Requirements Traceability for Embedded Software — an Industry Experience Report

Leesa Murray*¶ ‡ Alena Griffiths† Peter Lindsay‡ Paul Strooper* ‡

Abstract

Requirements traceability is an enabling capability for verification and validation of software-based systems. This paper reports on the outcomes of an industrial pilot project investigating the costs and potential benefits of adding requirements traceability to the development lifecycle of embedded software. The project forms part of an industry research collaboration which aims to increase integrity assurance for products containing embedded software.

KEY WORDS
Software requirements, tracing, embedded software

1 Introduction

Requirements traceability concerns the ability to identify requirements at different levels of abstraction, and to show that they have been implemented and tested. Traceability is a key capability for demonstrating and assessing software and system integrity, and is typically mandated by safety and reliability standards [8, 16, 17]. It can also play a key role in fault avoidance and fault removal, and for assuring dependability of embedded software.

This paper is an experience report on a project which used requirements traceability to analyse the development artefacts of three existing embedded software subsystems. The embedded software is also referred to as firmware. The work was carried out in a pilot project within the firmware development group of Foxboro Australia, a company which produces software-based control and automation systems used in transport and electricity distribution applications. The firmware is part of a range of products, some of which are used in safety-related systems. The project's findings are relevant to most high assurance software, not simply firmware.

Besides being mandated by safety and reliability standards, requirements traceability is a requirement of some of the company's customers. Foxboro already performs requirements tracing for some of its project-specific software, but wanted to extend traceability into the firmware which forms part of their standard product line, to increase assurance of integrity. Another motivation was to use traceability as an enabling technology for fault avoidance and fault removal, by supporting the following processes: checking coverage of requirements during design, to reveal overlooked requirements earlier in the lifecycle; test planning, by ensuring that all requirements are covered and by assessing the ratio of requirements to test cases; problem analysis, by helping identify the root causes of failures in specification, design, implementation, and/or test; and characterisation of products, product versions and product variants, by providing tracing between diverse sources of requirements.

There are many issues to consider when determining what tracing relationships should be created and maintained. Firmware requirements arise from many different sources, including core-product specifications, project-specific and customer requirements, third-party specifications such as standardised communications protocols, and hardware-specific requirements. The requirements typically reside in a wide range of documents, including system and component specifications and design documentation, and at varying degrees of abstraction and granularity. There are many possible tracing relationships both within and between the above documents, and through design to implementation and testing. As a result, there are many combinations of tracing relationships to choose from, each with associated costs and benefits.

In the pilot project, we conducted requirements tracing on development artefacts associated with three existing firmware subsystems. The project sought to include subsystems of different functional types, and possessing different requirement types, including in-house and third-party requirements. The tracing relationships included traces from functional requirements to design requirements, design requirements to test cases, and design requirements to source code. Rational Software Corporation's RequisitePro [14] requirements tracing and management tool was used, as it was already in use in other departments at Foxboro. Metrics were kept on resource usage, including learning time, the requirements identification process, and the trace creation process. At the end of the project, an assessment was made of the value of the information that the tracing exercise revealed. Two classes of recommendations were made. The first class involved direct recommendations to address the assurance gaps that the tracing information revealed. The second class involved recommendations that will permit the incorporation of requirements traceability

*School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Qld 4072, Australia. email: { leesam, pstroop }@itee.uq.edu.au

†Foxboro Australia, PO Box 4009, Eight Mile Plains, Qld 4113, Australia. email: { leesa.murray, alena.griffiths }@invensys.com

‡Software Verification Research Centre, The University of Queensland, Brisbane, Qld 4072, Australia. email: pal@svrc.uq.edu.au

as a standard part of product development, going forward.

The remainder of this paper describes the pilot project and its outcomes, and draws conclusions on the cost-effectiveness, benefits, and feasibility of the requirements tracing process. The paper is structured as follows: Section 2 discusses related work; Section 3 describes the goals of the pilot project and the nature of the subsystems to which traceability was added; Section 4 describes the outcomes of the pilot project; Section 5 discusses the significance of these outcomes; and Section 6 concludes.

2 Related Work

Jarke [9] defines requirements traceability as "the ability to describe and follow the life of a requirement, in both a forward and backward direction, ideally through the whole systems lifecycle." Jarke acknowledges that establishing and maintaining requirements traceability can be an expensive and politically sensitive endeavour, as developers are not exactly known for their love of documentation. He suggests that traceability should come as a side effect of daily productive work rather than via the imposition of additional bureaucracy. Our tracing was performed retrospectively after the product development was complete, and we agree with Jarke that it would be more cost-effective if incorporated as an integral part of the development process.

Ramesh et al. [13] present a framework for representing and developing a traceability scheme. Requirements traceability is a characteristic of a system where requirements are linked to their sources (backwards traceability) and to the artefacts created during the system development lifecycle based on those requirements (forwards traceability). A primary use of requirements traceability is for developers to prove to the customer that the requirements are understood, and the product complies with the requirements and has no unnecessary features. Therefore, it facilitates communication among those involved in a project. Ramesh [12] states that "requirements traceability is viewed as a measure of system quality and is mandated by many standards governing the development of systems." From his analysis of empirical studies conducted over four years, Ramesh identifies two distinct groups — low-end traceability users, who view traceability simply as a mandate from project clients and management, and high-end traceability users, who view traceability as an important component of a quality systems engineering process.

Kotonya and Sommerville [10] identify requirements management as "the process of managing changes to the system requirements." This is more easily achieved when traceability information is explicitly recorded. They suggest that an organisation maintains a set of traceability policies that should specify:

- the traceability information that should be maintained,
- the techniques that should be used to maintain the traceability information,
- a description of when the traceability information should be collected during the requirements engineering and system development process,
- a description of how to handle and document policy exceptions, and,
- the process used to ensure that the traceability information is updated after a change has occurred.

From the results of the pilot project, recommendations similar to these policies have been made to Foxboro for adoption by the firmware group.

Petty [11] recognises the two extremes of requirements engineering, stating that for many projects the requirements effort is almost nonexistent with requirements provided verbally and never documented, and for others requirements efforts can produce formal, military-style specifications. He explains that with embedded software, a product is being developed that has some of its functionality provided by the software. Ultimately it is the product that is sold, not the embedded software. His work makes it clear that a crisp distinction is essential to identify those requirements to be implemented in software, as opposed to those requirements that refer to the hardware or to the product generally.

Hammer et al. [5] and Rosenberg et al. [15] present project data collected from NASA's Goddard Flight Centre on requirements definition, verification, and management. Tools and metrics analysis are used in these three areas in an attempt to get requirements correct, the first time. The authors state that "prior to processing of requirements, the schema for the requirements management database must be developed", and discuss some critical issues associated with the schema design. The authors also discuss the tracing of requirements to test cases, and the ratios of requirements to test cases.

3 Pilot Project

The pilot project trialed requirements tracing processes and Rational Software Corporation's tracing tool, RequisitePro [14], for firmware subsystems. The goals of the project were to:

- Carry out requirements tracing on the development artefacts associated with a number of firmware subsystems. This included requirements traceability from functional specifications to design specifications, from functional and design specifications to test descriptions, and from design specifications to source code. The tracing was to be performed using a tool to store the tracing information.
- Analyse the information obtained from the tracing exercise, to see what it revealed about product quality and assurance, and to assess whether this information was considered valuable, relative to the cost of obtaining it.

- Investigate the feasibility and cost-effectiveness of the different traceability relations.
- Make recommendations for the incorporation of a requirements tracing process in the firmware development lifecycle, including the definition of document templates to assist in requirements traceability.

As the project sought to trace requirements from high levels of abstraction through design to source code and test cases, we use the generic term *trace item* to refer to a requirement at any level of abstraction, a source code fragment, or a test case.

A Goals-Question-Metrics (GQM) [1] exercise was conducted to identify metrics data to be collected during the project. The first goal for the GQM exercise was to *implement requirements tracing on the development artefacts associated with a number of existing firmware subsystems.*

Based on the GQM exercise, we decided to collect data on the types and numbers of documents being traced, the types and numbers of trace items, the types of tracing relationships, and the number and coverage/completeness of the traces for each of these tracing relationships.

The second goal from the GQM exercise was to *assess the feasibility and cost-effectiveness of requirements tracing in the firmware group.* The data collected for this goal relate to the cost of the requirements tracing and the problems that the requirements tracing is able to find.

The data collected during the pilot project is presented and analysed in Section 4. The majority of the data was gathered by using the querying facilities of RequisitePro.

For the pilot project, subsystems were selected representing two different aspects of firmware — core product, where the requirements and subsequent documentation were written within the firmware group; and communications protocols, where requirements, and test procedures and cases, were written by an industry-recognised standardising body. Three firmware subsystems were selected for this pilot: the Digital Inputs (DI) subsystem, the Power Supply and Battery Charger (PSBC) subsystem, and the Level 1 Slave subsystem of the Distributed Network Protocol 3.0 (DNP).

The DI and PSBC subsystems are part of one of Foxboro's Remote Terminal Unit (RTU) products. This product was chosen for the pilot project as it is a product that Foxboro continues to both maintain and evolve. As such, the information obtained through this exercise was expected to be of immediate value to Foxboro. All of the product's documentation was written by the firmware group.

The DI subsystem was chosen to trial tracing amongst the following trace items: functional requirements (abbreviated as FRS), design requirements (DRS), firmware test cases (FTC), and source code fragments (SRC). This allowed us to explore tracing throughout the complete firmware development lifecycle. The lower part of Figure 1 illustrates this. The boxes represent development artefacts which contain trace items of a certain kind, and the arrows represent the relationships between the trace items that we analysed.

In contrast, the PSBC subsystem was selected to trial the tracing of functional requirements (FRS) directly to test cases (FTC). The upper part of Figure 1 illustrates these trace items and relationships.

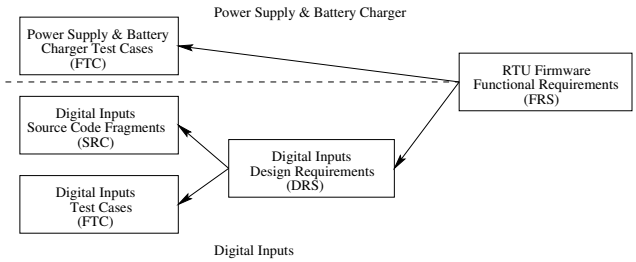

Figure 1. RTU trace items and relationships.

The DNP subsystem was chosen for this pilot as it has requirements and testing documentation that is written by the DNP Users Group, as well as some written by the firmware group. This provided the opportunity to trial the tracing process on third party produced documents (that is, the DNP Users Group produced documents) as well as those written by the firmware group. The level 1 slave subset of DNP was chosen to trial tracing of subset definitions [2] (DNP Users Group) to slave device profile requirements (firmware group) to functional requirements (firmware group) to application layer requirements [3] (DNP Users Group) to test cases [4] (DNP Users Group). Figure 2 illustrates this. Being able to show that a product is DNP level 1 compliant, and to identify the test cases to execute for regression testing when changes occur to the subsystem, were the two major factors in the selection of the tracing for the DNP slave subsystem.

Document templates were developed for firmware functional specifications and firmware test descriptions as part of the pilot project. Prior to the pilot project, templates did exist for these documents within the firmware group.

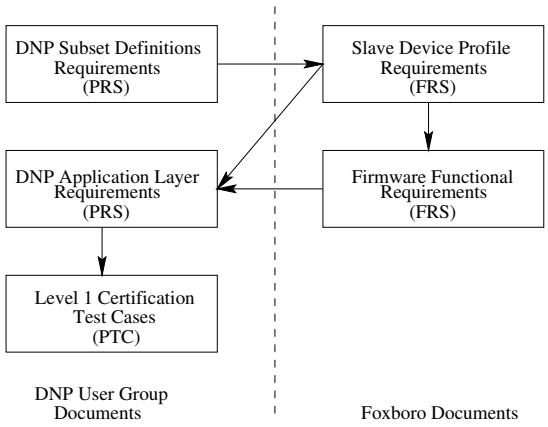

Figure 2. DNP trace items and relationships.

The existing templates were enhanced to better support requirements traceability for future firmware developments.

The resultant *"Firmware Functional Specification Template"* was written in accordance with the *"ANSI/IEEE Standard 830-1984 IEEE Guide to Software Requirements Specification"* [7]. Guidance on what to include in each section was provided in the form of hidden text which can be viewed by the user. The guidance given is detailed and includes examples where relevant. Specific references are made to requirements traceability. The resultant *"Firmware Test Description Template"* was written in accordance with the *"IEEE-829 — IEEE Standard for Software Test Documentation"* [6] and Foxboro's company procedure for test plans. It provided guidance on what to include in each section, again through the use of hidden text.

4 Outcomes and Findings

RequisitePro was chosen as the tracing tool because it was already in use in other development groups within Foxboro. Two databases were created in RequisitePro: one for the tracing associated with the DI and PSBC subsystems, and one for the DNP tracing. The majority of RequisitePro's functionality was used during the pilot project, except its facilities for version control.

Following database setup, relevant firmware artefacts (functional specifications, design specifications, source code, and test descriptions) were gathered and added to the database. Table 1 shows the number of each document type used in the pilot project.

Document types	Number Used
Firmware functional specification	3
Firmware design specification	2
Firmware test description	2
Protocol functional specification	2
Protocol test description	1
Source code file	9

Table 1. Document types.

Following the identification of the development artefacts, the trace items within those artefacts were identified. As illustrated in Figure 1, the trace item types used in the RTU project were: firmware functional requirement (FRS), firmware design requirement (DRS), firmware test case (FTC), and source code fragment (SRC). The trace item types used in the DNP project were: firmware functional requirement (FRS), protocol functional requirement (PRS), and protocol test case (PTC). Table 2 shows the number of items identified for each trace item type and their location, with a "—" representing that this data was not applicable or collected.

The exercise of trace item identification revealed a number of interesting facts. Firstly, as its name suggests, the RTU functional specification is in nature a product specification, containing both firmware and hardware requirements. For this reason, trace item types for firmware functional requirements and hardware functional requirements (HRSs) were created for the RTU project in the RequisitePro database. Since the HRSs were not traced any further during this pilot project they are not included in Table 2. Secondly, as Table 2 shows, the location of a trace item does not always reflect its type. That is, while one might typically expect the design specifications to collectively contain all trace items of type DRS, and no trace items of type FRS, in fact trace items of both types were found in the design specifications. This suggests that the RTU functional specification, encompassing both hardware and software requirements, is too broad in scope to include all functional firmware requirements at a level of abstraction that permits easy traceability to design. This points to the need for an intermediate level of abstraction in the documentation tree, which is one of the recommendations from this pilot project.

Subsystems:	DI	PSBC	DNP
FRSs	88	7	198
FRSs in functional specs	54	0	198
FRSs in design specs	34	7	0
DRSs	95	—	—
DRSs in functional specs	0	—	—
DRSs in design specs	95	—	—
FTCs	12	8	—
SRCs	150	—	—
PRSs	—	—	499
PRSs in protocol func specs	—	—	499
PTCs	—	—	38

Table 2. Trace items types - identification and location.

Subsystem tracing relation	Traces
DI — FRS $\rightarrow$ DRS	307
DI — DRS $\rightarrow$ FTC	46
DI — DRS $\rightarrow$ SRC	302
PSBC — FRS $\rightarrow$ FTC	16

Table 3. RTU trace relationships

Following trace item identification, tracing relations between those trace items were established. This involved one or more firmware engineers observing that a trace item of one type, say an FRS, logically traced to one or more other trace items, such as DRSs or FTCs.

Table 3 identifies the data for the tracing relation types used for the RTU subsystem. Table 4 shows the tracing data for the DNP tracing.

Subsystem tracing relation	Traces
DNP — PRS → FRS	165
DNP — FRS → FRS	370
DNP — FRS → PRS	2962
DNP — PRS → PTC	527

Table 4. DNP trace relationships

4.1 Traceability analysis

Following the identification of the tracing relationships, the database of tracing relations was then queried to reveal information about potential product or assurance deficiencies. For example, if one considers a trace relationship between design requirements and source code, then an observation that there are design requirements which do not trace to source code fragments may indicate that all requirements have not been implemented. Similarly, an observation that there are source code fragments which do not trace from any design requirements may indicate the introduction of undocumented features to the product. Analogous observations about the relationship between requirements and test cases would point to the existence of untested features, or of unnecessary test cases, respectively. Care is always required in interpreting the trace data, however, as observations of the kind mentioned may point to product quality or assurance deficiencies, but they can also simply point to a deficiency in the trace relation database.

Also, where the relationship between trace items is not one-to-one, one should not assume that because there is a trace from a particular requirement, that the requirement has been adequately implemented or tested. It is usually the case that a requirement will trace to a number of test cases and a number of lower level design requirements, so some semantic check for coverage is required, and in the general case, this cannot be automated.

For each tracing relation from trace items of type X to type Y, the number of items of type X that were **not traced from** was calculated, as was the number of items of type Y that were **not traced to**. These two measures were also expressed as percentages of the total number of items of type X and Y respectively, to assess the extent of the coverage deficiency or feature creep.

All "**not-traced-from**" and "**not-traced-to**" observations were then analysed by Foxboro personnel and research staff. This analysis was very detailed, relating to specific aspects of individual requirements, and for this reason it is not repeated here. However, the exercise gave rise to some more general observations.

Firstly, the observations are only as useful as the accuracy of the traceability database, so it is vital that this is accurate. The analysis of anomalies indicated that many "anomalies" were actually attributable to misclassification of requirements, missed traces, failure to identify all relevant development artefacts, etc. Many of these problems are due to the fact that the exercise reported in this paper involved an analysis of existing development artefacts. If the process of requirements traceability was an integral part of development, then the trace relations could be established during development, and the database reviewed as part of the normal development review processes. It is expected (and indeed, it has been Foxboro's experience in other areas of software development) that this would lead to cleaner levels of abstraction and improved requirements quality.

Secondly, not all trace items of a certain type will necessarily participate in a particular trace relation, so more sophisticated trace item subtyping is required in future. For example, a performance requirement will not usually be traceable to a specific fragment of source code, and so does not participate in the design to source code trace relation. It is possible to record this information in RequisitePro, and exclude such trace items from queries that seek to identify tracing anomalies.

Thirdly, the analysis revealed cases where assumptions had been made. For example, an aspect of the design may rest on an assumption about the operating system or its environment. A mechanism in the traceability process for documenting such assumptions and tracing from them needs to be incorporated.

Fourthly, where re-use occurs, this should be made explicit. For example, in one instance, the existence of functional requirements that were not traceable to design requirements was attributed to the fact that the functionality had been ported from a previous generation of the product and was not re-implemented. Such re-use should be made clear and used to close out higher level requirements.

Finally, a requirements traceability graph in which the number of traces from a particular trace item is more or less uniform, for all trace items of a particular type, can indicate either good use of abstraction or good test coverage. For example, if only a few functional requirements trace to a disproportionately large number of the design requirements, this may indicate under-specification, at the functional requirements level. Similarly, a trace relation from requirements to test cases in which some requirements trace to many test cases, and others trace to few or even no test cases, may indicate an imprudent distribution of test effort.

4.2 The cost of requirements tracing

It took 4.75 hours for one engineer to create the requirements database in RequisitePro for the RTU project, and 1 hour to set up the database for the DNP project. The requirements identification in the RTU functional specification took one engineer 16.5 hours to perform.

Table 5 shows the data for how long it took to identify the requirements (firmware functional requirements, firmware design requirements, source code fragments, and firmware test cases) for the RTU project, and how long it took to perform the tracing for the different tracing relations. Table 6 shows similar cost data for the DNP trac-

ing. In both cases, the requirements identification was performed by one engineer, and the tracing was performed by two engineers working together at the same time.

Subsystems:	DI	PSBC
DRS identification	7.25 hrs	—
FTC identification	1.5 hrs	1.5 hrs
SRC identification	3.0 hrs	—
Trace FRS to DRS	11.0 hrs	—
Trace FRS to FTC	—	0.5 hrs
Trace DRS to FTC	1.5 hrs	—
Trace DRS to SRC	5.25 hrs	—

Table 5. RTU requirements identification and tracing costs.

In the case of the RTU subsystems, the analysis of the anomalies revealed by querying the trace database took a total of four hours, and involved a member of the research team and a Foxboro engineer.

Subsystems:	DNP
PRS ident in subset definitions document	4.25 hrs
FRS ident in slave device profile	1.5 hrs
FRS ident in firmware functional spec	2.0 hrs
PRS ident in application layer document	19.5 hrs
PTC ident in certification procedure	2.0 hrs
Trace PRS to FRS	2.5 hrs
Trace FRS to FRS	1.5 hrs
Trace FRS to PRS	3.25 hrs
Trace PRS to PTC	3.5 hrs

Table 6. DNP requirements identification and tracing costs.

In the case of the DNP subsystem, the engineers analysed the trace anomalies as they performed the tracing exercise.

Given that the engineers who performed the tasks in this pilot project had no previous experience with requirements tracing, and had never used RequisitePro, the tracing costs are considered to be quite low. This is especially so considering the added assurance that the existing tracing now provides, and the problems that the tracing has highlighted.

5 Discussion

The tasks involved in this pilot project were all accomplished retrospectively, that is, the requirements identification tasks occurred after the documents had been written, reviewed, and accepted; not during the document creation and review phases, which would be ideal. Similarly, the tracing occurred after the subsystems had been implemented. Ideally tracing would occur as part of the development lifecycle phases, because this is the most efficient time to do so. This catches problems at their source and helps prevent them propagating throughout the development.

The RTU tracing project highlighted many interesting issues as we explored requirements identification and tracing across the firmware development lifecycle. From our experience, we believe that the RTU firmware functional specification is more like a product specification. The location of functional requirements in the subsystems' design documents reinforces this and identifies the need for another level of documentation between the current functional specification and design documents.

The DNP tracing involved a mix of in-house and third party documents. Maintaining the requirements database when in-house documents are updated is not as big a concern as for changes to third party documents. This is because the authors of in-house documents can update the requirements databases as part of the maintenance exercise and RequisitePro has facilities that recognise what requirements have changed. However, if the third party documents are changed, and obviously these changes will not occur from within RequisitePro, maintenance problems will arise. When using third party documents in requirements tracing, we obviously need to consider the overheads of radical changes occurring in new versions, and identify ways of managing these changes while keeping the requirements database current. These same maintenance issues apply to source code, as source code files are not Microsoft Word files, and therefore we cannot exploit RequisitePro's facilities for managing changes to source code files.

The personnel involved in the pilot project were not always the authors of the documents being used. This perhaps helped to highlight problems in the granularity and interpretation of requirements, rather than hindering the tracing exercise. To date, no independent verification of the tracing has occurred, as this was not a goal or task of the pilot project. However, reviewing the requirements identified and tracing as part of the document review process would only improve the quality of the documents and the system.

This pilot project has shown that requirements tracing for firmware is feasible. It attracts a low cost for reasonable benefit. Once integrated into Foxboro's firmware development lifecycle, the costs are expected to decrease further because the tracing will occur when the information is fresh in the engineers' minds. The benefits will also be more obvious as problems such as those discussed above will be identified at their source and prevented from propagating throughout the development.

6 Conclusions

In summary, a pilot project was used to analyse the development artefacts of three existing embedded software (firmware) systems. Requirements tracing was performed for a variety of requirements, including core-product requirements and third-party specifications. Requirements

tracing was carried through to firmware design specifications, source code and test descriptions.

The exercise revealed some deficiencies with existing development and testing methods, particularly with the associated documentation. The main problems identified were a large variation in the level of granularity of requirements in documents for similar type requirements, and the tracing between requirements in industry-standard documents and in-house documents. Functional requirements were often only recorded in design documents and not in subsystem specifications. Another problem was that too high a level of abstraction was used for functional requirements in subsystem specifications, resulting in excessive "fan out" of requirements. One direct outcome of the exercise was the development of improved document templates and documentation procedures, to address these problems and to assist in the adoption of the requirements tracing process.

Metrics data collected during the pilot project revealed that requirements identification and tracing is low cost, even when applied retrospectively. Although the precise cost to develop the artefacts used in the study is not available, the cost of requirements tracing is likely to be small compared to the total development cost. Additional benefits, by way of removing faults at their source, can be anticipated if tracing is incorporated directly into the development process during design, coding and test planning. A proposal has been made for the integration of requirements identification and tracing into the firmware development process.

Based on findings in the pilot project, we have recommended that the following are the most important tracing relationships to create and maintain: product functional requirements to firmware functional requirements; firmware functional requirements to firmware test case descriptions; and firmware functional requirements to firmware design requirements.

While we believe there is value in tracing from firmware design requirements to firmware test descriptions, we feel that the cost may not always be justified given that the risks of not performing this tracing can be mitigated by the recommended tracing relationships above. At this point we do not recommend tracing to source code because of the high cost of the tracing given that the files cannot be maintained in RequisitePro. Note also that the risk of unimplemented functionality can be mitigated through systematic module testing.

Acknowledgements

The work reported in this paper is supported by the ARC Strategic Partnerships with Industry - Research and Training (SPIRT) grant C49937058. We thank the firmware group at Foxboro for their assistance in this pilot project, in particular Paul Ellis, Jenny Burton, Patrick Johns, Andrew Lockwood, and Owen McNamara. We thank Brenton Atchison and Anthony MacDonald for their comments on previous versions of the paper.

References

[1] V. Basili and H. Rombach. The TAME project: towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6):759–773, 1988.

[2] DNP Users Group. *DNP V3.00 Subset Definitions*, 1995. Document number: P009-01G.SUB, Version 1.0.

[3] DNP Users Group. *DNP V3.00 Application Layer*, 1997. Version 0.03.

[4] DNP Users Group. *DNP V3.00 Intelligent Electronic Device (IED) Certification Procedure Subset Level 1*, 1999. Version 1.00.

[5] T. Hammer, L. Huffman, L. Rosenberg, W. Wilson, and L. Hyatt. Doing requirements right the first time! In *Software Technology Conference*, 1998.

[6] IEEE Computer Society. *IEEE Standard for Software Test Documentation*, 1983. ANSI/IEEE Std. 829–1983.

[7] IEEE Computer Society. *IEEE Guide to Software Requirements Specification*, 1984. ANSI/IEEE Std. 830–1984.

[8] International Electrical Commission. *Functional Safety: Safety-Related Systems. International Standard IEC 61508*, 1999.

[9] M. Jarke. Requirements tracing. *Communications of the ACM*, 41(12):32–36, 1998.

[10] G. Kotonya and I. Sommerville. *Requirements Engineering - Processes and Techniques*. Wiley, 1998.

[11] B. Petty. Requirements management using tables. *Embedded Systems Programming*, pages 54–60, Dec. 1998.

[12] B. Ramesh. Factors influencing requirements traceability practice. *Communications of the ACM*, 41(12):37–44, 1998.

[13] B. Ramesh, C. Stubbs, T. Powers, and M. Edwards. Requirements traceability: Theory and practice. *Annals of Software Engineering*, 3:397–415, 1997.

[14] Rational Software Corporation. *RequisitePro Installation Guide and User Manual*, 1997.

[15] L. Rosenberg, T. Hammer, and L. Huffman. Requirements, testing, and metrics. In *15th Annual Pacific Northwest Software Quality Conference*, 1998.

[16] RTCA Inc. *Software Considerations in Airborne Systems and Equipment Certification, RTCA/DO178-B*, 1992.

[17] UK Ministry of Defence. *Def Stan 00-55: Requirements for Safety Related Software in Defence Equipment*, 1997.