**Technical Review & Implementation Plan**

**Team 16: Walnut Crusade**

Colin Bradford

Nicholas Smith

Matthew Wilson

**Introduction**

The purpose of our project is to convert the current build system of the Apache Open Office project into a system that builds in Microsoft Visual Studio. Currently, the project can be built in Windows only by running CygWin, a Linux environment that runs in Windows, and building through that as if one were running in Linux. The goal is to allow a user to build easily in Windows through Visual Studio. The majority of the work we will need to do for this project lies in the conversion of makefiles into equivalent Visual Studio .vcxproj files, and building these converted modules in Visual Studio.

**Regular/Default Visual Studio Project**

Visual Studio automatically determines how to build a project/solution by default. This process requires that Visual Studio understands how to build/compile the project/solution. In its current state, the AOO project is unrecognizable by Visual Studio. Converting the AOO project to a project that Visual Studio can recognize would provide a clean solution, but would require more work and there would be more of a disconnect between this build system and the current one.

This system would create an authentic development experience in Visual Studio. It would not require the use of any external build systems. This solution provides the best experience for native Windows and Visual Studio developers which is one of our largest problems. However, there are some shortcomings to this solution.

There are two flaws with this approach: First, there would be a disconnect between the AOO project for Windows and UNIX/LINUX which would require more time to work around. A secondary goal of this project is to provide a method to generate UNIX makefiles using Visual Studio project files in order to avoid the need for two separate build systems.

Second, this approach will require more work than other solutions. The disconnect may not be a large problem in the short term, but in the long term there could be more problems than with a solution that allows for the projects to be more connected. The second concern is about the amount of work required to complete this approach. This approach would require understanding how Visual Studio builds projects/solutions with more detail, and then determining how to convert the current project such that Visual Studio can take advantage of its default building process.

**GNU Make as an External Build Tool Within Visual Studio**

Visual Studio allows for different build tools to be used when building the projects contained within it. GNU Make is one such build tool. GNU Make is a free build system which relies on *makefiles* to build its projects. These makefiles consist of a set of *rules*, each of which is used to build a *target*. Each rule consists of a list of *commands*, which tell the target how to build, and a list of *dependencies* which these commands use as input. Though a Linux-based build system, a Windows port exists.

jan: yes one of these ports are “nmake” which you describe as the 3 solution.

Because the current AOO build system utilizes makefiles, we would be able to base our implementation of off existing makefiles to some degree; a direct conversion would not be possible due to differences in syntax as well as the fact that certain makefiles are automatically generated. This would minimize the amount of effort needed to convert some modules.

Utilizing GNU Make to build modules from within Visual Studio would be a very straightforward way of migrating to a new, Windows-based build system. Most of the existing makefiles could be used as a model for new makefiles, and each module could be built within Windows and then subbed in to the old build system for integration testing.

GNU Make is a stable build system, and because it only compiles where it detects changes, it is efficient. New developers would likely be familiar with the format, as it is a popular and well-known build system. GNU Make has also been shown to work well with large projects with many dependencies.

The primary downside of utilizing GNU Make is that using it would mean relying on an external build tool. This is undesirable partly because it means more setup for Windows developers to build the system, and partly because the Windows port for GNU Make has not seen updates in several years, and if issues were to arise they may never be addressed.

jan: that is because makefile projects today either use nmake (part of vc) or cmake, so there are no real need for the pure gnu make. Btw, gnu make for Linux has also not had any real updates for a very long time.

**CMake**

Cmake is a method of creating cross platform build files by utilizing a common set of instructions, and allows for the specification of. This allows for the generation of Makefiles and Visual Studio project files utilizing the same source, which theoretically means that there should be a minimal amount of effort to maintain build systems for multiple platforms.

The biggest drawback of CMake is that it does not allow for individual conversion of modules: inserting modules built by the CMake system into the original build system would require that every module be converted before testing the system. Because of our need to test modules as we are converting them to ensure that the entire system builds, CMake would not work very well for our implementation, and this limitation would arguably rule it out entirely on its on. Additionally, despite CMake’s ability to generate multiple build file types, manual modification may still be needed to get the generated files to work properly.

**Decision**

Our choice between these three options is to create Visual Studio projects through the traditional methods, the first solution listed. We chose this option despite the fact that it would likely be the most difficult to implement due to the fact that the end result would be the easiest to use among the solutions. Reasons for this are because it provides a unified build system for each platform instead of relying on tools such as nmake, and because it will be the most convenient for the end user to utilize.

Although it may not match up with existing makefiles as well as other solutions, it should be possible to generate UNIX makefiles from Visual Studio project files. With regards to the time requirements, there are a few ways to mitigate this issue. The first is through the usage of conversion scripts to attempt to automate parts of the process. Once we have a good idea of how to convert modules, the three of us will be able to work on converting modules in parallel.

jan: since you brilliantly and correctly wrote in solution 1), that it is the most time demanding solution, and give long term problems for the project, I think you need to have a big more explanation, why choose the (in your wording) most bad solution.

jan: I agree with the choice, you simply just need to argue why choose that route, and counter all the negative arguments listed in 1). Once that is done you need to extent the document with 2 parts.

a) how do you solve the problems mentioned in 1)

b) how do you solve the problems that I and steve have pointed out to you (solve == solution in detail).