V&V through Inconsistency Tracking and Analysis[1]

Steve Easterbrook, John Callahan, and Virginie Wiels
NASA/WVU Software Research Lab
100 University Drive, Fairmont, West Virginia 26554
Contact: steve@research.ivv.nasa.gov

Abstract

In this paper we describe a research agenda for a study into the use of inconsistency analysis as a tool for software V&V, and in particular, the use of category theory as a basis for modeling consistency relationships between the various artifacts of software development, including specifications, design, test cases, etc. Our expectation is that inconsistency analysis is fundamental to much of the work of V&V, and that a systematic approach will have impacts across a wide range of V&V tasks. Two goals are described: an account of the contribution of inconsistency analysis to various V&V analyses, and the development of a formal framework for inconsistency analysis. Our approach to the development of the latter is through the refinement of the viewpoint framework using the language of category theory. We discuss the relationship between specification morphisms, as conventionally conceived in work on composition of specifications, and inter-viewpoint consistency relationships. We conclude that inconsistency analysis has the potential to address the 'air gaps' between methods used during the various phases of the software lifecycle.

1 Introduction

A key problem in the development of large scale software systems is to detect and correct inconsistencies that arise between the various artifacts (i.e. specifications, designs, code, etc.) of a software project, especially where these artifacts are developed and refined by geographically distributed teams. Existing software development environments tend to provide detailed support for individual phases of the software lifecycle, but leave 'air-gaps' between artifacts produced at different phases (e.g., between requirements specifications and design models). Such gaps hinder automated analysis and management of consistency within a project. This lack of consistency management is one of the leading causes of errors in large, complex software development efforts. Much of the effort of verification and validation (V&V) is concerned with detecting and tracking inconsistencies between project artifacts.

Recent studies of software development and V&V processes have indicated that inconsistency is commonplace throughout the software lifecycle [1]. Inconsistencies occur because different teams proceed at different paces, because requirements evolve during the project, because information is not always available when needed, and because methods and process models may not fit the local contingencies on a project. Although inconsistencies represent potential sources of error during development, it does not follow that consistency should be enforced. For practical purposes, in a large, distributed development effort is necessary to tolerate inconsistencies for periods of time during development. However, it is important to identify and track inconsistencies, as it is the undetected inconsistencies that are likely to cause errors, or result in re-work [2].

Two key research challenges for successful management of inconsistency are (1) to develop analysis techniques that reveal and characterise inconsistencies, and (2) to provide a modeling scheme to represent and track consistency relationships between the various artifacts of software development. A number of approaches have been proposed to address these problems in specific cases, each grounded in a specific software development method [3-5]. However, these approaches do not address the 'air gaps' *between methods*. For example, a requirements analysis method and a design method might each provide sophisticated consistency checking, but in most cases, the problem of checking consistency between the requirements and design models is largely a manual one. In large projects, it is normal to apply multiple methods in each phase, especially to achieve the breadth of analysis needed for V&V purposes.

[1] *The research described in this paper was carried out by West Virginia University under NASA cooperative agreement #NCC 2-979 and Grant #NAG 2-1134.*

In this paper we describe a research agenda for a study into the use of inconsistency analysis as a tool for software V&V, and in particular, the use of category theory as a basis for modeling consistency relationships between software specification and design artifacts. Our expectation is that inconsistency analysis underpins much of the work of V&V, and that a systematic approach will have impacts across a wide range of V&V tasks. Our choice of category theory as a basis for this work is based on the success of category theory as a framework for composing specifications, and especially for reasoning about the correctness of specification composition and refinement, through the use of specification morphisms [6].

2 Technical Objectives

The objectives of our research are:

- To investigate the extent to which inconsistency analysis supports and complements the various types of analysis performed in software verification and validation. We will concentrate on the consistency relationships (a) between different parts of a requirements specification; (b) between requirements and design models; (c) between design models and test cases.
- To develop a generalized schema for expressing arbitrary consistency relationships between software development artifacts, including requirements, designs and test cases. Existing schema for expressing consistency relationships rely on the well-formedness rules implied by the semantic models of individual development methods. Such approaches cannot therefore express relationships between artifacts developed using different methods, such as artifacts pertaining to different phases in the development lifecycle. A generalized schema is needed to express such relationships.
- To explore the applicability of our category-theoretic approach to allow composition of partial specifications or specifications only partially related to each other. Existing categorical approaches support specification of components and relationships between these components, and construction of the resulting system specification using composition operations (horizontal structuring). They also support the refinement of each component (vertical aspects) in a way that is compatible with horizontal composition (i.e. the composition of refined specifications is a refinement of the specification built by composition of the abstract components). We aim to extend this framework to be able to define partial specifications, partial relationships between specifications and to compose such specifications. On the vertical level, we are interested in formalizing consistency relationships between existing specifications of a component at different levels of abstraction. We will explore the use of this approach in a V&V context, to model *discovered* relationships between existing specification components, and to keep track of and reason about inconsistencies.

3 Approach

The project will concentrate on case studies within NASA IV&V efforts including projects such as the International Space Station (ISS), New Millennium Program (NMP), the Shuttle avionics software, and the Shuttle Checkout and Launch Control System (CLCS). Our analysis will focus on aspects of traceability between specifications, designs, code, and test plans for these projects. In each case, these artifacts are available in a mixture of formal and informal notations. Initially, we will concentrate on those parts of the requirements and designs that are already formally defined, or for which a formal translation can be derived with relative ease.

Our first case study will examine the change requests for the space shuttle avionics software. These documents offer a rich source of data for modeling, as they are extremely well documented, and specified at a relatively low level of detail (detailed processing requirements are often specified in HAL, which is really a simple imperative programming language). Verification and validation of these change requests involves a great deal of cross-checking between different parts of the affected specifications. Furthermore, there is currently no way of validating that the proposed changes are consistent with high level requirements, other than by extensive manual inspection, nor that the test cases derived from the change request adequately cover all the affected behaviors of the system. The current manual inspection process consists of a combination of syntactic consistency checking, coupled with validation through the application of domain expertise. We plan construct a set of formal models from the requirements specifications, selected from those affected by particular change requests. This set of formal models will then provide us with a concrete example against which we can assess the proposed approaches for consistency checking.

Our approach is based on the application of a number of related technologies, including techniques for composing specifications and specification refinements (E.g. SpecWare), the use of autogenerators for deriving implementations directly from design models (E.g. MatrixX), and the use of model checkers (e.g. SPIN) and theorem provers (e.g., PVS) for verifying the behavioural proper-

ties of specifications and design models. Each of these technologies provides a means to analyse some aspects of the consistency relationships we are targeting.

Specifically, we plan to investigate the following types of consistency relationship:

- Requirements ↔ Requirements: Functional requirements for embedded systems are typically represented as a set of mappings between events in the environment and control actions. Non-functional requirements (e.g. safety, performance, etc) are then represented as global constraints on the behaviour of the system. Requirements for large systems are specified by composing the specifications for different functional areas. The challenge for consistency management is to use the composition relationships between partial specifications to analyse consistency as the specifications evolve. The proposed research will apply SpecWare to specifications drawn from the case studies, and explore the use of specifications morphisms to track consistency relationships between specifications that are subjected to non-truth preserving evolution.
- Requirements ↔ Design: Requirements define the functional behaviour of a system, while design models define the internal structure (i.e. architecture) of a system. Consistency between design models and requirements is concerned with whether the design model exhibits the required behaviour, and satisfies all the constraints. Our experiments with model checking have demonstrated that if the design models (for real time control systems) are expressed as state machines, and requirements as temporal properties of those machines, consistency checking can be achieved by exploring the traces that satisfy both machines. To demonstrate that any individual requirement is satisfied, we describe its negation as a Büchi automaton, and check that there is no trace that is accepted by both machines [7]. The proposed research will compare this work on model checking with the support for specification elaborations provided by the specware approach, to seek ways of integrating the two approaches.
- Design ↔ Code: During development, the code must not only implement behaviors as specified by the design model, but models themselves may need to change based on discovered limitations of an implementation environment [8]. Managing the consistency between code and design models is important as the software evolves because any divergence may lead to serious flaws later in the lifecycle. Our work with formal testing has led us to an approach based on the generation of representative test cases from a formal model, so that a combination of model checking and testing instrumented code can be used to monitor consistency. The proposed research will examine how to automate this approach, and explore how this approach can be integrated with the work on requirements consistency, as a step towards full consistency checking from requirements through to code.

4 Previous Work

4.1 Inconsistency Management

Our previous work in the requirements area has been based on the viewpoints framework [9]. This framework supports distributed software development through a collection of loosely coupled heterogeneous objects ('viewpoints'), each of which provides a self-contained specification development tool. Global consistency is achieved through a series of pairwise consistency checks between viewpoints. The framework is method-independent, in that any method can be defined within the framework by designing a set of viewpoint templates and fine-grained process models. This allows us to explore the issues of consistency management both within and between methods. Our recent work has examined the application of this framework in a V&V context, in which partial formal specifications are represented as viewpoints and verified for consistency with other viewpoints. An advantage of this approach is that it permits the V&V effort to concentrate on just those portions of a specification that are critical, and test only the properties of interest, without developing complete formal models.

There is a growing body of work on managing consistency in specifications. Easterbrook & Nuseibeh [2] demonstrated how to delay the resolution of inconsistency, and provide a generic framework for expressing consistency relationships. Other work has developed detailed consistency checking schemes by making use of semantic models underlying a particular method to determine what consistency rules are needed and how to operationalize them. For example, Heitmeyer's work with consistency checking in SCR [4] 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 semantic properties of tables (e.g. coverage and disjointness). Similarly, Leveson's work on consistency checking in RSML [3] 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 using different modeling schemes. Such approaches do not help to bridge the 'air gaps' in software development, nor do they help with consistency checking between the various models developed for V&V purposes. There are implicit consistency relationships between the different models generated during a software lifecycle, but there is no overall 'method' to tell us what these relationships are. Our work with the viewpoints framework addressed this problem by recognizing that some consistency relationships are defined by the method, while others arise from the application domain, and some arise from local contingencies during development. It is unreasonable to expect that any method or integration of methods can be defined in sufficient detail such that all potential consistency relationships can be determined *a priori*. Instead, support is needed for discovered relationships, such that these can be recorded and monitored as the specifications evolve. Recent work on restructuring specifications using viewpoints indicates that such an approach is viable [10].

4.2 Specification Composition

Category theory has been used for a number of years as a framework for composing formal specifications, based on early work by Goguen (E.g. see [11]). Category theory provides an abstract framework for reasoning about specification composition through the use of specification morphisms. For example, if specifications are expressed as theories in some suitable logic, a morphism maps the terms of one specification into the terms of the other. Composition via specification morphisms can then be checked for correctness by proving that the axioms of one specification hold as theorems of second specification. This approach has been applied in a number of ways, including reasoning about the correctness of specification refinements, and for modularizing the specifications for complex systems. In the latter case, if the structure of the implementation reflects the structure of the specification, then verification is greatly facilitated, whether it is achieved through correctness proofs, or through specification-based testing.

Much of the work on composition of specification via category theory has concentrated on the composition of algebraic specifications. Fiadeiro and Maibaum [12] developed a framework based on temporal logic, in which each component of a system is described by a theory in temporal logic, and theories are interconnected using specification morphisms. Michel and Wiels [13] have extended this framework to allow the interface for a module to be specified separately from the body, with a set of morphisms relating the interface specification to the body specification, and a further set of morphisms defining composition relations between modules. This work also examined how validation properties can be decomposed across the specification components, so as to reduce the effort involved in proving them.

A number of tools are now available that support a category theoretic approach to construction of specifications (E.g. Specware [6] Moka [13]). These tools typically have a kernel that implements a basic categorical framework, on top of which is defined the specification language, such that specifications form the objects of a category whose morphisms represent relationships between specifications. The tool can then be coupled to a model checker or a theorem prover, to support the construction of the necessary proofs when morphisms are defined.

4.3 Specification Based Testing

In the area of managing consistency between the design artifacts and code in a project, we have used a concept known as *formal testing* [14] to maintain fidelity between finite state models of the behavior of complex communication protocols and the code that purports to implement the protocol specification. Through the use of model checkers, we are able to partition the state space of the protocol into equivalence partitions based on the high level requirements. We can then generate test cases corresponding to partitions and determine whether the implementation is in agreement with the protocol model. If a discrepancy appears, the code or the model or both can be refined as necessary. The use of testing to maintain high-fidelity between the design model and code means that any analysis of the model is congruent with the behavior of the implementation.

5 Detailed Research Tasks

The research proposed here will focus on the use of category theory to formalize and apply existing work on inconsistency management. Category theory is useful not only for the theoretical foundation it provides for reasoning about specification compositions, but also because it gives us a precise language for talking about the relationships between specifications. It is informative to attempt to translate the informal terms we have been using in the viewpoints work into the language of category theory. For example, if we consider inter-viewpoint consistency relations as specification morphisms, then we are forced to address explicitly the question of how vocabulary used in one viewpoint maps onto the vocabulary of another.

A specification morphism maps all of the terms of A into the terms of B, and requires that all axioms of A can be proven as theorems of B. Therefore, if the morphism can be constructed, and all the necessary proof obligations dispatched, then A and B are consistent. However, this is too strong as a definition of inter-viewpoint consistency, as it effectively requires that A be included in B (with renaming). Past work on specification morphisms has been focussed on using colimits as a means of specification composition (For example, figure 1a describes the simplest such construction, a coproduct).

However, in the viewpoints framework we are interested in describing partial overlap between specifications. For any pair of viewpoints, there would typically *not* be a specification morphism between them, but there may be constructions that relate the two of them using specification morphisms. For example figure 1b shows a pushout diagram where Y represents the overlap between specifications A and B. This area of overlap is one for which we want to define a consistency relationship. If we wish to capture this consistency relationship, we need to express three things: the area of overlap (Y), and each of the morphisms (ya and yb) that define how this area of overlap is expressed in each viewpoint. Furthermore, as there may be a number of areas of overlap between any two specifications, we obtain a diagram like the one of fig 1c. It is then interesting to study the meaning of the limit of such a diagram. Hence, we can decompose a consistency relationship into its constituent parts, in order to reason about partial consistency.

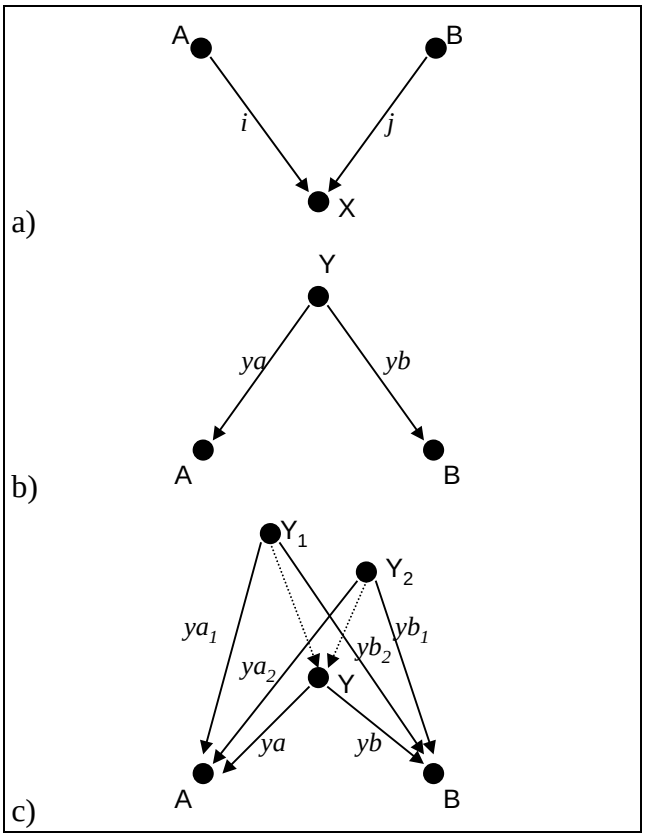

Figure 1: Universal constructions in category theory: (a) a coproduct; (b) a pushout diagram; (c) a cone.

5.1 Task 1: Tool analysis

This task will compare the available tools for categorical approaches to specification composition. We will compare Moka and Specware and explore how easily they can be extended to support different types of specification models, and different types of specification composition. For example, it should be possible to extend Specware to handle state machine models, in place of the algebraic specifications currently used. Similarly it should be possible to extend Moka to deal with morphisms between refinements of temporal specifications (at present it only deals with horizontal composition). As a result of this experimentation, we will chose one of these tools to base further research on.

5.2 Task 2: Composition of Evolving specifications

This task will explore the issues that arise when you modify specifications once morphisms have been defined between them. As specifications for large complex systems can be expected to evolve considerably during their lifetimes, we need to examine how much effort is needed to re-prove all the affected morphisms if a specification is altered, and whether an incremental approach can be used to avoid having to do large amounts of (re-)proofs. We will use the evolution of existing specifications on the case studies (e.g. using a series of versions of a single specification) to drive the investigation, although in practice we expect to have to abstract away a lot of detail in order to understand the essential nature of the changes.

The second part of this task is to explore how to support the resolution of inconsistencies. We will explore the role of partial consistency in the framework, examine how to represent consistency relationships in a categorical framework and study the use of categorical constructions to reason about the degree to which two viewpoints are consistent, and find consistent subsets of the viewpoints.

Our approach is based on the notion of *partial composition*. By partial composition we mean a relationship (morphism) between two specifications in which the specifications and the relationship between them may be

only partially defined. Traditional approaches to formalizing software development have focussed on a "correct by construction" approach, in which components are specified, proved correct, and then composed together in such a way as to preserve their properties. However, in a large project, practical considerations make such an approach awkward. In particular it is desirable to be able to mix specification and composition steps such that at any particular moment in the process we may have established only some of the properties of the components, and some of the composition relations, such that there may be many inconsistencies between them. In such a situation, we need to be able to reason about the degree of consistency, and the general effect of different development options on the consistency and completeness of the overall scheme. Also, for verification and validation, there is a need for frameworks that formalize and analyze informal specifications where "correctness by construction" was not an applicable development strategy.

5.3 Task 3 - (De)composition of consistency properties

Through case study work, we will begin to build a classification of the types of system property that can be decomposed (for verification purposes) using the specification structure achieved through the use of specification morphisms. Our aim is first to empirically validate the initial work in decomposing system properties, as described in [13], and secondly to extend this approach to consistency relationships between viewpoints, using the notion of consistency morphisms. Equally important during this task will be to find out what types of property cannot be decomposed by our approach.

5.4 Task 4 - Compositionality of test cases

This task will explore the link between specification composition and testing. In contrast with other approaches to creating large formal specifications (e.g. abstraction and projection) the composition of specifications via specification morphisms is expected to provide a good fit with the decomposition of system functionality for test purposes. We will explore how well the composition achieved in tools such as Moka and Specware helps in structuring test cases, especially in relation to our case studies. In particular, we will explore whether this approach allows us to scale up our work on the generation of test cases from model checkers.

5.5 Task 5 - Composition of heterogeneous specifications

The long term success of the approach depends on our ability to define specification morphisms between specifications written in different notations. Existing work on institution morphisms has demonstrated that category theory provides a suitable framework of the use of multiple logics in software development. We will explore how this work can be applied to the management of inconsistency between heterogeneous viewpoints. Our starting point for this task will be to investigate how to combine multiple model checkers.

6 Evaluation

The expected benefits of this approach are an improvement in software quality and safety, by increasing the ability of software developers to detect and track inconsistencies between development artifacts. Since it is difficult to assess direct effects of our work on qualitative properties in a quantitative fashion, we will measure specific forms of traceability provided by our approach relative to existing methods. Appropriate measures of traceability include:

- Does the analysis technique reveal classes of errors that were difficult or time consuming to detect using existing methods?
- Does the approach offer concrete representations for relationships between software development artifacts such that the relationship can be automatically tracked as the development proceeds?
- Does the approach allow for assessment of design alternatives based on a comparison of different ways of resolving inconsistencies?

Any improvement in traceability will have a significant impact on software development efforts in large organizations. For example, the lack of information about design decisions is a major factor in the high cost of software maintenance [15]. As software evolves through its "adoption" by different contractors, people, projects, and applications, it becomes more difficult to discover the reasons for various design decisions when trying to make changes to the software. Rediscovering such decisions in an expensive and time consuming task.

7 Summary

The lack of traceability between the activities of developers in large, complex projects leads to inconsistencies between the artifacts they produce during the software development process. Such inconsistencies, left

unmanaged, tend to become major sources of costly errors and rework in later phases of a project's lifecycle. To remedy this problem, we propose to investigate the use of new technology and methods that will help automate traceability between the artifacts and help manage inconsistencies that inevitably arise. By managing these inconsistencies properly, we hope to better control their potential effects on project quality and safety.

Our research addresses the twin challenges of developing analysis techniques that reveal and characterize inconsistencies during software development, and providing a modeling scheme to represent and track consistency relationships between the various artifacts developed during the software process.

A general solution to the problems of traceability and inconsistency is a difficult problem, but we intend to build upon previous, successful work by applying our work to additional real-world projects. Only through applied research can we refine potential solutions and discover new avenues that will lead to practical solutions to the complex problems inherent in large software development efforts.

8 References

[1] S. Easterbrook and J. Callahan, "Formal Methods for V&V of partial specifications: An experience report," *Proceedings, Third IEEE Symposium on Requirements Engineering (RE'97)*, Annapolis, Maryland, 5-8 January 1997.

[2] S. M. Easterbrook and B. Nuseibeh, "Using ViewPoints for Inconsistency Management," *Software Engineering Journal*, vol. 11, pp. 31-43, 1996.

[3] M. Heimdahl and N. Leveson, "Completeness and Consistency Analysis of State-Based Requirements," *IEEE Transactions on Software Engineering*, vol. 22, pp. 363-377, 1996.

[4] C. L. Heitmeyer, B. Labaw, and D. Kiskis, "Consistency Checking of SCR-Style Requirements Specifications," *Second IEEE Symposium on Requirements Engineering*, York, UK, March 27-29, 1995.

[5] D. Jackson, "Structuring Z Specifications with Views," *ACM Transactions on Software Engineering and Methodology*, vol. 4, pp. 365-389, 1995.

[6] Y. V. Srinivas and R. Jullig, "Specware(TM): Formal Support for Composing Software," *Proceedings of the Conference on Mathematics of Program Construction*, Kloster Irsee, Germany, July 1995.

[7] F. Schneider, S. M. Easterbrook, J. R. Callahan, and G. J. Holzmann, "Validating Requirements for Fault Tolerant Systems using Model Checking," *Third IEEE Conference on Requirements Engineering*, Colorado Springs, CO, April 6 - 10, 1998.

[8] W. Swartout and R. Balzer, "On the Inevitable Intertwining of Specification and Implementation," *Communications of the ACM*, vol. 25, pp. 438-440, 1982.

[9] A. C. W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh, "Inconsistency Handling in Multi-Perspective Specifications," *IEEE Transactions on Software Engineering*, vol. 20, pp. 569-578, 1994.

[10] B. Nuseibeh, A. Russo, and J. Kramer, "Restructuring Requirements Specifications for Managing Inconsistency and Change," *Third IEEE Conference on Requirements Engineering*, Colorado Springs, CO, April 6 - 10, 1998.

[11] R. M. Burstall and J. A. Goguen, "Putting theories together to make specifications," *Fifth International Joint Conference on Artificial Intelligence*, Cambridge MA.

[12] J. Fiadeiro and T. Maibaum, "Temporal Theories as modularisation units for concurrent system specifications," *Formal Aspects of Computing*, vol. 4, pp. 239-272, 1992.

[13] P. Michel and V. Wiels, "A framework for Modular Formal Specification and Verification," *Formal Methods Europe '97*, Graz, Austria, 15-19 September 1997.

[14] J. R. Callahan and T. L. Montgomery, "An Approach to Verification and Validation of a Reliable Multicasting Protocol," *International Symposium on Software Testing and Analysis (ISSTA'96)*, San Diego, CA, 8-10 January 1996.

[15] D. Garlan, G. E. Kaiser, and D. Notkin, "Using Tool Abstraction to Compose Systems," *IEEE Computer*, vol. 25, pp. 30-38, 1992.