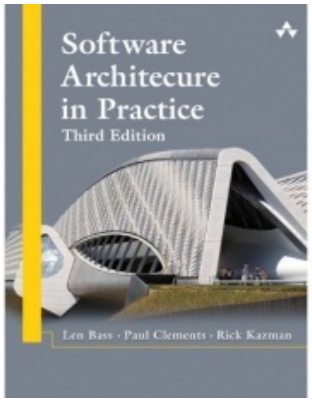
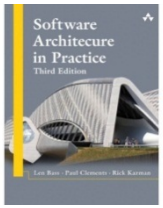
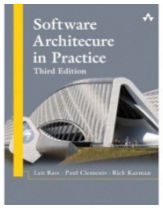


Chapter 4: Understanding Quality Attributes



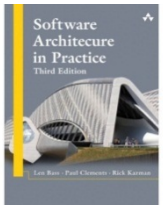
Chapter Outline

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



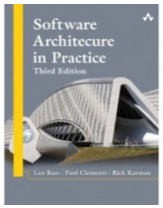
Architecture and Requirements

- 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.



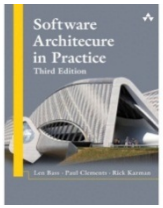
Functionality

- 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

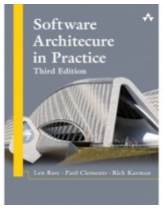
- If a functional requirement is "when the user presses the green button the Options dialog appears”: (Performance, Availabilty, Usability…..)
  - a performance QA annotation might describe how quickly the dialog will appear;
  - an availability QA annotation might describe how often this function will 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

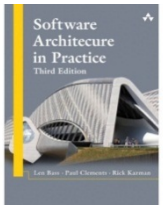
- There are three problems with previous discussions of quality attributes:

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.



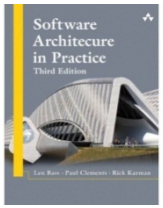
Quality Attribute Considerations

- 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.



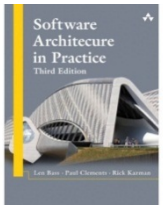
Specifying Quality Attribute Requirements

- 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**



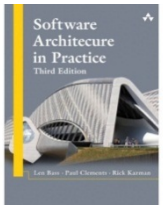
Specifying Quality Attribute Requirements

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:

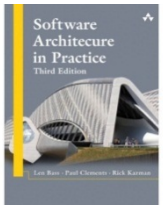
Source:
Internal/External: people, hardware, software, physical infrastructure, physical environment

Stimulus:
Fault: omission, crash, incorrect timing, incorrect response

Artifact:
Processors, communication channels, persistent storage, processes

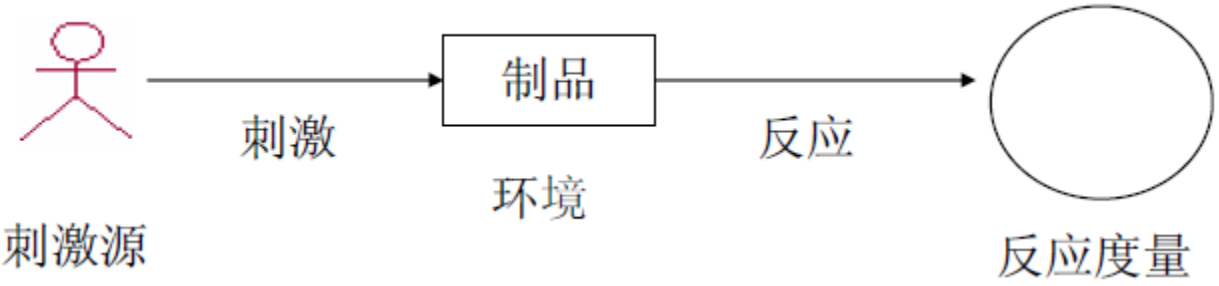
Response:
Prevent fault from becoming failure
Detect fault: log, notify
Recover from fault: disable event source, be unavailable, fix/mask, degraded mode

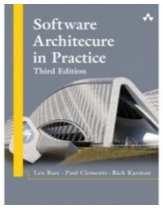
Response Measure:
Time system must be available
Availability percentage
Time in degraded mode
Repair time
Proportion of faults system handles



质量属性场景-ground quality in specific “use situation”

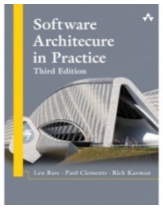
- 质量属性场景就是通过对某个实体与系统的一次交互的简要描述说明一个有关质量属性的特定需求，它由六部分组成：
- 刺激源：可以是风险承担者、计算机系统等。
- 刺激：可以看作是一个事件。
- 环境：系统当前的状态。
- 制品：系统中对事件作出反应的部分，可以是整个系统或系统的某一部分。
- 反应：事件到达后系统的相关行为。
- 反应度量：对反应结果提供某种形式的衡量。





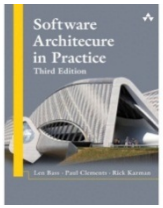
Achieving Quality Attributes Through Tactics

- 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.



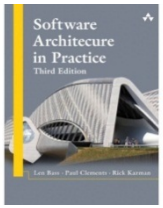
Achieving Quality Attributes Through Tactics

- Why do we do this? 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 the 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. By cataloguing tactics, we make design more systematic. You frequently will have a choice of multiple tactics to improve a particular quality attribute. The choice of which tactic to use depends on factors such as tradeoffs among other quality attributes and the cost to implement.



Guiding Quality Design Decisions

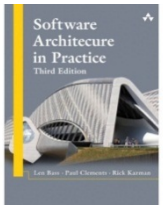
- Architecture design is a systematic approach to making design decisions.
- We categorize the **design decisions** that an architect needs to make as follows:
  1. Allocation of responsibilities
  2. Coordination model
  3. Data model
  4. Management of resources
  5. Mapping among architectural elements
  6. Binding time decisions
  7. Choice of technology



Allocation of Responsibilities

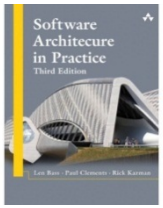
- Decisions involving allocation of responsibilities include:
  - identifying the important responsibilities including basic system functions, architectural infrastructure, and satisfaction of quality attributes.
  - determining how these responsibilities are allocated to non-runtime and runtime elements (namely, modules, components, and connectors).

Differences?



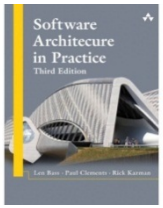
Coordination Model

- Decisions about the coordination model include:
  - identify the elements of the system that must coordinate, or are prohibited from coordinating
  - determining the properties of the coordination, such as timeliness, currency, completeness, correctness, and consistency
  - choosing the communication mechanisms that realize those properties. Important properties of the communication mechanisms include stateful vs. stateless, synchronous vs. asynchronous, guaranteed vs. non-guaranteed delivery, and performance-related properties such as throughput and latency



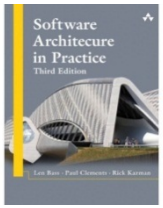
Data Model

- Decisions about the data model include:
  - choosing the major data abstractions, their operations, and their properties. This includes determining how the data items are created, initialized, accessed, persisted, manipulated, translated, and destroyed.
  - metadata needed for consistent interpretation of the data
  - organization of the data. This includes determining whether the data is going to be kept in a relational data base, a collection of objects or both



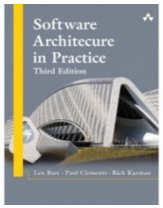
Management of Resources

- Decisions for management of resources include:
  - identifying the resources that must be managed and determining the limits for each
  - determining which system element(s) manage each resource
  - determining how resources are shared and the arbitration strategies employed when there is contention
  - determining the impact of saturation on different resources.



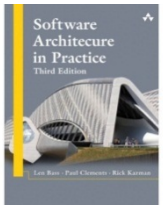
Mapping Among Architectural Elements

- Useful mappings include:
  - the mapping of modules and runtime elements to each other—that is, the runtime elements that are created from each module; the modules that contain the code for each runtime element
  - the assignment of runtime elements to processors
  - the assignment of items in the data model to data stores
  - the mapping of modules and runtime elements to units of delivery



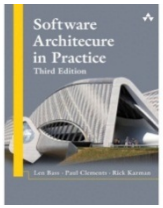
Binding Time

- The decisions in the other categories have an associated binding time decision. Examples of such binding time decisions include:
  - For allocation of responsibilities you can have build-time selection of modules via a parameterized build script.
  - For choice of coordination model you can design run-time negotiation of protocols.
  - For resource management you can design a system to accept new peripheral devices plugged in at run-time.
  - For choice of technology you can build an app-store for a smart phone that automatically downloads the appropriate version of the app.



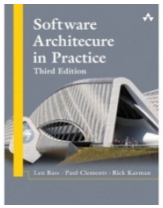
Choice of Technology

- Choice of technology decisions involve:
  - deciding which technologies are available to realize the decisions made in the other categories
  - determining whether the tools to support this technology (IDEs, simulators, testing tools, etc.) are adequate
  - determining the extent of internal familiarity and external support for the technology (such as courses, tutorials, examples, availability of contractors)
  - determining the side effects of choosing a technology such as a required coordination model or constrained resource management opportunities
  - determining whether a new technology is compatible with the existing technology stack



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.
- The seven categories of architectural design decisions are:
  1. Allocation of responsibilities
  2. Coordination model
  3. Data model
  4. Management of resources
  5. Mapping among architectural elements
  6. Binding time decisions
  7. Choice of technology