Quality Control in a Multi-Platform Multi-Product Software Company

Robert B. K. Dewar
Ada Core Technologies

Ada Core Technologies is an international company who produce and support the GNAT Professional family of Ada language tools: software suites comprising compilers, debuggers, integrated development environments, supplemental tools and packages, and other components. The company has been in business since 1994 and currently markets its products on platforms such as Windows, UNIX / Linux, and the Java Virtual Machine, and on embedded systems such as Tornado/VxWorks and LynxOS. With one or two major releases of each product annually, the company's livelihood depends on the quality of the software and its support. In many ways Ada Core Technologies faces the same issues as other growing software organizations, and this article is intended as a "case study" showing how a combination of automated tools and human management can help meet the QA challenge.

The report tracking system

One of the most important services provided by Ada Code Technologies is its support. Even if GNAT and all its tools attained zero defect status, this would be insufficient if the quality of the company's support services were not maintained. This is because the majority of reports from customers fall into one of the following categories:

1) Queries about the use or installation of GNAT or its tools, and, in particular, queries about the best way to use GNAT.
2) Queries about the semantics and usage of Ada 95, ensuring portability, correctness, and efficiency of code.
3) Claims of apparent defects, which turn out on closer examination to be reflections of incorrect programs or incorrect assumptions.
4) Requests for enhancements to the current technology.

In addition, since GNAT and its toolset are in a constant state of development, zero defect status is not in fact achievable, and there are reports in one further category:

5) Reporting of actual defects in GNAT or its toolset.

There are quality requirements for support:

a) rapid response
b) accuracy and completeness of responses
c) reliability of new versions if reports result in modifications to the technology.

To achieve these goals, Ada Core Technologies maintains a report tracking system shared across the company. When a report is received at a designated email address, it is logged automatically into the tracking database. An incident tracking number (TN) is assigned, and then the report is broadcast to all engineers who may be able to offer assistance.

If an engineer responds immediately, achieving goal (a), then the response goes back to the customer *and to all the other engineers*. This means that the response is reviewed for accuracy by everyone else who understands the issues. In the rare cases where adjustment or clarification of responses is needed, these adjustments and clarifications are transmitted to the customer immediately. This ensures that all responses are reviewed. Note that the response is not delayed for this review, because in practice it is extremely rare for a first rapid response to be inaccurate.

All responses and clarifications, as well as follow-ups from the customer, are automatically tracked by the tracking system, and placed in the database. This ensures a complete and permanent record of the transactions related to each report. Any engineer who has something to add can first read the entire transaction record so that they are aware of everything that has been said. This helps to ensure consistency and avoid duplication of responses.

If the response to a report is adequate following the above scenario then it is closed on request of the engineer who has assumed responsibility for the response, and the report is associated with this engineer's name. If at any time in the future, the customer sends a follow-up message, an automated warning system alerts everyone with a reminder message that there is an outstanding customer response, so that it is sure to be answered. The reminder is repeated every 24 hours automatically until a response is sent. The reminder system is entirely automatic.

If a report is not closed by immediate response following the above scenario, then it is assigned to a specific engineer. The reminder system tracks failure to respond to the customer, sending out periodic reminders until an answer has been sent (the effect of this reminder system is that it is very unusual for a response not to be sent within 24 hours of the filing of a report). A priority is associated with the report, and a combination of manual intervention from the “bug manager” (who is a senior member of management), and automated tools ensures that reports receive proper priority, so that they can be handled in a manner that meets criteria (a) and (b).

In the case of defects, the first goal in a response is to provide a work around, which will allow the customer to continue with development. This is particularly crucial for high priority problems.

In certain cases, it is necessary to correct the defect and provide a special pre-release, called a *wavefront release*, which is made available on request. See the separate section of this article on procedures for updating and testing the system, for a discussion of the measures taken to ensure quality in such correction procedures.

Identifying Defects in the Technology

The identification of defects occurs through four sources:

1. New official validation tests[1] appear which demonstrate defects in the compiler. If analysis shows that the test is correct and demonstrates a real defect, then corrective action is taken.
2. A customer reports a defect, and analysis of the report indicates that there is indeed a defect in GNAT or its tool suite.
3. A user of the public version of GNAT reports a defect, and analysis of the report indicates that there is indeed a defect in GNAT or its tool suite.
4. An employee or consultant of Ada Core Technologies observes a defect from code reading, or actual use of GNAT or its tool suite.

In all four cases, entries are made in the tracking system. Even in case 4, procedures require that a tracking number be established through the automated tracking system (a special customer number is used to identify such internal reports). Priorities are established appropriately (for example typically reports from customers will have much higher priority than reports from users of the public version), but every report is filed, and assigned to an engineer for eventual resolution.

The key element of the procedures to ensure quality is that absolutely everything is tracked through the tracking system. In particular all email discussions relating to a defect, or enquiry, or enhancement suggestion, are always tracked, and saved in the tracking system database, with time stamps, so that it is possible to look at any entry in the system, and follow the entire discussion. If significant verbal discussions or telephone conversations relating to a specific tracked item occur, then they are manually entered into the system (all email is tracked automatically).

The Ada Core Technologies Test Suite

Compiler and compiler tool technology is ideally suited to systematic regression testing, for several reasons:

1. The input and output are well defined, and generally do not rely on real-time scheduling or external event tracking.
2. The specification is relatively static, and well defined
3. The output can be automatically checked using appropriate tools

Ada Core Technologies places high priority on building a test suite that will exercise the toolset to the greatest possible extent. With modern machines, very large numbers of tests can be run in a reasonable time. The company's test suite consists of the following components:

[1] A public test suite, now known as ACATS (Ada Conformance Assessment Test Suite) and previously known as the ACVC (Ada Compiler Validation Capability) has traditionally been used as a test of the breadth and depth of language feature implementation. There is a set of internationally standardized (ISO) procedures for carrying out these tests.

1. The “fixed bugs” suite, consisting of past defect reports (from customers or other users) that have been fixed, as well as specially-written tests and a selection of critical official validation tests. Currently this suite contains

   - 7000 test cases
   - over 50,000 separate files
   - over 5 million source lines

   These test cases contain a mixture of Ada 83 legacy code and Ada 95 code. Reflecting the fact that the public version of GNAT is widely used in the academic environment, the test suite is rich in elaborate and thorough use of advanced Ada 95 constructs.

2. The ACATS (formerly ACVC) test suite. This consists of several thousand executable and non-executable tests with strictly required behavior derived directly from the Ada Reference Manual.

3. The Digital Test Suite. This test suite consists of another several thousand test cases, and is a mixture of adaptations of customer code tests, as well as specially-written tests of the DEC Ada 83 compiler. This suite is part of Ada Core Technologies’ relationship with Digital Equipment Coporation (now Compaq Corporation) in providing a compiler for the OpenVMS operating system on Alpha. Although some of the tests are VMS specific, requiring the test suite to be run on VMS, the majority of tests are platform independent and are thus useful for general testing of the technology.

4. Specialized test suites for tools, including a test suite for the editing functions of the Integrated Development Environment (GLIDE), and for the operations of the debugger (GDB). The main fixed-bugs suite includes extensive tests of the tools as well as the compiler, including tests of gnatbind, gnatlink, gnatfind, gnatxref, gnatmem, asis, GLADE, florist, and the entire GNAT run-time library.

The test suite is constantly expanded as new test cases are written to exercise new developments in the technology, or as defects are reported, found and fixed. The latter procedure helps to ensure that future changes in the technology do not cause regressions to already fixed problems,

Correcting Defects in the Technology

1. Repositories and Configuration Management

   The sources and scripts for the GNAT system and its tools are maintained in CVS repositories on a UNIX machine. These are completely backed up, both by hot backup which instantly duplicates all Configuration Management transactions in a mirror system, and by periodic transfers between sites (so that the complete set of sources exists at multiple geographically separated sites). The sources are in CVS form, so they have an automatic history of all previous versions.

   If a correction is required, then the engineer working on the issue may make updates to the CVS repository. The company does *not* restrict who can update

what in the repository, and instead relies on very stringent requirements for such updates.

2. Use of the Mailserver technology.

   No one is allowed to update the CVS repository unless they complete a "mailserver" regression test. This is an automated procedure in which the proposed patch is thoroughly tested for possible regressions before it is committed to the CM system.

   An engineer correcting a defect prepares a patch file representing the proposed patches complete with full documentation of the changes. The change is sent by e-mail to a special automated agent that rebuilds the compiler and tools incorporating the patch and which then runs the mailserver regression suite.

   This regression suite is a large subset (about 90% of the total) of the fixed bugs suite (some tests are excluded because of technical reasons that makes them suitable only for running in the nightly test runs, described below).

   The tests are run automatically, and a report is sent back to the originating engineer indicating if the patch has caused any changes in the results of any of the regression tests.

3. Checking in Changes

   If and only if the regression test suite run by the mailserver indicates either no changes, or the changes are benign (e.g. improved wording of an error message), then the engineer checks in changes. The check-in must be accompanied by a detailed revision history indicating why the change was made, and this revision history cross-references the tracking number of the report relating to the modification or correction. This allows full tracking of changes against reports.

   The check-in is done via an agent that sends out messages to all other engineers noting the check-ins that were made and the revision histories. Any engineer is free to review these changes, accessing the newly checked-in source versions in the repository, and the procedures require that at least one person manually review all changes made to the system.

4. Nightly Testing

   The above procedures are usually sufficient to ensure that changes do not cause unexpected regressions, however there are unusual scenarios under which the above testing can be insufficient:

   1. The change is unexpectedly target dependent. It is not feasible to run the mailserver on all targets, so it is usually run on one particular representative target. It is quite unusual for changes to GNAT or its toolset to be target dependent, but there are exceptions to this general rule.
   2. Two engineers can check in changes that individually are correct, but have some unexpected interaction that causes errors. Again this is very unusual, but is at least theoretically possible.

To ensure that neither of these cases occurs, and as a double check on the entire modification process, automated builds are carried out on all targets every night. These builds include a complete bootstrap from scratch to ensure that the system can be built from the current base compiler, and also complete regression testing including all test suites (internal "fixed bugs" regression suite, ACATS tests, DEC test suite, auxiliary test suite).

The regression test suites are all self-checking and generate a report indicating if there are any unexpected regressions. These reports are transmitted to every engineer in the morning, and if unexpected regressions have occurred, these are treated as high priority defect reports that must be addressed immediately, following the normal procedures for processing defect reports. In particular, if a regression is not instantly cleared, special "regression" entries are made in the database to indicate the regression and its status, with comments being tracked in the usual manner.

5. Clearing the Report

If the check-ins are successful, and have caused no regressions, and a double check by the engineer to whom the report is assigned indicates that the defect has been properly cleared, then the following steps are performed:

1. A message is sent to the customer if it was a customer-reported defect, indicating that the problem has been corrected, and if necessary, arranging for a "wavefront" release to be transmitted to the customer (see separate section on wavefront releases).
2. Where possible, the report is turned into an executable regression test, and moved to the fixed bugs regression suite. If this is not possible, the report is moved to a special "dead" part of the database.
3. An entry is made in the known-problems file, which is also stored in the CVS repository under configuration management control. This file gives details on all corrected defects, indicating the scope of the defect, the current status, and suggested work-arounds. The entries in the known-problems file are keyed to the tracking number for the related report, ensuring trackability.

Implementating New Functionality

Implementation of new functionality proceeds in the following steps

1. Initial suggestion of general idea behind new function
2. Discussion of general idea
3. Production of specification for enhancement
4. Review of enhancement specification
5. Actual implementation of the change

To start the process, either as a result of internal verbal discussions, or submissions of ideas from customers, or in fulfillment of external contracts requiring enhanced functionality, a special tracking number is opened so that all discussions in stages 1-4 can

be fully tracked. All knowledgeable engineers can participate in the electronic discussion as the enhancement passes through the first four stages, and as usual the tracking of all electronic discussion is automatic.

Again, if there are telephone or verbal discussions that relate to the enhancement project, then they are manually logged into the database, so that the database contains a full track of the complete discussions. This allows any engineer to join the discussions at any point by reviewing the tracked transactions.

At some point, a specification is filed, and discussed; once the specification is agreed on, the actual implementation can commence. Note that the specification will be part of the tracked discussion transactions related to the enhancement report in the data base, which has a tracking number assigned like any other report.

The procedures for actual implementation of the change are identical to those discussed under correction of defects, with the following modifications and differences:

1. When the enhancement is complete, a complete test must be written and filed in the fixed-bugs regression suite to test out the new functionality.
2. Instead of resulting in an entry in the known-problems file, the enhancement, once successfully tested and checked in, is registered in another file, the “features” file, which is also part of the main CVS repository under configuration management. The entry in the features file cross-references the tracking number of the enhancement report.
3. The documentation must be enhanced where appropriate, following the procedures described in the separate section on documentation.

Maintaining Accurate User Documentation

An easily overlooked but very important aspect of quality control is maintaining up to date, accurate, and complete user-level documentation. Unlike the situation with the compiler and tools, there is no automated way to do testing on the documentation to ensure quality, so manual methods are used.

The primary principle is that the documentation is maintained using the same formats, procedures, repositories, and configuration management procedures as the sources. Thus the documentation can be updated by any engineer following the same procedures used for updating sources.

The format chosen for the documentation uses ASCII files with a simple markup system, rather than some elaborate external documentation system, in order to achieve maximum accessibility to the documentation repositories. Ensuring this accessibility makes it easy for engineers to take the effort to update documentation.

The actual process of updating the documentation is to check out the corresponding ASCII file, and check it back in with modifications, together with a revision history entry that, as usual, contains a reference to the corresponding tracking number to which the update applies. This tracking number ensures full trackability. For example, it is easy to determine all the features file entries that have not resulted in tracked updates to the documentation, and to ensure that there are no omissions.

When documentation changes are checked in, they generate messages to other engineers who review the changes just as they would review changes to the sources.

Since maintenance of the user documentation is a direct responsibility of the engineers who fully understand the technical issues involved, the documentation remains current and accurate.

In addition, if customers or other users report problems in the documentation, these are treated like any other bug reports, and are tracked like any other reports, to be resolved by appropriate reviewed changes to the documentation files.

The actual manuals are rendered in HTML and Texinfo formats for browsing, ASCII text for editor access, and postscript format for high quality paper reproduction; this is done entirely automatically without any manual intervention.

Ensuring Quality of Sources and Technical Documentation

Readable source code is a prerequisite for high quality software, and several steps are taken to achieve this goal.

1. An extensive set of style requirements gives the GNAT software the same "look and feel" regardless of the author. Typically in the case of Ada sources, these are consistent with the suggestions in the Ada Quality and Style document.
2. The compiler itself enforces many of these restrictions through a flexible set of options.
3. The check-in protocol automatically runs a style verifier to ensure that all checked in files follow the style rules.
4. The review process for sources includes reviewing the style, particularly for requirements that are not automatable, e.g. documentation requirements discussed below.

All documentation of the code is contained directly within the source files themselves. Attempting to maintain separate documents in text form makes it very difficult to ensure that this separate documentation is up to date; full documentation in the code helps achieve the goal of up to date, complete, and accurate documentation.

Stringent and complete documentation of the sources is required at five separate levels:

1. *"Signpost" documentation* must identify the overall structure of the system, and the overall function of each unit.
2. *Specification documentation* must include full details of how all the interfaces and subprograms of a package are used. Since Ada syntax separates the specification from the implementation, the information in a spec must be complete enough for all "clients" without requiring any implementation details from the body.
3. *High level functional specification* in the implementation section of a package body must document the overall flow and algorithms
4. *Detailed low-level descriptions* must explain what each section of code is doing at an appropriate level of abstraction.

5. *The code itself* must be written in a simple, easy-to-follow style, to take advantage of the self-documenting quality of well-written Ada code at the lowest level.

The review of all changes made to the sources includes review of all aspects of the documentation by engineers experienced in understanding the style and requirements for the GNAT sources.

The goal is a completely uniform style throughout the compiler sources. The notion of *author-free code* is promoted; no author names are attached to code, and no one is considered to own any part of the technology. This encourages people to expand their knowledge of the system, and encourages an environment where people can make modifications to any part of the system, since they find code that is in a familiar style with complete documentation.

Preparation of Releases

The process of preparing a new release of GNAT follows a strict protocol designed to ensure quality and avoid regressions.

1. *Wavefront releases*. As previously described, automated scripts rebuild the system on all targets every night, generating reports showing if any regressions have been detected by the test suites. The result of these builds is a set of automatically-packaged daily releases which include the entire system in distributable form.

   These daily builds are referred to as "wavefront" releases, reflecting the fact that they are built at the front of the development sequence, using the most up to date system components. These wavefront releases serve several purposes:

   a) They are used internally by the ACT staff in their day to day use of the compiler, so that ACT engineers are always using, and consequently testing under actual development use, the latest build of the system.

   b) They are distributed to customers as prereleases of the technology in cases where it is essential to move to the current components to fix a problem for which no satisfactory work around can be established, allowing the customer to proceed in this case.

   c) In the hands of the selected customers who receive wavefronts, the current release is informally tested in the field using real customer applications, allowing early detection and correction of problems well before the final release.

2. *Preparation of the release*. At the point where it is decided to issue a new release, based on the accumulation of corrected defects, or the implementation of new features, a temporary code freeze is enforced where no more development is allowed, and only correction of serious defects is permitted. The preparation of the release then proceeds in the following steps:

   a) The nightly run logs are carefully checked to ensure that there are no outstanding regressions, and a base date is used for the release on all targets. The sources and builds for this date are then specially isolated as a candidate release, rebuilding with a version number that reflects the release status.

b) A testing book is prepared, and checked in with its own tracking number requiring a comprehensive set of procedural tests on all aspects of the system. These tests consist of actually installing the system on all targets as distributed, and then performing basic functionality tests to ensure that the packaging is successful and complete, and there are no missing components.

c) If any defects are found during this process, they are fixed by use of a special patch file that applies minimal tests to the candidate release, the regression tests are rerun, and steps a) and b) are repeated as necessary.

The testing against the testing book is logged in the report tracking system in the usual manner, so it can be determined when the testing is completed by clearing all entries in the testing book.

3. *Beta testing*. The candidate release is then announced to customers, who are invited to participate in a beta test period. In organizing this beta test the company ensures that enough large-scale users switch to the new release to add extensive field testing to the regression testing already done in house.

   If the beta testing shows any defects, these are fixed by updating the patch file for the release (which is itself checked in to the repository and is under configuration control like all files on which the release depends). Steps 2a) through 2c) are repeated; if necessary, an additional beta test period is instituted.

4. *Final release*. When the beta testing shows that the new release is stable and free of regressions, the final release is announced and made available for all customers.

   If any regressions or serious defects show up during initial use of a new release, then a refresh version of the release may be issued. This will include whatever corrections are needed to address the defects, and will follow the steps in sections 2 and 3 as required.

Documentation of Internal Procedures

The procedures described in this article, along with associated technical details, are fully defined in an internal ACT-Procedures file, which is stored in normal documentation format and is therefore available to the company's engineers for all forms of access, including HTML browsing on an internal website, and printing in high quality on paper. This procedures file is maintained in the main repository, and is itself under configuration control. Any change in procedures, or required clarifications, are checked in as changes to this file, and these check-ins themselves need to meet the normal requirements for detailed revision histories and review.

As an example of the completeness of these procedures, the preparation of this article was itself tracked under a specific tracking number. The preparation resulted in a file entered into the main repository, and maintained under configuration control. The associated tracking number for the preparation of the article acts as a repository for all comments, corrections, and suggestions from the entire staff at Ada Core Technologies, to ensure the highest quality in the article itself. An update was also made to the ACT-Procedures file to reflect the function and purpose of this article.

Metrics and Internal Evaluation of Procedures

Since all transactions are completely logged in the database, with time stamps, the company has full tracking of the operation of the quality control system. This database can allow staff to answer questions relating to the speed of replies to reports, number of regressions encountered etc. These measurements allow constant improvement by tuning the QA procedures, with the tuning being reflected in the ACT-Procedures file.

Conclusions

Quality Assurance requires dealing with complexity: managing multiple versions of the same software for different platforms, ensuring user documentation consistent with the product, tracking bugs and preventing regressions, tracking enhancements, and many other activities. Ada Core Technologies has implemented a largely-automated system for coping with these issues and has addressed quality at multiple levels:

- Source code with a consistent style and with multiple reviewers
- An audit trail for each reported bug so that its status is always known
- Extensive test suites from both public and internal sources
- Nightly test runs on all platforms
- Wavefront releases and beta test programs
- Well-defined procedures for configuration management

Through these processes the company has been able to develop a product suite comprising several million lines of code, through engineers distributed geographically in both the U.S. and Europe, and to regularly release new versions with enhancements and bug fixes. Quality Assurance may be difficult, but with the appropriate tools, people, and processes, it is manageable.