New views of mature ideas on software equality and productivity.

Robert Watkins and Mark Neal, Abbott Laboratories Diagnostic Division

WHY AND HOW OF REQUIREMENTS TRACING

While requirements traceability is explicitly required in US Department of Defense software contracts, it is often hard to sell in other situations. This article by Robert Watkins, software engineer, and Mark Neal, software product assurance manager, explains how Abbott Laboratories Diagnostics Division approaches traceability. The article is based on their presentation at a recent conference of the Dallas/Fort Worth Association for Software Engineering Excellence.

— Dave Card, Editor

MUCH DEBATE AND CONTROVERSY surrounds software-requirements tracing, primarily because, other than to fulfill contractual obligations, many developers are not sure why they should do it at all. Software requirements traceability is commonly practiced by US Department of Defense contractors and developers of safety-critical systems, but how necessary is it for other types of applications that are less critical?

TRACEABILITY CAN HELP IN MANAGING CHANGE AND COSTS FOR ALL TYPES OF PROJECTS.

At Abbott Laboratories Diagnostics Division, we have found traceability — which we define as a link or definable relationship between entities — to be indispensable and have mandated that each software version be traceable through specifications and validation documentation to the version or source from which it was developed. We instituted this traceability program in 1987 and have found that it greatly assists project management. In fact, we like to say, "You can't manage what you can't trace."

Our projects typically involve real-time, embedded, in vitro diagnostic instruments that approach 200,000 lines of code with moderate to high system complexity. We have found that traceability aids project managers in

- ♦ *Verification.* It helps us verify that software requirements have been allocated both to design and code and to test cases and procedures for verification. This ensures that only required functions are designed into the product (no added bells and whistles); as well as that all requirements have design components and associated qualifications test cases.
- ♦ *Cost reduction.* Traceability lets us allocate all product requirements early in the development cycle. The cost of waiting until the integration and system-test phase to correct defects (in untraceable components) may be as much as 30 times higher than doing it in the initial phases, according to Barry Boehm (*Software-Engineering Economics*, Prentice-Hall, 1981).
- ♦ *Accountability.* During internal or external audits, the project will have a better success rate if the data is available for auditors and you can prove that a requirement was successfully validated by associated test cases. Project managers have a better handle on cost, and customers are assured of getting the product they requested. By providing quantitative traceability analyses, project milestones can be more easily approved and deliverables more easily verified; thus customer confidence and satisfaction is enhanced.
- ♦ *Change management.* For each change, it is easier to determine what related elements of the design are affected. This helps keep documentation up to date as the implementation progresses. In addition, managers can identify test procedures that should be rerun to verify the change. This knowledge saves test resources and lets them streamline the schedule.

To illustrate our program, we describe how we applied it to an R&D project to develop an embedded, real-time in vitro diagnostic instrument. The software is part of a system to test human body fluids for conditions such as cancer or hepatitis. The software, written in C, has a medium to high complexity, with 175,000 lines of source code and approximately 1,600 separately identifiable software requirements that had to be traced. The development team consisted of 33 engineers: 18 assigned to software, 12 to independent verification and validation, one to software-quality-assurance, and two to configuration management.

We also describe how we applied the ATS, a

Editor: David Card
Software Productivity Solutions
122 N. 4th Avenue
Indialantic, FL 32903
dc@sei.cmu.edu

traceability system that automated many tasks in establishing traceable requirements. However, the *tool* was not what ultimately led to our success. It was the *process.*

Finally, we recognize that a traceability program is not without cost, but we hope to show that the cost is minor compared to the benefits.

TRACEABILITY PROGRAM. Our traceability program consists of identifying links and determining that those links are complete and accurate. The ATS helped us implement both these tasks.

♦ *Identify links.* We used processes, techniques and tools to establish and maintain relationships and correlate discrete inputs and outputs in development phases. Figure 1 shows the development phases and beginning and end products associated with each. These products serve as traceability inputs and outputs. During development phase *x*, outputs are used from phase *x*-1, its predecessor, as traceable elements. These products are then used to trace documents and build traceability matrices. They also serve as the basis for tools and procedures that aid in performing and verifying traceability, such as an incremental analysis or a one-time (system-wide) method.

♦ *Verification and analysis.* Processes also provide a way to verify predecessor-successor relationships. Establishing predecessor-successor relationships consists of applying methods to verify appropriate linking. These include formal inspections, formal reviews and audits by the project team or an independent group. You can also embed traceability in the development tools, which essentially involves creating a CASE environment. This offers the power and range of automation, but can be costly because such tools must be integrated into the existing development environment. However, we believe it would benefit all projects to plan early to embed as much traceability as possible into the document-generation process.

Once traceability has been identified, there must be a disciplined, well-documented and controlled engineering process to ensure that the links have been

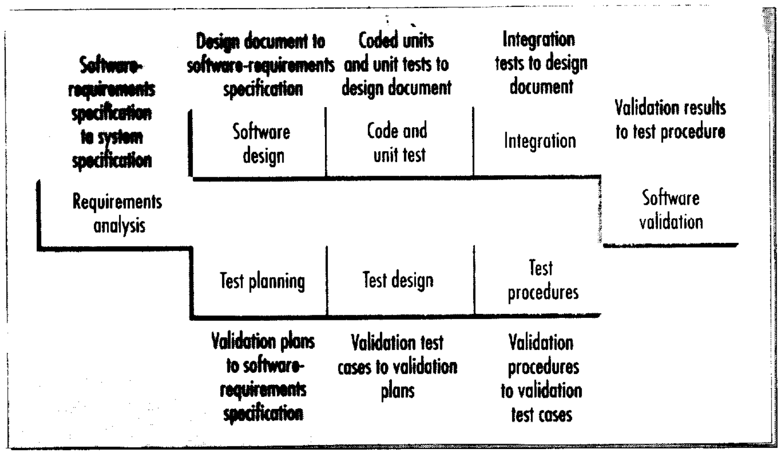
Software-requirements specification to system specification
Design document to software-requirements specification
Coded units and unit tests to design document
Integration tests to design document
Validation results to test procedure
Software design
Code and unit test
Integration
Requirements analysis
Software validation
Test planning
Test design
Test procedures
Validation plans to software-requirements specification
Validation test cases to validation plans
Validation procedures to validation test cases

Figure 1. Development phases and products give rise to requirements-traceability activities. Each development phase (rectangle) has associated with it definable products. From the products of phase x, we trace back to phase x– 1, its predecessor phase. For example, during software design, we trace back to the software-requirements specification, which is the output form the requirements-analysis phase. Testing proceeds in parallel with implementation.

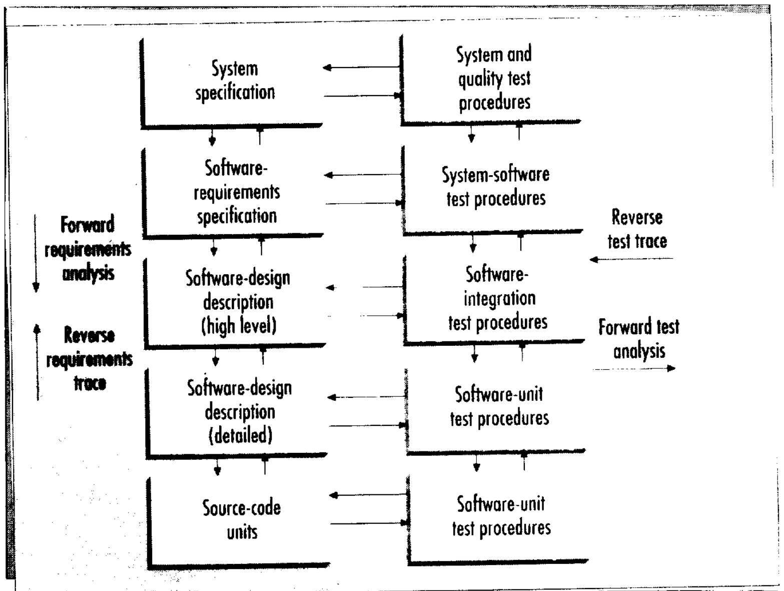
System specification
System and quality test procedures
Software-requirements specification
System-software test procedures
Forward requirements analysis
Reverse requirements trace
Reverse test trace
Software-design description (high level)
Software-integration test procedures
Forward test analysis
Software-design description (detailed)
Software-unit test procedures
Source-code units
Software-unit test procedures

Figure 2. How requirements are analyzed (forward) and traced (reverse) and tests traced (reverse) and analyzed (forward) within the documentation hierarchy.

identified correctly. The documentation hierarchy tree in Figure 2 depicts four aspects of traceability analysis: forward requirements analysis, reverse requirements trace, forward test analysis, and reverse test trace.

♦ *ATS.* We used the ATS as a management tool to support the myriad traces between software documents and test documents. We developed the ATS using version 2.0 of Microsoft's FoxPro, a relational DBMS program. For us, a key feature of FoxPro was Relational Query by Example, which let us build SQL reports on the fly to quickly analyze predecessor-successor relationships.

IMPLEMENTATION. Figure 2 shows the steps we used to trace each software requirement through its design elements (left) and corresponding test traces (right).

We used the ATS to generate two kinds of *traceability matrices*, documentation and test. These matrices let us perform both forward and reverse analyses to verify that predecessor-successor and successor-predecessor relationships exist and are complete and correct. Both matrices were vital.

The *documentation matrix* shows predecessor requirements, which we derived from decomposing documents from their predecessor documents. As each software requirement is allocated to lower level documents, a test strategy is necessary — one that maps how these requirements are to be verified and validated. As each test procedure is developed, the test engineer decides which requirement the test will verify and records that information, thus building a *test matrix.*

The ATS gave us timely feedback to assist in determining the development status by reporting design components not yet traced (implemented) during the life cycle. During the test phase, the ATS provided input to project managers to determine which requirements were covered by test cases. The team generated reports and identified resources to address untested areas or to add to areas that may not have had enough testing.

During requirements analysis and design, we imported trace matrices generated from CASE tools or spreadsheets to the ATS database. From this data, we conducted a forward trace to determine requirements with no allocated design components and a reverse trace to determine design components with no higher level requirement. We generated the analysis periodically to show parts of the design that the customer did not require and to point out incomplete requirements.

In preparation for formal audits, we generated reports to ensure that a particular phase had complete and accurate requirements traceability. When the report showed something missing, we updated the requirements or fixed the design documentation and updated the database.

LESSONS LEARNED. We learned a great deal from this application and continue to experiment with and evolve the traceability-analysis process. In particular, we found that we must establish documentation rules and formats much earlier in the life cycle. Documentation rules we developed over the course of the program are

♦ Make the document requirements section consistent from document to document.

♦ Use unique paragraph numbers for each discrete requirement. Otherwise, if you decide to do traceability later on, it may be very difficult.

♦ Append any additions to document paragraphs at the end of a section. Do not insert them. In our case, the ATS could not dynamically update the trace matrices, but it is better to use a numbering scheme and database that support dynamic updates if you can.

♦ Keep the paragraph number of any deleted paragraph and give it a "Deleted" tag. Information about what was deleted can be as significant as what was not.

Another lesson we learned is that traceability does cost. You need staff to support formal inspections or reviews. For most projects, you need some type of tool to assist in managing the trace matrices, whether it be a CASE tool, DBMS, or simple spreadsheet. We developed a relatively low-cost tool to handle our needs; we expect others can do the same. Whatever the cost, it is important to include it as well as the costs for developing and maintaining procedures for documentation formats and generation, traceability analysis, and formal inspections.

Our biggest lesson is that traceability is critical to project success. We believe it was one of the key factors in releasing our diagnostic instrument on time.

Finally, we determined that some type of traceability program is necessary when any one of these conditions holds:

♦ The customer requires it.

♦ The project is complex.

♦ You must see different types of documentation and their interrelationships.

♦ You must ensure that you built and tested the system you claimed.

♦ The system is safety critical.

These apply to most development projects. There was one type of project that might be a candidate for not having traceability — a requirements prototype with a very short development cycle. However, even this type of project could benefit from such a process, especially if the prototype is intended to evolve into a real product. ◆

Robert Watkins and Mark Neal can be contacted at Abbott Laboratories, PO Box 152020, MS 5-2, Irving, TX 75015-2020.

Fifth International Conference

Applications of Software Measurement

November 6 – 10, 1994
La Jolla, California

Featuring Tom DeMarco

Int'l experts DeMarco, Dion, Coallier, Kitchenham, and Abdel-Hamid

In depth Pre-conference Tutorials focusing on key measurement issues

Practitioner talks detailing tribulations and victories in measurement usage

The Measurement Tools & Services Exposition

For the complete ASM'94 brochure...

Call 904-268-8639 ext 214

ASM94

Fax to 904-268-0733

Organized by Software Quality Engineering