The Single Model Principle

Richard Paige and Jonathan Ostroff
Department of Computer Science, York University, Toronto, Canada.
{paige,jonathan}@cs.yorku.ca

Abstract

We contrast modelling languages that are founded on use of a single model with those founded on use of multiple models. We propose that to best support seamless and reversible development of reliable software, languages that follow the single model principle are superior. We define this principle precisely, and discuss when it is insufficient, particularly for early requirements engineering.

1. Introduction

Software development often makes use of modelling languages, e.g., UML [5] and Eiffel [3], for describing systems and requirements. There are two fundamental types of modelling languages: those that use a *single model*, perhaps with multiple views, to describe concerns of interest; and those that make use of multiple, independently constructed models. We summarize a critical comparison of these two types of languages, particularly for building reliable software. We refer the reader to a more detailed comparison in [6]. We focus on the use of modelling languages in late requirements engineering, design, and implementation; early requirements engineering is discussed in Section 3.

1.1. Using multiple models

A representative example of a modelling language that uses multiple models is UML. With UML, concerns are described using independently constructed models [5]. Each model describes concerns from a different perspective; information presented in one model may also be captured, albeit differently, in another model. By using multiple models, developers can work independently and can apply the most appropriate notations for their tasks. When it comes time to construct executable code, these models must be integrated into a single model that satisfies all the constraints and descriptions contained in the individuals.

1.2. Using a single model

Modelling languages that make use of a single model obey the *single model principle*, defined precisely in the sequel. An example of a modelling language that obeys the single model principle is Eiffel[1]. With Eiffel, a single model is constructed and used throughout software development, with automatic or semi-automatic generation of different views [4] of the model.

1.3. Seamlessness and reversibility

Seamless and reversible development [3, 7] is founded on the use of a set of common modelling abstractions throughout development. It allows avoidance of impedance mismatches that potentially introduce errors during development; it also allows code, designs, and requirements to be maintained and kept consistent. Seamless development has been shown to be particularly effective in producing reliable software systems [7].

We propose that to best enforce and support seamless and reversible development in late requirements engineering through implementation, a modelling language should obey the single model principle.

1.4. Insufficiency of a single model

We are positing a single model-based approach as superior to multiple model approaches for late requirements engineering through implementation. We do not claim that it is ideal for all of software development (as we discuss in Section 3, it is not appropriate for early requirements analysis). Nor do we claim that multiple *views* are valueless. Our perspective is that multiple views should be consistent by construction wherever possible, thus implying that they should be produced – ideally automatically – from a single model.

[1] It is a mistake to consider Eiffel as just a programming language; indeed, Meyer [3] calls it a method. Eiffel is also a modelling language, and can be used to write both abstract specifications and code.

Proceedings of the Fifth International Symposium on Requirements Engineering (RE'01)
1090-705X/01 $10.00 © 2001 IEEE

2. The Single Model Principle

In developing a modern software system, two general kinds of *abstractions* are typically produced: modules (units of encapsulation), and systems (collections of modules and other systems). Based on these abstractions, we can define the single model principle precisely.

Definition 1. *Single Model Principle.* A modelling language that obeys the single model principle satisfies the following three criteria:

1. **Conceptual integrity.** A language with conceptual integrity possesses *physical integrity* of descriptions: for each different kind of abstraction, all information about the abstraction is kept in exactly one physical place, e.g., a file. A language with conceptual integrity also provides *exactly one* way of describing concepts of interest. For example, Java possesses physical integrity, but not conceptual integrity: the concepts of class and interface are semantically redundant.

2. **Consistency of views.** The language must provide infrastructure that makes the checking of the consistency of different views of a model as automatable as possible, e.g., the frameworks of [4, 8]. It is preferable to guarantee consistency of views by construction.

3. **Wide-spectrum applicability.** The language is applicable to modelling concepts and constructs throughout late requirements engineering (when customer goals are well-defined, and alternatives have been considered) through to implementation.

Discussion of languages and methods that support the principle, and which fail to support the principle, e.g., UML, is presented in [6].

3. Where the principle is insufficient

The single model principle is insufficient in several situations. We discuss early requirements engineering here, and discuss dealing with inconsistency in [6]. Dynamic modelling (e.g., via sequence diagrams) is compatible with the principle, as discussed in [6].

Early requirements engineering is focussed on understanding, capturing, and analyzing specific customer goals. A modelling language like Eiffel, based on the single model principle, is insufficient for modelling goals. It provides no built-in techniques for modelling goals: they would have to be treated informally, and would not be directly expressible in executable code, thus defeating seamlessness. In order to treat goals formally, a richer modelling language, e.g., KAOS [1], or task-based modelling [2] could be applied. Impedance mismatches could be minimized by providing transformation rules from goal-based models to a language such as Eiffel.

4. Discussion and Conclusions

Independently generated multiple models of a system cause more problems than they solve in developing software. It is claimed that they are useful because they allow developers to work independently on different parts of a software system, and thereafter their individual work can be integrated. This causes problems, particularly for establishing an implementable design.

The multiple model approach offered by languages such as UML is not a good way to build reliable software. It is not a good mechanism for ensuring consistency, nor to help trace errors in programs back to errors in models. A single model approach, wherein different views of a system can be automatically or partly automatically generated from a single model of the system, should be preferred for developing reliable software systems.

We have used Eiffel to illustrate the single model principle, but our arguments are not limited to this modelling language: they apply to any language which provides a unique way of describing abstractions of a system of interest. It remains to be seen whether Eiffel is an appropriate language for implementing the principle.

References

[1] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.

[2] I. Graham. *Object-Oriented Methods*. Addison-Wesley, 2001.

[3] B. Meyer. *Object Oriented Software Construction, Second Edition*. Prentice Hall, 1997.

[4] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specifications. *IEEE Transactions on Software Engineering*, 20(10):760–773, October 1994.

[5] Object Modelling Group. UML Standard Guide 1.3, 1999.

[6] R. Paige and J. Ostroff. Producing reliable software via the single model principle. Technical Report CS-TR-2001-02, York University, www.cs.yorku.ca/techreports/2001/CS-2001-02.html, 2001.

[7] K. Walden and J.-M. Nerson. *Seamless Object Oriented Software Architecture*. Prentice Hall, 1995.

[8] P. Zave and M. Jackson. Conjunction as composition. *ACM Transactions on Software Engineering and Methodology*, 2(4), October 1993.

1090-705X/01 $10.00 © 2001 **IEEE**