Model Management and Inconsistency in Software Design

Steve Easterbrook
Department of Computer Science, University of Toronto
40 St George Street, Toronto, Ontario, M5S 2E4, Canada
http://www.cs.toronto.edu/~sme

Abstract

The management of inconsistency between multiple viewpoints is a central problem in the design of large software systems. However, many of the formalisms we use in software design cannot handle inconsistent descriptions. This leads to two common reactions: to abandon the formalisms (and use semi-formal or informal design notations) or to repair inconsistent descriptions at all costs, if necessary by discarding problematic information. Both reactions represent a retreat from support for a full range of design activities. Instead, we argue the need for formal reasoning systems that can tolerate inconsistent information. A key observation is that model management *is a central activity in software design. Rather than seeking to build a single consistent model, software designers need to reason about the inconsistencies and dependencies between a set of inter-related partial models. We are currently investigating the use of paraconsistent logics to reason with information from inconsistent models.*

1. Viewpoint Integration in SE

For the past 15 years, we have been studying the problem of viewpoint integration in Software Engineering. Viewpoints are used in SE to support a loosely-coupled distributed approach to software development, in which different participants are able to maintain their own (partial) models of the system and its requirements, without being constrained by the need to be consistent with other participants' models [3]. By exploring the relationships between viewpoints, and the inconsistencies that arise when intended relationships do not hold, the participants discover disagreements, and understand one another's perspectives better.

The key insight of the viewpoints work is to see software development as a problem of model management, with the attendant goal of seeking coherence in information drawn from disparate sources. Software designers create models in a variety of notations to capture their current understanding of the problem and these models are rarely static. Designers analyze their models in various ways, and use the results of these analyses to improve them. They create multiple versions of their models to explore design options, and to respond to changing requirements. Hence, most of the time, design models are likely to be incomplete and inconsistent. Managing inconsistency as these models evolve is a major challenge.

In its narrowest sense, *consistency* is usually taken to mean *syntactic consistency*. In a good modeling language, syntactic consistency should correspond to the developer's intuitive notion of a "well-formed model". Hence, syntactic inconsistencies indicate simple mistakes, or slips, made by the designer. In this view, detection and resolution of inconsistency can be thought of as "model hygiene".

In our work, we have taken a much broader view of consistency. In our view, an inconsistency occurs whenever some relationship that *should* hold (of a model) has been violated. This definition has an intentional flavour: someone (e.g. the designer) *intends* that certain relationships hold. Such relationships may be internal to a model (e.g. the definition of an element should be consistent with its use), or may refer to external relationships (e.g. a model should be consistent with a particular choice of semantics, with existing standards, with good practice guidelines, or with another model, etc). This definition of inconsistency spans the semantics and pragmatics (i.e. the intended meanings and uses) of model elements, in addition to their syntax.

This view has several interesting consequences. Firstly, by this definition, most conceptual models are inconsistent most of the time, and attempting to remove all inconsistency is usually infeasible. Design involves finding acceptable compromises, rather than seeking perfection. Hence, in our work on consistency management, we don't view detection and removal of inconsistency as the main goal; instead, we focus on tools to explore the consistency relationships, and on reasoning techniques that tolerate inconsistency [7].

Secondly, most of the interesting consistency relationships arise implicitly as models are developed. If we wish to provide automated tools for consistency management, such consistency relationships have to be captured and represented. Thirdly, because of the intentional nature of these relationships, the set of relevant consistency relationships for a given model will change over time as the developer's intent changes.

We have made significant progress in the past 15 years in our study of these ideas.

- We have developed a number of representation schemes for capturing and managing the consistency relationships in modeling languages. These include a

first order logic for checking XML documents [6], a production rule approach for checking UML models [5] and a structural mapping technique based on graph morphisms for graphical notations [8]

- We have developed a number of reasoning techniques that tolerate inconsistency. In general, these make use of paraconsistent logics, i.e. non-classical logics whose entailment relations are not explosive under contradiction. For example, we have explored the use of a family of multi-valued logics identified by Fitting [4], and demonstrated that we can build practical reasoning engines for these logics [1].
- We have developed a theoretical framework for combining information from multiple, inconsistent sources, without first resolving the inconsistencies [8]. The composition technique we use in this framework preserves information about relative certainty and inconsistency of the source models.

2. Implications for the Science of Design

Drawing on our experience with work on viewpoints and inconsistency management, we can make the following observations of a typical software design process:

- By its very nature, design involves the integration of information from a heterogenous collection of knowledge sources each provided by a particular stakeholder with a particular interest in the outcome.
- The knowledge from these disparate sources will not be static – it will evolve as the design process proceeds, and as the participants' understandings and expectations change.
- For much of the time, a designer's current understanding of an evolving design will be based on partial knowledge of these disparate sources. Hence, if designers are to build explicit models of their current understanding, these models must perforce be partial.
- For much of the time, such models will be inconsistent with one another, in terms of the meanings attached to model elements, and the ways in which those elements are used.
- A coherent design can only be achieved if the (intentional) consistency relationships between the partial models can be captured and made explicit.
- Analysis of an emerging design will only be possible if we have automated tools for testing these consistency relationships to identify inconsistencies.
- The process of investigating these inconsistencies and deciding how to resolve them is one of the key drivers that pushes the design process forward; it is certainly a key element in how teams of designers come to achieve a shared understanding of their designs [2].
- At any given moment, a snapshot of the current design will contain inconsistencies that are not immediately repairable, because such repair is not a matter of model hygiene, but is rather a substantial part of the design process itself.
- Hence, analysis tools that are intended to reason about a design as it evolves must be tolerant of inconsistency.

It should be clear by now that we believe a central problem in large-scale software design is the management of inconsistency between fragmentary design models. We believe our work on consistency management in the viewpoints framework suggests some promising ways forward. In particular, we believe we have practical solutions to two of the greatest challenges: representing the consistency relationships between models, and reasoning over composite models that contain inconsistencies. Several of the techniques described above are applicable.

3. References

[1] M. Chechik, B. Devereux, S. M. Easterbrook & A. Gurfinkel "Multi-Valued Symbolic Model-Checking". To appear, ACM Trans. on Software Engineering and Methodology, 2003.

[2] S. M. Easterbrook, "Coordination Breakdowns: Why Groupware is so Hard to Design". Proceedings, 28th Hawaii International Conference on Systems Sciences (HICSS-28), Hawaii, 3-6 January, 1995, Pp 191-199.

[3] S. M. Easterbrook & B. A. Nuseibeh "Managing Inconsistencies in an Evolving Specification". 2nd IEEE Int. Symp. on Requirements Engineering (RE'95), York, UK, p48-55. Apr 1995.

[4] M. Fitting "Kleene's three-valued logics and their children". Fundamenta Informaticae, 20, 113-131, 1994.

[5] W. Liu, S. M. Easterbrook & J. Mylopoulos, "Rule-Based Detection of Inconsistency in UML Model". Workshop on Consistency Problems in UML-Based Software Development, 5th Int. Conference on the Unified Modeling Language, Dresden, Germany, Oct 1, 2002.

[6] C. Nentwich, W. Emmerich, A. Finkelstein and E. Ellmer, "Flexible Consistency Checking" ACM Trans. on Software Engineering and Methodology 12 (1) 28-63, 2003.

[7] B. A. Nuseibeh, S. M. Easterbrook & A. Russo, "Making Inconsistency Respectable in Software Development", J. of Systems and Software, 58 (2) 171-180. 2001.

[8] M. Sabetzadeh & S. M. Easterbrook "Analysis of Inconsistency in Graph-Based Viewpoints: A Category-Theoretic Approach". 18th IEEE Int. Conf. on Automated Software Engineering, Montreal, Oct. 6-10, 2003.