CS CAPSTONE TECHNOLOGY REVIEW

NOVEMBER 14, 2017

# COVERAGEJSON RESPONSE HANDLER FOR OPENDAP

PREPARED FOR

NASA JET PROPULSION LABORATORY

LEWIS JOHN MCGIBBNEY

PREPARED BY

GROUP55

RILEY RIMER

**Abstract**

This document reviews the possible technologies choices that can be used for the CoverageJSON Response Handler for OPeNDAP project.

# 1 Technology: Testing Framework

## 1.1 Overview

C++ doesn't have a clear winner in terms of which Unit Testing framework to use. This section will cover a few of them and compare their pros and cons.

## 1.2 Criteria

1) Amount of work needed to add new tests.
2) Crash and exception handling.
3) Level of assert functionality.

## 1.3 Potential Choices

### *1.3.1 CppUnit*

CppUnit is one of the most used unit testing frameworks for C++. It has been around for a while and has had years of fine tuning to try to make it fully featured, but in turn it has become a bit obfuscated.

1) It takes quite a bit of work to create new tests in CppUnit. Simple tests end up needed a rather large amount of setup from within the test.
2) It handles crashes and exceptions really well. It uses wrappers around tests to catch all exceptions making the handling of the rather simple.
3) The assert functionality is decent. It is missing some assert functionality for comparative operations, but conveniently variables used in asserts are printed out when their is an assertion failure.

### *1.3.2 CppUnitLite*

CppUnitLite is naturally just a lighter version of CppUnit that was made by the original CppUnit developer. It is definitely very ”lite”, but that allows for a lot of customization to make it fit your own needs.

1) It is really easy to add new tests in CppUnitLite. It just requires the test declaration itself and nothing else.
2) CppUnitLite doesn't handle exceptions at all by default, but it can be added rather simply and be as robust as the user wants to make it.
3) The assert functionality rather bad for CppUnitLite compared to the other frameworks discussed in this paper. It natively only checks equality and it does not print out variable values on assertion failure.

### *1.3.3 Boost.test*

Boost.test is a newer tester framework than CppUnit. It has solid documentation and Boost itself includes other various functionalities.

1) It requires a rather minimal amount of work to create new unit tests. If the tests are part of a suite though it requires quite a bit more setup and the registration of the tests with the suite.
2) Boost.test seems to have the best exception handling out of the main C++ unit testing frameworks. It prints out information relevant to exceptions and also allows for exception handling to be disabled, allowing the exceptions to be caught in a debugger.
3) Has really fleshed out assert functionality. You can assert for all basic operations and it prints out variable values on assertion failure.

### 1.4 Discussion

Each of the frameworks available for unit testing in C++ have their own pros and cons. CppUnit is probably the most tried and tested frameworks but it seems rather dated in terms of functionality compared to Boost.test. CppUnitLite seems like it could be a very good unit testing framework once the setup and customization is done, but the required setup may not make it any better than boost.test.

### 1.5 Conclusion

Boost.test seems to be the clear winner in terms of setup, exception handling and assert functionality. It will be up to discussion whether this will be what will be used in this project considering our work will be part of an already existing framework and what they use will need to be considered.

## 2 Technology: Source Control

### 2.1 Overview

Source control possibilities is a relevant technology choice to investigate, despite GutHub being the norm that we will probably not be able to stray from.

### 2.2 Criteria

1) Usability
2) Version control
3) Work Separation

### 2.3 Potential Choices

#### *2.3.1 GitHub*

GitHub is probably the most used source control technology in current times. GitHub is primarily used in its command line form partly due to its GUI and available alternatives being rather lack luster.

1) GitHub is very usable once a user becomes accustomed to it, mainly due to most people opting to interface their repositories through command line rather than a GUI.
2) GitHub allows for decent control over versioning of a code base. Users can see the versions and changes that were made between versions easily by looking at the web version of the GitHub repository.
3) GitHub allows for merging between separate versions of code and branching. This allows for work to be conveniently separated, it can be rather hard to visualize and control merges through the command line interface though.

#### *2.3.2 TFS*

Team Foundation Server is the source control technology that can be used within the Visual Studio IDE. This integration with the IDE itself is one of the great things about TFS but is also a very big downside. Using Visual Studio as an IDE is a big decision to make and brings about a plethora of its own benefits and downsides. TFS really is only a relevant option if a project is committed to working within visual studio, which is only available on Windows.

1) TFS is very usable due to the simplicity of it being built right into visual studio, allowing clean graphical interaction with source control rather than command line interaction. TFS can also be used from within a command line interface, without having visual studio, but its command line use leaves much to be desired.
2) Version control is simple through the GUI for TFS, allowing similar functionality as GitHub but having it built into visual studio itself.
3) Work separation is rather good in TFS. Again this is due to merging capabilities being built within visual studio itself allowing users to easily see differences and select the portions of code they want to use in a somewhat more streamlined fashion than with GitHub.

#### *2.3.3 Subversion*

Subversion is a newer implementation of the Concurrent Versions System that brings it more up to date with current source control handling.

1) Similar usability to GitHub although somewhat different in some aspects. Branch operations are rather low cost in subversion and subversion allows for a large amount of plug ins for use in IDEs.
2) Subversion isn't distributed like GitHub is, which can bring about some problems with versioning. Some operations take much longer in subversion due to it pulling in all of the information from the code base rather than just relevant data like GitHub does.
3) Merging and branching in Subversion require much more work than with GitHub, making it rather bad comparatively for work separation.

### 2.4 Discussion

The main two contenders for this technology are GitHub and TFS. Subversion has too many drawbacks for it to seriously be considered for a project of this small scale. TFS can be very intuitive for source control, but it requires using Visual Studio to get the most out of it.

### 2.5 Conclusion

This project will definitely just be using GitHub for source control. Mainly due to the requirement that this project is open source and also due to the great usability of GitHub when compared to its competition. The only way TFS could be considered is if this project were to be developed using Visual Studio, which is highly unlikely.

## 3 Technology: Development Environment

### 3.1 Overview

There are multiple different options for doing C++ development and executing C++ files. Some of the most popular means are using an IDE like Eclipse or Visual Studio or just doing compiling and execution through command line calls GCC.

### 3.2 Criteria

1) Usability
2) Testing
3) Portability

## 3.3 Potential Choices

### *3.3.1 Visual Studio*

1) Visual studio has rather usability when it actually is working. It can be rather prone to having errors, especially for a faster development project such as this one.
2) Testing should be rather simple in visual studio as long as there is an appropriate plug in for the testing framework we use.
3) Code written on visual studio should be rather portable, but not nearly as much as compared to just compiling on the terminal.

### *3.3.2 Eclipse*

1) Usability for Eclipse is quite good considering it seems to have better code suggestion and auto complete functionality than what is present in Visual Studio.
2) Testing with Eclipse should be rather simple with the appropriate plugin to run our testing framework.
3) Portability of eclipse should be somewhat similar to what is seen with visual studio.

### *3.3.3 Command Line*

1) Usability is somewhat poor while coding without and IDE due to the lack of on the fly error checking and auto correction.
2) Testing should be rather simple through command line by just running the test frameworks.
3) Portability is best when compiling with GCC on the command line.

## 3.4 Discussion

IDEs can decrease the time it takes to write code and prevent bugs through built in error detection. Command line compiling through GCC has much higher portability than by running through an IDE.

## 3.5 Conclusion

For the development environment we will be compiling our code and running it through the command line, but we also will perhaps be using an IDE for the actual code creation to take advantage of the error checking and text suggestion functionality.