NOTE 170
The AIPS++ Code Review Process

Paul Shannon, NRAO

June 23, 1995

1 Introduction

The AIPS++ project adopted a formal procedure for quality checking code in the late summer of 1994. The procedure requires that all new core[1] source code be submitted to the "code cop" (*email: aips2-cop@nrao.edu*) before it can become sanctioned AIPS++ code. Once received, the cop assigns a reviewer, who judges the code against the **AIPS++ Standards and Guidelines** (note 167). A dialog between the programmer and the reviewer may follow, eventually leading to approval of the code, and its formal inclusion into AIPS++.

In this note, I discuss the review procedure

- in general
- from the perspective of the programmer submitting new code
- from the perspective of the reviewer
- with special attention to test programs

2 General

There are eight documents, in addition to this one, which are essential to the code preparation and review process. Both programmer and reviewer should study them carefully, and base their work upon them.

- *code/install/codedevl/template-class-h:* how to construct a class header file.
- *code/aips/implement/Lattices/Slicer.h:* good example of a class header.
- *code/install/codedevl/template-module-h:* how to construct a module headers file.
- *code/aips/implement/Tables.h:* good example of a module header.
- *code/install/codedevl/template-global-functions-h:* how to construct a header file for global functions.
- *code/aips/implement/Arrays/ArrayIO.h:* good example of a global function header file.

[1] *aips, dish, synthesis* and *vlbi* packages

- *code/doc/notes/167.latex:* Standards and Guidelines Checklist.
- *http://www.cv.nrao.edu/aips++/docs/html/cxx2html.html* An online guide to the documentation extractor, with good descriptions of the markup tags to be used in creating header files.

There are three main goals for the code review, all of which are equally important:

- Guarantee that the fundamental design of the class, as exposed in the public interface, is sensible and reasonably complete. We are more concerned about overall design than implementation details, because we assume that the programmer is competent; that the implementation may be too complicated to analyze in a reasonable amount of time; and that test programs and early usage will indicate errors and gross inefficiencies. The public interface, though, is everyone's concern, it is a good indicator of overall design, and it is painful to change once a class has been checked in.
- Make sure that the class header file is a sufficient guide to the class for client programmers. This is a very high standard, and demands a great deal of work from the programmer and the reviewer. "Sufficient" means that a programmer, previously unacquainted with the class, can read the header file and quickly reach a point where he or she can write code that uses the class. This standard of explanation is often met by well-written programming books, and is rarely met by programmers in their own internal documentation. It is, nonetheless, the standard for all AIPS++ source code.
- Use the test program(s) to make sure that memory leaks are discovered and removed. See that code coverage is about 90% complete[2].

A great deal can be learned from reading through a class, and running test and demo programs. But no review is complete until other programmers use the new code. For this reason, we have a package called **trial**.[3] This package is automatically distributed by the AIPS++ code distribution system. If you ask the code cop to install your code in the trial directory, it will be available for everyone to examine and use. (All of these possible users will understand that *trial* code is subject to change with little or no notice.)

3 For the Programmer

- When you are within a week of submitting code, send the cop email briefly explaining the purpose of your class, and when it will arrive. This lead time will allow the cop to begin the search for a suitable reviewer.
- If your class is called, for example *"NewClass"*, then you need to submit the following files:
    1. *NewClass.h*: the class header
    2. *NewClass.cc*: the class implementation
    3. *tNewClass.cc*[4]: a test program – see section devoted to this topic below

[2]Tools like "Purify" and "TestCenter" should be used if they are available.

[3]*(your-workspace)/code/trial* – at the same logical level as *aips, dish, vlbi, synthesis* and all of the site-specific packages

[4]If there is more than one test, exec, output or demo program, name subsequent files, for example *tNewClass1.cc, tNewClass2.cc* – the first program is implicitly number zero.

4. *dNewClass.cc* (optional): a short, heavily commented and well-focused program demonstrating how to use *NewClass*.

- Your code (header, source, test, and demo programs) should be combined into a standard unix "shar" file and emailed to *aips2-cop@nrao.edu*[5]. This shar file should preserve the directory structure needed to build the code once it is unshar'd.
- The cop will pass the shar file onto the reviewer, and tell you who the reviewer is.
- All subsequent email exchange between you and the reviewer should be CC'd to the cop.
- If you have test coverage and memory leak statistics (*strongly* recommended for all programmers who have access to the right tools) please send them to the cop in a separate email message.

4 For the Reviewer

Your job is a difficult one and requires a lot of time and effort. It may also require imagination and diplomacy. Imagination is needed because you must cultivate the perspective of the non-expert client programmer, intentionally forgetting things that you know, and scrutinizing the code (the class header in particular) to make sure that it explains everything the client programmer will need to know. You must also be able to take on the domain expert's point of view, and carefully examine the new class for the quality of its overall design. You may need considerable diplomacy to persuade a recalcitrant author that they need to make changes in their code and documentation.

Your responsibililites include these specific actions:

- Let the cop and the author know how long it will take to make the first review of the code, and how well you are sticking to that schedule as the time goes by.
- All email between you and the author which is related to the new class should be CC'd to the code cop.
- Be prepared for the cop to suggest changes beyond those you require.
- Base most of your evaluation on the header file for each class. A quick reading of the implementation file will allow you to judge the adequacy of its documentation, and detect any troubling features.

5 Test Programs

The programmer must provide one or more programs that thorougly test the newly submitted code. These programs should be very simple to run, and should exercise all aspects of the new code.[6]

[5] If the class source code amounts to many thousands of lines of text, explain this to the cop – it may be possible for the reviewer to examine the code "in place", on your machine, or checked in to the *trial* directory.

[6] A handy way to construct a test program for a new class is to copy the class header file into a new file – which will become the test program – and edit the member function declarations into member function calls. This simple approach guarantees that all of the member functions are called. The next step would be to intentionally call all of the member functions in such a way that all the exceptions are thrown: if you trigger and catch all of the exceptions,

To indicate a successful test, it will often suffice for the test program to simply print"OK" to standard output, and return *0* as the value of the main program. In the test program fails, it should print a descriptive error message, and return some non-zero value to the environment. Some classes and global functions (especially those which employ numerical methods) can only be adequately tested by comparing a list of expected results against actual results, making allowance for tolerable roundoff differences[7].

The programmer may use two strategies here:

1. Create tMyClass.out, which contains the *expected* results, and tMyClass.exec, a shell script which runs tMyClass, redirects the actual results to a file, and compares them to the contents of tMyClass.out.

2. Alternatively, the tMyClass program *itself* creates the new results file, and runs the comparison, using a *system* call; when the comparison of results is complete, tMyClass prints "OK" and returns *0* to the environment if it is successful, just as the simpler test programs do.

then you have gone a long way towards exercising all of the code. "Domain" coverage should be attempted too: exercise the member functions by calling them with a range of inputs, paying particular attention to the boundaries of the domain.

[7] We hope that, in time, the AIPS++ project will find or develop a "diff" tool that is smart enough to compare numbers, and distinguish between allowable differences (due to word-length round-off) and unacceptable numerical error.