Lessons learned from the November 1993 AIPS++ library release experience.

0. Allow sufficient time for testing and debugging after the release is believed to be complete.

   We believe that this time period should be one month, minumum.

   This time period begins when the release is believed to be complete and correct, so that if the testing and debugging were not done, the release could be made immediately. A "complete" release includes the following components:

   a. Directory structure and organization.
   b. List of source files which are to be contained in the release, including where they are to be obtained. These are not only code, but also documents, tools (e.g. the document extractor), etc.
   c. Makefiles for building the release
   d. Documentation generated for the release, including directions for building the release.
   e. Tools used for creating and packaging the release.
   f. List of generated files for the release, and their organization.

   We spent at least one month testing and debugging, after discussions where we wondered if we could release the next day "if everything worked out OK in the next 24 hours". Everything took longer than we optimistically thought.

   We started building the directory organization and Makefiles far too late in the process, and did not have a clear list of what was to be included in the release until later than we should have.

1. Test and debug thoroughly.

   Test not only the code, but also the directions and procedures for building the release as an outside user.

   We found and fixed several significant bugs during the testing for the release. This benefitted not only the release, but the AIPS++ code in general.

   The build procedures and Makefiles, and their directions, would have benefitted from more extensive testing and debugging.

2. Keep the AIPS++ installation up to date, instead of bringing it up to date for the release.

   Documentation and test programs and procedures should be kept current and up to standards, and test programs should be run periodically, as we work, so that we do not have to make a major effort to fix them for the release.

   We spent alot of effort for the release fixing code documentation, writing or fixing test programs, and fixing bugs which test programs would have caught if they had been run.

3. Keep the release build procedures simple and easy to use.

   Inherent in this is being clear on what the release is for, so that we can build it with these clear goals in mind without adding extra features which complicate matters.

   Our release was moderately complicated. It had two sections: "base" code which anyone could build and use, and "extra" code which either was alpha code or required extra libraries. The release also provided several services for the user, including putting the template repository object modules in a library and providing prebuilt binary libraries.

   While all very useful, these options and flexibilities made it hard to write complete directions which covered all possibilities and

situations. Judging from the problems which some users, both internal and external to AIPS++, have had, either these users are trying to do things which are not documentated sufficiently, they are not reading the documentation carefully enough (because it is too complicated?), or they are just forging ahead and finding the procedures not sufficiently intuitive.

4. On a speculative note, it would be preferable to be able to work on the release in parallel with normal AIPS++ development, instead of freezing AIPS++ code while we debug the release.

   The normal AIPS++ development environment does not allow for parallel code development. As a result, while we were debugging the release, we had to freeze all code except for changes made for the release, As the debugging dragged on, this was the cause for some impatience on the part of developers who were prevented from checking in (and therefore propagating to other developers) their work.

5. Another speculative note is that it would be nice for the release directory organization and Makefiles to be the same as those used in normal AIPS++ development.

   Since the requirements for a public release environment are different than those for an AIPS++ development environment, by principle 3) above, this may not be possible. However, it might be worth giving some further study to this idea. It might be possible to separate the extra capabilities required by an AIPS++ developer and layer them in some way.

   If it were possible for the release build system to be a subset of the general AIPS++ system, then it would save the time of building a new one, besides giving us all a more familiar environment in which to work while working on the release.

   It was generally agreed that for this release, a directory organization and Makefiles simpler than those used in normal AIPS++ development were required.