Roadmap for the CASA Group

Nick Elias, CASA developers, and CASA β testers
2009 November 30
Version 1.01

0. Introduction

Before I joined the CASA group as its supervisor, I knew that the job would present a multitude of challenges. Consider the following:

- There is a lot of administrative work, including pushing paper and attending meetings.
- Management of scientist/developers, as well as their interactions with testers and users, is often compared to "herding cats."
- The project has been running for many years, so there is a lot of built-in inertia that hinders procedural changes.
- The size of the group is close to the maximum recommended for a first-line manager (~ twenty).
- The members are divided into four teams dispersed across sixteen time zones. The other groups with which we interact are also widely distributed.
- Management duties for this group have been performed by more than one person, which is consistent with the flattened management structure of many academic institutions.

These challenges are formidable.

The CASA data-reduction software is critical for the success of the EVLA and ALMA instruments. Also, anyone involved with these instruments would be in a good position to use them for his/her own scientific observations. For these reasons, and for the challenges listed above, I applied for the CASA supervisor job and was very happy when I was hired.

In a standard scientific project, there is 1) a principal investigator, 2) a project scientist, 3) project members, and 4) a project manager or supervisor. For larger projects, the principal investigator and project scientist may be senior members of an advisory council. Apart from a principal investigator, the CASA team contains all of these elements.

The advisory council (in our case, the cabal) sets development priorities and provides some technical advice for the project. The supervisor (me) assigns/schedules tasks to the project members (developers). He/she also performs day-to-day administration, keeps track of the big picture, provides technical leadership, participates in high-level discussions with the advisory council, and reports progress and problems to the advisory council.

With this group structure in mind, I present my initial impressions, concerns, and solutions that form the basis of a disciplined roadmap for future development cycles. I also provide a brief appendix describing how I managed my previous software project.

The changes I propose, based on group input and my personal experience, are far less draconian, invasive, and complicated than those imposed in industry. They will make all aspects of development more efficient and less stressful. I am confident that all challenges will be overcome and that CASA will ultimately succeed.

1. Initial Impressions

The scientists and developers in the group, as well as the testers with whom we interact, are very knowledgeable, energetic, and passionate about radio astronomy and CASA. I do not make this statement simply to curry favor with employees or management. When I send a developer an email on a Sunday evening, very often he/she answers immediately. If you ask any radio astronomer about the NRAO scientific staff, they are always highly complimentary. Supervisors in industry should be as lucky as I am to manage such employees. Enough said.

I spent my first month as CASA supervisor learning how this group functions. Given my previous experience, I'm pretty sure that any other group of people under similar circumstances would have crashed and burned long ago. Apparently, the high-level of competence of developers and testers, as well as the quantity of their communication, partially mitigates the pitfalls of distributed project management. It was clear to upper management that a full-time supervisor was required, and I will do my best to meet their expectations.

2. Concerns and Solutions

In this section, I itemize my concerns and solutions. Some may overlap slightly, and they do not appear in a strict order of precedence. If you do not wish to read all of the details, you can just look at the summarized concern and solution statements (some solution descriptions are long).

The changes I propose will appear to slow the development process down, especially at the beginning while we grow accustomed to and tweak them, but I expect that increased efficiency will compensate for this loss. The developers and testers on this project work very hard, and I want to insure that their efforts are utilized as effectively as possible.

Comments and questions from all interested parties are welcome.

Concern 1: The roles of group members are not well defined.

In Section 1, I mentioned the concept of "distributed project management," which means that multiple members of the group are handling different parts of the project manager's job. This type of behavior has evolved out of necessity, but it is always a bad idea to continue it for long periods of time.

Solution 1: Assign proper responsibilities to all group members.

One person (the supervisor) must be responsible for the software and the developers, otherwise no one will be. The supervisor should be allowed considerable leeway when dealing with implementation issues. It is critical.

The developers are responsible for functioning within and contributing to the new roadmap procedures. For NRAO employees, I will base part of your yearly evaluation on these criteria. I also manage employees from other institutions. If I am asked by your supervisors to evaluate you, I will employ the same criteria. Your cooperation is important and will be greatly appreciated.

The cabal and the project scientist will provide development priorities (including those provided by non-cabal stakeholders) and some technical advice. Their participation is essential.

Concern 2: There is little long-term planning for CASA development.

Formal planning does not go far beyond the next development cycle. This fact makes effective long-term planning difficult.

The developers are very good with the "duct tape and a hammer" (or, "secure and pound into submission," "search and destroy") approach to development. This skill, which I believe has evolved from the independent can-do cowboy attitude of the astronomers who wrote software for the infant VLA, is ideal for hunting down and fixing bugs. I identify with this attitude and I like it very much and I want to keep it. Without an overall plan to follow and maintain, however, stomping bugs becomes a never-ending hell of broken code and continuous fire fighting with no end in sight.

Solution 2: Make long-term planning a higher priority.

The CASA supervisor will appoint an architect (Solution 10). This person should be intimately familiar with most or all aspects of the CASA package. He/she will spend approximately 40-50% of non-science time on long-term planning, including low-level implementation issues (e.g., scratch columns, splitting/combining/eliminating tasks, etc.) as well as "big-picture" issues (HPC, algorithms, listing/flagging/visualization of large datasets, scalable software, reorganizing I/O and CPU intensive applications as stand-alone parallel processing, etc.).

The architect, with input from developers and the cabal, will create an "architecture roadmap document." The architecture document describes the existing CASA package, future changes, and how/when the changes will be implemented. The latest technical review document already provides some of this information.

The CASA supervisor will deal mainly with day-to-day management of the project. The architect will meet with the CASA supervisor on a weekly basis to insure that new JIRA tickets are consistent with long-term plans (tickets may be modified or rejected on this basis; Solution 7). The architect will also take part in weekly cabal meetings to provide additional technical insight.

Concern 3: Assigning/prioritizing/scheduling JIRA tickets is haphazard.

At the present time, group members assign and prioritize JIRA tickets without the knowledge of the supervisor, which is a symptom of distributed project management. This behavior leads to unnecessary stress and broken code. It also makes short- and long- term planning impossible.

Solution 3: Make the supervisor the “gatekeeper” for JIRA tickets.

The JIRA system is perfectly acceptable. We are not using it effectively.

Developers should only modify code that has been captured as a JIRA ticket. No private coding arrangements are allowed, because they can come back to bite us in the future. **If the supervisor is supposed to be responsible, it is not fair to do development without his knowledge.**

Module-based default assignment does not appear to work well because some developers accrete more JIRA tickets than others. Also, it is difficult for the CASA supervisor to know exactly what the developers are doing unless he asks (the developers typically do not use the in-progress option in the JIRA system, the weekly reports represent only after-the-fact information, etc.). Therefore, all tickets should be initially assigned to the supervisor, signifying that they are unassigned. When tickets reach the top of the queue (Solution 7), the supervisor can assign them.

There has been some controversy about whether the CASA supervisor should be the default assignee. This policy does reduce the freedom of the developers, but given that they are distributed among multiple locations and that mid- to long- term planning, scheduling, and assignment have been somewhat haphazard, I would rather err slightly on the side of “micromanagement” as opposed to “laissez faire.” Besides, in the future most JIRA tickets will be reassigned anyway, so whether the supervisor or a developer is the default assignee makes little difference.

Here are four bullet points answering known objections:

- I do not want to use the JIRA system for mid- to long- term scheduling because it is clunky. I will use another document (exact form TBD).
- Each developer will be assigned a few prioritized tasks to insure that they are busy. This provides a buffer just in case the supervisor is out of the office for a few days.

- If the supervisor is out of the office for an extended period, the "as needed" deputy will handle the day-to-day management (Solution 10).
- The transition between the old and new assignment procedure will be gradual, to verify that the latter works smoothly. If the new assignment procedure does not work, I will abandon it.

Concern 4: Not enough information is provided to the developers to resolve JIRA tickets.

The JIRA system is capable of storing the required information for the successful completion of tasks. Insufficient information has led to various problems, including invalid or incorrect or non-maintainable code that must be rewritten.

Solution 4: The reporter must include the required information to the JIRA system.

The required information is:

- an issue type (the drop-down widget)
- a summary (entered in the summary field)
- a priority (the drop-down menu widget; Solution 5)
- a component list (the listbox widget)
- a detailed description, including operating system, 32/64 bit, result from casalog.version() command, input parameters, console/log outputs, data, scripts, references for math, etc. (the description area)

The assignee should be the CASA supervisor, since he is responsible for assignments (if that part of Solution 3 is implemented). The operating system, helpdesk ticket number, and ALMA/EVLA priority can be entered as well. The release date should be specified for all blocker and critical priority items. The due date widget and the time remaining fields are of limited use, given that we don't explicitly manage dependencies or estimate the critical paths.

The CASA supervisor reserves the right to modify all aspects of a JIRA ticket. If the reporter disagrees, he/she can appeal. Lively and respectful debate is encouraged, but the supervisor must ultimately make the decision (Solution 1). Modification includes closing duplicate tickets, splitting long/open-ended tickets (if efficiency is improved), as well as linking tickets to existing ones.

All projects make completion time estimates for their tasks. A JIRA ticket priority, without a pre-assignment completion time estimate, is insufficient for making effective scheduling and planning decisions. There appears to be reluctance within the CASA group toward making such estimates because of their inherently low accuracy. A mediocre estimate is better than no estimate. The supervisor will add a completion time estimate, based on his own experience and consultations with developers.

Some users might not know to formulate a good detailed description. The CASA supervisor may ask the reporter (who could be a developer or support person from the helpdesk) to augment or clarify the detailed description of a JIRA ticket. When assistance is required, the helpdesk or supervisor (and/or developer) will provide it. If no information is sent in a reasonable amount of time (~ a week), the supervisor reserves the right to close the ticket until the information is provided. Extensions will be granted for standard reasons, e.g., vacation, family emergency, conference, proposal due date, etc. Rudimentary training materials to instruct users on how to submit a helpdesk ticket that can be turned into a JIRA ticket (not just the mechanics of the web page) should be provided at all CASA workshops.

Concern 5: The JIRA ticket priorities are not well defined. “Priority inflation” is detrimental to the project.

It appears that different reporters have different standards for JIRA ticket priorities, making short- and long- term planning difficult. Single-word or numeric priorities are useless.

Solution 5: Unambiguously define the JIRA ticket priorities.

Here is the table for JIRA ticket priorities, in order of priority:

Bug	Blocker	The issue completely hinders data reduction for one or more supported use cases. It must be closed before the designated release (or patch).
Bug	Critical	The issue significantly hinders data reduction for one or more supported use cases. It should be closed before the designated release (or patch), except under extenuating circumstances. If it is not closed for the designated release (or patch), it should be promoted to a blocker for the next patch (or release).
Improvement or New Feature	Blocker	No workaround for the issue exists for one or more supported use cases. It must be closed before the designated release (or patch).
Improvement or New Feature	Critical	A complicated workaround exists for the issue for one or more supported use cases. It should be closed before the designated release (or patch), except under extenuating circumstances. If it is not closed for the designated release (or patch), it should be promoted to a blocker for the next patch (or release).
Bug	Major	The issue lightly hinders data reduction for one or more supported use cases. It does not have to be closed for the designated release (or patch).
Bug	Minor/Trivial	The issue does not hinder data reduction for one or more supported use cases. It does not have to be closed for the designated release (or patch).
Improvement or New Feature	Major	An easy workaround exists for the issue for one or more supported use cases. It does not have to be closed for the designated release (or patch).
Improvement or New Feature	Minor/Trivial	No workaround is required for the issue for one or more supported use cases, it will be implemented for convenience. It does not have to be closed for the designated release (or patch).

We make no distinction between minor and trivial priorities for JIRA tickets. If a low-priority issue suddenly becomes more important, its priority can be raised at the discretion of the supervisor. Some issues may have different priorities for EVLA and ALMA. In this case, due dates will determine which priority should be used. Most low-priority tickets will be handled during bug-fix periods (Solution 8). This strategy will help the CASA group respond to user tickets more rapidly.

Concern 6: When a JIRA ticket for a bug is closed, many times the same bug immediately appears after retest.

This problem is quite stressful for both tester and developer alike. It would be to our advantage to minimize stress, because we are moving from β versions to public release (the majority of the testers will have other duties and will not be constantly rechecking bug fixes).

Solution 6: Take significant steps to reduce this problem.

A few bullets:

- The existing build system may sometimes by the culprit. I have appointed a build system lead to remedy that situation.
- We employ little or no unit testing at the task level (Solution 7). This step represents the compromise "triage" which must be performed because we don't have enough time to create a complete unit testing system at the C++ level.
- Each task may have more than one unit test.
- Unit tests should use "toy" datasets so that they run quickly.
- Unit tests should be run by the developers as well as automatically by the build system.
- JIRA tickets should be resolved after successful tests on the stable build and closed after successful regression tests.

Like it or not, we (including me) will be judged by users as well as management. We must do better.

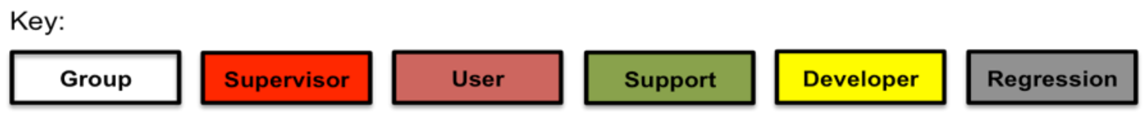
Concern 7: The JIRA ticket lifecycle is not well defined, which leads to distributed program management, ill-defined tickets, and forgotten open-ended tickets.

The JIRA system should be a repository of action items and their associated information, not a dumping ground. When we use JIRA correctly, it becomes a more effective tool for day-to-day management.

Solution 7: Provide simple flowchart(s) of the JIRA ticket lifecycle, including the steps and their responsible parties.

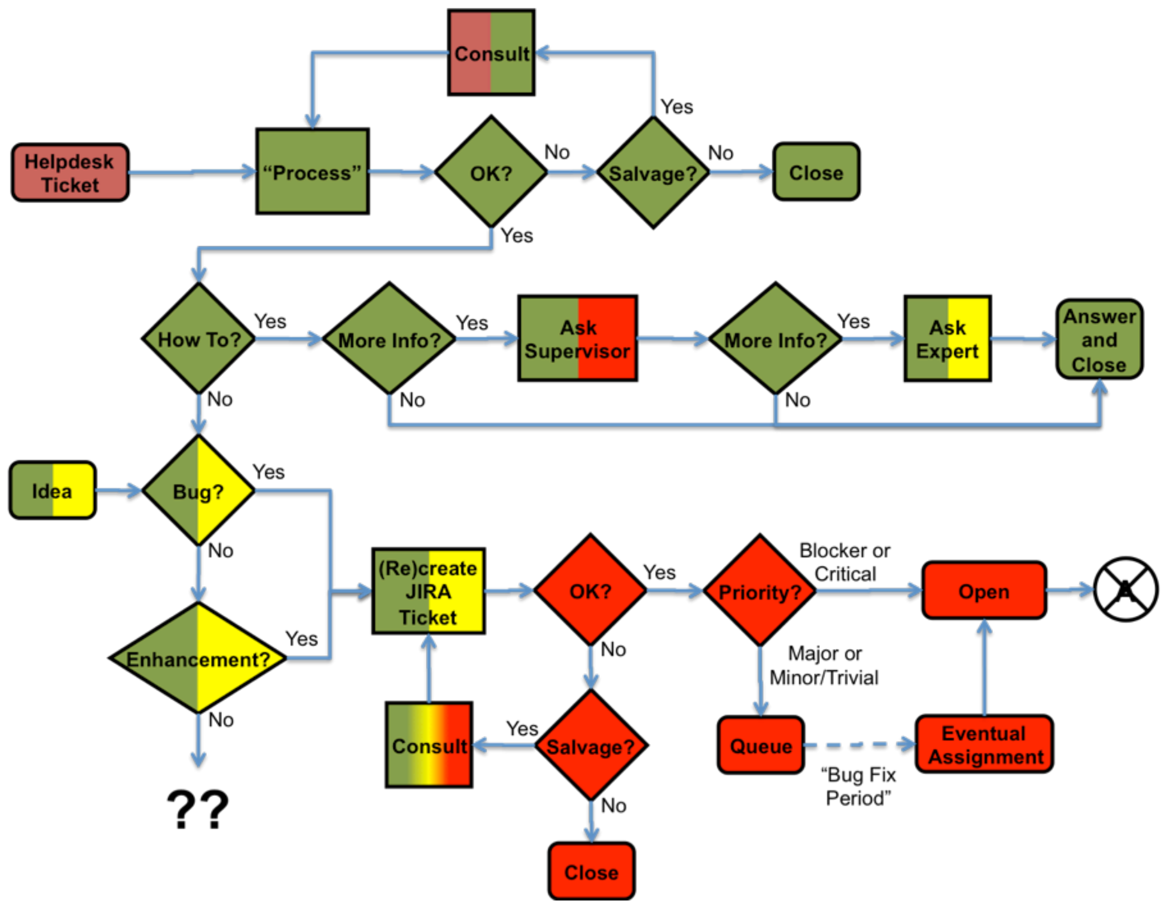
The JIRA ticket lifecycle can be described in terms of three connected flowcharts: Assignment, Development, and Closure. Assignment refers to the steps that lead to assigning a JIRA ticket. Development refers to the steps required for development. Closure refers to the steps required for testing and closing a JIRA ticket.

I explain the flowcharts further below, but first I must provide the color key for the flowchart boxes. They denote responsible parties.



A few boxes contain multiple colors, which denotes context-dependent co-responsibility. The standard font and outline color for the boxes is black, but many in the Development flowchart have different context-dependent colors.

Here is the Assignment flowchart:



As expected, most of the responsibilities fall to the support staff (helpdesk) and supervisor, while some fall to the user and developers. If you study the flowchart carefully, you will find that most steps are not too different from what we (try to) do now. The extra steps involve "front-end" improvements and clarifications of helpdesk and JIRA tickets, which I expect will minimize future problems. The support staff does not work for me, so I included only the minimum number steps based on their expected interactions with the CASA team.

There appears to be an attitude among reporters (I admit that I've been guilty, too) that goes like, "the developers are astronomers or astronomical developers who should be able to write code without hand holding," implying that spending a significant effort on front-end ticket descriptions is a waste of time. Good descriptions are definitely required, and the support staff and CASA supervisor will act as gatekeepers in that regard. On the other hand, we are supposed to be knowledgeable in radio interferometry and software development at some level, so descriptions must be sufficient without being excessive. One learns how to achieve the proper balance with experience.

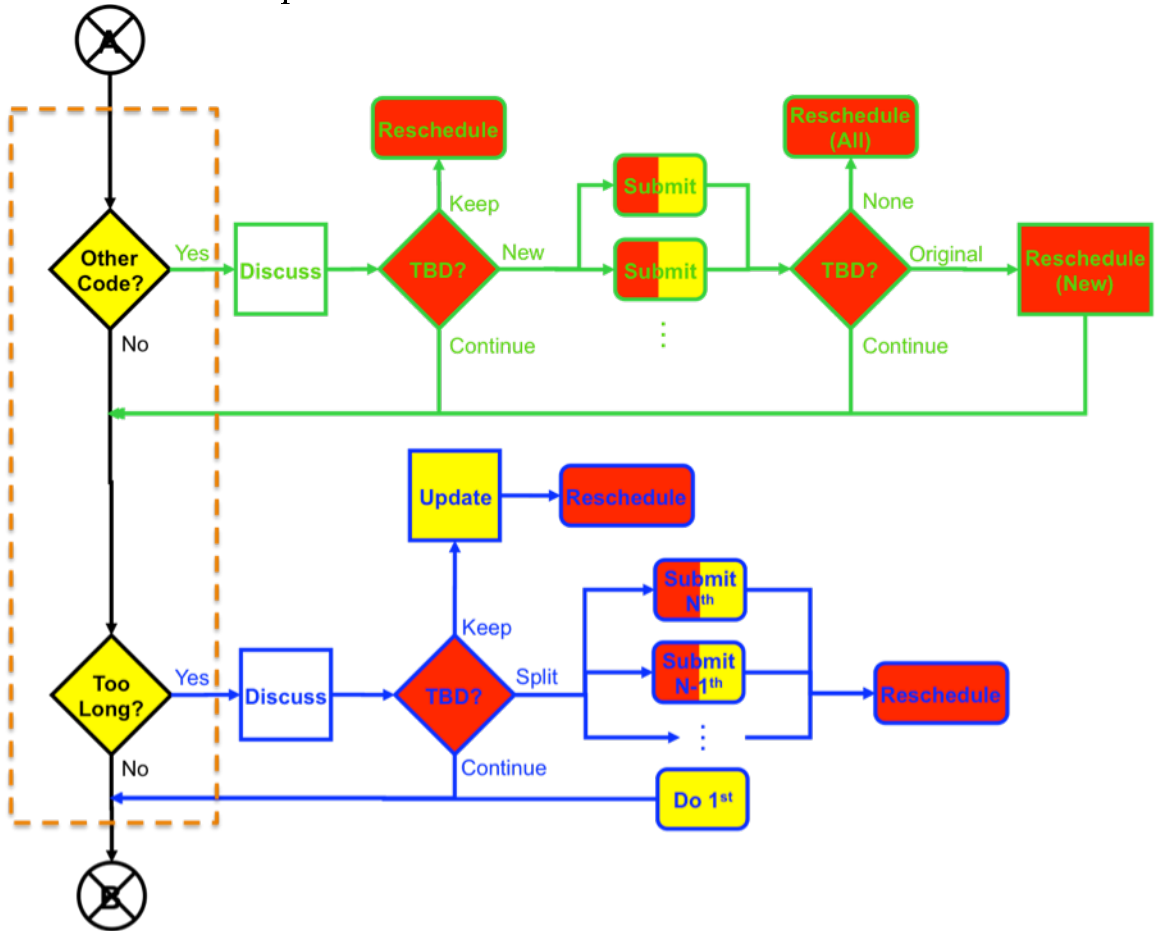
If the reporter is a user, he/she submits a ticket to the helpdesk. Duplicate tickets are closed. The support staff "processes" the ticket to determine if it is well worded. If it needs work and if it's salvageable, the user and support staff improve the ticket. Otherwise, the ticket is closed. "How To" tickets can be answered directly by the support staff. If more information is required, then inquiries can be sent to the supervisor or a developer.

If the helpdesk ticket is a bug or improvement, then the support staff creates a JIRA ticket. Note that the developers or support staff can create their own JIRA tickets independent of users (NB: the developer is involved only if he/she submits the ticket). The CASA supervisor then reviews the ticket. Duplicate tickets will be closed. Feature-improvement tickets will also be closed if they are inconsistent with long-term architecture plans (Solution 10).

If a JIRA ticket needs work and if it's salvageable, the reporter and the supervisor improve it. Otherwise, the ticket is closed. The supervisor may decide to submit new tickets (to capture dependencies and avoid breaking code) or split the existing ticket (to minimize the number of open-ended tickets), depending on the content. Those processes are shown explicitly in the Development flowchart.

If the resulting ticket has a Blocker or Critical priority, then it is immediately assigned to a developer. Major and minor/trivial tickets, on the other hand, are relegated to a queue and are assigned during bug-fix periods (Solution 8).

Here is the Development flowchart:



You may have noticed that there are no boxes corresponding to development and unit testing. I assume that these tasks can be performed anywhere within the dotted brown box. This flowchart appears to be complicated, but when explained according to groups of boxes the complexity is mitigated. It is valid for opened and reopened tickets. Tickets can be reopened by anyone.

The path that travels down the left side the diagram represents the ideal path of development, i.e., the ticket has been well described in the Assignment flowchart and the developer knows exactly what to do. He/she verifies that the ticket affects other code minimally or not at all and that it is not open-ended, and performs the development. The developer can ask the reporter more questions at any time (not explicitly shown).

The boxes with green and blue borders and letters, on the other hand, represent the hassles incurred when the assignment process fails to yield a well-written ticket. Moral: informal discussions in the assignment process avoid formal pain later in the development process.

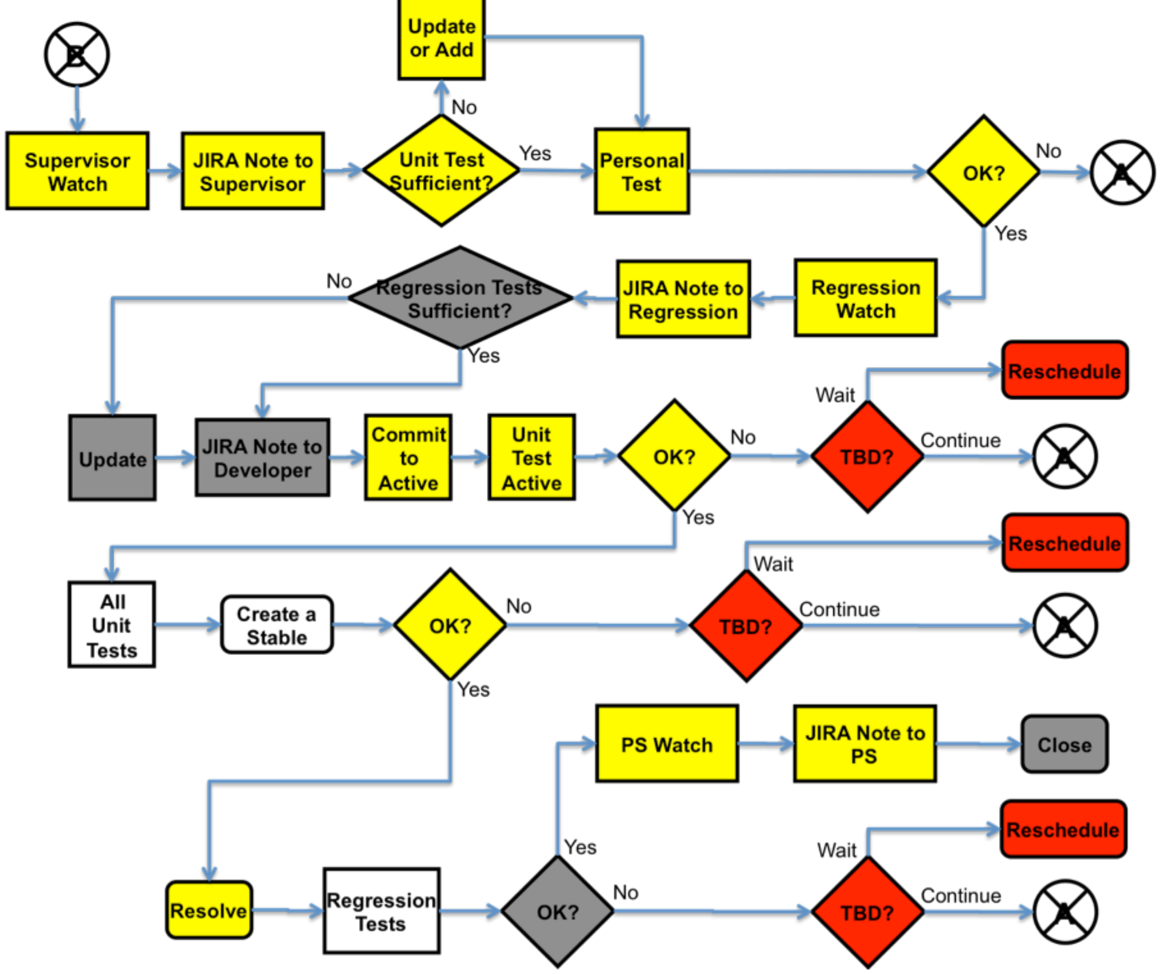
The green boxes are used to describe what happens when the developer determines that other code is significantly affected in an adverse manner. The group discussion provides input from experts in the other code. The supervisor then decides how to proceed: Keep everything as is and reschedule, create new linked or sub tickets, or continue. The new tickets can be created by the supervisor and/or the developer (TBD). The supervisor then decides whether to reschedule all tickets (including the original), reschedule the new tickets and continue with the original ticket, or continue with all tickets (not necessarily assigned to the same developer).

The blue boxes are used to describe what happens when the developer determines that the ticket should be divided into linked or sub tickets. As before, discussion among group members provides input from experts. The supervisor then decides either: 1) keep everything as is and reschedule (if progress has been made, the developer records it); 2) split the ticket into new linked or sub tickets (the supervisor and/or developer submits N-1 new tickets and reschedule, the developer continues with the first); or 3) continue as is.

Here is the Closure flowchart (next page).

Unit tests insure that the likelihood of a bug ending up in a stable build (and ultimately a prerelease or release build) diminishes as this flowchart proceeds, which will improve efficiency and relieve stress. Unit tests must be performed **using the data and scripts provided by the reporter (if applicable)**. The build process has been incorporated into the flowchart.

The developer notifies the supervisor that unit testing is ready before putting him on he watch list. This step is required because placing him on the watch list does not automatically inform him that the development has progressed to the point where unit testing begins.



The developer then determines whether the existing unit tests are sufficient to test the ticket. If not, the developer is responsible for updating them. Developers have not been required to perform this step before. It is now mandatory as part of our triage efforts.

The developer performs unit tests using his/her personal build. If unsuccessful, development continues, otherwise the developer notifies the regression lead (Solution 10) that the first unit tests are complete. Next, the regression lead updates the regression tests, if necessary. After the regression lead notifies the developer that he regression tests are ready, the developer commits code changes to the active branch, and the developer performs more unit tests there when the build is successful. If the unit tests are not successful, the supervisor determines whether development should be rescheduled or not.

All unit tests should be run on the active before a stable build is produced.

These tests will be run periodically. After a stable is created, the developer performs the unit test again (he/she may have several JIRA tickets to be tested after a stable is created). If unit tests are not successful, the supervisor determines whether development should be rescheduled or not for each ticket. Upon successful completion of the unit tests, the developer resolves the ticket.

The regression tests are run automatically. JIRA tickets that do not cause any of the regression tests to fail are closed by the regression lead, otherwise the supervisor determines whether development should be rescheduled or not. Before closure, the project scientist is notified of the closure so that he/she can maintain a list of documentation changes (Solution 15).

Concern 8: A large number of JIRA tickets with major or minor/trivial priorities are never handled.

Given the projected manpower estimates over the next few years for the developers, this problem will always remain (and will become more important as we proceed to public releases). The best we can do is continually and diligently deal with it. Closing all existing JIRA tickets and starting over is tempting, but I will not support this strategy because we lose a lot of hard-earned knowledge.

Solution 8: Regularly implement “bug-fix weeks” or “bug-fix days.” Schedule major and minor/trivial tickets according to completion time estimate, not priority, during bug-fix periods, to pick off the “low hanging fruit.”

Bug-fix weeks have been successfully employed once per development cycle. They are ideal for handling tickets with completion time estimates on the order of ~ one or two days. Bug-fix days have also been successfully employed, but they occurred more often. They target tickets with completion time estimates less than ~ half a day.

Since major and minor/trivial tickets do not significantly affect users compared to blocker and critical tickets, the priority is less important compared to the estimated completion time. Therefore, the estimated completion times will be considered before priority for these tickets during bug-fix periods.

In Solution 4, I mention that long/open-ended JIRA tickets may be split, if efficiency is improved. This strategy will make more tickets available for bug-fix periods.

It has been suggested to me that we might try bug-fix weeks dedicated to individual tasks. This idea has merit, but we must keep in mind that more planning will be required. I will ruminate on this strategy further.

It has also been suggested to me that an important way to convince users that CASA is a mature user-friendly package is to keep the on-line help up-to-date and well written. Therefore, I am motivated to implement one or two days per cycle as "help-fix days."

Concern 9: Compared to the cabal, the developers do not contribute a significant amount of advice on schedule priorities.

The cabal's main function is to set priorities. A few developers, however, expressed frustration that they have no say in such matters. We are wasting valuable knowledge and "in the trenches" experience. I agree with them.

Solution 9: The supervisor and architect will act as advocates for the developers when discussing priorities with the cabal.

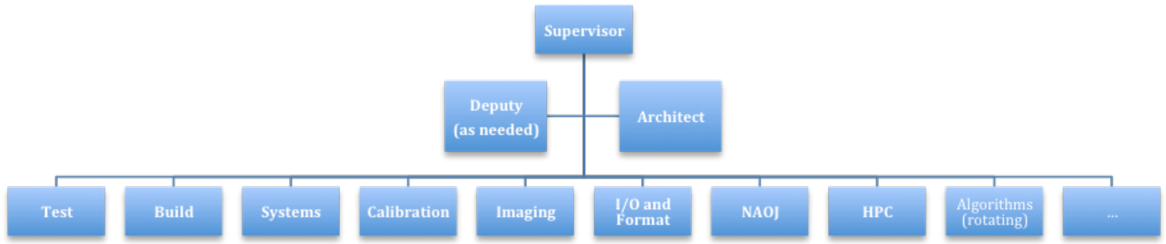
It is impractical for all developers to attend cabal meetings. Advocacy is the logical approach. Developers are encouraged to make suggestions to the supervisor and architect at any time.

Concern 10: The management structure of the supervisor and the developers is too flat, which makes ticket tracking difficult (distributed project management; Concern 1).

The flat management structure means that day-to-day management and long-term planning, nominally assigned to the CASA supervisor, is too much work for a single person, at least for now.

Solution 10: Develop a slightly extended management structure with leads. Obtain an "as needed" deputy (day-to-day management) and an architect (long-term planning).

Here is the proposed structure for the CASA developers (the cabal and project scientist are entities outside of this hierarchy):



The leads are "experts" and/or "liaisons." They **do not** represent an additional level of management. Experts will provide advice to those who work in their designated fields. Liaisons (NAOJ, HPC, algorithms) will be required to make brief reports to the supervisor on their respective activities. All leads should inform the supervisor when problems arise.

The build expert will have a significant amount of work to do for the next development cycle. That role will diminish in the future as the new build system stabilizes. The test liaison must also do a significant amount of work during the next development cycle, but there will be a continuing role as well.

The HPC liaison/expert will also become more significant during the next development cycle. Liaisons other than NAOJ may be required. The algorithms liaison, depending on the final form of the NRAO reorganization, will probably rotate among team members. The boxes in the organizational chart will be filled in the near future after consultations with the developers and management.

Because of the number of fields is comparable to the number of developers, any developer can be assigned a JIRA ticket in any field. Experts should answer questions about their fields, but at the same time developers should consider the time trade-off between asking questions and doing the research within the code.

Previous CASA reviews have recommended that someone perform architect duties. An architect normally produces the initial top-level design for a project. Since most of CASA has already been written, the role of the architect will be long-range planning to improve the architecture in terms of maintainability and reliability.

The architect will solicit suggestions from developers and interested parties and create a document that will be the basis for long-term development. Scheduling each long-term task will depend on internal/external priorities and size. The CASA supervisor and architect will meet weekly to insure that JIRA tickets are consistent with long-term plans. The architect will also attend weekly cabal meetings to provide technical assistance.

The CASA supervisor will be responsible for day-to-day management of the project. From time to time, he will be away from the office due to illness, vacation, or research (gasp). Unfortunately, daily management does not stop when the supervisor is unavailable, so an "as needed" deputy must be appointed. The deputy will meet with the supervisor on a weekly basis so that he/she can take over when required.

Concern 11: The six-month major release dates for CASA and ACS are identical, which means that responses between the two packages are delayed by a full cycle (or by half a cycle, if a patch is released).

Solution 11: Move the CASA R3.1 release date to 2010 October 15th and the CASA R3.2 release date to 2011 April 15th.

For Americans, these dates are best because they do not fall too near major holiday weekends or school vacations (don't complain about tax day, you have 3.5 months to finish your taxes). Between the CASA R3.0 and R3.1 releases, two patches will be required: 1) an "EVLA" patch on 2010 March 1, and 2) a "workshop" patch on 2010 June 1. Getting a release ready in time for the Technical Readiness Review is not so critical (the wish list will just be longer).

These suggestions are tentative at the present time, depending on external circumstances. Whether they are adopted or not, detailed release schedules will be required.

Concern 12: Reorganization is difficult while the project continues normally.

Analogy: Performing an appendectomy while the patient is living his/her normal life. Even under ideal circumstances, this procedure is ill advised and painful (even with local anesthetic). Scheduling a brief downtime is the best strategy for balancing schedule and sanity.

Solution 12: Declare a brief moratorium on JIRA ticket work and code modification. The supervisor and developers will work to implement the policies of this road map and properly plan for the next patch(es) and release.

I have been allowed to impose a four-week moratorium (I originally asked for more time, and someone has even suggested a complete development cycle). It will occur between late January and late March 2010, in conjunction with the developers' meeting that occurs approximately every two years. Emergency bug fixes will be allowed during the moratorium, but the CASA supervisor reserves the right to postpone them until after the moratorium.

We will use this moratorium for reviewing existing JIRA tickets, reviewing regression tests, redesigning the build system, creating teaching materials for new developers (and me!), and doing as much middle- and long- term planning as possible. The results of this JIRA ticket review include (but are not restricted to): 1) removing, modifying, or combining tickets; 2) reevaluating priorities; 3) changing assignees; and 4) improving description texts. Other administrative tasks could be performed as well.

<rant>
The VLA will shut down for almost two months in preparation for EVLA observations. No one raises any objections because this idea makes perfect sense. Since CASA has come under new management, it would also make perfect sense to temporarily stop active development for a while to make overdue procedural changes that are becoming even more critical because we are finished with β testing and formally exposing ourselves to the user community.

Given external pressures, objections have been raised about the timing of the moratorium. I am not surprised. Data-reduction software is the tail while

the rest of the instrument (hardware, embedded software, etc.) is the dog, which means that we get wagged (and often get crapped on). It is not fair, but then again where is the guarantee that life is supposed to be fair? It's the same for every project. We'll deal with it as best as we can.
</rant>

I feel better now …

Events from the last few weeks have reaffirmed my conviction that some sort of moratorium is required and that it should occur as soon as possible. We may have to be clever when it comes to implementing it, though. Perhaps I should schedule individual staggered moratoriums for the developers? Something else? I will try to implement the best possible strategies.

No matter what strategy I choose, we must get the highest possible benefit to make the exercise worthwhile. Therefore, I will create a detailed list of action items and assignees, including a precise schedule and taking into account external pressures, after the 2009 December 1 release.

Concern 13: Developers sometimes take vacations too close to a release date.

The developers work very hard and keep long hours. They deserve their vacations. Important development and testing has been delayed and scheduling made difficult because of vacations.

Solution 13: Ask the developers to schedule vacation time for the first four months of the development cycle.

Of course, this rule cannot be absolute. Friends always get married at the most inopportune times, the ocean temperature for that once-in-a-lifetime island vacation is warm only certain times of the year, conferences do not follow our schedule, etc. Family emergencies that require a significant number of personal days, of course, have priority over everything. An occasional long weekend is acceptable.

Remember: Try to give as much advance warning to the supervisor as possible. With the new team structure (Solution 10), members of the "goto group" should be able to cover most of the time. Do unto your colleagues as you would have them do unto you.

Concern 14: The developers who are scientists may not have significant time for in-office personal research.

Staff scientists are allotted 25% in-office research time. They should not have to spend 40 hours per week on service duties and then use their personal time for research on a regular basis (although many, including myself, do it anyway because they enjoy their jobs).

Solution 14: If possible, try to accommodate scientists so they can perform some in-office research during the standard 40-hour week.

During each six-month development cycle, scientists are entitled to approximately six workweeks of research. Allocating single days is useless because it is impossible to rapidly "shift gears" and get any significant work done. The optimum allocation is ~ a few workweeks, so two or three of these allocations can be scheduled during a single development cycle.

In-office research time should be performed during the first four months of the development cycle. If service demands on a developer do not allow for enough in-office research, it must be noted in his/her PEP. If non-office research is performed, it must also be noted in his/her PEP. Excellence in both service and research should be rewarded.

Concern 15: Steve Myers will shortly step down as project scientist after seven years of service.

The project scientist is an important single point of contact for technical advice, from a user's perspective, for any scientific project. He/she must be an active scientist, be knowledgeable about radio interferometry, and be interested in using the software. He/she is responsible for maintaining the CASA cookbook and other documents.

Solution 15: Appoint a new project scientist.

We are investigating different options.

NB: Steve told us that he was spending almost 100% of his time as project scientist. The new project scientist will probably be spending 25% of his/her time on those duties. In other words, we are not breaking even and we are effectively losing three people. Therefore, the job requirements for the new project scientist must be chosen carefully.

Concern 16: The CASA cookbook has elements of a reference manual.

Cookbooks are very beneficial for users to see how the software works. They provide a number of examples that can be adapted to real-life data reduction. Adaptations can be devised by looking at details contained within reference manuals.

Solution 16: The details should be moved to the task or tool reference manual.

The new project scientist will maintain the CASA cookbook. The task reference manual, tool reference manual, and on-line help will be assigned in the future. It may be possible to take on-line help and merge it automagically with other manuals so that text only has to be managed in a single place.

Concern 17: Who decides whether the software is ready for a release?

This concern is related to proper assignment of responsibilities (Concern 1).

Solution 17: The CASA project scientist and supervisor are responsible.

Decisions can also involve release postponements as well as patch dates.

Concern 18: Existing regression tests may not cover all use cases.

Our regression tests have been very useful for finding errors and fixing code. We must make sure, however, that all code is tested sufficiently, including individual tasks (using non-default parameter values) and end-to-end data reduction.

Solution 18: Find missing tests, if any, and implement them.

The JIRA ticket that groups all regression tests together is CAS-686. The regression test lead is in charge of maintaining the existing scripts and creating new ones.

With the help of designated scientist users, the regression test lead will create a document of use cases and devise a plan to create and manage use cases, based on EVLA and ALMA requirements. These scientists, and others, will also contribute scripts and data. Regression tests for individual tasks will be created and/or improved to exercise various combinations of input parameters (this cannot be done with end-to-end tests because they take too much time).

Special regressions for high-performance computing should be performed periodically (perhaps weekly). A plan for such tests must be developed, which includes data, scripts, hardware configuration, etc. These considerations will migrate into a plan for how external users interact with the cluster.

Interactive tasks cannot be run by regression tests. They must be run manually. A checklist should be created and maintained (Solution 7) so that interactive tests can be run periodically (perhaps monthly).

Concern 19: Code is submitted by non-CASA developers.

We want CASA to gain a wide acceptance among the astronomical community. One way of gaining acceptance is allowing users to develop modules that can be included in releases. Unfortunately, some of the code has not been of the best quality.

Solution 19: Create training and coding standards documents for CASA development. Pair each non-CASA developer with a CASA developer for consultation. Code that does not meet our standards will not be included in releases.

The training documents should be created for the developers' meeting. The non-CASA developer is responsible for unit testing that must be approved by the consulting CASA developer.

The non-CASA developer is also responsible for providing a script to the regression test lead (end-to-end and tool/task component level), who verifies that it is sufficient and runs satisfactorily. After acceptance, the CASA group takes responsibility for the code and regression scripts. The code can then be included in the next release.

We like the scenario where a group develops algorithms, develops the CASA code (most likely C++ and tool level), writes a report, and sends the complete package to us. We can then merge the code if and when we see fit. The more limited scenario where a group only develops algorithms is less desirable, since it involves much more work for CASA developers as well as algorithm specialists.

It has been suggested to me that useful scripts should be collected to a publically accessible repository web page. I support this idea, but there are details that must be considered. For example, who decides which scripts are web-page worthy? Who maintains the web page (not a CASA developer!) and where will it be located? How will the web page be organized? These questions are beyond my purview as CASA supervisor (although I am willing to participate in discussions if I can help).

Concern 20: There are a number of aips++ and CASA mailing lists in use.

Some mailing lists are obsolete. For example, aips2-naug is the old cabal mailing list containing many names that are no longer on the project. Also, casa-request is obsolete because of the helpdesks (we want to discourage "back doors").

Solution 20: The mailing lists should be reorganized.

We want to insure that the communication among group members remains strong. As a secondary benefit, mailing lists can be used as searchable repositories. We should employ the minimum number of mailing lists, to avoid confusion. Obsolete aips++ and CASA mailing lists should be removed.

Developer lists:
casa-staff – Developers who can be expected to interact directly with the CASA supervisor.
casa-devs – All developers. New developers can be added to this list for light (python, toolkit) or heavy (C++) development at the discretion of the CASA supervisor.
casa-weekly-report – Used to submit weekly reports to the CASA supervisor.

Suggested lists:
casa-cabal – The cabal.
casa-help – The helpdesk staff who deal with CASA.
These lists would be useful for the developers or CASA supervisor to convey important information to the cabal or helpdesk. I will not implement them until I receive an OK from the relevant parties.

Mailing lists for CASA users should be encouraged. For getting help on CASA usage, they may be able to provide a quicker response than the helpdesk (and perhaps even relieve some of the burden on the helpdesk). They should not deal with bugs or enhancements because they are reserved for the helpdesk.

Since CASA is most associated with NRAO, user mailing lists should be hosted and maintained by other organizations. If NRAO is formally involved, there is an implicit expectation that an NRAO person on the mailing list will answer a question. Disclaimers must be included on the mailing list homepage and when a subscription is confirmed. There is precedent from other software packages.

It may turn out that developers or support personnel subscribe to external user mailing lists and answer questions. Again, there is precedent for this behavior from other software packages. But, there should be only one formal NRAO interface to the support groups – the helpdesk.

Concern 21: Presently, users only have access to the latest release or patch.

If a user submits a bug fix or improvement suggestion, he/she will not have access to the fix or suggestion until the next release or patch (assuming it is fixed for the next patch).

Solution 21: Make stable builds available to users.

Stable builds should be distributed only to experienced users, along with the disclaimer that other things may be broken. The helpdesk support staff will make the decision on whether the user is experienced enough.

Concern 22: What is the procedure for supporting new operating systems and removing old operating systems?

This concern is related to proper assignment of responsibilities (Concern 1).

At present, we are supporting very old Linux operating systems. Also, we decided to support Mac OSX 10.6 relatively late in the present development cycle.

Solution 22: At the beginning of each development cycle, we must decide which operating systems are supported and removed for the following release.

Suggestions will be solicited from the developers, but the CASA supervisor and the systems lead will make the ultimate decision.

Concern 23: How are interactions between developers and the helpdesk managed?

The CASA developers have received relatively few JIRA tickets via the helpdesk. After the public release, we expect to get more.

Solution 23: The CASA supervisor must discuss these tickets with helpdesk support staff.

Such discussions will be even more important since the helpdesk software will be upgraded in the near future.

3.0 Conclusion

To those of you who might think that adopting these proposals will take too much time from our existing duties, I ask this question: Where is the time we are supposedly saving because we haven't implemented uniform procedures? The "time tax" we are charged due to constantly breaking code is no different than the "time tax" charged by a rickety build system. Effective day-to-day management and long-term planning are the keys.

After the R3.0 release, I (with your help) will begin to implement the solutions outlined in this roadmap. I will make a schedule for the JIRA ticket moratorium and assign other administrative duties. The moratorium will coincide with the developers' meeting in 2010 (date TBD; it may be coordinated with the simulation meeting).

Comments are welcome. I end this document with a relevant quote from my time in Germany:

Politics is the art of the possible. – Otto von Bismarck

I ask for your support.

A. My Personal History and How It Relates to CASA

Many of the developers have been working on aips++/CASA for a long time. I even contributed a bit of aips++ code. Others have worked on other projects in academia and industry. In this brief appendix, I wanted to share the experience of my previous software management job. I present this story to show what is possible even under adverse conditions, not to glorify me.

PRIMA is a near-IR dual-feed interferometer used to search for extrasolar planets via narrow-angle differential astrometry. The ESPRI consortium was trying to fill the work package manager position for PRIMA's astrometric data-reduction software (ADRS). I had previous experience in software design/development, optical interferometry, and exoplanet detection techniques, so I was a viable candidate. After a telephone interview with the consortium, they offered me the job and I moved to the

Universität Heidelberg (Landessternwarte) and the Max-Planck-Institut für Astronomie.

My predecessor and his team did some good work. Unfortunately, this work was not well organized, and the consortium was worried that the ADRS would not be finished in a satisfactory and timely manner. I was given *carte blanche* to do what was necessary to get the project back on track.

I found that the software itself was in sorry shape, so I pitched it. Read that sentence again if you don't believe your eyes. I estimated (and hoped and prayed) that the time remaining before the due date was sufficient to take this drastic step. The design document, on the other hand, was pretty good. If the previous team had worked on the design document a little more and actually followed it, I am sure that there would have been fewer problems.

I removed extraneous sections from the design document and clarified the language. My staff and I then added more material. I created a rudimentary development flow, schedule (based on completion time estimates for the high-level modules), and draft user manual. I updated all documents, including the schedule, as we learned new things.

My staff was small, consisting of a full-time developer, a full-time support astronomer specializing in near- and mid- IR interferometry observations, plus several part-time astronomers specializing in atmospheric turbulence and dispersion, astrometry, and instrumentation. The developer, who would walk on water for me, devised unit tests in addition to his development duties. The support astronomer, who really understood data reduction, independently devised and developed system tests as new modules were finished.

The strategy of tossing all existing software and concentrating more on the design was definitely a gamble on my part. I decided to go that route because of my third favorite Lincoln quote:

Give me six hours to chop down a tree, and I'll spend four hours sharpening the axe. – Abraham Lincoln

Fortunately, we succeeded. Despite restarting the development process with no time extension, the team produced an initial version of the software months ahead of schedule. Good planning was the key. Something could be said for my diplomatic skills as well. The ADRS team, with a little help from me, was brilliant. I want the CASA team to be brilliant, too.

I realize that ADRS and CASA are two very different projects. CASA is significantly larger in terms of software and staff size, making it more of a challenge to manage. CASA also has a lot more "inertia."

The two instruments we are required to support will be coming on-line shortly, so our options are limited. Restarting development from scratch is completely out of the question, which means that the operative words for this project in the foreseeable future must be "triage" and "refactoring." With the appropriate day-to-day management and long-term planning, of course.