RSM Code Initial Maturity Report

Version: 1.0

Written By:
Bruce Hammond, Consultant
Office of Modeling, SFWMD
March 2005

RSM Code Maturity Report

Summary review:

_____	_____
_____	_____
_____	_____
_____	_____

REVISION HISTORY

Version No.	Date	Reviser	Description of Revision
0.1	3/17/05	Bruce Hammond	Initial Draft
0.2	3/17/05	Jorge Rivera	Draft Revision
1.0	3/17/05	Bruce Hammond	Baseline

Table of Contents

1 INTRODUCTION

1.1 Professional Background

My professional background is 20 years of computer science, about 10 years in government work and 10 years commercial work. I have worked on numerous programs in both the public and private sector and good code design developed to high standards has been one of the main goals on just about all the programs I have been on. However, the reality in all those programs was that perfect code was unattainable, and many compromises had to be made to development considerations. Time and money were usually the biggest things driving compromises and usually the places the code suffered the most because of this were poor documentation and redundancy of algorithms, i.e. many algorithms were very similar or sometimes even identical because lack of communication between developers meant that they wouldn't use existing code, but instead coded routines that did essentially what was already being done in other routines elsewhere in the program. Compromises like these I found almost everywhere and were considered standard business practices, though no one liked to admit to readily. Other places I often found problems that were less tolerated, yet common, were in inefficiency in code and poorly tested code.

1.2 RSM Background

The Regional Simulation Model (RSM) is a modeling program developed over a period of about 10 years by scientists and engineers not necessarily trained in computer science. However the programming language selected for the model, C++, when used correctly provides many of the necessary components for good programming, even when used by people whose background is not computer science. That seems to be the case in with this project. The scientists and engineers who developed this program became well versed in the C++ language and consequently the RSM program is well formed and contains good structure.

The program, particularly the modeling algorithms were largely developed by 3 principal engineers. Though they didn't follow any stringent development standards, they had an informal methodology which all 3 largely followed so the code is fairly consistent in it structure and format.

1.3 Objectives

One of the main objectives for bringing in my expertise was to identify improvement opportunities in the RSM.

From the software package perspective, the things to look for in good software programs are:

- Maintainability – Is the code accessible, easy to change with good internal documentation
- Clarity – Is the code formatted such that it is easy to understand what it is addressing/solving
- good use of language and efficiency – Is the code utilizing the programming language constructs to its fullest capacity, rather than using workarounds

2 REPORT

2.1 Executive Summary

Here is the summary of initial findings with the RSM code, when evaluating for Maturity:

- RSM code is fairly consistent, despite no formal coding standards
- RSM program handles internal documentation well
- RSM test suite is outstanding
- RSM is a very well designed and developed software package

2.2 Summary

Typically a program is developed with formal coding standards so that all the developers have the same developing rules to work with. Though this program was developed without formal coding standards, it appears that there was an informal understanding among the developers so that by and large, the code is standardized. Documentation is a big problem no matter how the issue is looked at because if you have too little of it, understanding the code to maintain it becomes an issue and if you have a lot of it, keeping the documentation up-to-date becomes a huge problem. I can't tell you the number of times I have come across code documentation that was not updated, or updated poorly when the code changed. It actually became a hindrance to the development process because it didn't accurately portray what the code was doing and thus someone learning the code to maintain it wastes a lot of time trying to understand it, only to ultimately realize that it is incorrect and the code itself must then be studied for correct understanding.

The RSM program is not heavily documented, but I found that all the difficult algorithms and even many of the not so difficult ones are well documented. I personally think this is the best way to document because most of the documentation is more about the concepts driving an algorithm instead of pseudo-code or a bulletized list of functionality which the developer will end up finding from the code itself if he does his job correctly. Explaining the concept behind a coding routine and showing the algorithms in mathematical terms when necessary are very important for maintainability of the code and I think the RSM program handles documentation well.

Redundancy of algorithms is a problem in all programs. Though in the ideal world, reusable code that has a minimal amount of redundancy is the goal, it is never achieved. Developers, no matter how good the development environment, work in somewhat of a vacuum. They just don't know everything other developers are doing so functionality often gets repeated because one developer doesn't know another developer has done the same thing or something similar. Developing library functions is probably the best way to reduce redundancy and promote reusability of code. The RSM program has done a good job utilizing library functions, both those by outside vendors and those written in house. While I did find some redundant code, because of the libraries and the use of C++ overloading functionality, I think the amount of redundant code in the program is very small.

Inefficiencies in the code are another problem I mentioned and are probably one of the harder problems to recognize. While sometimes inefficiency shows up early in development, i.e. a program takes way too long to execute, it is more common that only after a program is in place for a while do some inefficiencies show up. Developers and long time users are probably the best people to point our inefficiencies in programs. Talking with developers of the RSM program, I got the impression that they felt that some input sets seem to take to long to execute. I don't know if this is because of inefficiency or if that is just a reality of the situation, i.e. sometimes programs aren't as fast as you would like. However the developers have pointed out some areas in the code for me to investigate whether the code is running less optimally than is possible. This is often a difficult thing to determine because the people who would know this the best are the developers, but their services are needed more at the new development side of things than the optimization side.

The last problem I mentioned that is quite frequent is testing. I have to say that the RSM test suite is one of the best test packages I have seen. Consistent regression testing is often very difficult and hard to standardize. The suite of test cases the RSM program has is really very good and should be continued to be maintained as the program matures.

In summary, I have to say the RSM program is a very well developed program. While there are numerous small adjustments that I can make, which I am listing below, the program was developed with very good methodologies. That isn't to say it is perfect and one big problem that I was apprised of with the program that I haven't mentioned yet, probably can't even be addressed at this time. That problem is the difficulty in bringing in a new scientist or engineer with a water moving algorithm to be simulated in the RSM program. Currently it involves training the scientist/engineer in C++ and then requiring them to become familiar with the current C++ class structure that exists in the code. For a non-computer scientist this is a huge hurdle and a big roadblock to adding new simulation models to the RSM program. The bad news I have about this is that there isn't much that can be done about the problem at this time. That is a reality you have to live with because of the current state of development of computer science. While computer science has made huge strides since the days of

assembly language and punch cards, there still is a long way to go until the day when non-computer scientist can easily take advantage of the computer. The current methodology that is being used of training engineers/scientists in C++ is really the most effective way to do this at this time. In the future, how distant is unclear, 4th and 5th level computer languages promise to be much easier for the non-computer scientist to use (assembly language is level 1, C and FORTRAN are level 2, C++, Ada, etc. are level 3). They will be more oriented to explaining a problem than learning the specifics of the language as is currently the case. Maybe someday you will be able to be like Scottie from Star Trek and just explain your problem to the computer, but currently, programming languages are a long way from that scenario and you are going to have to live with the realities of your situation, i.e. you need to train your scientists/engineers in how to take advantage of the C++ program.

3 PROPOSED IMPROVEMENTS

3.1 Summary

Here is a summary of some of the improvements that can be made to the code:

1. Reduce compiler warnings. This can be done by doing the following:
   a. Cast variables where type mismatching occurs.
   b. Reorder variable definitions when calling base functions.
2. Remove obsolete code.
3. Update “Software Coding Standards for C++” to reflect current coding standards, which appear to be consistent.
4. Add comments to code as necessary.
5. Identify code optimization opportunities and implement to speed up code where possible.
6. Improve the method for parsing input data as outlined by various Model developers.
7. Add use of automake, autogen, and autoconf, so that the code can be distributed to be built on different platforms.
8. Standardize error and warning messages (consolidate error trapping).
9. Check to see if the use of linked lists is the best way to organize data when searching.