SCARAB: DCNS-Dev Bug Tracking Requirements and Proposed Process

Richard Basch, Mark Curby, Craig Fields,
Bruce Lewis, Andy Oakland

June 17, 1993

Introduction

DCNS Development does not currently have a formalized process for tracking bug reports. This means that some reports may not be making it to the correct people, and that others may be being lost. It is hard for developers to search through the outstanding bug reports.

This document presents an enumeration of the requirements and "desirements" of the users and developers and a proposed process flowchart for tracking bugs within the DCNS Development group.

The process being presented is intended to meet those needs, does not depend on any specific individuals, and is being designed so that it can be put into action by September 1, 1993. However, it is recognized that this process may not be suitable for certain specialized reports, such as security problems that affect the authenticity of the environment, and in such cases, discretion and appropriate action is expected of those handling the problem report.

System Functionality

Requirements

Based on input from various people in DCNS and CSS, the following list of requirements was developed for the new bug-tracking system.

- The system should encourage bug reporting. It should not be an onerous task for users to submit a bug report; the system should allow for input from various sources, such as from OLC, electronic mail, telephone calls, or special clients. Users must be able to submit reports from any platform via electronic mail, and users should not have to be burdened with figuring out where to submit a bug report for a particular package; the bug reports should be centrally channelled and disseminated to the appropriate development group. In addition, the system should be friendly, and should encourage users to assist DCNS Development by reporting problems.

- The users should receive polite acknowledgments for their problem reports and be kept informed about the status of their reports. The status should include enough information such that the users understand where in the bug-tracking process their reports lie (a process diagram should be kept online for users to reference). A bug identification tag should be given to users so that they may ask for updated status on their reports at any time. In addition, the users should be notified when their reports have been resolved and when those resolutions will come into effect. Possible resolutions and notifications might include that the bug will not be addressed or that it has been fixed for the next release.

- Any workarounds that exist for a problem should be reported to the user and to the front end support groups, such as net-help and the consultants. The workarounds should be kept along with the problem report in case future problems of a similar nature arise.

- The bug tracking system should record a full history of the bug, any workarounds, and all action that has been taken. It must be possible to report the current status of a bug, all the steps that have been taken to address the problem report, and what still needs to be done to bring the problem to a resolution. All history information, submissions and developer comments should be archived.

- Access to the database of problem reports should be available to IS support organizations, such as CSS and NSS, and some users would also like to be able to access the database directly to determine the status of a bug or see if a bug has previously been reported. The database should be easy enough to use that the average user can find what they are looking for. All the information should be publicly available, unless other factors mandate that this process cannot be used for tracking the bug report (e.g. security holes that compromise the authenticity of the data on the system).

- It must be possible for other groups to submit reports to the DCNS Development bug-tracking system. Electronic mail is seen as a minimum requirement to satisfy this. Also, the system should allow us to forward reports which come in to DCNS-Dev, if action needs to be taken by another group. For example, it should be possible to forward a problem report to the appropriate vendor of a third-party package, or to other groups who maintain the software, such as SIPB. The system should keep track of forwarded reports to see that some action has or will be taken.

- A real database should be used so that people can access the database in the following ways in an expedient manner:

  - Report all bugs for a particular product/version number.
  - Report all bugs for a particular platform or operating system.
  - Report bugs by priority or severity.
  - Report who is responsible for a particular bug.
  - Report the status of a particular bug.
  - Report everything about a particular bug.
  - Retrieve records based on queries involving multiple keywords.

- Review all bugs.
- Access information about resolved bugs.
- Search for bug reports by date or submitter.
- Retrieve all bugs assigned to a developer and their disposition (eg. under investigation, fixing, auditing)
- Review bug reports by their current status (such as those that have not yet reached the "assigned" state).

- The developers will need to be able to manipulate the database in various ways:
  - It must be possible to link related bug reports together.
  - The developer must be able to assign a bug to himself or others.
  - It must be easy to add comments to a particular bug entry.
  - All interactions with the user must be logged.
- The system must include time-based triggers to alert "management" of bugs which have been sitting too long without action and may have been forgotten. This is to ensure that there is timely resolution of all reports and that "no bug is forgotten."
- Specialized clients and electronic mail templates (e.g. *sendbug*) must be provided to the users so that most of the common information required by a developer (i.e. system configuration, product version, how to reproduce the problem) to analyze a bug report is included with each bug submission.
- The system must minimally run under UNIX with a tty interface so that Mac and DOS developers can use *telnet* to access the database.
- The process must deal with suggestions and usage questions that come in to any bug reporting system, e.g. TechMail should use comm toolbox, or how do I convert my existing Unix mail files into TechMail. There should be a plan for dealing with these.

Desirements

It would be desirable to also have the following features in the system.

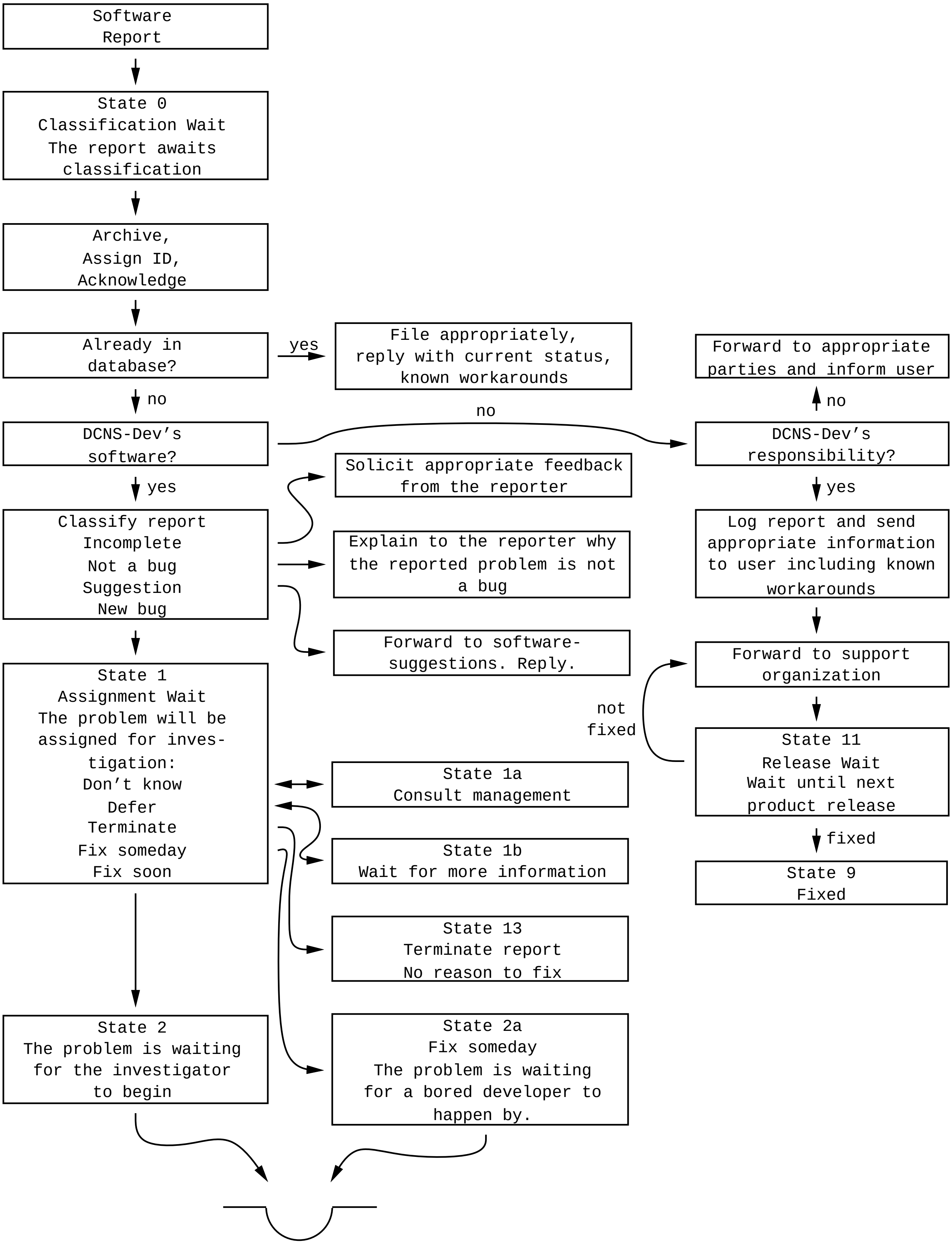
- The system should provide the hooks to allow it to be integrated into a larger IS help desk system in the future.
- The database should allow querying for the number of bug reports received for a particular bug and what users have reported problems.
- It should be possible to perform full text searches (e.g. WAIS) on the bug database.
- The database should be easily accessible to all clients - Windows, Mac, and Unix.

- The system should be able to track bug reports that affect multiple applications. For example, a fix to the Mac Kerberos library would affect TechMail, MacZephyr, and MacDiscuss.
- There should be an archive of old resolved bug reports, but these should not clog up the active database.
- Bugs should be recorded in a real database, with fields, indexing, and ability to search for anything by field or combination of fields. It should be able to store large amounts of information and still be accessible in a reasonable amount of time.
- The system should distinguish between bug reports and error lists. An error list would be a list of confirmed bugs for a product or platform. There might be multiple bug reports for any particular error list entry, while other bug reports might not result in an error list entry at all, e.g. because they are not reproducible, or a bug. (Some call bug reports “complaints” or “problem reports” and error list entries “defects.”)
- The system should allow for the gathering of resolution statistics, such as the mean-time between a submission and problem resolution.

Process

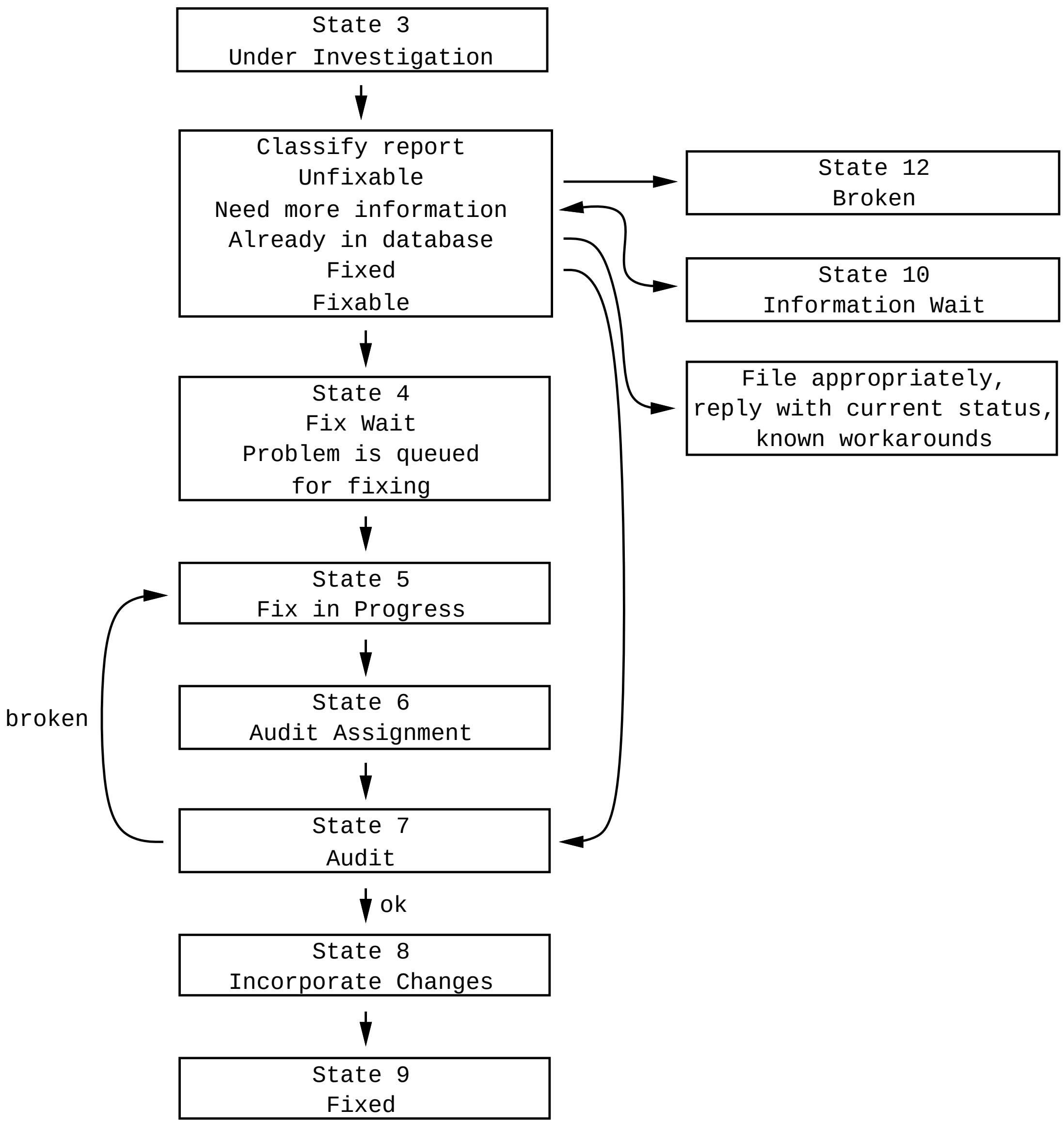
The attached flowchart describes the general process by which a bug report should be processed. It is important to note that steps can be skipped or done together, as appropriate for the particular bug. For instance, during the analysis stage, the bug might be fixed, in which case, the bug will flow quickly through the system. Also note that there is a simple, main path through the process progressing from state 0 through state 9. Newly reported bugs in our own software that are important to fix and fixable will follow this path. Other paths in the process flowchart exist as guidelines to handle exceptions. While all exceptions are not presented in the flowchart, what the flowchart does present should give enough information to determine the right thing to do.

A *jik* committee will be appointed, composed of two developers, and its membership should rotate on a monthly basis (each person should serve two consecutive months, but the terms do not start at the same time). This committee is responsible for triaging all the bugs and ensuring no bug is forgotten once it is in the system. The *jik* committee should meet weekly to discuss the triaging, and especially during the first year to train the other person, since most of the developers have never done bug triaging. The committee can decide within itself how they wish to handle the bugs, but it is expected that someone who is just performing triage for the first time will be meeting with the senior person more frequently to discuss the various problems.


Software Report
State 0
Classification Wait
The report awaits classification
Archive,
Assign ID,
Acknowledge
Already in database?
yes
File appropriately, reply with current status, known workarounds
no
DCNS-Dev's software?
no
DCNS-Dev's responsibility?
no
Forward to appropriate parties and inform user
yes
Log report and send appropriate information to user including known workarounds
yes
Classify report
Incomplete
Not a bug
Suggestion
New bug
Solicit appropriate feedback from the reporter
Explain to the reporter why the reported problem is not a bug
Forward to software-suggestions. Reply.
Forward to support organization
not fixed
State 11
Release Wait
Wait until next product release
fixed
State 9
Fixed
State 1
Assignment Wait
The problem will be assigned for investigation:
Don't know
Defer
Terminate
Fix someday
Fix soon
State 1a
Consult management
State 1b
Wait for more information
State 13
Terminate report
No reason to fix
State 2
The problem is waiting for the investigator to begin
State 2a
Fix someday
The problem is waiting for a bored developer to happen by.

Tab A

Slot B


State 3
Under Investigation
Classify report
Unfixable
Need more information
Already in database
Fixed
Fixable
State 12
Broken
State 10
Information Wait
File appropriately,
reply with current status,
known workarounds
State 4
Fix Wait
Problem is queued
for fixing
State 5
Fix in Progress
broken
State 6
Audit Assignment
State 7
Audit
ok
State 8
Incorporate Changes
State 9
Fixed

State 0: CLASSIFICATION WAIT

At this point, the report has been automatically archived and given a report identification number, but the report is not yet considered "active" in the database. The submitter has been notified that the report has been received and given the report id number for future correspondence (everyone should use this id to reference the report).

During this state, the *jik* committee will look at the report to determine what should be done with it next. One of three things may happen:

1. The software in question is not the responsibility of DCNS-Development, and will be forwarded accordingly.

2. A similar report has already been filed, in which case this report will be linked to the original report and the user will be notified of the current state and added to the appropriate interest lists, if desired.

3. The report has not been made before and must therefore be classified. The various ways in which the bug may be classified are:

- Incomplete report - the report was either incoherent or inadequate. More feedback from the user is required.
- Not a bug - the report describes the intended behavior of the product.
- Feature request - the user has not reported a bug; the report describes a potentially desirable behavior for the product.
- New bug - the user has reported a previously unlogged bug.

The bug report is then filed into the database along with all other information that should be tied to the report - the program or programs suspected of causing the problem, information pertaining to the reproducibility of the problem, and any other information that the triage team believes may help the developer in his efforts to fix the bug.

When the report has been classified, the user should be told the report's classification and any other classification dependent information, such as why the product behaves as it does or asking for additional information to complete the bug report, or what workarounds exist for their problem.

State 1: ASSIGNMENT WAIT

The *jik* team will decide one of the following fates for the bug report (like most lists, this one isn't exhaustive):

- Don't know: Death by bureaucracy
  The *jik* team for some reason cannot clearly determine a fate. Higher management will be consulted.

Reports awaiting such consultation live in state 1a. Once this consultation is obtained, the report is effectively in state 1 for determining what state it is to enter next.

- Defer: Death by slow torture
  The fate decision is postponed until a later time or event when more information becomes available. The event might be the completion of an investigation, the decision of Athena to support NeXT machines, the freezing over of hell, etc. State 1b holds reports so doomed. Once sufficient information is obtained, the report is effectively in state 1 for determining what state it is to enter next.
- Terminate: Death by firing squad
  The program or platform in question is to be or has been decommissioned, so fixing the problem would be a waste of effort. Reports with this unlucky fate meet their end in state 13.
- Fix someday: Death by mortal combat
  The bug report is pooled with other otherwise unassigned (low priority, no associated project) reports, to be attacked at the whims of developers who desire the challenge or practice. Or have a particular grudge. The arena for meeting these challenges is state 2a.
- Fix soon: Death by Daleks
  An appropriate person or project is assigned to exterminate the bug. The staging area for this machinery is state 2.

In the case of potentially long wait states (death by slow torture and mortal combat), the reporter should be notified of the particularly brutal nature of the execution. It is expected that most reports that reach state 1 will either take the path of mortal combat or the path of the Daleks.

State 2: INVESTIGATION WAIT

The report has been assigned to a person or project team for investigation. All we can do here is *sigpause(0)* and wait for a report from the investigation team or prod the investigation team if the bug is "forgotten." The report sits in this state until the investigation begins.

A variation of state 2, state 2a, is different from state 2 in that a report may remain in state 2a for considerably longer without being reviewed. State 2a reports have been deemed much less important than those in state 2, and no specific group or individual has been assigned to them. Instead, state 2a reports lie around waiting for someone to randomly decide to investigate them.

State 3: UNDER INVESTIGATION

Another round of classification occurs by the assigned investigation team. More in-depth thought and investigation go into the problem in order to identify the following characteristics:

- existence
  Attempt to verify via reproduction, code inspection, or theoretical considerations that the reported problem is possible.

- location
  Attempt at least to verify what program is causing the problem, if not exactly what code. It may be, however, that this is half the battle, and the difficulty should be deemed "hard," deferring further location to a later state.
- severity
  How dire are the consequences of encountering this problem?
- popularity
  How frequently is the problem tripped over? How important is the functionality it disables?
- difficulty of fixing
  How easy does it look like it would be to fix this problem?
- challenge of fixing
  How much fun will it be to fix this problem?
- banality of fixing
  How much would it suck to have to fix this?

Investigation of the problem may reveal any of the following:

- The problem is hopeless.
- The report does not identify a new bug belonging to DCNS-Dev after all; it may have traced the problem to an OS bug, a bug already entered in the database, etc., in which case the new information should be kicked back up the system to the tracker to be dealt with appropriately.
- There just isn't enough information to go on yet; the report may be suspended until more information arrives. State 10.
- The process of investigating the bug has resulted in a fix. Go directly to state 6. Do not pass go. Do not collect $200.
- The problem is fixable.

In the last case, come up with a priority to assign to the fixing of the particular problem based on the previously described characteristics.

The priority will then be entered into the database. If there were any non-obvious facts or decisions made in deriving the priority, those facts and comments should also be entered.

If the state of this report has changed interestingly since the last the reporter has heard, or there's an expected time for the fix, let the reporter know.

State 4: FIX WAIT

The problem is hopefully fairly well defined. The problem has been prioritized. Here, nothing happens; the problem is waiting for its number to come up.

State 5: FIX IN PROGRESS

Someone is working on fixing the problem.

State 6: AUDIT ASSIGNMENT

A fix has been submitted. At this time, the *jik* team should assign two people to independently audit the change, if no other audit procedure has already been used.

State 7: AUDIT

The assigned auditors should verify the fix is adequate. They should perform a code inspection, recompilation from source, and any other tests they feel necessary to ensure the fix is proper.

State 8: INCORPORATE CHANGES

The changes should be incorporated by the appropriate person on the project or in rel-eng, depending on the distribution mechanism of the package. For instance, software that is to be found on an Athena system pack should be submitted to rel-eng for incorporation.

State 9: FIXED

Hooray!!! Tell the user about it and when the fix will be available.

State 10: INFORMATION WAIT

The problem was under investigation, but more information is needed (via bug reports) to proceed. This is due to problems that are not well enough defined or not yet reproducible.

State 11: RELEASE WAIT

We are waiting for the next vendor release to see if the problem has been "fixed in the next release." If not, we send the report to the vendor again and continue to sit in this state.

State 12: BROKEN

This problem will never be resolved, or at least is not expected to be addressed by DCNS-Development.

State 13: TERMINATED

The problem report is irrelevant due to decommissioned software or platforms.