7 The Architecture Definition Process

Architecture definition starts early in the project lifecycle, when scope and requirements are often still unclear and the current view of the system may differ substantially from what is eventually built. For this reason, architecture definition tends to be a more fluid activity than later tasks such as designing, building, and testing, when the problem you are solving is better understood. When you start, you don't fully know the size and extent of your system, where the complexity is, what the most significant risks are, or where you will encounter conflict among your stakeholders.

In this chapter, we outline a simple process of architecture definition that applies (in some way) to most software development projects, irrespective of the development approaches used. You can use the process we describe with most forms of the software development lifecycle—from the very structured and formal to those founded on iterative or agile principles.

The material in this chapter will help you plan your own architecture definition work and align your plans with those for the other parts of the development. Of course, the way you do this will vary according to the needs of your project, the method you are following, the time available, and the skills of you and your team. You will be most successful if you use this chapter as a framework or starting point for developing your own personalized architecture definition process.

Guiding Principles

For an architecture definition process to be successful, it must adhere to the following principles.

- It must be driven by *stakeholder concerns*, as we discussed in Part I. As we will see, stakeholder concerns are the core—but by no means the only—inputs to the process. Furthermore, the process must *balance* these concerns effectively where they conflict or have incompatible implications.

- It must encourage the effective *communication* of architectural decisions, principles, and the solution itself to stakeholders.
- It must ensure, on an ongoing basis, that the architectural decisions and principles are *adhered to* throughout the lifecycle up to the final deployment.
- It must (as much as possible, given the fluid nature of architecture definition) be *structured*. In other words, it must comprise a series of one or more steps or tasks, with a clear definition of the objectives, inputs, and outputs of each step. Typically, the outputs from one step are the inputs to subsequent steps.
- It must be *pragmatic*—that is, it must consider real-world issues such as lack of time or money, shortage of specific technical skills, unclear or changing requirements, the existing context, and political considerations.
- It must be *flexible* so that it can be tailored to particular circumstances. (This is sometimes referred to as a *toolkit* or *framework* approach, with the idea that you use those elements of the toolkit you need and ignore the rest.)
- It must be *technology-neutral*. That is, it must not mandate that the solution is based around any specific technology, architectural pattern, or development style, nor should it dictate any particular modeling, diagramming, or documentation style.
- It must *integrate* with the chosen software development lifecycle.
- It must align with good *software engineering practices* and *quality management standards* (such as ISO 9001) so that it can integrate easily with existing approaches.

Having set out our ground rules, let's consider the context in which architecture definition operates, starting with where we want to end—its outcomes.

PROCESS OUTCOMES

Clearly, the main goal of architecture definition is to develop a sound architecture and to manage the production and maintenance of all of the elements of an AD that captures it. However, there are some desirable secondary outcomes or consequences of architecture definition, such as the following.

- *Clarification of requirements and of other inputs to the process*: Your stakeholders may not be absolutely clear what they want, and it may take some time to pin them down.
- *Management of stakeholders' expectations*: Your architecture will inevitably need to make compromises around your stakeholder concerns. It is

far better to make these compromises visible and clearly understood early in the life of the project than to let them emerge later.

- *Identification and evaluation of architectural options*: There is rarely just one solution to a problem. When there are several potential solutions, your analysis will reveal the strengths and weaknesses of each and justify the chosen solution.
- *Description of architectural acceptance criteria* (indirectly): Architecture definition should lead to a clear understanding of the conditions that must be met before the stakeholders will accept the architecture as conforming to their requirements (e.g., it must provide a particular function, achieve certain response times, or restart in less than a given time period).
- *Creation of a set of design inputs* (ideally): Such information as guidance for and constraints on the software design process will help ensure the integrity of your architecture.

Having defined the goals that our architecture definition process must meet, let us continue by considering the context that the process must work within.

The Process Context

Architecture forms the bridge between requirements and design, performing the tradeoffs necessary to satisfy the demands of both. In process terms, this means that architecture definition sits between requirements analysis and software construction (design, code, and test). A good model for the interaction between requirements, architecture, and construction is the Three Peaks model (see Figure 7–1), an extension of Bashar Nuseibeh's Twin Peaks model.

The three triangles (the peaks) in the diagram represent the major software development activities of requirements analysis, architecture definition, and construction; the widening of the shapes at their bases represent an increasing level of detail as time goes on while the system is developed. The curling arrows show how requirements and architecture as well as architecture and construction are intertwined at a progressively increasing level of detail during system development. Although the specification, architecture, and implementation of the system are quite distinct, as the Three Peaks model illustrates, they have profound effects on each other and so cannot be considered in isolation.

The following key relationships exist between software architecture and the requirements and construction activities of the software lifecycle.

- Requirements analysis provides the context for architecture definition by defining the scope and the system's desired functionality and quality properties.

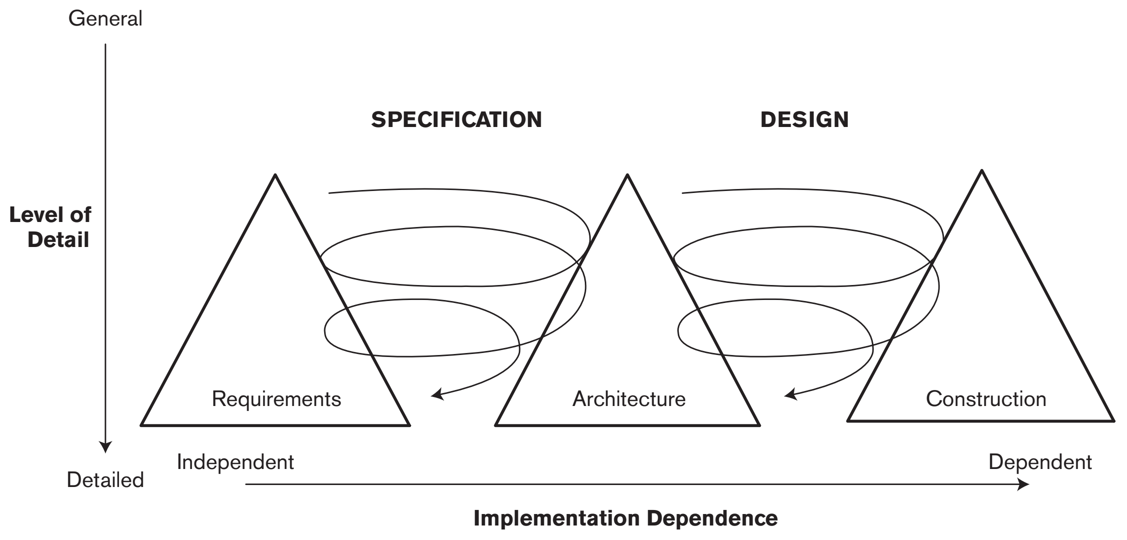

FIGURE 7–1 ARCHITECTURE DEFINITION CONTEXT—THE THREE PEAKS MODEL (BASED ON NUSEIBEH [NUSE01])

- Architecture definition often reveals inconsistent and missing requirements and also helps stakeholders understand the relative costs and complexities of meeting their concerns. This feeds back into requirements analysis to clarify and add requirements and to prioritize these when tradeoffs are made between stakeholders' aspirations and what can be achieved given time and budget constraints.
- When architecture definition has resulted in an architecture that appears to meet an acceptable set of user requirements, the construction of the system can be planned.
- Construction is often organized as a set of incremental deliveries, each of which aims to provide a useful set of functions and to leave the system in a stable, usable state (albeit an incomplete one). The construction of each increment provides further feedback to architecture definition, validating or indicating problems with the architecture as currently specified; hence, there is architecture definition activity throughout the lifecycle.

Requirements analysis, architecture definition, and software construction have a strong, interconnected set of relationships. Requirements analysis provides an initial context for architecture definition but is then itself affected by architecture definition as requirements are understood more fully. In turn, architecture definition drives the implementation process, but each piece of construction performed provides feedback about the effectiveness and utility of the architecture in use.

SUPPORTING ACTIVITIES

Our architecture definition process assumes that the following will be available to you and accepted by the sponsor and other stakeholders before you start:

- A definition of the system's baseline scope and context
- A definition of key stakeholder concerns

Our process also assumes that a broad range of stakeholders have been identified and engaged.

In reality, it is rare for the baseline scope and concerns to be captured to an appropriate level of detail at this early stage, and it is unlikely that any stakeholders (other than perhaps developers and occasionally users) will have been brought on board and engaged in the process. It can be a big challenge to discover and consolidate these inputs and then to gain agreement on them from an engaged stakeholder community before you can even think about a solution. We present a number of techniques for doing this in Chapters 8 and 9.

At the other end of the process, once you have an AD, you will often want to deliver a skeleton system implementation as the first development increment. Such an implementation can be very valuable because it offers a practical validation of the architecture, acts as a credibility test for the system's stakeholders, and provides a framework in which the development team can work.

The slightly extended UML activity diagram in Figure 7–2 shows how architecture definition relates to the following supporting activities. The square boxes with underlined names represent key inputs to and outputs from the process.

- Define the initial scope and context.
- Engage the stakeholders.
- Capture first-cut concerns.
- Define the architecture.
- Create a skeleton system.

Having defined the initial scope and context for your system with the acquiring stakeholders, you can then identify and engage all of the other stakeholders who have an interest in the system. Capturing their concerns provides a primary input, along with the scope and context, to architecture definition. (As we will see, both the scope and the concerns as defined at this point may change, subject to stakeholder agreement, during architecture definition.)

Architecture definition results in an AD and normally a set of guidelines and constraints to guide the system construction. Once you have an AD, you can (if you have the time and resources) create a skeleton system that will act as an evolvable prototype of the system you want to build.

We describe each of these supporting activities in Tables 7–1 through 7–5.

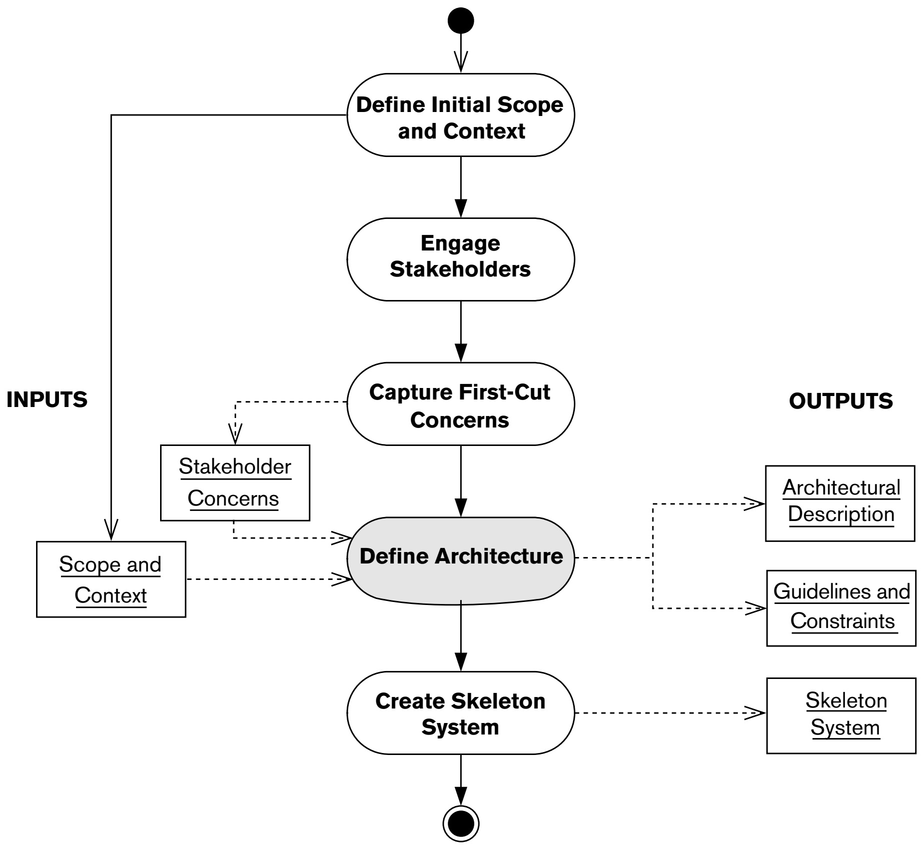

FIGURE 7–2 ACTIVITIES SUPPORTING ARCHITECTURE DEFINITION

TABLE 7–1 DEFINE THE INITIAL SCOPE AND CONTEXT

Aims	To clearly define the boundaries of the system's behavior and responsibilities and the operational and organizational context within which the system exists.
Inputs	Acquirer needs and vision; organizational strategy; enterprise IT architecture.
Outputs	Initial statements of the goals of the system and what is included and excluded from its responsibilities; initial system context definition.
Comments	This step is primarily a process of understanding strategic and organizational objectives and how the system helps meet them, along with some analysis to understand which other systems need to interact with this one. We talk about this activity in Chapter 8. Note that the scope as defined here may change (subject to stakeholder agreement) during architecture definition.

TABLE 7–2 ENGAGE THE STAKEHOLDERS

Aims	To identify the system's important stakeholders and to create a working relationship with them.
Inputs	Scope and context; organizational structure.
Outputs	Definition of each of the stakeholder groups, with one or more named, engaged people who will represent the group.
Comments	This step involves understanding the organizational context you are working in and identifying the key people who will be affected by the system. You can then start to get to know their representatives and begin building a working relationship with them. We talk more about this activity in Chapter 9.

TABLE 7–3 CAPTURE FIRST-CUT CONCERNS

Aims	To clearly understand the concerns that each stakeholder group has about the system and the priorities they place on each concern.
Inputs	Stakeholder list; scope and context.
Outputs	Initial definition of a set of prioritized concerns for each stakeholder group.
Comments	This step often starts with your initial stakeholder meetings. It normally involves a series of presentations and meetings with representatives of each stakeholder group that allow you to explain what you aim to achieve and allow the stakeholders to explain their interests in the system. We talk more about this activity in Chapter 9. Note that the concerns as defined here may change (subject to stakeholder agreement) during architecture definition.

TABLE 7–4 DEFINE THE ARCHITECTURE

Aims	To create the AD for the system.
Inputs	Stakeholder list; scope and context.
Outputs	AD; guidelines and constraints.
Comments	We describe this step in detail in the Architecture Definition Activities section of this chapter.

TABLE 7–5 CREATE A SKELETON SYSTEM

Aims	Optional step to create a working (albeit limited) implementation of your architecture that can evolve into a delivered system during the system construction phase of the lifecycle.
Inputs	AD; associated guidelines and constraints.
Outputs	A limited working system that illustrates that the system can address at least one of your scenarios.
Comments	If you have the time and resources available to allow the creation of a skeleton system, it forms an effective bridge between architecture definition and software construction. This step allows the architect and the developers to build a working system that can execute at least a simple functional scenario the system is meant to address. The skeleton system acts as a validation of your architecture (and an important proof point for many stakeholders) as well as a framework for the software construction phase.

ARCHITECTURE DEFINITION ACTIVITIES

Your biggest difficulty as an architect is the amount of uncertainty and change you face as you bring your stakeholders together. Although you will be working from an agreed-upon scope—or if not, you will produce one as a matter of urgency—this is likely to change as the implications of including or excluding certain features emerge and as the stakeholders better understand the significance of what they are requesting. Functional and quality property requirements are also likely to evolve, perhaps significantly.

For this reason, our architecture definition process is an iterative one. In other words, you need to repeat the main steps several times before you produce a finished AD. Of course, for a small or simple architecture, you may produce the completed AD after the first iteration, but for anything complex, unfamiliar, or contentious, a single iteration is unlikely to suffice. Also, the architecture will keep evolving as the system is developed, so you will return to this cycle of activities throughout the project.

The UML activity diagram in Figure 7–3 illustrates our process, which involves the following steps.

1. Consolidate the inputs.
2. Identify scenarios.
3. Identify the relevant architectural styles.
4. Produce a candidate architecture.
5. Explore the architectural options.
6. Evaluate the architecture with the stakeholders.

7A. Rework the architecture.

7B. Revisit the requirements.

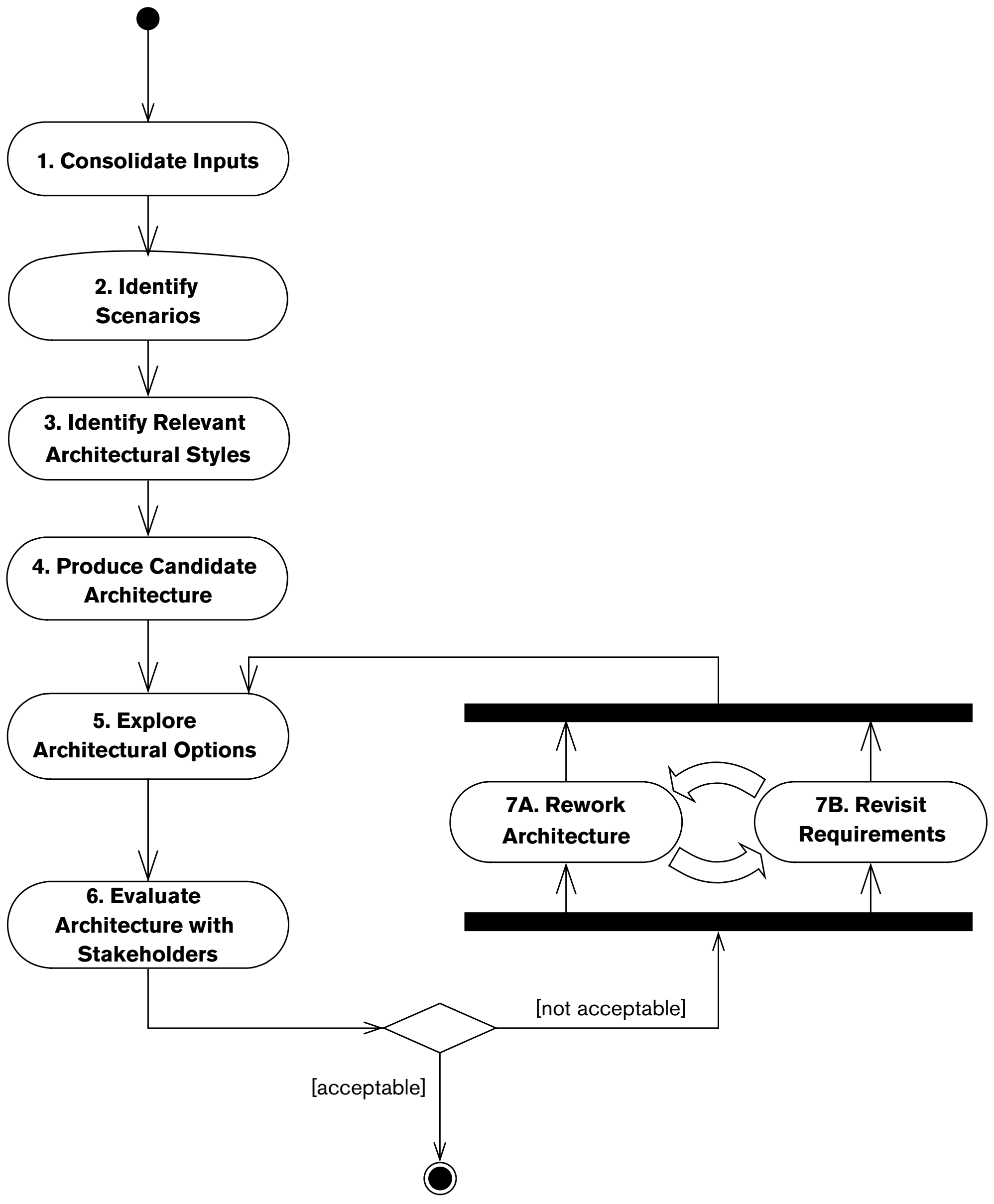
1. Consolidate Inputs
2. Identify Scenarios
3. Identify Relevant Architectural Styles
4. Produce Candidate Architecture
5. Explore Architectural Options
6. Evaluate Architecture with Stakeholders
7A. Rework Architecture
7B. Revisit Requirements
[not acceptable]
[acceptable]

FIGURE 7–3 DETAILS OF ARCHITECTURE DEFINITION

Although it is obviously a simplification of the reality of architecture definition, you may find our model useful in discussions with management, colleagues, and stakeholders.

The curved arrows between steps 7A and 7B in the figure indicate that these steps are not done in isolation: There is often a heavy interaction between them as reworking the architecture may suggest changes to requirements and vice versa. For example, simplifying the concurrency model may necessitate changes to the order in which the system performs some tasks. Of course, all such changes should be reviewed and ratified with stakeholders.

The individual steps in this process are described in Tables 7–6 through 7–13.

TABLE 7–6 STEP 1: CONSOLIDATE THE INPUTS

Aims	To understand, validate, and refine the initial inputs.
Inputs	Raw process inputs (scope and context definition, stakeholder concerns).
Outputs	Consolidated inputs, with major inconsistencies removed, open questions answered, and (at a minimum) areas requiring further exploration identified.
Activities	Take the raw process inputs, resolve inconsistencies between them, answer open questions, and delve deeper where necessary, to produce a solid baseline.
Comments	It is rare for you to be provided with a consistent, accurate, and agreed-upon set of process inputs. During this step you take the information available, fill in gaps, resolve inconsistencies, and obtain formal agreement from the key stakeholders.

TABLE 7–7 STEP 2: IDENTIFY SCENARIOS

Aims	To identify a set of scenarios that illustrates the system's most important requirements.
Inputs	Consolidated inputs (as currently defined).
Outputs	Architectural scenarios.
Activities	Produce a set of scenarios that characterize the most important attributes required of the architecture and can be used to evaluate how well a proposed architecture will meet the underlying functional and quality property requirements.
Comments	A scenario is a description of a situation that the system is likely to encounter, which allows assessment of the effectiveness of the architecture in that situation. Scenarios can be identified for required functional behavior ("How does the system do X?") and for desired quality properties ("How does the system cope with load Y?" or "How can the architecture support change Z?"). We explain how to approach this step in Chapter 10.

TABLE 7–8 STEP 3: IDENTIFY THE RELEVANT ARCHITECTURAL STYLES

Aims	To identify one or more proven architectural styles that could be used as a basis for the overall organization of the system.
Inputs	Consolidated inputs (as currently defined); architectural scenarios.
Outputs	Architectural styles to consider as the basis for the system's main architectural structures.
Activities	Review existing catalogs of architectural styles, and consider system organizations that have worked well for you before. Identify those that appear to be relevant to the architecture as you currently understand it.
Comments	Using an architectural style is a way to reuse architectural knowledge that has proved effective in previous situations. This can help you arrive at a suitable system organization without having to design it from scratch and so reduces the risks involved in using new, unproven ideas. We talk more about using architectural styles in Chapter 11.

TABLE 7–9 STEP 4: PRODUCE A CANDIDATE ARCHITECTURE

Aims	To create a first-cut architecture for the system that reflects its primary architectural concerns and that can act as a basis for further architectural evaluation and refinement.
Inputs	Consolidated inputs (as currently defined); relevant architectural styles, viewpoints, and perspectives.
Outputs	Draft architectural views.
Activities	Produce an initial set of architectural views to define your initial architectural ideas, using guidance from the viewpoints and perspectives and any relevant architectural styles.
Comments	Although they may contain gaps, inconsistencies, or errors, the draft views form a starting point for the more detailed architecture work later.

TABLE 7–10 STEP 5. EXPLORE THE ARCHITECTURAL OPTIONS

Aims	To explore the various architectural possibilities for the system and make the key architectural decisions to choose among them.
Inputs	Consolidated inputs; draft architectural views; architectural scenarios, viewpoints, and perspectives.
Outputs	More detailed or accurate architectural views for some parts of the architecture.

Continued on next page

TABLE 7–10 STEP 5. EXPLORE THE ARCHITECTURAL OPTIONS *(CONTINUED)*

Activities	Apply scenarios to the draft models to demonstrate that they are workable, that they meet requirements, and that there are no hidden problems. Take any areas of risk, concern, or uncertainty that are revealed and further explore the requirements, problems, and issues. Where there is more than one possible solution, evaluate the strengths and weaknesses of each (refer to Chapter 14 for guidance on how to do this) and select the best one.
Comments	The aim of this step is to fill in gaps, remove inconsistencies in the models, and provide extra detail where needed.

TABLE 7–11 STEP 6: EVALUATE THE ARCHITECTURE WITH THE STAKEHOLDERS

Aims	To work through an evaluation of the architecture with your key stakeholders, capture any problems or deficiencies, and gain the stakeholders' acceptance of the architecture.
Inputs	Consolidated inputs; architectural views and perspective outputs.
Outputs	Architectural review comments.
Activities	Evaluate your architecture with a representative collection of stakeholders. Capture and agree on any improvements to or comments on the models.
Comments	Although each group of stakeholders will have different interests, the overall objective is to confirm that stakeholder concerns are met and that the architecture is of good quality. You may have to work hard to obtain consensus if the concerns of different stakeholders conflict with one another. We talk about this activity in Chapter 14.

TABLE 7–12 STEP 7A: REWORK THE ARCHITECTURE

Aims	To address any concerns that have emerged during the evaluation task.
Inputs	Architectural views; architectural review comments; relevant architectural styles, viewpoints, and perspectives.
Outputs	Reworked architectural views; areas for further investigation (optional).
Activities	Take the results of the architectural evaluation and address them in order to produce an architecture that better meets its objectives. This step normally involves functional analysis, the use of viewpoints and perspectives, and prototyping.
Comments	This step is done concurrently and often quite collaboratively with step 7B. The two steps feed back into step 5.

TABLE 7–13 STEP 7B: REVISIT THE REQUIREMENTS

Aims	To consider any changes to the system's original requirements that may have to be made in light of architectural evaluation.
Inputs	Architectural views; architectural review comments.
Outputs	Revised requirements (if any).
Activities	The work done so far may reveal inadequacies or inconsistencies in requirements or requirements that are infeasible or expensive to implement. In this case, you may need to revisit these requirements with stakeholders and obtain their agreement to the necessary revisions.
Comments	This step is done concurrently and often quite collaboratively with step 7A. The two steps feed back into step 5.

PROCESS EXIT CRITERIA

In the ideal world, you would continue with architecture definition until the architecture was perfect and perfectly documented in the AD. However, because architecture definition necessarily takes a high-level view of the system to be built, it's hard in practice to be confident that your architecture is the right one for the problems you are trying to solve.

The best criterion available for determining that you have finished your architecture and are ready to move into the first construction iteration is that there are no significant comments from your architectural evaluation activities, and no significant changes are therefore required to the architecture. In other words, the stakeholders all agree that the AD—or, rather, the system it documents—adequately addresses their concerns.

PRINCIPLE Architecture definition can be considered complete once the formal review of the architectural description by stakeholders results in no significant comments or actions.

In practice, you are unlikely to achieve complete agreement, particularly when the stakeholder group is large or diverse or when requirements are complex. Usually you will finish architecture definition when most of the concerns of the more important stakeholders have been addressed and when you feel confident that the project can proceed with an acceptable level of risk. In some cases, however, you will find that some important stakeholder concerns are still outstanding when the allocated time for architecture definition ends. This is an unfortunate situation, but when time is limited, it may be unavoidable. It is essential in such cases that you prioritize your work to focus on the riskiest or most contentious areas so that at least these are resolved before you move into construction. In this way you can be relatively confident that your architecture is adequate to meet its most important challenges.

Don't forget to include yourself in the list of AD reviewers. Even if your stakeholders are happy with the architecture, if you are not, you should not consider it complete. You may have knowledge or understanding of the system that they don't, and it is your responsibility to ensure this is reflected in the architecture.

STRATEGY Include yourself in the reviewers of the architectural description, and do not finish initial architecture definition until you are satisfied that there are no significant issues with the architecture.

It is not hard to find yourself descending into a repeating cycle of further and deeper refinement and enlargement of your AD, with the result that you never build the system or that development goes ahead without you. This worst possible outcome is to be strongly resisted. In our experience, on all but the largest projects, you should aim to complete the production of the AD in one to three months.

STRATEGY Aim to produce an architectural description that is good enough to meet the needs of its users, rather than to strive for a perfect version that will take significantly more resources to complete without providing any real benefit to the system's stakeholders.

Of course, completion of the AD does not mean that you are no longer working as an architect. You'll be involved throughout, advising, leading, overseeing, resolving problems, revising the architecture as new knowledge emerges, and so on. This means that once the AD is baselined and placed under configuration control, it should continue to be a living document, kept up-to-date throughout the construction steps and into deployment.

ARCHITECTURE DEFINITION IN THE SOFTWARE DEVELOPMENT LIFECYCLE

Architecture definition does not replace the normal software development lifecycle but should be thought of as an integral part of it. In this section, we discuss how architecture definition fits into the common approaches to designing and building systems.

Waterfall Approaches

In the classic waterfall model, software development is viewed as a linear sequence of tasks, with each task using the outputs of the previous one as its inputs, and feeding into the next task in turn, as shown in Figure 7–4. So, for example, the functional specification provides the inputs to the design stage, the design provides the inputs to the build and unit test stage, and so on. When changes are required to the system, these feed backward to the preceding stage and possibly further up the waterfall. Although somewhat discredited as a development approach for large systems, due to its late feedback and inflexibility, the waterfall approach is still a useful and widely used model for the fundamental processes needed in a software development project.

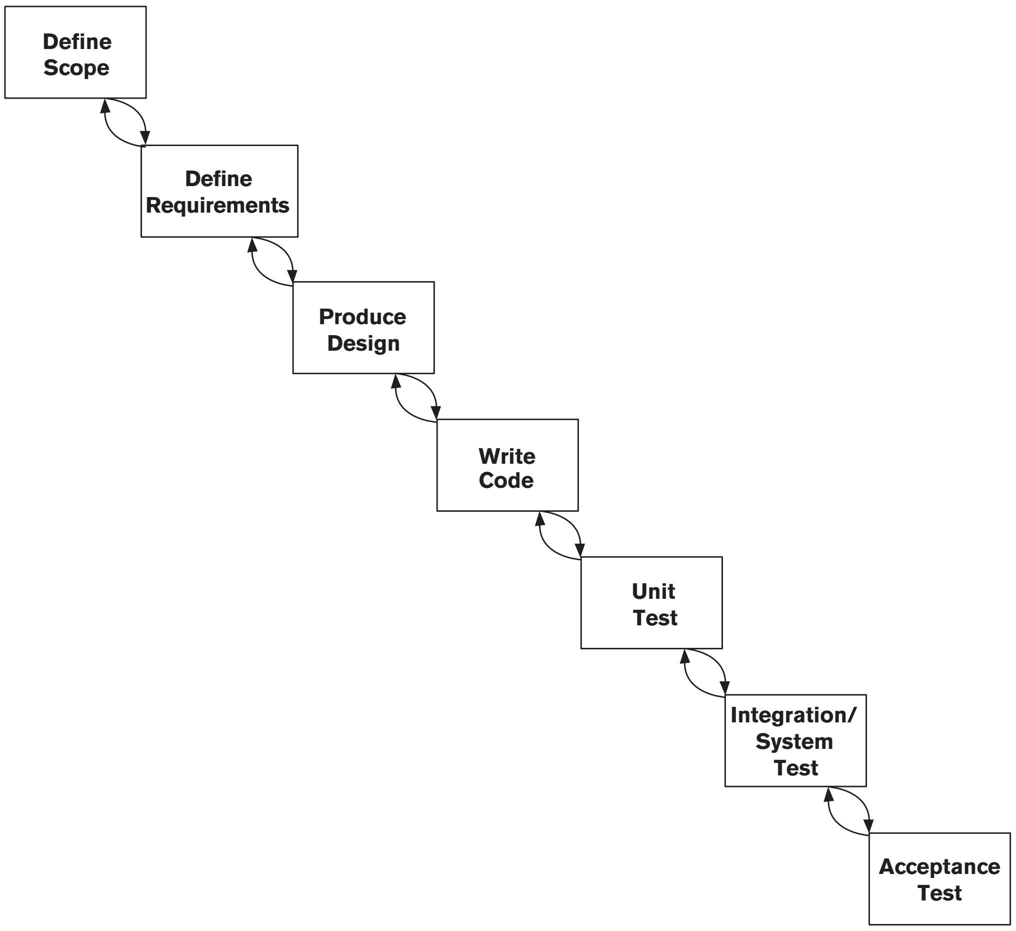

FIGURE 7–4 THE WATERFALL MODEL OF DEVELOPMENT

Architecture definition is easy to integrate with such a linear approach: It is usually viewed as a separate task early in the lifecycle (before, after, or sometimes alongside requirements definition).

Iterative Approaches

The motivation behind iterative approaches (such as Feature Driven Development and the Rational Unified Process) is to reduce risk by means of early delivery of partial functionality, as shown in Figure 7–5. Each iteration usually focuses on one area that presents significant risk because its requirements are unclear, for example, or because it is a complex or leading-edge element of the system. In most iterative approaches, the individual iterations are run as accelerated development projects in their own right, broken down into structured tasks with defined inputs and deliverables.

Typically, architecture definition would form part of the analysis phase, or it could alternatively run alongside the other tasks as an ongoing activity. Our architecture definition process is itself iterative, which dovetails quite nicely with such methods. (For the Rational Unified Process in particular, our approach fits well in its Elaboration phase.)

Agile Methods

Agile methods are a relatively recent development in software engineering. Agile methods are lightweight methods that focus on the rapid and continuous delivery of software to end users, encourage constant interaction between the customer and the software developers, and attempt to minimize the management

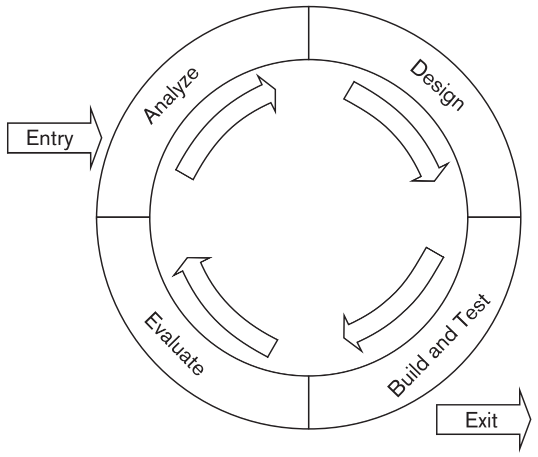

FIGURE 7–5 ITERATIVE DEVELOPMENT

overhead of the development process (in particular, a dramatic reduction in the amount of development documentation produced). Two of the best-known agile methods are Extreme Programming (XP) and Scrum.

At the time of writing, it would be fair to say that there is a fair amount of tension between software developers committed to agile methods (particularly XP) and software architects who expect to develop comprehensive ADs before embarking on detailed design and software development. Agile developers think of the architects as "bureaucratic document producers," whereas the architects think of the developers as "undisciplined hackers intent on development anarchy"; reality is probably somewhere in the middle.

Agile methods focus on delivering useful software quickly and constantly validating what is being built, both of which are positive results. On the other hand, embarking on large-scale development without understanding the problem thoroughly or having any idea where you're headed can lead to a large amount of rework that could have been avoided with more attention to early design.

From our relatively limited experience of agile methods, we suggest that they can be a good way to drive the development of each increment as you develop your system. The technical practices they encourage help create simple and reliable software that can be changed relatively easily. This can only be a good thing. As architects and developers ourselves, we suggest that architecture fits in above the construction increments, sets the context for the iterations, and helps maintain coherence and technical integrity across the system.

It's probably safe to say that the question of the relationship between software architecture and agile development will not be settled to everyone's satisfaction in the immediate future.

SUMMARY

In this chapter, we outlined a simple process of architecture definition, applicable to most software development projects, which you can use to help formulate your plans and schedules.

We started the chapter by defining the principles that our process should adhere to. It should be stakeholder-driven (of course), structured, pragmatic, flexible, and technology-neutral. It must also integrate with your existing software development lifecycle and with established best practices of software engineering.

We explained the context of the process and defined its outcomes; these obviously include specifying the architecture and producing the AD but often extend into other areas, such as better understanding of the problem being solved and management of stakeholder expectations.

We defined the essential inputs to the process: a baseline definition and stakeholder concerns. The baseline definition, which includes scope and context, must be determined and accepted at the start of the process. Stakeholder concerns, which include high-level functional and technical requirements and architectural constraints, are more likely to be discovered, elaborated, and refined as your analysis progresses.

We defined a simple, iterative process of architecture definition based on drawing up a set of architectural models, exploring some of their features in more detail, reviewing these with stakeholders, and reworking the models. Finally, we explained how this process aligns with existing development lifecycles such as the waterfall, iterative, and agile models.

FURTHER READING

Most books on software architecture include a description of some form of architecture definition process. Three books listed in the Bibliography include representative examples of different processes [GARL03, BASS03, BOSC00]. The first of these also discusses how architecture can coexist with agile processes.

A number of texts discuss the various modern software development processes, including XP [BECK00], Scrum [BEED02], Feature Driven Development [PALM02], and the Rational Unified Process [KRUC00].

The Twin Peaks model (on which we based our Three Peaks model) is described in an article in *IEEE Computer* [NUSE01].