考点：

Chapter 1. What is Software Architecture?

理解：软件体系结构（软件架构）的定义、架构模式的概念。

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both

一个程序的软件体系结构或者一个计算机系统是一个构架或者是一个系统的构架。它包括软件元素、这些元素的外观，以及它们之间的关系。

Architecture Is a Set of Software Structures

A structure is a set of elements held together by a relation

掌握：软件系统有哪几类结构？在每类结构里，元素及其之间的关系是什么？每类结构各有哪些常见的结构？其特点是什么？

There are three categories of structures:

Module structures show how a system is to be structured as a set of code or data units that have to be constructed or procured获得.

Component-and-connector structures show how the system is to be structured as a set of elements that have runtime behavior (components) and interactions (connectors).

Allocation structures show how the system will relate to nonsoftware structures in its environment (such as CPUs, file systems, networks, development teams, etc.).

了解：结构与视图是什么关系？好的结构的一些经验法则。

A view is a representation of a coherent set of architectural elements. A view is a representation of one or more structures.

Chapter 2. Why is Software Architecture Important?

理解：13个理由。

An architecture will inhibit or enable a system’s driving quality attributes.

The decisions made in an architecture allow you to reason about and manage change as the system evolves.

The analysis of an architecture enables early prediction of a system’s qualities.

A documented architecture enhances communication among stakeholders.

The architecture is a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.

An architecture defines a set of constraints on subsequent implementation.

The architecture dictates the structure of an organization, or vice versa.

An architecture can provide the basis for evolutionary prototyping.

An architecture is the key artifact that allows the architect and project manager to reason about cost and schedule.

An architecture can be created as a transferable, reusable model that form the heart of a product line.

Architecture-based development focuses attention on the assembly of components, rather than simply on their creation.

By restricting design alternatives, architecture channels the creativity of developers, reducing design and system complexity.

An architecture can be the foundation for training a new team member.

Chapter 3. The Many Contexts of Software Architecture

理解：技术环境、项目生命周期、商业环境、架构师职业环境中的软件体系结构。架构与环境的相互影响。

了解：涉众。

A stakeholder is anyone who has a stake（支柱） in the success of the system

Stakeholders typically have different specific concerns that they wish the system to guarantee or optimize.

You will need to know and understand the nature, source, and priority of constraints on the project as early as possible. Therefore, you must identify and actively engage the stakeholders to solicit请求 their needs and expectations.

Early engagement of stakeholders allows you to understand the constraints of the task, manage expectations, negotiate议定，商定 priorities, and make tradeoffs.

Chapter 4. Understanding Quality Attributes

了解：系统的功能需求。功能需求与系统架构的关系（一对多）。功能需求与质量需求的关系（无关）。系统约束。

Functional. These requirements are satisfied by including an appropriate set of responsibilities within the design.

Quality attribute. These requirements are satisfied by the structures and behaviors of the architecture.

Constraints. A constraint is a design decision that’s already been made.

理解：系统的质量需求。战术的概念。

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.

掌握：质量属性场景的概念和举例。质量设计的7种决策。

To express a quality attribute requirement we use a quality attribute scenario. The parts of the scenario are:

Source of stimulus刺激.

Stimulus

Environment.

Artifact人工制品.

Response.

Response measure.

The seven categories of architectural design decisions are:

Allocation of responsibilities

Coordination model

Data model

Management of resources

Mapping among architectural elements

Binding time decisions

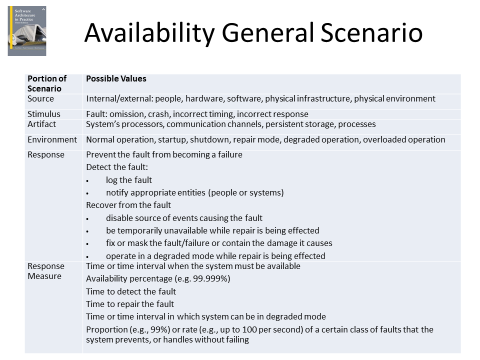
Choice of technology

Chapter 5. Availability

理解：可用性概念。

Availability refers to the ability of the system to be available for use when a fault occurs

了解：可用性公式。可用性一般场景。



掌握：可用性战术。可用性设计清单。



Sanity:明智

Degradation:退化

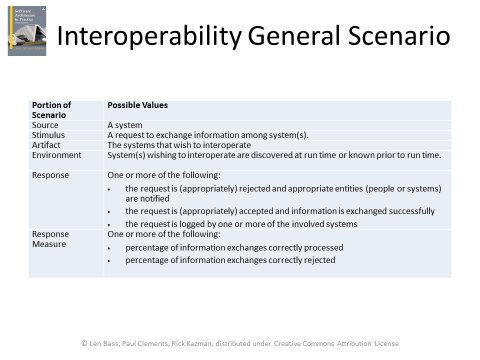
Escalation:升级

Chapter 6. Interoperability

理解：互操作性概念。

Interoperability is about the degree to which two or more systems can usefully exchange meaningful information.

了解：互操作性一般场景。



掌握：互操作性战术。互操作性设计清单。



Orchestrate:使和谐

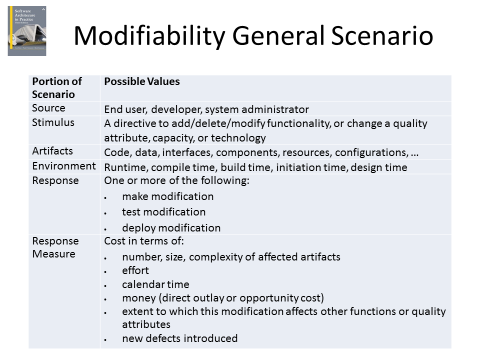
:

Chapter 7. Modifiability

理解：可修改性概念。

Modifiability is about change and our interest in it is in the cost and risk of making changes

了解：可修改性一般场景。



掌握：可修改性战术。可修改性设计清单。

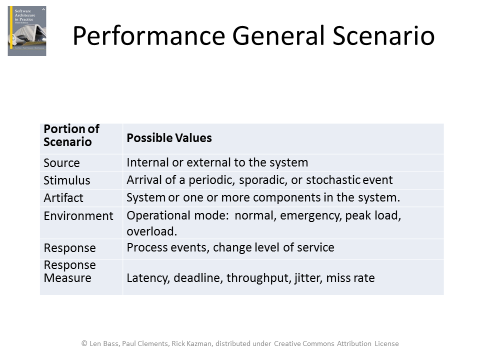


Chapter 8. Performance

理解：性能概念。

Performance is about time and the software system’s ability to meet timing requirements.

了解：性能公式。性能一般场景。



Jitter:极度紧张



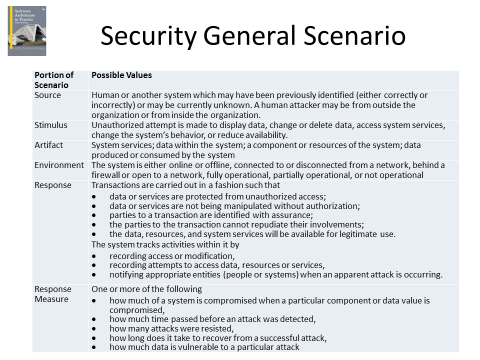
掌握：性能战术。性能设计清单。

Chapter 9. Security

理解：安全性概念。

Security is a measure of the system’s ability to protect data and information from unauthorized access while still providing access to people and systems that are authorized.

了解：安全一般场景。



掌握：安全性战术。安全性设计清单。



Chapter 10. Testability

理解：可测试性概念。

Software testability refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.

了解：可测试性一般场景。



掌握：可测试性战术。可测试性设计清单。

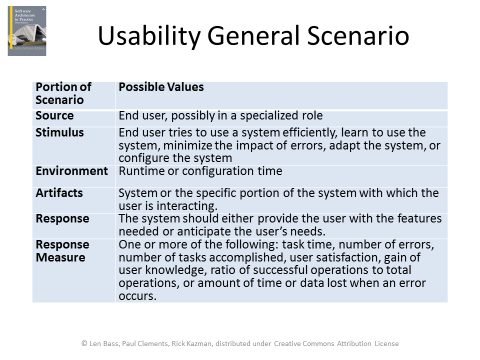


Chapter 11. Usability

理解：易用性概念。

Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides.

了解：易用性一般场景。



掌握：易用性战术。易用性设计清单。



Chapter 12. Other Quality Attributes

了解：其它软件质量属性如可变性、可移植性、开发可分布性、伸缩性、可部署性、移动性、可监控性、生命财产安全性。其它类别的质量属性如架构质量属性、商业属性、系统质量属性。ISO/IEC FCD 25010 产品质量标准。

其他软件质量属性：

Variability可变性: is a special form of modifiability. It refers to the ability of a system and its supporting artifacts to support the production of a set of variants that differ from each other in a preplanned fashion.

Portability可移植性: is also a special form of modifiability. Portability refers to the ease with which software that built to run on one platform can be changed to run on a different platform.

Development Distributability开发可分布性: is the quality of designing the software to support distributed software development.

Scalability伸缩性: Horizontal scalability (scaling out) refers to adding more resources to logical units such as adding another server to a cluster. Vertical scalability (scaling up) refers to adding more resources to a physical unit such as adding more memory to a computer.

Deployability可部署性: is concerned with how an executable arrives at a host platform and how it is invoked.

Mobility移动性: deals with the problems of movement and affordances of a platform (e.g. size, type of display, type of input devices, availability and volume of bandwidth, and battery life).

Monitorability可监控性: deals with the ability of the operations staff to monitor the system while it is executing.

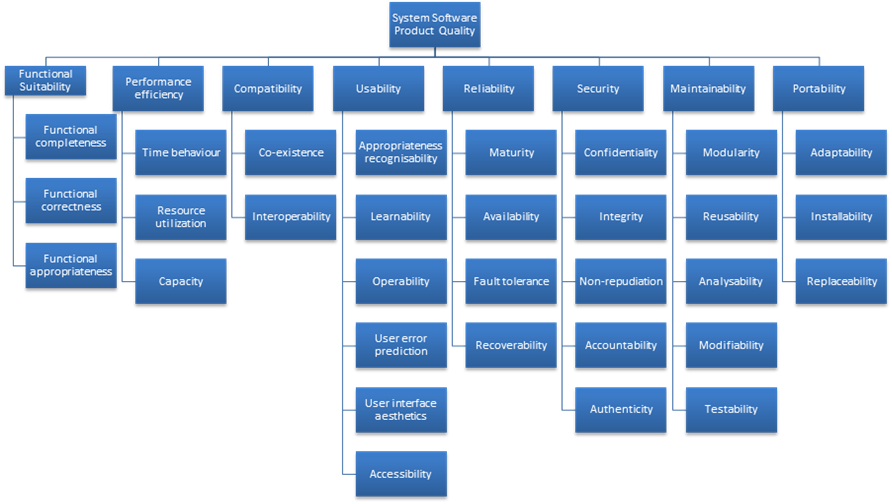
Safety生命财产安全性: Software safety is about the software’s ability to avoid entering states that cause or lead to damage, injury, or loss of life, and to recover and limit the damage when it does enter into bad states. The architectural concerns with safety are almost identical with those for availability (i.e. preventing, detecting, and recovering from failures).

其他类别质量属性

Conceptual Integrity: refers to consistency in the design of the architecture. It contributes to the understandability of the architecture. Conceptual integrity demands that the same thing is done in the same way through the architecture.

Marketability: Some systems are marketed by their architectures, and these architectures sometimes carry a meaning all their own, independent of what other quality attributes they bring to the system (e.g. service-oriented or cloud-based).

Quality in Use: qualities that pertain to the use of the system by various stakeholders.



理解：如何处理未知的质量属性。

Suppose you must deal with a quality attribute for which there is no compact body of knowledge, e.g. green computing.

What do you do?

Model the quality attribute

Assemble a set of tactics for the quality attribute

Construct design checklists

Chapter 13. Patterns and Tactics

了解：架构模式（架构风格）的概念。

An architectural pattern establishes a relationship between

A context. A recurring, common situation in the world that gives rise to a problem.

A problem. The problem, appropriately generalized, that arises in the given context.

A solution. A successful architectural resolution to the problem, appropriately abstracted. The solution for a pattern is determined and described by:

-A set of element types (for example, data repositories, processes, and objects)

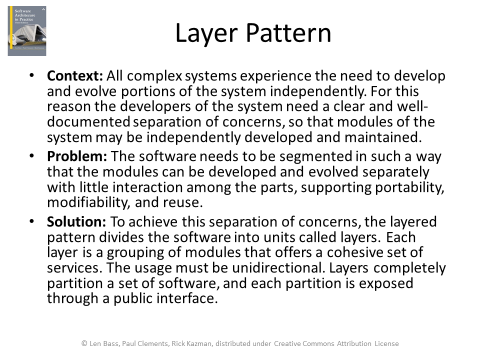
-A set of interaction mechanisms or connectors (for example, method calls, events, or message bus)

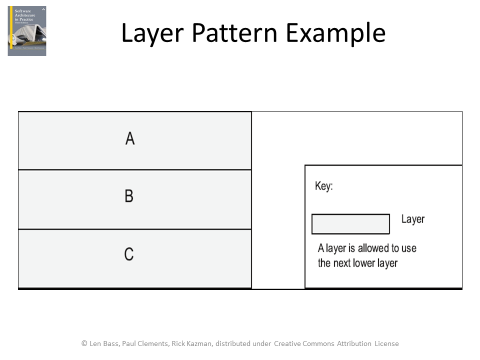
-A topological layout of the components

-A set of semantic constraints covering topology, element behavior, and interaction mechanisms

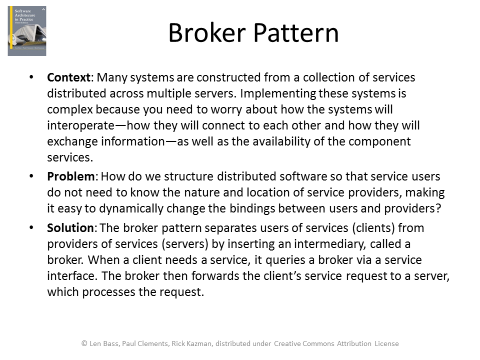
掌握：

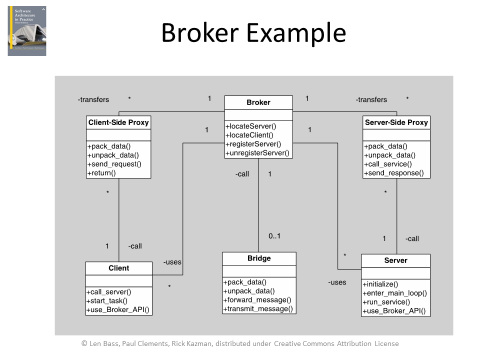
层次模式



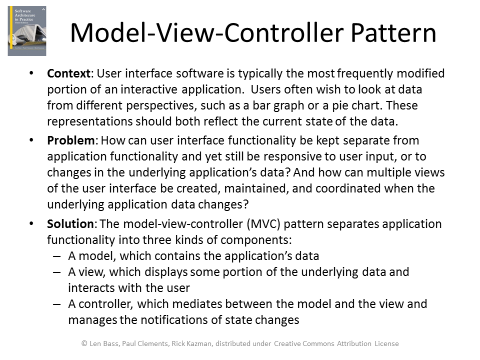


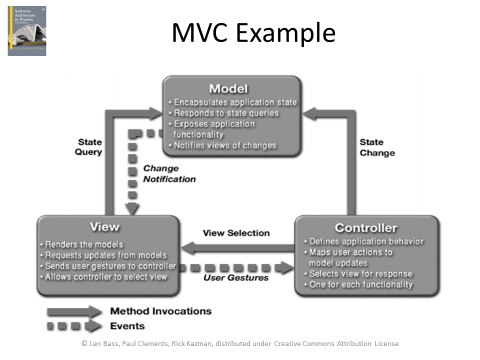
代理模式



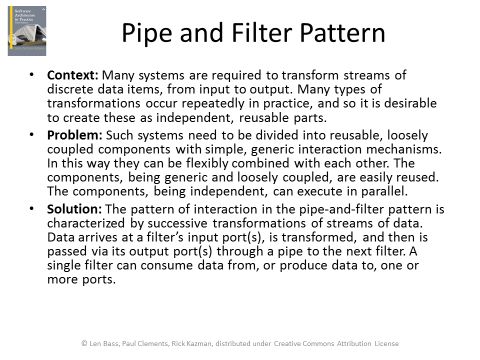


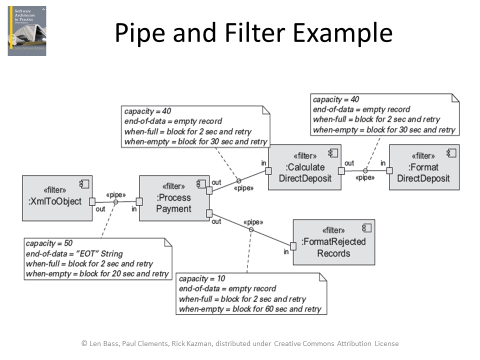
MVC模式



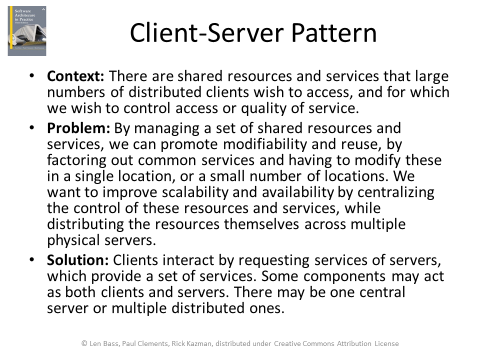


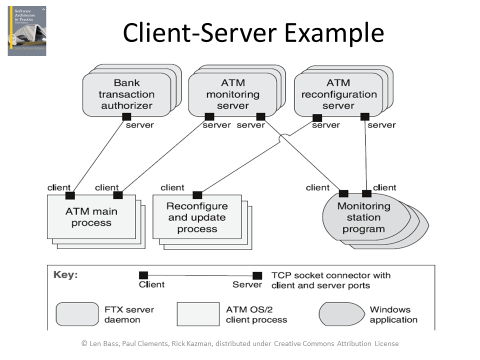
管道-过滤器模式



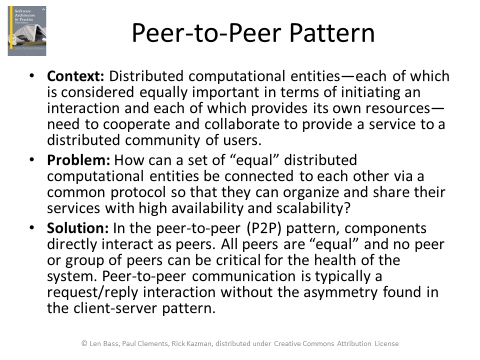


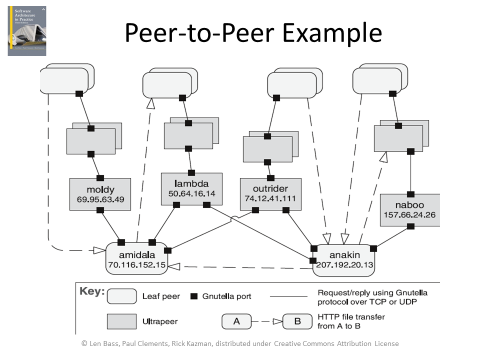
CS模式



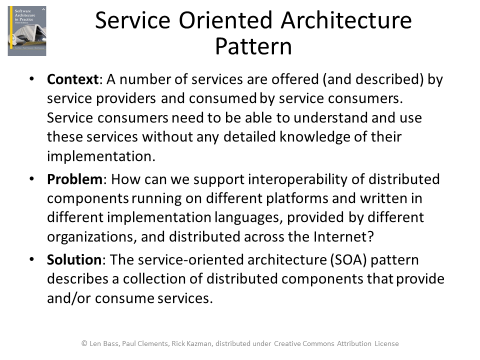


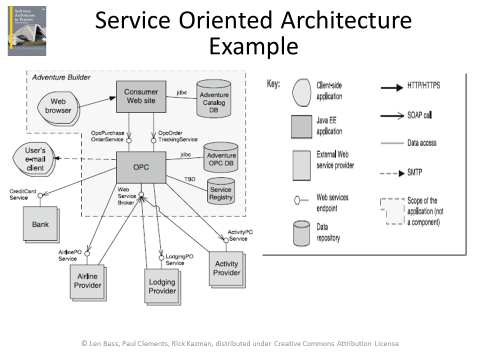
P2P模式



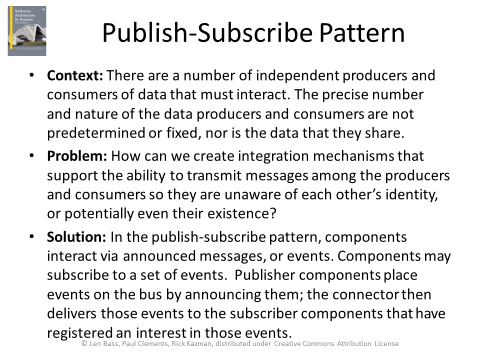


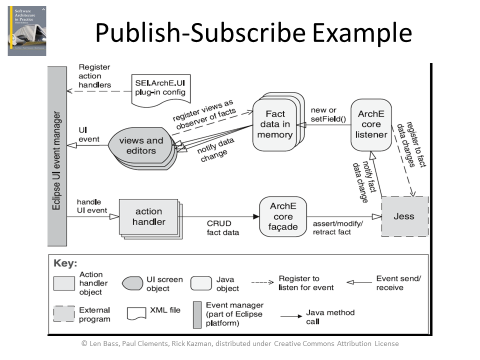
SOA模式



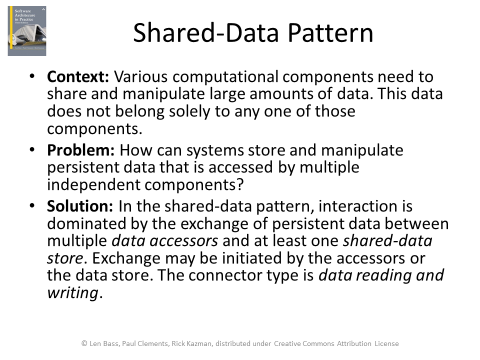


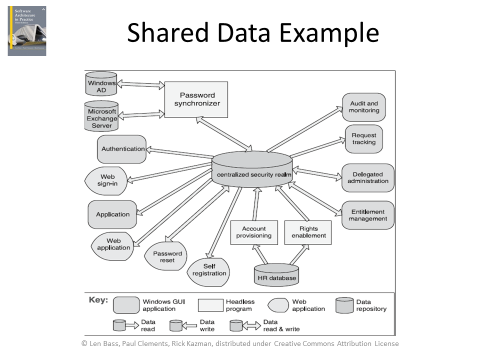
发布订阅模式



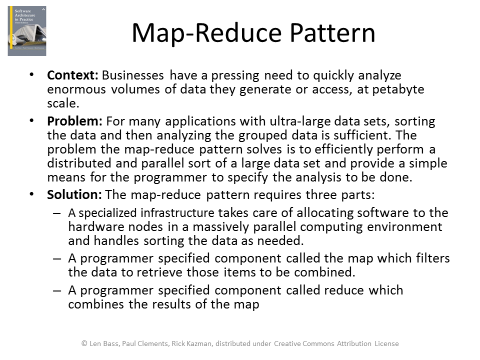


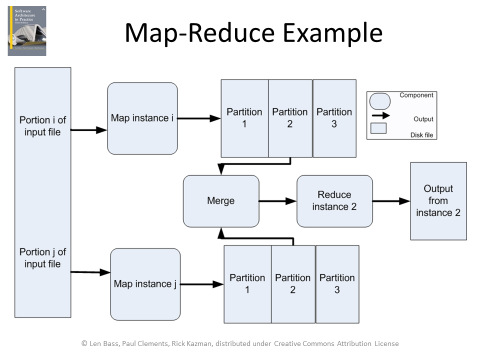
共享数据模式



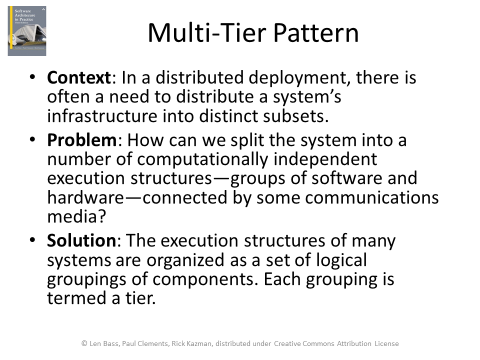


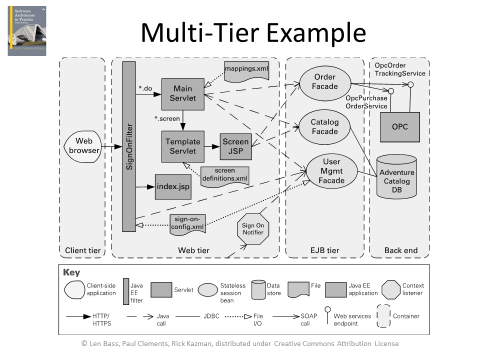
Map-Reduce模式





多级模式。





理解：模式与战术的关系。

Patterns are built from tactics; if a pattern is a molecule分子, a tactic is an atom.

MVC, for example utilizes the tactics:

-Increase semantic coherence

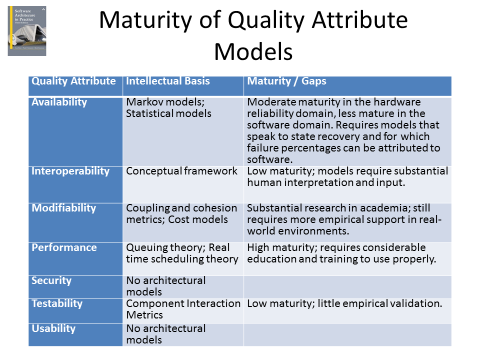
-Encapsulation

-Use an intermediary

-Use run time binding

Chapter 14. Quality Attribute Modeling and Analysis

了解：模型。常见质量属性模型的成熟度。



了解：思想实验。粗略分析。原型。模拟仿真。实验。

Thought experiment

A thought experiment is mentally or verbally working through a particular scenario.

-Commonly done by the architect during design to explore alternatives.

-Also done during evaluation/documentation to convince third parties of the wisdom of particular design choices

Experiment

Many tools can help perform experiments to determine behavior of a design

-Request generators can create synthetic loads to test scalability

-Monitors can perform non-intrusive resource usage detection.

Back of the envelope analysis

Analysis does not need to be precise or detailed.

Rough analysis serves for many purposes. E.g. “the volume of traffic generated by this source should be well within the bounds handled by modern infrastructure”

Only do deeper analysis for questionable areas or important requirements.

Prototype

These depend on having a partial or prototype implementation.

-Prototype alternatives for the most important decisions

-If possible, implement prototype in a fashion so that some of it can be re-used.

-Fault injection tools can induce faults to determine response of system under failure conditions.

Simulation

Event based simulators exist that can be used to simulate behavior of system under various loads

-Must create the simulation.

-Must have a variety of different loads and responses to check for.

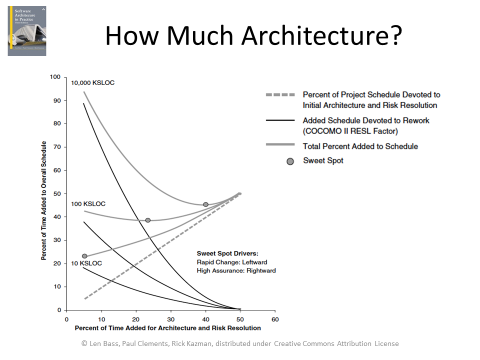
Chapter 15. Architectures in Agile Projects

了解：敏捷开发思想与准则。

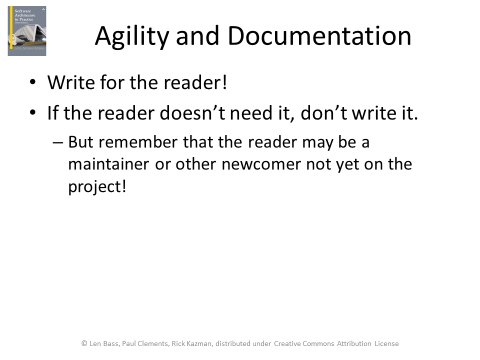
Twelve Agile Principles

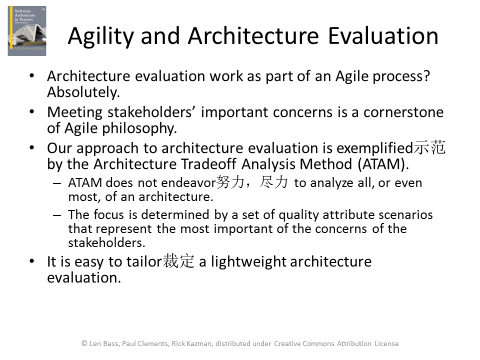
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

理解：敏捷开发的甜蜜点。



了解：敏捷开发与架构编档。敏捷开发与架构演化。





Chapter 16. Architecture and Requirements

理解：ASR。ASR的几种获取方法。QAW。

*architecturally significant requirement*

An architecturally significant requirement (ASR) is a requirement that will have a profound effect on the architecture.

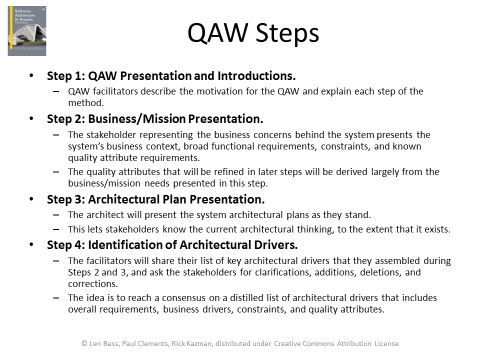
An obvious location to look for candidate ASRs is in the requirements documents or in user stories.

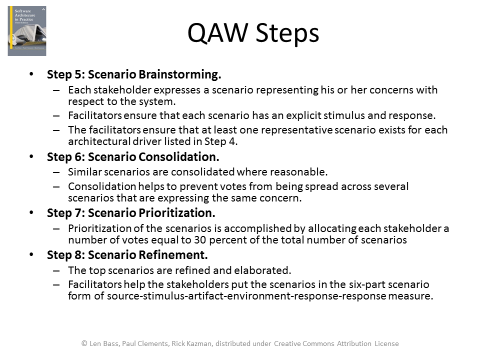
ASRs.tiff

Quality Attribute Workshop

The QAW is a facilitated, stakeholder-focused method to generate, prioritize, and refine quality attribute scenarios before the software architecture is completed.

The QAW is focused on system-level concerns and specifically the role that software will play in the system.





了解：商业目标场景。



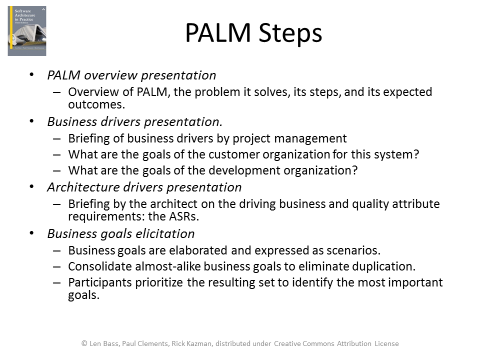
理解：PALM方法。

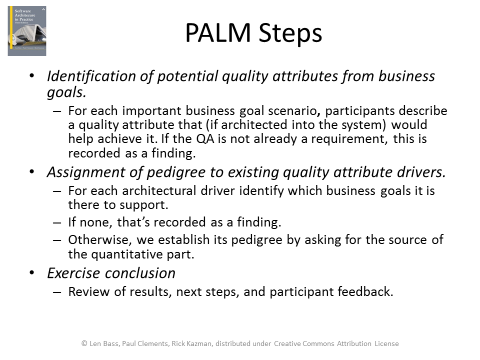
PALM: A Method for Eliciting引出 Business Goals

PALM is a seven-step method.

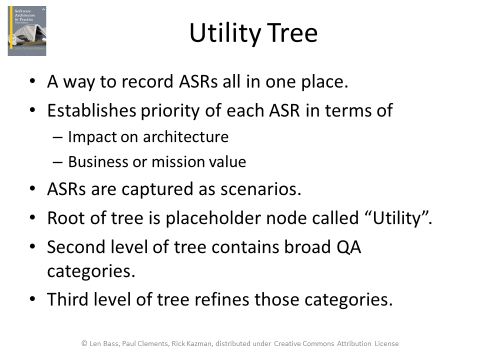
Nominally carried out over a day and a half in a workshop.

Attended by architect(s) and stakeholders who can speak to the relevant business goals.





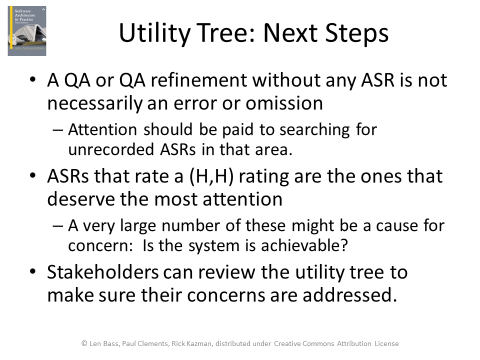
掌握：效用树。



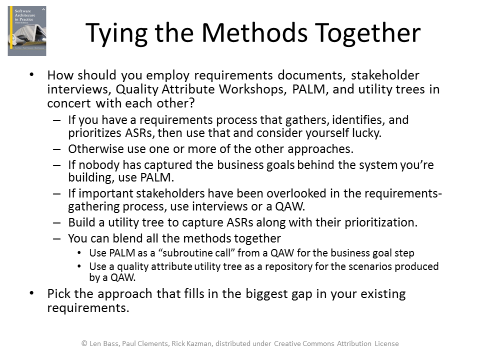
QA：质量属性

效用树范例（摘要，不需要记，了解就好）

UT1.tiff

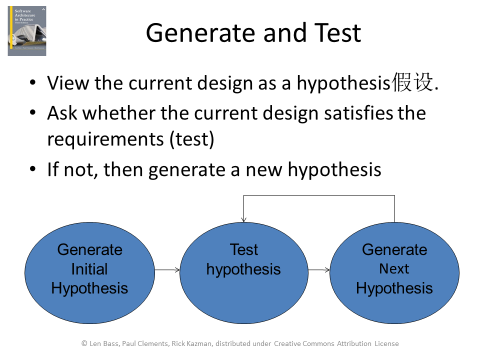


总结：什么时候用什么方法获取ASR

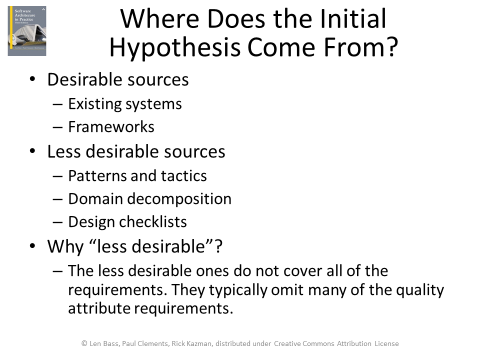


Chapter 17. Designing an Architecture

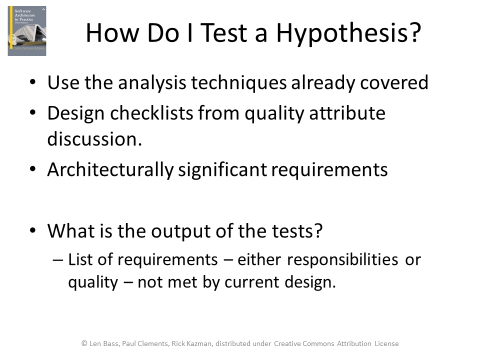
理解：Generate and Test（架构设计的假设检验法）。初始化、迭代、终结。

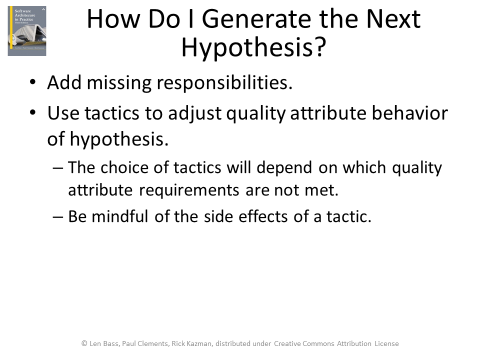


初始化（了解就行，下同）

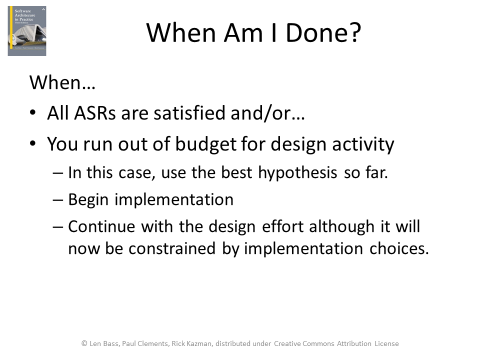


迭代

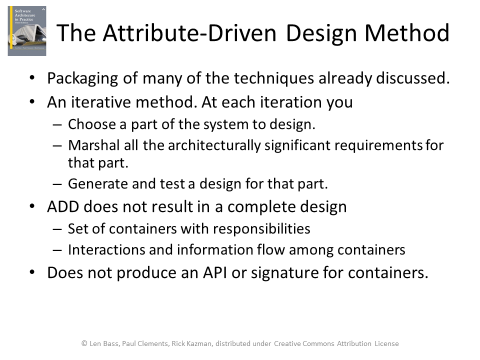




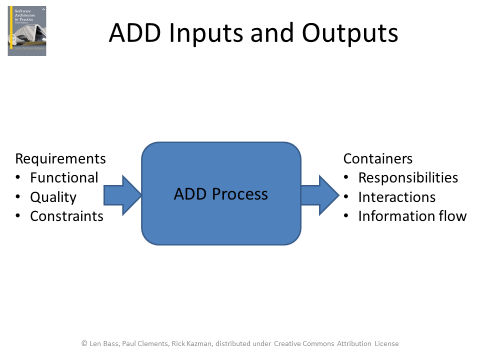
终结

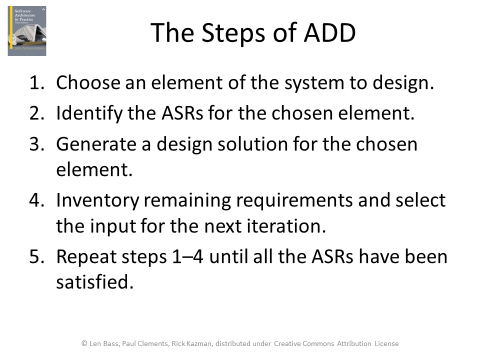


掌握：ADD方法。

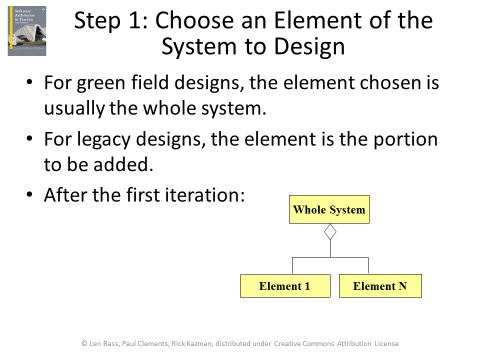


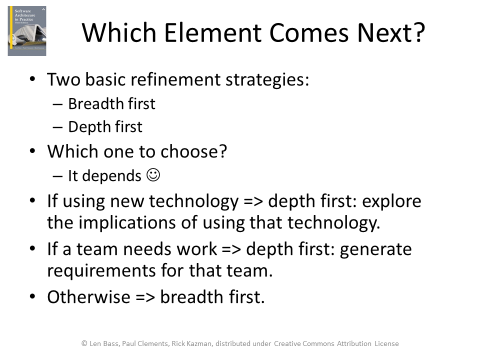
Marshal: 安排，整理

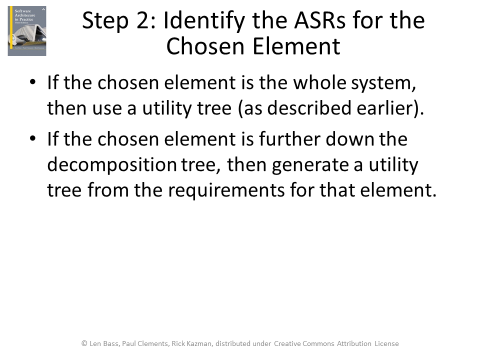


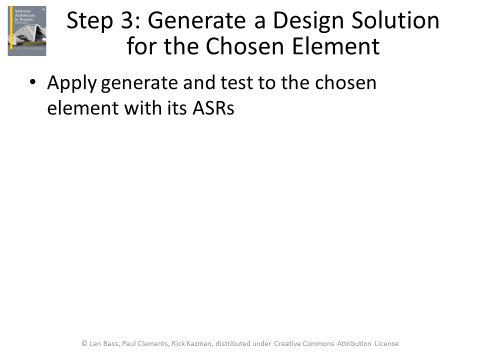


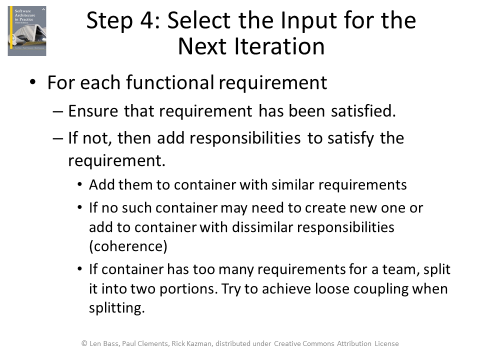
注意选择元素涉及深度优先广度优先的问题。了解就好





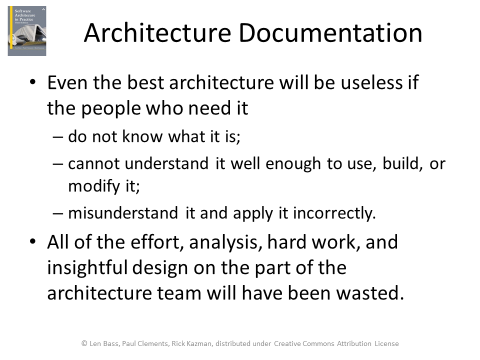


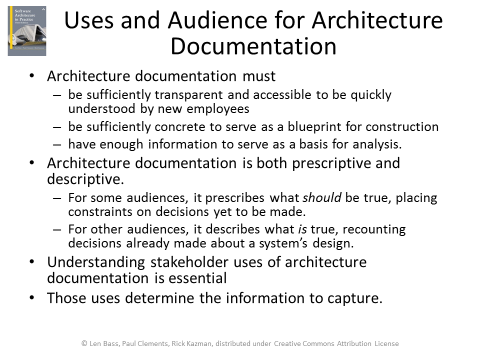




Chapter 18. Documenting Software Architectures

理解：架构编档的重要性。架构文档的用途、读者。

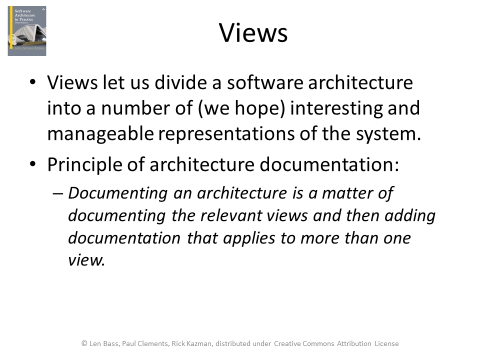


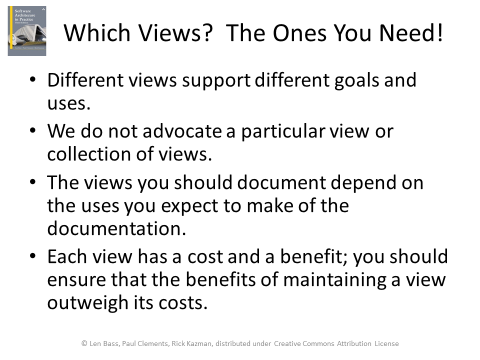


Prescriptive规定的

Descriptive记叙的，描述的

了解：如何选择视图进行编档。





Advocate: 拥护

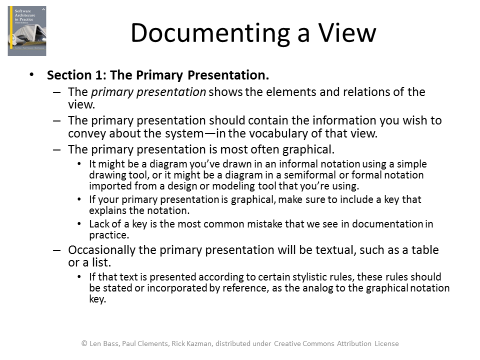
选择视图方法：

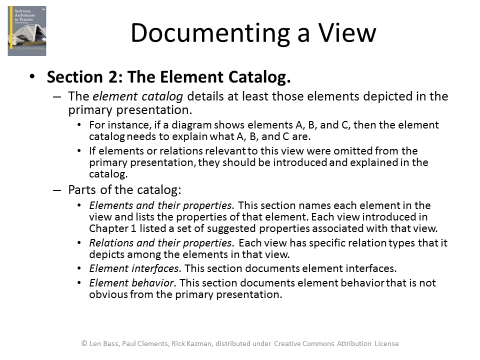
Step 1. Build a stakeholder/view table

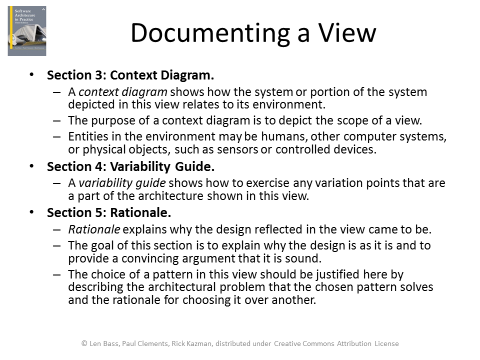
Step 2. Combine views to reduce their number

Step 3. Prioritize and stage.

掌握：如何对视图进行编档。视图文档模版。架构文档摘要信息。如何对系统行为进行编档。如何对质量属性进行编档。



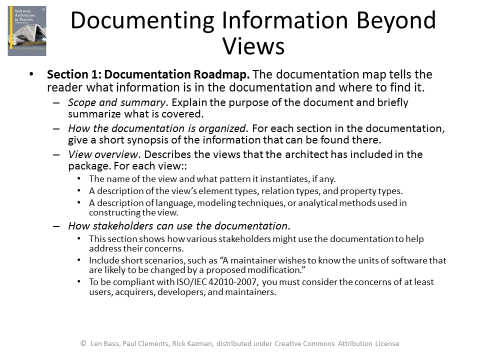


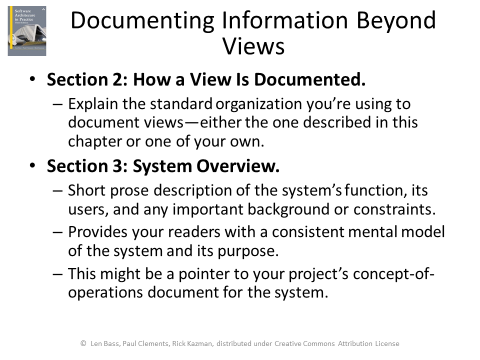


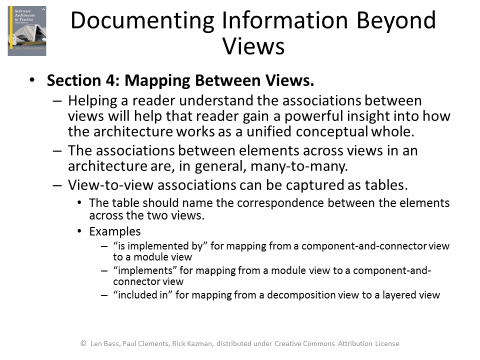
Rationale理论基础，基本原理

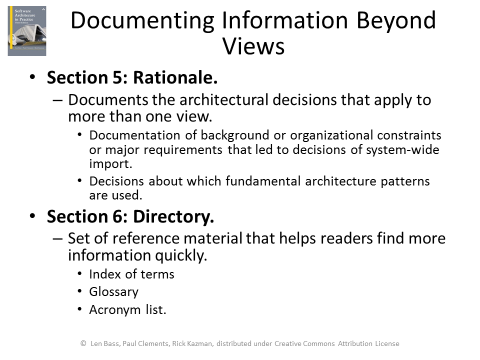
视图模板：

viewtemplate.tiff

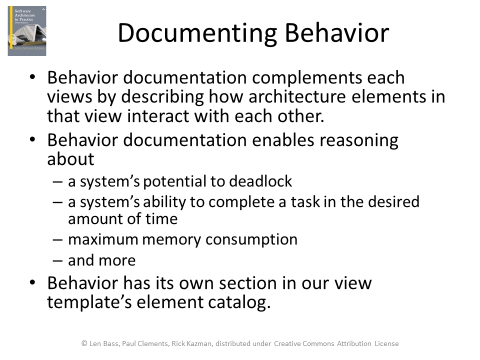








对系统行为编写文档

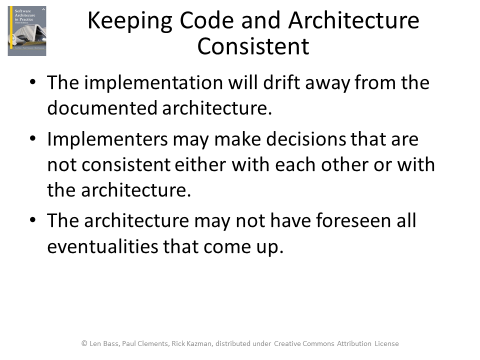


对质量属性编写文档

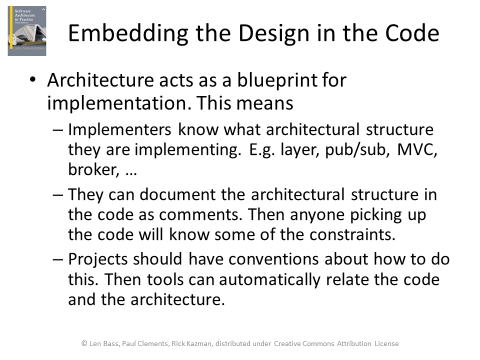


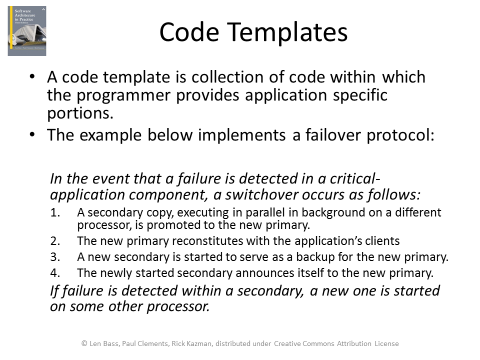
Chapter 19. Architecture, Implementation, and Testing

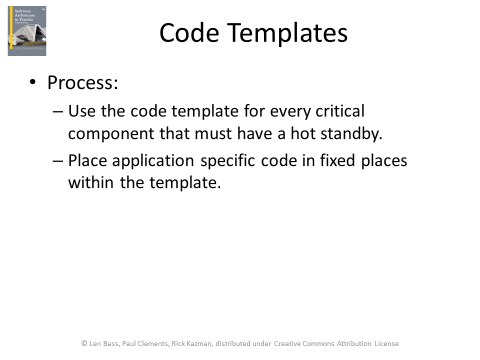
理解：实现与架构的一致性。

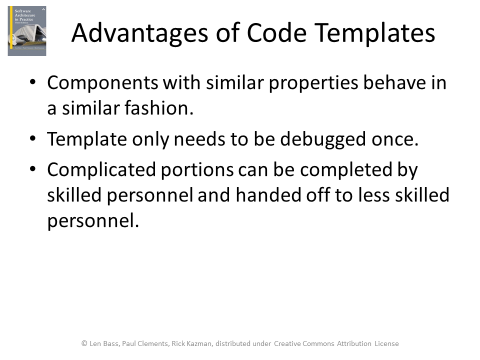


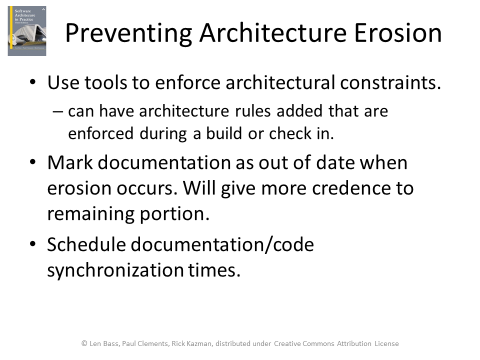
掌握：将架构嵌入代码。框架方法。代码模版方法。防止架构侵蚀。



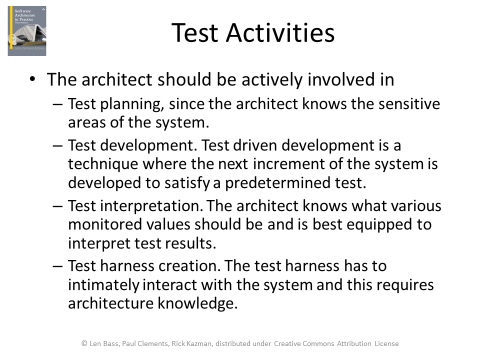






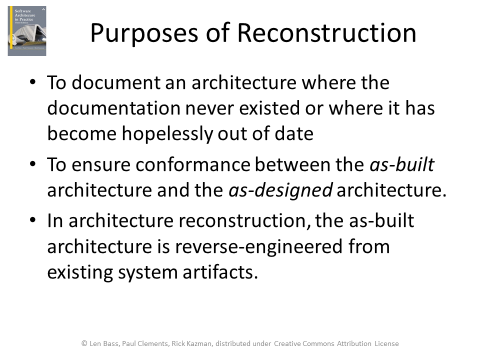


了解：架构师在测试中的角色。

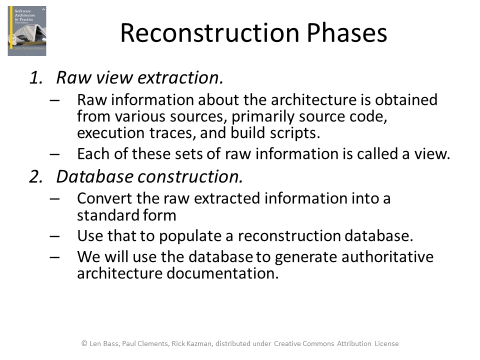


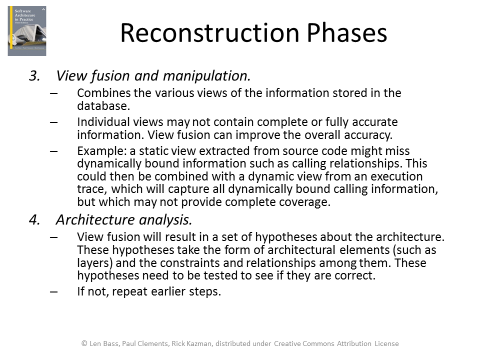
Chapter 20. Architecture Reconstruction and Conformance

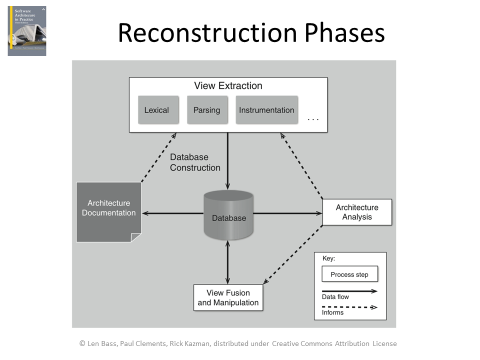
了解：架构重构的背景和目的。



理解：架构重构的阶段。每个阶段的方法。

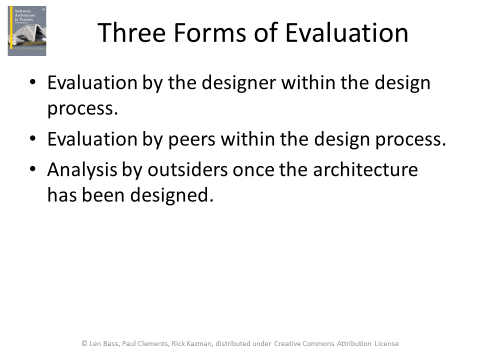


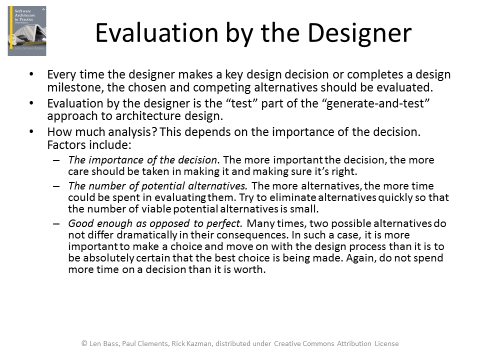


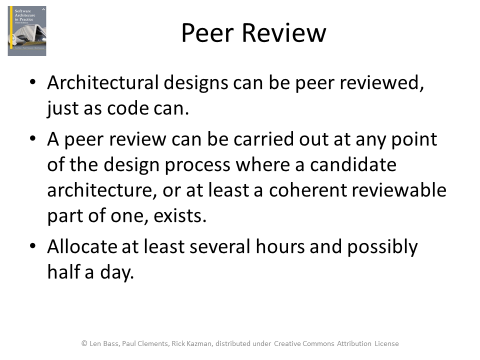


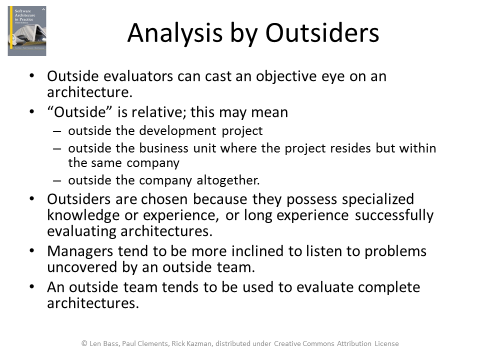
Chapter 21. Architecture Evaluation

了解：架构评审的3种形式及其特点。轻量级架构评审。

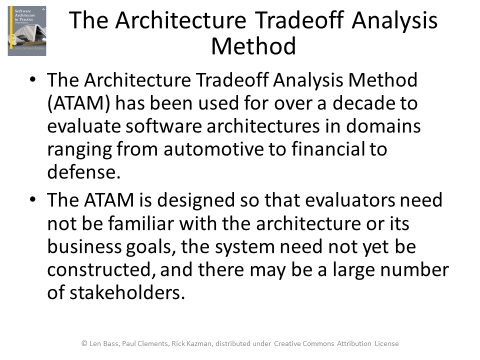


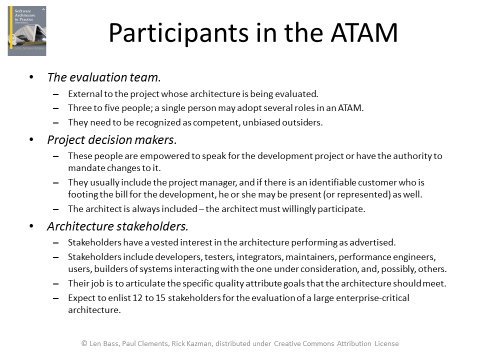


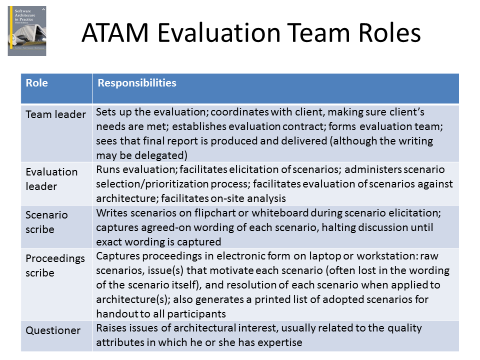


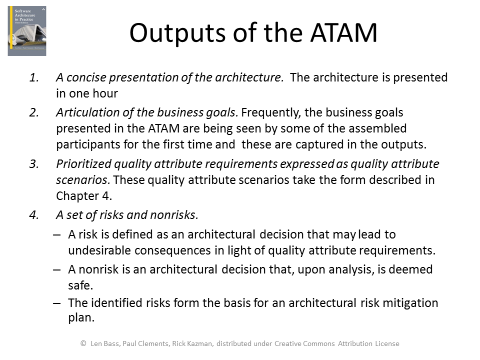


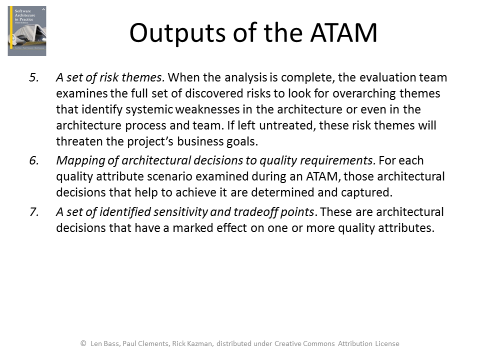
掌握：ATAM方法：目的、参与人员、步骤、采用的方法、结论。











步骤

Step 1: Present the ATAM

Step 2: Present Business Drivers

Step 3: Present the Architecture

Step 4: Identify Architectural Approaches

Step 5: Generate Utility Tree

Step 6: Analyze Architectural Approaches

Step 7: Brainstorm and Prioritize Scenarios

Step 8: Analyze Architectural Approaches

Step 9: Present Results