CS3201/2 Automatic Project Testing

The Automatic Project Testing offers a start-up point for your project development and testing. You may use the sample Visual Studio 2010 solution as an example on how to structure your code and testing files. Also, this solution can be used as the base to start the project that can be modified according to your need. In the final states of the development, the sample script can be used to automatically run the complete set of tests.

You should either use this sample or the AutoTester. This sample solution already contains the AutoTester library, so you do not need to download the AutoTester package from the website. However, you need to read the documentation about the AutoTester to be able to integrate your code with the AutoTester.

Download from CS3201/2 website the archive named "SampleTestingSolution.zip" and unzip it to your local directory.

The archive contains a folder and a file:

A. EmptyGeneralTesting folder containing a sample Visual Studio 2010 solution (EmptyGeneralTesting.sln)
B. General_testing.ps1 PowerShell script

A. The Sample Visual Studio 2010 Solution: EmptyGeneralTesting Folder

This sample solution (EmptyGeneralTesting.sln) can be used as a start-up framework for your project development and testing. This solution helps you organize code and test-cases and, then, run AutoTester. This sample solution contains CPPUnit headers and libraries and AutoTester library.

EmptyGeneralTesting.sln solution contains four projects:

1. **SPA:** This project contains the implementation for your SPA. Currently, only a few key headers and CPP files have been created and added to this project. The files belonging to project SPA are located under folder "EmptyGeneralTesting\source". You can add multiple folders/files with your source code.

   All the other three projects depend on this project. Hence, when building any of the other three projects, this project is automatically built.

   The output after successfully building this project is a static library, SPA.lib, located under "EmptyGeneralTesting\Debug".

2. **UnitTesting:** In this project, you put CPPUnit tests created for your SPA. Currently, only one CPP file has been created and added to this project. The files belonging to project UnitTesting are located under folder “EmptyGeneralTesting\UnitTesting\source”.

   This project depends on SPA project (SPA.lib) and on CPPUnit headers and library. The sample already contains the header files from CPPUnit (under EmptyGeneralTesting\include) and the CPPUnit library already built (under EmptyGeneralTesting\lib). However, you might need to rebuild CPPUnit on your computer and manually replace the libraries in EmptyGeneralTesting\lib with the newly generated ones.

   The output after successfully building this project is an executable file, UnitTesting.exe, located under “EmptyGeneralTesting\Debug”.

3. **IntegrationTesting:** In this project, you put integration tests created for your SPA. Currently, only one CPP file has been created and added to this project. The files belonging to project IntegrationTesting are located under folder “EmptyGeneralTesting\IntegrationTesting\source”.

   This project depends on SPA project (SPA.lib) and on CPPUnit headers and library. The sample already contains the header files from CPPUnit (under EmptyGeneralTesting\include) and the CPPUnit library already built (under EmptyGeneralTesting\lib). However, you might need to rebuild CPPUnit on your computer and replace manually replace the libraries in EmptyGeneralTesting\lib with the newly generated ones.

   The output after successfully building this project is an executable file, IntegrationTesting.exe, located under “EmptyGeneralTesting\Debug”.

4. **AutoTester:** In this project, you put system tests created for your SPA and you integrate with AutoTester. Currently, only a few file has been created and added to this project. The files belonging to project AutoTester are located under folder “EmptyGeneralTesting\AutoTester\source”.

   Follow the comments in EmptyGeneralTesting\AutoTester\source\TestWrapper.cpp and the instructions given for AutoTester on CS3201/2 website to fill in this file with the calls to your SPA (SPA.lib).

   The folder “EmptyGeneralTesting\AutoTester\tests” contains your SIMPLE code files and query files, in the format specified in the AutoTester documentation.

   This project depends on SPA project (SPA.lib) and on AutoTester library. The sample already contains the AutoTester libraries (under EmptyGeneralTesting\lib). These libraries are identical to the ones offered in the AutoTester package on the website (no need to replace them).

   The output after successfully building this project is an executable file, AutoTester.exe, located under “EmptyGeneralTesting\Debug”.

When building this solution, you can use two configurations: “Debug” and “Release”. In “Debug”, the output files are created in the corresponding folder “Debug” of the solution and projects. Similarly, the output files in “Release” are generated in the corresponding folder “Release” of the solution and projects. “Debug” configuration allows you to run in debug mode, while “Release” is used only in the final stages of the development and testing (submission).

Problems when building and linking projects in Visual Studio Solutions:

While the EmptyGeneralTesting solution builds and links successfully, you might have problems in linking your projects and adding dependencies once you start modifying this solution. Here are a few things that you need to check in case you have linking errors (given for “Debug” configuration, while “Release” configuration is similar):

1. Solution properties: under “Project Dependencies” show the correct chain of dependencies among the projects in your solution
2. Project properties –check the following:
    a. “Configuration Properties/General”: ensure that the “Configuration Type” is either Static Library(.lib – for SPA project) or Application (.exe – for UnitTesting, IntegrationTesting and AutoTester projects)
    b. “C/C++/Code Generation”: ensure that the “Runtime Library” is “Multi-threaded Debug (/MTd)”. A linking error appears if this field is not set properly.
    c. “C/C++ General”: ensure that “Additional Include Directories” are correctly filled in. Fill in this field if you use external headers/files that are not defined in the project. Be sure to add the correct path to the external files. A building error appears if this field is not set properly.
    d. “Linker/Input”: ensure that the “Additional Dependencies” contain the correct path and name to your libraries that you use in the project (CPPUnit_debug.lib, AutoTester_debug.lib, SPA.lib). A linking error appears if the dependencies are not set correctly. Also, a linking warning regarding debug information might appear when the libraries are not built using Debug information (/MT instead of /MTd – see c.). You may ignore this warning.

B. The PowerShell Script: general_testing.ps1

This script is useful in the final states of the development and helps you to automatically run all the tests you have prepared in a single command.

This script needs PowerShell for Windows: http://www.microsoft.com/powershell. Download PowerShell and install it. The first time when you run PowerShell, you need to change the execution policy to allow for any type of script to be run. To do this, run PowerShell as administrator and input the following command:

```
> Set-ExecutionPolicy Unrestricted
```

Before running general_testing.ps1 script, set the correct path (ending with "\") to the main solution you want to test in the first line of the script:

```
$solution_path = "C:\Path\To\Solution\EmptyGeneralTesting\";
```

To run the general_testing.ps1 script, use the following command:

```
> .\general_testing.ps1
```

This script runs unit, integration and system testing in sequence and prints out the results. You can customize the script to run only one type of testing. Check the comments in the script to understand better how to customize this script for your own specific needs.