

Choosing the Views



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. The equations [of physics that] we seek are the poetry of nature.

—Chen Ning Yang, 1957 Nobel Prize Winner for Physics, quoted in *Bill Moyers: A World of Ideas*, ed. Betty Sue Flowers, Doubleday, 1989, p. 313.

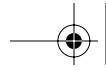
Before a view can be documented, it must be chosen by the architect. And that is the topic of this chapter: how an architect decides on the views to include in the documentation package.

In previous chapters, we explained how to represent all the various aspects of a software architecture. We discussed views that focus on coding aspects and on runtime aspects, and views that document the relationship of the software with its environment: module, C&C, and allocation viewtypes, respectively. Within a single development project, you will not document all the aspects of a software architecture. You will have to make decisions on what to document and to what level of detail. You also have to decide whether you want to define a new style to better support your project needs or to overlay two or more of the mentioned views.

But how many views are enough? How many are too many? And how complete does each view have to be? As a reader, you may be wondering whether we are going to impose an unrealistic documentation obligation on you, one that will produce beautiful exemplary documents that will never be used because the project will have run out of money at implementation time.

The reality is that all projects make cost/benefit trade-offs to pack all the work to be done into the time and the resources allocated for that work. Architecture documentation is no different.





We have tried to explain the benefits of each kind of documentation and to make a compelling case for when you would want to produce it. If you can't afford to produce a particular part of the architecture documentation package, you need to understand what the long-term cost will be for the short-term savings.

Understanding which views to produce at what time and to what level of detail can be answered only in the concrete context of a project. You can determine which views are required, when to create them, and to what level of detail they have to be described in order to make the development project successful only if you know

- What people you will have: which skills are available
- What budget is on hand
- What the schedule is
- Who 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.

9.1 Stakeholders and Their Documentation Needs

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.

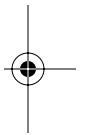
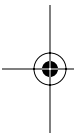
The set of stakeholders will vary, depending on the organization and the project. The list of stakeholders in this section is suggestive but is not intended to be complete.

The documents mentioned for those stakeholders are ones in which they probably are interested, but the need for other documentation will vary from case to case. For instance, a project manager might not be interested in any C&C view. But a product with a Web-based client-server architecture might have a C&C view showing parts of the client-server architecture that are of interest to the project manager. So take the following lists as a starting point and adapt them according to the needs of your project.

A project manager cares about schedule, resource assignments, and perhaps contingency plans to release a subset of the system for business reasons. This person is not interested in the detailed design of any element or the exact interface specifications beyond knowing whether those tasks have been completed. But the manager is interested in the system's overall

FOR MORE INFORMATION

Having decided on a nice set of views, you may want to ask your stakeholders whether the information shown is what they expect. Having a prototypical example helps tremendously. See Chapter 10 for more information about reviewing the documents.



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 but will certainly be interested in monitoring it. As shown in Figure 9.1, a project manager, then, will likely be interested in

- A top-level context diagram: module viewtype
- A decomposition, uses, and/or layered view: module viewtype
- A work assignment view: allocation viewtype
- A deployment view: allocation viewtype
- Overall purpose and constraints

A member of the development team, for whom the architecture provides marching orders, is given constraints on how that person does his or her 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 general idea behind the system. Although that information lies in the realm of requirements rather than architecture, a top-level context diagram 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.
- The elements with which the assigned part interfaces and what those interfaces are.

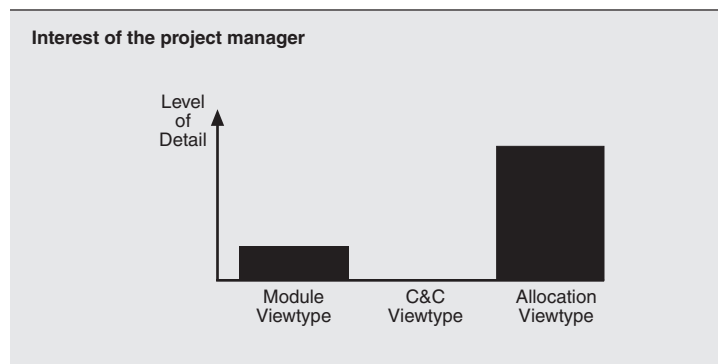


Figure 9.1
A project manager usually creates the work assignments and therefore needs some overview information of the software.



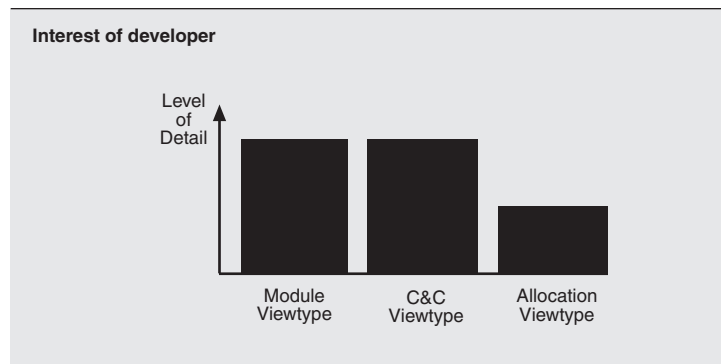
- The code assets the developer can make use of.
- The constraints, such as quality attributes, legacy systems interfaces, and budget, that must be met.

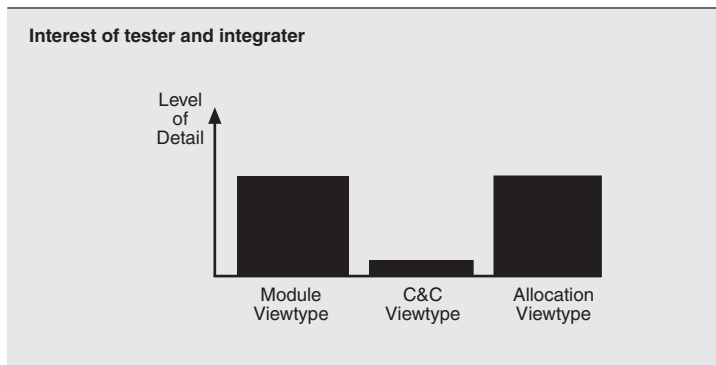
As shown in Figure 9.2, a developer, then, is likely to want to see

- A context diagram containing the module(s) he or she has been assigned: module viewtype
- A decomposition, uses, and layered view: module viewtype
- A view showing the component(s) the developer is working on and how they interact with other components at runtime: C&C viewtype
- A mapping between views, showing the module(s) as components: module viewtype, C&C viewtype
- The interface specification(s) of the developer's element(s) and the interface specifications of those elements with which they interact: module viewtype, C&C viewtype
- The variability guide to implement required variability: module viewtype
- An implementation view to find out where the assets he or she produces must go: allocation viewtype
- A generalization view showing other modules that the developer can use to accomplish his or her work assignment: module viewtype
- A deployment view: allocation viewtype
- The documentation that applies beyond views, including a system overview
- Rationale and constraints

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

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 viewtypes.



**Figure 9.3**

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

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 specification for the element. Integraters 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 integraters, then, are likely to want to see

- A context diagram showing the module(s) to be tested or integrated: module viewtype
- The interface specification(s) and behavior specification(s) of the module(s) and the interface specifications of those elements with which they interact: module viewtype, C&C viewtype
- An implementation view to find out where the assets that build the module are: allocation viewtype
- A deployment view: allocation viewtype

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

- A top-level context diagram: module viewtype and/or C&C viewtype
- Interface specifications for those elements with which their system will interact: module viewtype, C&C viewtype

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



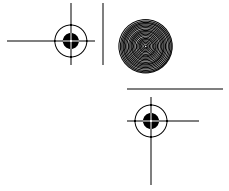
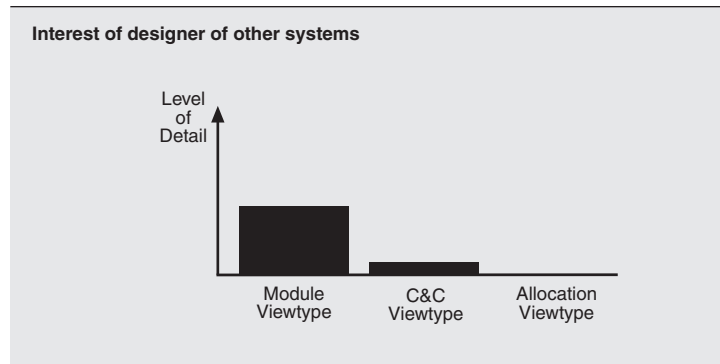


Figure 9.4
Designers of other systems are interested in interface specifications and important system behavior.



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.

DEFINITION

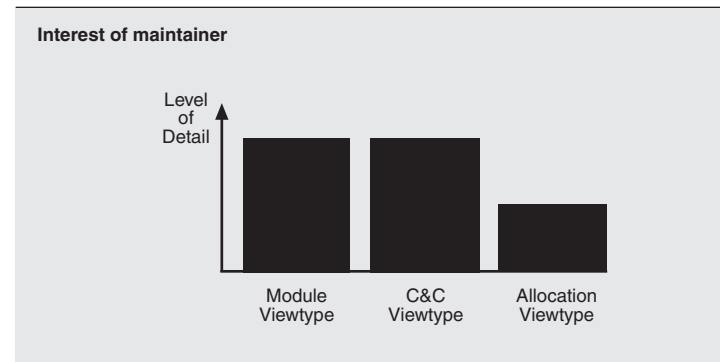
A **software product line** is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of reusable core assets in a prescribed way.

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
- A decomposition view: module viewtype
- A layered view: module viewtype
- Rationale and constraints

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 build-

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



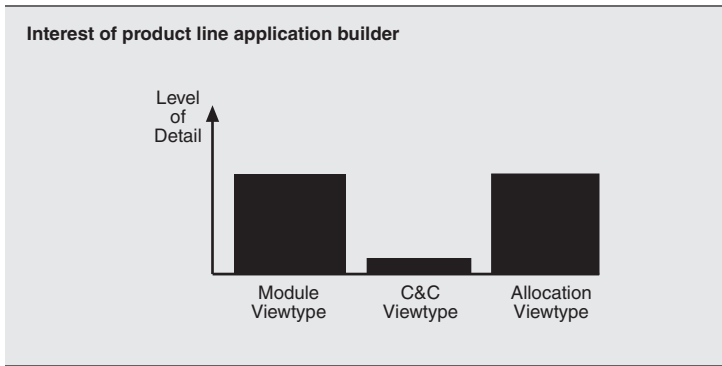


Figure 9.6
This person needs to understand what adaptations to make in order to build new products.

ers 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 integraters do.

As shown in Figure 9.6, a product line application builder, then, is likely to want to see

- The views mentioned for an integrater
- A variability guide: module and/or C&C viewtype

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

- A work assignment view, no doubt filtered to preserve the development organization's confidential information: allocation viewtype

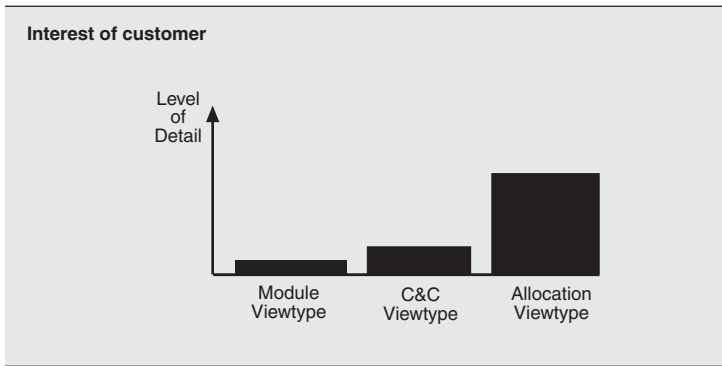


Figure 9.7
A customer is interested mainly in how the software works in the desired environment.





- A deployment view: allocation viewtype
- Analysis results: module and/or C&C viewtype
- A top-level context diagram in one or more C&C views: C&C viewtype

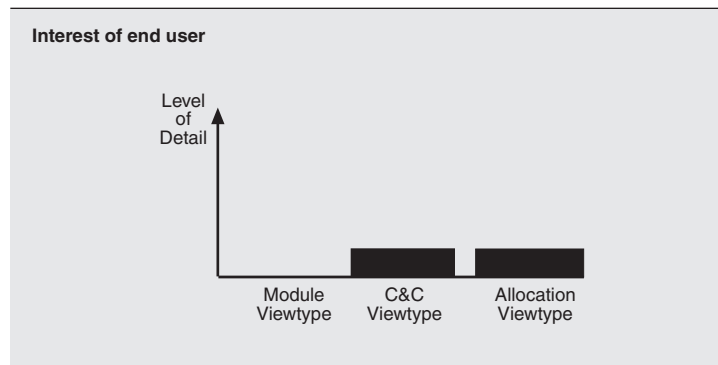
End users do not need to see the architecture, which is, after all, largely invisible to them. But they often gain useful insights about the system, what it does, and how they can use it effectively by examining the architecture. If end users or their representatives review your architecture, you may 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

- A view emphasizing flow of control and transformation of data, to see how inputs are transformed into outputs: C&C viewtype
- A deployment view to understand how functionality is allocated to the platforms with which the users interact: allocation viewtype
- Analysis results that deal with properties of interest to them, such as performance or reliability: module and/or C&C viewtype

Analysts are interested in the ability of the design to meet the system's quality objectives. The architecture serves as the fodder for architectural 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

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.





tools require information about resource consumption, scheduling policies, dependencies, and so forth.

Recently, architecture evaluation and analysis methods have emerged as repeatable, robust, low-cost ways to make sure that an architecture will deliver the required quality attributes before the project commits to implementation based on it. The Architecture Trade-off Analysis Method (ATAM) exemplifies this new breed of methods. ATAM relies on suitable architecture documentation to do its work. Although ATAM does not prescribe specific documents that are required, it does offer general guidelines.

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

- Views of the module viewtype family: module viewtype
- A deployment view: allocation viewtype
- A communicating-processes view: C&C viewtype
- Applicable component-and-connector views: C&C viewtype

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, behavioral 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

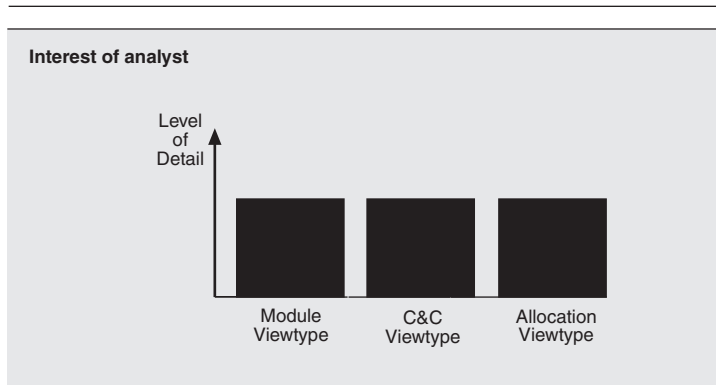


Figure 9.9

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





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 help 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 is used to track where information goes and is exposed; a module decomposition view, 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 in general 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.

Infrastructure support personnel set up and maintain the infrastructure that supports the development and build 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 and/or implementation view. Especially for configuration management, you have to provide a variability guide.

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

- A decomposition view: module viewtype
- A uses view: module viewtype
- An implementation view: allocation viewtype



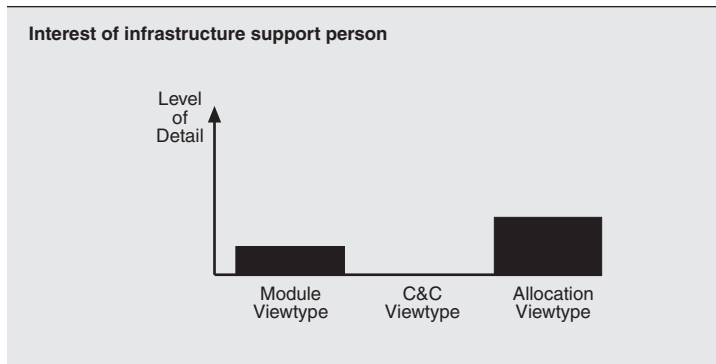


Figure 9.10
Infrastructure support people need to understand the software artifacts produced to provide tool support.

- A variability guide: module viewtype, C&C viewtype
- A deployment view, allocation viewtype

New stakeholders will want to see introductory, background, and broadly scoped information: top-level context diagrams, architectural constraints, overall rationale, and root-level view packets as shown in Figure 9.11. In general, anyone new to the system will want to see the same kind of information as his or her counterparts who are more familiar with the system but will want to see it in less detail.

Future architects are the most avid readers of architectural 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 will be especially keen to have access to comprehensive and candid rationale and design information.

To summarize, the views you choose depend on the views you expect to use. For most nontrivial systems, you should

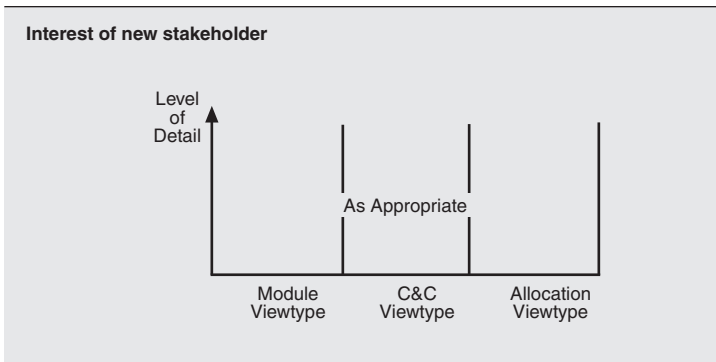


Figure 9.11
New stakeholders need to have the same information as their counterparts.

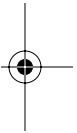
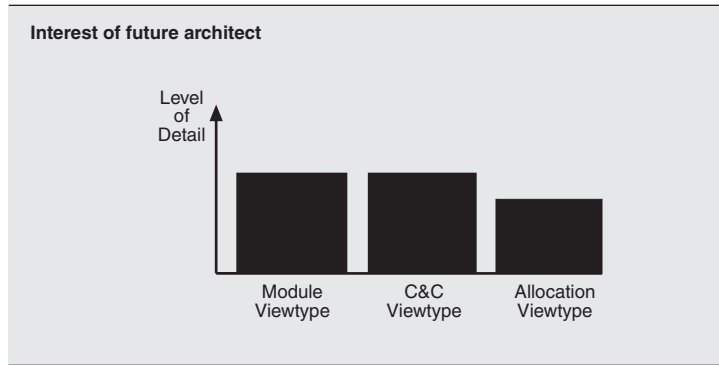




Figure 9.12
A future architect has strong interest in all the architecture documentation.



ADVICE

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

expect to choose at least one view from each of the three viewtypes presented in this book: module, component-and-connector, and allocation. Beyond that, choose specific views based on anticipated uses by your stakeholders. The guidelines presented in this section are rules of thumb with which to begin. Remember that each view you select comes with a benefit but also a cost. You will undoubtedly wish to combine some views or to have one view serve in another's place; for instance, a work assignment view includes the information in a decomposition view, so you may not need both. Table 9.1 summarizes these guidelines.

ADVICE

Ask 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, and talk 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,

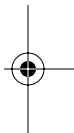


Table 9.1: Summary of documentation needs

Stakeholder	Module Views		C&C Views	Allocation Views			Other						
	Decomposition	Generalization		Layered	Deployment	Implementation	Work Assignment	Interface Specification	Context Diagrams	Mapping between Views	Variability Guides	Analysis Results	Rationale and Constraints
Project manager	s	s	s	d	d	d	o						s
Member of development team	d	d	d	s	s		d						s
Tester and integrator	d	d	s	s	s								s
Designer of other systems													
Maintainer	d	d	d	s	s		d						d
Product line application builder	d	s	o	s	s								s
Customer				o									
End user			s	s									s
Analyst	d	s	s	d			d						d
Infrastructure support personnel	s			s	d								s
New stakeholder	x	x	x	x	x		x						x
Current and future architect	d	d	d	d	s		d						d

Key: d = detailed information, s = some details, o = overview information, x = anything



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 round table 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. 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.

PERSPECTIVES

Architecture Trade-off Analysis Method

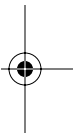
Until recently, there were no reliable methods that would let us subject an architecture to a test to see whether it would deliver the required functionality and, at least as important, the required quality attributes of performance, modifiability, usability, security, availability, and so forth. The architect had to rely on his or her own past experience, styles and patterns in books, or, more likely, folklore. Only when code was developed, whether prototype or production, could the architecture be validated: Code testing served as architecture testing. But by then, changing the architecture was often prohibitively expensive.

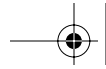
Now, however, architecture evaluation methods have emerged that let us validate an architecture while it is still a paper design, before it has been hardened into code. As architecture evaluation matures to become a standard part of architecture-based development methods, architecture documentation takes on an additional use: serving as the fuel for an evaluation.

One of the most mature evaluation methods is the **Architecture Trade-off Analysis Method (ATAM)**. Under ATAM, a four- or five-person evaluation team is gathered along with a set of stakeholders for the system whose architecture is being evaluated: designers, maintainers, end users, system administrators, and so forth. The analysis phase consists of nine steps.

DEFINITION

ATAM (Architecture Tradeoff Analysis Method) is an architecture evaluation method developed by the Software Engineering Institute.





1. *Present the ATAM.* The evaluation team leader describes the evaluation method to the participants, tries to set their expectations, and answers questions they may have.
2. *Present business drivers.* A project spokesperson, usually the project manager or the system customer, identifies the business goals that are motivating the development effort and hence what will be the primary architectural drivers, such as high availability, time to market, or high security.
3. *Present the architecture.* The architect describes the architecture, focusing on how it addresses the business drivers.
4. *Identify architectural approaches.* ATAM focuses on analyzing an architecture by understanding the architectural styles and approaches that it embodies. Approaches and styles, including those described in this and other books, have known characteristics in terms of how they promote or preclude certain quality attributes. In this step, the team compiles a list by asking the architect to explicitly name any identifiable approaches used but also captures any other approaches mentioned during the architecture presentation in the previous step.
5. *Generate quality attribute utility tree.* The quality factors that comprise system utility—performance, availability, security, modifiability, usability, and so on—are elicited. Then refinements are added. For example, security might be refined to disclose that data confidentiality and data integrity are important. Finally, the refinements are made operational by eliciting detailed scenarios that express the qualities. The utility tree serves to make concrete the quality attribute requirements, forcing the architect and customer representatives to define the relevant quality requirements precisely. Participants prioritize the utility tree scenarios according to how important each scenario is to the system and by how difficult the architect expects it will be to achieve.
6. *Analyze architectural approaches.* At this point, a prioritized set of concrete quality requirements from step 5 and a set of architectural approaches used in the architecture from step 4 exist. Step 6 sizes up how well suited they are to each other. Here, the evaluation team can probe for the architectural approaches that realize the important quality attributes. This is done with an eye to documenting these architectural decisions and identifying their risks, nonrisks, sensitivity points, and trade-offs. The evaluation team probes for sufficient information about each architectural approach to conduct





a rudimentary analysis about the attribute for which the approach is relevant.

7. *Brainstorm and prioritize scenarios.* A larger set of scenarios is elicited from the group of stakeholders. Whereas the utility tree scenarios were generated using quality attributes as the context, here the evaluation team asks the stakeholders to contribute scenarios that speak to stakeholder roles. A maintainer will propose a scenario relevant to the architecture's ability to support maintenance, for example. These new scenarios are then prioritized by means of a facilitated voting process involving the entire stakeholder group.
8. *Analyze architectural approaches.* This step reiterates the activities of step 6, using the highly ranked scenarios from step 7. This analysis may uncover additional architectural approaches, risks, sensitivity points, and trade-off points, which are then documented.
9. *Present results.* Finally, the collected information from the ATAM needs to be summarized and presented back to the stakeholders. This presentation typically takes the form of a verbal report accompanied by slides but might also be accompanied by a more complete written report delivered subsequent to the ATAM. In this presentation, the evaluation leader recapitulates all the information collected in the steps of the method.

ATAM outputs are

- The documentation of architectural approaches
- The quality attribute utility tree, including the scenarios and their prioritization
- The set of attribute-based analysis questions
- The mapping from approaches to achievement of quality attributes
- The risks and nonrisks discovered, and how the risks might undermine the architecture's business drivers
- The sensitivity points and trade-off points found

A savvy architect can and should turn these outputs into part of the project's documentation legacy, which brings us full circle: The effort to prepare documentation to support an evaluation is paid back in full. Not only is the architecture validated or weaknesses discovered in time for repair, but also these outputs can be incorporated into the documentation as a part of the design rationale and analysis results.

—P. C. C.





9.2 Making the Choice

This section presents a procedure for choosing the views, and applies that procedure to two real-world systems.

Here is a simple three-step procedure for choosing the views for your project.

- **Step 1. Produce a candidate view list.** For this step, begin by building a stakeholder/view 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 (C&C views, the layered view) only apply to systems designed according to the corresponding styles.

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 consists of those views for which some stakeholder has a vested interest.

- **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.

First, 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.

Next, look for views that are good candidates to become combined views. For small and medium projects, the work assignment and implementation views are often easily overlaid with the module decomposition view. The decomposition view also pairs well with the layered and uses views. Where different parts of a system exhibit different component-and-connector styles, the corresponding views might be easily overlaid. Finally, the deployment view usually combines well with whatever C&C view shows the components that are allocated to hardware elements—the communicating-processes view, for example.

FOR MORE INFORMATION

Combined views are discussed in Section 6.3.





- **Step 3. Prioritize.** 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:
 - 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 demand 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.

9.3 Two Examples

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

9.3.1 A Small Project: A-7E

The U.S. Navy's A-7E avionics program, used as a source for some of the documentation examples in this book, was one of the first software engineering projects that paid special attention to engineering and documenting its architecture as a set of related but separate views. It was by most standards a small project, with staff size of ten or less for most of its duration. Here are A-7E architecture views under the three-step method outlined above.

Step 1: Produce a Candidate View List

Stakeholders include the current and future architects, the project manager, members of the development team, testers and integrators, and maintainers. The architecture was designed to deliver three primary qualities: modifiability, real-time performance, and the ability to be fielded in incremental subsets. Hence, it is important to analyze the architecture for these qualities, and so by extension, the analysts become stakeholders. The agency responsible for funding the project was also a stakeholder; their interest was in knowing how the architecture leads to a system more maintainable than other avionics programs of the same generation.



Applicable views in the module viewtype include the decomposition, uses, and layered views. Generalization is not employed in the A-7E architecture. The system is structured as a set of cooperating sequential processes, and so in the C&C realm the communicating-processes view applies. The system also has a central data store called a Data Banker; its function is to isolate producers and consumers of data from each other to enhance modifiability. Therefore, the shared-data C&C view also applies. In the allocation viewtype, the work assignment, implementation, and deployment view, all apply as well.

Table 9.2 shows the stakeholders for the A-7E architecture documentation and the views useful to each. At this point, the candidate view list contains eight views.

Table 9.2: A-7E stakeholders and the architecture documentation they might find most useful

Stakeholder	Module Views			C&C Views		Allocation Views		
	Decomposition	Uses	Layered	Communicating-processes	Shared-data	Deployment	Implementation	Work Assignment
Current and future architect	d	d	d	d	d	d	s	s
Project manager	s	s	s		o	d	o	d
Member of development team	d	d	d	d	d	s	s	d
Testers and integraters		d		d	s	s	d	
Maintainer	d	d	d	d	d	s	s	s
Analyst for performance	d	d	d	d	s	d		
Analyst for modifiability	d	d	d	s	s	d	o	o
Analyst for "subsettability"	d	d	d	s	s	d		o
Funding agency	o	o	o					

Key: d = detailed information, s = some details, o = overview information

Step 2: Combine Views

The A-7E's hardware environment features a uniprocessor, and so the deployment view can be dispensed with. By adopting modules (as defined in the module decomposition view) as the basic unit of work assignment and development file structure, the work assignment and implementation views can easily be



combined with the decomposition view. These simple decisions quickly eliminate three of the eight views from the candidate list.

Consultations with stakeholders revealed that the shared-data view was unnecessary. Performance analysts can build performance and schedulability models using the communicating-processes view. Developers, testers, integrators, and maintainers can do their jobs with the module decomposition view¹ (which tells them what the systems' parts were) and the layered view (which tells them what those parts are allowed to use).

After this step, four views remain: decomposition, uses, layered, and communicating-processes.

Step 3: Prioritize

FOR MORE INFORMATION

An excerpt from the A-7E module decomposition view is given in Section 2.1.

Most of the stakeholders are served by the module decomposition view, and so that was chosen as the first undertaking. Because of the tight correspondence between modules and work assignments in this project, the creation of a module is also the commissioning of a work assignment. Before that work assignment can be carried out, the programmers need to know what other software they are allowed to use. Hence, the module decomposition and layered views proceeded hand in hand through successively finer levels of detail.

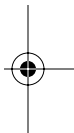
Communicating-processes views are created once the modules are decomposed to a sufficiently fine grain so that the communication and synchronization needs can be decided and then documented.

The uses view has the lowest priority, targeted for documentation only during the module implementation phase. The allowed-to-use information of the layered view constrains programmers but still leaves them with considerable latitude. As a trivial example, a programmer writing a “double(x)” routine is allowed to use multiplication (2x) or addition (x+x) to implement that function. The uses view reflects the programmers' actual choices. Since the uses view is not employed until it is time to begin fielding subsets, it is the last A-7E view to emerge.

9.3.2 A Large Project: ECS

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

1. The module decomposition view was able to partially supplant a C&C view in this case because in the A-7E architecture there is a straightforward, almost one-to-one mapping between modules and components.





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.

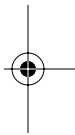
Step 1: Produce a Candidate View List

Stakeholders for the ECS architecture include the usual suspects: the current and future architect, developers, testers and integraters, 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 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 architectural stakeholders.

ECS is a highly visible and highly funded project which 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 require 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 all 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



**FOR MORE INFORMATION**

Implementation refinements are discussed in Section 6.1.2.

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.)

In addition to the five C&C views, all four of the module views discussed in Chapter 2 apply to ECS, as do all three of the allocation views discussed in Chapter 5.

Table 9.3 shows the stakeholders for the ECS architecture documentation and the views useful to each. At this point, the candidate view list contains 12 views.

Step 2: Combine Views

Because of the large size of this project and the number of different development organizations involved, the work assignment view (normally a good candidate for combination) would likely be kept separate. 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 would also be kept separate.

Three of the C&C views would prove good candidates for combination. Augmenting the shared-data view with other components and connectors that interact in client-server or peer-to-peer fashion allows those three views to become one. The pipe-and-filter view can be discarded; the shared-data 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.

Finally, recording uses information as a property of the decomposition view yields a combination of the decomposition and uses views.

After this step, eight views remain:

- In the module viewtype, decomposition, layered, and generalization
- In the C&C viewtype, shared-data and communicating-processes
- In the allocation viewtype, deployment, implementation, and work assignment.

Step 3: Prioritize

To let the project begin to make progress requires putting contracts in place, which in turn requires coarse-grained decomposition. Thus, the higher levels of the decomposition and work assignment views would likely receive the highest priority.

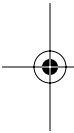


Table 9.3: ECS stakeholders and architecture documentation they might find most useful.

Stakeholder	Module Views				C&C Views					Allocation Views		
	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
Contractors' project managers	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 integraters	s	s	d	s	o	d	d	d	s	s	d	
Maintainer	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 details, o = overview information

In ECS, the layering in the architecture is very coarse-grained and can be quickly described. Similarly, generalization occurs largely in only one of the three major subsystems, is also coarse-grained, and can also be quickly described. Hence, these two views might be given next priority because they can be quickly dispatched.



The shared-data, communicating-processes, and deployment views would follow, nailing down details of runtime interaction only hinted at by the module-based views. During this phase, the architect can see if the communicating processes map straightforwardly to components in the shared-data view, in which case those two views could also be combined.

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

The result is four “full-fledged” views (decomposition, work assignment, shared-data/communicating-processes, and deployment), and three minor ones that stop at high levels or can be deferred.

9.4 Summary Checklist

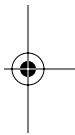
ADVICE

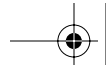
Think twice if you need a new style. Perhaps an overlay is good enough. If existing styles are not sufficient for your purposes, define a new style and produce the necessary style guide for it.

- 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.
- You should expect to choose at least one view from each of the three viewtypes: module, component-and-connector, and allocation.
- You will probably wish to combine some views or to have one view serve in another's place.

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. If you're given the resources to produce whatever architecture documentation you need, what views would you call for, 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

A central theme of [Hofmeister+ 00] is the coordinated use of separate (in their case, four) views to engineer and document software-intensive 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.

