind, permanent changes are only possible if each student has in some way a separate database. But giving every student a separate database is bad for scalability. Although solutions like an overlay file system with UnionFS are possible to keep disc space usage to a minimum, it increases complexity and lowers operation speed.

Another way is to revert every change a student makes to the database. The 'Data Manipulation' part of SQL (e.g. `SELECT`, `INSERT`, `UPDATE`, `DELETE`) can be used within the standard SQL transaction system[1]. Separation of transactions would be standard behaviour and reverting is as simple as a rollback.
Using other parts of SQL, like the Data Definition Statements (e.g. `CREATE / DROP VIEW`, `CREATE / DROP TABLE`) or transaction commands would be harder to support because they cannot be used within transactions. And database engines usually use a locking system to separate concurrent transactions, which in the best case results in small amounts of latency but in the worst case can result in deadlock.

### What types of queries to support

Some types of queries are easier to execute in a secure way than other types of queries. Thus, a decision has te be made regarding the types of queries that we will allow students to write and execute on the system.

Read-only queries are the bare basics and a common use of a database. Implementation of this feature only would be straight-forward.
Supporting write commands like `INSERT`, `UPDATE` and `DELETE` is a bit harder, because they change the database. This feature may therefore violate the safety constraints, unless it is carefully implemented so that no student can see another's changes (related: Permanent changes or reverting changes).
Adding support for `CREATE/DROP VIEW` can introduce another challenge, because statements cannot be used inside transactions. If the reverting changes feature is used, then support for views will have to be specially added outside transactions. Safety concerns will have to be reconsidered. A possible solution security-wise would be to randomize the views names in the execution. But this would require some code analysis and transformation. Statements to create and drop tables fall in the same category.
The hardest type of statements to allow are the transaction control statements. Allowing those to be written by the student would require different databases per LabBack thread because students can possibly create deadlock situations which would keep the database server occupied. Support for this feature would force the choice to not use reverting changes using transactions.

### Checking the MySQL code

The MySQL code the student sends cannot be trusted and has to be checked on unsupported[2] parts of MySQL before being sent on to the database server. The JDBC implementation of MySQL (Connector/J) also requires one statement at a time.

So the code has to be checked, and then sent per statement. Regular expressions are an option, but it's either a lot of work, or fragile, or both. The official MySQL parser is open-source, but the definition is for yacc, which doesn't

[1]as long as the database engine supports it!
[2]The statements that *we* don't support in LabBack.

compile to Java. There are other open-source SQL parsers (e.g. Akibans[3]), but not for the MySQL dialect. Creating a new parser is another possibility, with Spoofax for example.

### Unit Testing

The ability to automatically test student code is an important feature for our company supervisors, because automation allows more assignments to be issued to more students, which is one of the selling points of WebLab.
Unit testing frameworks within SQL are available, but there are some problems with those. utMySQL [4] is licensed under GPL (a copyleft license), but WebLab and LabBack are not. MyTAP [5] has a less restrictive license but is unfinished and hasn't been worked on in years.

Normal unit testing consists of assertions on the existence of certain structures and equality of output of operations with an expected output. Using a unit testing framework of a normal programming language is possible.

It also looks feasible to use some "magic" (parsed) comments in MySQL and a result-set equality checker to check the output of the students solution against the output of the teachers solution.

### Database setup from within WebLab

Because MySQL code has to be executed on a specific database. The teacher of a course should have some way to specify what database he wants to use for the assignments. We have to choose how this will be possible.
The simplest way would be a text input which takes a `mysqldump` file and perhaps a database name. The possible problem with this approach is not being able to enforce a particular database engine or table names. This can interfere with the way the security can be implemented. For example, not all database engines of MySQL support transactions.

### More than one database per course

An additional choice regarding the configuration of databases, is whether we should support adding multiple databases per course. This would make the general design a lot more flexible, but would require changes to the data model of WebLab assignments. Scalability is of course dependent on the size of the databases.

### Managing databases from WebLab

Somewhere in the interface of WebLab, additional options need to be added to add, delete and manage the databases that may be needed for MySQL assignments.
One possibility is to add all of these database options to the page for editing a ProgrammingQuestion. The advantage of this approach is that the database options can be hidden if the language of the ProgrammingQuestion is not set to SQL.

[3]http://akiban.github.io/sql-parser/
[4]http://utmysql.sourceforge.net/
[5]http://theory.github.io/mytap/

Another possibility is to add the link to a database management page to the list of things that can be viewed/edited in a CourseEdition. This would make sense if More than one database per course is implemented in such a way that databases are bound to a CourseEdition. More global management options can be added in this way. A disadvantage is that the link to the page would also be visible for courses that have nothing to do with MySQL assignments or databases.

### 5.1.2 Choices

For every option listed in Options we have made a choice based on what would be reasonable, scalable and safe. The choices are explained below.

#### Permanent changes or reverting changes

Students should be able to make small changes to the database (e.g. with `INSERT`) without any visible effect for the other students. Two options could be identified to make this work: either give each student a separate copy of the database, or have one database in which student changes are done in separate transactions, which can be rolled back to prevent permanent changes.
We have chosen to use one database, relying on the transaction system to keep changes invisible and temporary. The strongest argument against using multiple databases was the scalability of the system. Our supervisors have expressed their need to keep WebLab highly scalable and mentioned that Labback should be able to be deployed distributed across several servers to accommodate a growing number of students. If each server runs more or less independently, each server would need a copy of the database for each student, destroying all hopes of scalability. We would not argue that it is impossible to achieve personal databases that can be scaled - it could perhaps be realised by using separate database servers, but only that it would be difficult.
Using transactions seems to be a more sensible choice. If there is already a transaction system that can hide changes from other users and that gives the option to roll back instead of committing the changes to the real database, which is exactly what we want, why not use it? The downside of this choice is that it excludes the possibility of supporting `CREATE / DROP / ALTER TABLE` queries and transaction control statements. This will not be a problem for the Web & Database Technology course, however. Even using statements such as `INSERT` is already a step up from using only `SELECT` queries, as is currently the case in the lab.

#### What types of queries to support

The possibilities of what queries we should support range all the way from safe read-only queries to hard-to-control transaction statements. Because we have chosen to use the MySQL transaction system to revert the impact of write queries such as `INSERT`, we can at least support those statements in addition to the necessary read-only queries. Anything harder than that (creating views and tables, and transaction statements) does not fit in a transaction, which will make it hard to support in a safe way.
In the interview with Dr. Hidders, we learned that using views is currently an important part of the MySQL lab, so we decided that we had to support those

as well, even though it will make matters more difficult. We came up with the idea to somehow filter the student code, extracting all `CREATE VIEW` and `DROP VIEW` statements. The views can then be created before the transaction, next the remaining student code can be run within a transaction which is rolled back at the end, after which the views can be dropped again. This leaves the issue that the views can be used by other students in the time frame between the creation and the dropping of the views. We decided to solve this by adding a generated unique id to the view name. In this way, using the view of another student is not impossible, but guessing the view name is comparable to guessing someone's user name and password, so the risk would be sufficiently small.

### Checking the MySQL code

Because regular expressions are rather fragile unless used as a full parser, we decided against that option.
The official MySQL parser definition is for yacc, which doesn't compile to Java. (The biggest problem being the C/C++ code snippets called ACTIONS that are embedded in the yacc definition).
We briefly tried the Akiban SQL parser, but it was quite incompatible with the MySQL dialect.

Then we heard from one of our supervisors that a partial SQL grammar was already available in SDF[**?** ], which is used in Spoofax. And there is also a case-study in SugarJ[**?** ], which transforms a BNF[6] syntax definition to SDF[7].

After looking for a BNF definition of MySQL, which we could not find, we choose to extend the existing SQL grammar files ourselves. Although we could find a BNF for SQL, the generated SDF would not be nice to edit, and the existing SQL grammar already had some non-standard statements implemented that are also in the MySQL dialect.

### Unit Testing

After looking into different options, we decided against an existing unit testing framework in MySQL. Mostly because of copy-left licenses (utMySQL) or because it was an incomplete, seemingly abandoned project (MyTAP).

A different language with unit testing framework would mean extra complexity for the back-end and extra complexity for the user.
Given that parsed comments were already used to implement unit testing for Scala, we figured it would be a consistent and easy way to do it.

### Database setup from within WebLab

The choice between simple and secure was heavily influenced by time constraints this time. The teacher will have to be trusted to upload the right MySQL code, and the file that is uploaded is executed on the MySQL server without filtering.

[6]Backus-Naur Form [**?** ]

[7]`https://github.com/seba--/sugarj/blob/master/case-studies/java-pet-store/src/java/sugar/`

#### More than one database per course

We have chosen to support the option of having multiple databases per course, because this keeps the design flexible for future courses. We do not lose much by this choice, as long as the size of the databases is kept under control. There is no real difference in scalability between using one big database or several smaller ones.

#### Managing databases from WebLab

The possible places to add database management functionality are the edit page of a ProgrammingQuestion or in the menu of a CourseEdition. We have chosen to add most of the management functionality to the 'Manage databases' option of the CourseEdition menu. Because there is more space here, we can offer multiple management options that will all fit. There is still some additional functionality on the edit page of the ProgrammingQuestion, mostly to change the database associated with that ProgrammingQuestion. Even though database management is not used for all courses, it makes more sense that databases that are kept on a CourEdition-level can be managed on a CourseEdition-level. This also allows nice options such as setting a default database for that CourseEdition.

### 5.1.3 Technical details

The technical details below are structure by system. That is, we first describe details on the implementation of features in WebLab, then the details in LabBack and then describe the parser, which is a Spoofax project.

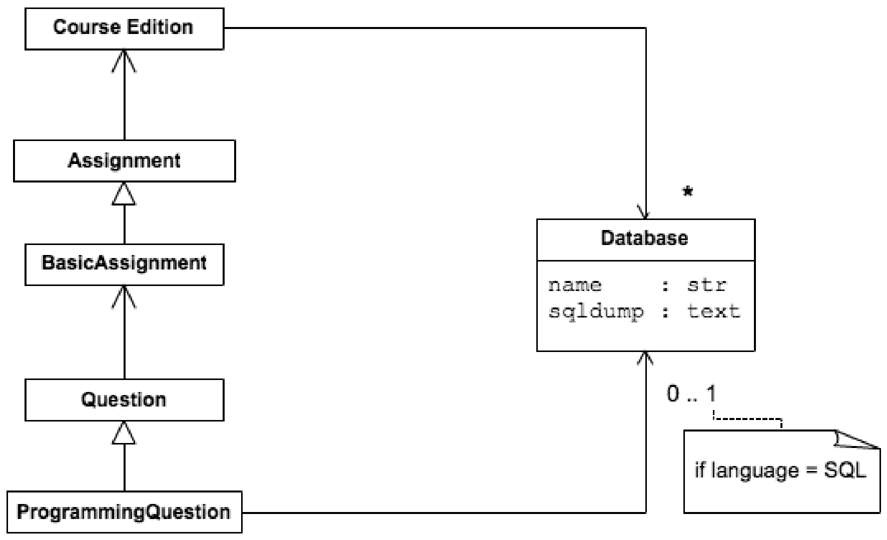
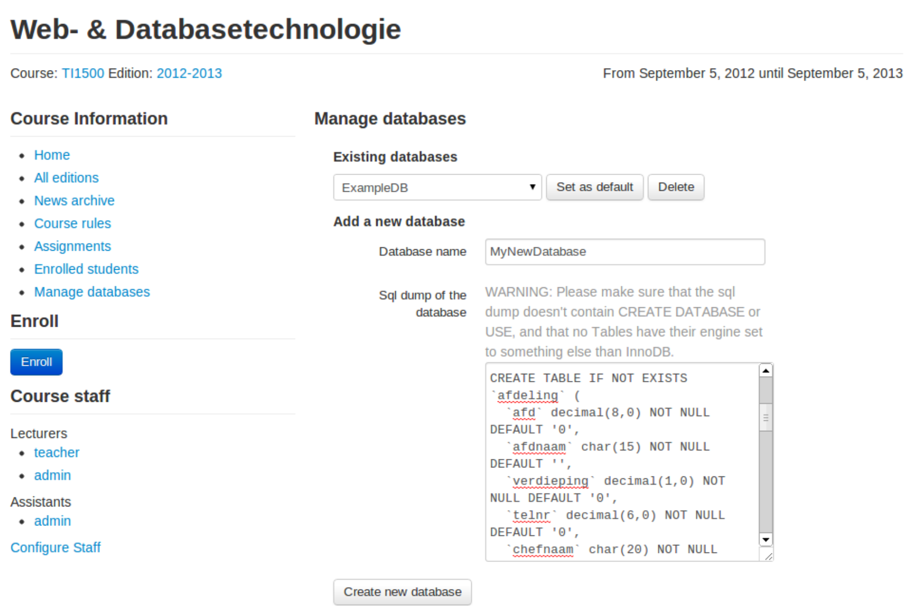


Figure 5.1: Class diagram of the Database entity in WebLab.

### WebLab

The interesting WebLab changes for MySQL have to do with the management of databases. To this purpose we have added a Database entity to WebLab, which contains the name of the database and the SQL code required to (re-)create it.

Figure 5.1 shows how the Database entity is connected to the rest of the system. Each ProgrammingQuestion that has its language set to SQL can have its own database, if required. Alternatively, it is also possible to have multiple ProgrammingQuestions make use of the same database. The databases that can be used for a ProgrammingQuestion are limited to those of the CourseEdition that it belongs to. The databases associated with a CourseEdition can be managed in the newly added database management page, which can be seen in Figure 5.2


Web- & Databasetechnologie

Course: TI1500 Edition: 2012-2013

From September 5, 2012 until September 5, 2013

Course Information

- Home
- All editions
- News archive
- Course rules
- Assignments
- Enrolled students
- Manage databases

Enroll

Enroll

Course staff

Lecturers
- teacher
- admin

Assistants
- admin

Configure Staff

Manage databases

Existing databases

ExampleDB | Set as default | Delete

Add a new database

Database name: MyNewDatabase

Sql dump of the database: WARNING: Please make sure that the sql dump doesn't contain CREATE DATABASE or USE, and that no Tables have their engine set to something else than InnoDB.

```
CREATE TABLE IF NOT EXISTS
`afdeling` (
  `afd` decimal(8,0) NOT NULL
DEFAULT '0',
  `afdnaam` char(15) NOT NULL
DEFAULT '',
  `verdieping` decimal(1,0) NOT
NULL DEFAULT '0',
  `telnr` decimal(6,0) NOT NULL
DEFAULT '0',
  `chefnaam` char(20) NOT NULL
```

Create new database

Figure 5.2: Screenshot of the database management page of a CourseEdition.

The database management page is normally reached from the CourseEdition menu show on the left side of Figure 5.2. The second way to get there is through a link in the edit page of a MySQL ProgrammingQuestion. The following things can be done on the database management page:

- Set an existing database as default
- Delete an existing database
- Add a new database

When a database is set as the default for a CourseEdition, all new MySQL assignments created for that CourseEdition will have that database selected at first. This can be especially useful for a course that only uses one database, like the Web and Database Technology course.

When deleting a database, the system first checks for MySQL ProgrammingQuestions that use that database. A warning is shown if such dependent ProgrammingQuestions exist.
To add a database to the CourseEdition, a name and an sql dump of the database are needed. This information is sufficient to build a database from scratch. To prevent nameclashes in the MySQL server, the unique `id` of the CourseEdition is added to the name of the database before it is sent to the server. It is impossible to add two databases with the same name to one CourseEdition, making this method safe.

### LabBack

As explained in Current System, the job of LabBack is to read each tasks from an XML file, execute the task, and write the result back in the file. When a new task file is found, LabBack starts a separate `ExecutionThread` of the correct subtype which is charged with handling everything required for that task. Thus, to make LabBack work with MySQL tasks we needed to make a subclass of `ExecutionThread` that could handle a MySQL task from beginning to end. We made this subclass and named it `SQLExecutionThread`.
The `SQLExecutionThread` uses several specialized classes to do its work. The relationship between these classes is shown in Figure 5.3.

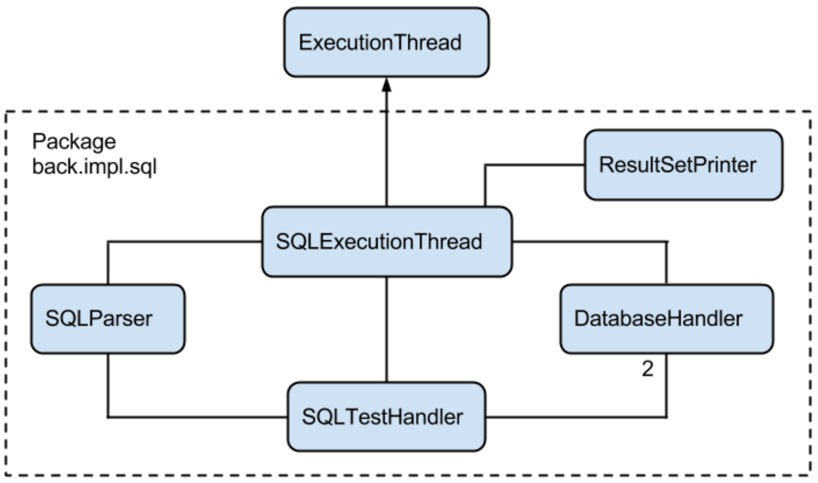


Figure 5.3: Class diagram depicting the MySQL classes in LabBack.

Figure 5.3 shows all the classes that have been added to handle MySQL tasks. One `SQLExecutionThread` is created whenever a MySQL task needs to be handled. This thread first reads the files containing student code. It then checks the type of the MySQL task, which can either Compile or Run. Because MySQL code is not compiled, but interpreted, we chose to change the 'Compile' task behaviour to something more useful. The special Compile behaviour for a MySQL task is that the student query is parsed and run on the database, after which the potential result is returned to WebLab. The Run behaviour has been kept the same as for other tasks, which is to run the unit tests and report on the outcome. The execution of Compile and Run tasks is shown in Figure 5.4.

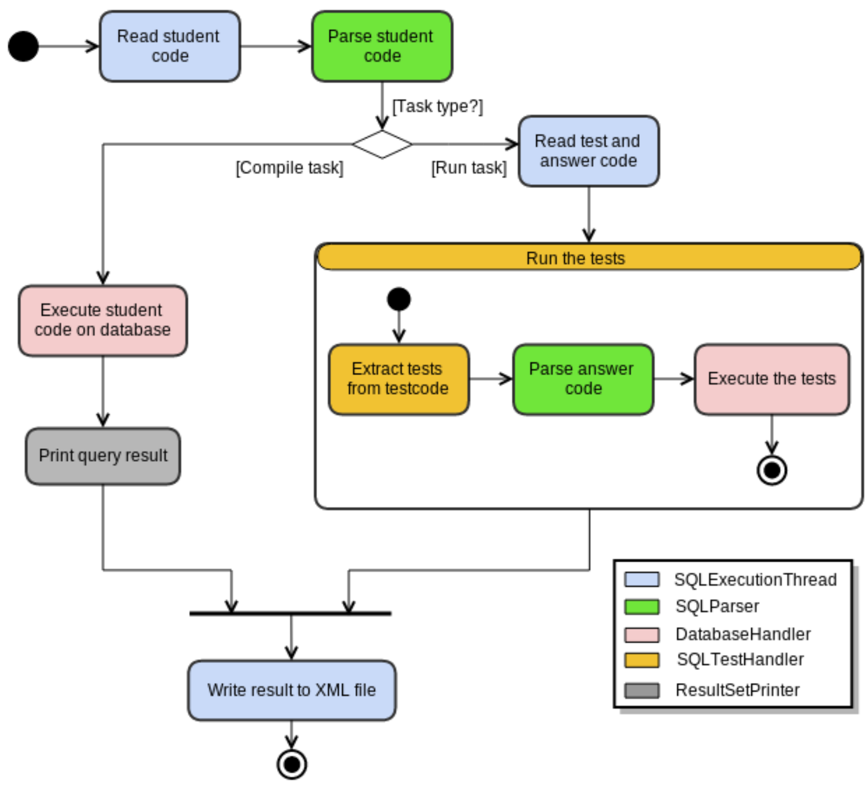


Figure 5.4: Activity diagram showing how a MySQL task is handled in LabBack.

To complete a Compile task, the `SQLExecutionThread` passes the solution code through the `SQLParser` class, that makes a call to the actual Parser. This results in MySQL code that should be safe to execute on the database. To execute this clean code, it is passed to the `DatabaseHandler`, which is the only class that makes a direct connection with the database. Finally, if the query resulted in a result table, it is printed by the `ResultSetPrinter`, using Unicode characters to print the table in text format.
To complete a Run task, the `SQLExecutionThread` reads the answer code and the tests from the test file and passes everything, along with the student code, to the `SQLTestHandler`. This class handles the rest of the testing process and returns the test scores together with a list of messages about what happened in the tests. The `SQLTestHandler` starts by reading the test file and breaking it up into the individual tests, expecting the test format described in Unit Testing. The tests are then executed one by one. To run a test, the test query is first parsed and run as a setup before the user query. Then the test code is run as a setup before the answer query. The results of these runs are compared using the static `resultSetEquals` method. The test succeeds if the results are equal and fails if they are not, or if the parsing or execution of the test resulted in an

error. If the test failed, a message is added to the test results explaining which test failed. When all tests have been run, the `SQLTestHandler` adds the results together and returns them to the `SQLExecutionThread`, who passes the result to WebLab.

### Unit Testing

To realise MySQL unit testing in WebLab, we decided to make our own test structure using MySQL itself along with 'magic comments', meaning comments that are parsed and thus actually influence the behaviour of the code.
Normally, a unit test runs a small piece of code and checks if the outcome is that which the user specified as correct. This is not practical for MySQL code, because outcomes are not simple values but tables with content, and predicting an exact outcome requires knowledge of the database. Our idea for testing MySQL code was to compare the result of a query that needs to be tested with the correct solution query. In WebLab, it is possible to add secret answer code to an assignment, so we figured it could be used for this purpose.
The most basic test would be to run the user code and the answer code on the database directly, and check that the results are equal. In our testing framework, this is the behavior of an empty test. If a test is not empty, but contains for example an `INSERT` statement, this test query is used as setup. To use it as a setup, it is executed before the user or answer code. For an example, see Listing 5.1.

Listing 5.1: Example of a MySQL test suite with magic comments.

```sql
-- tests: 2
-- test: empty test

-- test: cookieTest
INSERT INTO afdeling
VALUES
(0, 'testafd', 2, 1234, 'Cookie monster');
```

Student code

```sql
SELECT *
FROM afdeling
WHERE chefnaam <> 'Cookie Monster';
```

Answer code

```sql
SELECT *
FROM afdeling;
-- Student code should be like this
```

Listing 5.1 shows a small test suite containing two tests. The first one is the empty test. If we assume that there are no rows in table `afdeling` that have `chefnaam = 'Cookie monster'`, the result of the user and answer queries will be the same and the empty test will succeed. The second test contains two `INSERT` statements. With this statement as a setup before the user and answer queries are run, the results will be different this time. The test fails, showing the user that his query is not yet perfect.

### Parser

The MySQL parser was created using Spoofax. Starting with an old project[8], we adapted and extended the grammar to support a subset of the MySQL dialect.

[8]http://strategoxt.org/Stratego/SqlFront

| | |
|---|---|
| **Data Definition Statements** | `CREATE VIEW`<br>`DROP VIEW` |
| **Data Manipulation Statements** | `DELETE`<br>`INSERT`<br>`SELECT`<br>`UPDATE`<br>subqueries |
| **Other** | line comments starting with `--` |

Table 5.1: Supported MySQL statements

As a syntax reference we used the MySQL manual[9]. The subset of MySQL statements we support is in Table 5.1.

After supporting these statements we patched the generated pretty-printer with some rules that overwrite the generated ones. With this pretty-printer came the possibility to write a strategy (in Stratego) that outputs a list of strings, each with one MySQL statement.

Because we decided to revert student changes using transactions, and couldn't find a way to add `VIEW`s to transactions, we had to do some code rewriting aswell. Using more strategies, we added functionality that hoists `CREATE VIEW` statements and generates `DROP VIEW` statements to match the `CREATE VIEW` statements regardless of the `DROP VIEW` statements the student wrote.
Another security measure we took was to add `LIMIT 50` to every outermost `SELECT` statement. This rule was added to prevent excessive memory usage, without changing the MySQL statements semantically. We also took care to leave stricter `LIMIT` clauses written by students alone, but edit the less strict ones.

## 5.2 Java

Java support in the old version of the WebLab project seemed to be straightforward since the JVM is already installed and there was already support for Scala. However, there are some key differences which made the process more difficult. Firstly the different options are discussed, followed by the particular choice that was made from those options, and finally the technical details about how the choices were implemented.

### 5.2.1 Options

After doing research and analysis of the current system, there are four major choices that must be made in order to support Object-oriented Programming in Java. The first is the choice of compiler, the second is the choice of class loader, and the final choices are choosing what to do with the file names and how to support assignments with multiple classes.

[9]http://dev.mysql.com/doc/refman/5.5/en/sql-syntax.html

### Compiler

There are two main choices when it comes to Java compilers. The standard `javac` compiler by Oracle and the ECJ compiler that the Eclipse IDE uses internally. Both can be called programmatically since Java 1.5; the `javac` compiler comes standard with the latest Java versions and the ECJ compiler can be downloaded as a separate jar file. The main difference between these compilers though is that the ECJ compiler is an incremental compiler while the `javac` compiler is not. Incremental compiling does not necessarily have any advantage over full compilation. When the code segments are small an incremental compiler will not be able to build a subset of the code. A smaller subset would be impossible to create in situations where only one class and method is used. This is often the case for short assignments. However, to support larger projects and multiple file compilation on the server an incremental compiler is better suited.

### Testing

In order to facilitate automatic grading, code must be tested using a specific testing framework and students must also be able to test their own code. There are over 30 unit testing frameworks available for Java. Each one has its own purpose or speciality. One of the options that should be considered is the JUnit testing framework.

### Class Loader

In order to run user code on the JVM a class loader is used to read in the '.class' files. Both Java and Scala compile down to byte code which is stored in these files and thus can make use of the same class loader. However, the current system already has a class loader to support Scala. This version cannot be used to support Java because Java files are compiled to physical files while Scala compiles to virtual files. The two options here are to redesign the class loader to support both languages or to create a new class loader specifically for Java.

### File Names

Currently there are two files which the user can edit: the solution and the tests. These files are saved to the server as 'solution.ext' and 'tests.ext' where 'ext' is the file extension used for the particular language. The problem is that per Java convention class names should begin with an upper-case letter and the file name should must match the class name. Thus class names must be lower-case or the file names must be changed for Java specifically. Another way also exists, using the built in package-private classes. This would allow the files to stay lower-case while allowing upper-case class names.

### Multi-Class Support

Due to the limited number of files that the current system can support, assignments that use more than one class cannot simply be created. The options for this feature are to allow more files in the system and to create a project management sub-system. The other option is to use Java in a creative manner; multi-class assignments can be done if all the classes in one file are package-private and not public.

### 5.2.2 Choices

For each of the option sets discussed earlier, a choice had to be made as to which option should be actually be implemented.

#### Compiler

There were two options concerning which compiler to use. The standard `javac` compiler by Oracle or the the ECJ compiler by the Eclipse Foundation. The choice was made to use the ECJ compiler mainly because of its incremental nature. As mentioned, this has no immediate benefits but is very useful for when multi-file support is introduced into the system or when users start compiler larger sets of code. Another reason is that in practice the ECJ compiler is more memory efficient than the `javac`. The experience of people who have compiled several thousand classes simultaneously is that the `javac` requires much more memory than the ECJ compiler. Explicit benchmarking would have to be done to verify this phenomenon. But on a closed system with limited resources, such as a server, this phenomenon is rather important and should be factored in. For these reasons the choice was made to use the ECJ compiler to compile Java code.

#### Testing

From the over 30 unit testing frameworks for Java, the JUnit framework was chosen for the following reasons. Firstly, several other courses in the bachelor phase also use this framework to do unit testing. Secondly, this framework conforms to the xUnit standard which makes it easier to learn than many other frameworks. Finally, this framework has been in development for quite a long time and is rather stable. For these reasons JUnit is the framework that was chosen to facilitate testing of the user's Java code.

#### Class Loader

The two options concerning the class loaders for Scala and Java were to have separate class loaders or to have one class loader. The single class loader option was chosen even though it requires some changes in the system as a whole. The functionalities of a Scala class loader and a Java class loader were too similar and would just cause code duplication bugs later on. For example, they both use the same black list to block certain classes from being loaded for security reasons. Since both Scala and Java are run on the JVM, a change in one black list would necessarily mean a change in the other. For this reason a single class loader for both Scala and Java was chosen.

#### File Names

The last set of options was whether or not to change the names of the Java solution and test files such that the class names could conform to the Java coding conventions. Since this is an educational tool and students are supposed to learn better style, coding, and program designing. For this reason the choice was made to change the name of the generated Java test file, specifically from 'test' to 'UTest'. For other languages this change does not have to happen and thus it only affects the new Java support and not the old system. This relatively minor choice has a relatively large impact and we chose to not take the easy way out in order to give teachers and students the opportunity to adhere to the official Java standards.

#### Multi-Class Support

Initially the idea was to expand the current system with more files and project management. This has however become an unimplemented feature due to time constraints. Thus the choice was made to leave the system as-is and to allow multi-class support via Java itself. In Java, public classes must have the same name as the file that contains the source code. This is done to speed up searching through packages and libraries for specific classes that must be loaded into the system. This constraint is not placed on private classes or package-private classes. The difference between these two types of classes is that the scope of the private classes is limited to the class itself or to an outer class if the class is also an inner class. Package-private classes on the other hand are visible to any class that is in the same package but out of scope of any class in a different package. Thus a java source code file can contain multiple package-private classes that can interact with each other since they all belong to the same package. In the case of this project, an assignment that wants to make use of multiple classes should create multiple package-private classes that all belong to the default package. If the tests are also in the same package then the system has support for more than one class at a time.

### 5.2.3 Technical Details

Making a choice is one thing but implementing that choice in an already existing structure is quite another.

#### Compiler

The ECJ compiler can be downloaded from the Eclipse Foundation website as a standalone JAR package. The system uses the ECJ-4.3M5a version for the reason that the newest to date version, version ECJ-4.3M6, contains a critical bug. An interesting issue was found with this compiler when the system was run in secure mode using the Java security manager. In the ECJ-4.3M6 version a bug was introduced in which a critical class was not signed correctly resulting in a compilation failure due to a security exception in the system class loader. The ECJ-4.3M version does not have this issue but breaks the existing code because the BatchCompiler class is in a different package. The system uses the ECJ-4.3M5a such that it compiles but does not break the code.

Along with the compiler, a helper class was introduced to format the options correctly. It transforms a list of options and file names into a single command string. The compiler class in the ECJ package, is called the `BatchCompiler` and it accepts the formatted string from the helper class. The compiler can then be called using the `JavaCompilerWrapper` which takes the source directory as single argument. This allows for simple calls to compile all the Java files found in this directory.

### Testing

Testing of the user code via the JUnit testing framework is done by calling the `JavaTestWrapper`. Similar to the `JavaCompilerWrapper`, it takes a single directory as argument and runs every JUnit test that is found directly in that directory.

### Class Loader

The `JavaTestWrapper` is where the class loader, the `InsulatingClassLoader`, is called that loads the user code into the system. This class loader is used by both the Java and Scala test wrappers. The choice was made to use the same class loader for both Scala and Java. However, Scala uses virtual files and Java uses physical files to store class data. A new overarching abstract file type was created in order to resolve these differences. The new class loader uses the abstract file and thus a single class loader can be used for both systems. The directory that is given as argument to the wrapper is wrapped as an abstract directory such that it can be a Scala virtual directory or a Java physical directory.

### File Names

In order to allow proper naming of the solution and test classes according to the Java standard convention, some changes had to be made to the system. Firstly, the test file written to the server had to be named properly. Tests are called `UTest.java` instead of the standard `test.java`. The name `Test.java` cannot be used because of the `Test` annotation in the JUnit testing framework. Thus instead of having to use fully qualified annotation names during testing, the test file and class were renamed. The test class must be a public class in order for it to be JUnit compatible thus the file; the file and class name have to be the same in order for it to compile. The other major change to the system was to the parsed output when the specification tests were run. A new filter was written to account for the different names used when the code language was Java.

### Multi-Class Support

The technical details are very limited because the only changes that have to be made are to the assignment code itself. For this reason, we chose to use the built in functionality of Java rather than rewrite a good portion of the system.

## 5.3 Group support

All the assignments for the Web & Database Technology course are performed in groups of two students. The students are allowed to form their own pairs. However, it would be best to create a more flexible system, so it can be used by other courses as well.
In this section, like the previous sections, will start with listing the options, then continue with the choices made and end with some implementation details.

### 5.3.1 Options

In creating a system to support the group assignments, several design decisions have to be made. A flexibility system can, for example, be created by having a flexible group size. Another is having both group assignments and individual assignments within the same course.

#### Implementation of the group assignment

The implementation of a group assignment can be done in two different ways. The first way is to create a special type of assignment, a `GroupAssignment`. The second way is to implement the group-feature within the already existing `Assignment`. In this way, all current assignments can be marked as an assignment to be made with groups.

The `GroupAssignment` option is an option that can be created stand-alone. The current `Assignment` and `Question` classes do not need to be edited. Therefore, it will not modify the existing system. However, it is not possible to switch an assignment later on into an assignment to be made in groups. Also, the existing assignments in the system can not be transformed into a `GroupAssignment` easily.

If all assignments will get the option to be a group assignment, it will be possible to transform all existing questions into a group assignment. However, all code of the existing assignments (essay question, multiple choice question and a programming question) needs to be edited. This may cause conflicts with the current system.

#### Concurrency editing of assignment

Another aspect of the implementation of group assignments is the way concurrent editing is supported: can two team members work together on one assignment? Or do they have to wait their turn?

An option could be to implement a collaborative real-time editor where all group members can see each other typing and editing the code; something like Google Docs what features. This is a very neat solution, but would require a lot of work.

Another possibility is to create a locking system: two users are not able to edit the same submission at the same time. If one edits it, the other one has to wait until the first one is done. It would be possible to create some extra features, like viewing rights while the other is editing or a possibility to 'steal' the lock from the owner.

It is also possible to leave the responsibility of working together to the group themselves. You will always be able to edit the submission, but it is possible that you overwrite someone elses code.

#### Groups

Groups can be implemented in several ways. It's possible to have a group for the entire course, or to have different groups per assignment.

A group for the entire course can be created before assignments have been published. However, this will also have the consequence that all group assignments will be performed with a group of the same size. It is not possible to create a few assignments for a group of four persons and some assignments to be made in pairs.

It is also possible to create a separate group per assignment. For every assignment all students will have to form a new group, which can be of a different size. Students do have to wait until the assignment is published before they can create their groups. This can be quite time consuming if for example twice a week a new assignment is published.

#### Creation of groups

The creation of groups can be done by students or by the teacher. The enrolling of students into groups could also be done by the students themselves or by the teacher.

A first option would be that the students will take care of the complete process: they will have to create new groups themselves and enrol into these groups. A disadvantage can be that students will not divide themselves equally which may cause a lot of half filled groups. A minimum group size might help with this problem.

Another option is that the teacher will create the groups for the students in which students can enrol themselves. This will lead to a number of groups that is perfectly suited for the number of students.

It is also possible that the teacher will create all groups and enrol all students into these groups (perhaps randomly). With this option the students do not have the freedom to choose with whom they want to work.

### 5.3.2 Choices

For every option a choice has to be made. Choices are made using the current system (how well the solution fits), the time available to implement and the flexibility of the system.

#### Implementation of the group assignment

For the implementation of the group assignments, two options could be identified: a special `GroupAssignment` or by implementing it in the already existing `Assignment` entity. At first, the option to implement a special `GroupAssignment` was chosen, because the current system does not need to be modified for this stand-alone option. Therefore, it could be easily added to the running system later on. Also, it did not seem very realistic that already existing assignments need to be transformed into group assignments: all assignments are created each course edition and not copied.

However, to implement the special `GroupAssignment`, a lot of functions needed to be copied, only with small adjustments. It was not possible to override only a small part of the `BasicAssignment`. And nesting an assignment within another assignment (in a `GroupAssignment` some `BasicAssignment` would be included) caused a lot of complications. These complications were: grades (the grade of the nested `Assignment` had to be copied to the `GroupAssignment`), direct access to the nested `Assignment` could not be easily prevented, and more.

Therefore, it was decided that the best option to implement would be an extra option in the group assignment. All different `Assignments` had to be tweaked a bit to let this option work, but after that everything worked smoothly, without code duplication or complications that occurred suddenly.

#### Concurrency editing of assignment

A complete collaborative real-time editor would be very nice to add to the WebLab system. However, it would require a lot of time to implement. A complete Bachelorproject could be spend on concurrent editing only. Therefore, it will be a feature to be added on the wishlist.

No concurrency editing control at all is not an option: students will get very frustrated when their submissions disappear because some one else of their group simply overrides it. A locking option would be a good compromise. If someone edits the assignment, he/she gets a lock on it. If another user tries to edit it, it will see a screen where the lock can be stolen. Furthermore, the lock is automatically released after 3 hours of inactivity from the owner of the lock.

#### Groups

The options were to have a group per assignment or a group per course edition. Because of the flexibility to have different group sizes within one course edition, the option to have groups per assignment was chosen. A disadvantage of this option is that the groups have to be created again for every assignment. However, this can be easily overcome by creating a group `AssignmentCollection`. All assignments within the collection will then be linked to the same group, which only has to be created once.
(So you can still use this feature to set groups once for the whole course).

#### Creation of groups

For the creation of groups there were three options: the student creates and enrols to the groups, the teacher creates and enrols the students and a mixed form where the teacher creates but the students themselves enrol to the groups.

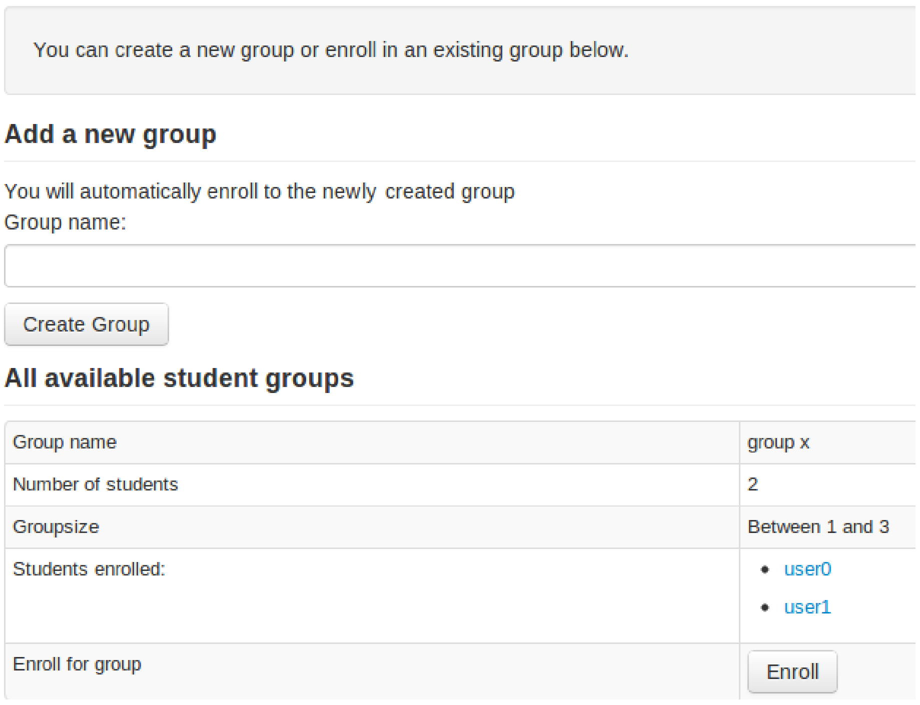
However, the three options did not seem to exclude each other. Therefore, all three were implemented. On the edit page of a group assignment the teacher can choose which of the three options he wants to enable. Therefore, it is both possible to have a group assignment with randomly created groups of a certain size as well as groups created and formed by students themselves.

### 5.3.3 Technical details

The group functionality consists of two parts: the groups themselves and the group assignment. Both parts will be discussed in the following section.

#### Student groups

If an assignment has been marked as a group assignment and students are allowed to enrol and create their own groups, students are shown a view like in Figure 5.5. Students can create groups and enrol into existing and available ones. If a group is full —i.e. the group has the maximum amount of students enrolled— it will not be shown in the list of groups anymore.


You can create a new group or enroll in an existing group below.

**Add a new group**

You will automatically enroll to the newly created group

Group name:

**All available student groups**

| | |
|---|---|
| Group name | group x |
| Number of students | 2 |
| Groupsize | Between 1 and 3 |
| Students enrolled: | • user0<br>• user1 |
| Enroll for group | Enroll |

Figure 5.5: Student view when not enrolled to a group

If a student enrols, it will not be added to the group immediately. One of the other group members will have to accept the student by clicking on the accept button, as shown in Figure 5.6. This is also the view shown to every student enrolled to a certain group.

A teacher can also edit the groups. The special edit page for teachers is shown in Figure 5.7. If a teacher clicks on a user, a pop-over will appear. The teacher can then enrol the user to a group of unenrol the user from a certain group. If a user is unenrolled, there is an extra option to send his submission (plain text) by e-mail. This is because the user will no longer be able to view the submission, since he/she is no longer a member of the group.

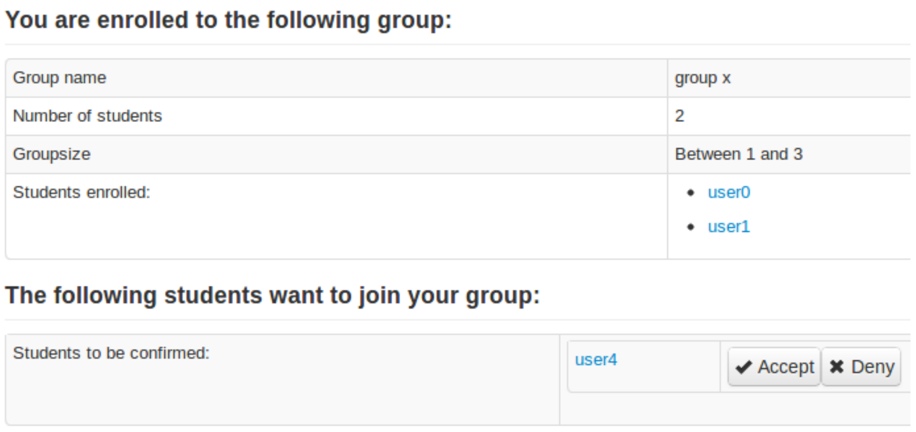
For the implementation of the student groups two new files were created: the `studentgroup-model` and the `studentgroup-view`. In the view, the code for the two different pages (student and teacher view) are defined, including all the buttons and pop-overs.

In the model, all functions are defined. These include enrolling and unenrolling students, confirming students enrolment, adding groups and creating random groups.

### Group assignment

A teacher or course manager can select whether an assignment is to be made with or without groups at the edit assignment page which is shown in Figure 5.8.

The group assignment has been implemented by editing several classes in the current system. First of all, the extra tab ‘Groups’ on the edit page has

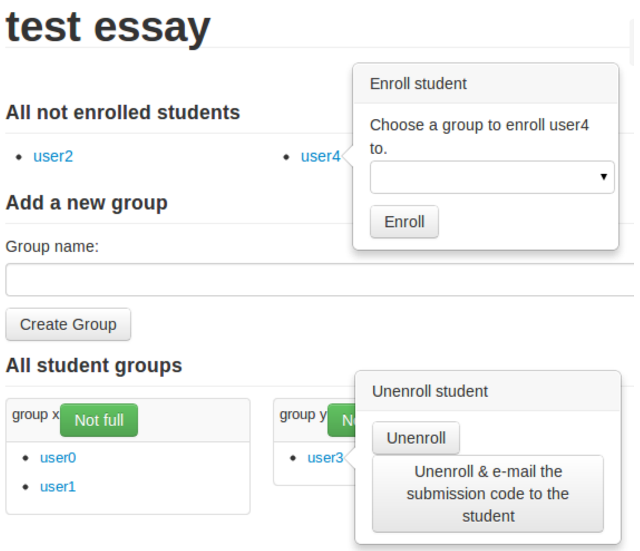

You are enrolled to the following group:

| | |
|---|---|
| Group name | group x |
| Number of students | 2 |
| Groupsize | Between 1 and 3 |
| Students enrolled: | • user0<br>• user1 |

The following students want to join your group:

| | |
|---|---|
| Students to be confirmed: | user4 ✔ Accept ✖ Deny |

Figure 5.6: Student view when enrolled to a group


test essay

All not enrolled students

- user2
- user4

Enroll student

Choose a group to enroll user4 to.

Enroll

Add a new group

Group name:

Create Group

All student groups

group x Not full
- user0
- user1

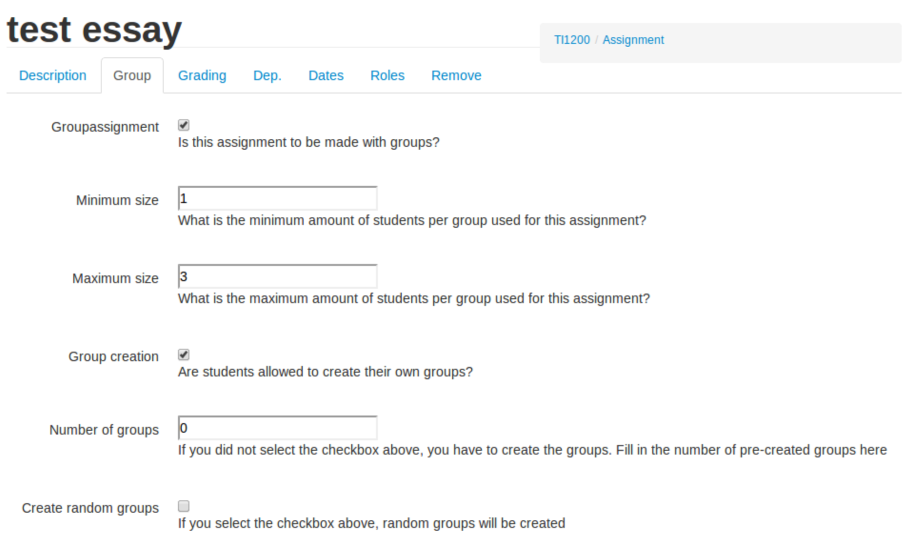
group y N
- user3

Unenroll student

Unenroll

Unenroll & e-mail the submission code to the student

Figure 5.7: Teacher view when editing groups


test essay

TI1200 / Assignment

Groupassignment ☑
Is this assignment to be made with groups?

Minimum size: 1
What is the minimum amount of students per group used for this assignment?

Maximum size: 3
What is the maximum amount of students per group used for this assignment?

Group creation ☑
Are students allowed to create their own groups?

Number of groups: 0
If you did not select the checkbox above, you have to create the groups. Fill in the number of pre-created groups here

Create random groups ☐
If you select the checkbox above, random groups will be created

Figure 5.8: Teacher view when editing an assignment

been added to the general `Assignment-view`. The `AssignmentSubmission` also needs to be edited. The `AssignmentSubmission` is normally coupled to a single student. Now it also saves a reference to the student group if the assignment is to be made with groups. Furthermore, the access control rules had to be edited. Generally, each view uses a `mayView()` and a `mayEdit()` method for access control. These methods were edited for the `Assignment` and the `AssignmentSubmission` entities, so all group members got access to a group member submission.

On the edit page several other options are available. The teacher can set the minimum and maximum group sizes. The teacher can also select whether students are allowed to create their own groups. He can also set a number of pre-created groups. A teacher can always add and delete groups after the creation process. It is also possible to have the system create the groups and enrol the students randomly.

The option to create random groups is implemented using two algorithms. First of all the students are randomized. In WebDSL no random list function is included. It does have a function that produces a random float number between 0 and 1. Using this random float number, a random list is created using the modern version of Fisher and Yates' algorithm [? ]. This algorithm works as follows:

```
To shuffle an array a of n elements (indices 0..n-1):
  for i from n-1 downto 1 do
       j := random integer with 0 <= j <= i
       exchange a[j] and a[i]
```

The second part of creating random groups consists of determining how many groups have to be created. First, a number of groups is decided by calculating the ratio between the students and the maximum group size. This ratio needs to be corrected for half groups. Next, there has to be a check whether the number of groups also applies to the second rule: the minimum amount of students per group. If this does not hold, a message is shown to the teacher to change the minimum or maximum size of the group.

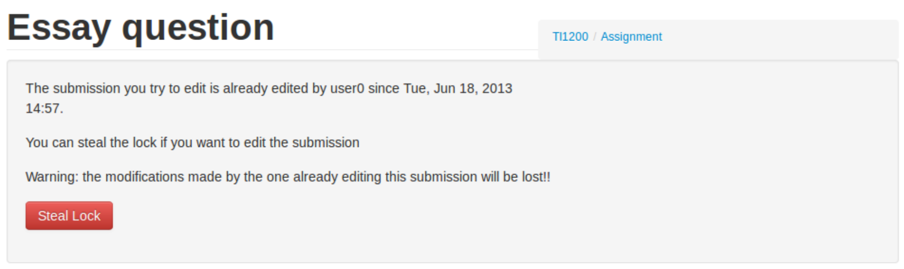
```
var numofgroups := ((students.length + maxgroupsize-1) / (maxgroupsize));
if( (students.length.floatValue()/numofgroups.floatValue()).floor() >=
    mingroupsize) {
   //create groups and fill them with students
}
else{
   message("Not possible to create the groups with the given minimum and
       maximumsize. Please decrease the minimumsize or increase the
       maximumsize and try again");
}
```

The last part consists of adding students to the groups. This is done by looping through the list of random students. The first student is added to group 1, the second to group 2, etc. If every group has one member, the next student will be added to the first group again. By adding the students in this way, all groups will be as even as possible: there is at most a difference of one student between the group-sizes of all the groups.

### Locking

If a student edits a group assignment submission, he/she will receive a lock on the submission. A student can have multiple locks so he can work on different assignments at the same time. If the submission is saved or submitted, the lock will be released and other students are free to obtain the lock on their group submission themselves.

If another student tries to edit the submission while someone else is editing it, a lock screen like the one in Figure 5.9 is shown. The student can steal this lock by clicking on the button. Then he/she can edit the assignment.



The submission you try to edit is already edited by user0 since Tue, Jun 18, 2013 14:57.

You can steal the lock if you want to edit the submission

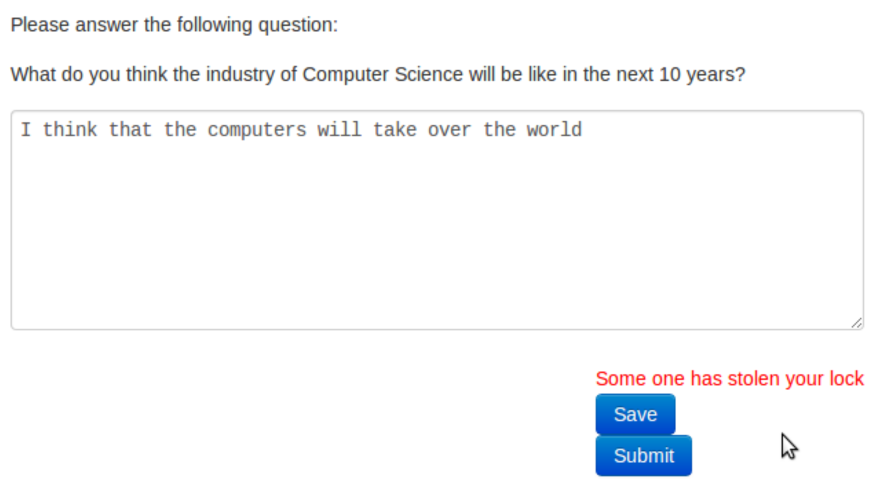
Warning: the modifications made by the one already editing this submission will be lost!!

Figure 5.9: View shown when another member of your group already edits the assignment

The student who originally had the lock will get notified about the stolen lock when he tries to save or submit the assignment (see Figure 5.10). He/she then has the opportunity to copy his code and save it locally.

## 5.4 Random Assignment

In the Web & Database Technology course, every group receives a small set of exercises, which are arbitrarily chosen from a large database filled with assign-


Please answer the following question:

What do you think the industry of Computer Science will be like in the next 10 years?

I think that the computers will take over the world

Some one has stolen your lock

Figure 5.10: View shown to a groupmember when his lock has been stolen

ments. The possibility to give each student a random set of assignments should also be implemented in WebLab. In this section the different implementation options will be discussed, followed by the choices taken and the technical details of the implementation itself.

### 5.4.1 Options

To create the random assignments, it would be best to reuse the current system as much as possible. Therefore, it would make sense to reuse the already existing `AssignmentCollection` in some way. However, different options are then still available: where to save the overall collection and the personal collection. These will be discussed in the first subsection. Another thing to consider is the timing: when will all the personal collections be created? The options for that will be discussed at the second subsection.

#### Saving the overall & personal collections

The first possibility to implement the random assignment, is to create a completely new type of `Assignment`, a `RandomAssignmentCollection`. A normal `AssignmentCollection` is saved as an attribute within this entity. In this `AssignmentCollection` the overall collection of all the assignments is saved. Furthermore, a number of exercises per student is saved within the `RandomAssignmentCollection`. For each student a `PersonalRandomAssignmentCollection` is created. Within this entity another `AssignmentCollection` is saved, with only the selected (personal) assignments included.

The second option is like the first one, except that there is no special `RandomAssignmentCollection` is created. The default `AssignmentCollection` is used instead, extended with two extra fields to mark the collection as a random collection and to set the number of assignments per student. There will be a `PersonalRandomAssignmentCollection` created per student.

The third option is to not create any new entities at all. Instead, the `AssignmentCollection` is used for saving all the assignments. The personal collection will be saved in the `StudentInCourse` profile, so an option per student.

#### When to create the personal collections

Creation the personal collection can be done at two points: at the publication of the assignment itself, or when a student opens the exercise for the first time.

When creating the random personal collections at once, the collections can be created in such a way that for example all exercises are performed by at least one student and are evenly spread among the group.

However, it is not easily possible to add a new student after the division of exercises has already been done. This can be overcome by only generating the personal selection when the student visits the page for the first time.

### 5.4.2 Choices

For every option a choice has to be made. Choices are made using the current system (how well does the solution fit), the time available to implement and the flexibility of the option.

#### Saving the overall & personal collections

Three options were identified, differentiating in the new entities to be created: there could be a `RandomAssignmentCollection` and a `PersonalRandomAssignmentCollection`, only a `PersonalRandomAssignmentCollection`, or no new entities.

Although it was stated in the options that reuse of existing entities (mainly the `AssignmentCollection`) would be preferable, the option of not creating any new entities would not work very well. In this option, the `StudentInCourse` entity is used to save the personal collections of students. This doesn't give enough flexibility: remember that Web & Database requires both group assignments and random assignments. Saving the personal (group) collection in the profile of one of the students of the group doesn't make sense at all. Furthermore, saving the collection in the students' profile doesn't lead to a clear object-oriented structure of the system.

The first option involves creating a completely new type of an assignment, while little extra functionality is needed: the number of assignments per student and marking an assignment as a random assignment. A lot of dependencies and functionalities within the system - for example the grading of assignments - will cause that a lot of existing code will have to be edited and copied to make a new type of `Assignment` work correctly.

Therefore our choice is to use the `AssignmentCollection` for saving the overall set of assignments and a special `PersonalRandomAssignmentCollection` for saving the personal set of assignments per student.

#### When to create the personal collections

Two options were defined: creating the personal collections at the publication date, or creating the personal collections when a student tries to view the exercise for the first time.

An important factor is that creating the personal collections at the publication date, does not deal with students that enroll later on at the course. Also, personal collections are created for students who do not want to follow the course completely (for example students that already completed the assignments previous year but failed for the final exam).

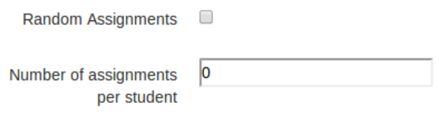
So the best option is to create the personal selections when a student opens the assignment for the first time, so each student will have a personal selection for sure.

### 5.4.3 Technical details

Implementing the random assignments consists of two different tasks. First of all, it requires implementing the options needed to create an overall collection in which all the assignments will be added. Secondly, the personal collections need to be created. Apart from implementing these two options, there is also another feature to be aware of: the random assignments should also work properly with groups. How this goal was reached can be read in the last subsection.

#### Implementing the random collection of assignments

First of all, the general `AssignmentCollection` has an option to mark it as a random assignment. There is also the number of assignments per students that should be given. This was simply added to the default edit form of an `AssignmentCollection`, as shown in Figure 5.11.



Random Assignments ☐

Number of assignments per student: 0

Figure 5.11: Additions to set an `AssignmentCollection` as random

**Implementing the personal collections**

The PersonalRandomAssignmentCollection (PRAC) is an extension of the general AssignmentCollection, as shown in Figure 5.12.

An AssignmentCollection has an attribute assignments, which is a list of assignments that have this as a parent. However, this attribute can not be used at a PRAC, because this would mean that all parents would have to change from the general AssignmentCollection to a specific PRAC (an assignment can only have one parent). Therefore, another reference to the assignments is used, called personalselection.

Another attribute added to the PRAC is the parentcollection. This is a link to the overall collection where all the assignments were selected from. The general field parent refers to the parentcollection.parent. Furthermore, the PRAC also has the attribute student: the owner of the personal collection.

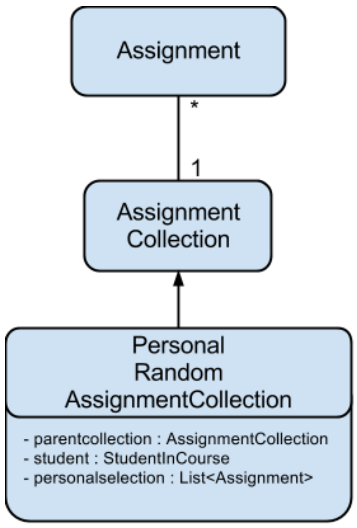

Assignment
*
1
Assignment Collection
Personal Random AssignmentCollection
- parentcollection : AssignmentCollection
- student : StudentInCourse
- personalselection : List<Assignment>

Figure 5.12: Random assignment diagram

If a student visits the AssignmentCollection for the first time, a random personal collection is created. Generating the random collection is done as follows:

1. Get the assignments of the overall collection (parentcollection)
2. Check if the number of assignments in the parentcollection is equal to or larger than the number of assignments per student
3. Generate a randomized list of assignments
4. Add the correct number of assignments to the PRAC from the randomized list of assignments, starting at the front of the list

Generating a randomized list is done by Fisher and Yates' algorithm [**?** ], the same algorithm that is used for randomized groups. This algorithm is described in section 5.3.3.

After generating the PRAC for a student, it is shown to him directly. An example of an overall collection created by the teacher is shown in Figure 5.13, just like a normal AssignmentCollection.

If the teacher marked the AssignmentCollection as random and set the number of assignments per students on two, the student will be shown a view like you see in Figure 5.14.

#### Combination random assignments with groups

The random assignments should also work properly with the group support as described in section 5.3.

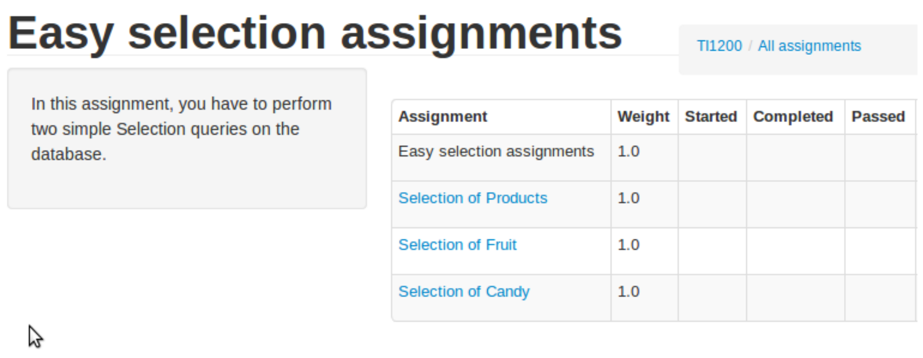
The first change needed to make the combination work, is to copy the groups to the personally created random assignments. This is done immediately after creating and filling the PRAC. The groups from the parent collection (for example, randomly generated by the teacher) are automatically copied to each of the assignments.

Furthermore, it needs to be able to find a PRAC that belongs to a group. A reference to the group is not saved in the PRAC. Instead, the normal findGroup() method is used to find the group belonging to the above assignment collection. Next, each student in the group is looked up to see whether he/she has a PRAC assigned to him. Then the other students will get to see this PRAC as well.

Because the PRAC is a subclass of AssignmentCollection and its submissions are just the normal submissions as used everywhere in the system, no further modifications where needed to implement the group functionality: this was already implemented before.

## 5.5 Unimplemented Features

Some of the features we identified were not implemented due to time constraints. This section contains our considerations on the different features. In particular subsection 5.5.3 contains a lot of information because the feature was dropped only after the design phase was already finished.
Future developers of this project may want to take the given ideas and designs into consideration.


Easy selection assignments

TI1200 / All assignments

In this assignment, you have to perform two simple Selection queries on the database.

| Assignment | Weight | Started | Completed | Passed |
|---|---|---|---|---|
| Easy selection assignments | 1.0 | | | |
| Selection of Products | 1.0 | | | |
| Selection of Fruit | 1.0 | | | |
| Selection of Candy | 1.0 | | | |

Figure 5.13: The overall AssignmentCollection

### 5.5.1 Multiple Tries

Students and groups may have to submit an assignment, receive feedback, and submit again. Thus they have multiple tries to submit an assignment. Any number of tries may be given and a penalty could be imposed on anyone who needs an additional number of submissions. This should be compatible with the old single student model as well as the newly implemented groups.

Some research was done into how this feature could be integrated into the new system and there are two options. The first option is to use copies of an assignment and then to provide back links so the previous tries can be easily accessed. The other option is to expand the functionality of the Assignment entity by using a set of submissions, deadlines, and grades.

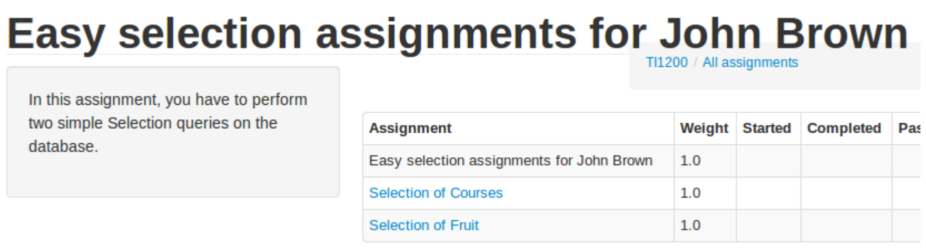
The first option is problematic because each student or group is working with a different set of assignments. The first group finishes the assignment in one try while another takes three tries. This creates unnecessary copies.

The second option is cleaner to implement and does not create unnecessary copies of assignments. To include this functionality in the system only front-end changes have to be made. First a grader has to be able to tell a group that they have to resubmit the assignment; this would involve creating a 'Try Again' button or checkbox. Then the grading scheme must be changed according to a certain strategy. One strategy is to take the highest grade while another is to take the last grade in a list of grades. Whatever strategy is ultimately chosen a function has to be written to select a certain grade from a list of grades. Finally some changes have to be made to the `Assignment Submission` entity. The submitted status must be set when the last try is approved and not on the submission of the first try. Another change is that a list or set of grades must be used instead of a single grade.

### 5.5.2 TA Tests

When we decided to make unit testing available for MySQL assignments, one issue immediately arose. The current set of MySQL assignments is very big (±900) and it would be a lot of work to write a test set for each of these assignments. This problem might be solved if TA's of the course could add a few specification tests; while grading the assignments for example.

The point of allowing TAs to add tests is to be able to steadily expand the test set over time. One of the rules of WebLab, however, is that specification tests cannot normally be added after the assignment is published —i.e. available


Easy selection assignments for John Brown

TI1200 / All assignments

In this assignment, you have to perform two simple Selection queries on the database.

| Assignment | Weight | Started | Completed | Pas |
|---|---|---|---|---|
| Easy selection assignments for John Brown | 1.0 | | | |
| Selection of Courses | 1.0 | | | |
| Selection of Fruit | 1.0 | | | |

Figure 5.14: Personal selection of assignments

to the students— because this could change the grade of a student after the student has already checked his test score. To accommodate this rule, it may be best to simply gather the TA tests after the assignment deadline, to use them for next year's course edition. This would mean, however, that the first edition of the course on WebLab would contain almost no tests. If someone would want to implement this feature later on, these are the things that should be considered.

### 5.5.3 Project Assignment[10]

This feature is allows students and groups to work on code that involves more than one class. Code can then be split into different packages and more complex systems can be used as assignment material. For example, an assignment could be a full term project instead of a just weekly assignment.

After doing the research into what would have to be changed in the system to provide support for Project Assignments, we conclude that there are two options. The first option would be to re-engineer the system architecture of the assignments such that each regular assignment can be placed in a Project Assignment. This would, from a software engineering perspective, be the clean solution because it makes the system very modular and allows for broad future expansions. However, this would require changing a running system, rewriting the full architecture, and then migrating all the old data into the new system. This is beyond the scope of this project and highly impractical. The other option is to use the current system and fit the project management into it by extending the current Assignment class with another subclass. This option is much more practical because it requires no data migration and or other major changes to the running system. It is however not a very clean solution because it trades in modularity for practicality; the system will be less expandable and will create problems along the line with other, more advanced features.

#### File Management

In order to be able to support complex directory structures, the system must be able to manage virtual files and save them to the database. This file management system has been implemented and tested. There are `Directory` and `File` instances, both of which are subclasses of `AbstractFile`. Each `AbstractFile` has a `MetaData` object associated with it that stores information such as the visibility. Each `Directory` has its own `DirectoryMetaData` such that it can do recursive operations. Finally each `File` has its own `FileMetaData` in which the language and file extension are stored. Note that in the actual implementation `File` has been renamed to `ProgrammingFile` because WebDSL already had a standard type name `File`. A class diagram is shown in Figure 5.15.

#### File Management Integration

The file management system that was discussed earlier has to be integrated with the Project Assignment. The Project Assignment is a subclass of the `Assignment` entity in which a root directory is installed and permissions to make changes should be regulated via the access control rules. This new assignment must also have group support so a project can be done as a group.

[10]also known as the multi-file feature.

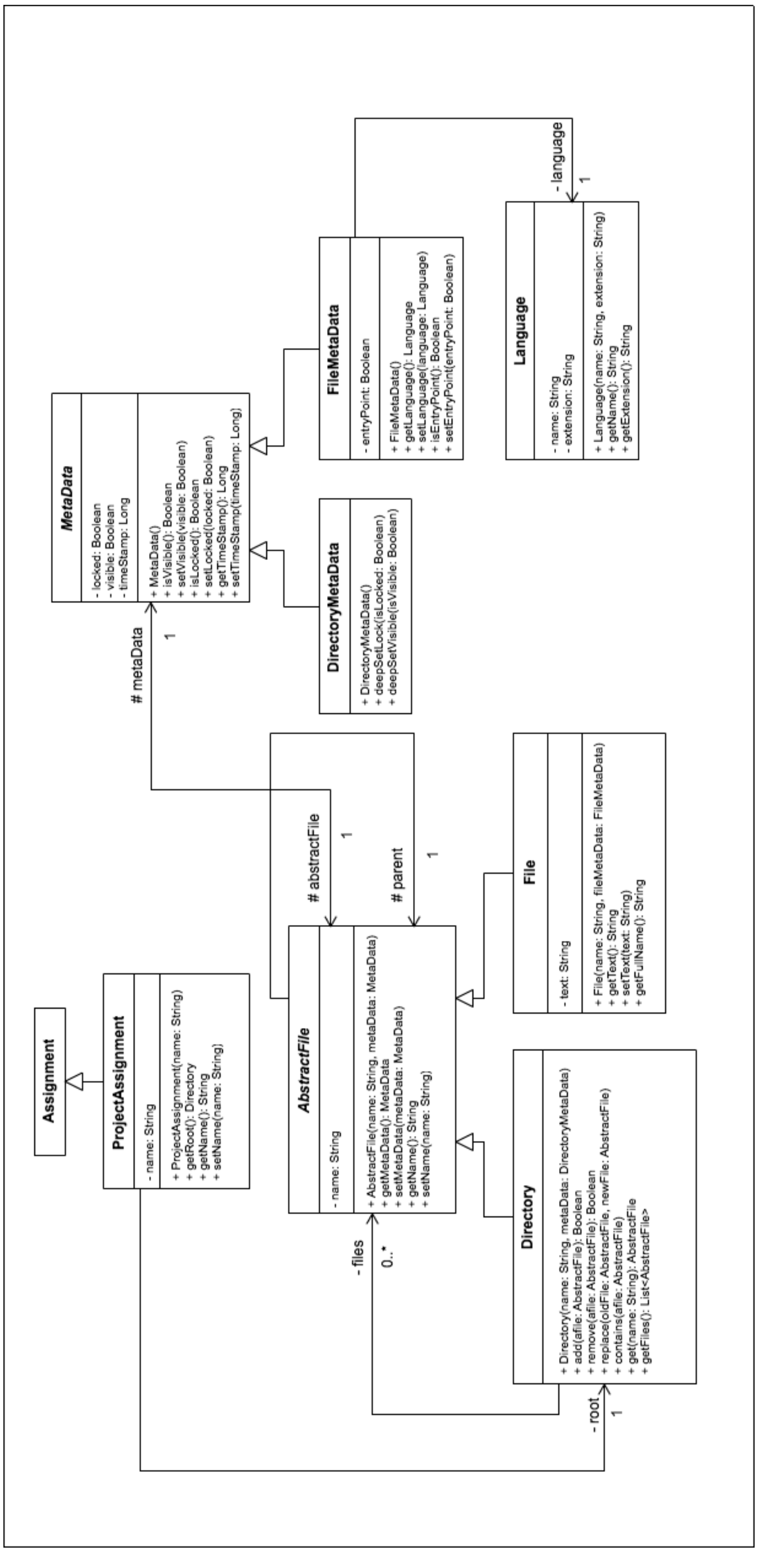

Assignment
ProjectAssignment
- name: String
+ ProjectAssignment(name: String)
+ getRoot(): Directory
+ getName(): String
+ setName(name: String)
AbstractFile
- name: String
+ AbstractFile(name: String, metaData: MetaData)
+ getMetaData(): MetaData
+ setMetaData(metaData: MetaData)
+ getName(): String
+ setName(name: String)
File
- text: String
+ File(name: String, fileMetaData: FileMetaData)
+ getText(): String
+ setText(text: String)
+ getFullName(): String
Directory
+ Directory(name: String, metaData: DirectoryMetaData)
+ add(afile: AbstractFile): Boolean
+ remove(afile: AbstractFile): Boolean
+ replace(oldFile: AbstractFile, newFile: AbstractFile)
+ contains(afile: AbstractFile)
+ get(name: String): AbstractFile
+ getFiles(): List<AbstractFile>
MetaData
- locked: Boolean
- visible: Boolean
- timeStamp: Long
+ MetaData()
+ isVisible(): Boolean
+ setVisible(visible: Boolean)
+ isLocked(): Boolean
+ setLocked(locked: Boolean)
+ getTimeStamp(): Long
+ setTimeStamp(timeStamp: Long)
DirectoryMetaData
+ DirectoryMetaData()
+ deepSetLock(isLocked: Boolean)
+ deepSetVisible(isVisible: Boolean)
FileMetaData
- entryPoint: Boolean
+ FileMetaData()
+ getLanguage(): Language
+ setLanguage(language: Language)
+ isEntryPoint(): Boolean
+ setEntryPoint(entryPoint: Boolean)
Language
- name: String
- extension: String
+ Language(name: String, extension: String)
+ getName(): String
+ getExtension(): String
- language
1
# metaData
1
# abstractFile
1
# parent
1
- files
0..*
- root
1

Figure 5.15: File Management Class Diagram

### User Interface Changes

The biggest changes will occur in the user interface. Firstly the directory structure of the project should be viewable as a tree. Then per user the view may change based on certain preferences and privileges. Course managers should be able to view everything including any otherwise hidden directories and files (Figure 5.16a). They should be able to add, delete, and edit any directory or file except for the root directory itself. The root directory should be permanently hidden in order to prevent any unexpected editing behaviour. Students should only see directories and files that are set to "visible" (Figure 5.16b). They should only be able to edit anything that isn't locked. Finally graders should be able to see all the code, the student's solution as well as the manager's solution (Figure 5.16c).

### Entry Points

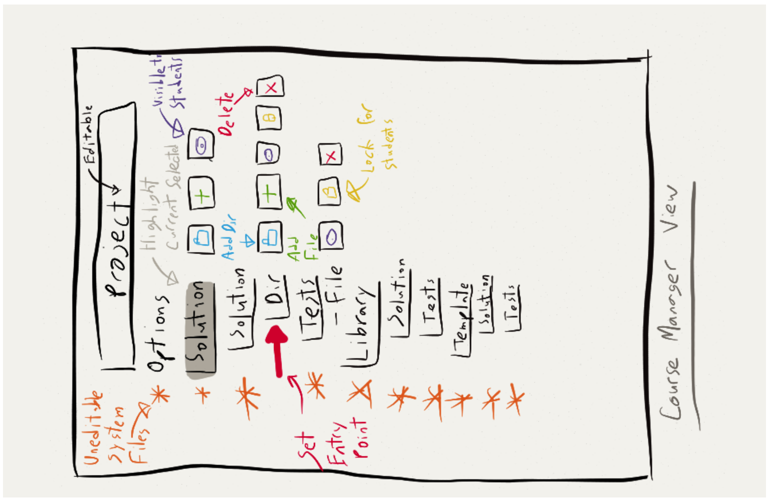
Another major change is the response of the system when the save and run buttons are clicked. In the current system it is clear which files are to be compiled and run. In a project an entry point must be set, manually or automatically. For example, if a file is open then the response might be to compile the project and run the current file. Otherwise, the selected file in the files view will be run. Finally if no file is selected then the user could always set an entry point or the system can give an error indicating that it does not know which file to run.

### Shared Changes

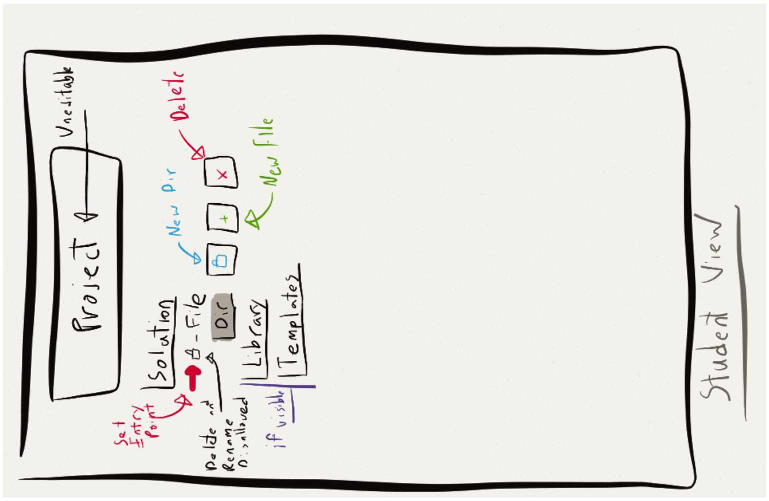
In the shared package, the code that regulates data transfers between the front and back-end, some changes must be made as well. The `XMLExecutionTask` has to be changed to contain the code of multiple files and directories such that a whole project can be compiled. Furthermore, the shared framework needs to know what file to run as entry point so this can be passed along to the back-end. Things get even more complex if projects contain files of different languages because then a build order must be created in order to compile the code correctly.

### Back End Changes

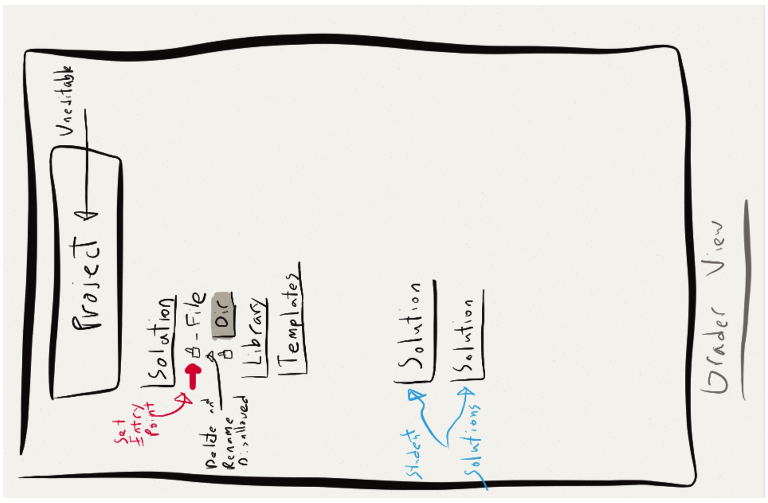
The last changes would have to be made in the back end. Compilation has to be done according to a specified build order and if it fails at any stage it should return the appropriate output. Also, any output that is collected when the code is compiled and run must be filtered such that no system information is seen by the user. Finally, the right tests must be run which could involve several testing frameworks in different languages.



(a) Manager View



(b) Student View



(c) Grader View

Figure 5.16: Project Management Views

# 6 Testing

Aside from implementing new features in the WebLab system we have also tested many of these features. The system itself has only been tested using some manual testing. There is no framework to facilitate system wide testing because of the many different languages and sub-systems. Therefore we have applied several techniques to test different pieces of our additions. The WebDSL test feature that is included in the language has been used to test parts of the front end model while the ScalaTest framework has been used to test various classes in the back end. Some testing has been done by hand as well; this includes the Java Security testing, SDF testing, and other front end testing. Unit testing has been used where possible but due to system dependencies this is limited in many cases. Integration and System testing have been used more but debugging is much more difficult. The system as whole remains untested and is likely to stay this way because of the demands to keep the system running and the complexity.

First the testing of the current system will be discussed, followed by an overview of the tests we performed for each part of the system in section 6.2. Next, WebDSL Testing will be discussed followed by Testing Labback with XML files, Scala Testing, Java Security Testing, Selenium Testing, and finally SDF Testing.

## 6.1 Current System Tests

The version of WebLab that we started this project with had virtually no testing. There were no WebDSL tests to verify that the front-end model behaved as expected. There were no Scala tests to verify any of the back-end procedures. There were no security tests to prove that certain security leaks could not be exploited. The only tests that were used were by-hand verification via the front-end web interface. Although it was a little surprising it was not wholly unexpected since this seems to happen more often in the industry. If the system compiles and the buttons seem to do what they should then the system "works". This approach often works and in the case of WebLab, most of the time it runs fine.

Testing, in general, is used to not only verify that the current system is working, but also that future releases do not introduce new bugs in the old code.

New functionalities can accidentally damage older code and create bugs that are not easily detected. Using backward compatibility testing, the integrity of the system can continuously be checked and new features can be added more easily. However, there is a trade-off, because all the time that is spent testing, is time that cannot be used to write code. Many software projects are under pressure to finish quickly and to meet short deadlines.

WebLab seems to have suffered from this phenomenon as well. The project is quite complex and can no longer easily be split into units; many parts are too interconnected to split up and test separately. This high amount of coupling largely influences how much we can test effectively. Some brand new code pieces can be unit tested, while changes to the existing code can only readily be checked via integration and system testing. Some functionality is buried so deep in the system that the only way to test this code is via the front end, the 'button pressing testing'. The remainder of this chapter is about what we have tested and how we used several different methods with varying results.

## 6.2 Overview of Testing

There are two types of tests performed for testing the complete system: unit testing and system testing. Unit testing is done via WebDSL tests for the WebDSL files in the front end and via ScalaTest for the back end. System testing is done using XML Testing for MySQL and Java Security Testing for Java. Selenium front end testing is used to test Group support, the Project Assignment, and the Random Assignment. Each of the testing methods is described in the sections below.

| **Feature** | | **Unit** | **System** |
|---|---|---|---|
| **MySQL** | WebLab | - | - |
| | Running queries | ScalaTest | XML Testing |
| | Unit Testing | ScalaTest | XML Testing |
| | Parser | ScalaTest | XML Testing |
| **Java** | Compiler | - | Security Testing |
| | Unit Testing | - | Security Testing |
| | Class Loader | ScalaTest | Security Testing |
| **Group support** | Student Groups | WebDSL | Selenium |
| | Group Assignment | WebDSL | Selenium |
| | Locking | - | Selenium |
| **Project Assign.** | File Management | WebDSL | - |
| **Random Assign.** | Random Collection | - | Selenium |
| | Personal Collections | WebDSL | Selenium |

Table 6.1: Overview of testing methods

In Table 6.1 it is shown for each component in the added features what kind of testing was performed. Some parts do not have tests. The MySQL WebLab part (creating databases via the front end) has not been tested due to a lack of time. The Java Compiler and the Unit Testing parts have not have been Unit tested by us because the standard ECJ compiler and JUnit testing framework have been used. We assume that these large packages have been tested by their makers.

The locking of assignments and the random collections have not been unit tested due to dependencies in the current system; all features were added by changing already existing functions slightly.

File management has not been system tested because it has not been implemented in the system itself, therefore it was not possible to test the integration with the rest of the system.

## 6.3 WebDSL Testing

WebDSL has rudimentary testing functionality. Specifically a test template can be created to run unit tests. However, there are several issues, the biggest of which is creating a clean test environment. Unit testing is meant to test a specific component or unit. It should not be dependent on anything else in the system including any previously run tests. WebDSL creates a new database, preferably an in-memory database, for each test run. Then a specific test block or unit test is executed. The system should then clean the database by dumping everything and recreating the tables. However, what really happens is that the database is cleaned by deleting each row in the database without checking for constraint violations. This way of cleaning will fail most of the time and thus the database is not cleaned between tests. To fix this issue, the user must programmatically clean the database by setting any and all references to null. Once this is done, the WebDSL unit testing can be accomplished. The testing example in appendix E is an example of successful clean unit testing in WebDSL.

For more advanced and broader tests, this system clearly lacks support. Where in unit tests a clean environment is preferable, in integration and system testing the opposite is preferable. The automatic database clean is extremely counterproductive because data needed to run the system is cleaned as well. To fix this one of two things must be done. The first option is to allow testing without cleaning the database such that critical data is not deleted. The other option is to make it possible to have a default state for the database whereby a configuration file or database dump can be used to initialise the testing database. The Group Assignment support as whole cannot be tested and the pieces that can be tested cause problems due to a high amount of coupling to the rest of the system. The current WebDSL testing suit, whereby an attempt is made to clean the database, is used but lacks any sort of proper support and runs with integrity constraint violation errors.

## 6.4 Testing Labback with XML files

The two parts of the system, WebLab and LabBack, are very loosely coupled. As described in Current System, they communicate only by writing and reading from files. Consequently, it is entirely possible to run only one of the two, and communicate with it by manually manipulating the files. We have used this strategy to our advantage by making a number of test files that can be used to test LabBack. First, in Structure of the tests the structure of the test files is explained. Then follows Usage of the tests.

### 6.4.1 Structure of the tests

When testing LabBack through the file system, you are mimicking WebLab. When WebLab has a task for LabBack, it writes a directory with the code files and an XML file containing the job details. See Listing 6.1 for an example of such an XML file.

Listing 6.1: Example of a task file: select.xml

```
<task>
  <id>select</id>
  <data-dir>/tmp/webapp/select</data-dir>
  <compile></compile>
  <code-name></code-name>
  <status>
    <pending></pending>
  </status>
  <compile-result></compile-result>
  <priv-compile-result></priv-compile-result>
  <exec-result>
    <num-cases>-1</num-cases>
    <num-failures>-1</num-failures>
  </exec-result>
  <recorded-out enabled="true"></recorded-out>
  <lang>SQL</lang>
</task>
```

In the case of this example, LabBack also expects a directory named select (as given by the `<data-dir>` tag), which would contain two files: `solution.sql` and `test.sql`. Together, this XML file and the directory with the two code files form one task for LabBack to handle, so they can be used together as one test. To run the test, you only have to copy the task file and the directory to LabBack's watch directory (normally ic/tmp/webapp). The processing of the task can be observed by LabBack's console output or log. When this is done, the output is written between the `<compile>` tags in the XML file.

### 6.4.2 Usage of the tests

Testing the system by manually copying files may seem strange, but there are certainly advantages. For one, because the test is placed right between LabBack and WebLab, it is a complete integration test for the LabBack code. Every aspect of dealing with a task is tested in one go.
An interesting aspect of this way of testing is that it actually provides a way to test the paralellism of LabBack, which is an important feature. By copying multiple tests at once into LabBack's watch directory, a scenario is simulated where multiple students are working with the system and try to compile their code. This can reveal concurrency issues, which are especially likely with MySQL code running on the same database.

**Our XML task tests** We chose to use this type of tests to test the LabBack execution of MySQL assignments. Because the MySQL code is all executed on the same database in parallel, we were certainly interested in revealing any concurrency issues. We made 19 MySQL tasks for testing, which contain a broad range of different queries. At this moment, all of these tasks can be successfully executed by LabBack when they are added at the same time.

## 6.5 Scala Testing

ScalaTest is a complete testing package for Scala code, comparable to the well known JUnit tests for Java. In a typical Scala unit test, the `expect` keyword is used to define the expected outcome of a piece of code (see Listing 6.2). We used ScalaTest to write unit tests for the MySQL classes in LabBack [1] and for some of the file system classes that were added for Java support in Labback. We have approximately 70 hand-written unit tests. These tests are mostly meant to cover the special cases and some broad categories of MySQL queries.

Executing MySQL code can be a risky business and because this part of the system contained most of the newly-written, potentially faulty code, we wanted to test this part extensively. We asked for a copy of the database containing the solutions for the assignments that are currently used in the MySQL lab. This database is used by the current system used for the MySQL lab, Papillon, to distribute assignments to students and allow the TAs to grade them. Using a small Scala class made specifically for this purpose, we extracted the 897 solution queries from the database and transformed them into 897 new Scala tests. See Listing 6.2 for an example of a test generated using one of the solutions in the Papillon database. The test runs the SQLTestHandler with an empty test, using the query from Papillon as both student and answer code. The main purpose of the test is to make sure that the code does not cause any parse or execution exceptions. The expected result is that one test is run, with zero failures and an empty list of messages.

Listing 6.2: A Scala unit test, generated from a query out of Papillon.

```
//Automatically generated test from papillon example code #5
  def test_runTests_papillon5 {
    val code = """ SELECT COUNT(afd) FROM afdeling WHERE verdieping = 1;
"""
    expect((1, 0, List())) {
      val handler = createHandler(code, code)
      //solution == answer
      handler.runTests(Map("emptyTest" -> ""), code, code)
    }
  }
```

Using this new abundance of tests, we were able to find and remove some last bugs in the MySQL code parsing and execution. The tests also serve as a proof to Dr. Hidders that the system can at least execute the current assignments correctly, making it usable for next year's course.

## 6.6 Java Security Testing

Testing is not only used to test the functionality of a system, but also the security of the system. For the Java Support feature extensive security testing was done by creating assignments with the WebLab system and executing them. These assignments can be found in appendix D.

[1]see Figure 5.3

Three different methods of security are used to make the system secure. Firstly the Java Security Manager limits many functionalities such as File I/O operations and the creation of sockets that connect to a certain port. Thus overwriting, deleting, or otherwise editing the tests via the physical file is not possible. The Java Security Manager also disallows any editing via the use of reflection; given the wrong security permissions, users would have been able edit test scores or other critical sections of code. Finally, it is also not possible to force the JVM to quit. This can normally be done via the `System.exit()` command.

However, the Security Manager does not protect two specific attack forms. Fork bombs that could cripple the system by taking over all the process ids cannot be blocked by the Java Security Manager. Thus another security measure was introduced in the class loader that reads in the user code. This class loader has a specific blacklist of classes that may not be loaded into the system. The `java.lang.Thread` class is blacklisted for the above reason and for other security reasons. This means that the system is secure against malicious attacks of this kind but limits the user in what they can do with this system. The `java.net.ServerSocket` class has been blocked for a similar reason. Using this class, server ports can be used and never released. This eventually means that the server has no more free ports and could potentially crash when a new port is requested for some process.

Even this is not enough to protect against everything we have thought of because it is possible to see the specification test names and fields using reflection. The only way to protect against this is to never return any user code output when it is run against the specification tests. The current system already did this for various reasons.

This form of testing cannot easily be repeated since it requires a running system. However, these were some of the more critical tests because one of the highest priorities of this project is keep the system secure. Future releases of WebLab should be run with this test set to show that the Java Support feature is still as secure as it was in the previous versions.

## 6.7 Selenium Testing

Another useful testing tool that we used was the Selenium automated web testing tool[2]. This open source tool can be used in several different ways, but for our purposes we used the Selenium IDE.

This tool allows a user to record mouse clicks, typing of text, and page navigation. Furthermore, several verification checks can be added. For example a message that should occur somewhere on the page, or the number of elements in a list. Then after recording these actions into a script, the system plays this XML script and automatically follows the same steps as the user and checks all the verifications set.

This form of testing can be used to check certain front end functionalities such as clicking a button to create a new group. After changes are made in the back end, the functionality of certain front end elements can easily be checked via this type of automated testing.
Selenium is also robust enough to allow placement changes in the front end.

[2] http://docs.seleniumhq.org/projects/ide/

Thus if a particular link first appears on the left side of the screen and then in the next version on the right side Selenium will still be able to follow the link as long as the identifier has not changed. This identifier can be the html class or id, or even the link text itself.

Front end testing via tools such as the Selenium IDE is an excellent way to quickly test changes to various parts of the system. The only drawback is that when the front end is changed too much, a lot of tests need to be edited. However, it is possible to create independent actions and verifications, so the complete recording doesn't have to take place again.

## 6.8 SDF Testing

The parser, written in SDF, was "manually" tested during development using Spoofax, and automated testing was done using Scala (described in section 6.5).

Spoofax offers hot-swapping capabilities of an Eclipse editor for the language that you develop, so simply opening an SQL file would give you syntax highlighting and any parse errors that occurred. It also provides a menu with so-called builders, which are bound to Stratego strategies. Among the standard ones is a builder that generates a new file containing the Abstract Syntax Tree (AST) from the contents of the opened file.

The project we used as a base already had a `test` directory with some SQL files. At first we added to those tests to test specific scenarios by hand. Later we added almost 900 sql files with valid example solutions to the questions used in the Web&Database Technology course. These were also used in the automatic tests from Scala, to test both parsing and execution.

# 7 Process

The following chapter describes our project methodology, both the planned process and the actual process, and what we have learnt from this project.

## 7.1 Planned process

We had our first meeting about the project on March 19, before it began. The discussion was about the assignment details, our view on the assignment, and the project methodology. We all agreed on an iterative approach for the software development.
The method would not be strictly Scrum, but the following was agreed upon:

- daily meeting would be after lunchtime with at least one of the supervisors
- possible features would be listed beforehand
- research on a feature would be part of a sprint
- implementation of a feature would always be preceded by a discussion with one of the supervisors
- the sprint duration would be a week

The work would be done in groups of two, possibly using pair programming.

The project code was distributed over two private git repositories. We would get read-access to both repositories, and write-access to a fork of each repository.

Our university supervisor, Andy Zaidman, heard about the plan of an iterative approach and recommended an online tool called Planbox[1].

Since there were no hardware requirements three of us would use Windows and one would use a Mac. As far as development software went, three of us preferred to use the Eclipse IDE, while one preferred a simple code editor and command line tools.

[1] http://planbox.com/

### 7.1.1 Code evaluation

A standard part of the Bachelor Project is an evaluation of the code quality by the Software Improvement Group[2] (SIG). Eight weeks into the project, the code would be reviewed for the first time. Only our own code would be reviewed, independent of the existing system.

We were expected to use this feedback and submit our code for re-evaluation at the end of the project. The project took eleven weeks in total.

## 7.2 Actual process

**Git repositories** The plan for the first week was to have a quick start, but it was not meant to be. The forks were set up, but through some administrative problems only one of us could actually access the repository. But we still had read access to the original repositories, so the week was spent getting the system running on our machines.

**Operating system** During the first week we discovered that the system was not fully cross-platform and that the compiler for WebDSL was very slow on Windows. So using a dual-boot with Ubuntu Linux, we were able to get the system running.

**Planning tool** After the first week we got access to a Planbox project and started using it. But with a small team in a separate room with a white board, it turned out to be easier to use the whiteboard for project planning.

We made lists of features, tasks for feature under development, added persons per task, and ticked off completed tasks. Using a global project time-line we kept an overview of our own (soft) deadlines and the hard deadlines of the project.

**Development environment** When one of our supervisors found out that three of us were using the Eclipse IDE, but the fourth was not, he decided for us that we should all use Eclipse. Both for consistency in build process and to make sure that everyone had experience using a complete IDE by the end of their Bachelor.

**Version Control usage** During the project the first features were developed and put on the master branch of the forked repositories. Around halfway during the project we were told to use branches more often and were given the link[3] to a guide for a branching model our supervisors were adopting for their projects. Because the first features we developed were already intertwined with the repository, we decide to create new branches for new orthogonal features only.

**Meetings** Meeting once a day with a supervisior was not practical, so instead there would usually be an official meeting once a week where we would present our progress. Another time in the week one of the supervisors would come by our room to talk.

[2]http://www.sig.eu/en/

[3]http://nvie.com/posts/a-successful-git-branching-model/

As the project progressed one or both of our supervisors would be away for a few days and face-to-face meetings would not be planned. Instead communication would be by email, followed by a meeting in our room when a supervisor had time. Although this was a lot less structured than originally planned, it worked out rather well.

**Timeline** Since there were several facets to this project the timeline is not a smooth burn down chart but rather a parallel progress diagram, shown in Figure 7.1.

**Workspace and attendance** Very fortunate for the project development was that a room was available on the floor of our supervisors. That way, they were only a few doors away. Another benefit was the other people on that floor have expertise in the WebDSL language for example. Danny Groenewegen was a big help to us with any WebDSL related problems and was able to shed some light on how to use Spoofax too.

The whole research group worked from home on Mondays. We did so too at first. Later in the project we decided to schedule our own work-at-home days when it suited us better. Usually the work of writing documentation was something that could be done from home.

### 7.2.1 Code evaluation and reaction

Our code could not be easily evaluated because our work is entangled with the existing system. Only the Scala code was actually evaluated. This part scored three out of five stars at the time of the first evaluation, which means the maintainability is average.

The full evaluation text is included in appendix B but note that it is in Dutch. Our code could be improved particularly on accounts of Unit Size and Unit Complexity. This was mostly due to methods that were too long.

After we received the evaluation we took another good look at the Scala code and split up large methods into smaller ones. Apart from that we were disappointed that we did not receive feedback on the other parts of the code we sent, but understand that WebDSL and Spoofax are not very mainstream.

At the end of the project we submitted our code for a second review. The code now scores four out of five stars. The review text mentions that the greatest increase of scores in categories Unit Size and Unit Complexity. SIG also notes that apart from improvement in maintainability, the volume of test-code has also risen.

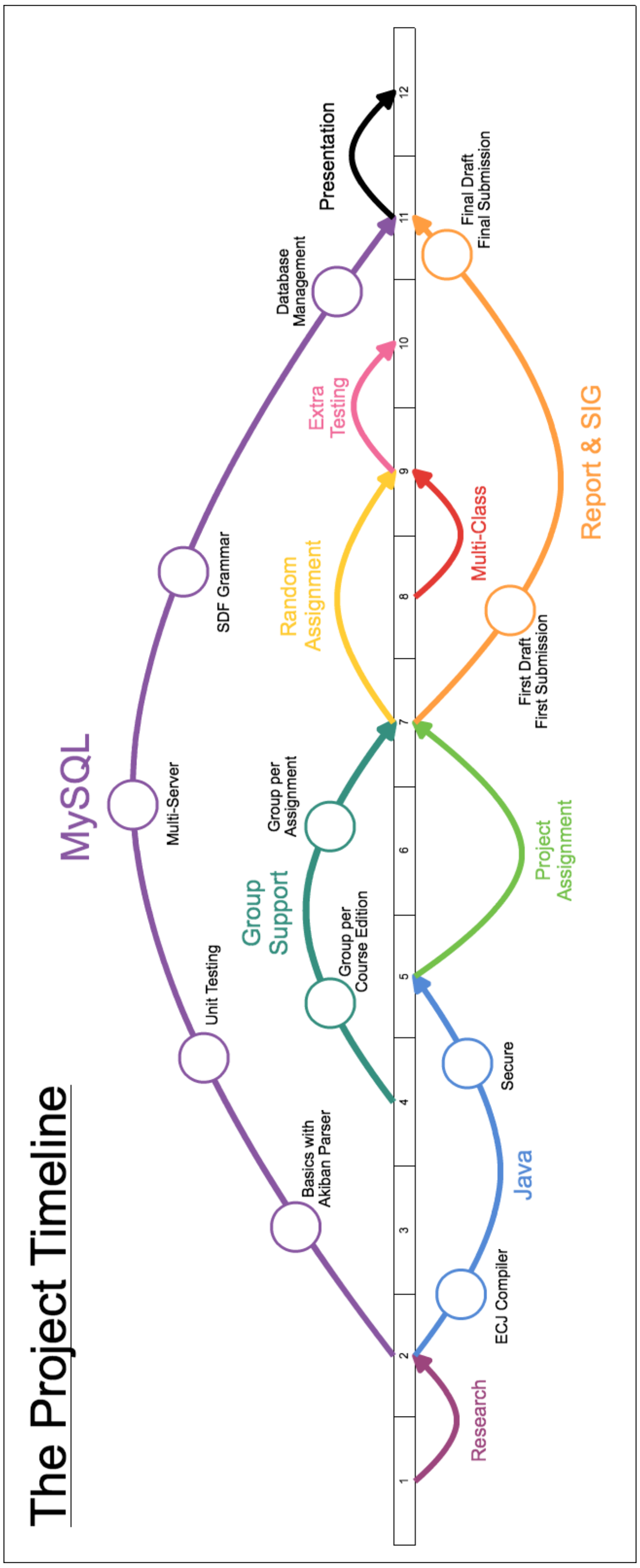


Figure 7.1: Project Timeline

## 7.3 Personal reflections

We conclude this chapter with our personal reflections on the project.

### 7.3.1 Reflection Marieke

As a Teaching Assistants for several courses and as a member of the Facultary Student Council of EEMCS (Electrical Engineering, Mathematics and Computer Science), you might say that I am quite interested in the educational part of the university. Therefore, it is not a surprise that I went looking for a Bachelor project in which I was able to improve the education in some way. I ended up at this project, improving the WebLab system used by students to make their practical assignments.

The project was a pure maintenance project; everything should fit neatly in the current system. This was difficult to start with at first. I never had an experience like that before. It was not only the existing system which gave troubles (not knowing what the underlying ideas were due to a lack of documentation), but also the programming language WebDSL itself. A completely new language which worked differently at some points than you would expect at first.

But most frustrating about WebLab was the long compilation and building time of approximately 10 minutes. Just too short to work on something different (e.g. the report), but too long to just sit back and wait. Further programming while the compilation is performed was usually useless because I did not know the system. I was not sure at all whether my modifications would work right away or whether it would need further refinements. Only while testing was it possible to write another set of unit tests while the compilation and running of the tests was performed.

We decided to work in teams at the start of the project. However, it worked out that I have worked alone the entire project on my own parts, the group support and the random assignments. I started with the group assignments, which was a slow process. I was constantly trying new things and discovering how the WebDSL language worked. I think this part would have gone much faster if we could discover WebLab and program our first parts together. The last part of the project, where I implemented the random assignments, went very fast. I knew exactly which part of the code was responsible for what and where I had to search for certain functionality. This was perfectly suited to be handled on my own without the need to program in pairs.

To conclude, I think I have reached the goal to improve the educational system by improving WebLab. I think the added features will provide an easier way for students to learn how to program. Furthermore I have learnt to work with an existing system, discovering how things should work and improving it afterwards. This experience will definitely be useful in my later career.

### 7.3.2 Reflection Bastiaan

At first, when I started this project, I though I was working on another academic exercise. I have been a teaching assistant for the Concepts of Programming Languages course for two years and have experience using WebLab. So naturally you think you know how it works and it cannot be such a hard thing to work on. Luckily this project turned out to be different and highly educational. This project has given me experience in working on a real-world, up and running 'work in progress'. This is not a just another exercise where all you have to worry about is if it will be done on time. Here I learnt how to deal with a project that I have no full control over, you have to use the system as much as possible to accomplish a goal; even when you disagree with some of the choices that have previously been made. That was the most frustrating part of the project, accepting the choices others have made before you and building on them even though you think it should be done differently.

The team I worked with on this project worked well together. We communicated well, worked cooperatively, and got along fine. This does not happen very often and in my experience this is one of the best groups I have worked with here at the TU Delft.

Most of the work that I put into this project is the Java support functionality. I have been programming in Java for nearly five years, starting in high school with the two Advanced Placement Computer Science courses I took. Over the last years here at the TU I have followed courses about structured programming in Java and have worked on multiple personal projects including two Android applications. All in all I am not surprised that I worked on this part of the project and I rather enjoyed it. I learnt new ways to use the language and the concepts behind the language. I got to try things I would usually never have thought of doing and I was allowed to do testing such as the Java security testing. It was a challenge, a challenge to think differently about something you know very well. The things I have learnt from this project will be an essential part of my future skill set; how to work with an existing system, how to ensure a certain level of security, why testing is so important, why documentation is needed, how to create modular software, and the list goes on.

Finally, I want to thank everyone who made this project into the great learning experience that it was, I greatly enjoyed it and learnt a lot.

### 7.3.3 Reflection Tim

When I joined the project group, I was pretty excited to be working on WebLab. As a TA I have already worked with WebLab for two years, and I really like how the system offers the students a ton of small programming exercises that are fun to work on. At our first pre-project meeting, a multitude of features were revealed that could be part of the project. After some discussion, we chose to add the Java and MySQL programming languages as our project. Jeff and I would do the MySQL part, because we had been TAs for Web and Database Technology before.
In the beginning, I was a bit taken aback by the size and complexity of the WebLab and LabBack projects. I only had experience with greenfield software projects, not with any maintenance jobs. In addition, the first week consisted of a very bumpy ride to even get the current system working on Windows. After this failed struggle, we finally managed to get the system working on an Ubuntu Linux installation. At this point, it felt like a lot of time and effort had been wasted.
On the bright side, we had the system working and in the process we had learnt a lot about the structure of the project. It proved quite doable to add new code to the system, and this is where we started picking up speed. About two thirds into the project, we lost some steam with the MySQL features due to some bugs

that kept pestering us. In the end, we managed to overcome those problems. While adding MySQL support took longer than we hoped, I am satisfied that we managed to make it work.
I learnt a lot of things in this project. Mostly I learnt how to work on an existing system. It is important to analyse it at first and to be careful that you do not disrupt the current workflow of the system when adding new parts. I also learnt quite a lot about using Git, about working with branches, and which commands to avoid or be careful with. Finally, I also learnt about using other people's open source projects in your project, what to look out for and how to compare the options that you have. So in conclusion, I am pretty satisfied with the features that we added and I think I learnt a lot of things that will be useful for future projects.

### 7.3.4 Reflection Jeff

I think this project was a success both as a software development project and as a learning experience.

The teamwork went well, and we all got along. We could all agree on the priority of the project compared to work or other courses. Everyone put an equal amount of effort into the project.

As a software development project this was an interesting project because although it was not in a company outside of the university, it was still very realistic and new for us because it was not a greenfield software engineering project. There was a system in place and it would have to keep running. Re-engineering of the system would take too much time and part was already re-engineered (LabBack 2.0) but not running yet. Part of the system would generate a database, making it hard to move things around in that code because transitioning the old data to the newly generated database model would be lots of manual labour.
So I have certainly learnt a lot from that, not to mention defining a grammar in SDF and doing some code transformation/generation with Stratego. It was also great to learn how to use topic branches in Git, split functionality to a branch with Git cherrypick, and how a pull request works.
I can now also claim to have some experience with Eclipse. Some of its features are really nice. Sadly when language support plugins are still young, the features that work so well for other languages do not work quite as well for that one. Crashes, memory leaks, and NullPointerExceptions seem to be part of the Eclipse experience when you do not use Java.

But on the whole things were great. I had fun, learnt a lot, and worked with a team to create a functionality that will be used in the future to improve on the teaching of programming courses at the university.

# 8 Conclusion

In this report we presented the current system consisting of WebLab and LabBack. The scope of this project was outlined as supporting the course Object-oriented Programming and the database part of the course Web & Database Technology.

To make the transition to WebLab as easy as possible, we analysed the current situation and documented requests from the course teachers. Our supervisors also commented on the priority of features. Not all features were implementable due to time constraints. Some of the considered features include support for:

- compiling/executing/testing the Java programming language
- compiling/executing/testing the MySQL database query language
- groups of students working on one assignment
- assignments with more than one file for the solution
- choosing assignments arbitrarily from a set of assignments
- retrying a failed assignment

These and other features have been discussed in detail. Security constraints and other preservations of the current system properties were also taken into consideration. We discussed what options and choices we had per feature request. We also provided implementation details for the features that were actually implemented and advice for any feature that we did not complete due to the limited time for this project. Next we explained how we tested various parts of the system for as much as possible; the mix of languages and architecture make it difficult to fully test the whole system. Finally we explained the process that we followed to complete this project and gave our personal opinions on how the project went. We would once again like to thank everyone involved for making this an interesting and educational experience.

# Bibliography

[] D. Clegg and R. Barker. *Case method fast-track: a RAD approach.* Addison-Wesley Longman Publishing Co., Inc., 1994.

[] R. Durstenfeld. Algorithm 235: random permutation. *Communications of the ACM*, 7(7):420, 1964.

[] S. Erdweg, P. G. Giarrusso, and T. Rendel. Language composition untangled. In *Proceedings of Workshop on Language Descriptions, Tools and Applications (LDTA)*, 2012. to appear.

[] J. Heering, P. R. Hendriks, P. Klint, and J. Rekers. The syntax definition formalism sdfreference manual. *ACM Sigplan Notices*, 24(11):43–75, 1989.

[] P. E. Naur. Revised report on the algorithmic language algol 60. *Commun. ACM*, 6(1):1–17, 1963.

[] V. A. Vergu. Labback: An extendible platform for secure and robust in-the-cloud automatic assessment of student programs. diploma thesis, TU Delft, Delft University of Technology.

# A
# Project Proposal

**Project Proposal**
BSc Technische Informatica

offered by
TU Delft EWI, department Software Engineering
Mekelweg 4
2628 CD Delft

http://swerl.tudelft.nl/
015-2787088
E.Visser@tudelft.nl

**Company description**
The TU Delft Software Engineering Research Group (SERG) aims at developing a deep understanding of how people build and evolve software systems; developing novel methods, techniques and tools that advance the way in which software is built and modified; ensuring that our research results have a lasting impact in software development practice; and offering students an education that prepares them to take a leading role in complex software development projects.

**Project description**
About a year ago, the SERG launched its digital learning environment Weblab (http://department.st.ewi.tudelft.nl/weblab/), using LabBack, a master thesis project of Vlad Vergu. Weblab gives the students access to a cloud programming interface in their web browser, allowing them to make tutorials, assignments and even exams without having to install or configure any software on their own computers. LabBack supports automatic assessment of programs, which provides possibilities for larger amounts of exercises without significantly increasing grading workload.
One of the courses that is supported by this environment is the first year CS course Concepts of Programming Languages. In the future, the plan is to expand Weblab in a lot of areas to make it the ultimate online learning environment. The goal of this project is to make Weblab support more of the first-year CS courses.
Specifically, the students will develop support for the courses Object-Oriented Programming and (the database part of) Web and Database Technology. Furthermore, the Weblab system needs to be improved generally.
In order to realise this, the following steps have to be taken.

**Java (Object-Oriented Programming)**

Must-have requirements
- Gather specific requirements from Andy Zaidman, teacher of the Object Oriented programming course.

- Develop basic support for Java code:
  - Syntax highlighting
  - Java code compilation
- Develop support for the testing of Java code.

<u>Could-have requirements</u>

- Develop support for secure file I/O.
- Develop support for threading.
- Add the possibility to pause and run threads.
- Add functionality to use System.in.

**SQL (Web & Database Technology)**

<u>Must-have requirements</u>

- Gather specific requirements from Jan Hidders, teacher of the Web&Database course.
- Develop support for queries using SELECT and JOIN.
- Develop support for queries using UPDATE, INSERT and DELETE, where the database will be restored after a session.
- Add (unit) testing functionality.
- Display useful error messages for queries.

<u>Could-have requirements</u>

- Develop support for queries using VIEWs.

<u>Would-have requirements</u>

- Develop support for queries using CREATE and DROP.

**General features**

<u>Must-have requirements</u>

- Gather specific requirements from Eelco Visser and Vlad Vergu.
- Develop support for using multiple files in one programming assignment.
- Develop support for giving each user a randomized subset of questions.

<u>Would-have requirements</u>

- Add adaptive learning functionality to weblab.

**Auxiliary Information**

This project is supposed to follow an iterative workflow. Weekly meetings will be planned with the supervisors. The research report may grow with each iteration.

**Project team**

Supervision

| | | |
|---|---|---|
| Name Company Supervisor | Email | Phone |
| Eelco Visser | E.Visser@tudelft.nl | 015-2787088 |
| Vlad Vergu | V.A.Vergu@tudelft.nl | 06-40227013 |
| Name TUDelft Supervisor | Email | Phone |
| Andy Zaidman | A.E.Zaidman@tudelft.nl | 015-2784385 |

Student team

| Name | Study ID | Email | Phone | Prerequisites* fulfilled signature |
|---|---|---|---|---|
| Jeff Smits | 4104889 | J.Smits-1@student.tudelft.nl | 06-19658043 | [illegible] |
| Bastiaan Reijm | 4062973 | reijm.bastiaan@gmail.com | 06-53373460 | [illegible] |
| Tim de Jong | 4081544 | T.K.deJong@student.tudelft.nl | 06-44427903 | [illegible] |
| Marieke v/d Tuin | 4079299 | msvandertuin@gmail.com | 06-31051029 | [illegible] |

*Prerequisites: by signing this box, as a student, I declare that I fulfill the prerequisites of the TI3800 bachelor project as stated in the digital study guide.

# B SIG Feedback

The Software Improvement Group (SIG) analyses the code written by students during the Bachelor Project. It is analysed twice, once during the project, so the students can use the feedback; and once at the end of the project.

The feedback itself is in Dutch.

## B.1 First Analysis

*Code submitted on the 14th of June 2013*
*Feedback received on the 21st of June 2013*

[Analyse]
Jullie project bestaat zo te zien uit een Scala-backend en een WebDSL-frontend. Het zal jullie niet verbazen dat de SIG hgeen automatische analyse-tooling voor WebDSL heeft, dus onderstaande opmerking gaan vooral over het Scala-gedeelte. Om het nog eens extra ingewikkeld te maken hebben jullie bestaande bestanden aangepast. Het is voor ons niet mogelijk om die aanpassingen apart te beoordelen (aangezien we niet delen van bestanden apart kunnen beoordelen), dus we hebben naar het geheel gekeken.
De Scala-code scoort 3 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code gemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere scores voor Unit Size en Unit Complexity.
Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in jullie project zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zijn vaak een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen. Een goed voorbeeld van zo'n methode is extractTests in SQLTestHandler.scala.
Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te

begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen. Een voorbeeld is run() in JavaExecutionThread.scala. Jullie gebruiken hier wederom commentaar om stukjes aan te geven, terwijl je er vrij makkelijker aparre methodes van kunt maken.
De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

## B.2 Second Analysis

*Code submitted on the 3rd of July 2013*
*Feedback received on the 4th of July 2013*

[Hermeting]
In de tweede upload zien we dat zowel het codevolume als de score voor onderhoudbaarheid zijn gestegen. De Scala-code scoort met 4 sterren nu bovengemiddeld, dat was in de eerste upload nog 3 sterren. De grootste stijging in deelscores zit in Unit Size en Unit Complexity, de gebieden die in de vorige analyse als verbeterpunt werden aangemerkt.
Het is goed om te zien dat naast een verbetering in de onderhoudbaarheid er ook nog steeds een stijging in het volume van de test-code te zien is.
Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.

# C Deployment Strategy

This document contains the instructions for updating the existing WebLab/LabBack system. Note that these instructions were only tested to work on Ubuntu 12.04.

In §C.1 the building and running of the current system is described first, because this isn't documented anywhere else yet.
Then the installation of the extra MySQL server is explained, followed by building instructions for the MySQL parser. Then the manual steps for updating the WebLab database are explained.
This document is concluded with some text on running the tests.

## C.1 Building and running the current system

This section describes how to build and run the main system, both WebLab and LabBack. The particular commands have been tested on Ubuntu, but should be similar for other operating systems.

### C.1.1 Basic Build and Run

These are the basic build and run instructions for each system.

#### LabBack - The Back End

**Step 1**:
Execute the following command set:

```
cd <labback>/WebAppSrv
ant jar-frontend
```

**Step 2**:
Copy the `<labback>/WebAppSrv/dist/execdaemon-frontend.jar` file to the `<weblab>/lib` directory.

**Step 3**:
Execute the following command set:

**Note that this is not a secure run configuration, see §C.1.2 for that.**

```
cd <labback>/WebAppSrv
ant run-backend
```

#### WebLab - The Front End

```
cd <weblab>
webdsl build deploy
```

from a seperate terminal:

```
sudo service tomcat7 start
```

### C.1.2 Alternative Build and Run Procedures

Besides the basic build and run instructions there are also several alternatives.

#### Clean Back End

To clean the back end the following command set should be executed:

```
cd <labback>/WebAppSrv
ant clean
```

Afterwards the build and run process can be executed.

#### Clean Front End

To clean the front end the following command set should be executed:

```
cd <weblab>
webdsl clean
```

#### Run without Building

To run the front end without rebuilding it, run the LabBack normally and restart tomcat with the following command.

```
sudo service tomcat7 restart
```

This will start up Tomcat with the last built `.war` file.

#### Running LabBack Securely

To run the back end in secure mode, run and build WebLab normally and execute the following command set:

```
cd <labback>/WebAppSrv
ant run-backend-secure
```

#### Using the WebDSL Tomcat Installation

To run the front end on a Tomcat server that was distributed with WebDSL instead of a separate installation, build and run the back end normally and execute the following command:

```
webdsl run
```

## C.2 Integration of the new features

In order to integrate the new features with the current system, a few steps have to be taken. These steps are explained below.

### C.2.1 MySQL server

To support MySQL execution and testing in LabBack, an extra MySQL server has to be installed. This is done to isolate the execution and testing of the foreign database code from the MySQL server which is already used to support WebLab.
(The diff files shown in this section are also available in the attachments of this PDF.)

#### Server configuration

The extra MySQL needs to be configured to use a different port, socket, pid, data directory, binary log and optionally error log. The changes for this configuration were made to the file `/etc/mysql/my.cnf` and are given in Listing C.1.

#### AppArmor permission configuration

The directories to be used by the extra MySQL server have extra permission settings. The configuration file `/etc/apparmor.d/usr.sbin.mysqld` needs to be altered for this. The changes are given in Listing C.2.

#### Server data-directory initialisation

The server now has access to everything it needs, but the data directory still needs to be initialised. There are a few commands required to set this up, which can be found in Listing C.3.

#### Server start/stop and connection

The server can now be started with command `sudo mysql_multi start 1`. Similarly the server can be stopped with `sudo mysql_multi stop 1`.

You can connect to the server from the command line using using option `-S socketfile`. For example: `mysql -u root -S /var/run/mysqld/mysqld1.sock`.

### C.2.2 MySQL parser

We extended an old project containing a partial SQL grammar as a Spoofax project. From this project we made a parser and a transformer to easily execute MySQL code without most safety issues like deadlock, memory overflow and accessing other students views.

The parser is used as a JAR file in LabBack.

```
diff --git a/my.cnf b/my.cnf
index 0f16d34..aa3afbd 100644
--- a/my.cnf
+++ b/my.cnf
@@ -107,6 +107,21 @@ max_binlog_size         = 100M
 # ssl-cert=/etc/mysql/server-cert.pem
 # ssl-key=/etc/mysql/server-key.pem

+[mysqld_multi]
+mysqld          = /usr/bin/mysqld_safe
+mysqladmin      = /usr/bin/mysqladmin
+user            = mysql
+
+[mysqld1]
+server-id       = 1
+pid-file        = /var/run/mysqld/mysqld1.pid
+socket          = /var/run/mysqld/mysqld1.sock
+port            = 3307
+datadir         = /var/lib/mysql1
+log-bin         = /var/lib/mysql1/mysqld1-bin.log
+log-bin-index   = /var/lib/mysql1/mysqld1-bin.index
+log-error       = /var/log/mysql1/error.log
+
 [mysqldump]
 quick
 quote-names
```
Listing C.1: The changes to the MySQL configuration file as given by git diff.

```
diff --git a/usr.sbin.mysqld b/usr.sbin.mysqld
index 0f06349..02a03d6 100644
--- a/usr.sbin.mysqld
+++ b/usr.sbin.mysqld
@@ -29,10 +29,25 @@
   /usr/share/mysql/** r,
   /var/log/mysql.log rw,
   /var/log/mysql.err rw,
+  /var/log/mysql[1-9].log rw,
+  /var/log/mysql[1-9].err rw,
+  /var/lib/mysql/ r,
   /var/lib/mysql/** rwk,
+  /var/lib/mysql[1-9]/ r,
+  /var/lib/mysql[1-9]/** rwk,
+  /var/log/mysql/ r,
   /var/log/mysql/* rw,
+  /var/log/mysql[1-9]/ r,
+  /var/log/mysql[1-9]/* rw,
+  /var/run/mysqld/mysqld.pid w,
   /var/run/mysqld/mysqld.sock w,
+  /var/run/mysqld/mysqld[1-9].pid w,
+  /var/run/mysqld/mysqld[1-9].sock w,
+  /run/mysqld/mysqld.pid w,
   /run/mysqld/mysqld.sock w,
+  /run/mysqld/mysqld[1-9].pid w,
+  /run/mysqld/mysqld[1-9].sock w,
+
   /sys/devices/system/cpu/ r,

   # Site-specific additions and overrides. See local/README
        for details.
```

Listing C.2: The changes to the AppArmor mysqld configuration file as given by git diff.

```
cd /var/lib/
sudo mkdir mysql1
sudo chmod 700 mysql1
sudo chown mysql:mysql mysql1
sudo mysql_install_db --user=mysql --datadir=/var/lib/mysql1
     --basedir=/usr
```

Listing C.3: The commands to initialise the data directory for the extra MySQL server.

**Building**

To build the parser, use git to checkout the project from github[1]. Import it into Eclipse with the Spoofax plugin installed. Build the project.

Then use the bash script in the root directory of the project, named `build-jar`. This builds the `sql2clean.jar` file in the same directory, using the `sql2clean/` directory for temporary stuff.

Now copy this `sql2clean.jar` file to the `lib/` folder in the LabBack project (`<labback>/WebAppSrv/lib/`).

### C.2.3 Updating the WebLab database

During development, we have been careful with our changes to the existing WebLab entities, because we did not want to invalidate the current WebLab database. The result is that the old database can still be used with the new features, if a small change is made.

For a property that has been added to a existing entity, the value in the old database will be `NULL`. For most new properties, it is necessary to replace this `NULL` value with the default value of the property. This can be done by running the queries shown in Listing C.4 on the WebLab database with a user that has `UPDATE` rights[2].

```sql
-- Updating the ProgramRun entity with the correct defaults
UPDATE _ProgramRun SET _dbname='';

-- Updating the Assignment entity with the correct defaults
UPDATE _Assignment SET _group = 0 WHERE _group IS NULL;
UPDATE _Assignment SET _maxgroupsize = 3 WHERE _maxgroupsize IS NULL;
UPDATE _Assignment SET _mingroupsize = 1 WHERE _mingroupsize IS NULL;
UPDATE _Assignment SET _numberofgroups = 0 WHERE _numberofgroups IS NULL;
UPDATE _Assignment SET _randomgroups = 0 WHERE _randomgroups IS NULL;
UPDATE _Assignment SET _studentgroupcreation = 1 WHERE
    _studentgroupcreation IS NULL;
UPDATE _Assignment SET _random = 0 WHERE _random IS NULL;
UPDATE _Assignment SET _random_assignmentscount = 0 WHERE
    _random_assignmentscount IS NULL;

-- Updating the CourseEdition entity with the correct defaults
UPDATE _CourseEdition SET _maxgroupsize = 3 WHERE _maxgroupsize IS NULL;
UPDATE _CourseEdition SET _mingroupsize = 1 WHERE _mingroupsize IS NULL;
```

Listing C.4: Update queries for the WebLab database.

The result should be that all rows in the `_ProgramRun` table in the WebLab database have their `_dbname` fields set to an empty string (this can appear as an empty field) instead of `NULL`.

[1] https://github.com/tkdejong/sql-front
[2] also attached to the PDF as `update-weblab.sql`

## C.3 Running the tests

After following all of the previous steps, you should now be able to build Weblab and Labback. In order to verify that the newly added features work correctly, you can simply run the tests. We have added different kinds of test. First, the WebDSL tests are explained. Then, we show how to work with our tests written for ScalaTest.

### C.3.1 WebDSL tests

We used WebDSL tests to test the file management system used for the Project Assignment, the group support and the random assignments. The three test files are located in:

1. `tests/file-management-test.app`
2. `tests/studentgroup-tests.app`
3. `tests/random-assignment-collection-tests.app`

The tests can be run using the command `webdsl check` if you want to use your current settings in the application.ini file to test. If you use the command `webdsl test weblab`, a new application.ini file with a separate Sqlite database file will be created, as can be read in the WebDSL testing manual[3].

While running the test you may note that several errors will pop up between the testing. This is due to the cleaning up of the database that does not happen correctly, as explained the report (section 6.3).

### C.3.2 ScalaTest

In order to test the new features in Labback, we have added some ScalaTest[4] test suites to the project. The test classes are located in the packages contained in `<labback>/WebAppSrv/test`. The easiest way to run them is to through the ScalaTest Eclipse plugin[5].
Alternatively, you can run them from the command line. This requires an installation of the Scala compiler. The steps to running the tests in this way re-explained in the ScalaTest userguide[6].

[3]http://webdsl.org/selectpage/Manual/Testing
[4]www.scalatest.org
[5]http://www.scalatest.org/user_guide/using_scalatest_with_eclipse
[6]http://www.scalatest.org/user_guide/using_the_runner

# D Java Security Tests

These are the Java Security Tests that have been executed via WebLab to test for security leaks and vunerabilities.

## D.1 JVM Forced Shutdown

The Security Manager prevents the termination of the JVM and throws an exception.

```java
public class Solution {
  public void call() {
    System.exit(0);
  }
}
```

## D.2 Threads

The Class Loader does not load the Thread class because it is blacklisted.

```java
public class Solution {
  public void call() throws InterruptedException {
    Thread.sleep(10000);
  }
}
```

## D.3 File I/O

These tests were designed to test if a user can manipulate the specification tests using File I/O operations.

### D.3.1 Test Reading

The Security Manager blocks this and throws an exception.

```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class Solution {
  public void call() {
    try {
      FileReader fr = new FileReader(new File("UTest.java"));
      BufferedReader br = new BufferedReader(fr);
      StringBuilder sb = new StringBuilder();

      String s;
      while((s = br.readLine()) != null) {
        sb.append(s);
      }

      br.close();

      String ans = sb.toString();
      System.out.println(ans);
      System.err.println(ans);
    }
    catch(IOException e) {
    }
  }
}
```

### D.3.2 Retrieving the System Parent

The Security Manager blocks this and throws an exception.

```java
import java.io.File;

public class Solution {
  public void call() {
    File file = new File("UTest.java");

    if (file.exists()) {
      File parent = file.getParentFile();
      File doubleParent = parent.getParentFile();

      for (File f : doubleParent.listFiles()) {
        System.out.println(f);
        System.err.println(f);
      }
    }
  }
}
```

### D.3.3 Overwriting the Tests

The Security Manager blocks this and throws an exception.

```
public class Solution {
  public void call() {
    try {
      FileWriter fw = new FileWriter(new File("UTest.java"));
      BufferedWriter bw = new BufferedWriter(fw);

      String[] content = new String[] {
        "////<UTest>",
        "",
        "import org.junit.*;",
        "import static org.junit.Assert.*;",
        "",
        "public class UTest",
        "{",
        "  private Solution sol;",
        "",
        "  @Before",
        "  public void setup()",
        "  {",
        "    this.sol = new Solution();",
        "  }",
        "",
        "  @Test",
        "  public void test()",
        "  {",
        "     assertEquals(\"testing\", \"TeStInG\".toLowerCase());",
        "  }",
        "}"
      };

      for (String s : content) {
        bw.write(s + "\r\n");
      }

      bw.close();
    }
    catch(IOException e) {
    }
  }
}
```

### D.3.4 Deleting the Tests

The Security Manager blocks this and throws an exception.

```
import java.io.File;

public class Solution {
  public void call() {
    File file = new File("UTest.java");

    if (file.exists() && !file.delete()) {
      file.deleteOnExit();
    }
  }
}
```

### D.3.5 List the Root Files

No files are listed because only files that the Security Manager approves are listed.

```java
import java.io.File;

public class Solution {
  public void call() {
    for (File f : File.listRoots()) {
      System.out.println(f);
      System.err.println(f);
    }
  }
}
```

## D.4 Sockets

These tests were designed to test if students can use sockets to communicate via WebLab.

### D.4.1 Server Socket

The Class Loader does not load the ServerSocket class because it has been blacklisted.

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.PrintWriter;

import java.net.InetAddress;
import java.net.Socket;
import java.net.ServerSocket;

public class Solution {
  public void call() {
    int port = 4568;
    ServerSocket serverSocket = null;
    try {
      serverSocket = new ServerSocket(port, 0, InetAddress.getLocalHost());
    }
    catch (IOException ioe) {
      System.err.println("Could not listen on port: " + port);
      return;
    }

    Socket clientSocket = null;
    try {
      clientSocket = serverSocket.accept();
    }
    catch(IOException e) {
      System.err.println("Accept Failed");
      return;
    }

    try {
      PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
      BufferedReader in = new BufferedReader(new InputStreamReader(
                          clientSocket.getInputStream()));

      out.println("What is the answer to question 3?");

      String answer = in.readLine();
      out.close();
      in.close();
      clientSocket.close();
      serverSocket.close();

      System.out.println(answer);
    }
    catch (IOException ioe) {
      return;
    }
  }
}
```

### D.4.2 Client Socket

The Security Manager disallows the creation of a Socket and throws an exception.

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.PrintWriter;

import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Solution {
  public void call() {
    Socket socket = null;
    PrintWriter out = null;
    BufferedReader in = null;

    try {
      socket = new Socket(InetAddress.getLocalHost(), 4569);
      out = new PrintWriter(socket.getOutputStream(), true);
      in = new BufferedReader(
            new InputStreamReader(
              socket.getInputStream()));
    }
    catch (UnknownHostException e) {
      System.err.println("Connection Failed : Unknown Host");
      return;
    }
    catch (IOException ioe) {
      System.err.println("Connection Failed : I/O failed");
      return;
    }

    try {
      process(in.readLine());
      out.close();
      in.close();
      socket.close();
    }
    catch(IOException ioe) {
      return;
    }
  }

  private String process(String question) {
    System.out.println(question);

    Matcher matcher = Pattern.compile(
      "What is the answer to question (\\d)\\?").matcher(question);

    if (matcher.matches() && matcher.group(1).equals("3")) {
      return "c";
    } else {
      return "I don't know, give me 5 minutes";
    }
  }
}
```

## D.5 Reflection

The Security Manager disallows any editing via Reflection and the lack of system output stops anyone from seeing the test names.

```java
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class Solution {
   public int f(int n) {
     reflect();
     return n+1;
   }

   public void reflect() {
     try {
       Class<?> c = Class.forName("UTest");

       System.out.println("Name: " + c.getName());
       System.out.println("Annotation: " + c.isAnnotation());
       System.out.println("Array: " + c.isArray());
       System.out.println("Enum: " + c.isEnum());

       Field[] fields = c.getDeclaredFields();

       System.out.println("Fields");
       for (Field f : fields) {
         System.out.println(f.getName());
       }

       Field f = c.getDeclaredField("sol");
       f.setAccessible(true);
       f.set(c, null);

       Method[] methods = c.getDeclaredMethods();

       System.out.println("Methods");
       for (Method m : methods) {
         System.out.println(m.getName());
       }
     }
     catch(Exception e) {
       e.printStackTrace();
     }
   }
}
```

# E WebDSL Testing

This is an example of a clean way to do Unit Testing in WebDSL.

## E.1 Testing Example

The general way to make a unit test in WebDSL is to setup the page, define the entities, define the setup functions, define a clean function, and finally define the tests.

### E.1.1 Page Setup

This required if the tests are run from the main app file.

```
application TestingExample

define page root(){}
```

### E.1.2 Entities

Three entities *A*, *B*, and *C* are defined such that *B* is a subclass of *A*. Every *A* contains a *C* and a list of *C*s.

```
entity A {
	name :: String
	list -> List<C>
	c    -> C
}

entity B : A {}

entity C {
	name :: String
	n    :: Int
	a    -> A
}
```

### E.1.3 Test Setup Functions

These functions are used to reduce code clutter in the actual test methods.

```
//fill an A with default values
function fillA(a: A) {
        var ref := C{name := a.name, n := 0, a := a};
        a.c := ref;

        var list : List<C> := List<C>();
        list.add(ref);

        a.list := list;
}

// make a new A
function makeA(name: String): A {
        var res := A{name := name};
        fillA(res);
        res.save();
        return res;
}

// make a new B
function makeB(name: String): B {
        var res := B{name := name};
        fillA(res);
        res.save();
        return res;
}
```

### E.1.4 Test Cleanup

This function cleans the database before the system itself tries to clean the database. All references to entities are set to null and all containers such as sets and lists are cleared. Note: Do not set the reference of a set or list to null.

```
// clean the database
function clean() {
        for (a: A)
        {
                var c := a.c;
                a.c := null;
                a.list.clear();
                c.delete();
                a.delete();
        }
}
```

### E.1.5 Tests

These are two tests. First the test entities are created, then some assertions are executed, and finally the environment is cleaned.

```
test test0 {
        var a0 := makeA("a0");
        var a1 := makeA("a1");

        assert(a0.c.name == "a0");
        assert(a1.c.name == "a1");
        assert(a0 != a1);

        clean();
}

test test1 {
        var a0 := makeA("a0");
        var ab := makeA("b0");
        var b0 := makeB("b0");

        assert(a0.c.name == "a0");
        assert(ab.c.name == b0.name);
        assert(a0 != ab);
        assert(a0 != b0);
        assert(ab != b0);

        clean();
}
```