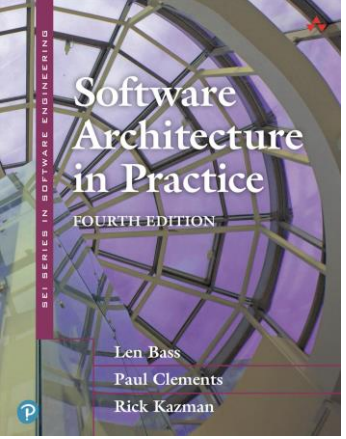
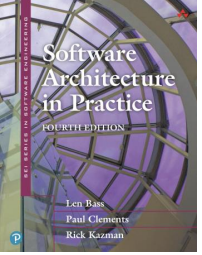


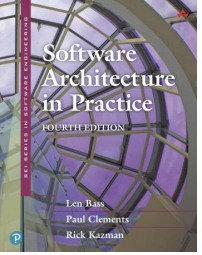
Chapter 3: Understanding Quality Attributes

Quality is never an accident; it is always the result of high intention, sincere effort, intelligent direction and skillful execution.
—William A. Foster



Chapter Outline

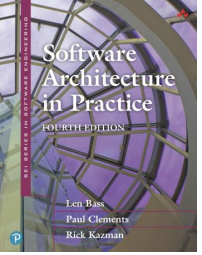
- Architecture and Requirements
- Functionality
- Quality Attribute Considerations
- Specifying Quality Attribute Requirements
- Achieving Quality Attributes through Tactics
- Guiding Quality Design Decisions
- Summary



Architecture and Requirements

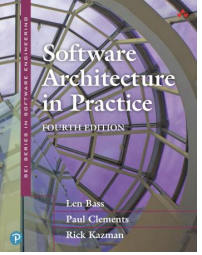
System requirements can be categorized as:

- Functional requirements. These requirements state what the system must do, how it must behave or react to run-time stimuli.
- Quality attribute requirements. These requirements annotate (qualify) functional requirements. Qualification might be how fast the function must be performed, how resilient it must be to erroneous input, how easy the function is to learn, etc.
- Constraints. A constraint is a design decision with zero degrees of freedom. That is, it's a design decision that has already been made for you.



Functionality

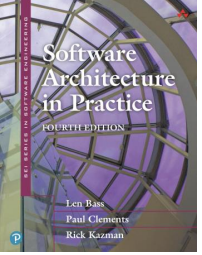
- Functionality is the ability of the system to do the work for which it was intended.
- Functionality has a strange relationship to architecture:
  - functionality does not determine architecture; given a set of required functionality, there is no end to the architectures you could create to satisfy that functionality
  - functionality and quality attributes are orthogonal



Quality Attribute Considerations

If a functional requirement is "when the user presses the green button the Options dialog appears”:

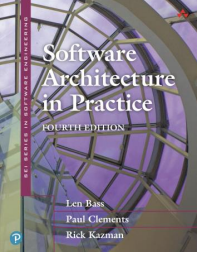
- a performance QA annotation might describe how quickly the dialog will appear;
- an availability QA annotation might describe how often this function will fail, and how quickly it will be repaired;
- a usability QA annotation might describe how easy it is to learn this function.



Quality Attribute Considerations

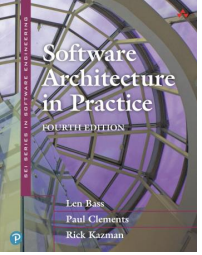
There are three problems with previous discussions of quality attributes:

1. The definitions provided for an attribute are not testable. It is meaningless to say that a system will be “modifiable”.
2. Endless time is wasted on arguing over which quality a concern belongs to. Is a system failure due to a denial of service attack an aspect of availability, performance, security, or usability?
3. Each attribute community has developed its own vocabulary.



Quality Attribute Considerations

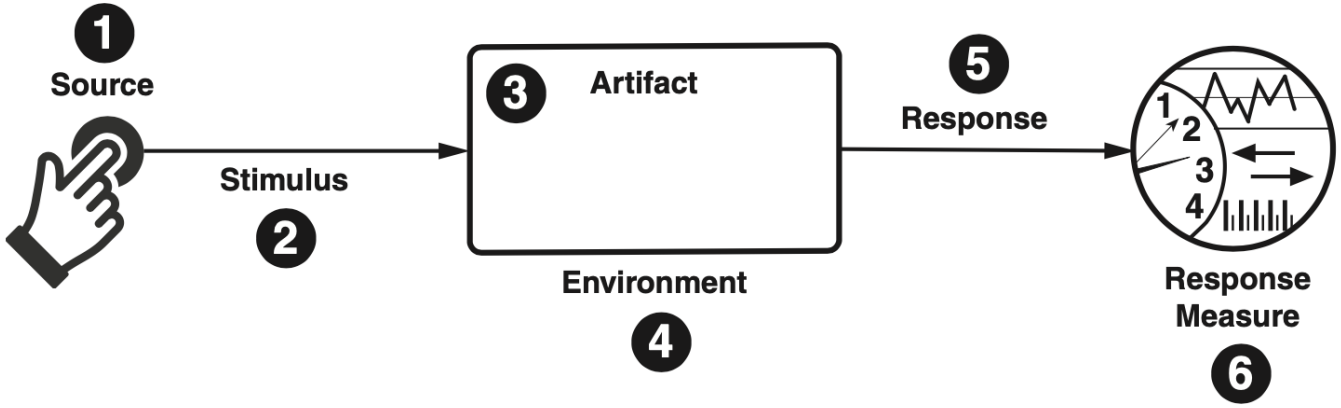
- A solution to the first two of these problems (untestable definitions and overlapping concerns) is to use *quality attribute scenarios* as a means of characterizing quality attributes.
- A solution to the third problem is to provide a discussion of each attribute—concentrating on its underlying concerns—to illustrate the concepts that are fundamental to that attribute community.

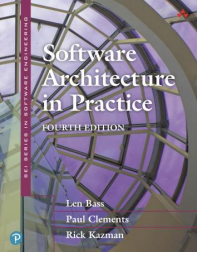


Specifying Quality Attribute Requirements

- We use a common form to specify all quality attribute requirements as scenarios.
- Our representation of quality attribute scenarios has these parts:

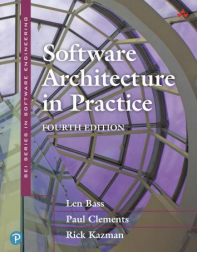
1. **Stimulus**
2. **Stimulus source**
3. **Response**
4. **Response measure**
5. **Environment**
6. **Artifact**





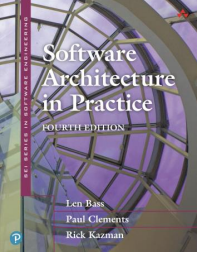
Specifying Quality Attribute Requirements

1. **Source of stimulus**. This is some entity (a human, a computer system, or any other actuator) that generated the stimulus.
2. **Stimulus**. The stimulus is a condition that requires a response when it arrives at a system.
3. **Environment**. The stimulus occurs under certain conditions. The system may be in an overload condition or in normal operation, or some other relevant state. For many systems, "normal" operation can refer to one of a number of modes.
4. **Artifact**. Some artifact is stimulated. This may be a collection of systems, the whole system, or some piece or pieces of it.
5. **Response**. The response is the activity undertaken as the result of the arrival of the stimulus.
6. **Response measure**. When the response occurs, it should be measurable in some fashion so that the requirement can be tested.



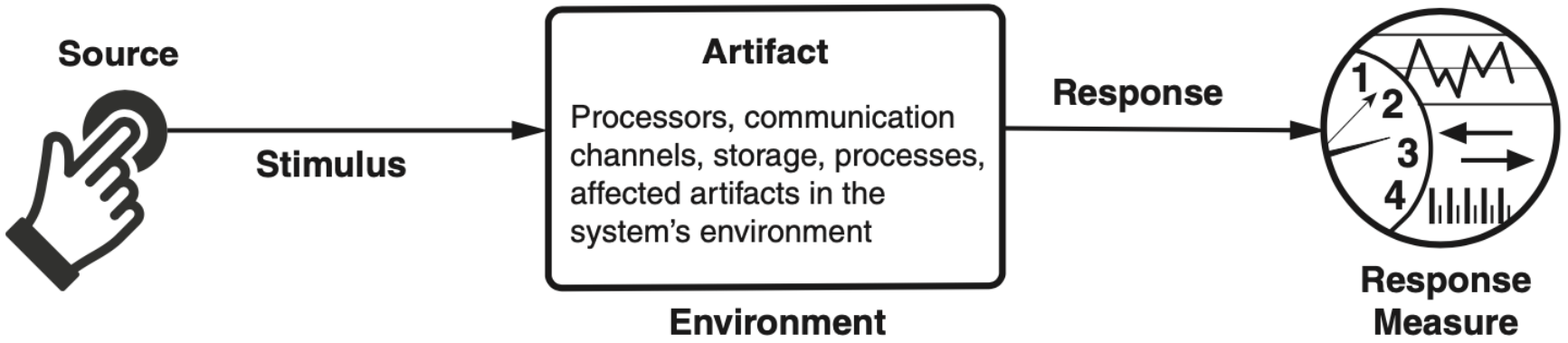
Specifying Quality Attribute Requirements

We distinguish *general* quality attribute scenarios (“general scenarios”)—those that are system independent and can, potentially, pertain to any system—from *concrete* quality attribute scenarios (concrete scenarios)—those that are specific to the particular system under consideration.

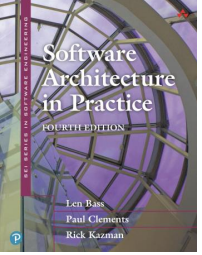


Specifying Quality Attribute Requirements

Example general scenario for availability:

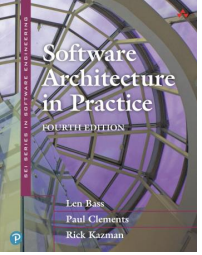


Source	Stimulus	Environment	Response	Response Measure
Internal/external: people, hardware, software, physical infrastructure, physical environment	Fault: omission, crash, incorrect timing, incorrect response	Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation	Prevent the fault from becoming a failure Detect the fault Recover from the fault	Time or time interval when the system must be available Availability percentage Time to detect the fault



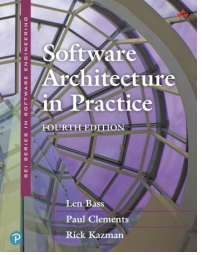
Achieving Quality Attributes Through Patterns and Tactics

- There are a collection of primitive design techniques that an architect can use to achieve a quality attribute response.
- We call these architectural design primitives *tactics*.
- Tactics, like design patterns, are techniques that architects have been using for years. We do not *invent* tactics, we simply capture what architects do in practice.



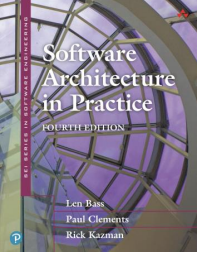
Achieving Quality Attributes Through Patterns and Tactics

- An architectural pattern describes a recurring design problem that arises in specific design contexts and presents a well-proven architectural solution for the problem.
- The solution is specified by describing the roles of its constituent elements, their responsibilities and relationships, and the ways in which they collaborate.



Achieving QAs Through Patterns and Tactics

- Patterns typically comprise multiple design decisions and, in fact, often comprise multiple quality attribute tactics.
- We say that patterns often bundle tactics and, consequently, frequently make tradeoffs among quality attributes.

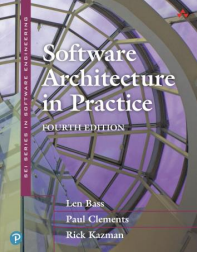


Achieving QAs Through Tactics

Patterns are ubiquitous. Why do we do also bother with tactics?

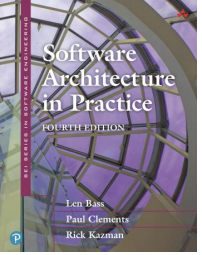
There are three reasons:

1. Design patterns are complex; they are a bundle of design decisions. But patterns are often difficult to apply as is; architects need to modify and adapt them. By understanding tactics, an architect can assess options for augmenting an existing pattern to achieve a quality attribute goal.
2. If no pattern exists to realize the architect's design goal, tactics allow the architect to construct a design fragment from "first principles".
3. Tactics provide a way of making design and analysis more systematic.



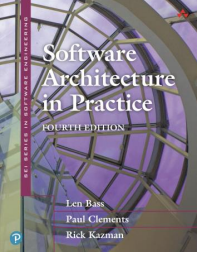
Analyzing QA Design Decisions: Tactics-Based Questionnaires

- Analyzing how well quality attributes have been achieved is a critical part of the task of designing an architecture.
- You shouldn’t wait until your design is "complete" before you begin to do it.
- Opportunities for quality attribute analysis crop up at many different points in the software development life cycle, even very early ones.



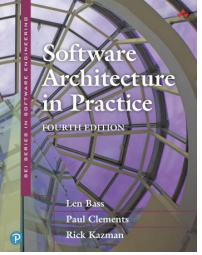
Analyzing QA Design Decisions: Tactics-Based Questionnaires

- The accuracy of the analysis and expected degree of confidence in the analysis results will vary according to the maturity of the available artifacts.
- Tactics-based questionnaires can be helpful in gaining insights into the architecture's ability to support the needed quality attributes.
- Tactics-based questionnaires will be provided in Chapters 4-13.



Analyzing QA Design Decisions: Tactics-Based Questionnaires

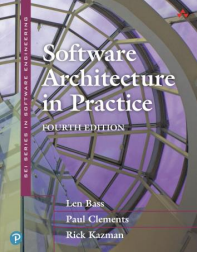
- The accuracy of the analysis and expected degree of confidence in the analysis results will vary according to the maturity of the available artifacts.
- Tactics-based questionnaires can be helpful in gaining insights into the architecture's ability to support the needed quality attributes.
- Tactics-based questionnaires will be provided in Chapters 4-13.



Analyzing QA Design Decisions: Tactics-Based Questionnaires

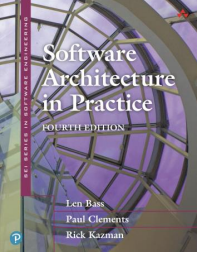
For each question in the questionnaire, the analyst records the following information:

1. Whether each tactic is supported by the system's architecture.
2. Whether there are any obvious risks in the use (or nonuse) of this tactic. If the tactic has been used, record how it is realized in the system, or how it is intended to be realized (e.g., via custom code, generic frameworks, or externally produced components).
3. The specific design decisions made to realize the tactic and where in the code base the implementation (realization) may be found. This is useful for auditing and architecture reconstruction purposes.
4. Any rationale or assumptions made in the realization of this tactic.



Summary

- Requirements for a system come in three categories.
  1. Functional. These requirements are satisfied by including an appropriate set of responsibilities within the design.
  2. Quality attribute. These requirements are satisfied by the structures and behaviors of the architecture.
  3. Constraints. A constraint is a design decision that's already been made.
- To express a quality attribute requirement we use a quality attribute scenario. The parts of the scenario are:
  1. Source of stimulus.
  2. Stimulus
  3. Environment.
  4. Artifact.
  5. Response.
  6. Response measure.



Summary

- An architectural tactic is a design decision that affects a quality attribute response. The focus of a tactic is on a single quality attribute response.
- Architectural patterns can be seen as “packages” of tactics.
- An analyst can understand the decisions made in an architecture through the use of a tactics-based checklist.
- This lightweight architecture analysis technique can provide insights into the strengths and weaknesses of the architecture in a very short amount of time.