Appears in Proceedings, Third IEEE International Symposium on Requirements Engineering (RE'97), January 5-8th, 1997, Annapolis, Maryland, USA.

Formal Methods for V&V of partial specifications: An experience report

Steve Easterbrook and John Callahan
{steve,callahan}@cs.wvu.edu
NASA/West Virginia University Software IV&V Facility
100 University Drive
Fairmont, WV 26554

Abstract

This paper describes our work exploring the suitability of formal specification methods for independent verification and validation (IV&V) of software specifications for large, safety critical systems. An IV&V contractor often has to perform rapid analysis on incomplete specifications, with no control over how those specifications are represented. Lightweight formal methods show significant promise in this context, as they offer a way of uncovering major errors, without the burden of full proofs of correctness. We describe an experiment in the application of the method SCR to testing for consistency properties of a partial model of the requirements for Fault Detection Isolation and Recovery on the space station. We conclude that the insights gained from formalizing a specification is valuable, and it is the process of formalization, rather than the end product that is important. It was only necessary to build enough of the formal model to test the properties in which we were interested. Maintenance of fidelity between multiple representations of the same requirements (as they evolve) is still a problem, and deserves further study.

1 Introduction

Requirements engineering methods typically provide a set of notations for expressing software specifications, together with tools for checking properties of specifications, such as completeness and consistency. In general, such methods demand a full commitment. It is assumed that the method will be used to construct a complete specification, which will then act as a baseline for subsequent development phases. However, to validate and verify large specifications for safety-critical real-time systems, it is sensible to apply a number of different methods, to overcome weaknesses and biases of each individual method. For example, a formal method might be used to model a critical portion of an informal specification, to check safety and liveness properties of that portion. In order to manage the application of multiple methods, it is necessary to develop and maintain alternative representations of partial specifications, and to express the relationships between them.

This paper describes some preliminary work on the use of formal specification as a tool for Independent Verification and Validation (IV&V). Our intention is to use formal methods not as a part of the development process itself, but as a 'shadow' activity, performed by an independent team of experts. Our long-term expectation is that this approach will turn out to be a less painful way of introducing formal methods into well-established, large-scale software development processes.

There are a number of questions that need to be addressed before formal methods can be used in this way. Most published case studies of formal methods have focussed on the use of a formal specification as a baseline from which design and code can be verified [3]. In contrast, we have been applying formal methods for intermittent "spot checks" to test for errors as the requirements evolve. The term "lightweight formal methods" has been used to describe this approach [15]. In this context, the the formal specification is dispensable – what is important are the insights gained from *the process of* formalizing partial views of the requirements and from validating properties of the resulting models. However, it is still necessary to demonstrate fidelity between the original (informal) specification, and the formal model. Furthermore, iterative application of this approach can be greatly facilitated if the relationships between the partial views are captured.

The context for this work is the development of software for the International Space Station (ISS) project. Boeing Space and Defense Group Houston (Prime) is responsible for supervising the overall development and integration of International Space Station software. There are three Product Groups (PGs), McDonnell Douglas Aerospace, Rockwell Aerospace - Rocketdyne and Boeing Space and Defense Group Huntsville, who are developing several key Computer Software Configuration Items (CSCIs). There are also several International Partners (IPs) including Russia, Japan, Canada, and the European Space Agency, who are developing software that will need to be incorporated into ISS. With over 45 flight computers and an estimated 1.1 million source lines of flight code, the potential problems are considerable. Software IV&V is currently being performed by Intermetrics, under an interim contract. The Intermetrics team is based at Fairmont, W.Va., with personnel stationed in Houston and Huntsville in order to interact with the development teams.

In section 2, we outline the IV&V process, and discuss the aspects of this process that hinder effective IV&V. With this as background, the remainder of the paper focuses on the use of methods and tools within this process. We present two experiments in the use of formal specification. For these we used a combination of AND/OR tables [8], and the Software Cost Reduction (SCR) approach [9]. The first experiment involved the translation of a portion of the Fault Detection, Isolation and Recovery (FDIR) specification into a formal notation. This experiment confirmed that the natural language used in the Software Requirements Specification (SRS) documents is inherently ambiguous, and that the task of generating formal specifications from this documentation is fraught with difficulty. In the second experiment, we applied an automated consistency checking tool, to test some formal properties of the specification. Although this experiment demonstrated that important disjointness properties did not hold, the results did not add any more value to the analysis. The first experiment had already demonstrated that the way in which these requirements were expressed was a problem. The errors found in the second experiment were attributable to the same problem.

Application of formal methods in this context was not always easy. The informal specification from which we derived our models did not permit an easy translation into a state-based model. We encountered severe problems in demonstrating fidelity, and providing traceability between the two. Section 5 discusses these problems, and sketches out further work aimed at eliciting relationships between partial specifications by extracting information from fine-grained process capture.

We conclude that in an IV&V context, the analytical benefits offered by formal methods have to be weighed against the effort needed to maintain fidelity between a formal model and the informal specification used by the development team. An IV&V team needs to be able to perform partial analyses on partial specifications, without being tied to any one formalism. The analysis carried out must be sufficient to reveal important problems, as opposed to surface defects. Further analysis is a waste of effort until these problems have been fixed. This conclusion implies a change of perspective for the use of formal methods: while the specification is still evolving it is important to identify quickly any major defects; it is not necessary to perform a complete analysis. Tools that are geared towards finding and characterizing such problems (E.g. SCR* [10], Nitpick [11], etc.) are more useful than tools geared towards proving correctness (E.g. theorem provers).

2 The IV&V Process

For *Independent* Verification and Validation (IV&V), the software customer hires a separate contractor to analyze the products and process of the software development contractor. This analysis is performed in parallel with the development process, throughout the software lifecycle, and in no way replaces in-house V&V. IV&V is applied in high-cost and safety-critical projects to overcome analysis bias and reduce development risk. The customer relies on the IV&V contractor as an informed, unbiased advocate to assess the status of a project's schedule, cost, and the viability of its product during development. In full IV&V, the IV&V contractor has managerial, financial and technical independence, and reports to the customer, not the developer. Most importantly, the IV&V contractor should be engaged as early as possible in the project: studies have shown that IV&V has the biggest impact in the early phases, especially in the requirements phase [13].

An example IV&V activity is the analysis of specifications on the Space Station project. An SRS is written by the relevant development contractor for each Software Configuration Item (CSCI). These are written in natural language, and follow the format of DOD-STD-2167A. The IV&V contractor periodically receives copies of the SRS documents, in various stages of completion. These are analyzed for technical integrity by the IV&V contractor, in order to identify any requirements problems and risks. The kind of analysis performed will vary according to the level and the type of specification, and will cover issues such as clarity, testability, traceability, consistency and completeness. If problems are identified, the IV&V contractor may recommend that either the requirements be rewritten, or the problem be tracked through subsequent phases.

Performing IV&V on large projects is far from straightforward. Problems faced by the IV&V contractor include:

resource allocation – A complete, detailed analysis of the entire system is infeasible. Effort has to be allocated so as to maximize effectiveness. For example, a criticality and risk analysis might be performed to determine which components need the most scrutiny. Timing is also a factor; effort needs to be allocated at the right points in the development of a product (e.g. a document), so that the product is mature enough to be analyzed, but not so mature that it cannot be changed.

short timescales – To be most effective, IV&V reports are needed as quickly as possible. There is always a delay between the delivery of an interim product to the IV&V team, and the completion of analysis of that product. During this time, the development process continues. Hence, if IV&V analysis takes too long, the results might be available too late to be useful. In general, the earlier an error is reported, the cheaper it is to correct.

lack of access – Contact between the development team and the IV&V team is difficult to manage. The IV&V team needs to maintain independence, whilst ensuring they obtain enough information from the developers to do their job. From the developers' point of view, interaction with the IV&V team represents a cost overhead, which can interfere with project deadlines. Inevitably, the

IV&V contractor has less access to the development team than is ideal.

evolving products – Documentation from the development team is usually made available to the IV&V contractor in draft form, to facilitate early analysis. The drawback is that documents may be revised while the IV&V team is analyzing them, making the results of the analysis irrelevant before it is finished.

reporting the right problems – The IV&V contractor has, by necessity, considerable discretion over the kinds of analysis to perform on different products. It also has discretion over which problems to report. It is vital to the effective use of IV&V that the IV&V contractor prioritizes the problems it identifies. If too many trivial problems are reported, this may swamp the communication channels with the developer and the customer.

lack of voice – The IV&V contractor may have difficulty in getting its message across, especially when the development contractor disputes IV&V's assessment. Often, problems found by IV&V have cost and schedule implications, and in such circumstances the customer may be more willing to listen to assurances from the developer. The effectiveness of IV&V then depends on having a high-placed advocate within the customer organization.

Despite these problems, IV&V has been shown to be a cost-effective means of improving the quality of the software product, and providing extra assurance for high-cost, safety-critical projects [12]. In addition to providing analysis of project artifacts (e.g. requirements, code, test plans), the presence of IV&V in the lifecycle also has a positive effect on the quality of the software. Our work suggests that the *interaction* between the IV&V and development teams drives improvements in both products and processes. This effect, however, is difficult to capture and quantify.

3 Methods and Tools in IV&V

An important aspect of IV&V is the choice of the right methods and tools. Ideally, an IV&V contractor will have access to all the tools used by the development team, including the ability to share all project databases. However, the IV&V team also needs to supplement these with additional methods and tools, to address any gaps or weaknesses in the coverage of the developer's tools. These additional tools need to complement the developer's tools, so that interoperability does not become a problem. The use of these additional tools is an important factor in ensuring that IV&V is truly independent.

It is often the case that the use of a particular method or tool by the IV&V team leads to the adoption of that method or tool by the developers. In part this is due to the 'watchdog effect': if the developers know that their product will be analyzed in a particular way, it is in their interest to perform the analysis themselves before releasing it. If this seems to be a rather negative reason to adopt a technique, there is also a positive aspect. Because the IV&V team is out of the critical path for the software development effort, they have more scope for experimentation with new techniques than the developers [1]. Hence, in some ways the IV&V team can play a role as a proving ground for new techniques, and can come to be an agent of process improvement. For these reasons, we believe that IV&V offers a practical route through which formal methods may be introduced into projects that would otherwise not be able to adopt them.

There are still problems to be overcome whenever the IV&V team adopts a tool that is not used by the developers. Compatibility with the developers' tools is important. For example, if the IV&V team uses a formal specification tool, the informal specification delivered by the developers will need to be translated into the formal specification language not just once, but each time the developers produce a new draft. Any problems identified by using the tool must be traced back to the informal specification, before they can be reported. There must be a reasonable assurance that the formal specification remains faithful to the original, otherwise any analysis performed on it is worthless. Hence, keeping track of the relationship between the formal and informal specifications is vital.

4 Experiments with formal methods

Having described the role that an IV&V contractor plays in the software process, and outlined the issues involved in the selection of tools and techniques for IV&V, we now present our work on the use of formal methods in the IV&V of requirements specifications. We performed two experiments. The first was a formalisation of individual requirements statements into a tabular form, to improve clarity. The second was the development of a formal model of these requirements, which was then tested for consistency.

Currently, the development contractors on the Space Station project use natural language specifications extensively. We are working with the IV&V team to explore how formal methods can enhance the kinds of analysis they perform on the developer's informal specifications. Here, we will report our work with the Fault Detection, Isolation and Recovery requirements for the main command and control bus. An example requirement is given in figure 1.

Our initial interest in formal methods was twofold. First, it was clear that the informal specifications were hard to understand, and would benefit from a clearer representation. We needed a notation that was both precise and easy to read. Leveson's AND/OR tables [8] provided us with a solution. During the development of the RSML specifications for TCAS II, Leveson adopted these AND/OR tables in preference to predicate calculus, as they were readable by a wide range of people. This tabular representation was well suited to the Space Station FDIR requirements (see table 1), as it mapped directly onto the individual requirements statements.

Second, we needed a way to verify that the specified functionality was internally consistent. For the FDIR

(2.16.3.f) While acting as the bus controller, the C&C MDM CSCI shall set the e,c,w, indicator identified in Table 3.2.16-II for the corresponding RT to "failed" and set the failure status to "failed" for all RT's on the bus upon detection of transaction errors of selected messages to RTs whose 1553 FDIR is not inhibited in two consecutive processing frames within 100 millisec of detection of the second transaction error if; a backup BC is available, the BC has been switched in the last 20 sec, the SPD card reset capability is inhibited, or the SPD card has been reset in the last 10 major (10-second) frames, and either:

1. the transaction errors are from multiple RT's, the current channel has been reset within the last major frame, or
2. the transaction errors are from multiple RT's, the bus channel's reset capability is inhibited, and the current channel has not been reset within the last major frame.

Figure 1: An example of a level 3 requirement for FDIR of the Command and Control bus for Space Station. This requirement specifies the circumstances under which all remote terminals (RTs) on the bus should be switched to their backups.

requirements, this meant checking that the conditions specified for each recovery action were mutually exclusive, and that the requirements covered all possible conditions. Hand checking these properties would have been hard, so we sought a tool to help. We examined several tools, before selecting SCR* [10]. SCR offered two important advantages. First, the notation was primarily tabular, which appeared to be an important aid to readability. Second, the tool had automated checking for properties such as coverage and disjointess of a state based model [9]. In addition, this tool did not require us to build a complete formal model of the Bus FDIR functionality in order to check these properties.

4.1 Experiment 1: Translation

Our first experiment concerned the translation of requirements like that shown in Figure 1 into a formal notation. Leveson's AND/OR tables allowed us to represent arbitrary combinations of conjunctions and disjunctions without ambiguity, and in a form that was clearly readable. Table 1 shows the tabular form of the requirement in Figure 1.

For the IV&V team, this was a significant improvement in readability. More importantly, the process of producing the tables ensured that the analysts fully understood the requirement. This benefit is very important for IV&V. In many cases, just reading a specification is insufficient to really appreciate the detail. Short of repeating the development process from scratch, it can be hard for the IV&V analyst to understand a specification in the same way that its authors understand it. Translating it into a table, however, proved to be a valuable clarification process.

There was, unfortunately, a problem. Translation of a single requirement, like the one above, was not a straightforward task. Translation of this requirement took several attempts until we were happy with the table, and even then we were not convinced that it was right.

We conducted an experiment to investigate the problem. We gave the English language version to four different people, all of whom had some experience of representing requirements using tables, and asked them to produce the tabular form. Two of these people were domain experts, and two were not. We were interested in exploring the scope for misinterpretation of the requirements from the point of view of both domain experts who write such requirements, and other stakeholders, such as the programmer who would have to implement them.

We received four different answers. These differed in both the number of conditions identified (i.e. number of rows in the table) and the number of combinations under which the function would be activated (i.e. columns in the table). The version shown in Table 1 is a synthesis of the four answers, representing what we currently believe is the intended interpretation.

The differences in the responses show that the original requirement is riddled with ambiguities. For example, the mixture of 'ands' and 'ors' in the requirement is a problem because, unlike programming languages, English does not have any standard precedence rules. It is not clear how to scope the various subclauses, either. For example, the timing condition 'within 100 millisec...' could refer to the inhibition of the FDIR, or to one or both of the required setting operations. With a little domain knowledge, it is possible to eliminate some interpretations, but this is by no means a trivial task, and there is no guarantee that everyone who needs to read this requirement will get it right.

The experiment demonstrated three important results. Firstly, the tabular forms were very helpful in resolving misunderstandings. For example, it would be difficult to discover that our four subjects had different interpretations of the original requirement without asking them to re-write it. By re-writing it in tabular form, we could identify exactly where the disagreements lay, and then take each discrepancy in turn and discuss what we thought the most likely interpretation was. From this, we were able to synthesize a 'best' interpretation. Obtaining individual translations and comparing them was more effective in identifying differences in our understandings than our initial

		OR			
	C&C MDM acting as the bus controller	T	T	T	T
	Detection of transaction errors in two consecutive processing frames	T	T	T	T
	errors are on selected messages	T	T	T	T
	the RT's 1553 FDIR is not inhibited	T	T	T	T
	A backup BC is available	T	T	T	T
A	The BC has been switched in the last 20 seconds	T	T	T	T
N	The SPD card reset capability is inhibited	T	T	.	.
D	The SPD card has been reset in the last 10 major (10 second) frames	.	.	T	T
	The transaction errors are from multiple RTs	T	T	T	T
	The current channel has been reset within the last major frame	T	F	T	F
	The bus channel's reset capability is inhibited	.	T	.	T

Table 1: A Leveson-style table for requirement 2.16.3.f. This table summarizes the conditional part of the requirement in Figure 1, showing four combinations of conditions (the four columns) under which the specified action should be carried out).

attempts to work together to produce a single translation. This confirms a hypothesis described in [5], that negotiating requirements conflicts is more effective if we start with a precise description of each person's individual viewpoint. Note that our final version was different from all four of the individual versions, implying that if the final version is correct, all four individual attempts were wrong!

This leads to the second result, which is that translation of informal requirements into a formal notation is error prone. All four of our subjects had some experience of using such tables, so the problem lies not in the correct use of the notation, but in the interpretation of the informal statement of requirements. The requirement we used in the experiment is perhaps an extreme example, given its rather convoluted English. However, there is enough scope for misinterpretation in the process of formalizing the requirements to cause us to worry about the fidelity of our formal models.

The third result is that the whole process was remarkably good at identifying ambiguities in the original specification. By producing different interpretations and comparing them, we were able to identify a systematic pattern of ambiguities in the way the English language requirements were written. Hence, even if the IV&V team fail to persuade the development team to adopt a tabular notation, they can at least help them to correct the ambiguities in the English.

In fact, the development contractors have used the tabular notation occasionally, in the most recent versions of the specifications. Initially, they resisted the IV&V team's requests to adopt a tabular notation, largely because of schedule constraints. They have now begun to use the notation for revisions of the specifications, especially in areas where reviewers had had problems with readability. We regard this as a small but important process improvement, inspired by the IV&V team.

4.2 Experiment 2: Analysis of Partial Specifications

Our second goal was to check some of the properties of the FDIR specification that could not be checked by hand. One of the important validity checks for these requirements is that an action is specified for each possible combination of failure conditions. Another check is that no combination of conditions has conflicting actions specified for it. We refer to these as coverage and disjointness checks, respectively [14].

In practice, there were two approaches that IV&V could take to verify such properties. They could obtain the development team's failure model, validate this model, and then verify the requirements against the model. Or they could generate their own behavioral model of the requirements as described, verify that it is internally consistent, and then validate this against their understanding of the system. The latter approach was chosen, partly because the IV&V team has had difficulty obtaining the original models on which the specification is based, and partly because the latter approach was more likely to overcome analysis bias.

We chose SCR as an appropriate model to perform these analyses for a number of reasons. First, the tabular notation used in SCR maps onto the AND/OR tables we had already generated in a fairly systematic way. Each AND/OR table represents a single row in a mode transition table in SCR. Second, there was a tool (SCR*) available for checking SCR specifications which included both coverage and disjointness tests, and which had a simulator built in for animating the complete state-based model. A model checker was being added. Furthermore, the consistency checker in the SCR* tool provides counter-examples whenever an inconsistency is found. Our early experiments with a theorem prover (PVS [14]) were abandoned because when a proof failed, it took too long to discover the problem. The provision of counter-examples is important in tracing problems back to the informal specification, and in convincing the development team that there really is a problem.

The first step was to produce an SCR model of the specified FDIR behavior. At this stage we had six AND/OR tables, similar to the one shown in Table 1, representing the six paragraphs, a to f, of section 2.16.3 of the requirements. Each paragraph isolates one failure mode, and specifies an appropriate action. We merged these into a single table, modeling each failure mode as a separate SCR mode (Table 2).

Merging the AND/OR tables to produce Table 2 was not straightforward. Although there were a number of conditions common to several of the tables, the wording varied, and it was not always obvious whether similar sounding phrases actually referred to the same condition, due to inconsistencies in the use of terminology. For example the condition "the bus has been switched in the major (10-second) frame" appeared in one paragraph, and "the bus has been switched in the last major frame" appeared in another. We initially assumed these to be identical. However, this led to an inconsistency in the table. In fact the former refers to the *current* frame, while the latter refers to the *previous* frame. There were numerous places where we had to make assumptions to proceed, and we carefully recorded these as annotations to the original text, to be checked with the developers.

The modes we have identified are not present explicitly in the informal specification. Our modes correspond intuitively to failure modes, but might not be a particularly good choice for simulation or model checking purposes, because they really express output events rather than states. However, they suit our purpose, as the table in this form can be checked directly for coverage and disjointness without completing the model. In fact, the complete model would be complicated: a clock would be needed to implement the bus processing frames, together with several timers to keep track of historical state. Even then, SCR cannot (currently) represent timing conditions on the required functions.

Having created the table, we then checked it for coverage and disjointness. Not surprisingly, the table is not disjoint: in fact there is an overlap between every possible pair of rows. Analysis of the counterexamples provided by the SCR* tool indicates a systematic under-specification of the conditions. The original model of the FDIR system was a procedural model with an explicit order on the checks that need to be performed. The specification does not have this explicit ordering, and the described conditions do not adequately express this ordering. However, this result was not a surprise: the IV&V team had already submitted a report suggesting that the ordering be made explicit in the specification.

While we were producing this analysis, a new draft of the specification was released. The section specifying Bus FDIR requirements had been re-written, partly due to issues raised by the IV&V team, both before and after our first experiment. The new version is much clearer (but does not use our tables). It is also much simpler: several failure modes and at least half the conditions expressed in Table 2 have been removed, and the disjointness problem described above has been corrected.

Hence our formal analysis was redundant before it was complete. In practice, it would have been possible to perform the analysis much earlier: we delayed the work until a full release of the SCR* tool was available. However, we can now apply the same technique to other parts of the specifications, and expect that in some cases it will identify new problems, while in others it will supply concrete evidence of known problems. Once the requirements are stable, we plan to build a complete model of the FDIR subsystem, and use a model checker to study its behavior under repeated and intermittent fault conditions.

5 Discussion

We have described our on-going work with formal methods as a tool for an Independent V&V team to perform analysis of software requirements. Our initial results are encouraging: the translation process was extremely valuable in identifying ambiguities and improving our understanding of the specification. In this process, a number of errors were found. Analysis of a partial formal specification demonstrated an important error in the specification, and appears to be a powerful means of gaining maximal results from minimal effort. We constructed just enough of a model to test the properties we were interested in, without any further commitment to the method.

However, our experiments have revealed two related problems: it is hard to guarantee fidelity between informal and formal specifications, and it is hard to manage consistency between partial specifications expressed in different notations.

Although the major finding of our formal analysis is valid, we are not confident that the partial model is faithful to the version of the developer's specification on which it is based. This fidelity issue is more of a problem in IV&V than in development. A formal model developed by the IV&V team cannot replace the informal specification. The IV&V team must therefore either persuade the developers to adopt formal notations themselves, or take care to maintain fidelity between the developers' informal specifications and their own formal models. With the current state of practice, wholesale adoption of formal methods by the developers on an existing project is unlikely [4].

The fidelity problem is important to IV&V because the formal models developed by IV&V are produced for the purposes of checking the developer's specifications. The models are only useful for this purpose if they are accurate representations of the developer's specifications. Also, when analysis of the formal models reveals problems in the specifications, these problems must be traced back to the informal specification before they can be reported.

Although the fidelity problem seriously affects the utility of any formal analysis performed by the IV&V team, we should point out that it does not affect all the benefits of formal specification. The process of translating pieces of the informal specification into a formal notation has benefit not just for the analysis

Current Mode	Conditions											Next Mode
	errors in two cons. frames	bus swch'd last frame	bus switch inhibit	bus swch'd this frame	backup BC avail.	BC swch'd in last 20 sec	card reset inhibit	card reset last 10 frames	errors from mult. RTs	channel reset last frame	channel reset inhibit	
Normal	@T	-	-	F	-	-	-	-	-	-	-	switch buses
	@T	-	T	F	-	-	-	-	-	-	F	reset the
	@T	T	-	F	-	-	-	-	-	-	F	channel
	@T	-	-	-	-	-	F	F	T	T	-	reset the
	@T	-	-	-	-	-	F	F	T	F	T	card
	@T	T	-	-	-	-	-	-	F	T	-	switch RT
	@T	F	T	-	-	-	-	-	F	T	-	to backup
	@T	T	-	-	-	-	-	-	F	F	T	
	@T	F	T	-	-	-	-	-	F	F	T	
	@T	-	-	-	T	F	T	-	T	T	-	switch BC
	@T	-	-	-	T	F	T	-	T	F	T	to backup
	@T	-	-	-	T	F	-	T	T	T	-	
	@T	-	-	-	T	F	-	T	T	F	T	
	@T	-	-	-	T	T	T	-	T	T	-	switch
	@T	-	-	-	T	T	T	-	T	F	T	all RTs
	@T	-	-	-	T	T	-	T	T	T	-	
	@T	-	-	-	T	T	-	T	T	F	T	

Table 2: An SCR Mode transition table. Each of the central columns represents a condition, showing whether it should be true or false; '-' means "don't care"; '@T' indicates a trigger condition for the mode transition. The four columns of table 1 correspond to the last four rows of this table. The semantics of SCR require this table to represent a function, so that the disjunction of all the rows covers all possible conditions (coverage), and the conjunction of any two rows is false (disjointness).

that it leads to, but also for the removal of ambiguities and for improved understanding. For this benefit, it is the *process* of formalization, rather than the end product that is important.

The fidelity problem is really a special case of a more general problem: management of consistency between partial specifications expressed in different notations. For instance, the AND/OR tables have a clear relationship with the SCR mode tables, but if we make a correction to one of the AND/OR tables, it is fairly tedious to identify the corresponding correction in the SCR tables. Similarly, each time the developers issue a new informal specification, we need to update our tabular representations. Although it may seem that the use of both AND/OR tables and SCR models together would compound this problem, the opposite is true. The AND/OR tables mapped clearly onto the textual requirements, while the relationship between the AND/OR tables and the SCR model was relatively straight forward. Therefore, the use of AND/OR tables as an intermediate representation reduced the traceability gap, and made it easier to keep the formal model up to date. There remains, however, a significant bookkeeping problem.

There is a growing body of work on handling inconsistency in specifications. Our previous work demonstrated how to delay the resolution of inconsistency, and provided a generic framework for expressing consistency relationships [6]. Other work has taken consistency checking further, making use of semantic models underlying a method to determine what consistency rules are needed and how to operationalize them. For example, Heitmeyer's work with consistency checking in SCR [9] uses the semantics of SCR to define a series of consistency rules ranging from simple syntactic checks (e.g. that all names are unique) to sophisticated properties of tables (e.g. coverage and disjointness). Similarly, Leveson's work on consistency checking in RSML [8] uses the semantics of the statechart formalism to determine a set of consistency rules that can be tested, tractably, using a high level abstract model. In both these approaches, the completeness of the formal specifications is important, and consistency checking is seen as part of the process of obtaining a complete, consistent specification.

Unfortunately, these approaches do not help with consistency checking between partial specifications expressed in different notations. Because the IV&V process is concurrent with and complementary to the development process, there is an unusually large amount of flexibility in how a formal method can be used. There is no need to make a commitment to any one formal notation, just as there is no need to develop complete specifications. In fact, the aim of the IV&V agent is not to perform complete analyses, but to do just enough analysis to check specific aspects of the software. Development of complete formal models is therefore unnecessary and may be counter-productive. For example, in our second experiment, the limited analysis we performed on a partial model was sufficient to reveal a major problem; the existence of this problem meant that any further effort to complete the model would have been wasted.

While the use of partial specifications offers greater flexibility in the use of methods and tools, it also means that we do not have a well-defined method from which to generate a set of consistency relationships. There are implicit consistency relationships between the assorted partial specifications drawn from different methods, but there is no overall 'method' to to tell us what these relationships are. Actually, there is a method: the problem is that it is implicit, and to some extent is generated on the fly. For example, there is a method for generating SCR mode tables from the AND/OR tables, but the method was not defined before we did it. With some effort, we could formalize this method, and define semantic relationships between the two types of table. However, this effort will only be worthwhile if we intend to re-use the method extensively. In the meantime, we would like to have tools to help us keep track of consistency relationships in our opportunistic use of partial specifications.

In our previous work defining consistency relationships between viewpoints, we assumed that the majority of such rules are defined by the method [6]. The viewpoints framework explicitly supports the process of method definition, in which, among other things, the inter-viewpoint relationships are defined. Hence the general problem of defining arbitrary relationships between any two notations is avoided. However, we also recognized that some consistency relationships could not be defined in this way, and gave the example of a user-defined synonym relationship between two different labels. We also outlined an approach to discovering such relationships through low level process monitoring. We now regard this type of consistency relationship as vital to any approach involving partial specifications.

Without a method to define *a priori* consistency relationships, we are forced to discover the relationships as the work proceeds. In fact this is not as hard as it sounds. By recording low level actions on the partial specifications, we begin to build up a fine-grained process model, which can provide information about consistency relationships. For example, by observing cut and paste operations during the creation of our AND/OR tables and our SCR mode tables, it is possible to determine the relationship between rows in the AND/OR tables and rows in the mode table. In the weakest case, this will provide us with a simple traceability link. In fact, we believe we can do better than this. There is enough information in the edit actions not just to identify traceability links, but to define the relationship expressed by the link. For example, it should be possible to determine enough information to define a consistency rule that can automatically check that each column of the AND/OR table is consistent with its corresponding row in the mode table. We plan to explore this avenue further, by capturing and analyzing this kind of process information.

6 Conclusions

This paper has described our initial work in the use of formal methods in an IV&V project. We have discussed how the demands placed on methods and tools in IV&V are different from their use in a development context. We have also discussed how IV&V can act as a process improvement agent, and hence can be a fruitful way of introducing formal methods into large projects.

As with all potential uses of a new method, any extra effort needed to use the method must be more than offset by the benefits it brings. Use of a method in IV&V is no different. We can divide the benefits of using a formal method such as SCR into two areas:

1. The process of translating portions of a specification into a tabular notation helps to detect ambiguities and increase readability, even if the translation is only partial. The process can also be used to catch misunderstandings, thus increasing the confidence that the IV&V team is interpreting the specification correctly. The process of having several analysts produce their own tabular translations was particularly useful in this respect. Differences in the tables they produced allowed us to pinpoint exactly what the disagreement was about.

2. The resulting tables can be analyzed for attributes such as coverage and disjointness. This is a substantial contribution to the IV&V team's efforts to check the technical integrity of the specifications. Such attributes are particularly hard to analyze from the informal specifications. Most importantly, this analysis can be conducted without the need to build complete models.

The problems we encountered in applying formal methods were as follows:

1. The process of translating into a formal notation is error-prone. Only by duplicating the translation effort were we able to discover just how much scope there is for misinterpretation. Luckily, the resulting tables are very readable. Therefore it is much easier to compare different tables than it is to compare different versions of the informal specification.

2. For IV&V, fidelity and traceability between the informal and formal specifications is difficult to guarantee. The value of any analysis carried out by IV&V on the formal model is entirely dependent on how faithful the formal model is to the developer's informal specification. The IV&V's formal model can not be used in place of the informal specifications produced by the developers.

3. Opportunistic use of partial specifications means that there is not a well-defined method from which to derive consistency rules. Maintenance of consistency in our partial specifications became a real problem.

The problems of consistency checking in partial specifications written in different notations is important enough to warrant more attention. We plan to study the problem in more detail by developing a set of tools based on the ViewPoint framework [7], which will allow us to model relationships between partial specifications written by different people. We are also exploring how this problem relates to that of linking test case scenarios to requirements [2]. Finally, we are continuing the experiments described in this paper by examining how model checking can be used to validate the specifications.

Acknowledgments

Our thanks are due to Chuck Neppach and Dan McCaugherty for many interesting discussions of the work presented here, and to Frank Schneider, Edward Addy, John Hinkle, George Sabolish, Todd Montgomery and Butch Neal for detailed comments on earlier drafts of this paper. This work is supported by NASA Cooperative Research Agreement NCCW-0040.

References

[1] V. Basili. The experience factory and its relationship to other improvement paradigms. In *Proceedings of the 4th European Software Engineering Conference, Garmish-Partenkirchen, Germany*, September 1993.

[2] J. Callahan and T. Montgomery. An approach to verification and validation of a reliable multicast protocol. In *Proceedings of the ACM International Symposium on Software Testing and Analysis (ISSTA)*, January 1996.

[3] D. Craigen, S. L. Gerhart, and T. Ralston. Formal methods reality check: Industrial usage. *IEEE Transactions on Software Engineering*, 21(2):90–98, 1995.

[4] D. H. Craigen, S. L. Gerhart, and T. J. Ralston. An international survey of industrial applications of formal methods, vol 1: Purpose, approach, analysis and conclusions. Technical Report NRL/FR/5546–93-9581, Naval Research Laboratory, 1993.

[5] S. M. Easterbrook. Handling conflict between domain descriptions with computer supported negotiation. *Knowledge Acquisition: An International Journal*, 3(4):255–289, 1991.

[6] S. M. Easterbrook and B. A. Nuseibeh. Managing inconsistencies in evolving specifications. In *Second IEEE International Symposium on Requirements Engineering*, pages 48–55, March 1995.

[7] S. M. Easterbrook and B. A. Nuseibeh. Using viewpoints for inconsistency management. *BCS/IEE Software Engineering Journal*, 11(1), January 1996.

[8] M. Heimdahl and N. Leveson. Completeness and consistency analysis of state-based requirements. In *Proceedings of the 17th International Conference on Software Engineering*, pages 3–14, April 1995.

[9] C. Heitemeyer, B. Labaw, and D. Kiskis. Consistency checking of scr-style requirements specifications. In *Second IEEE International Symposium on Requirements Engineering*, pages 56–63, March 1995.

[10] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. Scr*: A toolset for specifying and analyzing requirements. In *Tenth Annual Conference on Computer Assurance (COMPASS '95)*, pages 109–122, June 1995.

[11] D. Jackson and C. A. Damon. Elements of style: Analysing a software design feature with a counterexample detector. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 239–249, January 1996.

[12] Jet Propulsion Lab. Cost-effectiveness of software independent verification and validation. Technical Report NASA RTOP 323-51-72, NASA JPL, October 1985.

[13] R. O. Lewis. *Independent Verification and Validation: A Lifecycle Engineering Process for Quality Software.* J. Wiley & Sons, 1992.

[14] S. Owre, J. Rushby, and N. Shankar. Analysing tabular and state-transition specifications in pvs. Technical Report CSL-95-12, Computer Science Laboratory, SRI International, 1995.

[15] H. Saiedain, J. P. Bowen, R. W. Butler, D. L. Dill, R. L. Glass, A. Hall, M. G. Hinchey, C. M. Holloway, D. Jackson, C. B. Jones, M. J. Lutz, D. L. Parnas, J. Rushby, J. Wing, and P. Zave. An invitation to formal methods. *IEEE Computer*, 29(4):16–30, 1996.