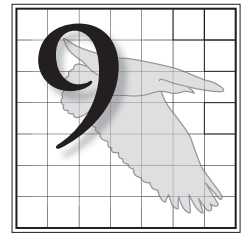

Choosing the Views

As we have seen, a large part of designing the architecture for a system consists of choosing and designing software structures, often as described in terms of architecture styles. Choosing, for example, a service-oriented style for your system means putting a service-oriented structure in place and populating it with services and their interconnections. To the extent that you write down that structure, and the interfaces and behavior of the elements, you've created a view of your architecture, because a view is a representation of a structure.

In other words, documenting your design decisions as you make them (something we strongly recommend) produces views, which are the heart of an architecture document. It is most likely that these views are sketches more than finished products ready for public release; this will give you the freedom to back up and rethink design decisions that turn out to be problematic without having wasted time on cosmetic polish. (In some cases, they might *literally* be sketches—see Figure 11.8 for an example.)

By the time you're ready to release an architecture document, then, you're likely to have a fairly well worked-out collection of architecture views. At some point you'll need to decide which to take to completion, with how much detail, and which to include in a release. You'll also need to decide which views can be usefully combined with others, so as to reduce the total number of views in the document and reveal important relations among the views.

And that is the topic of this chapter: how an architect decides on the views to include in the documentation package.

We have tried to explain the benefits of each kind of documentation, to help you decide under what circumstances you would want to produce it. Understanding which views to produce at

Poetry is a condensation of thought. You write in a few lines a very complicated thought. And when you do this, it becomes very beautiful poetry. It becomes powerful poetry.

—Chen Ning Yang, winner of the Nobel Prize in Physics, 1957 (quoted in Moyers 1989, p. 313)

Combined views can be produced by defining a hybrid style, or by making an overlay. These are discussed in Section 6.6.

If you can't afford to produce a particular part of the architecture documentation package, at least make sure you understand what the long-term cost will be for the short-term savings. Use the formula in Section P.2.4 in the prologue to help you estimate the cost and benefit.

what time and with how much detail can be reached only in the concrete context of a project. You can determine which views are required, when to create them, and how much detail to include in order to make the development project successful if you know the following:

- What people, and with what skills, are available
- With which standards you have to comply
- What budget is on hand
- What the schedule is
- What the information needs of the important stakeholders are

This chapter is about helping you make those determinations. Once the entire documentation package has been assembled, or at opportune milestones along the way, it should be reviewed for quality, suitability, and fitness for purpose by those who are going to use it.

Chapter 11 covers the review of architecture documents by stakeholders.

9.1 Stakeholders and Their Documentation Needs

All fine architectural values are human values, else not valuable.

—Frank Lloyd Wright

To choose the appropriate set of views, you must identify the stakeholders that depend on software architecture documentation. You must also understand each stakeholder's information needs.

The set of stakeholders will vary, depending on the organization and the project. The list of stakeholders in this section is suggestive but is not intended to be complete. As an architect, one of your primary obligations is to understand who the stakeholders are for your project. Similarly, the documentation needs we lay out for each stakeholder are typical, but not definitive. So take the following discussion as a starting point and adapt them according to the needs of your project and your stakeholders.

Project managers

Project managers care about schedule, resource assignments, and perhaps contingency plans to release a subset of the system for business reasons. To create a schedule, the project manager needs information about the modules to be implemented, with some information about their complexity, such as the list of responsibilities, as well as dependencies that exist to other modules, which may suggest a certain sequence in the implementation. This person is not interested in the design specifics of any element or the exact interface beyond knowing whether those tasks have been completed. But the manager is interested in the system's overall purpose and constraints; its interaction with other systems, which may suggest an organization-to-organization interface that the manager will have to establish; and the hardware environment, which the manager may have

to procure. The project manager might create or help create the work assignment view, in which case he or she will need a decomposition view to do it.

As shown in Figure 9.1, a project manager, then, will likely be interested in

- Module views: decomposition and uses and/or layered
- Allocation views: deployment and work assignment
- Other: top-level context diagrams showing interacting systems and system overview and purpose

Members of the development team

Members of the development team, for whom the architecture provides marching orders, are given constraints on how they do their job. Sometimes a developer is given responsibility for an element he or she did not implement, such as a commercial off-the-shelf product. Someone still has to be responsible for that element, to make sure that it performs as advertised and to tailor it as necessary. This person will want to know the following:

- The general idea behind the system. Although that information lies in the realm of requirements rather than architecture, a top-level context diagram or system overview can go a long way to provide the information.
- Which element the developer has been assigned, that is, where functionality should be implemented.
- The details of the assigned element, including the data model with which it must operate.
- The elements with which the assigned part interfaces and what those interfaces are.
- The code assets the developer can make use of.

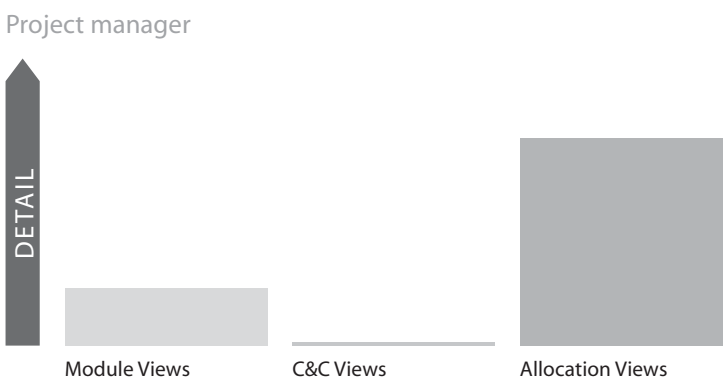

Figure 9.1
A project manager usually creates the work assignments and therefore needs some overview information of the software.

- The constraints, such as quality attributes, legacy systems interfaces, and budget, that must be met.

As shown in Figure 9.2, a developer, then, is likely to want to see

- Module views: decomposition, uses and/or layered, and generalization
- Component-and-connector (C&C) views: various, showing the component(s) the developer was assigned and the components they interact with
- Allocation views: deployment, implementation, and install
- Other: system overview; a context diagram containing the module(s) he or she has been assigned; the interface documentation of the developer's element(s) and the interface documentation of those elements with which they interact; a variability guide to implement required variability; and rationale and constraints

Testers and integrators

Testers and integrators are stakeholders for whom the architecture specifies the correct black-box behavior of the pieces that must fit together. A unit tester of an element will want to see the same information as a developer of that element, with an emphasis on behavior specifications. A black-box tester will need to see the interface documentation for the element. Integrators and system testers need to see collections of interfaces, behavior specifications, and a uses view so they can work with incremental subsets.

As shown in Figure 9.3, testers and integrators, then, are likely to want to see

- Module views: decomposition, uses, and data model
- C&C views: all

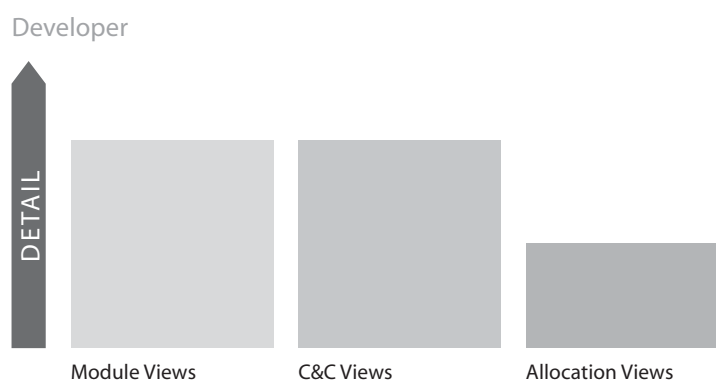

Figure 9.2
Developers have interest mainly in the software itself and therefore create detailed module and C&C views and have some interest in allocation views.

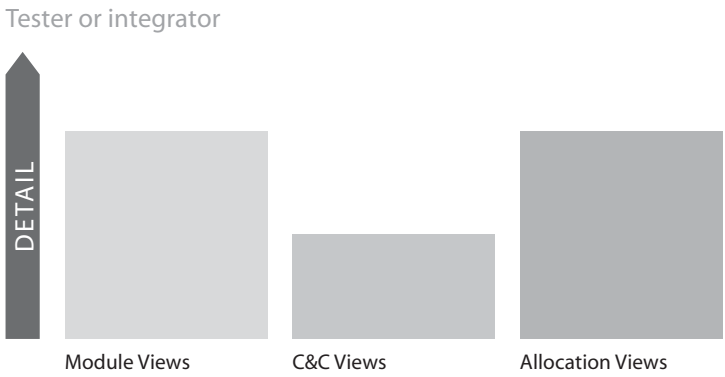

Figure 9.3
Testers and integrators need context and interface information, along with information about where the software runs and how to build incremental parts.

- Allocation views: deployment; install; and implementation, to find out where the assets to build the module are
- Other: context diagrams showing the module(s) to be tested or integrated; the interface documentation and behavior specification(s) of the module(s) and the interface documentation of those elements with which they interact

Designers of other systems

Designers of other systems with which this one must interoperate are stakeholders. For these people, the architecture defines the set of operations provided and required, as well as the protocols for their operation. As shown in Figure 9.4, these stakeholders will likely want to see

- Interface documentations for those elements with which their system will interact, as found in module and/or C&C views
- The data model for the system with which their system will interact

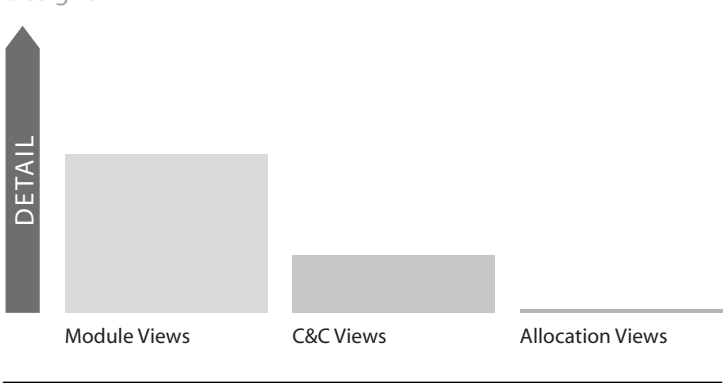

Figure 9.4
Designers of other systems are interested in interface documentation and important system behavior.

- Top-level context diagrams from various views showing the interaction

Maintainers

Maintainers use architecture as a starting point for maintenance activities, revealing the areas a prospective change will affect. Maintainers will want to see the same information as developers, for they both must make their changes within the same constraints. But maintainers will also want to see a decomposition view that allows them to pinpoint the locations where a change will need to be carried out, and perhaps a uses view to help build an impact analysis to fully scope out the effects of the change. Maintainers will also want to see design rationale that will give them the benefit of the architect's original thinking and save them time by letting them see already discarded design alternatives.

As shown in Figure 9.5, a maintainer, then, is likely to want to see the views as mentioned for the developers of a system, with special emphasis on

- Module views: decomposition, layered, and data model
- C&C views: all
- Allocation views: deployment, implementation, and install
- Other: rationale and constraints

Application builders

Application builders in a **software product line** tailor the core assets according to preplanned and built-in variability mechanisms, add whatever special-purpose code is necessary, and instantiate new members of the product line. Application builders will need to see the variability guides for the various elements, to facilitate tailoring. After that, application builders need to see largely the same information as integrators do.

A **software product line** is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of reusable core assets in a prescribed way. (Clements and Northrop 2001)

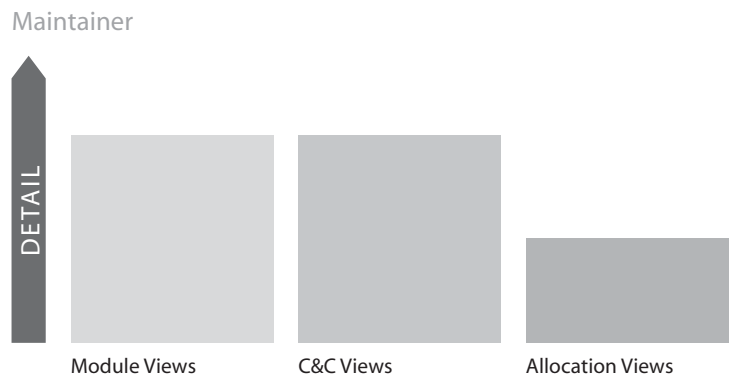

Figure 9.5
A maintainer has the same information needs as a developer but with a stronger emphasis on design rationale and variability.

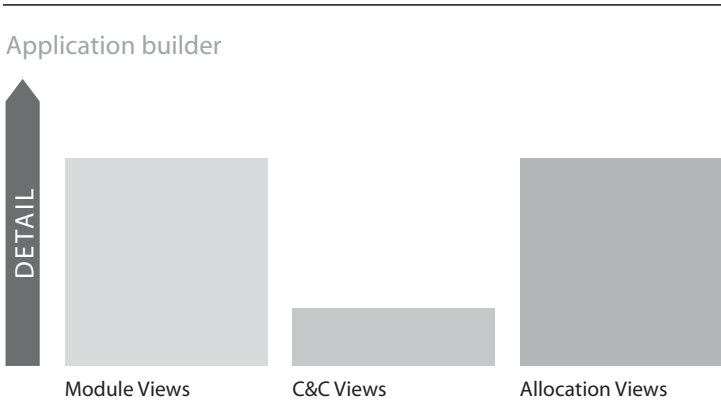

Figure 9.6
An application builder needs to understand what adaptations to make in order to build new products.

As shown in Figure 9.6, a product-line application builder, then, is likely to want to see the views mentioned for an integrator, plus

- A variability guide, as given in module and/or C&C views

Customers

Customers are the stakeholders who pay for the development of specially commissioned projects. Customers are interested in cost and progress and convincing arguments that the architecture and resulting system will meet the quality and functional requirements. Customers will also have to support the environment in which the system will run and will want to know that the system will interoperate with other systems in that environment.

As shown in Figure 9.7, the customer, then, is likely to want to see

- C&C views: the analysis results will be of particular interest

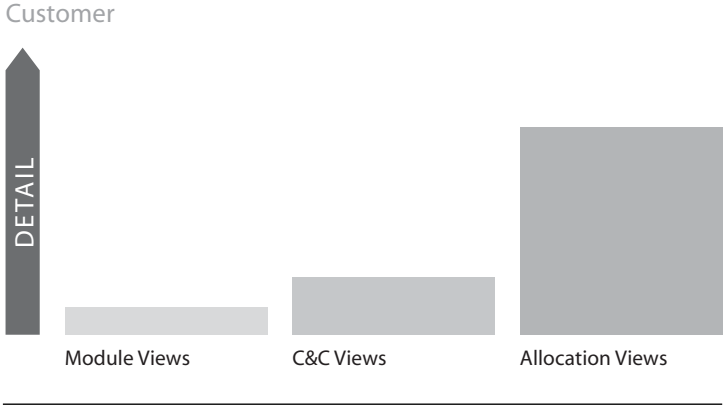

Figure 9.7
A customer is interested mainly in how the software works in the desired environment.

- Allocation views: work assignment view, no doubt filtered to preserve the development organization's confidential information, and a deployment view
- Other: a top-level context diagram in one or more C&C views

End users

End users do not need to see the architecture, which is, after all, largely invisible to them. But they often gain useful insights about the system, what it does, and how they can use it effectively by examining the architecture. If end users or their representatives review your architecture, you may be able to uncover design discrepancies that would otherwise have gone unnoticed until deployment.

To serve this purpose and as shown in Figure 9.8, an end user is likely to be interested in

- C&C views: views emphasizing flow of control and transformation of data, to see how inputs are transformed into outputs; analysis results dealing with properties of interest, such as performance or reliability
- Allocation views: a deployment view to understand how functionality is allocated to the platforms with which the users interact

Analysts

Analysts are interested in the ability of the design to meet the system's quality objectives. The architecture serves as the fodder for architecture evaluation methods and must contain the information necessary to evaluate such quality attributes as security, performance, usability, availability, and modifiability. For performance engineers, for example, architecture provides the model that drives such analytical tools as rate-monotonic real-time schedulability analysis, simulations and simulation generators, theorem provers, and model checkers. These tools require information about resource consumption, scheduling policies, dependencies, and so forth.

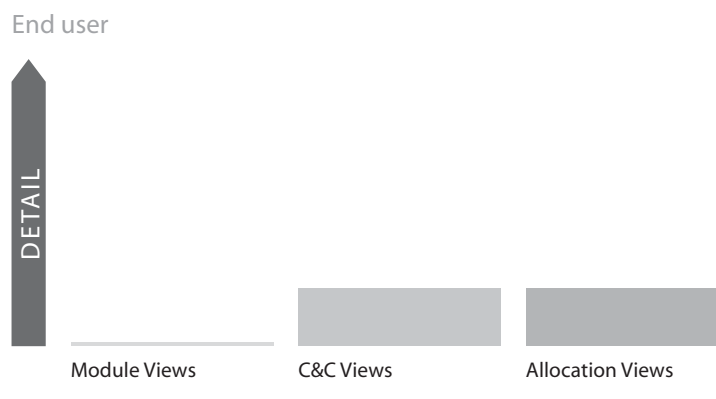

Figure 9.8
An end user needs to have an overview of the software, how it runs on the platform, and how it interacts with other software.

In addition to generalized analysis, architectures can be evaluated for the following and other quality attributes, each of which suggests certain documentation obligations.

- *Performance.* To analyze for performance, performance engineers build models that calculate how long things take. Plan to provide a communicating-processes view to support performance modeling. In addition, performance engineers are likely to want to see a deployment view, behavior documentation, and those C&C views that help to track execution.
- *Accuracy.* Accuracy of the computed result is a critical quality in many applications, including numerical computations, the simulation of complex physical processes, and many embedded systems in which outputs are produced that cause actions to take place in the real world. To analyze for accuracy, a C&C view showing flow and transformation of data is often useful because it shows the path that inputs take on their way to becoming outputs, and it helps identify places where numerical computations can degrade accuracy.
- *Modifiability.* To gauge the impact of an expected change, a uses view and a decomposition view are most helpful. Those views show dependencies and will help with impact analysis. But to reason about the runtime effects of a proposed change requires a C&C view as well, such as a communicating-processes view, to make sure that the change does not introduce deadlock.
- *Security.* A deployment view is used to see outside connections, as are context diagrams. A C&C view showing data flow and security controls is used to track where information goes and is exposed; a decomposition view is used to find where authentication and integrity concerns are handled. Denial of service is loss of performance, and so the security analyst will want to see the same information as the performance analyst.
- *Availability.* A C&C communicating-processes view will help analyze for deadlock, as well as synchronization and data consistency problems. In addition, C&C views show how redundancy, failover, and other availability mechanisms kick in as needed. A deployment view is used to show possible points of failure and backups. Reliability numbers for a module might be defined as a property in a module view, which is added to the mix.
- *Usability.* A decomposition view will enable analysis of system state information presented to the user; help with determination of data reuse; assign responsibility for usability-related operations, such as cut-and-paste and undo; and other things. A C&C communicating-processes view will enable analysis of cancellation possibilities, failure recovery, and so on.

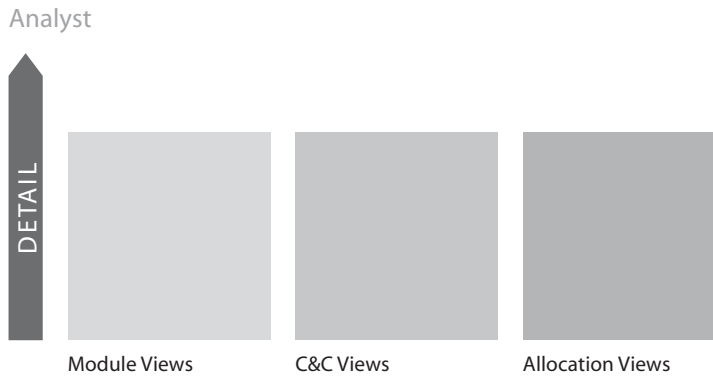

Figure 9.9
An analyst needs information from all views. Depending on the specific analysis, other, more detailed information might be required.

As shown in Figure 9.9, an analyst is likely to be interested in

- Module views: various
- C&C views: various, but especially those showing processes
- Allocation views: deployment

Infrastructure support personnel

Infrastructure support personnel set up and maintain the infrastructure that supports the development, build, and production environments of the system. You need to provide documentation about the parts that are accessible in the infrastructure. Those parts are usually elements shown in a decomposition, C&C, install, and/or implementation view. A variability guide is particularly useful to help set up the software configuration management environment.

As shown in Figure 9.10, infrastructure support people likely want to see

- Module views: decomposition and uses
- C&C views: various, to see what will run on the infrastructure

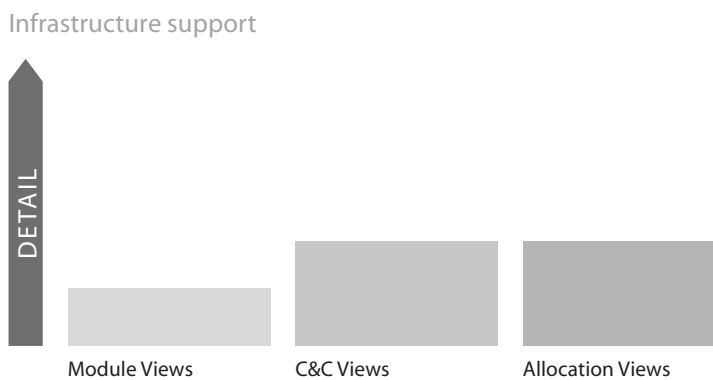

Figure 9.10
Infrastructure support people need to understand the software artifacts produced to provide tool support.

- Allocation views: deployment and install, to see where the software (including the infrastructure) will run; implementation
- Other: variability guides

New stakeholders

New stakeholders will want to see introductory, background, and broadly scoped information: top-level context diagrams, architecture constraints, overall rationale, and root-level views, as shown in Figure 9.11. People new to the system will usually want to see the same kind of information as their counterparts who are more familiar with the system, but new people will want to see it in less detail.

Future architects

Future architects are the most avid readers of architecture documentation, with a vested interest in everything. After the current architect has been promoted for producing the exemplary documentation, the replacement will want to know all the key design decisions and why they were made. As shown in Figure 9.12, future architects are interested in it all, but they will be

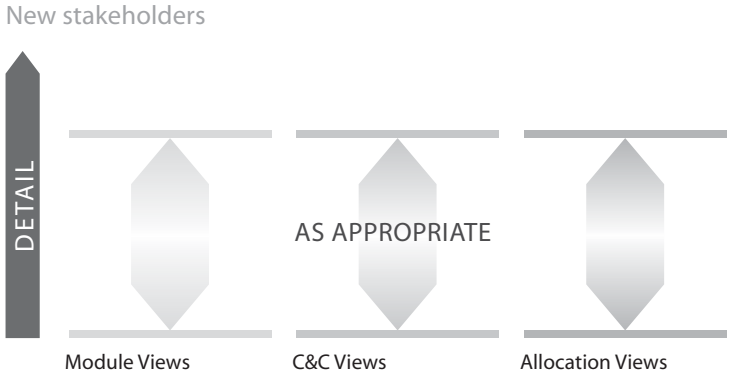

Figure 9.11
New stakeholders need to have the same information as their counterparts.

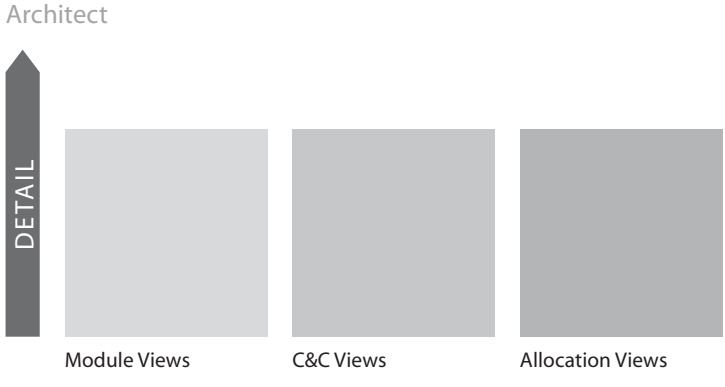

Figure 9.12
A future architect has strong interest in all the architecture documentation.

Table 9.1 Summary of documentation needs

	Module Views					C&C Views	Allocation Views				Other Documentation					
	Decomposition	Uses	Generalization	Layered	Data Model	Various	Deployment	Implementation	Install	Work Assignment	Interface Documentation	Context Diagrams	Mapping Between Views	Variability Guides	Analysis Results	Rationale and Constraints
Project managers	s	s		s			d			d		o				s
Members of development team	d	d	d	d	d	d	s	s	d		d	d	d	d		s
Testers and integrators	d	d	d	d	d	s	s	s	s		d	d	s	d		s
Designers of other systems					s						d	o				
Maintainers	d	d	d	d	d	d	s	s			d	d	d	d		d
Product-line application builders	d	d	s	o	s	s	s	s	s		s	d	s	d		s
Customers							o			o		o			s	
End users						s	s		o						s	
Analysts	d	d	s	d	d	s	d		s		d	d		s	d	s
Infrastructure support personnel	s	s			s	s	d	d	o					s		
New stakeholders	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Current and future architects	d	d	d	d	d	d	d	s	d	s	d	d	d	d	d	d
Key: d = detailed information, s = some details, o = overview information, x = anything																

especially keen to have access to comprehensive and candid rationale and design information.

Table 9.1 summarizes the documentation needs of the stakeholders presented in this section.

9.2 A Method for Choosing the Views

At a minimum, expect to have at least one module view, at least one C&C view, and at least one allocation view in your architecture document.

This section presents a three-step method for choosing the views.

- **Step 1. Build a stakeholder/view table**. For this step, begin by building a table for your project, like that in Table 9.1.

  Enumerate the stakeholders for your project's software architecture documentation down the rows. Your stakeholder list is likely to be different from the one in Table 9.1; however, be as comprehensive as you can. For the columns, enumerate the views that apply to your system. As discussed in the prologue, some views (such as decomposition, uses, and work assignment) apply to every system, while others (various C&C views, the layered view) apply only to systems

designed according to the corresponding styles. That is, you can produce a layered view only if your system is layered; you can produce a client-server view only if you used the client-server style; and so on. For the columns, make sure to include the views or view sketches you already have as a result of your design work so far.

Once you have the rows and columns defined, fill in each cell to describe how much information the stakeholder requires from the view: none, overview only, moderate detail, or high detail. The candidate view list going into step 2 now consists of those views for which some stakeholder has a vested interest.

Decide for which stakeholders you need to provide architecture documentation. Understand what type of information they need and with how much detail. Use this information to decide what views are needed and how to structure them into view packages to support your stakeholders.

PERSPECTIVES

Listening to the Stakeholders

It is asking a lot of an architect to divine the specific needs of each stakeholder, and so it is a very good idea to make the effort to communicate with stakeholders, or people who can speak for those roles. Talk with them about how they will best be served by the documentation you are about to produce. Practitioners of architecture evaluation almost always report that one of the most rewarding side effects of an evaluation exercise comes from assembling an architecture's stakeholders around a table and watching them interact and build consensus among themselves. Architects seldom practice this team-building exercise among their stakeholders, but a savvy architect understands that success or failure of an architecture comes from knowing who the stakeholders are and how their interests can be served. The same holds true for architecture documentation.

Before the architecture documentation effort begins, plan to contact your stakeholders. This will, at the very least, compel you to name them. For a large project in which the documentation is a sizable line item in the budget, it may even be worthwhile to hold a half-day or full-day roundtable workshop. Invite at least one person to speak for each stakeholder role of importance in your project. Begin the workshop by having each stakeholder explain the kind of information he or she will need to carry out his or her assigned tasks. Have a scribe record each stakeholder's answer on a flip chart for all to see. Then present a documentation plan: the set of views you've chosen, the supporting documentation, and the cross-view information you plan to supplement them with. Stakeholders may not necessarily understand what the views mean that you present. Have some examples ready to show how a specific view looks and what kind of information it will show. Finally, perform a cross-check to find requested but missing information and planned but unneeded documentation. Whether you hold a full-blown workshop or talk to your stakeholders informally, the result will be vastly increased buy-in for your documentation efforts and a clearer understanding on everyone's part of what the role of the architecture and its documentation will be.

The information that stakeholders need will not always align nicely with the information the architect was planning to produce. This is why it's so important to ask the stakeholders what they need. But you also have to listen to the answers.

We once ran a workshop with an architecture team to fix some issues that had arisen during an architecture evaluation. The evaluation revealed what we thought was a very well documented architecture. But during the follow-up workshop, the project manager raised the same issue over and over again: "Give me the data that I need to create a reliable project plan. It doesn't matter how long the project lasts, as long as we can reliably meet our delivery promises."

Each time, the chief architect made the same reply: "The information you need is in the architecture document." That statement was technically true, but not particularly helpful. The project manager needed a uses view including a list of module responsibilities. The architect provided it (using UML output from a tool), but the manager wasn't satisfied. "I cannot find the information in the document," he said. "I don't understand what those symbols mean and I don't have time to spend searching the document." The right documentation for him would have been an extraction of the effort/dependency information of modules from the UML model, rendered in plain text, and packaged separately.

Sadly, about a year later the project was canceled. The customer had lost faith in the company's ability to deliver what they promised.

In another case, we saw an architecture documented using the Kruchten 4+1 view set. The system was a typical three-tiered client-server architecture in which the middle tier was a framework that defined the applications as plug-ins. Once you knew that much, it was straightforward to come up with the views. The customer, on the other hand, knew that he was responsible for the system's maintenance after it was delivered. He always made the same demand: "Tell me what and where I have to change when I want to change the content of a specific Web page." He clearly had a "page-oriented view" in mind. This need was not satisfied by any of the views that had been documented using the very reasonable 4+1 approach. He might have been well served by a view showing which plug-ins or parts of plug-ins contributed to producing a Web page.

Keep an open mind when listening to your stakeholders. They'll tell you what they need. Many times it won't be what you were planning to provide, but many times (as in these two cases) what they need is easily produced from the information you already have at hand. And it might make the difference between success and failure.

—F.B. and P.C.

Section 6.6 discusses how to combine views, and which views are often easy and useful to combine.

- **Step 2. Combine views.** The candidate view list from step 1 is likely to yield an impractically large number of views. This step will winnow the list to manageable size.

  Look for views in the table that require only overview, or that serve very few stakeholders. See if the stakeholders

could be equally well served by another view having a stronger constituency.

When combining views it is useful to consider the costs associated with producing and maintaining a view. There are at least two sources of the cost. First is the cost required to generate the view, and second is the cost required to maintain it and keep it consistent with other views.

- **Step 3. Prioritize and stage**. After step 2 you should have the minimum set of views needed to serve your stakeholder community. At this point you need to decide what to do first. How you decide depends on the details specific to your project, but here are some things to consider:
  - Not all the information needs of all the stakeholders must be satisfied to the full extent. Providing 80 percent of the requested information goes a long way, and it might be "good enough" so that the stakeholders can do their job. Check with the stakeholder if a subset of information would be sufficient. They typically prefer a product that is delivered on time and in budget over getting the perfect documentation.
  - You don't have to complete one view before starting another. People can make progress with overview-level information, so a breadth-first approach is often the best.
  - Some stakeholders' interests supersede others. A project manager, or the management of a company with which yours is partnering, often demands attention and information early and often.
  - If your architecture has not yet been validated or evaluated for fitness of purpose, then documentation to support that activity merits high priority.
  - Resist the temptation to relegate rationale documentation to the "do when we have time" category, because rationale is best captured when fresh.

The decomposition view is a particularly helpful view to release early. High-level decompositions are often easy to design, and with this information the project manager can start to build development teams, put training in place, scour the commercial markets or legacy repositories for modules that fill the bill, and start producing budgets and schedules.

Use view packets (discussed in Section 10.1.3) as a mechanism to let you provide overviews or less-detailed documentation to certain stakeholders.

9.3 Example

This section provides an example of applying the procedure in the previous section to select a set of views for a project.

ECS is a system for capturing, storing, distributing, processing, and making available extremely high volumes of data from a constellation of earth-observing satellites. By any measure, ECS is a very large project. Many hundreds of people are involved in its design, development, deployment, sustainment, and use. Here is how the three-step view selection approach might have turned out, had it been applied to the ECS software architecture.

Step 1: Produce a Candidate View List

Stakeholders for the ECS architecture include the usual suspects: the current and future architect, developers, testers and integrators, and maintainers. But the size and complexity of ECS, plus the fact that it is a government system whose development is assigned to a team of contractors, add complicating factors. In this case, there is not one project manager, but several: one for the government and one for each of the contractors. Each contractor organization has its own assigned part of the system to develop and, hence, its own team of developers and testers. ECS relies heavily on commercial off-the-shelf (COTS) components, so the people responsible for selecting COTS candidate components, qualifying them, selecting the winners, and integrating them into the system play a major role. We'll call these stakeholders COTS engineers.

The important quality attributes for ECS begin with performance. Data must be ingested into the system to keep up with the rate at which it floods in from the satellites. Processing the raw data into more sophisticated and complex "data products" must also be done every day to stay ahead of the flow. Finally, requests from the science community for data and data analysis must be handled in a timely fashion. Data integrity, security, and availability round out the important list of quality attributes and make the analysts concerned with these qualities important architecture stakeholders.

ECS is a highly visible and highly funded project that attracts oversight attention. The funding authorities require at least overview insight into the architecture to make sure the money over which they have control is being spent wisely. Finally, the science community using ECS to measure and predict global climate change also requires insight into how the system works, so they can better set their expectations about its capabilities.

At least five of the component-and-connector views discussed in Chapter 4 and four of the module views of Chapter 2 apply to ECS. It is primarily a shared-data system. Its components interact in both client-server and peer-to-peer fashion. Many of those components are communicating processes. And while the system is not actually built using pipes and filters, the pipe-and-filter style is a very useful paradigm to provide an overview to some of the stakeholders. (Information more detailed than the overview will be in a different view, becoming an implementation refinement of the pipe-and-filter view.)

Table 9.2 shows the stakeholders for the ECS architecture documentation and the views useful to each. At this point, the candidate view list contains 12 views.

Table 9.2 ECS stakeholders and architecture documentation they might find most useful

	Module Views				C&C Views					Allocation Views		
Stakeholder	**Decomposition**	**Generalization**	**Uses**	**Layered**	**Pipe-and-filter**	**Shared-data**	**Client-server**	**Peer-to-peer**	**Communicating-processes**	**Deployment**	**Implementation**	**Work assignment**
Current and future architect	d	d	d	d	s	d	d	d	d	d	s	s
Government project manager	d	o	o	s	o	s	o	o	o	s		d
Contractor's project manager	s	o	s	s	o	s	s	s	o	d	s	d
Member of development team	d	d	d	d	o	d	d	d	d	s	s	d
Testers and integrators	s	s	d	s	o	d	d	d	s	s	d	
Maintainers	d	d	d	d	o	d	d	d	d	s	s	s
COTS engineers	d	s		d		d	d	d	s	d		d
Analyst for performance	d	s	d	s	o	d	d	d	d	d		
Analyst for data integrity	s	s	s	d	o	d	d	d	d	d		
Analyst for security	d	s	d	d	o	s	d	d	d	d	o	o
Analyst for availability	d	s	d	d				s	s	d		o
Funding agency	o				o	o				o		
Users in science community	o				o	o				o		
Key: d = detailed information, s = some detail, o = overview												

Step 2: Combine Views

As usual, the C&C views provided good candidates for combination. In the case of ECS, augmenting the shared-data view with other components and connectors that interact in client-server or peer-to-peer fashion allowed those three views to become one. The communicating processes mapped straightforwardly to components in this combined view, allowing it to be folded in as well. The pipe-and-filter view can be discarded; the combined C&C view plus some key behavioral traces showing the data pipeline from satellite to scientist would provide the same intuitive overview to the less detail-oriented stakeholders.

Similarly, some of the module views were combined. Recording uses information as a property of the decomposition view yields a combination of the decomposition and uses views.

It would have been easy to combine the work assignment and implementation views with decomposition as well. However, because of the large size of this project and the number of different development organizations involved, the work assignment view was kept separate. Also, this view was of key

interest to managers and the funding agency, who did not want to see details of the modules. Similarly, because a large number of stakeholders interested in the module decomposition would not be interested in how the modules were allocated to files in the development environment, the implementation view was also kept separate.

After this step, the following views remain:

- Three module views: decomposition/uses, layered, and generalization
- One C&C view: shared-data/client-server/peer-to-peer/communicating-processes
- Three allocation views: deployment, implementation, and work assignment

We entered step 2 with 12 candidate views, too many to be efficiently maintained. Now there are 7.

Step 3: Prioritize

To let the project begin to make progress required putting contracts in place, which in turn required coarse-grained decomposition. Turning out the higher levels of the decomposition and work assignment views received the highest priority, in order to meet these needs.

In ECS, the layering in the architecture was very coarse grained and can be described quickly. Similarly, generalization occurred largely in only one of the three major subsystems, was also coarse grained, and was able to be described quickly. These two views were given next priority.

The combined C&C view and the deployment view followed, nailing down details of runtime interaction only hinted at by the module views. This allowed analysis for performance to begin.

Finally, because the implementation view can be relegated to each contractor's own internal development effort, it received the lowest priority from the point of view of the overall system.

The result was four "full-fledged" views (decomposition, work assignment, the combined C&C view, and deployment) plus three minor ones that are coarse grained or can be deferred.

PERSPECTIVES

How Not to Introduce an Architecture

With John Klein

Several years ago, I was chief architect for a business unit at a large software product development company. My manager, the vice president of engineering for the business unit, approached me one spring day and challenged me to define a single, unified architecture that could be applied to all current products in our portfolio, and that would also support our best guess of future needs. Recognizing the relationship between architecture and organization, he wanted to use this new architecture as the basis for a major reorganization of the 300-person software engineering team, and he wanted to roll out this reorganization at an engineering management meeting planned for late summer. So, my team of five architects had just 90 days to define enough of a system architecture that our VP could build an organization around it.

After a stakeholder analysis, we determined that we needed to produce a number of views. Three of the views are described elsewhere in this book:

- A decomposition view (which we called our "Information Hiding Module Guide)
- A uses view
- A C&C communicating-processes view

In addition to these, we decided to create the following:

- A "technology view," which would be a type of allocation view that would map modules to implementation technology (programming language and middleware)
- A "design model," which was an information model for product configuration and customization data
- An "integration view," to specify the external interfaces of our products
- A "Zoo of Examples," which was "information beyond views" that showed how to use the architecture to create products

Our documentation plan also included an "Architecture Description Overview," which contained the stakeholder analysis, descriptions of each view, and a set of roadmaps to help different stakeholders navigate the documentation.

We began by focusing on the Information Hiding Module Guide. We wanted to meet our VP's need to reorganize based on the architecture; the information hiding decomposition provided a natural basis for structuring the development organization. Also, we recognized that we were developing a software product-line architecture, and we saw the value of structuring the information hiding decomposition to encapsulate the variation points that the architecture would support.

We spent much of our 90-day schedule working on the Module Guide, essentially a detailed decomposition view. We also completed the Architecture Description Overview, which we thought clearly showed our vision for documenting the architecture. Finally, we created a set of "marketecture" diagrams, which we used to communicate the architecture to our executives.

As the deadline approached, the team was feeling proud of our efforts. We felt we had achieved the goal of developing enough architecture to drive the reorganization. Besides, we thought, nobody really expected us to create the entire architecture in just 90 days! We felt we had done enough.

Were we ever wrong!

On the appointed day, we presented the Architecture Description Overview and Module Guide during an "all hands" conference call with the entire software engineering team. There were some polite questions, but we sensed that there was a lot of confusion among the team members. Our organization did not have much exposure to architecture-centric practices. The idea of multiple views of an architecture was understood by some but not all of the staff. Also, the thought that you would incrementally release subsets of the architecture documentation was foreign. Finally, the first view we released showed the information hiding structure, which the staff found difficult to grasp and appreciate.

In retrospect, I realize that the decomposition view is simply a well-organized "parts list" for all of the products in our product line. Of course people were confused—we didn't show which parts go into which products, didn't describe how to assemble the parts into products, and didn't provide a picture of any of the final built products. We confused the development team so much that most of the development staff pointed to the Module Guide as "the architecture document." Here's what we learned.

In most cases, the first release of documentation for an architecture will not be complete and whole. Recommendations in this book, such as "Begin your documentation with a standard outline," will help you and your stakeholders understand the vision and the intended final structure of the documentation, but the first time you release the documentation, people will read the parts you have completed and try to make sense of them.

Stage your architecture design and documentation to deliver coherent subsets to stakeholders. Make each subset internally consistent and complete, and present a chunk of the architecture that will make sense to your stakeholders. Specifically, include some C&C views in early releases so that stakeholders can understand how the system will function at runtime. This is usually a more natural perspective to begin reasoning about the system, as compared to a module view showing design-time structure.

Include in each stage subsets of several views, rather than the approach we took of delivering views sequentially. Provide your stakeholders a complete specification of a subset of the system: a "parts list" (module view), assembly instructions (allocation view and "beyond views"), and a picture of the running

system (C&C view). This allows them to make complete sense of each incremental release of the architecture documentation.

Failure to do these things may damage your credibility, stakeholders may lose interest, and your project may fail, as ours eventually did.

9.4 Summary Checklist

- What views you choose depends on who the important stakeholders are, what budget is on hand, what the schedule is, and what skills are available. It also depends on what structures are present in the architecture.
- You should expect to choose at least one of each of the three different types of views: module, component-and-connector, and allocation.
- You should expect to combine some views to reduce the number of views you have to create, keep consistent, and maintain in your architecture document.
- Prioritize and stage your release of views to serve important project needs early.

9.5 Discussion Questions

1. Suppose that your company has just purchased another company and that you've been given the task of merging a system in your company with a similar system in the purchased company. What views of the other system's architecture would you like to see, and why? Would you ask for the same views for both systems?
2. Some architects speak of a "security view" or documentation of a "security architecture." What do you suppose they mean? What might this consist of?
3. How would you make a cost/benefit argument for the inclusion or exclusion of a particular view in an architecture documentation package? If you could summon up any data you needed to support your case, what data would you want?

9.6 For Further Reading

Around 2001, practitioners at Nokia developed the Rapid7 approach to produce high-quality usable documentation in an Agile environment. The central approach to Rapid7 is to hold a stakeholder workshop at each document delivery milestone

in the project. The workshop is facilitated to produce a document outline that stakeholders will actually use. For more information, see the paper by Kylmäkoski (2003).

A central theme of the book by Hofmeister, Nord, and Soni (2000) is the coordinated use of separate (in their case, four) views to engineer and document software systems. Their treatment provides an excellent foundation for the philosophy behind choosing the views: providing information to stakeholders, and points of engineering leverage to the architect, based on expected needs of the system being built.