The Process of Software Architecting

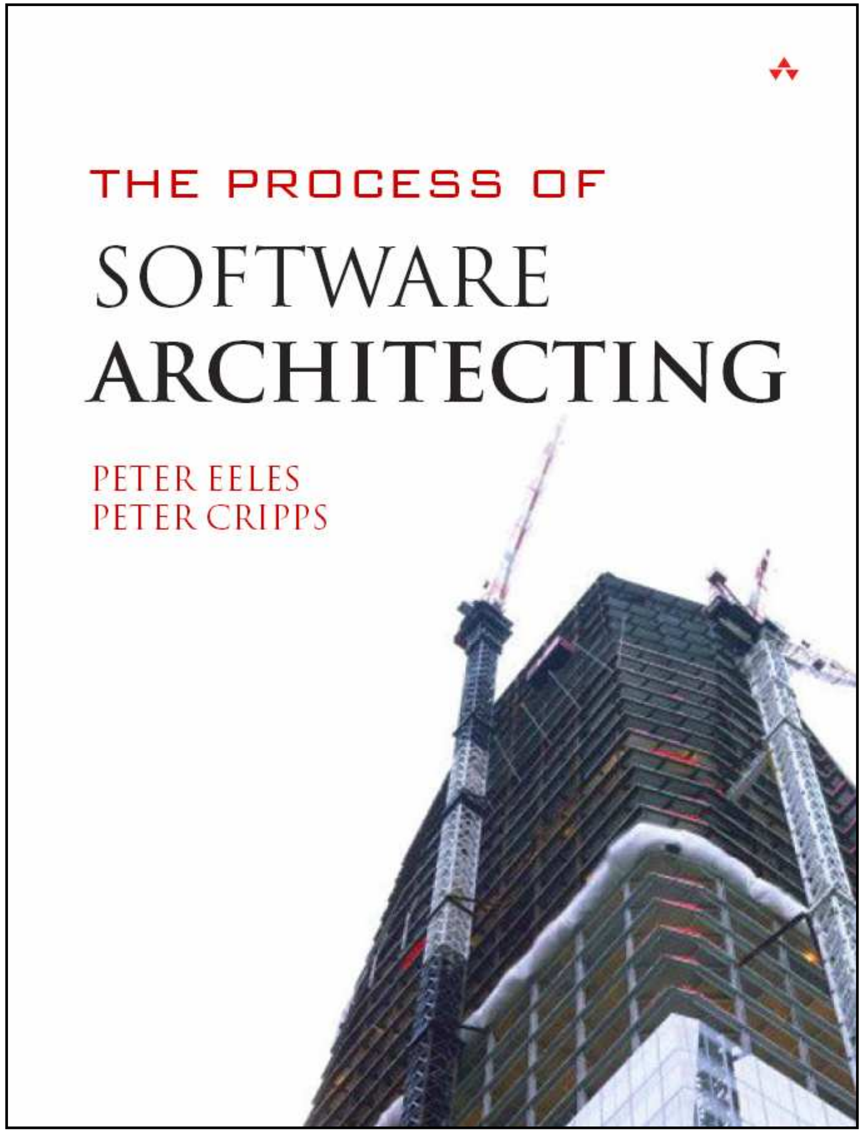
Peter Eeles

Executive IT Architect
IBM UK
peter.eeles@uk.ibm.com

Agenda

- ➡ Introduction
- Architecture, Architect, Architecting
- Method fundamentals
- Documenting a software architecture
- Reusable architecture assets
- A day in the life
- Summary

Coming Soon!

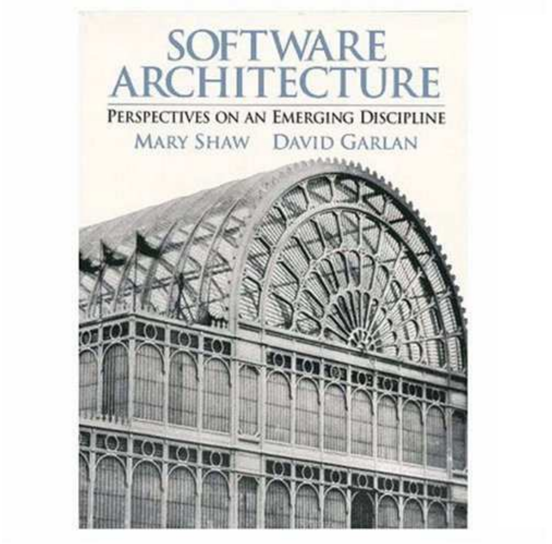

THE PROCESS OF

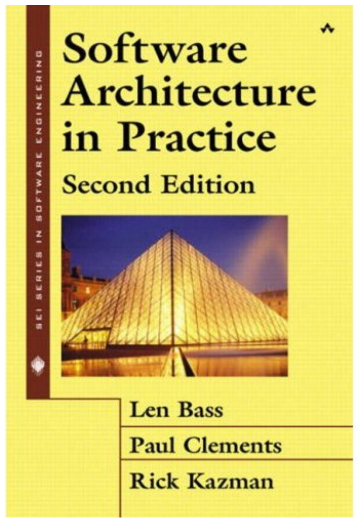
SOFTWARE ARCHITECTING

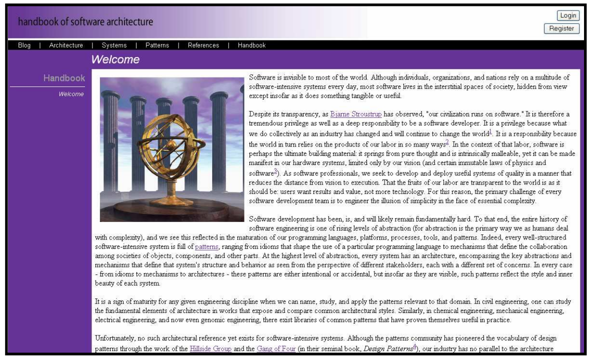
PETER EELES
PETER CRIPPS

Inspiration

- *“If I have seen further it is only by standing on the shoulders of Giants”*
  - Sir Isaac Newton, letter to Robert Hooke, 15th February 1676




Software Architecture in Practice
Second Edition
SEI SERIES IN SOFTWARE ENGINEERING
Len Bass
Paul Clements
Rick Kazman


handbook of software architecture

Login
Register

Blog | Architecture | Systems | Patterns | References | Handbook

Handbook
Welcome

Welcome

Software is invisible to most of the world. Although individuals, organizations, and nations rely on a multitude of software-intensive systems every day, most software lives in the interstitial spaces of society, hidden from view except insofar as it does something tangible or useful.

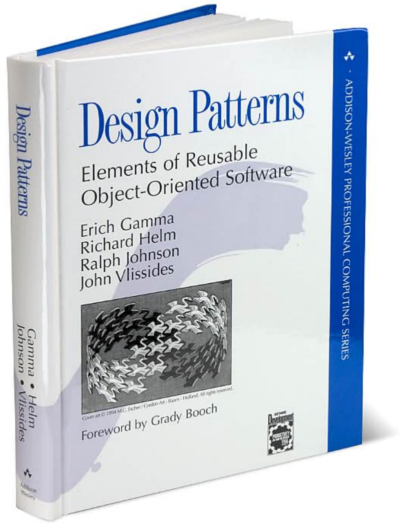
Despite its transparency, as Bjarne Stroustrup has observed, "our civilization runs on software." It is therefore a tremendous privilege as well as a deep responsibility to be a software developer. It is a privilege because what we do collectively as an industry has changed and will continue to change the world[1]. It is a responsibility because the world in turn relies on the products of our labor in so many ways[2]. In the context of that labor, software is perhaps the ultimate building material: it springs from pure thought and is intrinsically malleable, yet it can be made manifest in our hardware systems, limited only by our vision (and certain immutable laws of physics and software[3]). As software professionals, we seek to develop and deploy useful systems of quality in a manner that reduces the distance from vision to execution. That the fruits of our labor are transparent to the world is as it should be: users want results and value, not more technology. For this reason, the primary challenge of every software development team is to engineer the illusion of simplicity in the face of essential complexity.

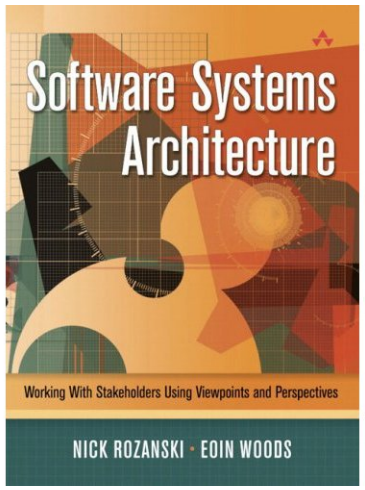
Software development has been, is, and will likely remain fundamentally hard. To that end, the entire history of software engineering is one of rising levels of abstraction (for abstraction is the primary way we as humans deal with complexity), and we see this reflected in the maturation of our programming languages, platforms, processes, tools, and patterns. Indeed, every well-structured software-intensive system is full of patterns, ranging from idioms that shape the use of a particular programming language to mechanisms that define the collaboration among societies of objects, components, and other parts. At the highest level of abstraction, every system has an architecture, encompassing the key abstractions and mechanisms that define that system's structure and behavior as seen from the perspective of different stakeholders, each with a different set of concerns. In every case - from idioms to mechanisms to architectures - these patterns are either intentional or accidental, but insofar as they are visible, such patterns reflect the style and inner beauty of each system.

It is a sign of maturity for any given engineering discipline when we can name, study, and apply the patterns relevant to that domain. In civil engineering, one can study the fundamental elements of architecture in works that expose and compare common architectural styles. Similarly, in chemical engineering, mechanical engineering, electrical engineering, and now even genomic engineering, there exist libraries of common patterns that have proven themselves useful in practice.

Unfortunately, no such architectural reference yet exists for software-intensive systems. Although the patterns community has pioneered the vocabulary of design patterns through the work of the Hillside Group and the Gang of Four (in their seminal book, *Design Patterns*[4]), our industry has no parallel to the architecture

~~www.booch.com/architecture~~
www.handbookofsoftwarearchitecture.com

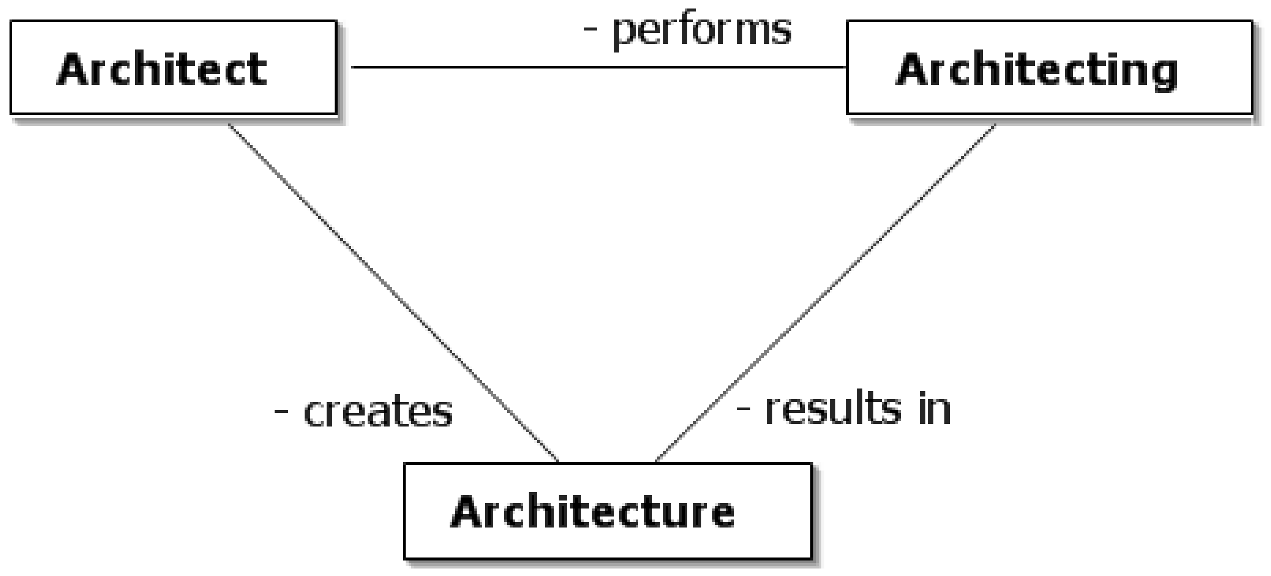



Software Systems Architecture
Working With Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI · EOIN WOODS

Agenda

- Introduction
- ⇨ Architecture, Architect, Architecting
- Method fundamentals
- Documenting a software architecture
- Reusable architecture assets
- A day in the life
- Summary

Core Concepts



Architecture

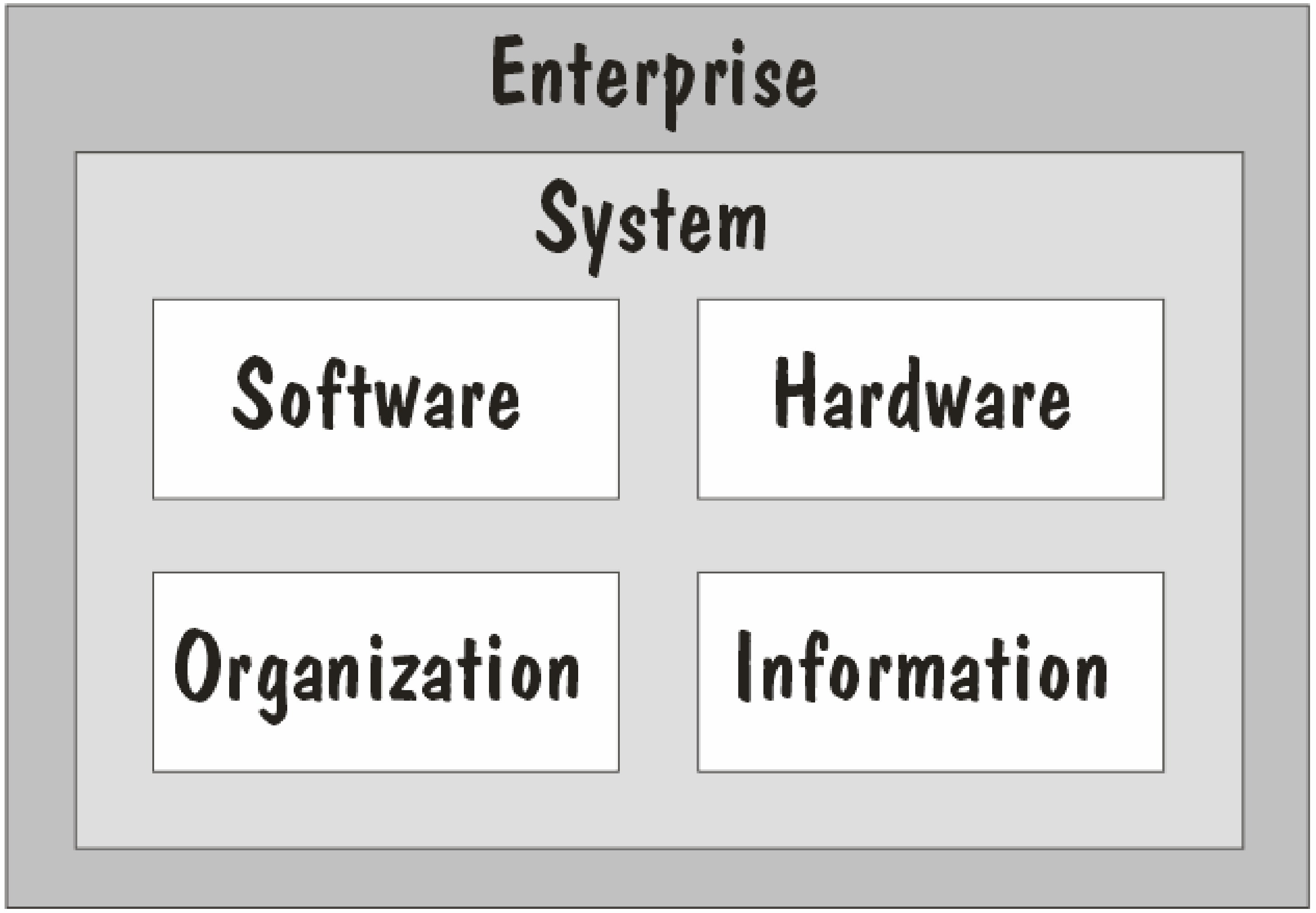
- *Architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. [IEEE 1471]*
- *The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [Bass]*
- *[Architecture is] the organizational structure and associated behavior of a system. An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include classes, components and subsystems. [UML 1.5]*

Architecture versus Design

All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.

- Grady Booch

An architecture has a particular scope



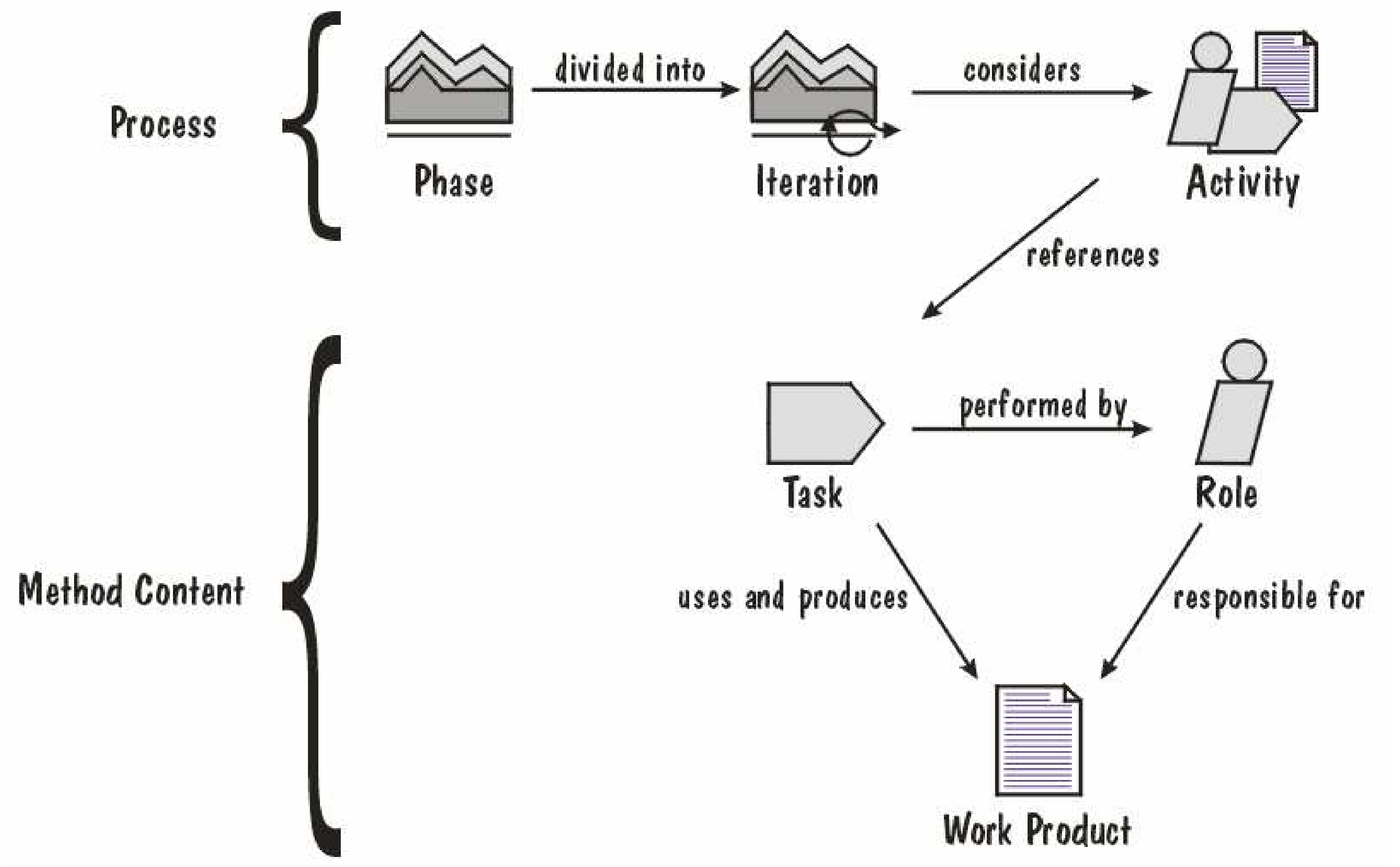
The benefits of architecting

- Architecting helps manage complexity
- Architecting ensures architectural integrity
- Architecting provides a basis for reuse
- Architecting addresses system qualities
- Architecting drives consensus
- Architecting reduces maintenance costs
- Architecting supports impact analysis
- Architecting supports the planning process

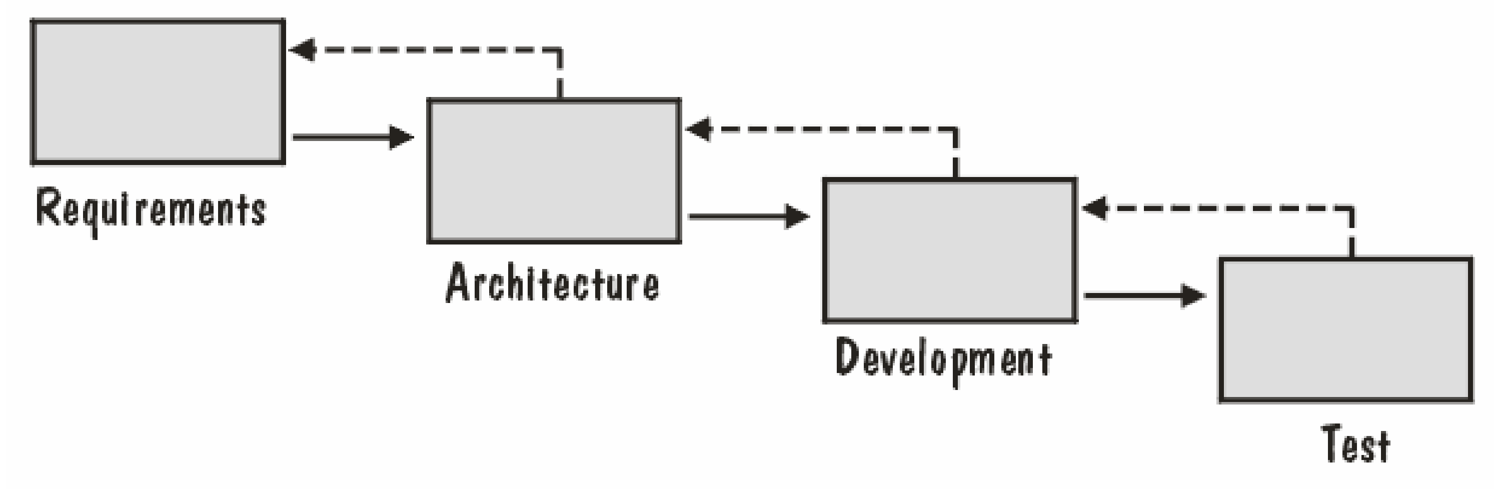
Agenda

- Introduction
- Architecture, Architect, Architecting
- Method fundamentals
- Documenting a software architecture
- Reusable architecture assets
- A day in the life
- Summary

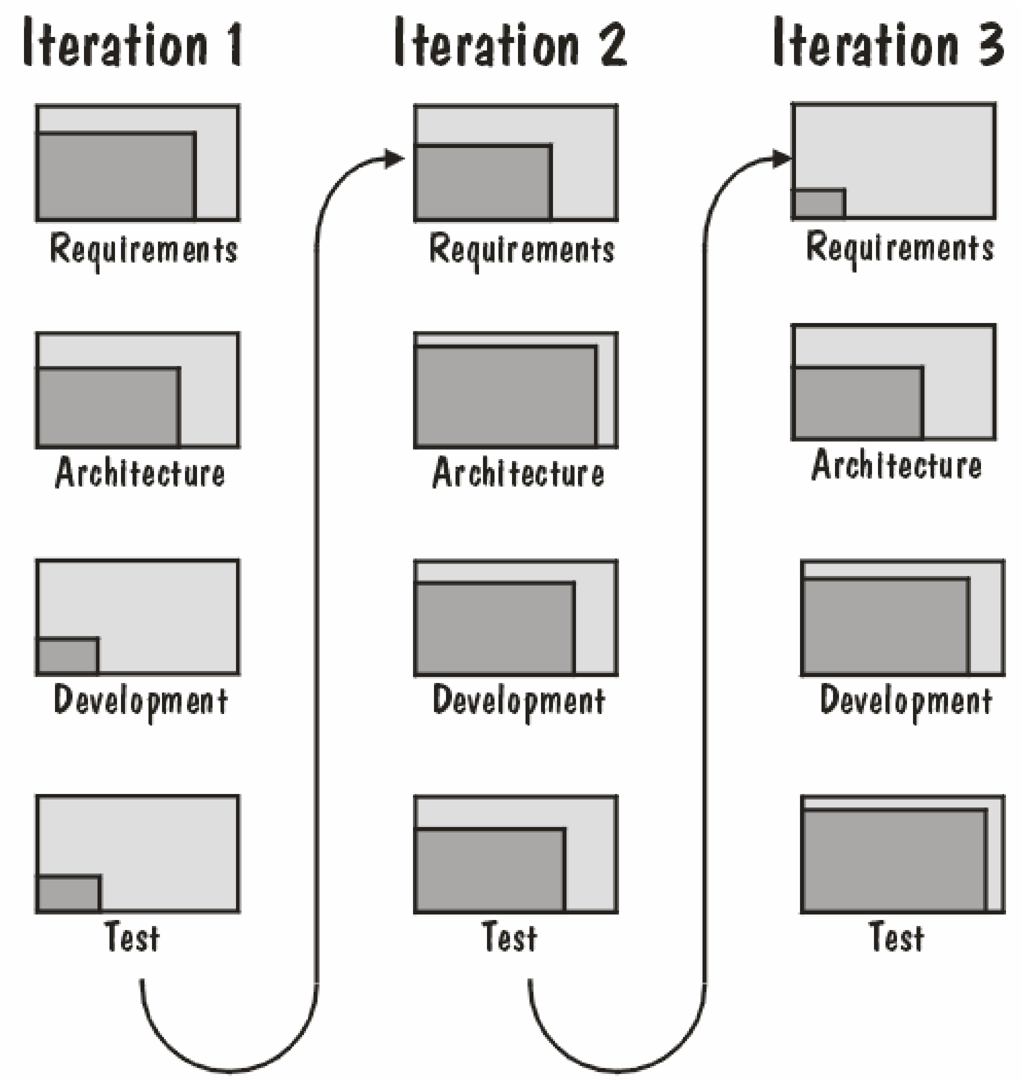
Key Method Concepts



A Waterfall Process



An Iterative Process



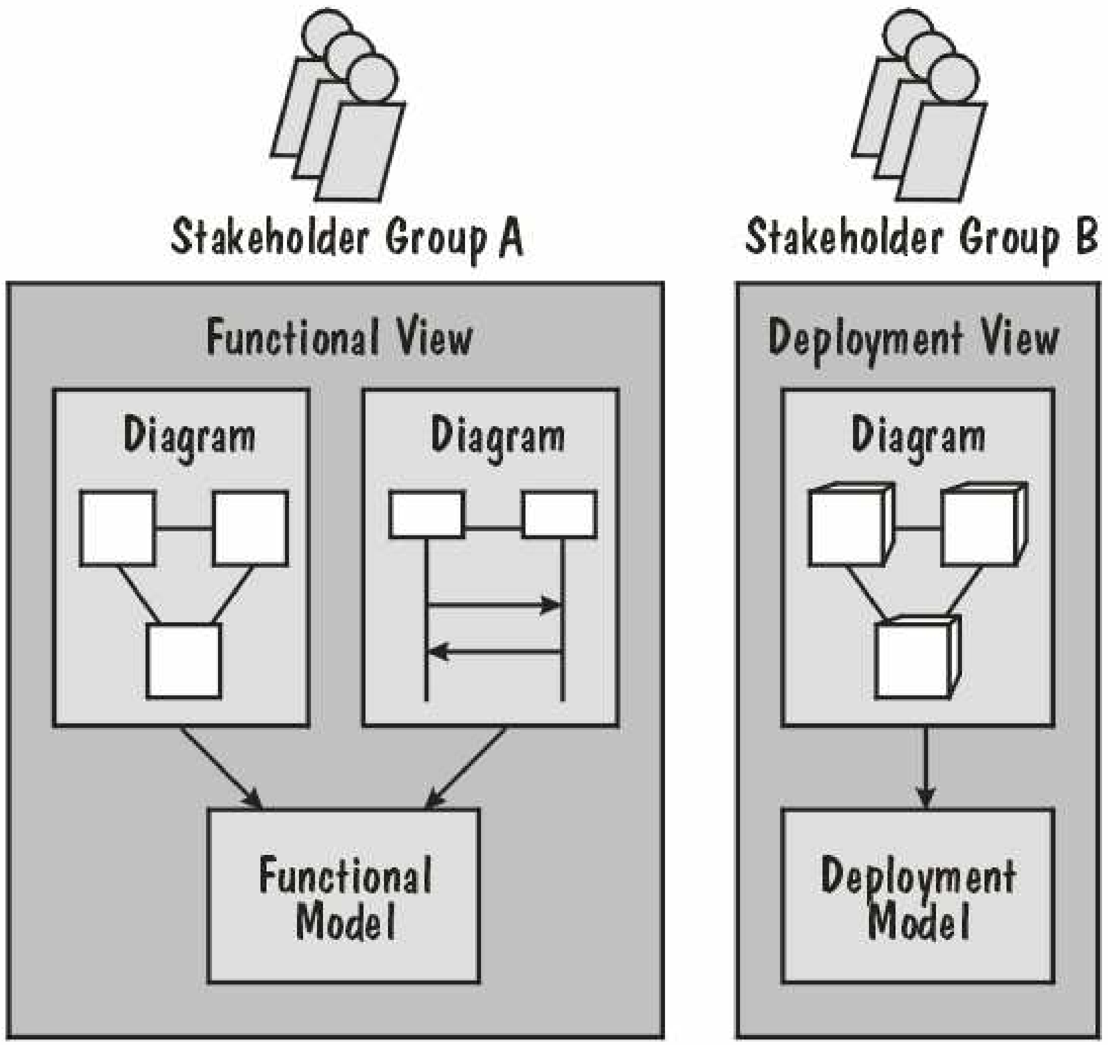
Agile

- Agile Manifesto
  - Individuals and interactions over processes and tools.
  - Working software over comprehensive documentation.
  - Customer collaboration over contract negotiation.
  - Responding to change over following a plan.
- *Scrum is a management and control process that cuts through complexity to focus on building software to meet business needs. Scrum is <u>superimposed on top of and wraps existing engineering practices, development methodologies and standards</u>. [Schwaber]*

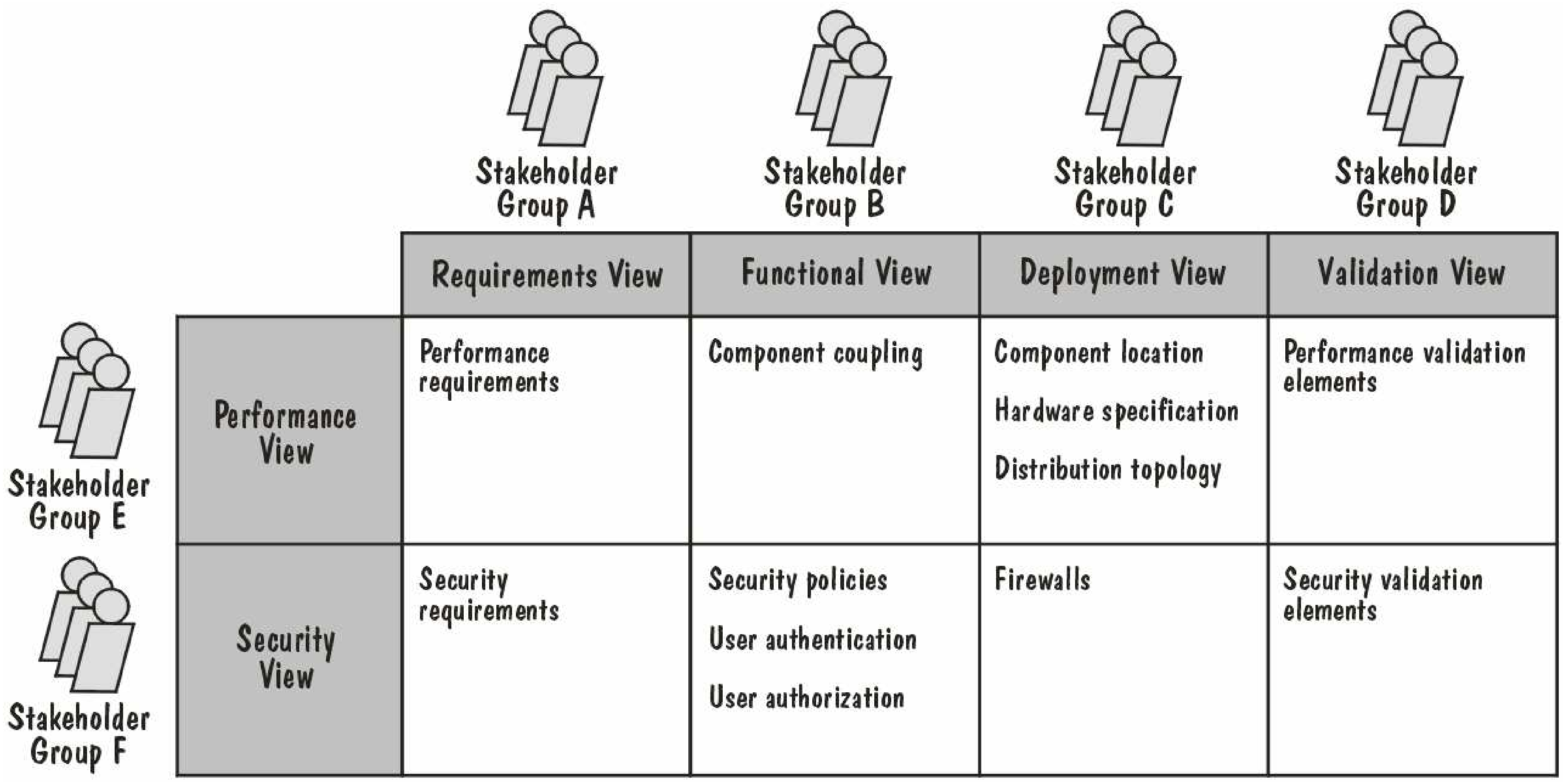
Agenda

- Introduction
- Architecture, Architect, Architecting
- Method fundamentals
- ➡ Documenting a software architecture
- Reusable architecture assets
- A day in the life
- Summary

Views, Diagrams and Models

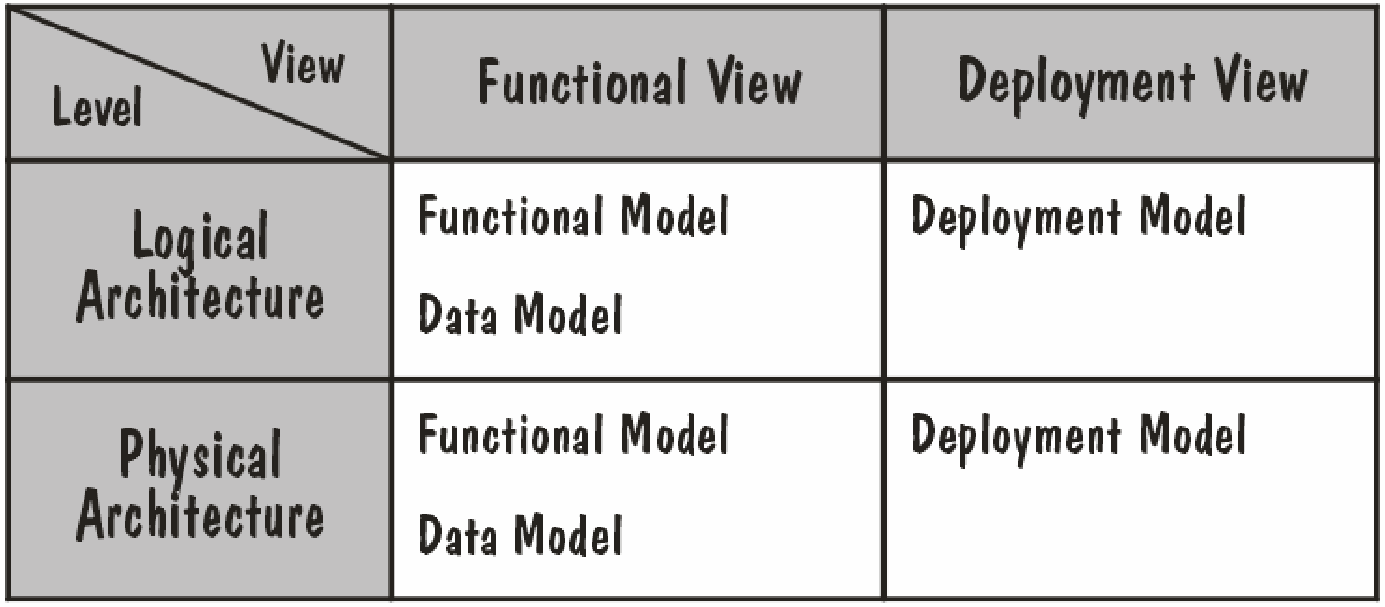


Basic Views and Cross-Cutting Views



	Requirements View	Functional View	Deployment View	Validation View
Performance View	Performance requirements	Component coupling	Component location Hardware specification Distribution topology	Performance validation elements
Security View	Security requirements	Security policies User authentication User authorization	Firewalls	Security validation elements

Views, Models and Levels of Realization

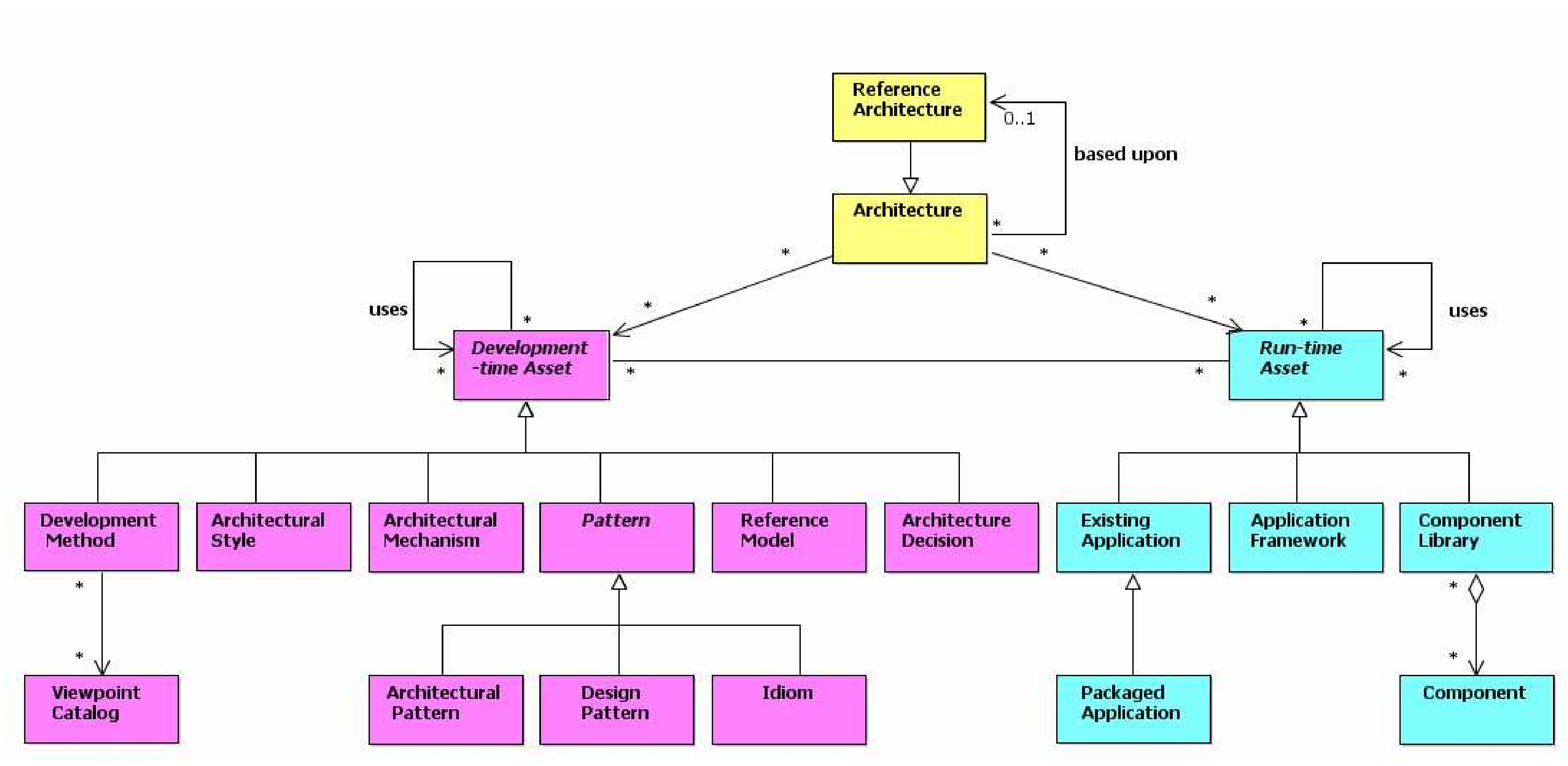


View / Level	Functional View	Deployment View
Logical Architecture	Functional Model Data Model	Deployment Model
Physical Architecture	Functional Model Data Model	Deployment Model

Agenda

- Introduction
- Architecture, Architect, Architecting
- Method fundamentals
- Documenting a software architecture
- ➡ Reusable architecture assets
- A day in the life
- Summary

A Metamodel of Architecture Assets

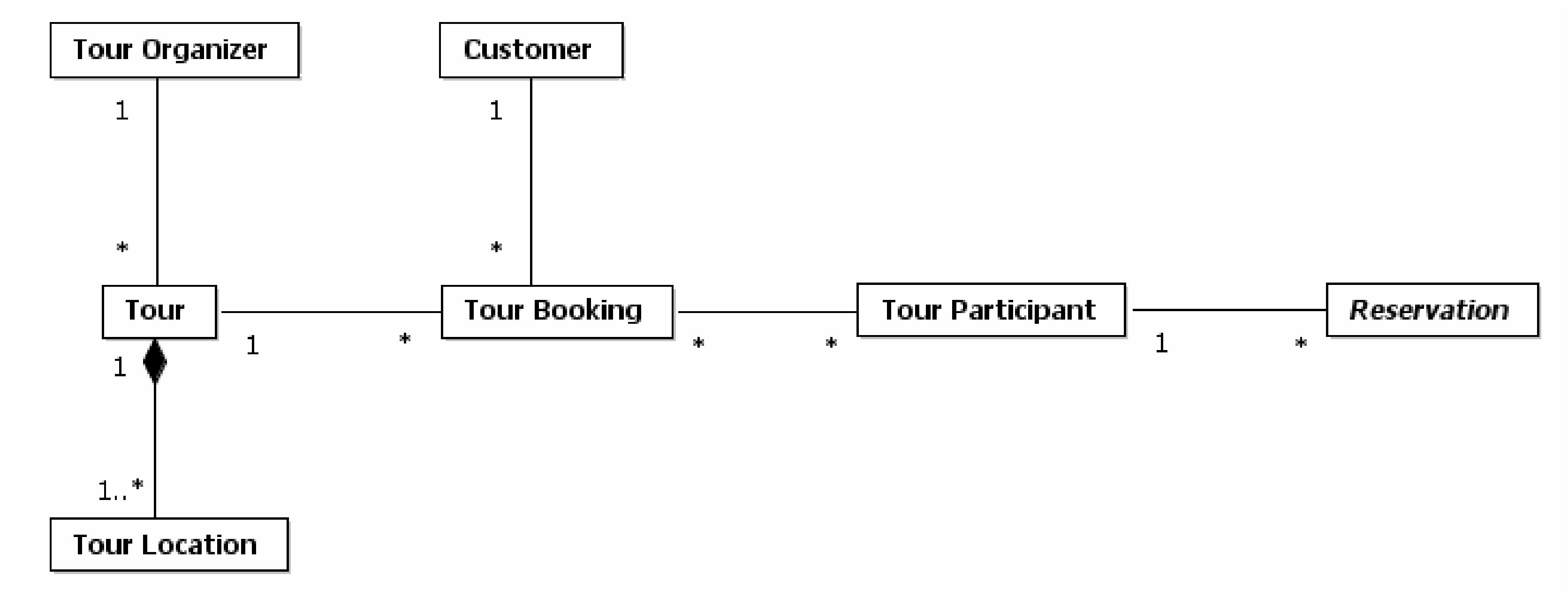


Agenda

- Introduction
- Architecture, Architect, Architecting
- Method fundamentals
- Documenting a software architecture
- Reusable architecture assets
- ➡ A day in the life
- Summary

Inputs

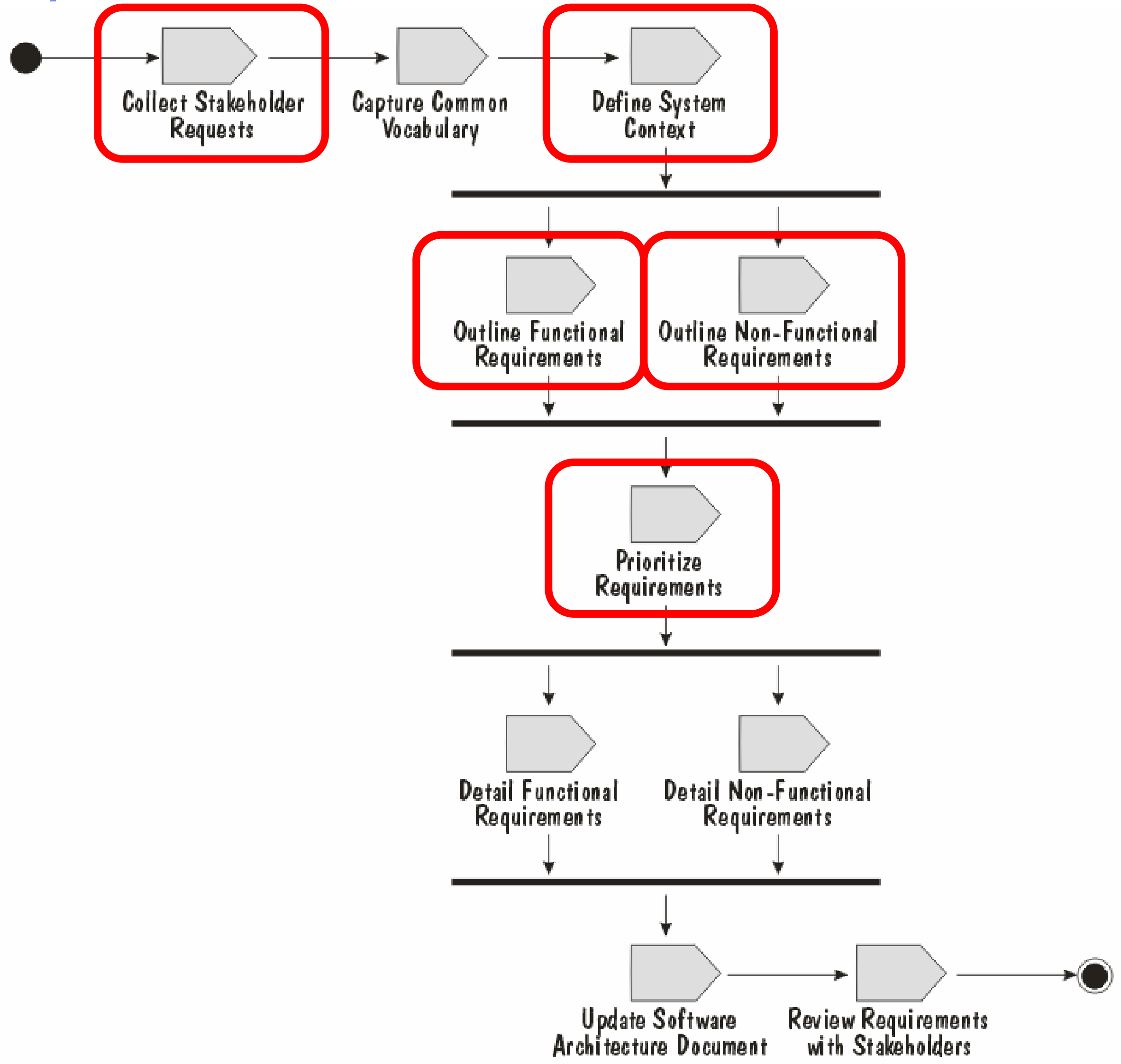
- Business Entity Model
- Business Process Model
- Business Rules
- Existing IT Environment
- Vision



Types of Requirements

- Functional requirements
  - *Describe the behaviors (functions or services) of the [IT] system that support user goals, tasks or activities. [Malan]*
- Non-functional requirements
  - *Non-functional requirements include constraints and qualities. [Malan]*
  - Constraint
    - *A constraint is a restriction on the degree of freedom we have in providing a solution. [Leffingwell]*
  - Quality
    - *[System] qualities are properties or characteristics of the system that its stakeholders care about and hence will affect their degree of satisfaction with the system. [Malan]*

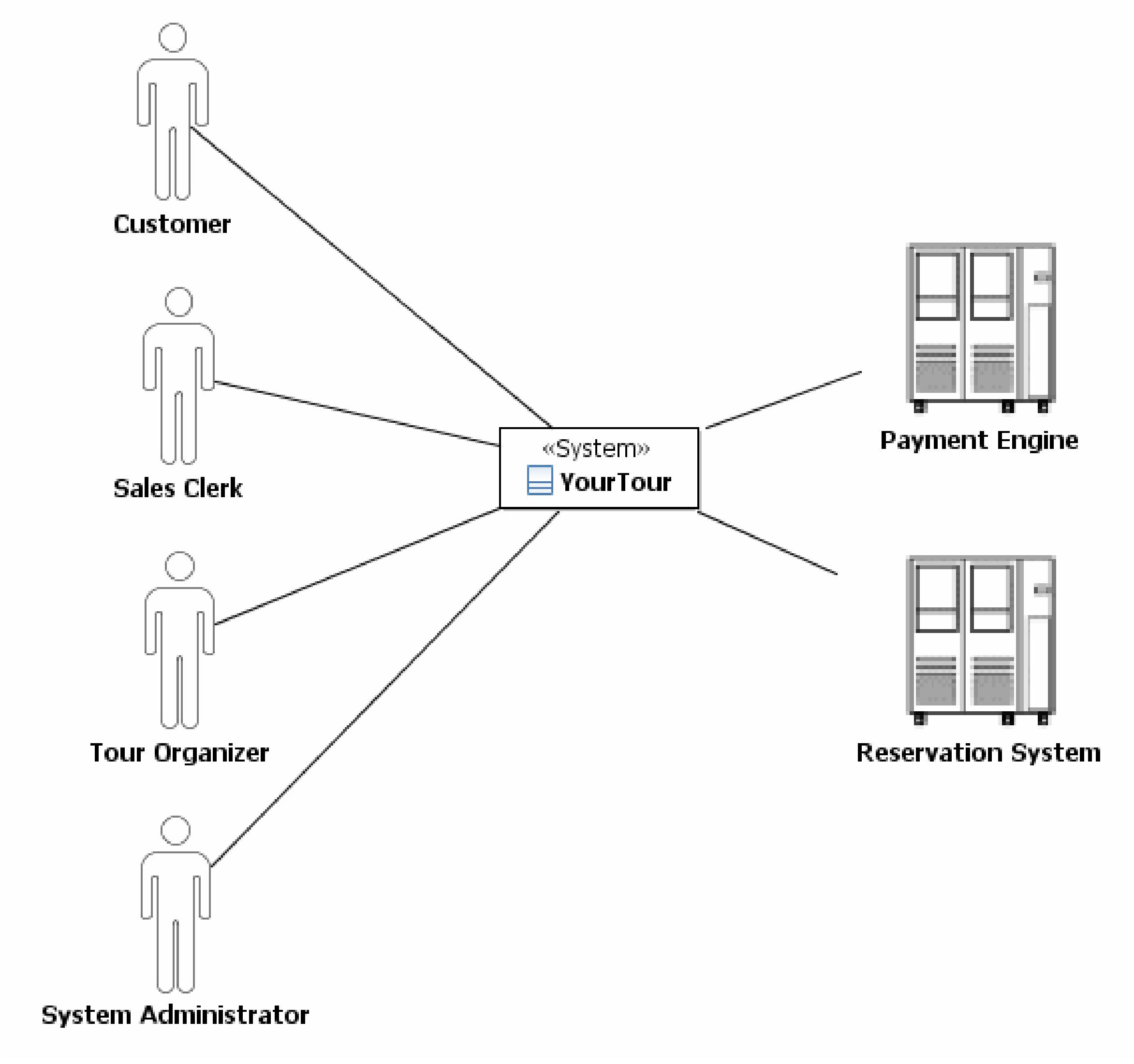
Define Requirements


Collect Stakeholder Requests
Capture Common Vocabulary
Define System Context
Outline Functional Requirements
Outline Non-Functional Requirements
Prioritize Requirements
Detail Functional Requirements
Detail Non-Functional Requirements
Update Software Architecture Document
Review Requirements with Stakeholders

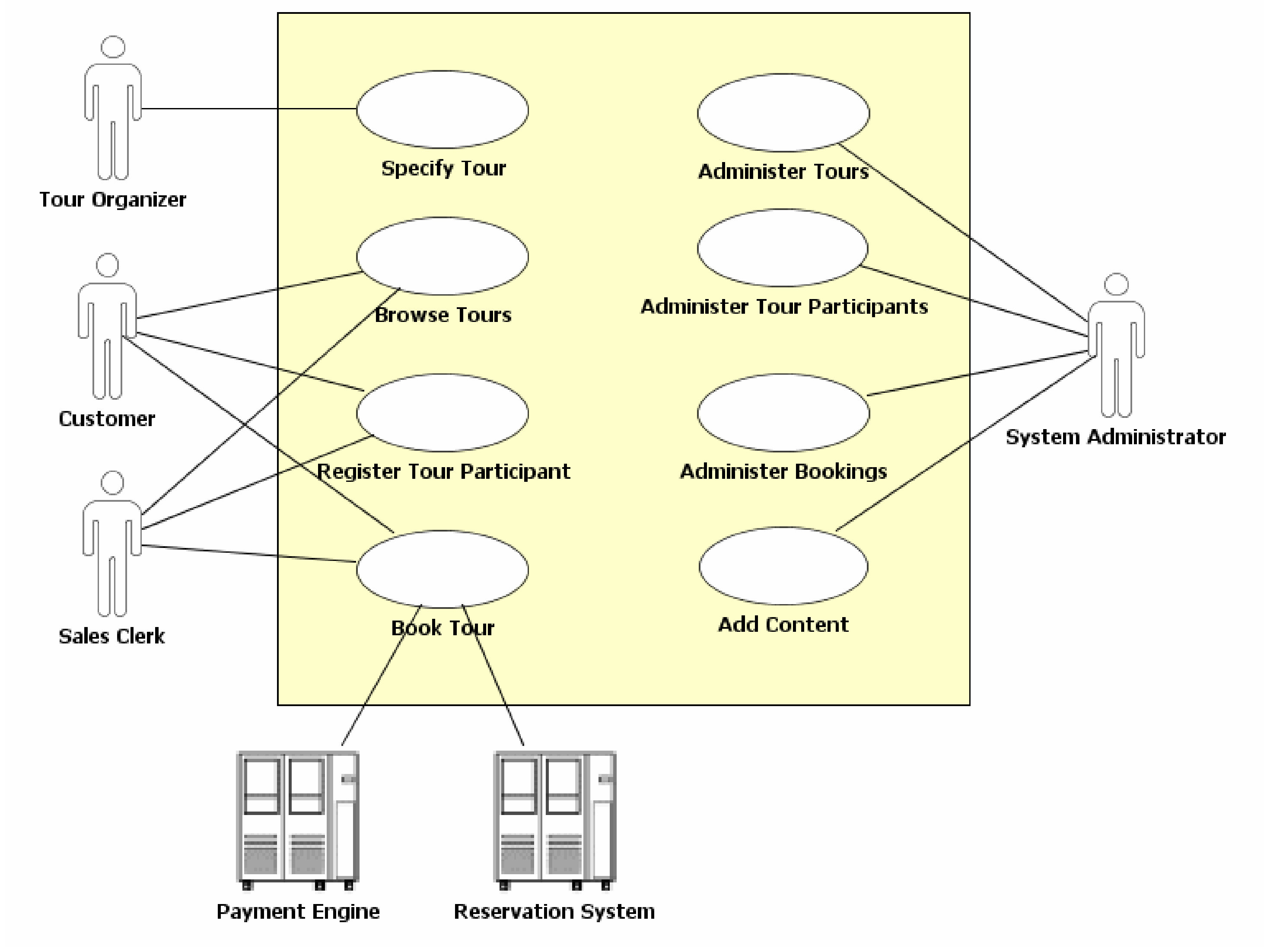
Task: Collect Stakeholder Requests

- Pitfall: Treating Requests as Requirements
- Pitfall: The Shopping Cart Mentality
- Pitfall: The Questions are too Technical
- Pitfall: Requests Are Too General
- Pitfall: Requests Are Not Measurable
- Pitfall: Talking with the Wrong People
- Pitfall: All Requests Are Equal

Task: Define System Context



Task: Outline Functional Requirements

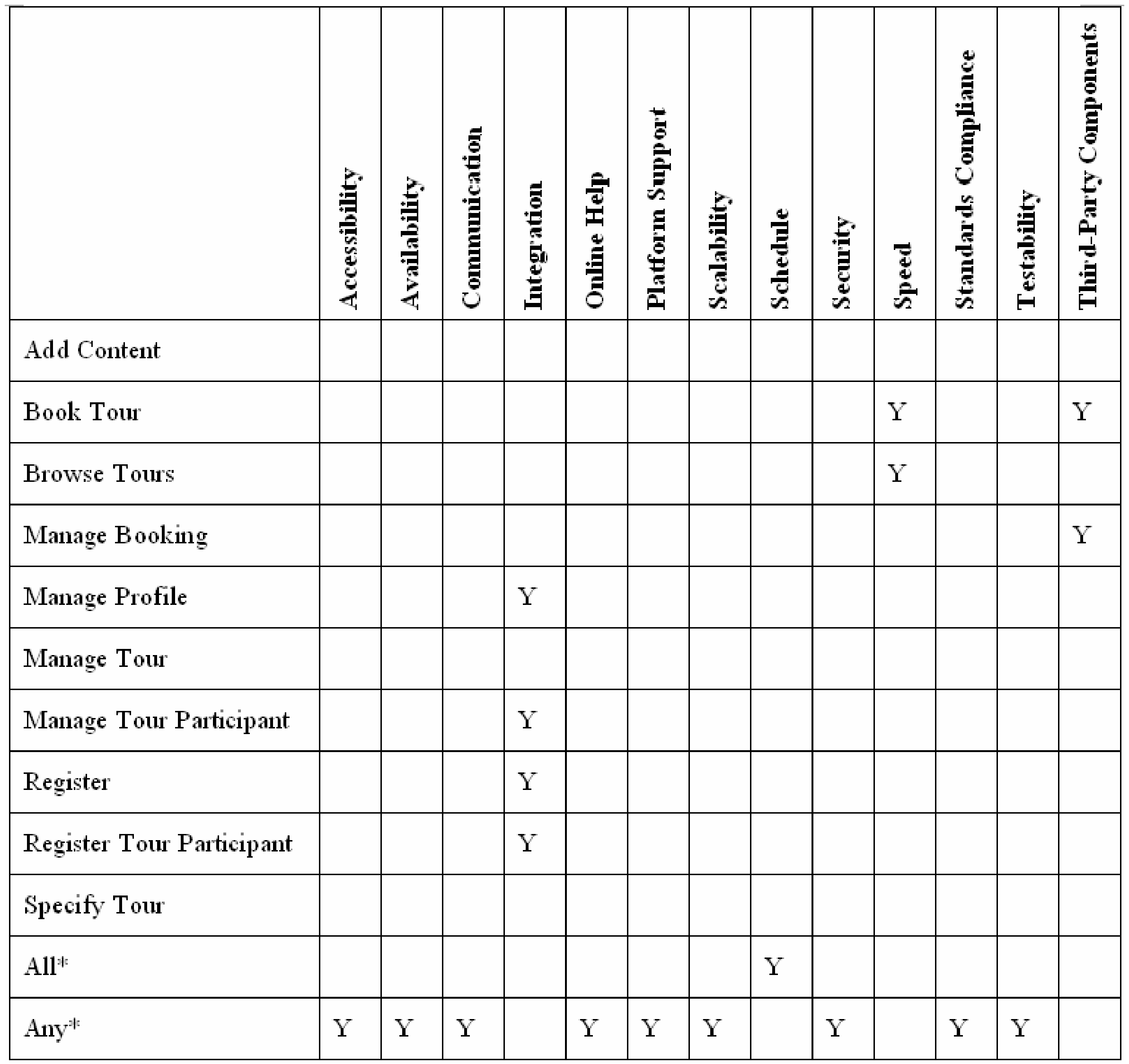


Task: Outline Non-Functional Requirements

- Usability Requirements
- Reliability Requirements
- Performance Requirements
- Supportability Requirements
- Constraints
  - Business Constraints
  - Architecture Constraints
  - Development Constraints
  - Physical Constraints

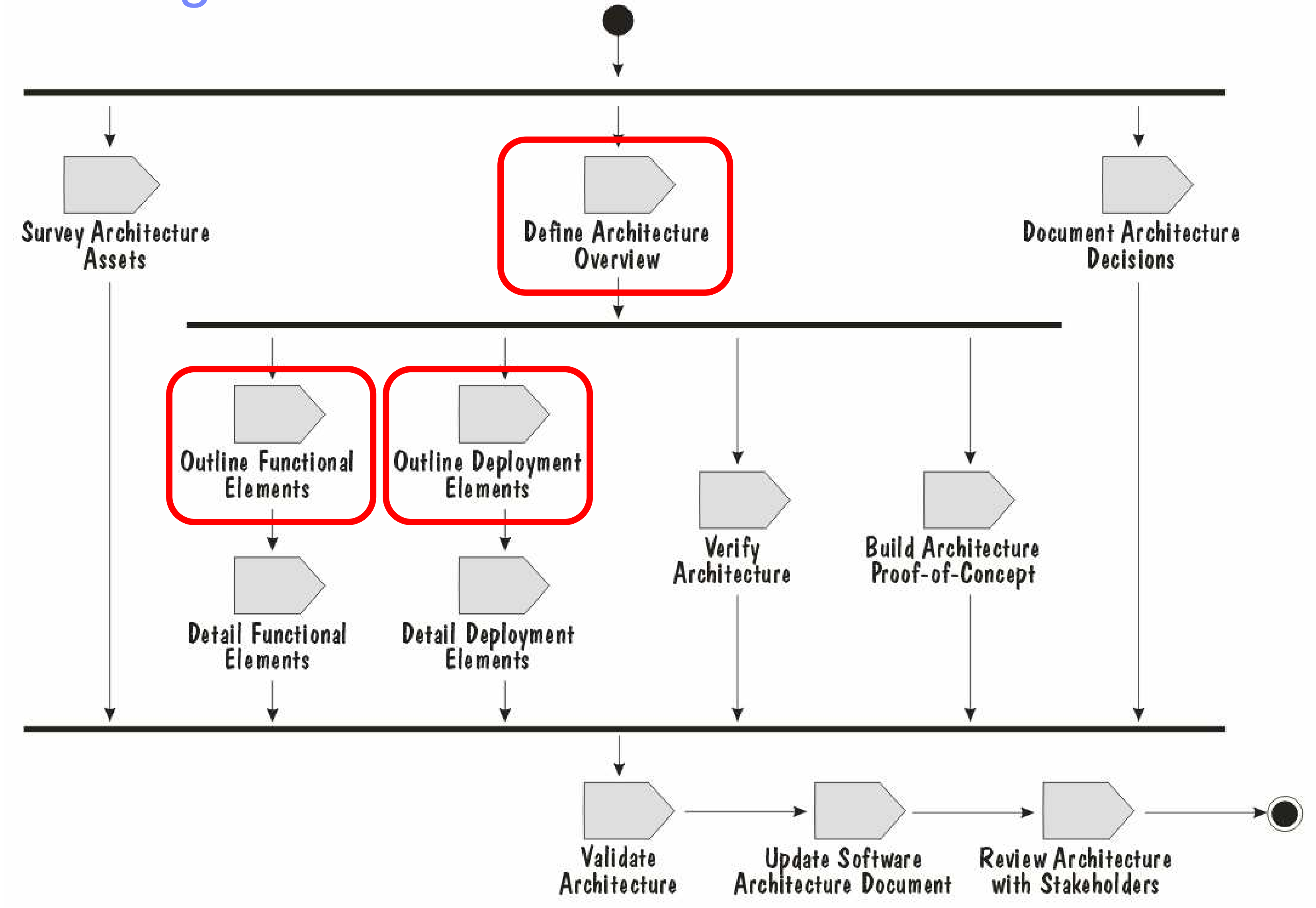
“Brownfield sites are those in which redevelopment or reuse of the site is complicated by existing contaminants. Greenfield sites are clean, previously undeveloped land”. [Hopkins]

Task: Prioritize Requirements

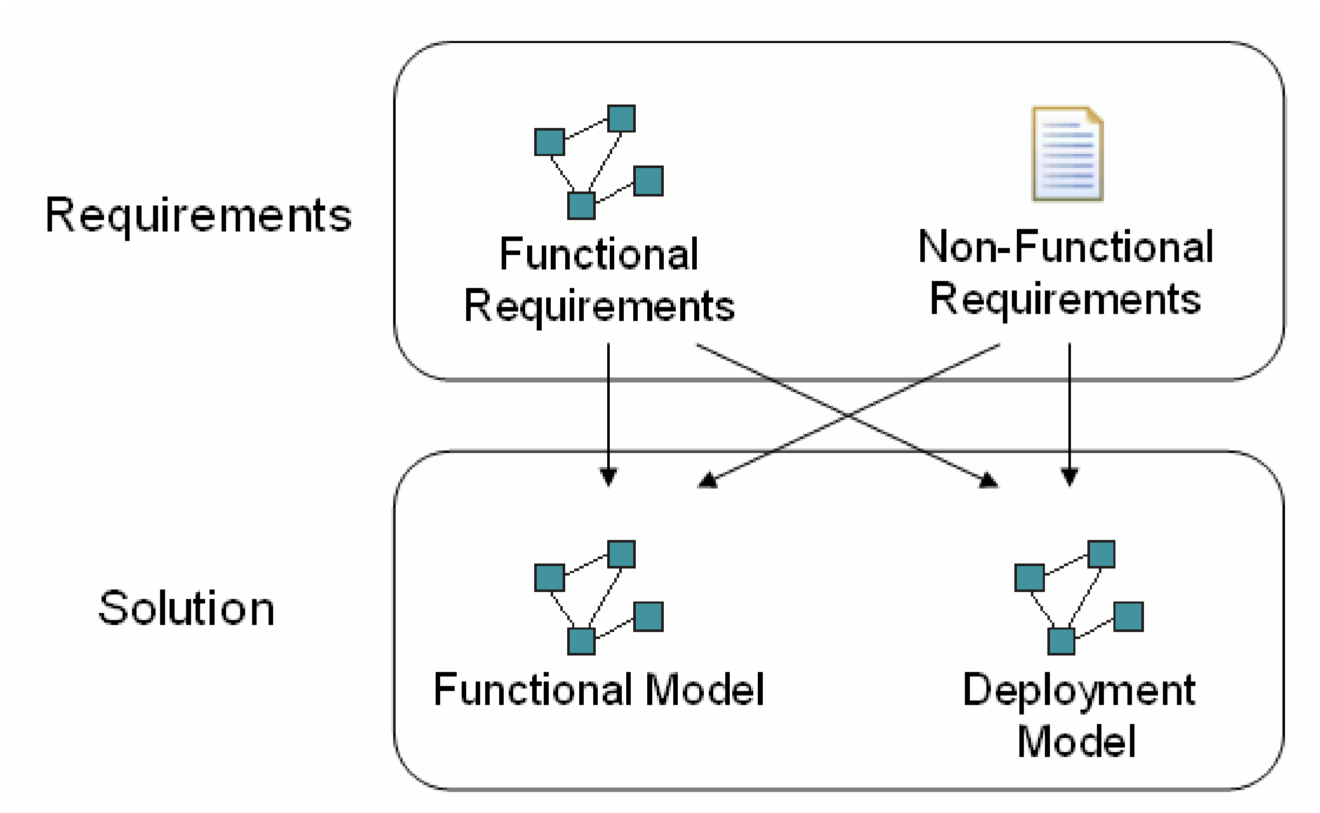


	Accessibility	Availability	Communication	Integration	Online Help	Platform Support	Scalability	Schedule	Security	Speed	Standards Compliance	Testability	Third-Party Components
Add Content													
Book Tour										Y			Y
Browse Tours										Y			
Manage Booking													Y
Manage Profile				Y									
Manage Tour													
Manage Tour Participant				Y									
Register				Y									
Register Tour Participant				Y									
Specify Tour													
All*								Y					
Any*	Y	Y	Y		Y	Y	Y		Y		Y	Y	

Create Logical Architecture


Survey Architecture Assets
Define Architecture Overview
Document Architecture Decisions
Outline Functional Elements
Outline Deployment Elements
Verify Architecture
Build Architecture Proof-of-Concept
Detail Functional Elements
Detail Deployment Elements
Validate Architecture
Update Software Architecture Document
Review Architecture with Stakeholders

From Requirements to Solution



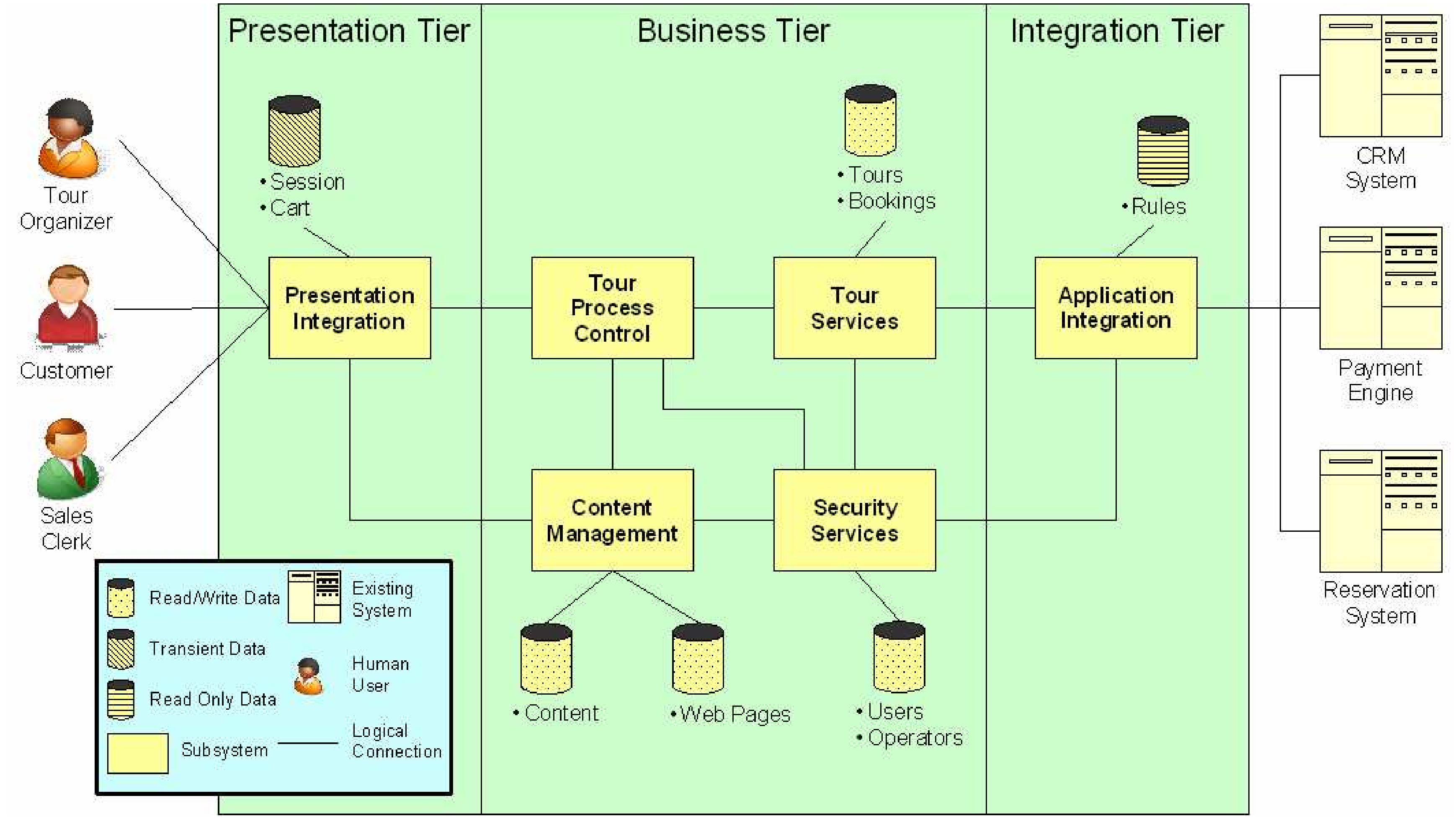
Approaches

- Attribute Driven Design (ADD) Method
  - Developed at the Software Engineering Institute
  - Quality attributes drive the derivation of the architecture
  - Underpinned by architectural tactics and patterns
- Siemens' 4 Views (S4V) method
  - Developed at Siemens Corporate Research
  - Starts with a global analysis of the factors that influence the architecture
  - Iteratively addresses challenges across four views (conceptual, execution, module and code architecture)
- The Rational Unified Process (RUP)
  - Developed at Rational Software (now IBM Rational)
  - Driven by architecturally-significant requirements
  - Each iteration considers the key architectural elements of the solution, before realizing the requirements using these solution elements

How Much Logical Architecture?

- Minimizing Logical Architecture
  - The logical architecture is simply a means of getting to a physical architecture as quickly as possible
  - In some cases, no logical architecture may be required at all
    - E.g. The requirements for the system are similar to those of an existing system
    - E.g. We are using a packaged application or integrating with an existing system
- Logical Architecture as an Investment
  - A valuable asset if a technology change is anticipated at some point in the future

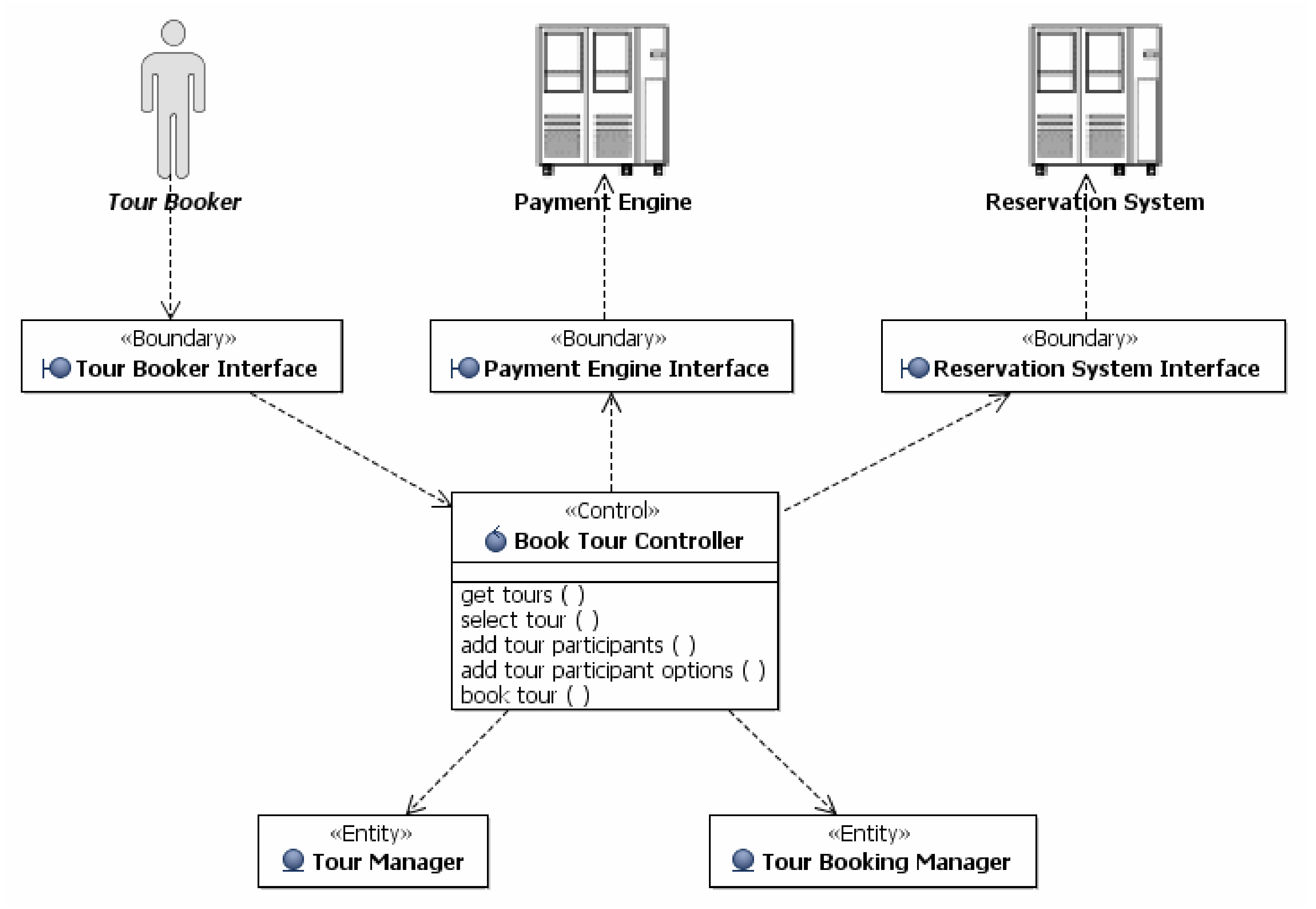
Task: Define Architecture Overview



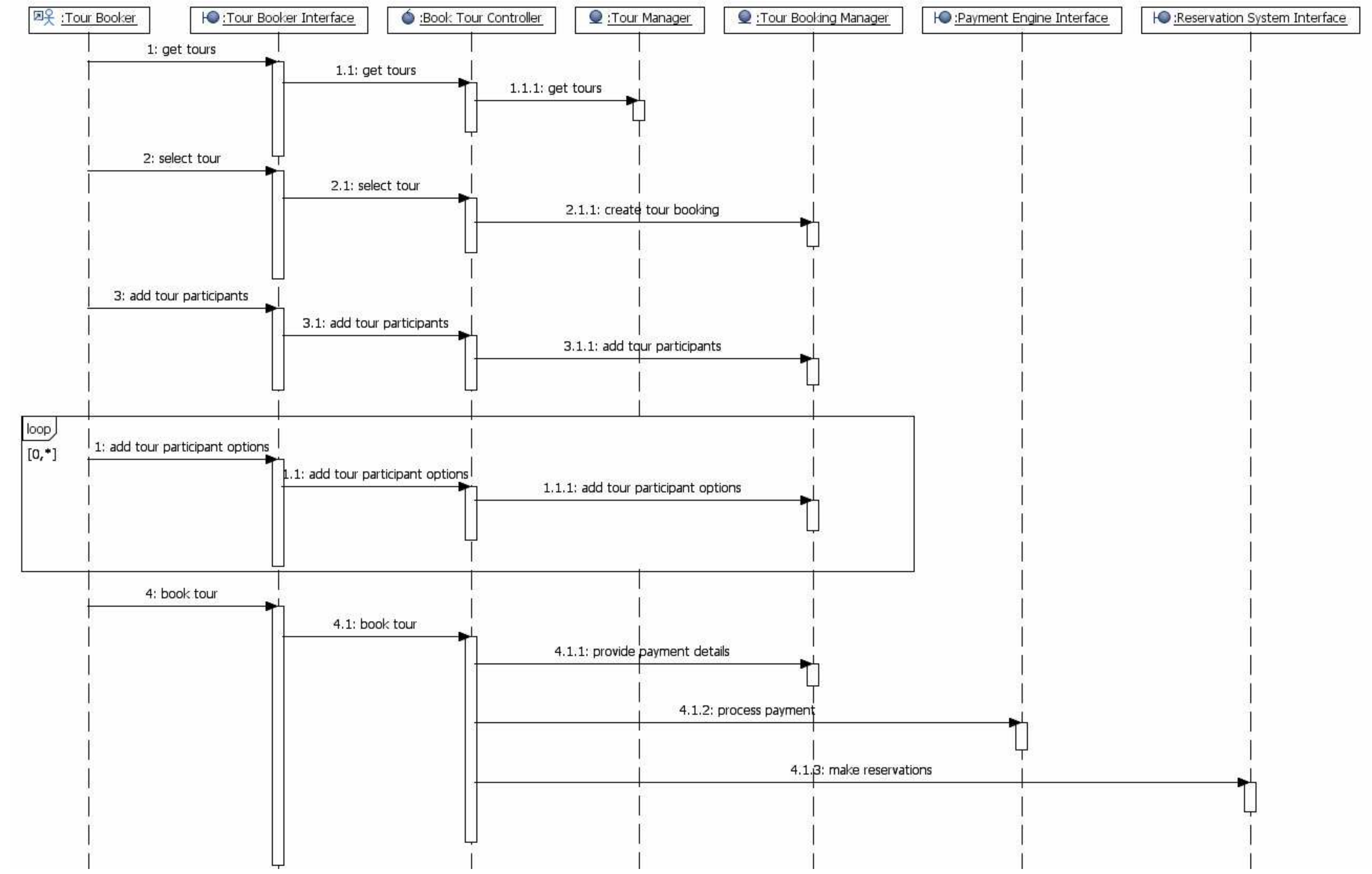
Task: Outline Functional Elements

- Component identification
  - Business Entity Model
    - Clustering of related entities
  - Functional requirements
    - Boundary, control and entity components
  - Non-functional requirements
    - Constraints
    - Components that address specific technical challenges (e.g. security)
  - Business rules
    - Business rules component(s)
  - Architecture decisions
    - Use of particular assets (e.g. packages, patterns)

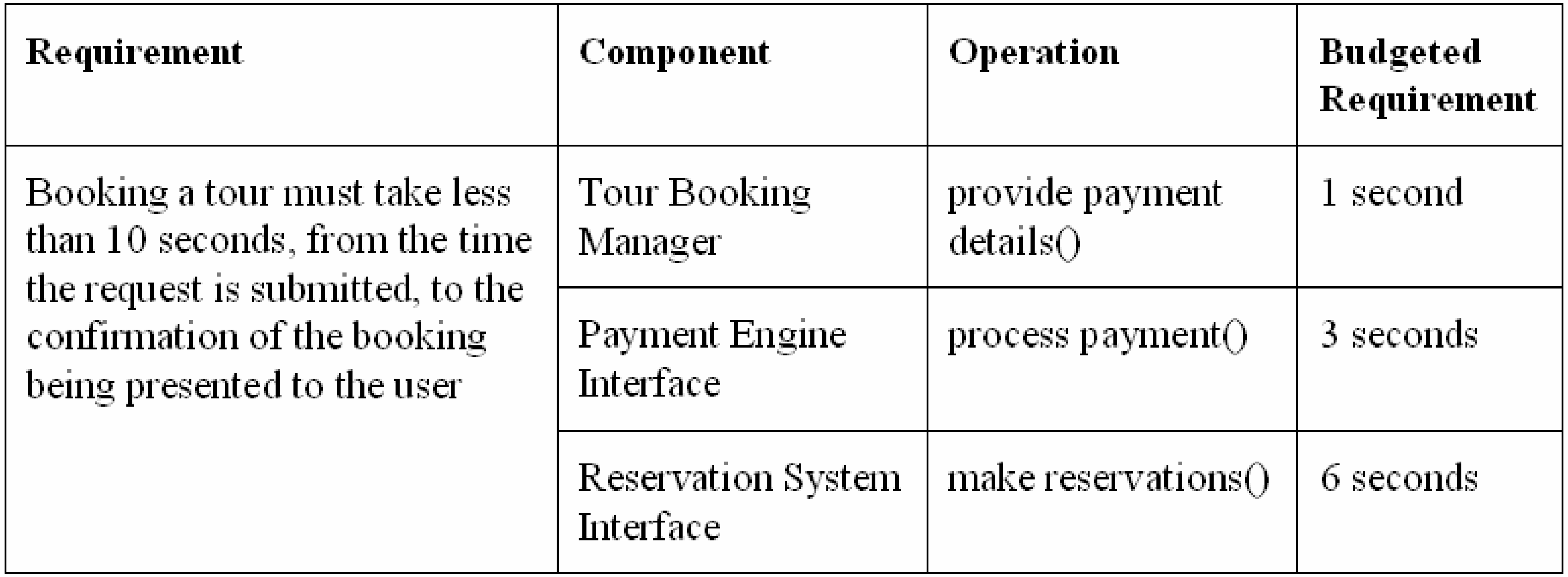
Task: Outline Functional Elements


Tour Booker
Payment Engine
Reservation System
«Boundary» Tour Booker Interface
«Boundary» Payment Engine Interface
«Boundary» Reservation System Interface
«Control» Book Tour Controller
get tours ( )
select tour ( )
add tour participants ( )
add tour participant options ( )
book tour ( )
«Entity» Tour Manager
«Entity» Tour Booking Manager

Task: Outline Functional Elements

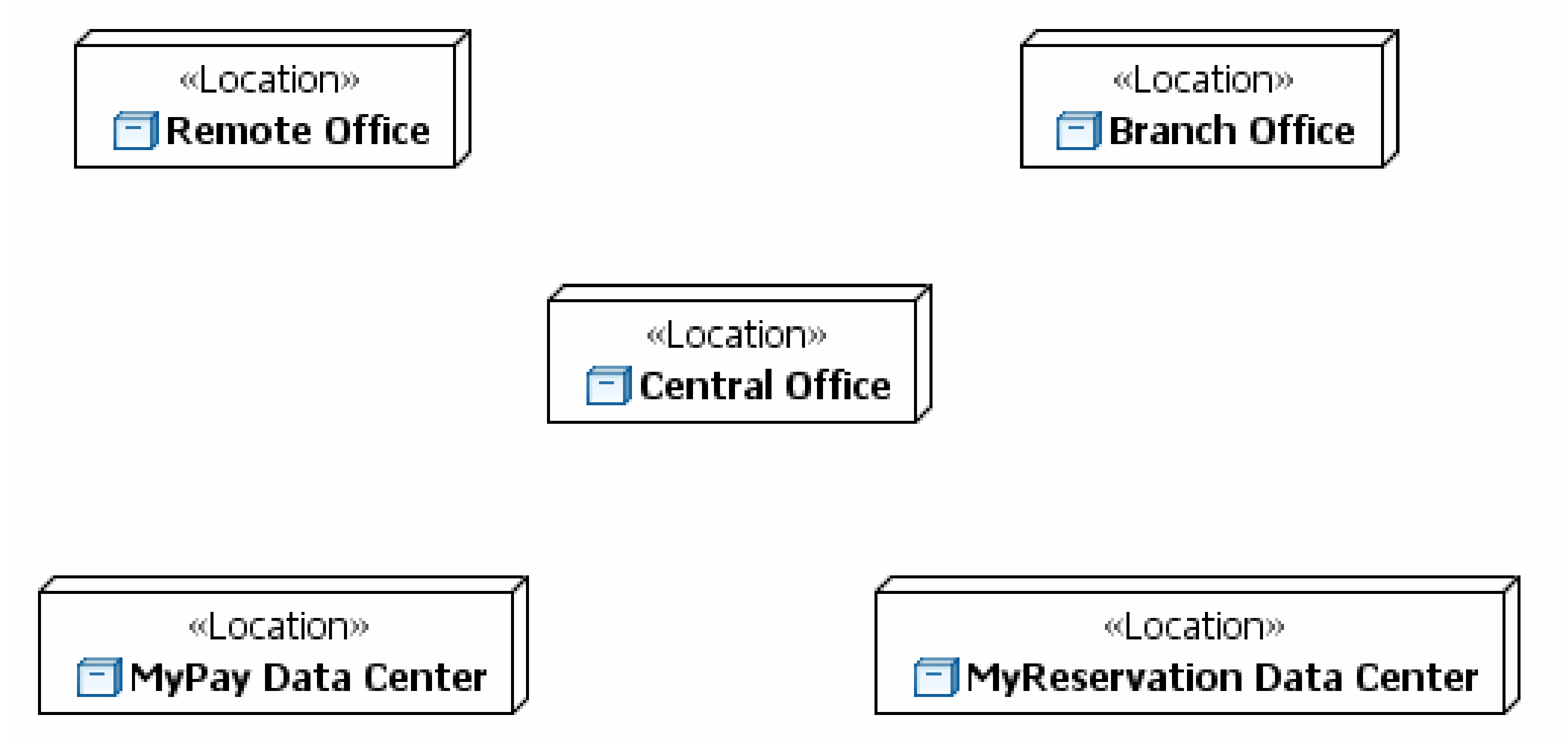


Assigning NFRs to Components

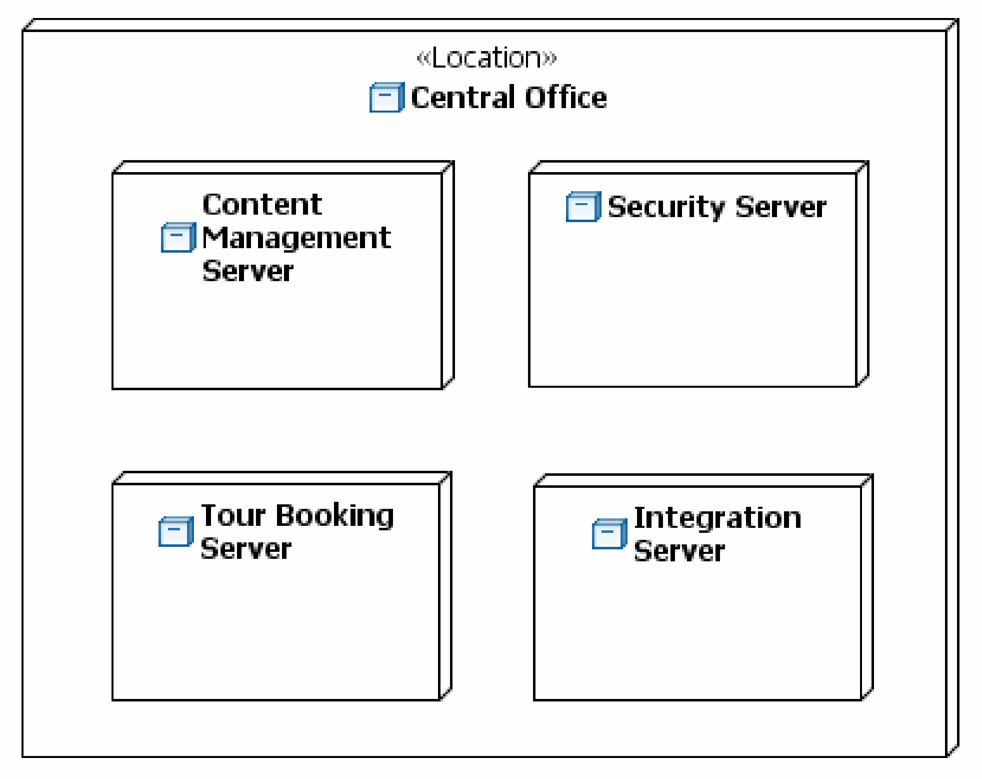


Requirement	Component	Operation	Budgeted Requirement
Booking a tour must take less than 10 seconds, from the time the request is submitted, to the confirmation of the booking being presented to the user	Tour Booking Manager	provide payment details()	1 second
	Payment Engine Interface	process payment()	3 seconds
	Reservation System Interface	make reservations()	6 seconds

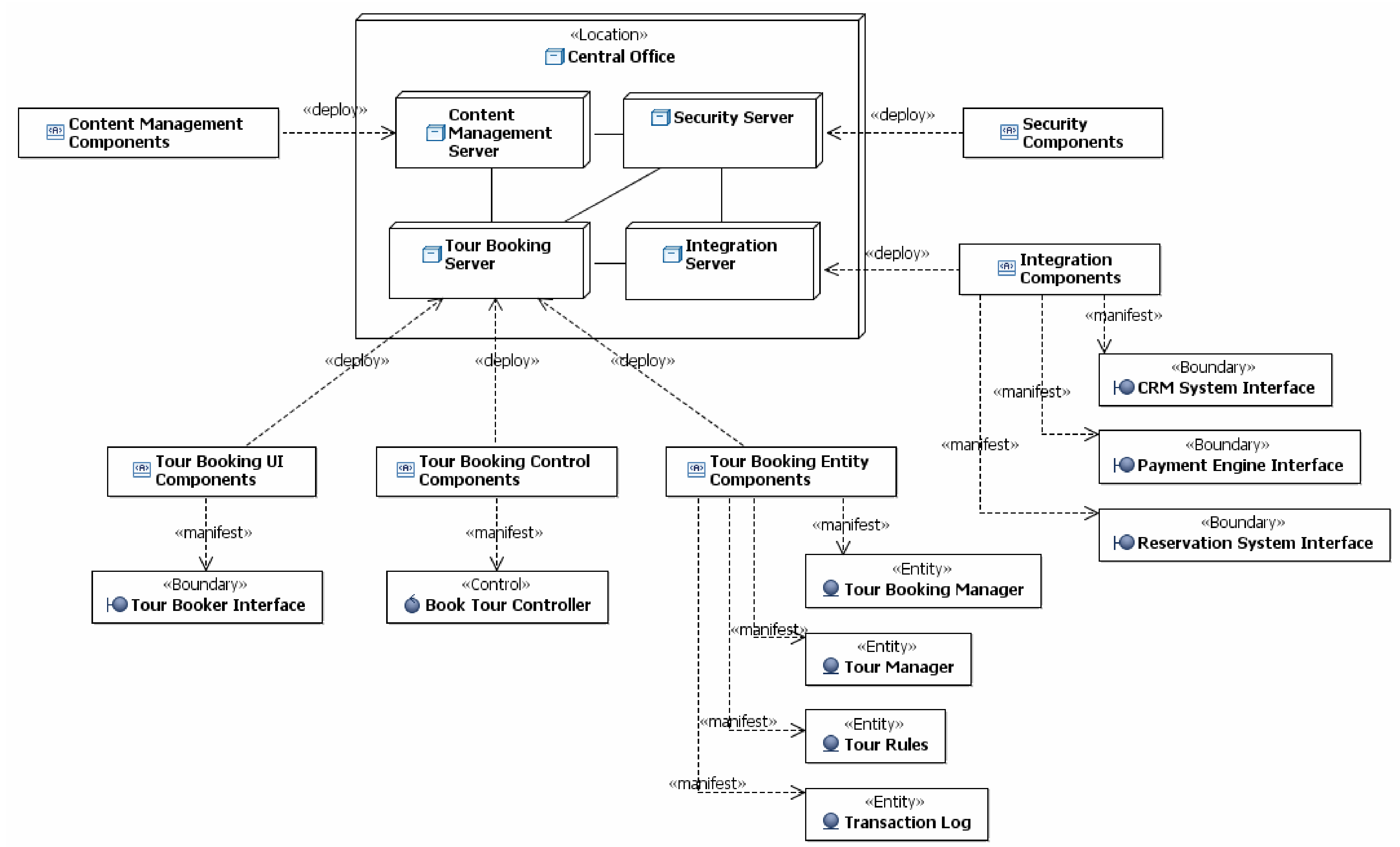
Task: Outline Deployment Elements



Task: Outline Deployment Elements



Task: Outline Deployment Elements


«Location»
Central Office
Content Management Server
Security Server
Tour Booking Server
Integration Server
Content Management Components
«deploy»
Security Components
«deploy»
Integration Components
«deploy»
«manifest»
«Boundary» CRM System Interface
«manifest»
«Boundary» Payment Engine Interface
«manifest»
«Boundary» Reservation System Interface
«deploy»
«deploy»
«deploy»
Tour Booking UI Components
Tour Booking Control Components
Tour Booking Entity Components
«manifest»
«Boundary» Tour Booker Interface
«manifest»
«Control» Book Tour Controller
«manifest»
«Entity» Tour Booking Manager
«manifest»
«Entity» Tour Manager
«manifest»
«Entity» Tour Rules
«manifest»
«Entity» Transaction Log

Summary

- The process of architecting …
  - Spans software engineering disciplines
  - Applies across the project lifecycle
  - Draws upon proven experience (practices, standards and other assets)
  - Is built upon solid engineering principles

Questions

Thank
You