Documenting Software Architectures: Views and Beyond

Paul Clements[*], David Garlan[**], Reed Little*, Robert Nord*, Judith Stafford*
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 USA
{clements, little, rn, jas}@sei.cmu.edu, garlan@cs.cmu.edu

Software architecture has emerged as a foundational concept for the successful development of large, complex systems. Signs that the field is maturing to become an engineering discipline include textbooks on the subject; the work in repeatable design, exemplified by architectural styles and patterns; robust methods for evaluating software architectures; international conferences devoted to it; and recognition of architecture as a professional practice.

However, treatment of architecture to date has largely concentrated on its design, and to a lesser extent, its validation. But effectively *documenting* an architecture is as important as crafting it, because if the architecture is not understood (or worse, misunderstood) it cannot meet its goals as the unifying vision for system and software development.

Three years ago, researchers at the Software Engineering Institute and the Carnegie Mellon School of Computer Science set out to answer the question: "How should you document an architecture so that others can successfully use it, maintain it, and build a system from it?" The result of that work is an approach we loosely call "views and beyond." [1]

Modern software architecture practice embraces the concept of architectural *views*. A view is a representation of a set of system elements and relations associated with them. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures, which we represent as views.

Some authors prescribe a fixed set of views with which to engineer and communicate an architecture. Rational's Unified Process, for example, is based on Kruchten's 4+1 view approach to software [4]. The Siemens Four Views model [3] is another example. A recent trend, however, is to recognize that architects should produce whatever views are useful for the system at hand. IEEE 1471 exemplifies this philosophy; it holds that an architecture description consists of a set of views, each of which conforms to a *viewpoint*, which in turn is a realization of the concerns of one or more stakeholders.

This philosophy about views leads to the fundamental principle of the views-and-beyond approach:

> *Documenting an architecture is a matter of documenting the relevant views, and then adding documentation that applies to more than one view.*

This tutorial provides a practical and comprehensive approach to software architecture documentation that helps architects (a) understand what views are available with which to document an architecture and how each can be used in the system's lifecycle; (b) how to choose the set of views that will be most valuable to the architecture's community of stakeholders; (c) the information needed to document a view; and (d) the information needed to document information that applies across views. The tutorial presents the information in the context of prevailing prescriptive models for architecture, including the Rational Unified Process [4], Siemens Four Views [3], ANSI/IEEE-1471-2000 [2], and the UML. This tutorial is intended to help software architects and software lead designers whose job includes producing architectural documentation. The tutorial is also aimed at software technical managers whose job includes overseeing and managing the architecture documentation process, including the incorporation of stakeholder interests and needs.

Outline

The major topics of the tutorial include:

Principles of sound documentation. This lecture will cover seven fundamental principles that apply to all technical documentation, but especially to architecture documentation: (1) Write from the point of view of the reader; (2) Avoid unnecessary repetition; (3) Avoid ambiguity; (4) Use a standard organization; (5) Record rationale; (6) Keep documentation current but not too current; and (7) Review documentation for fitness of purpose.

---

[*] This author is with the Software Engineering Institute at Carnegie Mellon University.
[**] This author is with the School of Computer Science at Carnegie Mellon University.

Proceedings of the 25th International Conference on Software Engineering (ICSE'03)
0270-5257/03 $17.00 © 2003 IEEE

Viewtypes, styles, and views. The first principle for documenting software architectures is to document the relevant views and then document the information that applies beyond views. But what views are relevant? Rational's 4+1 approach prescribes one set; the Siemens Four Views approach prescribes another set. In order to choose the relevant views to document, the software architect must understand what views are available. Fundamentally, architects must think about the system in three ways:

(1) how the system is structured as a set of implementation units;

(2) how the system is structured as a set of elements that interact dynamically with each other at runtime; and

(3) how the software structures correspond to structures in the system's environment.

Views correspond to these informational categories, which we call *viewtypes*. Views in the *module viewtype* show elements that correspond to implementation units. Views in the *component-and-connector viewtype* show elements in execution. Views in the *allocation viewtype* show how the software relates to environmental entities. Within each viewtype, there are well-known patterns of design decisions (*styles*) that the architect can re-use. A style applied to a system is a *view*. This lecture establishes the important concepts of viewtype, style, and view, with examples.

The Style Zoo. This lecture will cover some of the styles commonly seen in each of the three viewtypes. Module styles covered include decomposition, layered, and generalization. Component-and-connector styles covered include shared-data, communicating-processes, client-server, and pipe-and-filter. Allocation styles covered include the implementation, work assignment, deployment. Each style is presented covering its elements and relations, what the style is useful for, what other styles are related (or often confused with), and some common ways to represent the style using formal and informal notations, especially the UML.

Advanced concepts. This lecture covers several advanced topics that a software architect must be comfortable with in order to produce a sound documentation package. These concepts include (1) refinement, the mechanism by which information is "chunked" into digestible units; (2) context diagrams, which show the relation of the software being documented to its environment; (3) combined views, which are a mechanism to convey different kinds of information in a single view in a disciplined, structured fashion; (4) variability, which is about showing how the architecture can vary at build-time or runtime; and (5) software interfaces, which covers how to document the interfaces to the architectural elements shown in the views.

Choosing the views. This lecture presents a simple 3-step procedure for choosing the relevant views. The procedure is stakeholder-based, using stakeholders (or stakeholder proxies) to determine the uses to which the documentation will be put, and the information best positioned to satisfy those uses. This ensures that the resulting documentation package will be one that the stakeholders find helpful and useful..

Building the documentation package. This lecture presents a seven-part template that architects can use to document views, and a 7-part template that architects can use to document the information beyond views.

Conclusions. This lecture maps the concepts and templates explored in this tutorial with well-known architectural prescriptions, including the 4+1 approach of the Rational Unified Process, the Siemens Four Views approach, and the ANSI/IEEE-1471-2000 recommended best practice for documenting architectures for software-intensive systems. The lecture concludes by re-capping the highlights of the tutorial, and asking for feedback.

References

[1] Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, Boston, 2002.

[2] IEEE 2000. IEEE Product No.: SH94869-TBR: Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Standard No. 1471-2000. Available at http://shop.ieee.org/store/.

[3] Hofmeister, C., R. Nord, and D. Soni. *Applied Software Architecture*, Addison-Wesley, Boston, 2000.

[4] Kruchten, P. *The Rational Unified Process: An Introduction*, Second Edition. Addison-Wesley, Boston, 2001.