# IUT Design Document

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Description** | **Date** | **Author** |
| 1.0 | Initial version |  | Weizhao |
| … |  |  |  |
| 1.2 | Add implementation structure overview, refine document structure | 2016-5-6 | Ziyi |
| 1.2.1 | Refine 3.1.4 by adding disadvantages of AspectC++ | 2016-5-10 | Weizhao |
| 1.2.2 | Add 3.3 Case Stuty of CCTZ | 2016-5-15 | Yu Feiyue |

Index

[IUT Design Document 1](#_Toc450302218)

[1 Problem Description 2](#_Toc450302219)

[2 Implementation Overview 2](#_Toc450302220)

[3 Test-Function Mapper 2](#_Toc450302221)

[3.1 AspectC++ (Aspect-Oriented Programming) 2](#_Toc450302222)

[3.1.1 Basic Implementation 3](#_Toc450302223)

[3.1.2 Coding 3](#_Toc450302224)

[3.1.3 Compiling (Weaving) 3](#_Toc450302225)

[3.1.4 Pros and Cons 3](#_Toc450302226)

[3.2 gcov 4](#_Toc450302227)

[3.2.1 Basic Implementation 5](#_Toc450302228)

[3.2.2 Pros and Cons 5](#_Toc450302229)

[4 Differ 6](#_Toc450302230)

# Problem Description

In order to figure out the mapping relationship between the test cases and the functions in the source code of the target project, an approach is needed to statically or dynamically trace the execution of every function (including inline function or virtual function etc.) and record it.

# Implementation Overview

Project

v1

Test-Function Mapper

Project

v2

DB

DB Operator

Differ

Tests

Affected Tests

Executor

GCov

AspectC++

CDT

Clang

# Test-Function Mapper

## AspectC++ (Aspect-Oriented Programming)

Aspect-oriented programming (AOP) allows modularizing cross-cutting concerns in a single module, an aspect. Aspects can modify existing classes, but most commonly they provide 'advice' that runs before, after, or around existing functionality.

AspectC++ ([http://www.aspectc.org](http://www.aspectc.org/)) is an aspect-oriented extension of C and C++ languages.

For the IUT project, AspectC++ enables us to log the “signature” (type, name and scope) of every function before, after, or around its execution WITHOUT revising the source code of the target project.

### Basic Implementation

### Coding

There are several concepts in AspectC++ which are essential for writing AspectC++ codes such as *aspect, advice, join* *point and pointcut etc.* (refer to <http://www.aspectc.org/doc/ac-quickref.pdf>).

Aspects are a special AspectC++ language element, which can be used to implement crosscutting concerns in separate modules. Aspect definitions have to be implemented in special “aspect header files”, which normally have the filename extension “.ah”. And in the definition of an *aspect*, we can define *pointcut* to describe where we want our aspect code to be “woven” and define *advice*s that execute aspect code when the target program reaches the *join point*s of the *pointcut*.

### Compiling (Weaving)

AspectC++ requires an specific compiler to “weave” the code we write in the “.ah” file to the target program – ac++/ag++.

The program ac++ is a compiler for the AspectC++ programming language. It is implemented as a preprocessor that transforms AspectC++ code into ordinary C++ code. After the code transformation the output of ac++ can be compiled to executable code with ordinary C++ compilers like GNU g++.

The ag++ program provides a more intuitive frontend to ac++ in a GNU environment. It basically wraps the functionality of the aspect weaver and the c++ compiler into one single program.

The usage of ac++/ag++ is similar to g++, a simple example (assume “.ah” files share the same directory of the “main.cc” file) is

“ag++ main.cc –o main”

### Pros and Cons

#### Pros

1. Based on AspectC++ APIs, simple to implement.

This method is based on AspectC++, a well-built extension for C/C++, which provides rich APIs for us to use to solve our problem. The idea of AOP has been popular for years and has been widely known and accepted. It’s very simple for us to follow the AspectC++ reference to write aspect code and compile it.

1. No change of source code the target project.

AspectC++ allows us to weave our aspect code into the target program without modifying its code. It simply uses *advice* which runs before, after, or around existing functionality. Therefore, there is no need to make an extra copy of the target project and modify it.

1. Few LoC, reduce the possibility of error.

With appropriate uses of AspectC++ APIs, it’s easy for us to achieve our goal with few lines of code. The *match expression*s provided by AspectC++ allows us to easily match certain functions or variables with a given pattern. With less code comes less possibility of error of the program we write, which enhances robustness of the program.

#### Cons

1. Require a specific compiler to compile.

As is mentioned above, AspectC++ needs specific compiler to compile the aspect code, weaving it to the target program. Nevertheless, the usage of the specific compiler is quite similar to GNU g++. Actually, ag++ basically wraps the functionality of the aspect weaver and the c++ compiler into one single program.

1. As for ac++, it’s practically not feasible use ac++ to weave aspects into target code. Because the path of every file that is included in the target code must be specified using the “-I <path>” flag of ac++ command, including even the standard libraries such as “iostream”. As it’s quite difficult and complex to include all the paths, ac++ cannot satisfy our requirement.
2. As for ag++, it’s possible to using ag++ to do the weaving job. We first need to using a C++ compiler (e.g. GNU g++, Clang c++) to generate dependency files, and then use ag++ to weave the code and generate the object files, and finally we can use the C++ compiler to link the object files to generate executables. It easy when the project structure is very simple and it’s compiled by typing commands in terminal, but when there is a big project with many source files and is built by makefile file, it would be quite tricky to adopt ag++ to the project.
3. Need to learn AspectC++.

It takes time to get familiar with the AspectC++ language elements and their usage. And There isn’t much support to find online for asking about issues that we may encounter during AspectC++ development.

1. Might have performance problem.

We’ve applied AspectC++ on some very simple C++ programs, add compared the time cost of compilation between ag++ and g++. It can be known from the result that the “weaving” process can cost some overhead. However, the time of compilation using ag++ seems to be nearly the same. The factors that affect the performance cannot be sure until we try some larger projects with more source code and more files.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Name*** | **#source file** | **#aspect file** | **Time for g++**  **(s)** | **Time for ag++**  **(s)** |
| **Helloworld** | 2 | 1 | 0.267 | 1.210 |
| **Modules** | 2 | 4 | 0.030 | 1.173 |
| **Coverage** | 1 | 2 | 0.030 | 1.894 |
| **Profiling** | 1 | 2 | 0.032 | 1.204 |
| **Singleton** | 3 | 2 | 0.034 | 1.173 |
| **Another\_demo** | 5 | 1 | 0.272 | 1.188 |

1. AspectC++’s support for macros is poor.

In the manual for ac++ (<http://aspectc.org/doc/ac-compilerman.pdf)>, “Macros” is marked as “Unimplemented Features”.

1. AspectC++’s support for filename extensions is disappointing.

By default, AspectC++ only support source files with “.cc” extension. Although a “-e <extension>” flag is to specify the file extension name, it’s still tricky to use this flag.

## gcov

gcov is a code coverage tool for C or C++. Add CXXFLAGS = -fprofile-arcs -ftest-coverage to makefile and add -lgcov when link. Run the target program. gcov -b <filename>. <filename>. gcov is a detailed coverage report file which contains function coverage information.

The goal is to obtain, for each function, whether it is entered. gcov is a code coverage tool for C/C++, which can be used here.

### Basic Implementation

#### Modify makefile

Add -fprofile-arcs -ftest-coverage to CXXFLAGS. Add -lgcov when linking.

#### Make and run

Make the project and run it. You will get .gcno and .gcna file. Do not delete them.

#### gcov -b filename

gcov [options] files

gcov accepts many options. We use -b (short for –branch-probabilities). Write branch frequencies to the output file, and write branch summary info to the standard output. This option allows you to see how often each branch in your program was taken. Unconditional branches will not be shown, unless the -u option is given. The function level summaries are shown when using -b. Traverse the directory recursively and invoke gcov. Some .gcov files are generated after invoking gcov. They are human-readable text files which describe the coverage information.

#### Analyse \*.gcov

Analyse all .gocv files. The .gocv file list the source code appending some coverage information. The function level information is formatted as “function \_Z6globalv called 5 returned 100% blocks executed 100%”. Identify the pattern and collect the relevant data – whether a function is executed or not.

### Pros and Cons

#### Pros

It does not need too much extra code. A basic implementation needs less then 200 LOC.

#### Cons

It does more work than we need because it counts execution times of every line. This will waste some time.

It is not easy to modify makefile for large-scale project.

3.3 Case Study

CCTZ is a C++ library for translating between absolute and civil times using the rules of a time zone. It has 6603 lines of code.

The testing is build by bazel by default in this project. We prefer makefile. Install gtest to system. Modify makefile to build the testing object file. Our program can successfully get the coverage information of every function in the project. But gcov also records the libraries' execution like stl and gtest which are useless. We only analyse the code of project by checking the filename.

# Differ

# Benchmarks

|  |  |  |
| --- | --- | --- |
| Project | LOC | Link |
| Apache Avro 1.8.0 | 15257 | https://projects.apache.org/project.html?avro |
| Xalan-c 1.11 | 130588 | http://xalan.apache.org/xalan-c/download.html |