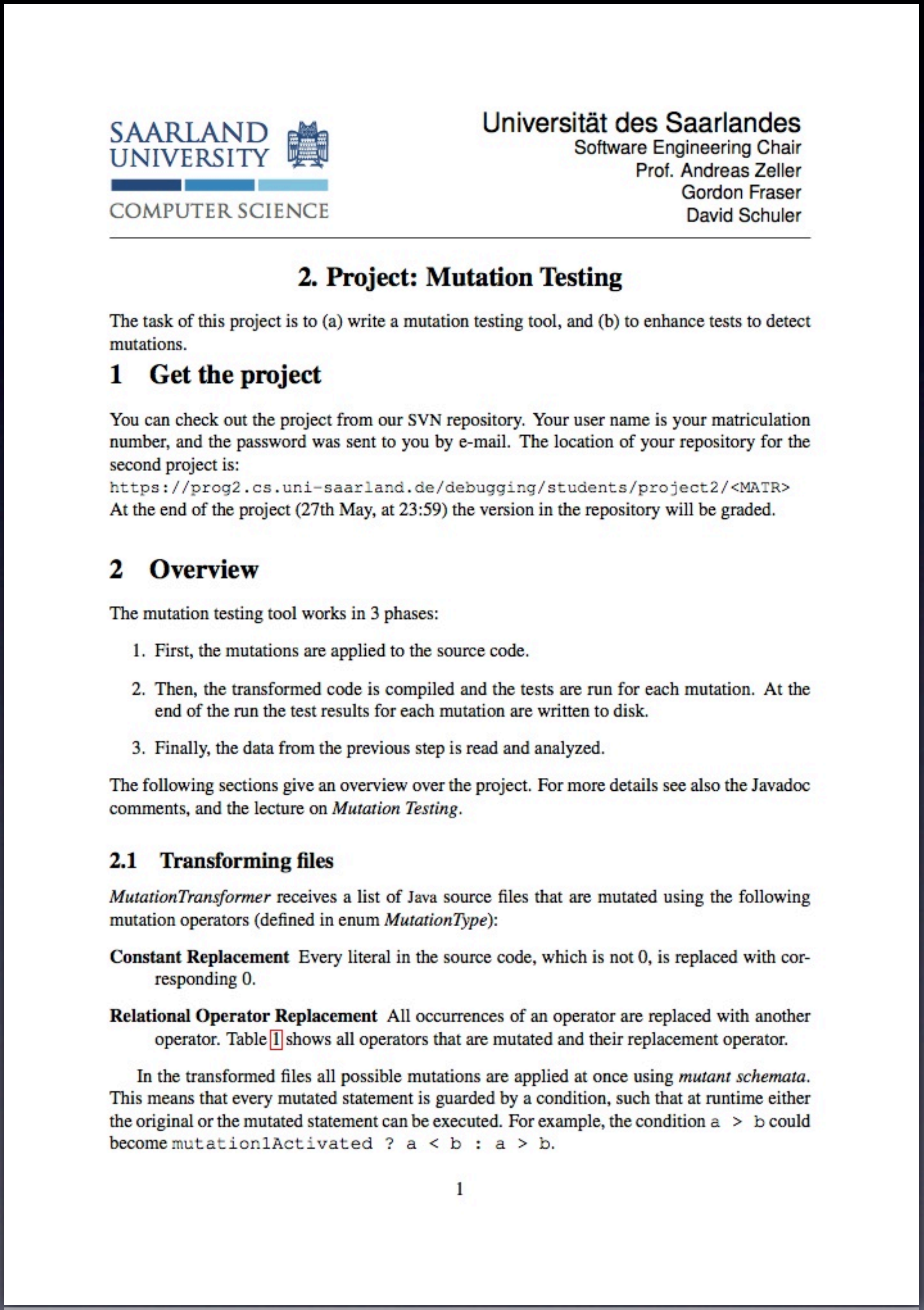
Testing and Debugging Project 2: Mutation Testing

Mutation Testing

- Project will start next week.
- Presentation today because of holiday.
- Project will be available from SVN next week thursday.



SAARLAND UNIVERSITY
COMPUTER SCIENCE

Universität des Saarlandes
Software Engineering Chair
Prof. Andreas Zeller
Gordon Fraser
David Schuler

2. Project: Mutation Testing

The task of this project is to (a) write a mutation testing tool, and (b) to enhance tests to detect mutations.

1 Get the project

You can check out the project from our SVN repository. Your user name is your matriculation number, and the password was sent to you by e-mail. The location of your repository for the second project is:
`https://prog2.cs.uni-saarland.de/debugging/students/project2/<MATR>`
At the end of the project (27th May, at 23:59) the version in the repository will be graded.

2 Overview

The mutation testing tool works in 3 phases:

1. First, the mutations are applied to the source code.
2. Then, the transformed code is compiled and the tests are run for each mutation. At the end of the run the test results for each mutation are written to disk.
3. Finally, the data from the previous step is read and analyzed.

The following sections give an overview over the project. For more details see also the Javadoc comments, and the lecture on *Mutation Testing*.

2.1 Transforming files

MutationTransformer receives a list of Java source files that are mutated using the following mutation operators (defined in enum *MutationType*):

Constant Replacement Every literal in the source code, which is not 0, is replaced with corresponding 0.

Relational Operator Replacement All occurrences of an operator are replaced with another operator. Table 1 shows all operators that are mutated and their replacement operator.

In the transformed files all possible mutations are applied at once using *mutant schemata*. This means that every mutated statement is guarded by a condition, such that at runtime either the original or the mutated statement can be executed. For example, the condition `a > b` could become `mutation1Activated ? a < b : a > b`.

1

3 Phases

- Transform source files.
- Compile the transformed files.
- Run tests for all mutations.

Mutation Operators

- Relational Operator Replacement (ROR)

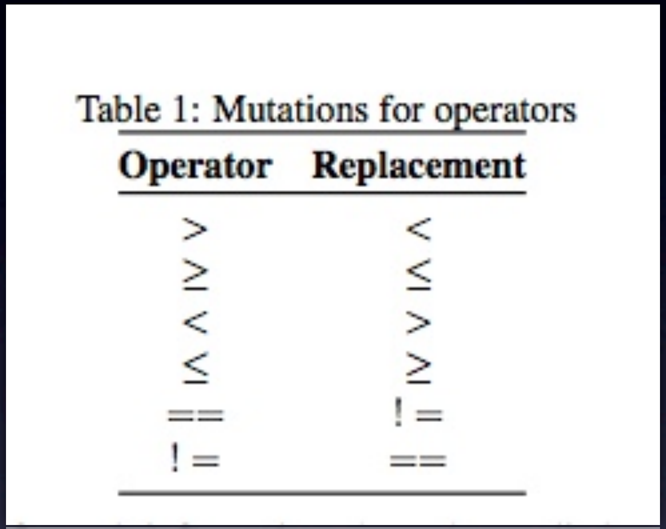


Table 1: Mutations for operators

Operator	Replacement
>	<
≥	≤
<	>
≤	≥
==	! =
! =	==

- Constant Replacement (CRP)

Assumption

- Use mutant schemata
- Only one mutation per operator/constant
- No higher order mutants required
- =>Makes implementation easier

Example

```
if( (a > b) || ( x >= 5)
```

```
if( (a < b) || ( x >= 5)
```

```
if( (a > b) || ( x <= 5)
```

```
if( (a > b) || ( x >= 0)
```

Example

```
if( (a > b) || ( x >= 5)
```

```
if( m1 ? (a < b) :(a > b) ||
    m2 ? (x <= 5) : m3 ? (x >= 0) : (x >= 5)
```

Type of Variables

```
int i = 1;
double x  = ( i + 1. ) / 2
```

```
int i = 1;
double x  = ( i + 0 ) / 2
```

```
int i = 1;
double x  = ( i + 0.) / 2
```

Compiling

- Mutated source files are compiled by our test script.
- *ant check-mutation*

Executing Mutations

- Check if none of the tests fails.
- Run all tests on every mutant.
- Record the results.

Detecting Mutations

- Extend test cases from first project for Variance class such that it detects all detectable mutants.
- Mutation in *serialVersionUID* not expected to be detectable.

Project 1 - Fault localization

- You do not need to connect Classes to tests.
- Just take all coverage data that is in the specified directory.

Utility Methods

- *org.apache.commons.io.FileUtils*, provides utility methods to read write files.
- Collections can be converted to arrays using the *toArray()* method.
- *org.apache.commons.lang.ArrayUtils* provides utility methods to convert arrays, e.g. *Integer[]* to *int[]*.

Static Initializer

```
public class A {
  static {
    //set up code
  }
...
}
```

- Called once when the class is loaded.
- Java Language Specification:

  8.7 Static Initializers

  Any static initializers declared in a class are executed when the class is initialized and, together with any field initializers (§8.3.2) for class variables, may be used to initialize the class variables of the class (§12.4).

Secret Tests

- Some constructs will not be in the secret tests:
  - synchronized blocks
  - assert statements
  - empty statement (;)
  - labels
  - class declarations inside methods (e.g. *class SomeClass* {};