

# **Precondition Enforcement Analysis for Quality Assurance**

## ***PROJECT PLAN***

Master project	
Project period	5 April – 4 October 2004
Student name	Nadja M. Beeli
Status	10 <sup>th</sup> semester
Email address	nbeeli@student.ethz.ch
Supervisor name	Karine Arnout

# 1. PROJECT DESCRIPTION

## *Overview*

Preconditions in components represent obligations imposed by each component to its clients. It's the responsibility of each client to ensure the precondition. A violation of this responsibility is a major error, as illustrated dramatically by the \$10-billion software bug of the first Ariane-5 launcher. Quality of software is improved by checking whether the client fulfils the precondition.

## *Scope of the work*

Finding out whether a particular call ensures the precondition can be trivial (as when the call is preceded by an "if" listing the exact precondition), very hard (requiring full program proving abilities) or in-between. When precondition compliance is not obvious, the project manager should be warned. Such warnings can be a central tool for systematic, contract-based quality assurance.

## *Intended results*

This project is dedicated to writing a program analyzer that will assess whether calls satisfy preconditions, and, when this is impossible to determine, produce a quality assurance report leading to improvement of the code, in the form of either bug fixes or documented "check" instructions. The tool should be practical and usable, and it should keep a history of its results, to support the evolution of components and their callers. The benefit to practical software developers can be considerable.

## *Implementation steps*

STEP I: A static analyser conservatively detects, whether a precondition of a call is always fulfilled by the caller. As an early implementation step, the analyser only detects calls, whose precondition is ensured to be satisfied by a preceding 'If'-statement with a matching condition. Naturally, the condition of the 'If'-guard must not be falsified again before the call, be it intermediate statements or modification of concurrently running threads. The result of the analysis is a *list of suspicious calls* (LOSC), i.e. calls with a precondition that has not been conservatively proved to always hold. This analysis report is printed to the standard console output.

STEP II: The analyser is enhanced for the support of more accurate condition verification. It should be possible to deduce the compliance of the precondition from a guarding 'If'-condition that does not necessarily match the precondition.

STEP III: Calls whose preconditions are not detected as being fulfilled but which the user considers to be true, are manually put on a separate *list of false positives* (LOFP). A call is not reported anymore in the list of suspicious calls, if it is specified in the list of false positives. The LOFP is persistent, in order to enhance the incremental development and software quality analysis.

GUI: A graphical user interface is provided showing two lists, the LOSC and the LOFP. The user can comfortably move elements from one list to the other. The GUI is built with the vision2 library of Eiffel.

## 2. BACKGROUND MATERIAL

### *Reading list*

Chapters in OOSC2 [2] in particular:

- Chapter 10: Genericity
- Chapter 11: Design by Contract: building reliable software
- Chapter 12: When the contract is broken: exception handling
- Chapter 14: Introduction to inheritance
- Chapter 15: Multiple inheritance
- Chapter 23: Principles of class design
- Chapter 26: A sense of style

## 3. PROJECT MANAGEMENT

### *Objectives and priorities*

<i>Objective</i>	<i>Priority</i>
Software architecture	1
STEP I	1
STEP II	1
STEP III	1
GUI	3
Thesis report	1

### *Criteria for success*

The criterion for success is the quality of the software and the documentation. The result may be a partial implementation of the objectives without implying any penalty on the success of the project.

Quality of software:

- Use of Design by Contract
  - Routine pre- and postconditions
  - Class invariants
  - Loop invariants
- Careful design
  - Design patterns
  - Extendibility
  - Reusability
  - Careful abstraction
- Core principles of OOSC2 [2]
  - Command/query separation

- Simple interfaces
- Uniform access
- Information hiding
- Etc.
- Style guidelines
- Correct and robust code
- Readability of the source code
- Ease of use

Quality of documentation:

- Completeness
- Understandable documentation
- Usefulness
- Structure

### *Method of work*

The technologies involved are:

- Gobo Eiffel [3]
- Programming Language: Eiffel [2]

### *Quality management*

Quality will be ensured by:

- Weekly progress reports to the supervisor (on Friday)
- Detailed progress reports for each milestone
- Review of each milestone by the supervisor concluded by a meeting (see validation steps below)
- Documentation (see documentation below)

### *Documentation*

- Progress reports: Short weekly reports and detailed reports for each milestone will describe the progress and the eventual encountered difficulties to the supervisor.
- Thesis report: The thesis report consists of the final developer manual, the final user manual and a theoretical part discussing the profiling.

### *Validation steps*

The validation of each milestone comprises:

- **Report:** Sending detailed report and the relevant parts of the work to the supervisor for review.
- **Meeting:** Organizing a meeting with the supervisor for presentation and discussion of the conducted work.

- **Revision:** Revision of parts or all work for this milestone, depending on the conclusion of the supervisor.

## 4. PLAN WITH MILESTONES

### Project steps

Milestone	Objective
M1	Software architecture
M2	STEP I
M3	STEP II
M4	STEP III
M5	GUI
M6	Thesis report

### Deadline

Milestone	Deadline
M1	2004-04-21
M2	2004-06-18
M3	2004-07-16
M4	2004-08-13
M5	2004-09-03
M6	2004-09-31

### Tentative schedule

	Start	End	5 Apr 2004	12 Apr 2004	19 Apr 2004	26 Apr 2004	3 Mai 2004	10 Mai 2004	17 Mai 2004	24 Mai 2004	31 Mai 2004	7 Jun 2004	14 Jun 2004	21 Jun 2004	28 Jun 2004	5 Jul 2004	12 Jul 2004	19 Jul 2004	26 Jul 2004	2 Aug 2004	9 Aug 2004	16 Aug 2004	23 Aug 2004	30 Aug 2004	6 Sep 2004	13 Sep 2004	20 Sep 2004	27 Sep 2004	4 Oct 2004	11 Oct 2004	18 Oct 2004
<b>Task</b>																															
<b>Master Project</b>	15	41																													
<b>Presentation</b>																															
P1: Presentation	43	43																													P1
<b>Design</b>																															
M1: Software architecture	15	29		M1																											
<b>Implementation</b>																															
M2: STEP I	16	18			M2																										
M3: STEP II	19	27												M3																	
M4: STEP III	28	33																			M4										
M5: GUI	34	37																					M5								
<b>Documentation</b>																															
M6: Thesis report	21	41																											M7		

## REFERENCES

- [1] Chair of Software Engineering: *Semester-/Diplomarbeiten*; Online at: <http://se.inf.ethz.ch/projects/index.html>, consulted in March 2004.
- [2] Bertrand Meyer: *Object-Oriented Software Construction, 2nd edition*, Prentice Hall, 1997.
- [3] Éric Bezault: Gobo Eiffel Project. Retrieved March 2004 from <http://www.gobosoft.com/eiffel/gobo/index.html>