EIFFEL TEST STUDIO
Locating faults in external code
Master Thesis Project Plan

Reto Ghioldi
Department of Computer Science
ETH Zürich

September 6, 2005

Project period	8.September 2005 - 9.March 2006
Student name	Reto Ghioldi
Student no.	00-920-777
Status	9th semester
E-Mail	ghioldir@student.ethz.ch
Supervisors	Ilinca Ciupa, Andreas Leitner
Supervising Professor	Prof. Bertrand Meyer

1

1 Project Description

1.1 Overview

Eiffel Test Studio is a tool for fully automatic testing of Eiffel classes, based on the contracts present in them [1]. However, external code (e.g. external `C/C++` code) is usually not equipped with contracts.

This means that often the manifestation of a bug occurs far from the actual location of the bug in the software and there is no indication of the type of error that occured. There are tools such as memory profilers or memory leak tracers that are able to detect some types of these software faults.

1.2 Scope of the work

The aim of this project is to research if and how such a tool can be integrated into Test Studio to provide information about faults occuring in external code.

1.3 Intended results

After the project's ending, there should be provided

- an analysis of theoretical and technical aspects of possible approches.

- requirements for external testing.

- a list of third-party tools able to fullfill these requirements.

- an implementation written in `Eiffel`/C.

- a set of use and test cases.

- a lasting link concept between Eiffel Test Studio and possible third-party analysis tools.

- a documentation and API description.

- an extensive technical report of all the results.

2 Background Material

2.1 Reading list

2.1.1 Object-Oriented Software Construction

Several chapters from the well known book *Object-Oriented Software Construction* [3], in particular:

- Chapter 1: Software quality

- Chapter 8: The run-time structure: objects

- Chapter 9: Memory management

- Chapter 11: Design by Contract: building reliable software

- Chapter 12: When the contract is broken: exception handling

- Chapter 17: Typing

- Chapter 28: The software construction process

- Chapter 34: Emulating object technology in non-O-O environments

- Chapter 35: Simula to Java and beyond: major O-O languages and environments

2.1.2 Valgrind publications

The Valgrind developers [5] have written some interesting academic publications like

- *Bounds-Checking Entire Programs Without Recompiling* by Nicholas Nethercote and Jeremy Fitzhardinge [2004].

- *Dynamic Binary Analysis and Instrumentation* by Nicholas Nethercote [2004].

- *Valgrind: A Program Supervision Framework* by Nicholas Nethercote and Julian Seward [2003]

2.2 Tools

Various tools will be used during the project time. The following incomplete list provides a shorten overall view

- Eiffel Studio 5.6 [6]

- Eiffel Test Studio 1.0+ [1]

- Valgrind 3.01+ (and other binary analysis tools) [4]

- GNU Compiler Collection (GCC) 3.4+ [7]

- ReadHat Linux

- LaTeX

3   Project Management

3.1   Objectives and priorities

Objective	Priority
Carefull project schedule	1
Test requirements and tool selection which meet them	1
Technique to detect *real* bugs in external code	1
Generic adapter classes from TestStudio to an external tool	3
Plattform independance (related to the binary analysis tool)	3
Set of use cases and test classes	2
API description and further documentation	2
Technical report	1

3.2   Criteria for success

With a proper integrated binary analysis tool and in connection with Eiffel Test Studio, it should be possible to detect bugs in external code. The test output should support the programer with comments and hints to crystalize the real bugs in extrenal code if there is any ambiguity.

The new feature has to be integrated in the existing Eiffel Test Studio in a seamless and flexible way. The performance penalty of the tool integration is reduced to a minimum.

3.3   Method of work

As development platform, Read Hat Linux with Eiffel Studio 5.6 will be used. The focus of the binary analysis tool will be on Valgrind [4].

For the source code exchange a concurrent versions system `CVS` [8] will be used.

3.4   Quality management

Weekly meetings and a carefull schedule should help to reach the project goals and result into high quality code.

3.4.1   Documentation

The documentation will be provided by a technical report, as well as an API description of the implementation and carefully commented code in the sense of design by contract [3].

3.4.2   Validation steps

A set of test classes and use cases will be developed. The implemented features should be able to detect more and more of these constructed pro-

gramming errors.

The number of detected bugs and the tools output quality (comments and hints) can be used as a quality measure.

4 Plan with Milestones

4.1 Project steps

After an analysis of the current system and the state of the art techniques for testing external code, a proof of concept implementation will be written.

With such a base, the implementation should become as plattform independant as possible and shouldn't rely heavily on a third-party tool.

Last but not least the comments and hints on *real* bugs should be improved.

4.2 Deadline

On 9.March 2006 the project has to reach its end.

4.3 Tentative schedule

In the following project schedule, each task is splitted up into phases

1. Analysis & Reading of background material

2. Design

3. Implementation

4. Testing

5. Review

6. Documentation

A loopback to an earlier phases is always possible if the quality of the output isn't satisfactory. Grey frames in the following gant chart represent actual implementation work, milestones are printed black.
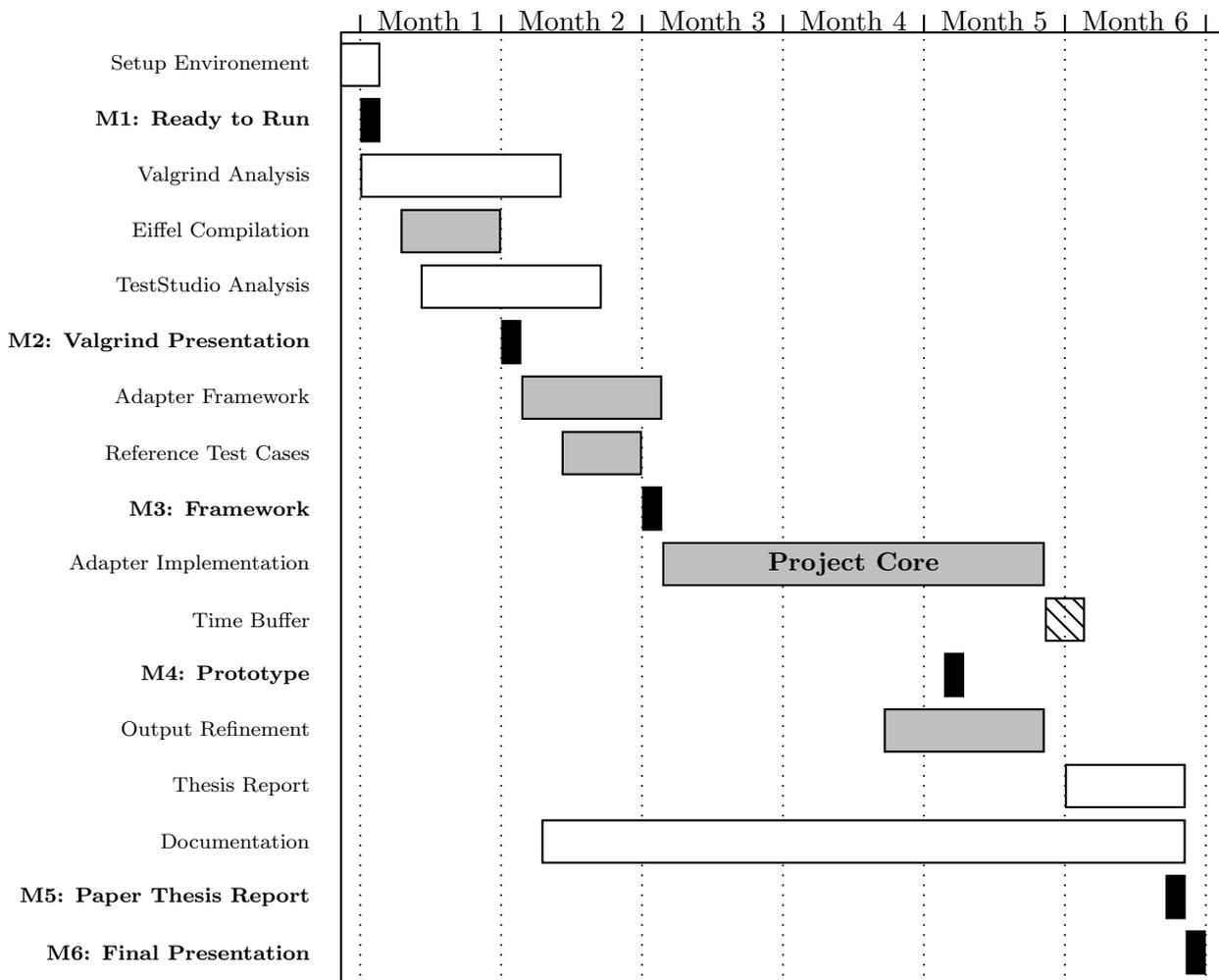
The milestones of the projects are in detail

- **M1** (Ready to Run): 2005 - 09 - 11

- **M2** (Valgrind, Presentation): 2005 - 10 - 03

- **M3** (Framework, Presentation): 2005 - 11 - 07

- **M4** (Prototype, Presentation): 2006 - 01 - 09

- **M5** (Printed Version of the Thesis Report): 2006 - 03 - 02

- **M6** (Final Project Presentation): 2006 - 03 - 09

References

[1] Eiffel Test Studio: An Automatic Contract-Based Testing Environment; `http://se.inf.ethz.ch/people/leitner/test_studio/`, consulted in September 2005.

[2] Chair of Software Engineering: Semester- and Master projects; Online at: `se.inf.ethz.ch/projects/`, consulted in September 2005.

[3] Bertrand Meyer: Object-Oriented Software Construction, 2nd edition, Prentice Hall, 1997.

[4] Valgrind, a suite of tools for debugging and profiling Linux programs; Online at: `www.valgrind.org`, consulted in September 2005.

[5] Academic publications written by Valgrind developers; Online at: `http://valgrind.org/docs/pubs.html`, consulted in September 2005.

[6] EiffelStudio - A Complete Integrated Development Environment for Eiffel; Online at: `www.eiffel.com`, consulted in September 2005.

[7] GCC, the GNU Compiler Collection, front ends for C, C++, Objective-C, Fortran, Java and Ada, as well as libraries for these languages (libstdc++, libgcj, etc).; Online at: `www.gnu.org/software/gcc/gcc.html`, consulted in September 2005.

[8] CVS, the GNU Concurrent Versions System; Online at: `http://www.nongnu.org/cvs/`, consulted in September 2005.