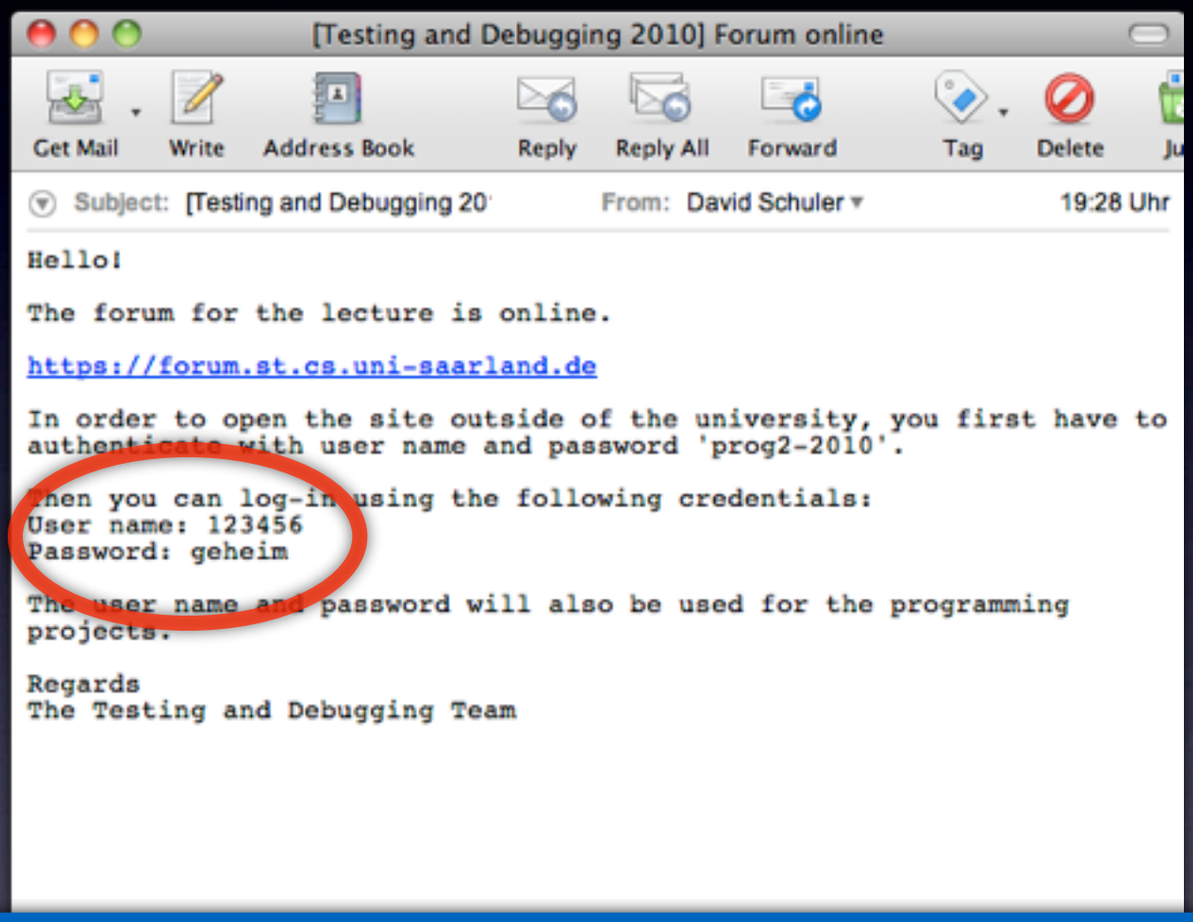
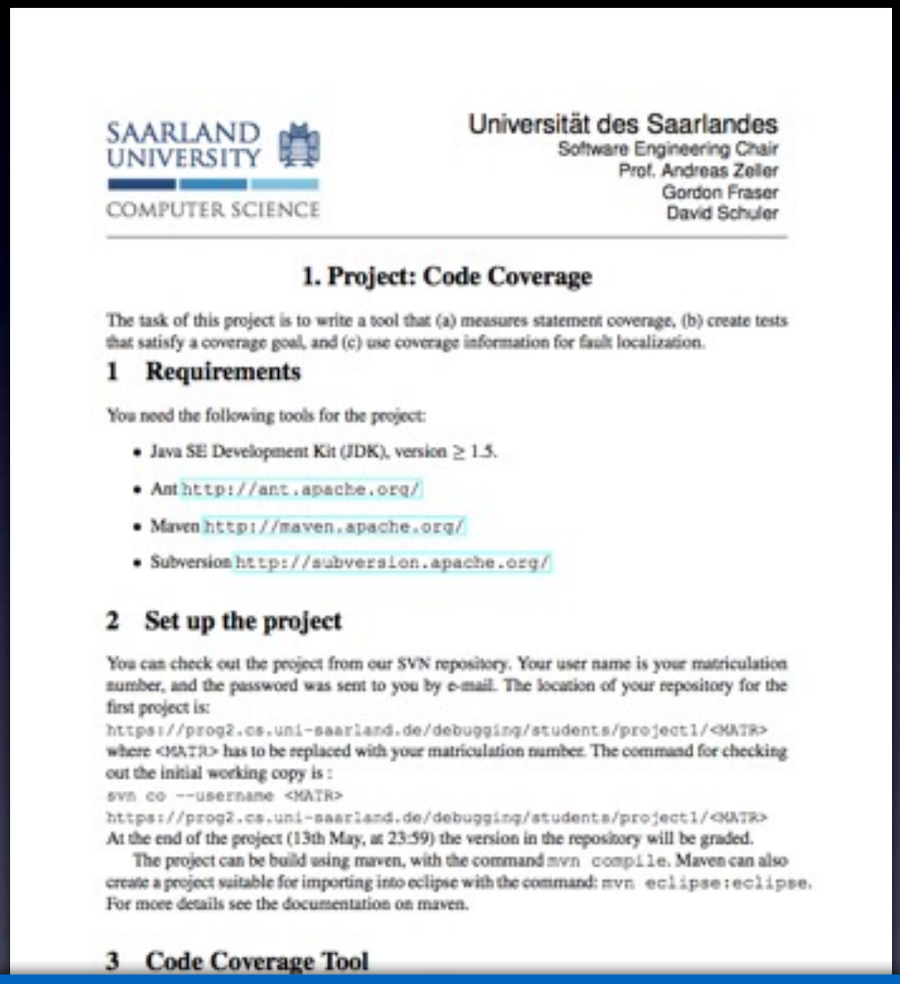
Testing and Debugging
Project 1:
Code Coverage

Projects




SAARLAND UNIVERSITY
COMPUTER SCIENCE

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

1. Project: Code Coverage

The task of this project is to write a tool that (a) measures statement coverage, (b) create tests that satisfy a coverage goal, and (c) use coverage information for fault localization.

1 Requirements

You need the following tools for the project:

- Java SE Development Kit (JDK), version ≥ 1.5.
- Ant `http://ant.apache.org/`
- Maven `http://maven.apache.org/`
- Subversion `http://subversion.apache.org/`

2 Set up 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 first project is:
`https://prog2.cs.uni-saarland.de/debugging/students/project1/<MATR>`
where `<MATR>` has to be replaced with your matriculation number. The command for checking out the initial working copy is:
`svn co --username <MATR>`
`https://prog2.cs.uni-saarland.de/debugging/students/project1/<MATR>`
At the end of the project (13th May, at 23:59) the version in the repository will be graded.

The project can be build using maven, with the command `mvn compile`. Maven can also create a project suitable for importing into eclipse with the command: `mvn eclipse:eclipse`. For more details see the documentation on maven.

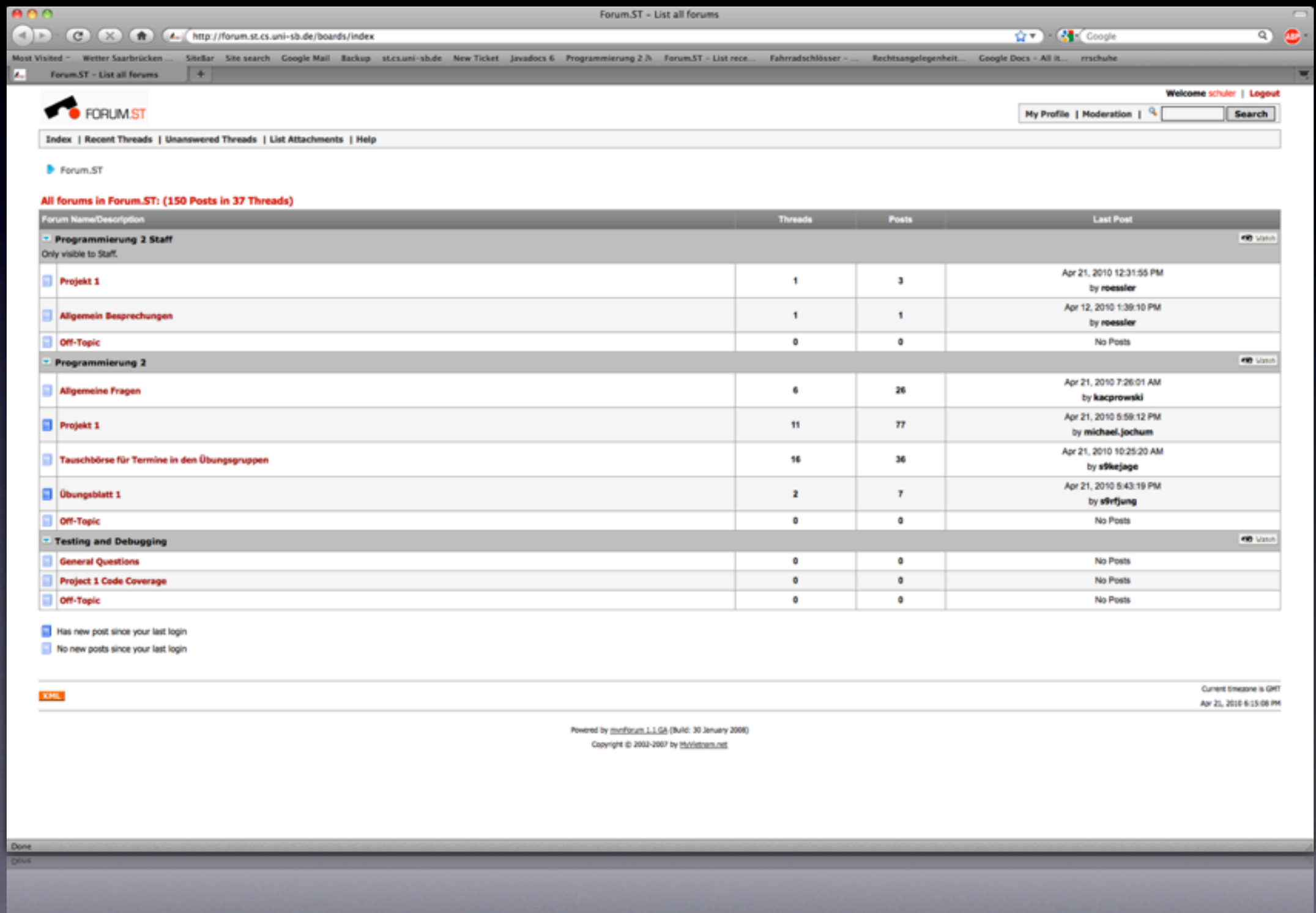
3 Code Coverage Tool

http://www.st.cs.uni-saarland.de/edu/testingdebugging10/

Public Project

- If you are not yet registered, you should register and meanwhile you can use the project:
- User name: *1234567*
- Password: *tad10*

Forum


Forum.ST - List all forums
http://forum.st.cs.uni-sb.de/boards/index
Welcome schuler | Logout
FORUM.ST
My Profile | Moderation |
Search
Index | Recent Threads | Unanswered Threads | List Attachments | Help
Forum.ST
All forums in Forum.ST: (150 Posts in 37 Threads)
Forum Name/Description
Threads
Posts
Last Post
Programmierung 2 Staff
Only visible to Staff.
Projekt 1 1 3 Apr 21, 2010 12:31:55 PM by roessler
Allgemein Besprechungen 1 1 Apr 12, 2010 1:39:10 PM by roessler
Off-Topic 0 0 No Posts
Programmierung 2
Allgemeine Fragen 6 26 Apr 21, 2010 7:26:01 AM by kacprowski
Projekt 1 11 77 Apr 21, 2010 5:59:12 PM by michael.jochum
Tauschbörse für Termine in den Übungsgruppen 16 36 Apr 21, 2010 10:25:20 AM by s9kejage
Übungsblatt 1 2 7 Apr 21, 2010 5:43:19 PM by s9rfjung
Off-Topic 0 0 No Posts
Testing and Debugging
General Questions 0 0 No Posts
Project 1 Code Coverage 0 0 No Posts
Off-Topic 0 0 No Posts
Has new post since your last login
No new posts since your last login
XML
Current timezone is GMT
Apr 21, 2010 6:15:08 PM
Powered by myvBforum 1.1 GA (Build: 30 January 2008)
Copyright © 2002-2007 by MyVietnam.net
Done

Forum

- Questions can be asked on the forum: https://forum.st.cs.uni-sb.de/boards/index
- Password and user name outside the university: prog2-2010
- Your personal user name and password is the same as for the project.

Task

(a) Write a tool that measures statement coverage.

(b) Create tests that satisfy a coverage goal.

(c) Implement the Tarantula fault localization technique.

Set up the project

- Check out the project from the given URL
- Use Maven to build it:
  - *mvn compile*
  - *mvn assembly:assembly*
  - *mvn eclipse:eclipse*

(a) Statement coverage

- We recommend Eclipse AST (Abstract Syntax Tree) to instrument the source code.
  - AST uses visitor pattern - visit and endVisit methods.
  - Instrumentation should log the covered statements and write the data to disk.
  - See lecture: Advanced Coverage Criteria

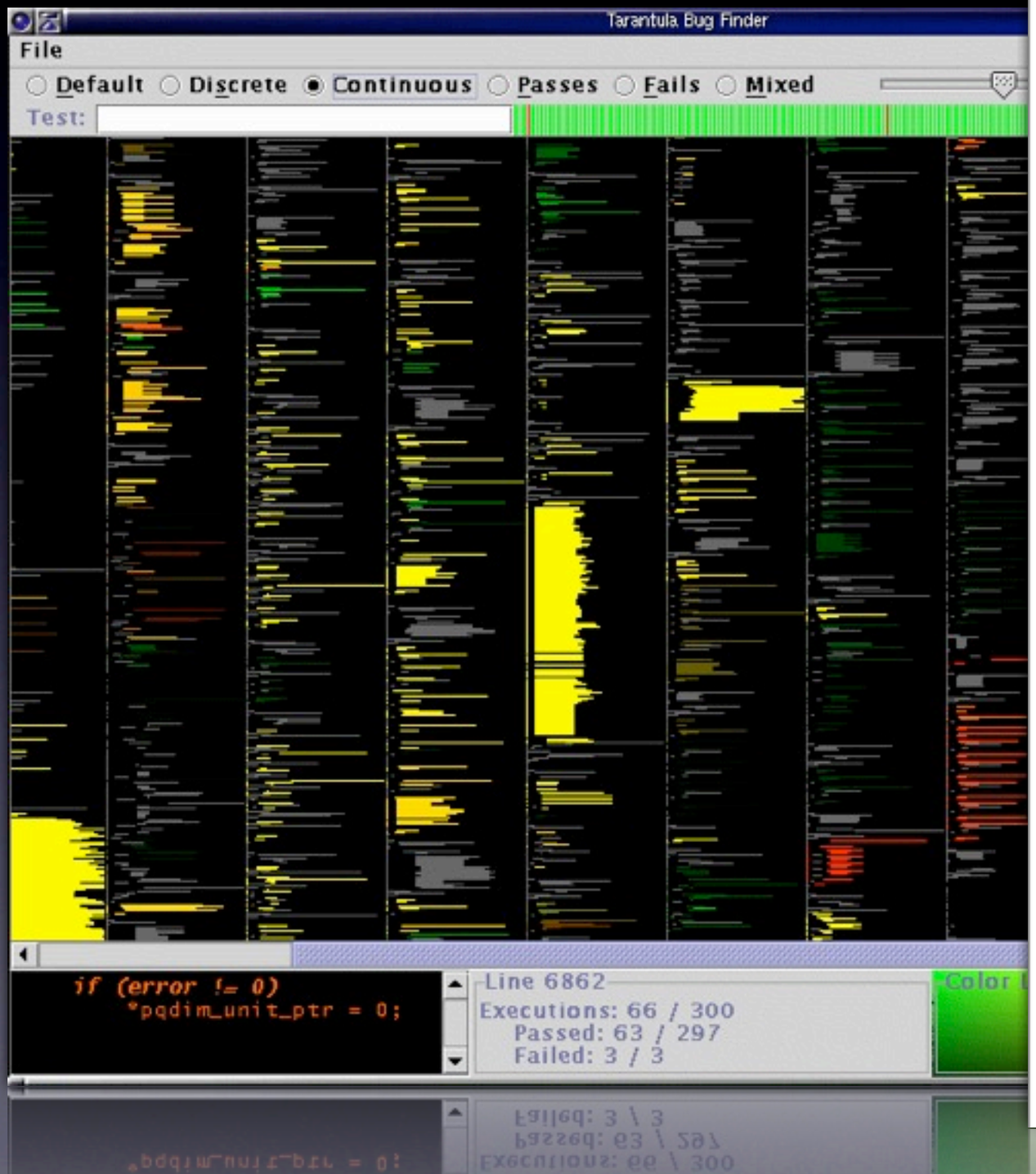
Running Tests

- *ant check-coverage* runs integration tests.
- Produces 2 summary files: check-coverage-passing.txt and check-coverage-failing.txt.

(b) Coverage Goal

- 100% Coverage has to be reached for two classes in commons-math.
- JUnit 4 Templates in the project have to be completed.
- Tests from the commons-math project are allowed.

(c) Fault Localization



Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique

James A. Jones and Mary Jean Harrold
College of Computing, Georgia Institute of Technology
Atlanta, Georgia, U.S.A.
jjones@cc.gatech.edu, harrold@cc.gatech.edu

Abstract

The high cost of locating faults in programs has motivated the development of techniques that assist in fault localization by automating part of the process of searching for faults. Empirical studies that compare these techniques have reported the relative effectiveness of four existing techniques on a set of subjects. These studies compare the rankings that the techniques compute for statements in the subject programs and the effectiveness of these rankings in locating the faults. However, it is unknown how these four techniques compare with Tarantula, another existing fault-localization technique, although this technique also provides a way to rank statements in terms of their suspiciousness. Thus, we performed a study to compare the Tarantula technique with the four techniques previously compared. This paper presents our study—it overviews the Tarantula technique along with the four other techniques studied, describes our experiment, and reports and discusses the results. Our studies show that, on the same set of subjects, the Tarantula technique consistently outperforms the other four techniques in terms of effectiveness in fault localization, and is comparable in efficiency to the least expensive of the other four techniques.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Reliability, Experimentation

Keywords

Fault localization, automated debugging, program analysis, empirical study

ASE'05, November 7–11, 2005, Long Beach, California, USA.

1. INTRODUCTION

Debugging software is an expensive and mostly manual process. Of all debugging activities, locating the faults, or *fault localization*, is the most expensive [14]. This expense occurs in both the time and the cost required to find the fault. Because of this high cost, any improvement in the process of finding faults can greatly decrease the cost of debugging.

In practice, software developers locate faults in their programs using a highly involved, manual process. This process usually begins when the developers run the program with a test case (or test suite) and observe failures in the program. The developers then choose a particular failed test case to run, and iteratively place breakpoints using a symbolic debugger, observe the state until an erroneous state is reached, and backtrack until the fault is found. This process can be quite time-consuming.

To reduce the time required to locate faults, and thus the expense of debugging, researchers have investigated ways of helping to automate this process of searching for faults. Some existing techniques use coverage information provided by test suites to compute likely faulty statements (e.g., [1, 4, 5, 10, 12]). Other techniques perform a binary search of the memory state using one failing test case and one passing test case to find likely faulty statements (e.g., [2, 15]). Still other techniques are based on the remote monitoring and statistical sampling of programs after they are deployed (e.g., [6, 7, 8, 9]). Most papers reporting these techniques also report empirical studies that evaluate the presented technique in terms of its effectiveness and efficiency. However, because of the difficulty in comparing techniques developed on different platforms for different languages and using different programs and test suites, few empirical studies have reported comparison of existing techniques.

Although comparing the fault localization of techniques is difficult, several recent studies have compared four existing techniques in terms of their ability to localise faults. Renieris and Reiss [12] presented their technique, called *Nearest Neighbor*, and compared it to two techniques that use set union and intersection operations on coverage data (similar to those presented in [1] and [11]). Their studies show that, on a given set of subjects, the Nearest-Neighbor technique performs more effectively than the two set-based approaches. Later, Cleve and Zeller [2] presented their technique, called *Cause Transitions*, compared it to the Nearest-Neighbor technique, and found that, on the same set of subjects, Cause Transitions consistently performs better than Nearest Neighbor (and thus set union and intersection tech-

(c) Fault Localization

- Implement the Tarantula technique. (presented in the next lecture)
- Paper can be obtained from our web site. (Password and user name: tad10)
- *ant check-fault-localization* runs an integration test.

Grading

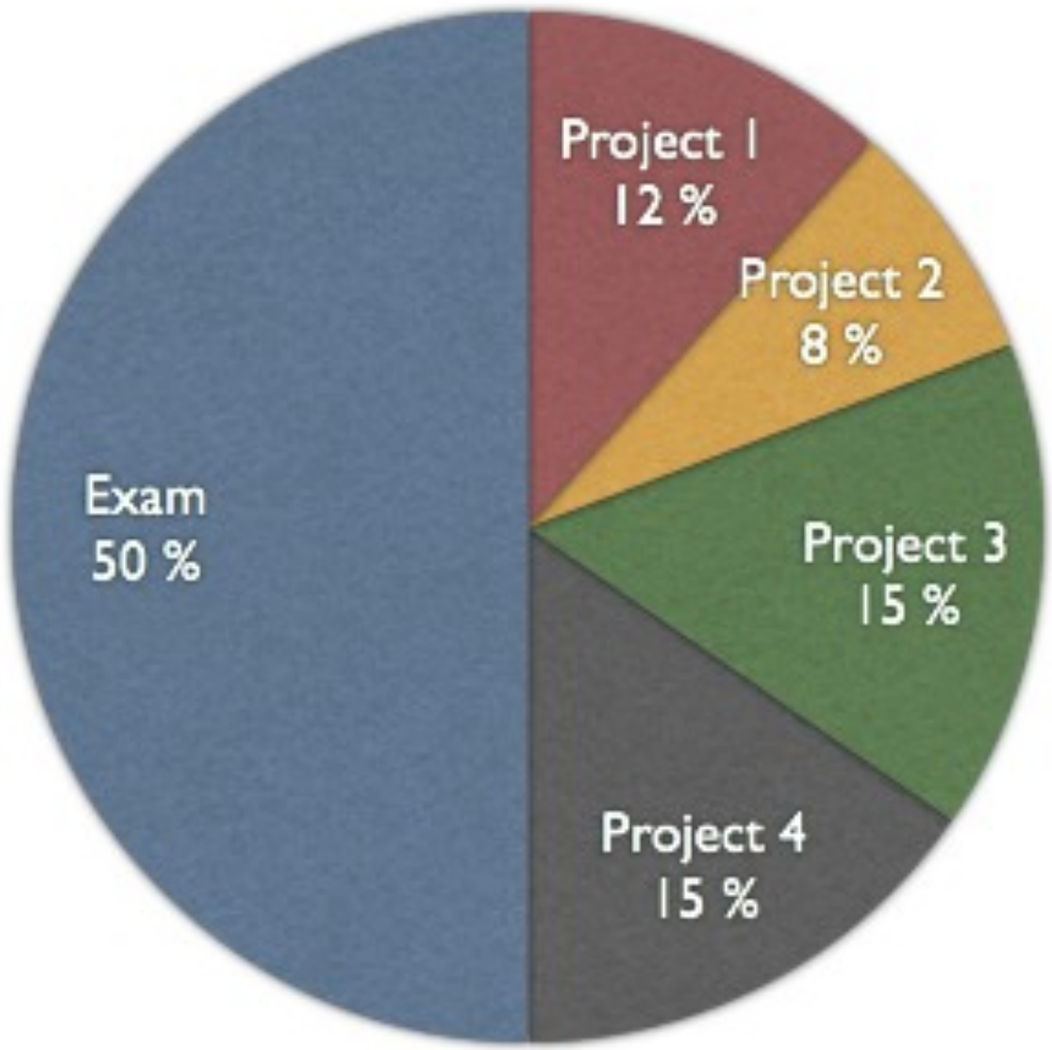
- Version in svn, of 13th May will be graded.
- Public tests have to be passed.
- Secret tests are used for grading.

Grading for lecture

Grading

To pass, you need

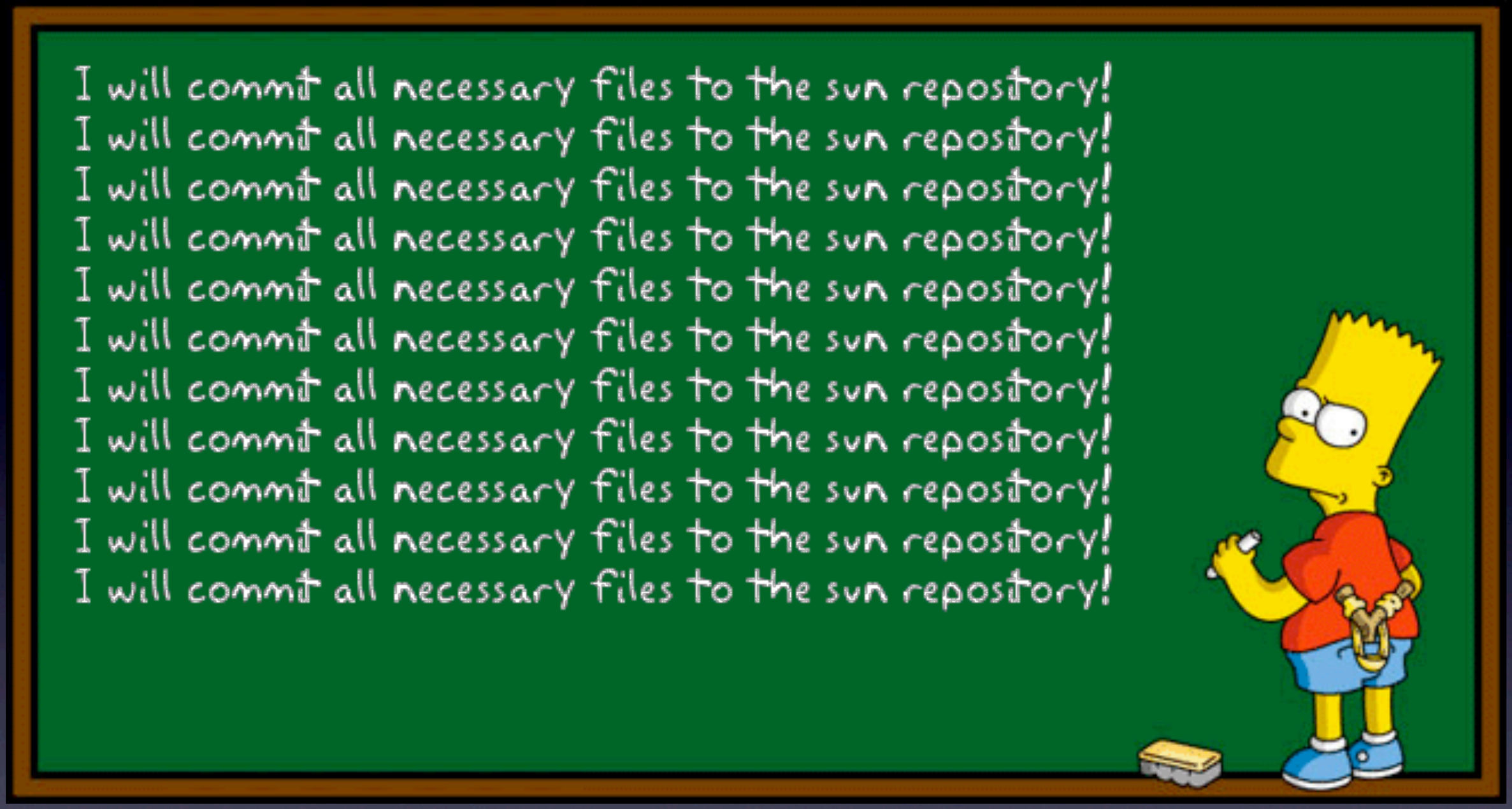
- 60% of exam points *and*
- 60% of project points



Grading Scheme

for Project 1

Grade	Requirement
4	at least 90% of public and 50 % of secret tests
3	100% of public and at least 50 % of secret tests
2	100% of public and at least 90 % of secret tests
1	100% of public and 100 % of secret tests



- Use *svn status*.
- Do a fresh checkout and check whether project compiles and passes tests.

Specification is incomplete

- We try to test only for specified behavior.
- Tests will be made publicly available.
- You can write tests that are run against the reference implementations.

Example

```
1.  for(int i = 0;
2.          i < 2;
3.          i++;) {
```

- We only expect the start (line 1) to be coverable.
- See Test: *AnotherLoopCoverageTest*

Constructors

```
1. public Foo()
2.     super();
3.     this.val = 1;
```

Only line 3 is expected to be coverable.

- *this* and *super* constructor calls are not expected to be covered
- "The first statement of a constructor body may be an explicit invocation of another constructor of the same class or of the direct superclass (§8.8.7.1)."

Miscellaneous

- You are allowed to use 3rd party libraries.
- If you want to use a library not listed in pom.xml, contact us.
- pom.xml and build.xml and src/test will be overwritten for secret tests.