Requirements traceability: Theory and practice

Balasubramaniam Ramesh*

Code SM/RA, Naval Postgraduate School, 555 Dyer Road, Monterey, CA 93943, USA

Curtis Stubbs

Air Test and Evaluation Center Squadron One (*VX-1*), *Patuxent River, MD 20670, USA*

Timothy Powers

US Coast Guard Commander (*TMM*), *17th Coast Guard District, Juneau, AK 99802, USA*

Michael Edwards

Code B44, NSWCDD, 10901 New Hampshire Avenue, Silver Spring, MD 20903, USA

Current literature as well as standards that mandate requirements traceability do not provide a comprehensive model of what information should be captured and used as a part of a traceability scheme, leading to wide variation in the quality and usefulness of traceability practice across systems development efforts. In this paper, we present a framework for representing and developing a traceability scheme. The experiences of an organization using traceability as an important component of a quality software engineering process are discussed. Models describing the traceability practice in the organization, as well as issues and lessons learned, both from organizational and technical perspectives, from implementing a comprehensive traceability practice are presented.

1. Introduction

A major concern in the development of complex, large-scale computer intensive systems, especially those with evolving requirements, is ensuring that the design of the system meets the current set of requirements. In this context, it is essential to maintain the traceability of requirements to various outputs or artifacts produced during the system design process.

Requirements traceability has been described as a measure of system quality [Roetzheim 1991] and has been mandated for US Government systems for many years through standards, such as MIL-STD-2167A and MIL-STD-498 which replaced

* Current address: Department of Computer Information Systems, Georgia State University, Atlanta, GA 30303, USA. E-mail: bramesh@cis.gsu.edu.

it [DoD 1988]. It is estimated that the US Department of Defense spends nearly 4% of the total life cycle costs in capturing and maintaining traceability information, amounting to billions of dollars each year. Despite this massive outlay of funds, recent field studies suggest that the quality and content of traceability information varies widely across system development efforts, with much of the captured information out-dated and useless [Ramesh *et al.* 1995]. The following discussion highlights a primary reason for such a poor state of current practice: a) varied and often conflicting views among different stakeholders in systems development on the goals, objectives, components and representation of requirements traceability, and b) the inadequate elaboration, in the standards and current literature, on what types of traceability information needs to be captured and maintained to achieve the benefits of traceability.

1.1. Definitions

Hamilton and Beeby [1991] characterize traceability as the ability to "discover the history of every feature of a system" and determine the impacts of proposed changes. Edwards and Howell [1992] define traceability as a technique used to "provide a relationship between the requirements, the design, and the final implementation of the system". These relationships allow designers to show that the design meets the requirements and help early recognition of those requirements not satisfied by the design. Another definition of traceability examines the systems description techniques and how they allow changes to any part of the design–requirements, specification, implementation-are traced throughout the system [Greenspan and McGowan 1978]. The ANSI/IEEE Standard 830-1984 states that a software requirements specification is traceable if (a) the origin of each of its requirements is clear and if (b) it facilitates the referencing of each requirement in future development or enhancement documentation [IEEE 1984]. Thus, requirements traceability is a characteristic of a system in which the requirements are clearly linked to their sources (backwards traceability) and to the artifacts created during the system development life cycle based on these requirements (forward traceability) [Gotel and Finkelstein 1994].

1.2. Goals and objectives

In software development, the objective of requirements traceability is to ensure that the software produced meets the expectations of the user.

Stehle [1990] lists some objectives of requirements traceability, namely to:

- promote a contractor and contracted method of working;
- demonstrate that each requirement has been satisfied;
- demonstrate that each component of the system satisfies a requirement.

Thus, one of the primary uses of requirements traceability is for system developers to prove to the customer that requirements have been understood, the product complies with those requirements, and the product has no unnecessary features, referred to as gold-plating [Wright 1991].

To effectively prove requirements compliance, requirements traceability should also address how the requirements are arrived at and the design rationale that identifies not only the decisions, but also the supporting/opposing reasons behind those decisions [Ramesh and Dhar 1992; Curtis *et al.* 1988]. Mis-communication between the customer and systems engineer often results in project delays, cost overruns, the delivery of projects that do not meet customer specifications, and project cancellations. Traceability facilitates communication among those involved in a project to alleviate some of these problems.

During the design phase, traceability can be achieved by linking design elements to requirements in a bi-directional manner across all design stages and design views [Marconi 1991]. Traceability allows designers and maintainers to keep track of what happens when a change request is implemented. This enables the implications of a requirements change to be determined before system redesign takes place [Edwards and Howell 1992]. Systems evolution requires a better understanding of the requirements. This can only be achieved by tracing requirements back to their sources [Pinheiro and Goguen 1996]. Additionally, traceability provides a chain of accountability within the development process [Cordes and Carver 1989].

In large scale software development activities, especially in contexts where such activities are contracted out, having a precise method for ensuring that requirements are met by the design and guaranteeing that the current set of requirements are met by the evolving system is vital. Gathman and Halker [1990] state that the absence of clear traceability of designs to requirements may create serious problems in configuration control and lead to slippages in delivery schedules.

1.3. Components of a traceability scheme

A number of stakeholders (including project sponsors, project managers, analysts, designers, maintainers, and end users), each with a different set of goals and priorities, are involved in the systems development process. The traceability needs of these stakeholders differ depending on their role and perspective.

From a requirements management perspective, Fiksel [1994] advocates the capture of information necessary to assist system developers with requirements capture, tracing and verification. Similarly, from a designer's perspective, Smithers *et al.* [1991] suggest that the following elements of design history be documented as a part of a traceability scheme:

- results of the design,
- justifications of the results,
- important decisions or assumptions made during design,
- contexts of the design solutions.

Capturing design rationale in large scale projects is the focus of the systems such as gIBIS [Conklin and Begeman 1988] and SYBYL [Lee 1990], and the QOC approach

[Maclean *et al.* 1991]. gIBIS provides a hypertext interface to the Issue Based Information Systems method, a rhetorical model for representing argumentation processes. It can been used to capture issues or concerns, alternative ways of addressing them, and arguments for and against these alternatives in a decision situation. SYBYL is organized around decisions graphs which record the pros and cons of choosing from a set of alternatives to satisfy a goal. The DRL languages supported by SYBYL provides a variety of primitives similar to those provided by IBIS. In the Questions, Options and Criteria (QOC) approach, the criteria for evaluating decision alternatives and how well the various options perform with respect to the criteria are explicitly represented.

According to Brown [1987] test procedures should be traceable to requirements or designs so that they can be updated whenever errors are discovered. Schneidewind [1982] suggests the tracing of errors to the applicable design specifications and user requirements. According to Keuffel [1990] tracing requirements to data structures and functions will make maintenance easier and more accurate. Murine [1986] defines security traceability, a component of Software Security Metrics, as "those security requirements that provide a thread from the system security requirements to the implementation with respect to software development and security environment". Baldo [1990] advocates capturing the context and the constraints of the development process to be able to assist the users of a software component in reusing it on another application.

1.4. Representation format

A variety of mechanisms for representing traceability information have been proposed. Schneidewind [1982] provides several formats for representing traceability information including, event tables, condition tables, and selector tables. Macmillan and Vosburgh [1986] recommend that a requirements traceability matrix be created to map system requirements to software functions and serve as "the basis for a completeness indicator". West [1991] also suggests the use of matrix analysis to "develop design requirements from customer requirements, product functions from design requirements, test requirements and process requirements from customer and design requirements, and so on". The matrices are then linked, with the output on one as the input of another, throughout the development process. Jackson [1991] presents a method of traceability using key phrases to express relationships between entities.

1.5. Traceability tools

A number of traceability tools with a wide variety of capabilities have been developed by the industry, both for in-house use and as commercial products. These include ARTS [Dorfman and Flynn 1984], Teamwork/RQT [McCausland 1991], DOORS [QSS 1995], SLATE [TD Technologies 1996], RTM [Marconi 1991], and RDD-100 [Alford 1991]. The capabilities of these tools include mechanisms to create parent/child relationships, functional hierarchies, definition of keywords and attributes

to requirements and other system artifacts, ad-hoc and pre-defined querying, requirements extraction from documents, customized report generation, and maintenance of information about allocation of requirements to system components or functions. Most of these tools provide mechanisms to represent various types of linkages between different artifacts. However, interpretation of the meanings of these linkages is left to the user. Recent research recognizes the importance of capturing the precise semantics of traceability relationships (where feasible) so that automated reasoning with that information can be facilitated. Tools such as TOORS [Pinheiro and Goguen 1996] and PRO-ART developed by the NATURE project [Jarke *et al*. 1993; Jarke and Pohl 1993; Pohl and Jacobs 1994] are representative of this category. TOORS represents requirements and relationships among requirements as objects. Relationships specified by axioms, rather than simple named links, are used for maintaining and reasoning with traceability information. The NATURE project views requirements determination as the "process of establishing visions in contexts" organized into four worlds dealing with subject, usage, system and development. It provides tools for improving the requirements engineering process with traceability to both formal and informal knowledge.

1.6. Conclusions

There are wide variations in the format and content of traceability information that is considered useful in different contexts. There is a clear need for the development of traceability models with well defined semantics and guidelines for their use in software development activities. A comprehensive scheme for maintaining traceability, especially for complex, computer-based systems, requires that all system components, not just software, created at various stages of the development process, be linked to the requirements. These components include hardware, software, humanware, manuals, policies, and procedures. In order to achieve this objective, it is essential that traceability be maintained through all phases of the systems development process, from the requirements as stated or contracted by the customer, through analysis, design, implementation, to testing the final product. Such traceability information can be used by various stakeholders to show that the requirements have been met, the rationale behind design decisions are sound, and for establishing change control and maintenance mechanisms etc. The focus of this paper is to illustrate such a practice of traceability using a case study and highlight some lessons learned.

First, we present a framework for representing and developing a comprehensive traceability scheme using a meta-model. This framework is intended to provide a uniform representation of various traceability information captured and used in system development activities. Next, we describe the case study in which this framework was used in order to understand and describe the traceability practice in an organization which views traceability as an important component of quality software engineering process. The focus of the discussion is not to prescribe a traceability scheme, but to present a framework for organizations to understand and describe their current traceability practices and develop policies for integrated traceability schemes. Finally, we

discuss several lessons learned from implementing such a comprehensive traceability practice – both from organizational and technical perspectives.

2. Requirements traceability framework

A primary objective of this research is to describe a comprehensive practice of traceability. Our challenge is to present various components of traceability information using a uniform framework to facilitate not only the understanding of the practice at this organization, but also the development of similar models in different organizational contexts.

2.1. Traceability meta-model

We have developed a meta-model for representing traceability information at different levels of abstraction. This model can be used to represent different traceability linkages among various objects produced during system life cycle. The model provides primitives to represent agents, inputs, and outputs of the software development process as well as linkages between them. We describe the traceability models at various levels of abstraction or granularity to recognize, that in practice, various organizations may implement components of traceability at different levels of detail. The meta-model shown in Fig. 1 can be described in brief as follows:

Stakeholders represent the agents in systems development life cycle. Examples of *Stakeholders* include customer or project sponsor, project manager, systems analyst, program sponsor, designer etc.

Objects represent the inputs and outputs at different phases of the system development process. Examples of specific *objects* include Requirements, Assumptions, Decisions, Rationale, Alternatives, Critical Success Factors, Derived requirements etc.

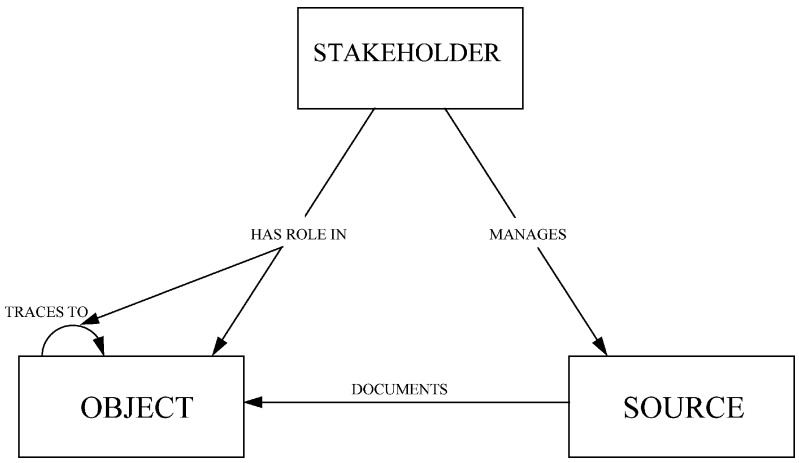

Figure 1. Traceability meta-model.

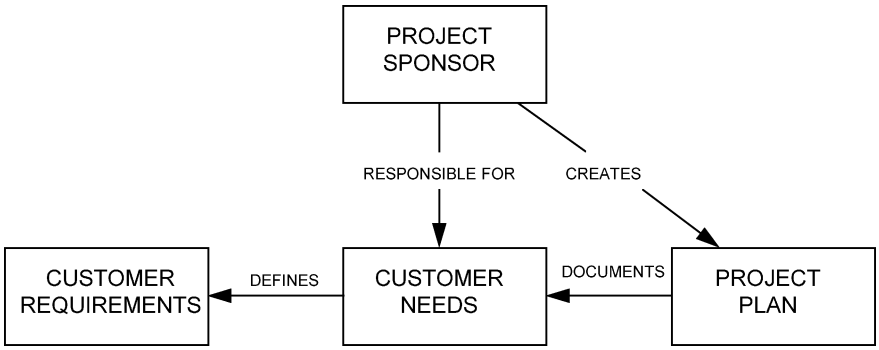

Figure 2. Specialized traceability model.

These represent the major conceptual elements among which traceability can be maintained during the various life cycle stages such as requirements analysis, design and implementation. Traceability across various *objects* is represented by the **traces-to** links.

All *objects* are documented by *Sources* which may be physical media such as documents or references (say to people or undocumented "policies"). Specific types of *sources* include system specifications, notes, memoranda, and telephone calls as well as references to specific persons. *Stakeholders* manage *sources* and play different roles in the creation, use and maintenance of various *objects* and traceability links among them.

Each component and linkage in this meta-model can be specialized and instantiated to produce organization and project specific models and instances of traceability information. Such a specialization illustrating some aspects of the traceability practice in requirements management is illustrated in Fig. 2. In this figure, PROJECT SPONSOR and PROJECT PLAN are specializations of *stakeholder* and *source* respectively. CUSTOMER NEEDS and CUSTOMER REQUIREMENTS are specializations of *objects*. The links in the meta-model can also be specialized. In Fig. 2, the link (**defines**) between CUSTOMER NEEDS and CUSTOMER REQUIREMENTS is a specialization of the **traces-to** link in the meta-model. The **responsible-for** link is a specialization of the **has-role-in** link. Similarly, other links in Fig. 2 are specializations of the corresponding links in the meta-model. Traceability information about customer needs captured here include its relationship to customer requirements, the source where it is documented, and the stakeholder who is responsible for it.

We now illustrate (in Fig. 3) how such a traceability model can be instantiated. Here, the BRANCH CHIEF is the project sponsor who is responsible for the MISSIONS NEEDS STATEMENTS (representing CUSTOMER NEEDS) that are documented in the PROJECT MASTER PLAN. SOFTWARE REQUIREMENTS represent CUSTOMER REQUIREMENTS. Traceability between SOFTWARE REQUIREMENTS and MISSIONS NEEDS STATEMENTS are explicitly recorded.

In summary, the framework described here can be used to describe traceability information on what the various objects of interest are, how they are related, where

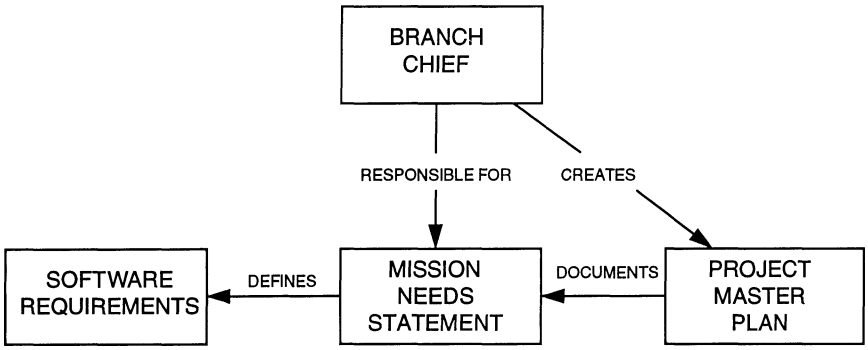

Figure 3. Traceability example.

they are documented and what role various stakeholders play in managing them. This framework was found useful in understanding and documenting the traceability practice in the case study described below.

3. Case study

3.1. Background

The results presented in this paper are based on our independent assessment of the use of requirements traceability by a government organization (hereafter referred to as TIEB) in their day to day operations as well as on a specific project (referred to as ADIP). TIEB was chosen as the subject of this study after an extensive search for an organization that uses traceability as an important component of its systems engineering practice.

TIEB provides embedded computer systems and software support for government as well as private industry customers. As its operations are similar to those of commercial organizations, the lessons learned from implementing traceability in TIEB are likely to be of general interest.

The ADIP project involved the redesign of a flight control software program for a jet aircraft from Jovial programming language to Ada, incorporating enhancements. The software contains approximately 75,000 lines of code and over 3,000 requirements. The new system had to be functionally equivalent to the existing system so that the pilots would not require extensive retraining.

Though TIEB had not developed formal traceability models, it was interested in documenting current practice and defining uniform traceability policies.

3.2. Data collection

Several on-site interviews, focus group discussions and observation sessions were conducted over a period of several months. One-on-one interviews were conducted with the organization's senior management, the project manager, system designers, and test/audit personnel to determine how the various stakeholders viewed

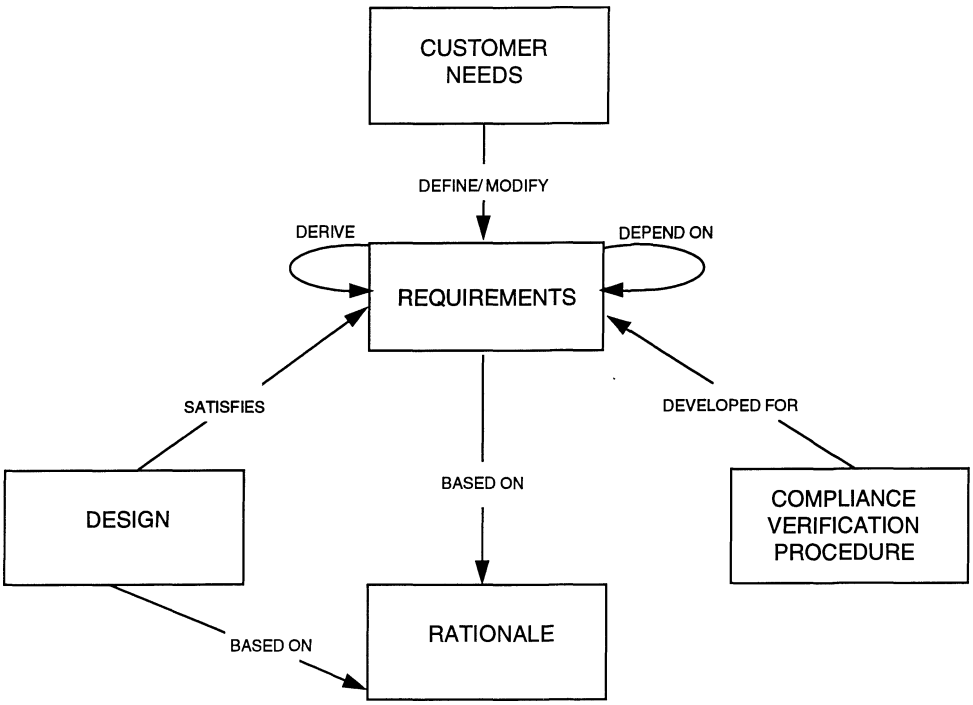

Figure 4. Requirements/design traceability model.

traceability, the extent to which each stakeholder used traceability in their operations, and the perceived benefits of using traceability. Documents produced by the project participants were collected and analyzed to understand the content, format and sources of various traceability information and the roles different stakeholders had in their creation, maintenance and use. Finally, our analyses were reviewed and validated by the project personnel.

3.3. Traceability models for TIEB

We now present two models that describe the traceability information captured and used at TIEB. Figure 4 shows how various aspects of requirements and designs are linked together. In this figure, REQUIREMENTS, CUSTOMER NEEDS, DESIGN, RATIONALE, and COMPLIANCE VERIFICATION PROCEDURE are all specializations of *objects* in the meta-model. The links between these *objects* (**based-on, depend-on, developed-for**, etc.) are specializations of the traces-to link in the meta model.

The model can be interpreted as follows: TIEB maintains information on customer needs and these needs define/modify requirements. When new requirements are derived from existing requirements, the linkage is maintained. Also, when a set of requirements depend on other requirements (e.g., when a hardware requirement depends on a software requirement) the dependencies are documented. For each requirement,

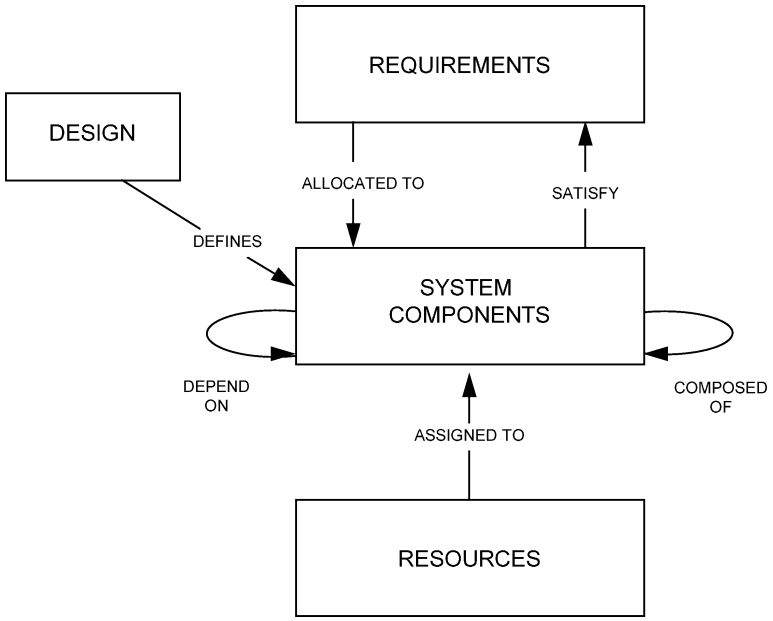

Figure 5. System component traceability model.

the rationale behind its creation is maintained. Compliance verification procedures such as tests, simulations, etc. that need to be performed to ensure that a requirement is satisfied are also developed for (and linked to) each requirement. Designs that satisfy specific requirements are clearly identified. The rationale behind design decisions are also identified and linked to designs.

Figure 5 shows a similar model describing how requirements and designs relate to system components. As before, REQUIREMENTS, SYSTEM COMPONENTS, RESOURCES and DESIGN are specializations of *objects* in the meta-model. The various linkages among them (**depend on**, **assigned to**, etc.) are specializations of the **traces-to** link in the meta-model. At TIEB, information about the allocation of requirements to system components is explicitly identified. Further, the resources (such as personnel and physical resources) assigned to system components are identified. When a system component is composed of lower level components, such a breakdown is explicitly represented. Information on dependencies among system components is also maintained. Requirements can be related to various levels of system components.

The operationalization of an aspect of the scheme in Fig. 4, in fact, is shown in Figs. 2 and 3. As shown in Fig. 4, TIEB requires that traceability links between customer needs and requirements are maintained. Figure 3 represents how this is done. The project sponsor (TIEB Branch Chief) creates and maintains the project master plan that documents customer needs. The project leader (TIEB section chief) maintains the Software Requirements Specification (SRS) documents (not shown in the figure). The SRS contains software requirements and each requirement in this document must have specific links to mission need statements from which they were arrived at. Similarly,

each of the traceability links shown in Figs. 4 and 5 are operationalized with specific guidelines. The following discussion provides an overview of how various traceability information is captured and used by TIEB.

3.4. Traceability implementation at TIEB

3.4.1. Senior management

The senior management of TIEB views the use of requirements traceability as "a must for survival". According to the management "Traceability ensures customer satisfaction by providing us a documented means by which to prove to the customer that all of the stated requirements are met and that the job is completed". Specifically, during customer reviews, traceability documents showing the relationship between design diagrams and requirements (example of DESIGN-satisfies-REQUIREMENTS link in Fig. 4) and requirements and code segments (example of REQUIREMENTS-allocated to-SYSTEM COMPONENTS in Fig. 5) are evaluated.

Equally important, from their standpoint, is the need to minimize the possibility of missing a stated or derived requirement in the process of developing large, complex systems. In the case of the ADIP, missing even a single critical requirement could be catastrophic for the pilots flying the aircraft. Therefore, explicit links are created between original and derived requirements in the SRS and other requirement documents (example of REQUIREMENTS-derive-REQUIREMENTS link in Fig. 4). Also, when dependencies among requirements (say, across different procedures governing the navigation and tracking of targets) are identified (example of REQUIREMENTS-depend on-REQUIREMENTS link in Fig. 4), they are recorded.

Management uses traceability in evaluating and accepting potential projects. With a complete list of validated requirements, TIEB management estimates the size, scope and projected staffing level required for the project, ultimately developing a bid for the project. Traceability information relating requirements and these projections are maintained in order to assess the impact of changes. This is done by identifying the major components required to satisfy the requirements and relating different resources needed to implement those components (examples of REQUIREMENTS-allocated to-SYSTEM COMPONENTS and RESOURCES-assigned to-SYSTEM COMPONENTS links in Fig. 5).

The traceability information linking requirements, system components and resources provides the manager with a means for tracking staff progress on the project. By tracing the requirements down to the Computer Software Unit (CSU) level, management can readily assign tasks and track completion status. For instance, a traceability matrix is generated to provide management with the status of each module of the system being developed, with information on completion status and projected completion date for the various tasks. This information assists in change management, and in projecting the future workload and costs of any proposed changes.

3.4.2. Requirements engineer

In ADIP, the available documentation on original system requirements and their rationale is incomplete. Therefore, it is nearly impossible to *trace back from requirements* [Gotel and Finkelstein 1994] to their sources. However, TIEB is *reengineering* requirements and their rationale (creating REQUIREMENTS-based on-RATIONALE links in Fig. 4), whenever feasible. These are documented in the engineer's notebook and traceability matrices. Required documentation includes information on how requirements are identified and/or modified by engineering change proposals.

3.4.3. Project manager

The project manager believes that proper use of traceability provides a means of showing s/he is in full control of the project. The project manager tracks the original as well as derived requirements to the CSU level (using REQUIREMENTS-allocated to-SYSTEM COMPONENTS links). This information is used to assure the customer that all requirements are understood and validated, that derived requirements are documented and validated, and that the resulting system design will meet all of the stated requirements. By tracing requirements to the CSU level, derived requirements become readily apparent and are entered into the system.

Through the design phase, the project manager uses traceability to track project status and personnel assignments at the CSU level. A GANTT chart is completed by each engineer with the rows representing tasks assigned and the columns representing weekly work periods. This information is consolidated by the project manager in developing weekly project status reports and detecting project delays.

Upon receiving a request for a change to the system, the project manager, with the assistance of the lead and design engineers, tracks the requirement being changed through the use of Ada Structure Graphs (ASGs) maintained in the CASE tool to determine the extent of the proposed change. This provides the project manager with a means of estimating the costs for each change. This information can then be relayed to the customer and cost-benefit analysis undertaken.

3.4.4. System designer/engineer

Engineer's Notebook are used to capture design rationale. This information could prove invaluable throughout life cycle maintenance and in the development of similar or subsequent systems. Although the project is not at the maintenance stage yet, the system designer foresees using requirements traceability extensively in tracing changes to code modules and documentation. The system designer uses the links between design diagrams and requirements captured in the CASE tool to trace proposed requirement changes to the CSU level, thus identifying which modules a change will effect. The system architecture are developed using Ada Structure Graphs (ASGs). Using the traceability tool, the system functional requirements are mapped to the ASGs, providing a level of confidence that the Ada system design was complete.

4. Implementation issues and lessons

In this section, we present several observations from the case study that could potentially benefit other organizations interested in implementing a comprehensive traceability scheme. Wherever applicable, lessons learned and the conclusions drawn below are related to the literature cited in section 1 and the traceability models discussed in section 3.

4.1. Development of traceability schemes

Initially, TIEB had not clearly articulated a formal methodology for its traceability practice. The absence of a well defined model specifying what information needed to be captured by whom, and how it should be used, had led to wide variations in the traceability information maintained by different project participants. For instance, TIEB required design notebooks be maintained by engineers to document design rationale (an example of the DESIGN-based on-RATIONALE link in Fig. 4). However, as there were no clear guidelines on the form and content of this information, the quality, quantity, and usefulness of the design rationale maintained by different engineers were vastly different. Some engineers spent as much as one day every week documenting critical decision rationale, whereas others spent very little time on this activity. One design team created detailed notes on all major design decisions. The notes contained the following information: the problems encountered, the alternatives evaluated, the reason for the desired solution, and critical assumptions affecting that decision. These categories correspond to the elements of design history that Smithers *et al.* [1991] suggest capturing as a part of a traceability scheme. The team documented its failures, heuristics learned and *work arounds* discovered. In contrast, a few engineers initially viewed these design notebooks as personal "memory joggers" and maintained only sketchy details that were not very useful to other project members. As discussed in section 1, a lack of common understanding of the role of traceability diminishes its usefulness. The management of TIEB identified this as a critical problem. To ensure the quality and consistency of rationale captured across the organization, TIEB then defined templates for several reports to be produced using this information. As noted by Curtis *et al.* [1988], facilitation of communication among project team members is critical for project success. In TIEB, traceability to clearly articulated rationale behind critical decisions was used as a mechanism to achieve this. Further, periodic reviews of the material by the project groups were instituted to provide a "chain of accountability" [Cordes and Carver 1989].

TIEB had slowly evolved its traceability practice by carefully identifying different pieces of information that needed to be integrated. For example, the "design notebooks" were separated into two categories: tools/methods related issues and design issues. Common problems encountered with the use of the CASE tools were documented in a central repository accessible to all the users of those tools. Issues specific to design diagrams were attached as notes to those diagrams within the CASE

tool where they were created. As discussed above, these notes included detailed information on major design decisions.

For a traceability scheme to be successful, it is necessary to have complete "buy-in" by all the participants. A project review revealed that often the person who captures a piece of traceability information (e.g., requirements rationale) is not the same person who may use it later (e.g., design engineer). However, the management tried to instill a "life cycle" view of costs and benefits as advocated in recent literature [Edwards and Howell 1992]. It provided clear directives in the task descriptions and adequate allowances in task schedules for the "overhead" involved in capturing traceability information. Further, TIEB also "educated" participants through project meetings to review the role and uses of various parts of the traceability information being captured. There was a clear consensus among the participants that the "quality" of traceability information showed remarkable improvement as a result of this "education".

4.2. Implementation strategies

Senior management at TIEB made an organizational commitment to the introduction of requirements traceability, fully realizing that during the initial stages significant outlays of expenditure in hardware and software tools would be involved. Management also accepted the likelihood of productivity decline due to the additional workload. Finally, they also anticipated schedule changes and delays. All of these factors were taken into account when selecting a pilot project for the introduction of a comprehensive traceability practice. For this reason, an "internal" project, one without stringent deliverable requirements was chosen. Successful implementation of traceability practice was identified as an important measure of project performance. In fact, the budget for documentation costs was set at twice the amount allocated for a similar system of comparable size and complexity. This estimate still fell far short of the actual costs associated with traceability practice. However, the project was considered a success as it achieved the desired goal of implementing a comprehensive traceability practice. This "*loss leader*" strategy made the organizational commitment clear to the project participants ensuring their cooperation.

Given the broad scope of traceability information that needed to be captured and used, TIEB management decided that it would be essential to use a traceability CASE tool. The management also realized that introducing a CASE tool, even in a mature organization, would involve significant costs. Therefore, a commitment was made not to consider the costs of learning the CASE tool as attributable to the project alone, but rather as a cost of process improvement. As Roetzheim [1991] suggests, costs associated with implementing traceability were treated as long term investments to improve the quality of the products and processes at TIEB. These costs were treated as part of the major software process improvement effort that the organization was involved in and were "amortized" over several projects.

Though other CASE tools had been successfully introduced in the organization, introduction of a traceability tool posed special challenges. Some of the participants

did not view the traceability tool as a direct facilitator of their work, especially when the information they captured was for the use of other project participants. Again, TIEB management had to "sell" its view that implementing traceability was an important component in improving systems, a view shared by recent literature [Pohl and Jacobs 1994]. Specifically, TIEB viewed traceability as a way to increase their SEI CMM rating, a goal essential to the very survival of the organization.

4.3. Costs due to lack of traceability

The ADIP project had some unique characteristics that made it suitable for introducing requirements traceability. The project involved re-engineering a system that contained very little documentation or traceability. But a primary requirement of the customer was that the new system be functionally equivalent to the existing system for a variety of operational reasons such as avoiding extensive user re-training. These factors made the project team realize that a lack of comprehensive traceability information can be very costly during the life cycle of a system. In fact, the first step in the project was to clearly identify the original requirements and their rationale. This involved identifying low level requirements resulting from various levels of decisions, such as hardware and interface design decisions. The available "design documents" contained primarily pseudocode and did not contain adequate details on "high level" designs (e.g., structure and data flow diagrams) needed for the project. The development team reviewed trade studies and operators manuals, and examined code line by line in an effort to understand the original requirements and design. Ultimately, the project had to contract at a significant cost, engineers who worked on the original project, to assist in re-creating relevant information. It was estimated that ten employees lost over six months of productive work time, resulting in over 60 worker-months lost. This effort resulted in "reengineering" of requirements with their rationale, designs with their rationale, and system/subsystem components. Further, links among these (shown in Figs. 4 and 5) were also recreated wherever feasible. This experience confirms Baldo's [1990] assertion that traceability to the context of the system development process will help in system reuse and evolution. With the operational life of legacy systems being stretched due to budget pressures (especially in the government sector), scenarios such as the one above may become commonplace.

4.4. Traceability pay-off during system evolution

In ADIP, dependencies among hardware/software/interface requirements (shown as REQUIREMENTS-depend on-REQUIREMENTS link in Fig. 4) were recovered. These efforts had immediate pay-offs when the system had to undergo memory and processor upgrades. The traceability information helped identify important issues to be resolved before the hardware was upgraded. This experience illustrates that identification of dependencies among requirements (as advocated by Pinheiro and Goguen [1996]) is especially critical in large scale embedded systems that often undergo several hardware and software "upgrades" during their useful life span.

4.5. Technical impediments

TIEB realized that there were some serious technical impediments to implementing a comprehensive traceability practice: incompatibility among tools, and tailorability of tools.

In several instances traceability between requirements and designs was impossible to recreate. Therefore, it was not possible to relate some parts of implementations with the original requirements. TIEB realized that any breakdown of traceability linkages at any development phase would render much of the traceability effort in the later phases ineffective. As a result, TIEB advocated the capture and maintenance of requirements traceability information throughout the Systems Development Life Cycle. It was realized that standardizing the tools used throughout the organization was an important step in reaching this goal. However, as an organization that acts as a contractor to a variety of government and private organizations, it is not always feasible for TIEB to use a standardized set of tools. Further, TIEB is often engaged in large scale projects which involve several organizations, all using different platforms and tools.

Many of the tools used in various phases of the project did not exchange information seamlessly, leading to difficulties in maintaining traceability. The following are some examples:

- Cadre's Teamwork was used as a design and prototyping tool. Its interface with the Tartan Ada Cross Compiler used for implementation did not function appropriately, making the exchange of information (and hence the traceability across) design, prototyping and implementation difficult.
- Interleaf was used as the document management system to produce MIL-STD-2167A deliverables as well as other internal documents. Automated filters to transfer the vast amount of design information maintained in Teamwork did not work as desired. Nearly eight person months were spent manually correcting problems resulting from this poor interface to produce documents of the desired quality. Finally, many of the documents produced were *static*, i.e., changes made in one tool were not reflected in the other.
- The CADRE RQT traceability tool (which has since been discontinued) used in the project, did not provide the capabilities needed to produce required reports on costs and schedules related to requirements. Duplicating this information outside the tool involved considerable rework. TIEB realized that the functionality of the CASE tool (selected for a variety of reasons such as cost, compatibility, etc.) may limit its usefulness as a traceability tool.

4.6. Need for automation

Requirements traceability can involve capture and use of vast amount of information. TIEB created the required templates in the CASE tool. Facilities for automatically capturing standard information such as name, project/module, or other

common characteristics greatly enhanced the usefulness of templates, freeing the user from mundane data entry.

As discussed in section 5.3, TIEB had to *recover* aspects of requirement, design and implementation at a considerable cost. However, towards the final stages of the project, TIEB management realized that the costs of recovery could have been reduced if some of the new automated tools that were just entering the market had been available at the beginning of the project. These included tools that

a) can parse textual documents to identify statements meeting specified criteria as requirements and maintain the links between the sources and identified requirements, and

b) reengineering tools that can be tailored to different languages for recovering a variety of "design" representations from code, while maintaining traceability links between them.

5. Conclusions

The experience at TIEB confirms Roetzheim's [1991] observation that traceability can be used a tool to improve system quality. From the senior management to the system maintenance personnel, every person recognized that traceability was essential to the successful completion of a project. The TIEB experience suggests that the costs associated with implementing a comprehensive traceability scheme can be justified in terms of producing a better quality product, better systems development and maintenance processes, and the potential for lower life cycle costs. The organization moved from Level 1 to Level 3 of the SEI CMM and strongly believes that their comprehensive practice of requirements traceability (well beyond the "narrow" interpretation of CMM requirements) was an important factor in this achievement.

Acknowledgements

This research was supported by the Office of Naval Research Engineering Complex Systems project of the Naval Surface Warfare Center Dahlgren Division. The authors also thank Gary Weiss, Carolynn Drudik, Dave Bertiaux and their group at the TIEB for participating in the study.

References

Alford, M. (1991), "Strengthening the Systems Engineering Process," In *Proceedings of NCOSE*, San Jose, CA.

Baldo, J. (1990), "Reuse in Practice Workshop Summery," *IDA*.

Brown, B.J. (1987), "Assurance of Software Quality," In *SEI Curriculum Model SEI-CM-7–1.1* (*Preliminary*), Carnegie Mellon University, Software Engineering Institute.

Conklin, J. and M. Begeman (1988), "gIBIS: A Hypertext Tool for Exploratory Policy Discussion," *ACM Transactions on Office Information Systems 6*, 303–331.

Cordes, D.W. and D.L. Carver (1989), "Evaluation for User Requirements Documents," *Information and Software Technology 34*, 4.

Curtis, B., H. Kransner, and N. Iscoe (1988), "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM 31*, 1268–1287.

DoD (1988), "Defense System Software Development," *DoD-STD-2167A*.

Dorfman, M. and R.F. Flynn (1984), "Arts-An Automated Requirements Specification and Traceability System," *Journal of Systems and Software 4*.

Edwards, M. and S. Howell (1992), "A Methodology for Requirements Specification and Traceability for Large Real-Time Complex Systems," Technical Report, Naval Surface Warfare Center.

Fiksel, J.D. (1994), "New Requirements Management Software Supports Concurrent Engineering," In *CimFlex Teknowledge Corporation*, Washington, DC.

Gathman, T. and D. Halker (1990), *Towards a Manageable Solution to the Iterative Development of Embedded Knowledge-Based Systems*, Rockwell International Corporation.

Gotel, O. and A. Finkelstein (1994), "An analysis of the requirements traceability problem," In *Proceedings of the First International Conference on Requirements Engineering*, Colorado Springs, CO.

Greenspan, S.J. and C.L. McGowan (1978), "Structuring Software Development for Reliability," *Microelectronics and Reliability 17*.

Hamilton, V.L. and M.L. Beeby (1991), "Issues of Traceability in Integrating Tools," In *Proceedings of the Colloquium by the Institution of Electrical Engineers Professional Group C1* (*Software Engineering*), *London*.

IEEE (1984), "IEEE Guide to Software Requirements Specifications," ANSI/IEEE Standard 830–1984, New York, USA.

Jackson, J. (1991), "A Keyphrase Based Traceability Scheme," In *Proceedings of the Colloquium by the Institution of Electrical Engineers Professional Group C1* (*Software Engineering*), *London*.

Jarke, M., J. Bubenko, C. Rolland, A. Sutcliff, and Y. Vassiliou (1993), "Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis," In *Proceedings of the First IEEE International Symposium on Requirements Engineering*, pp. 19–31.

Jarke, M. and K. Pohl (1993), "Establishing Visions in Context: Toward a Model of Requirements Engineering," In *14th International Conference on Information Systems*.

Keuffel, W. (1990), "Extra Time Saves Extra Money," *Computer Language*.

Lee, J. (1990), "SIBYL: A qualitative decision management system," In *Artificial Intelligence at MIT: Expanding Frontiers*, P. Winston and S. Shellard, Eds., MIT Press, Cambridge, MA, pp. 106–133.

Maclean, A., R. Young, V. Bellotti, and T.P. Moran (1991), "Questions, Options, and Criteria: Elements of Design Space Analysis," *Human Computer Interaction 6*, 201–250.

Macmillan, J. and J.R. Vosburgh (1986), *Software Quality Indicators*, Scientific Systems.

Marconi Systems Technology (1991), *RTM Requirements and Traceability Management* (*Product Overview*), Arlington, VA.

McCausland, C.D. (1991), "A Case Study in Traceability," In *Proceedings of the Colloquium by the Institution of Electronic Engineers Professional Group C1* (*Software Engineering*), *London*.

Murine, G. (1986), "Secure Software's Impact on Reliability," *Computers and Security 5*.

Pinheiro, F.A.C. and J. Goguen (1996), "An Object-Oriented Tool for Tracing Requirements," *IEEE Software*, 52–64.

Pohl, K. and S. Jacobs (1994), "PRO-ART: PROcess based Approach to Requirements Traceability," NATURE Report Series 94–07, RWTH Aachen, Germany.

QSS Ltd (1995), *Dynamic Object Oriented Requirements System, Reference Manual, version 2.1*, Oxford.

Ramesh, B. and V. Dhar (1992), "Supporting Systems Development by Capturing Deliberations During Requirements Engineering," *IEEE Transactions On Software Engineering 18*, 498–510.

Ramesh, B., T. Powers, C. Stubbs, and M. Edwards (1995), "Implementing Requirements Traceability," In *Proceedings of the IEEE International Symposium on Requirements Engineering*, *York, UK*, pp. 89–95.

Roetzheim, W.H. (1991), *Developing Software to Government Standards*, Prentice Hall, Englewood Cliffs, NJ.

Schneidewind, N. (1982), "Software Maintenance," Technical Report, Naval Postgraduate School.

Smithers, R., M.X. Tang, and N. Tomes (1991), "The Maintenance of Design History in AI-based Design," In *Proceedings of the Colloquium by the Institution of Electrical Engineers Professional Group C1* (*Software Engineers*), *London*.

Stehle, G. (1990), "Requirements Traceability for Real Time Systems," In *Proceedings of EuroCASE II*, *London*.

TD Technologies (1996), *SLATE User Manual*, Dallas, TX.

West, M. (1991), "The Use of Quality Function Deployment in Software Development," In *Proceedings of the Colloquium by the Institution of Electrical Engineers Professional Group C1* (*Software Engineering*), *London*.

Wright, S. (1991), "Requirements Traceability – What? Why? and How?," In *Proceedings of the Colloquium by the Institution of Electrical Engineers Professional Group C1* (*Software Engineering*), *London*.