Chapter

39

DOCUMENTING ARCHITECTURE: UML & THE N+1 VIEW MODEL

They have computers, and they may have other weapons of mass destruction.

—USA government official

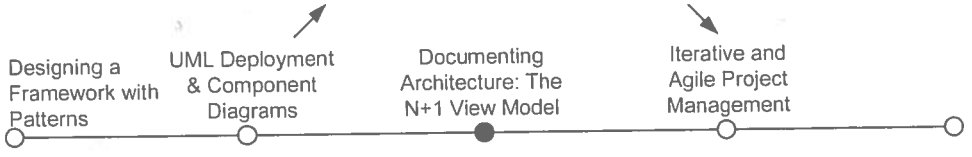
Objectives

- Create useful architecture documentation based on the N+1 (or 4+1) view model.
- Apply various UML diagram types.

Introduction

Once an architecture takes shape, it may be useful to describe it, so that new developers can learn the big ideas of the system, or so that there is a common view from which to discuss changes. In the UP, the artifact that describes this is the **Software Architecture Document** (SAD). The chapter introduces the SAD and its contents.

What's Next? Having summarized relevant UML notation, this chapter is an introduction to documenting an architecture with UML and the well-known N+1 view model. The next explores more issues in iterative development, and managing an iterative project with agile practices.



39.1 The SAD and Its Architectural Views

The Software Architect Document

In addition to the UML package, class, and interaction diagrams, another key artifact in the UP Design Model is the SAD. It describes the big ideas in the architecture, including the decisions of architectural analysis. Practically, it is a *learning aid* for developers who need to understand the essential ideas of the system.

The essence of the SAD is a summary of the architectural decisions (such as with technical memos) and the N+1 architectural views.

Motivation: Why Create a SAD?

When someone joins the development team, it's useful if the project coach can say, "Welcome to the NextGen project! Please go to the project website and read the ten page SAD in order to get an introduction to the big ideas." And later, during a subsequent release, when new people work on the system, a SAD can be a learning aid to speed their comprehension.

Therefore, it should be written with this audience and goal in mind: What do I need to say (and draw in the UML) that will quickly help someone understand the major ideas in this system?

Architectural Views

Having an architecture is one thing; a useful description is something else.

In [Kruchten95], the influential and widely adopted idea of describing an architecture with multiple views was promoted; its multiple-view model is now considered the state of the practice. The essential idea of an **architectural view** is this:

> *Definition: Architectural View*
>
> A view of the system architecture from a given perspective; it focuses primarily on structure, modularity, essential components, and the main control flows. [RUP].
>
> An important aspect of the view missing from this RUP definition is the *motivation*. That is, an architectural view should explain why the architecture is the way it is.
>
> An architectural view is a window onto the system from a particular perspective that emphasizes the key noteworthy information or ideas, and ignores the rest.

An architectural view is a tool of communication, education, or thought; it is expressed in text and UML diagrams.

For example, the NextGen package and interaction diagrams shown in Chapter 34 on layering and logical architecture show the big ideas of the logical structure of the software architecture. In the SAD, the architect will create a section called *Logical View*, insert those UML diagrams, and add some written commentary on what each package and layer is for, and the motivation behind the logical design.

A key idea of the architectural views—which concretely are text and diagrams—is that they do *not* describe *all* of the system from some perspective, but only outstanding ideas from that perspective. A view is, if you will, the "one-minute elevator" description: What are the most important things you would say in one minute in an elevator to a colleague on this perspective?

Architectural views may be created:

- after the system is built, as a summary and learning aid for future developers
- at the end of certain iteration milestones (such as the end of elaboration) to serve as a learning aid for the current development team, and new members
- speculatively, during early iterations, as an aid in creative design work, recognizing that the original view will change as design and implementation proceeds

The N+1 (or 4+1) View Model

In his seminal paper, Kruchten not only promoted documenting an architecture from different views, but more specifically, showing the **4+1** views, which today has expanded more generally to the **N+1** views, reflecting the myriad concerns in a system.

Briefly, the 4 views described in the paper are: logical, process, deployment, and data. These are described in a following section. The '+1' view is the **use case view**, a summary of the most architecturally significant use cases or scenarios, and perhaps a summary of use-case realizations for these. The use case view pulls together a common story that ties together an understanding of the other views and how they interrelate.

Architectural Views in More Detail

Myriad views are possible, each reflecting a major architectural viewpoint on to a system; here is a list of common views:

1. **Logical**
    - Conceptual organization of the software in terms of the most important layers, subsystems, packages, frameworks, classes, and interfaces. Also summarizes the functionality of the major soft-

ware elements, such as each subsystem.

- Shows outstanding use-case realization scenarios (as interaction diagrams) that illustrate key aspects of the system.
- A view onto the UP Design Model, visualized with UML package, class, and interaction diagrams.

2. **Process**
   - Processes and threads. Their responsibilities, collaborations, and the allocation of logical elements (layers, subsystems, classes, ...) to them.
   - A view onto the UP Design Model, visualized with UML class and interaction diagrams, using the UML process and thread notation.
3. **Deployment**
   - Physical deployment of processes and components to processing nodes, and the physical network configuration between nodes.
   - A view onto the UP Deployment Model, visualized with UML deployment diagrams. Normally, the "view" is simply the entire model rather than a subset, as all of it is noteworthy. See Chapter 38 for the UML deployment diagram notation.
4. **Data**
   - Overview of the data flows, persistent data schema, the schema mapping from objects to persistent data (usually in a relational database), the mechanism of mapping from objects to a database, database stored procedures and triggers.
   - In part, a view onto the UP Data Model, visualized with UML class diagrams used to describe a data model.
   - Data flows can be shown with UML activity diagrams.
5. **Security**
   - Overview of the security schemes, and points within the architecture that security is applied, such as HTTP authentication, database authentication, and so forth.
   - Could be a view onto the UP Deployment Model, visualized with UML deployment diagrams that highlight the key points of security, and related files.
6. **Implementation**
   - First, a definition of the **Implementation Model**: In contrast to the other UP models, which are text and diagrams, this "model" *is* the actual source code, executables, and so forth. It has two parts: 1) deliverables, and 2) things that create deliverables (such as source code and graphics). The Implementation Model is all of this

39.2

stuff, including Web pages, DLLs, executables, source code, and so forth, and their organization—such as source code in Java packages, and bytecode organized into JAR files.

- The implementation view is a summary description of the noteworthy organization of deliverables and the things that create deliverables (such as the source code).
- A view onto the UP Implementation Model, expressed in text and visualized with UML package and component diagrams.

7. **Development**
   - This view summarizes information developers need to know about the setup of the development environment. For example, how are all the files organized in terms of directories, and why? How does a build and smoke test run? How is version control used?
8. **Use case**
   - Summary of the most architecturally significant use cases and their non-functional requirements. That is, those use cases that, by their implementation, illustrate significant architectural coverage or that exercise many architectural elements. For example, the *Process Sale* use case, when fully implemented, has these qualities.
   - A view onto the UP Use-Case Model, expressed in text and visualized with UML use case diagrams and perhaps with use-case realizations in UML interaction diagrams.

Guideline: Don't Forget the Motivation!

Each view includes not only diagrams, but text that expands and clarifies. In this prose section, an often forgotten but tremendously important section is to discuss the *motivation*. *Why* is the security the way it is? *Why* are the three major software components deployed on two computers rather than three? Indeed, this section often becomes more important than any other when it comes time to make significant changes to the architecture.

39.2 Notation: The Structure of a SAD

The following SAD structure is essentially the format used in the UP: