Chapter 16: Architecture and Requirements

Architecturally Significant Requirement

- Not all requirements are created equal
- An **Architecturally Significant Requirement** (ASR) is a requirement that will have a profound effect on the architecture.
- How do we find those?

Approaches to Capture ASRs

- From Requirements Document
- By Interviewing Stakeholders
- By Understanding the Business Goals
- In Utility Tree

ASRs and Requirements Documents

- An obvious location to look for candidate ASRs is in the requirements documents
- Requirements should be in requirements documents!
- Unfortunately, this is not usually the case.

Don’t Get Your Hopes Up

- Many projects don’t create or maintain the detailed, high-quality requirements documents.
- Standard requirements pay more attention to functionality than quality attributes.
- The architecture **is driven by quality attribute requirements** rather than functionalities
- Most requirements specification does not affect the architecture.

Don’t Get Your Hopes Up

- Quality attributes are often captured poorly, e.g.
  - “The system shall be modular”
  - “The system shall exhibit high usability”
  - “The system shall meet users’ performance expectations”
- Much of what is useful to an architect is not in even the best requirements document
  - ASRs often derive from **business goals** in the development organization itself

Gathering ASRs from Stakeholders

- **Stakeholders** often have no idea what QAs they want in a system
  - if you insist on quantitative QA requirements, you're likely to get numbers that are arbitrary.
  - at least some of those requirements will be very difficult to satisfy.
- **Architects** often have very good ideas about what QAs are reasonable to provide.
- **Interviewing the stakeholders** is the surest way to learn what they know and need.

Gathering ASRs from Stakeholders

- **The results of stakeholder interviews** should include
  - a list of architectural drivers (驱动因素)
  - a set of QA scenarios that the stakeholders (as a group) prioritized.

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.

Quality Attribute Scenario: Example

- Our vehicle information system sends our current location to the traffic monitoring system.
- The traffic monitoring system combines our location with other information, overlays this information on a Google Map, and ***broadcasts*** it.
- Our location information is correctly included with a probability of 99.9%.

Quality Attribute Scenario: Example

- The developer wishes to change the user interface by modifying the code at design time. The modifications are made with no side effects within three hours.
  - **Stimulus** – Wishes to change UI
  - **Artifact** – Code
  - **Environment**: Design time
  - **Response** – Change made
  - **Response measure** – No side effects in three hours
  - **Source** - Developer

Quality Attribute Scenario: Example

- Users initiate transactions under normal operations. **The system** processes the transactions with an average latency of two seconds.
  - Stimulus: transaction arrivals
  - Source: users
  - Artifact: **the system**
  - Response: process the transactions
  - Response measure: average latency of two seconds
  - Environment: under normal operation

Quality Attribute Scenario: Example

- A disgruntled employee from a remote location attempts to modify the pay rate table during normal operations. The system maintains an audit trail and the correct data is restored within a day.
  - Stimulus: unauthorized attempts to modify the pay rate table
  - Stimulus source: a disgruntled employee
  - Artifact: the system with pay rate table
  - Environment: during normal operation
  - Response: maintains an audit trail
  - Response measure: correct data is restored within a day

Quality Attribute Scenario: Example

- The user downloads a new application and is using it productively after two minutes of experimentation.
  - Source: user
  - Stimulus: download a new application
  - Artifact: system
  - Environment: runtime
  - Response: user uses application productively
  - Response measure: within two minutes of experimentation

QAW Steps

- **Step 1: QAW Presentation and Introductions.**
  - QAW facilitators describe the motivation for the QAW and explain each step of the QAW.
- **Step 2: Business/Mission Presentation.**
  - The stakeholder representing the business concerns presents the system's business context, broad functional requirements, constraints, and known quality attribute requirements.
  - The quality attributes will be derived largely from the business/mission needs
- **Step 3: Architectural Plan Presentation.**
  - The architect will present the system architectural plans
  - This lets stakeholders know the current architectural thinking

QAW Steps

- **Step 4: Identification of Architectural Drivers.**
  - The facilitators will share **their list of key architectural drivers** assembled （整理，组装） during Steps 2 and 3,
  - **Architectural drivers** includes overall requirements, business drivers, constraints, and quality attributes.
  - ask the stakeholders for clarifications, additions, deletions, and corrections, and **achieve a consensus** on the architectural drivers

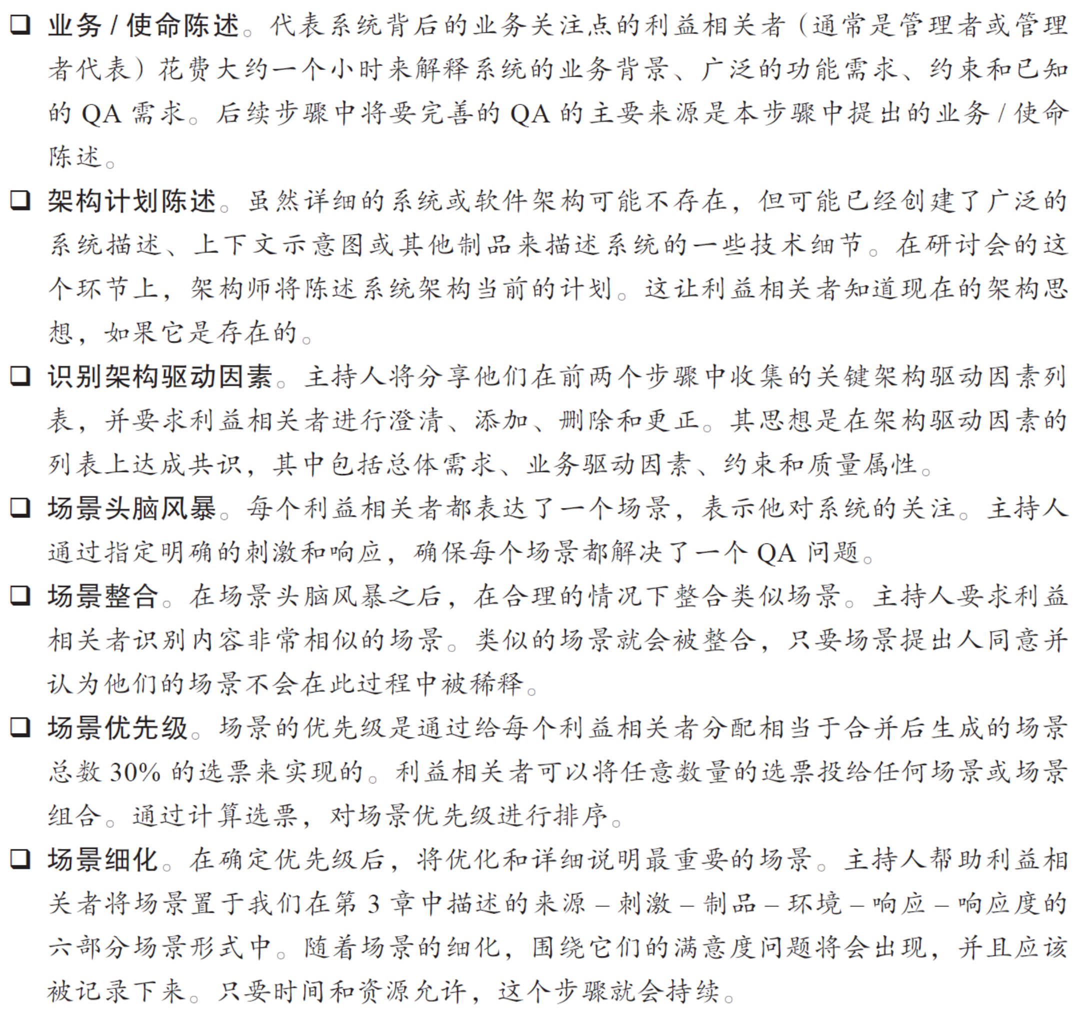
QAW Steps

- **Step 5: Scenario Brainstorming.**
  - Each stakeholder expresses a scenario representing his or her concerns with respect to the system.
  - Facilitators ensure that each scenario has an explicit stimulus and response.
  - Make at least **one representative scenario** for each architectural driver listed in Step 4.

QAW Steps

- **Step 6: Scenario Consolidation.**
  - Similar scenarios are consolidated where reasonable.
- **Step 7: Scenario Prioritization.**
  - Allocating each stakeholder a number of votes equal to 30 percent of the total number of scenarios
  - Each stakeholder allocate their votes to scenario
- **Step 8: Scenario Refinement.**
  - The top scenarios are refined and elaborated.
  - Facilitators help the stakeholders put the scenarios in the six-part scenario form

Reference Book P217



- **业务 / 使命陈述**。代表系统背后的业务关注点的利益相关者（通常是管理者或管理者代表）花费大约一个小时来解释系统的业务背景、广泛的功能需求、约束和已知的 QA 需求。后续步骤中将要完善的 QA 的主要来源是本步骤中提出的业务 / 使命陈述。
- **架构计划陈述**。虽然详细的系统或软件架构可能不存在，但可能已经创建了广泛的系统描述、上下文示意图或其他制品来描述系统的一些技术细节。在研讨会的这个环节上，架构师将陈述系统架构当前的计划。这让利益相关者知道现在的架构思想，如果它是存在的。
- **识别架构驱动因素**。主持人将分享他们在前两个步骤中收集的关键架构驱动因素列表，并要求利益相关者进行澄清、添加、删除和更正。其思想是在架构驱动因素的列表上达成共识，其中包括总体需求、业务驱动因素、约束和质量属性。
- **场景头脑风暴**。每个利益相关者都表达了一个场景，表示他对系统的关注。主持人通过指定明确的刺激和响应，确保每个场景都解决了一个 QA 问题。
- **场景整合**。在场景头脑风暴之后，在合理的情况下整合类似场景。主持人要求利益相关者识别内容非常相似的场景。类似的场景就会被整合，只要场景提出人同意并认为他们的场景不会在此过程中被稀释。
- **场景优先级**。场景的优先级是通过给每个利益相关者分配相当于合并后生成的场景总数 30% 的选票来实现的。利益相关者可以将任意数量的选票投给任何场景或场景组合。通过计算选票，对场景优先级进行排序。
- **场景细化**。在确定优先级后，将优化和详细说明最重要的场景。主持人帮助利益相关者将场景置于我们在第 3 章中描述的来源 – 刺激 – 制品 – 环境 – 响应 – 响应度的六部分场景形式中。随着场景的细化，围绕它们的满意度问题将会出现，并且应该被记录下来。只要时间和资源允许，这个步骤就会持续。

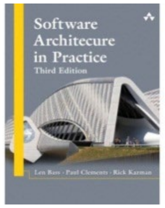
Capturing ASRs in a Utility Tree

An ASR must have the following characteristics:

- *A profound impact on the architecture*
  - Including this requirement will very likely result in a different architecture than if it were not included.
- *A high business or mission value*
  - If the architecture is going to satisfy this requirement it must be of high value to important stakeholders.

Utility Tree

- A way to record ASRs all in one place.
- Establishes priority of each ASR in terms of
  - Impact on architecture
  - Business or mission value
- ASRs are captured as scenarios.
- Root of tree is placeholder node called “Utility”.
- Second level of tree contains broad QA categories.
- Third level of tree refines those categories.
- Leaf nodes are the concrete quality attribute scenarios



Utility Tree Example (excerpt)

	Quality Attribute	**Attribute Refinement**	**ASR**
	Performance	Transaction response time	A user updates a patient's account in response to a change-of-address notification while the system is under peak load, and the transaction completes in less than 0.75 second. (H,M)
		Throughput	
	Usability	Proficiency training	
Utility		Normal operations	
	Configurability	User-defined changes	
	Maintainability	Routine changes	
		Upgrades to commercial components	

Key:
H=high
M=medium
L=low

Utility Tree: Next Steps

- ASRs that rate a (H,H) rating are the ones that deserve the most attention
  - A very large number of these might be a cause for concern: Is the system achievable?
  - (H,M): business’s value, its effect on the architecture
- Stakeholders can review the utility tree to make sure their concerns are addressed.

https://www.bilibili.com/video/BV1Nxg4eTEpQ

textbook p307

Tying the Methods Together

- How should you employ requirements documents, stakeholder interviews, Quality Attribute Workshops, and utility trees together?
  - If important stakeholders have been overlooked in the requirements-gathering process, use interviews or a QAW.
  - Use a **quality attribute utility tree** as a repository for the scenarios produced by a **QAW**.

Summary

- Architectures are driven by Architecturally Significant Requirements (ASRs):
  – requirements that will have profound effects on the architecture.
- ASRs may be captured
  – from requirements documents,
  – by interviewing stakeholders, or
  – by conducting a Quality Attribute Workshop.

Summary

- A useful representation of quality attribute requirements is in a utility tree.
- The utility tree helps to capture these requirements in a structured form.
- Scenarios are prioritized.
- This prioritized set defines your “marching orders” as an architect.

1 Which one of the following tactics is for improving the availability?

A. Resource management and scheduling

B. Increase module cohesion

C. Passive redundancy

D. Discover services

2. Modifiability is about the change and our interest in it is the cost and risk of making the change. Which of the following statements is NOT true about modifiability.

A. Using an intermediary can help improve modifiability

B. The change can be made during the implementation or execution

C. Making modules bigger can help increase modifiability

D. Publish/Subscribe tactic is used for improving modifiability

3. About Agile architecture, which of the following statement is NOT true?

A. Large-scale successful projects need a blend of agile and architecture

B. For small projects with uncertain requirements, spending too much time on the architecture design is not a good choice

C. For large project with unstable requirement, it is not needed to design an architecture at the early phase of project

D. For a large and complex system with stable requirement, it is good to devote much time on the architecture

Chapter 17: Designing an Architecture

Design Strategy

- Decomposition
- Designing to Architecturally Significant Requirements
- Generate and Test

Decomposition

- Architecture determines quality attributes
- **Important quality attributes** are characteristics of the *whole* system.
- Design begins with the whole system
  - The whole system is decomposed into parts
  - Each part may inherit all or part of the quality attribute requirements

Designing to Architecturally Significant Requirements

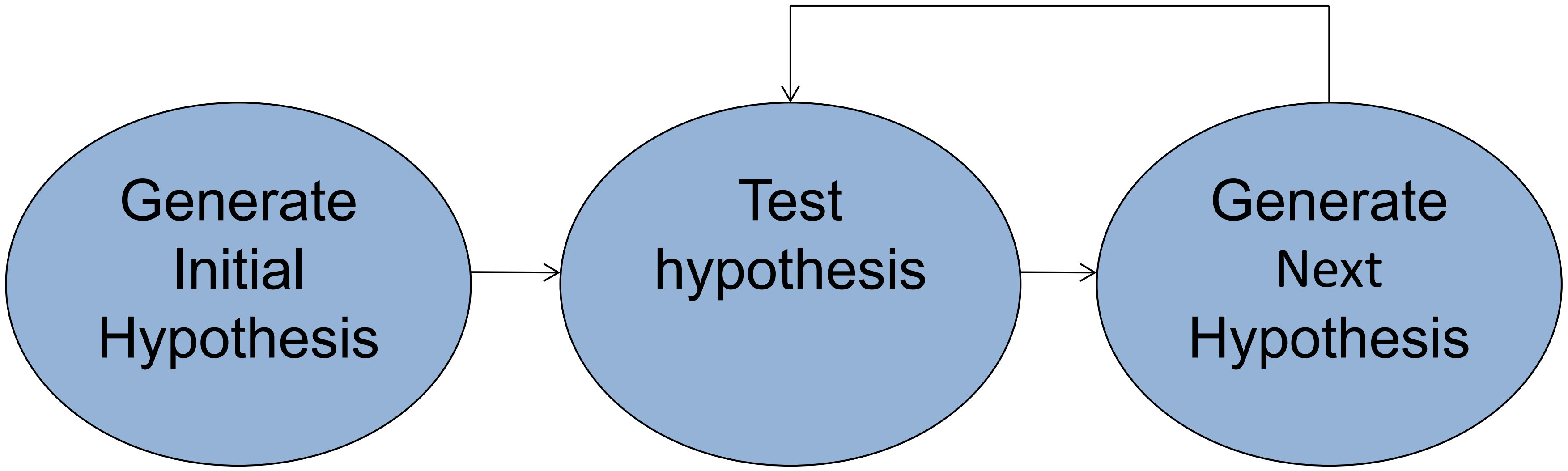
- Remember architecturally significant requirements (ASRs)?
- These are the requirements that you must satisfy with the design
  - There are a small number of these
  - They are the most important (by definition)
- Questions:
  - Do I design for one ASR at a time or all at once?

How Many ASRs Simultaneously?

- If you are inexperienced in design then design for the ASRs one at a time beginning with the most important.
- As you gain experience, you will be able to design for multiple ASRs simultaneously.

Generate and Test

- View the **current design as a hypothesis**.
- Ask whether the current design satisfies the requirements (**test**)
- If not, then **generate** a new hypothesis



Raises the Following Questions

- Where does initial hypothesis come from?
- How do I test a hypothesis?
- How do I generate the next hypothesis?
- When am I done?

Where Does the Initial Hypothesis Come From?

- **Existing systems**
  - Very few systems are completely constructed from the scratch
- **Frameworks**
  - A partial design that provides services that are common in particular domains, e.g., web applications, middleware
  - A design framework may constrain communication to be via a broker, or publish-subscribe system
- **Other sources: pattern & tactics**

How Do I Test a Hypothesis?

- The **analysis technique** described previously
- What is the output of the tests?
  - List of requirements – either responsibilities or quality – **not met** by current design.

How Do I Generate the Next Hypothesis?

- Add missing responsibilities.
- Use tactics to adjust quality attribute behavior of hypothesis.
  - The choice of tactics will depend on which quality attribute requirements are not met.
  - Be mindful of the side effects of a tactic.

When Am I Done?

- All ASRs are satisfied and/or…
- You run out of budget for design activity
  - In this case, use the best hypothesis so far and begin implementation
  - To relax or eliminate the requirement
  - To argue for more budges

The Attribute-Driven Design Method

The Attribute-Driven Design (ADD) method is a packaging of the strategies that we have just discussed

The Attribute-Driven Design Method

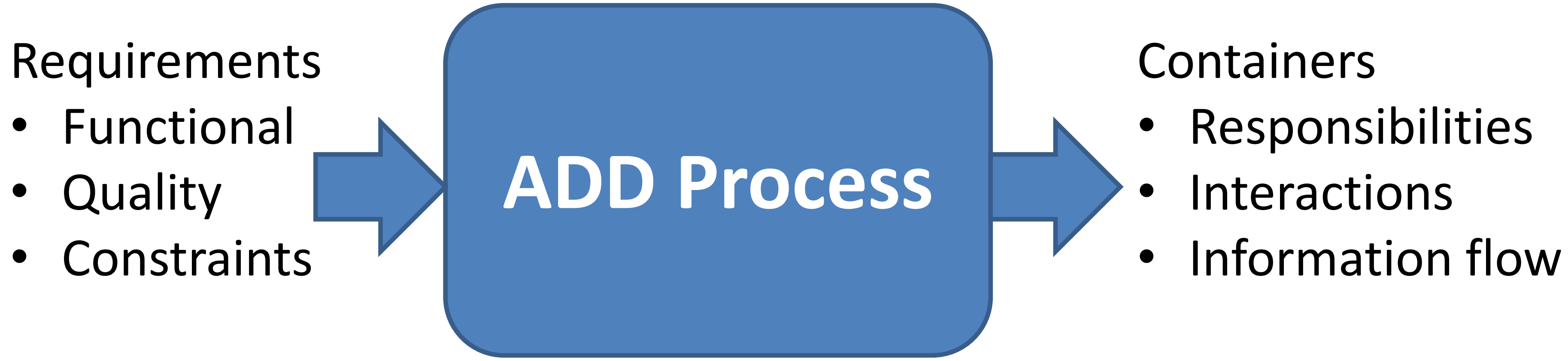
- An iterative method. At each iteration you
  - **Choose a part** of the system to design.
  - **Marshal (整理)** all the architecturally significant requirements for that part.
  - **Generate and test a design** for that part.
- ADD does not result in a complete design
  - Set of containers with responsibilities
  - Interactions and information flow among containers
- Does not produce an API for containers.

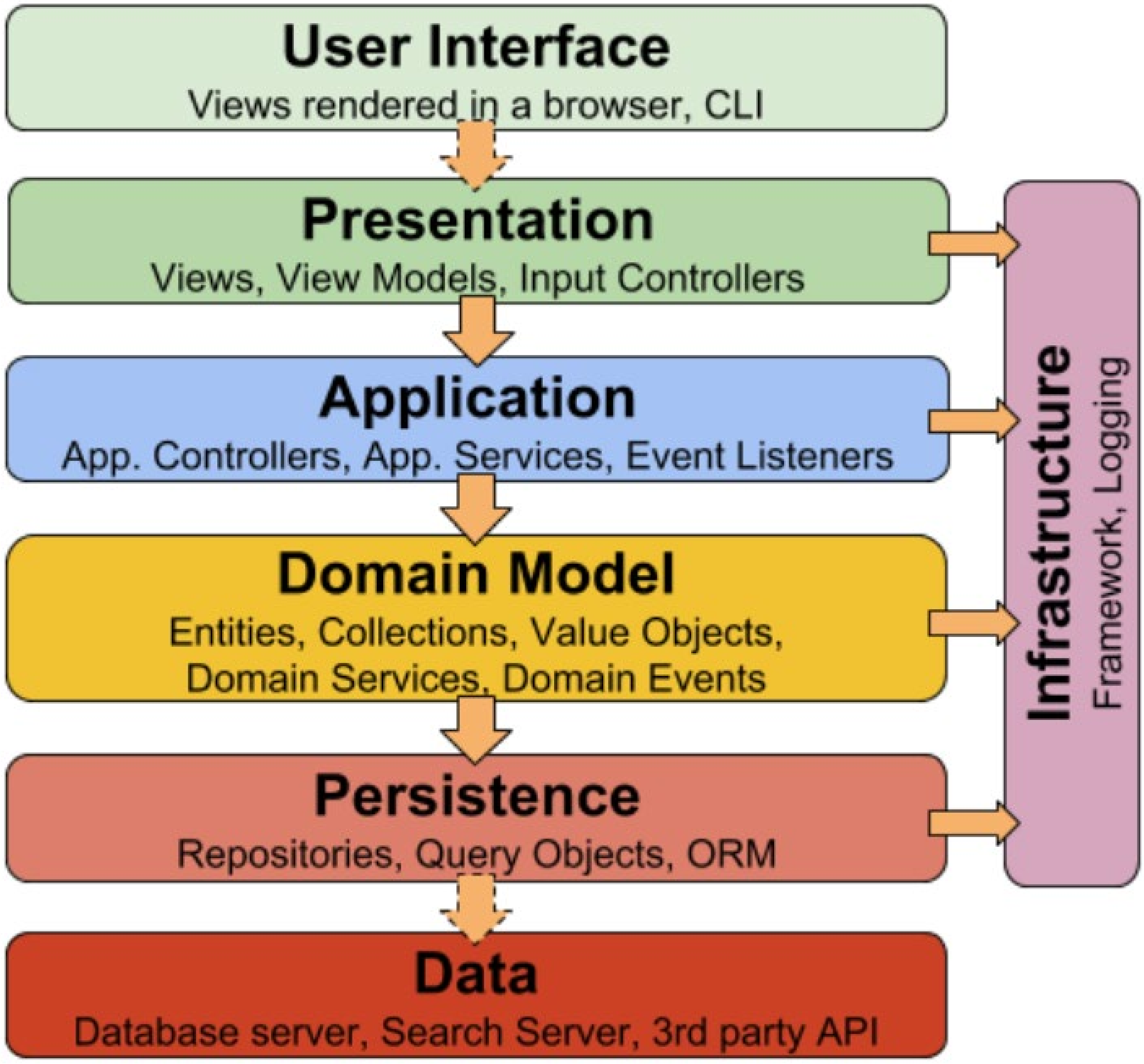
ADD Inputs

- Requirements: Functional, quality (ASR), constraints
- A context description
  - What are the boundary of the system being designed?
    - Scope: What is inside the system and what is outside the system
  - What are the external systems, devices, users and environment conditions with which the system being designed must interact?
    - An example of accommodating environment conditions can be seen in a system that must be sent into space

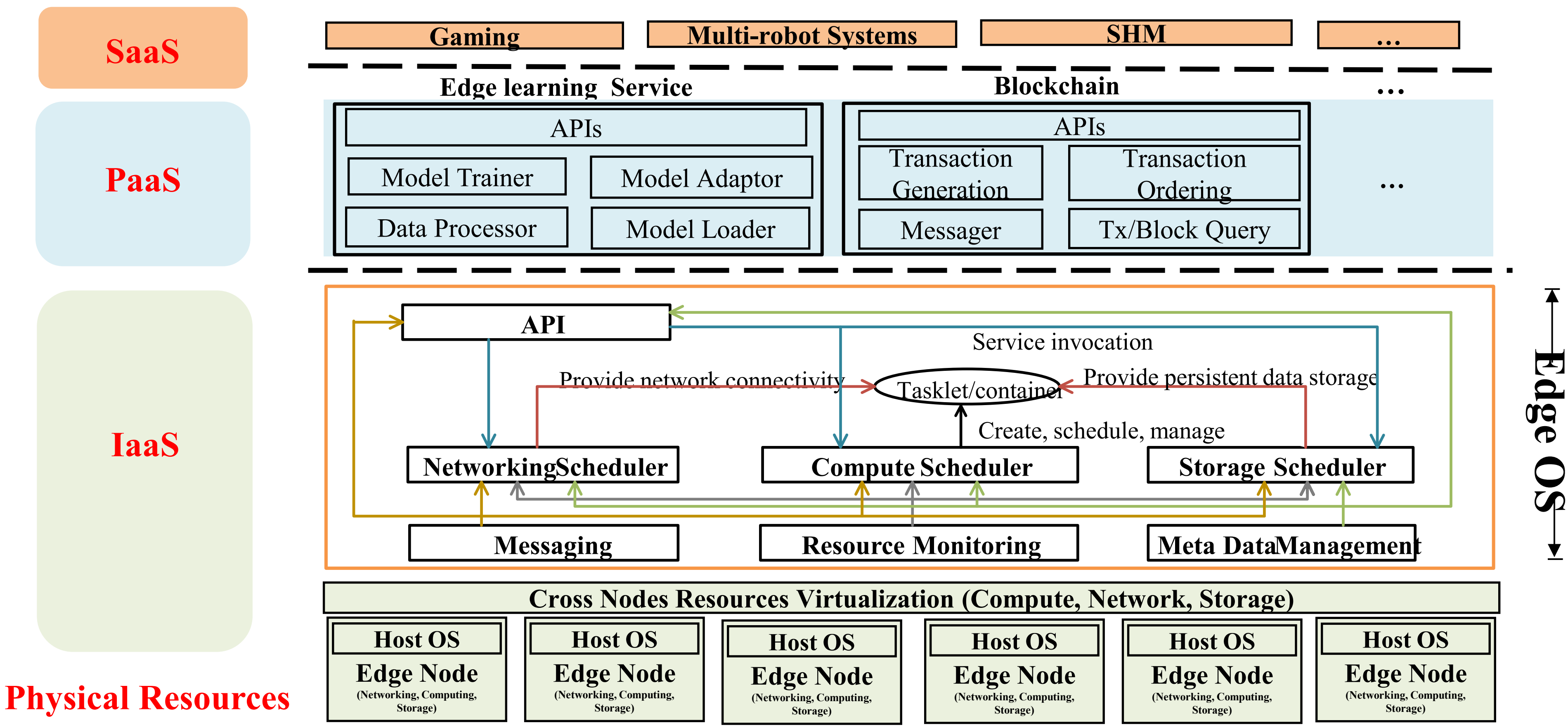
ADD Outputs

- Architectural elements and their relationship
  - Responsibility of elements
  - Interactions
  - Information flow among the elements




User Interface
Views rendered in a browser, CLI
Presentation
Views, View Models, Input Controllers
Application
App. Controllers, App. Services, Event Listeners
Domain Model
Entities, Collections, Value Objects, Domain Services, Domain Events
Persistence
Repositories, Query Objects, ORM
Data
Database server, Search Server, 3rd party API
Infrastructure
Framework, Logging

EdgeOS

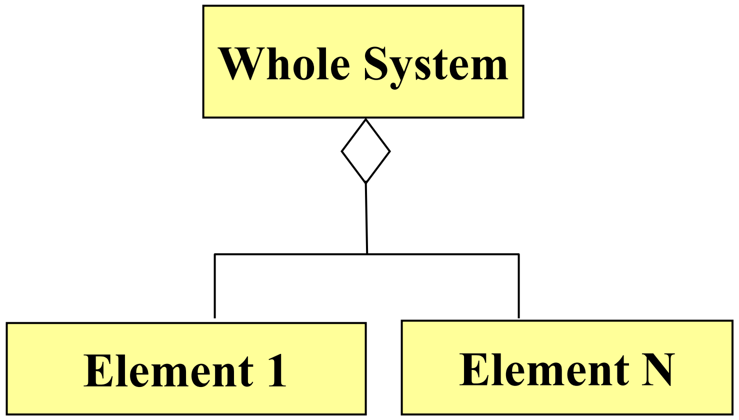

SaaS
PaaS
IaaS
Physical Resources
Gaming
Multi-robot Systems
SHM
…
Edge learning Service
Blockchain
…
APIs
Model Trainer
Model Adaptor
Data Processor
Model Loader
APIs
Transaction Generation
Transaction Ordering
Messager
Tx/Block Query
…
API
Service invocation
Provide network connectivity
Tasklet/container
Provide persistent data storage
Create, schedule, manage
NetworkingScheduler
Compute Scheduler
Storage Scheduler
Messaging
Resource Monitoring
Meta DataManagement
Edge OS
Cross Nodes Resources Virtualization (Compute, Network, Storage)
Host OS
Edge Node
(Networking, Computing, Storage)
Host OS
Edge Node
(Networking, Computing, Storage)
Host OS
Edge Node
(Networking, Computing, Storage)
Host OS
Edge Node
(Networking, Computing, Storage)
Host OS
Edge Node
(Networking, Computing, Storage)
Host OS
Edge Node
(Networking, Computing, Storage)

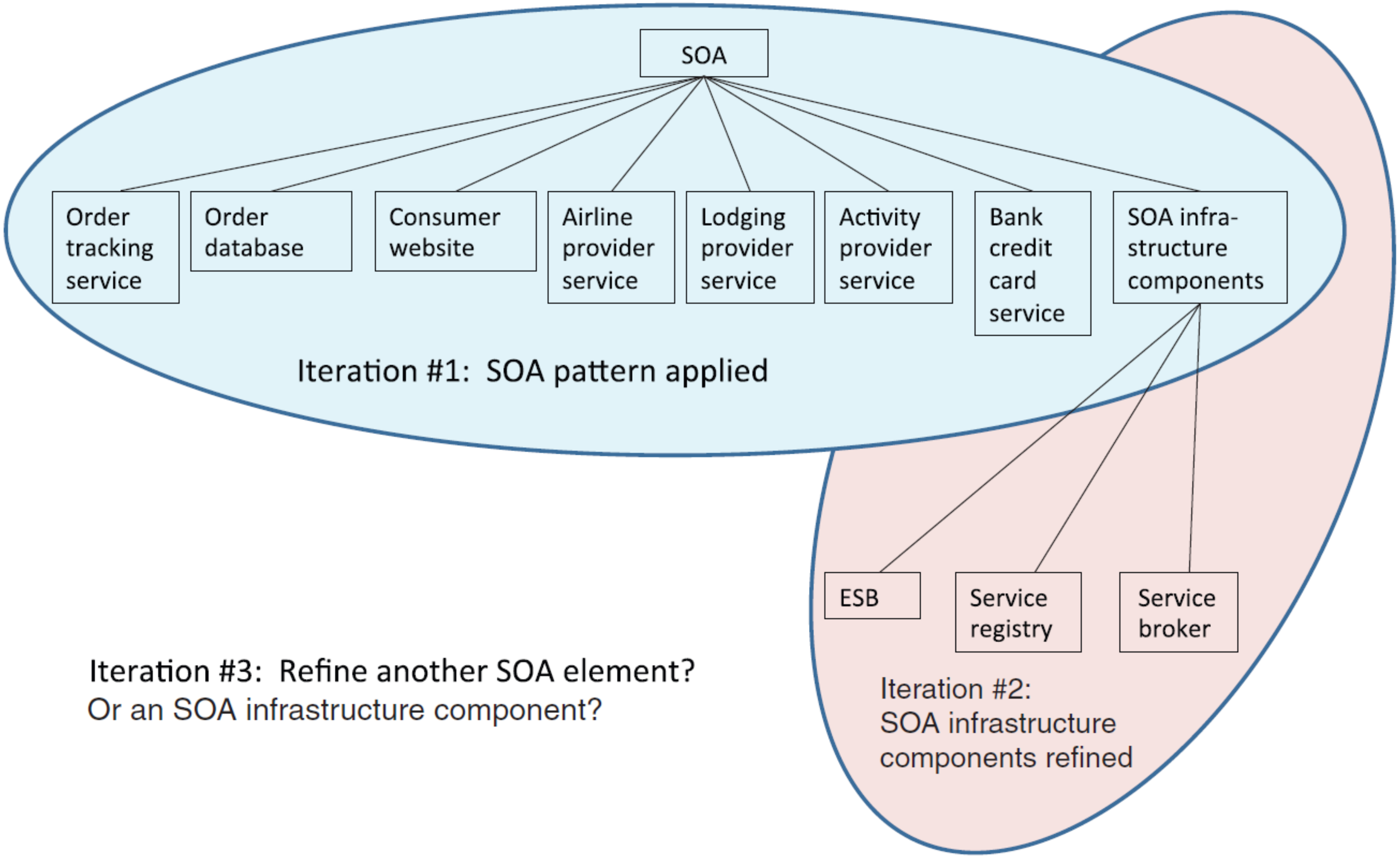
The Steps of ADD

1. Choose an element of the system to design.
2. Identify the ASRs for the chosen element.
3. Generate a design solution for the chosen element.
4. Inventory remaining requirements and select the input for the next iteration.
5. Repeat steps 1–4 until all the ASRs have been satisfied.

Step 1: Choose an Element of the System to Design

- For **green field (新建) designs**, the element chosen is usually the whole system.
- For **legacy (遗产) designs**, the element is the portion to be added.
- After the first iteration:





Iteration #3: Refine another SOA element?
Or an SOA infrastructure component?

Which Element Comes Next?

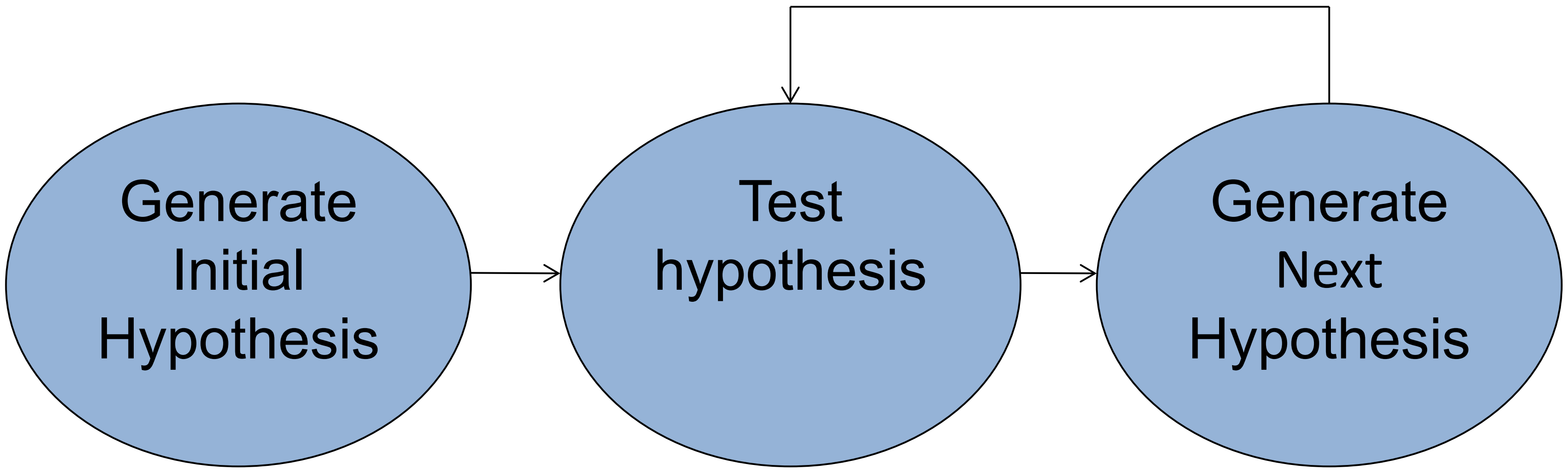
- Two basic refinement strategies:
  - Breadth first
  - Depth first
- Which one to choose?
- If using new technology => **depth** first: explore the implications of using that technology.
- If a team needs work => **depth** first: generate requirements for that team.
- Otherwise => breadth first.

Step 2: Identify the ASRs for the Chosen Element

- If the chosen element is the whole system, then use a utility tree (as described earlier).
- If the chosen element is further down the decomposition tree, then generate a utility tree from the requirements for that element.

Step 3: Generate a Design Solution for the Chosen Element

- Apply generate and test to the chosen element with its ASRs

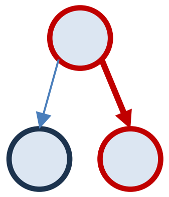


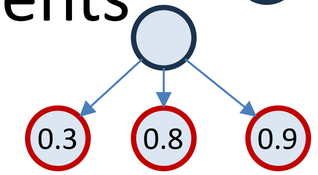
Step 4: Select the Input for the Next Iteration

- For each **functional requirement**
  - Ensure that requirement has been **satisfied**.
  - If not, then add responsibilities to satisfy the requirement.
    - **Add** them to container with similar requirements
    - If no such container, may need to **create new one** or add to container with dissimilar responsibilities (coherence)
    - If container has too many requirements for a team, **split** it into two portions. Try to achieve loose coupling when splitting.

Quality Attribute Requirements

- If the quality attribute requirement has been **satisfied**, it does not need to be further considered.
- If the quality attribute requirement has not been satisfied then either
  - **Delegate (委托)** it to one of the child elements
  - **Split (拆分)** it among the child elements
- If the quality attribute **cannot be satisfied**, see if it can be weakened. If it cannot be satisfied or weakened then it cannot be met.





Repeat Steps 1–4 Until All ASRs are Satisfied

- At end of step 4, each child element will have associated with it a set of:
  - functional requirements,
  - quality attribute requirements, and
  - constraints.
- This sets up the child element for the next iteration of the method.

Example

Assuming we are designing an e-commerce website that needs to meet quality attributes such as "performance" (e.g., low response time) and "availability" (e.g., system always being available). To ensure that the system responds quickly under high traffic, we can decompose the website's architecture into several components:

- **Frontend servers**: Handle user interfaces and requests.
- **Load balancer:** Ensures that user requests are distributed across different backend servers, preventing single points of failure.
- **Database:** Stores user and product information, ensuring data reliability.

Example

We found that "performance" and "availability" are ASRs.

- **Performance:** To improve performance, the architect may decide to use a "caching mechanism" to accelerate the loading of frequently accessed product information, reducing the frequency of database queries.
- **Availability:** To enhance system availability, the architect might choose a "master-slave replication" architecture for the database, ensuring that if one database fails, the system can switch to a backup database to continue providing services.

Example

Once the architecture design is completed, the development team will begin implementing and testing the system:

- **Generation:** The development team builds an initial version of the system based on the architecture design, including the frontend, backend, and database.
- **Hypo. and Test:** The team will conduct stress testing simulating a large number of users accessing the website (10,000 request/1s ?), checking the response time of the frontend servers (10,000 request completed/1s ?), and the functionality of the load balancer.

Example

Once the architecture design is completed, the development team will begin implementing and testing the system:

- **Generation:** The development team builds an initial version of the system based on the architecture design, including the frontend, backend, and database.
- **Hypo. and Test:** They will also test the availability of the database, simulating a database failure to ensure the backup database can take over seamlessly (100ms?).

Summary

- Designing Strategies
  - Decomposition
  - Designing to Architecturally Significant Requirements
  - Generate and Test
- Attribute Driven Design
  - **Choose a part** of the system to design.
  - **Marshal** all the architecturally significant requirements for that part.
  - **Generate and test a design** for that part.

Chapter 19: Architecture, Implementation, and Testing

Architecture and Implementation

- Techniques to keep the code and the architecture consistent:
  - Embedding the design in the code
  - Frameworks
  - Code templates

Embedding the Design in the Code

- Architecture acts as a **blueprint** for implementation. This means
  - Implementers know what architectural structure they are implementing. E.g. layer, pub/sub, MVC, broker, …
  - They can document the architectural structure in the code as comments. Then anyone picking up the code will know some of the constraints.
  - Then tools can automatically relate the code and the architecture.

Frameworks

- A **framework** is a reusable set of libraries or classes organized around a particular theme.
- A programmer uses the services provided by a framework.
- Examples are
  - Django (Python): Django is a high-level Python web framework that follows the MVC pattern (referred to as MVT: Model-View-Template in Django).
  - Spring MVC (Java): Spring MVC is an MVC framework in Java, often used for building Java-based web applications.

Code Templates

- **A code template** is collection of code within which the programmer provides application specific portions.
- For example?
- Process:
  - Use the code template for every critical component
  - Place application specific code in fixed places within the template.

Advantages of Code Templates

- Components with similar properties behave in a similar fashion.
- Template only needs to be **debugged once**.
- Complicated portions can be completed by skilled personnel and handed off to less skilled personnel.

- Any disadvantage?

Architecture and Testing

- Architecture design determines **the feasibility and efficiency of testing**. A good architecture makes unit testing, integration testing, and performance testing easier.
- Architecture influences **testing strategies and methods**. Characteristics considered in architecture design, such as modularity, interfaces, and dependency management, provide support for testing and affect the depth and coverage of the tests.
- Architecture and testing **work together to ensure system quality.** A good architecture not only supports the system in running as expected but also provides stronger support for testing, thereby ensuring the quality of the system.

Architecture and Testing

- Two Levels of Testing
- **Unit testing**: tests running on specific pieces of software.
  - Components or Modules
- What is needed to test:
  - **Responsibilities** for functional correctness
  - **Performance** through synthetic loads
  - **Availability** through fault injection.
  - **Modifiability** requirements can also be tested by assigning changes to test teams
  - **Security** is tested by having the test executes various attack scenarios.

Two Levels of Testing

- **Integration Testing**: to test what happens when separate software units start to work together
- Integration test can test functionality, performance, availability, and security.

Black-Box Testing

- **Black-Box Testing** treats the software as an "black box" without any knowledge about the internal design, structure, or implementation
- The tester's only source information are its requirements
- How does architecture help in black-box testing?
  - Interface
- It helps the tester understand what portions of the requirements related to the specified subsystem

White-Box Testing

- **White-Box testing** makes full use of internal structures, algorithms, and control and data flows of a unit of software
- Tests exercises all control paths of a unit
- It is often used in unit testing

Test Activities

- The architect should be actively involved in
  - **Test planning** is to allocate the resources, i.e., time, labor, technologies, tools, hardware or equipment
  - **Test development** is the procedure in which tests are written, test case are chosen, and test datasets are created.
    - Test driven development is a technique where the system is developed to satisfy a predetermined test.
  - **Test execution**. Testers apply the tests to the software and capture the record errors

Test Activities

- The architect should be actively involved in
  - **Test reporting and defect analysis**
  - **Test harness creation** includes the test engine and test script repository;
  - The primary purpose is to automate the testing process.

Summary

- Implementation
  - **Implementation activities** can embed architecture knowledge in the code
  - **Templates** can be used for critical sections that reoccur
  - **Architecture erosion** can be prevented through use of tools and management processes
- Testing
  - **Unit and integration tests** depend on architectural knowledge and a test harness.
  - The architect should be involved in **a wide variety of test activities**.

Chapter 21: Architecture Evaluation

Three Forms of Evaluation

- Evaluation by the designer within the design process.
- Evaluation by peers within the design process.
- Analysis by outsiders once the architecture has been designed.

Evaluation by the Designer

- Every time the designer makes a key design decision, the chosen alternatives should be evaluated.
- The “test” part of the “generate-and-test” approach
- How much analysis? Three factors include:
  - **The importance of the decision**.
  - **The number of potential alternatives**. More alternatives need more time in evaluating them.
  - **Good enough as opposed to perfect**. Do not spend more time on a decision than it is worth.

Peer Review

- Architectural designs can be peer reviewed, just as code can.
- A peer review can be carried out at any point of the design process where a candidate architecture, or at least a coherent reviewable part of one, exists.
- Allocate at least several hours and possibly half a day.

Peer Review Steps

1. The reviewers determine a number of quality attribute scenarios to drive the review.
2. The architect presents the portion of the architecture to be evaluated.
   – The purpose is to make the reviewers understand the architecture.
3. For each scenario, the designer walks through the architecture and explains how the scenario is satisfied.
4. Potential problems are captured.

Analysis by Outsiders

- "Outside" is relative; this may mean
  - outside the development project
  - outside the business unit where the project resides but within the same company
  - outside the company.
- Outsiders are chosen because they possess specialized knowledge or experience
- Managers tend to be more inclined (倾向于) to listen to problems uncovered by an outside team.
- Be used to evaluate complete architectures.

Contextual Factors for Evaluation

For peer reviews or outside analysis

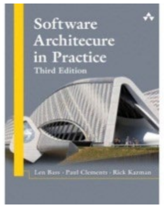
- Whether the result is public or private
- The number and skill of evaluators
- Which stakeholders will participate
- How the business goals are understood by the evaluators

The Architecture Tradeoff Analysis Method (ATAM)

- The **Architecture Tradeoff Analysis Method (ATAM)** has been used for over a decade to evaluate software architectures in domains ranging from automotive to financial to defense.
- The ATAM is designed so that evaluators need not be familiar with the architecture or its business goals, the system need not yet be constructed, and there may be a large number of stakeholders.

Participants in the ATAM

- The evaluation team
  - 3 to 5 people
  - Competent, unbiased outsiders
- Project decision makers
  - the project manager, the architect and the customer who bill for the development (甲方)
- Architecture stakeholders
  - developers, testers, integrators, maintainers, performance engineers, users, builders of systems
  - to articulate the specific quality attribute goals
  - 12 to 15 stakeholders for a large enterprise-critical architecture



ATAM Evaluation Team Roles

Role	Responsibilities
Team leader	Sets up the evaluation; coordinates with client, making sure client's needs are met; establishes evaluation contract; forms evaluation team; sees that final report is produced and delivered (although the writing may be delegated)
Evaluation leader	Runs evaluation; facilitates elicitation of scenarios; administers scenario selection/prioritization process; facilitates evaluation of scenarios against architecture; facilitates on-site analysis
Scenario scribe	Writes scenarios on flipchart or whiteboard during scenario elicitation; captures agreed-on wording of each scenario, halting discussion until exact wording is captured
Proceedings scribe	Captures proceedings in electronic form on laptop or workstation: raw scenarios, issue(s) that motivate each scenario (often lost in the wording of the scenario itself), and resolution of each scenario when applied to architecture(s); also generates a printed list of adopted scenarios for handout to all participants
Questioner	Raises issues of architectural interest, usually related to the quality attributes in which he or she has expertise

Primary Outputs of the ATAM

- A set of risks and nonrisks
  - A **risk** is defined as an architectural decision that may lead to undesirable consequences in light of quality attribute requirements.
  - A **nonrisk** is an architectural decision that is deemed safe
- A set of **risk themes**
  - examines the full set of risks to look for themes that identify system weaknesses in the architecture.
  - These risk themes will threaten the project's business goals

Other Outputs of the ATAM

1. A concise presentation of the architecture
2. Articulation of the business goals.
3. Prioritized quality attribute requirements expressed as quality attribute scenarios.
4. Mapping of architectural decisions to quality requirements.
5. A set of identified sensitivity and tradeoff points: architectural decisions that have a marked effect on one or more quality attributes.

Step 1: Present the ATAM

- The **evaluation leader** describes the ATAM steps in brief and the outputs of the evaluation

Step 2: Present Business Drivers

- The **project decision maker** (ideally the project manager or the system's customer) presents a system overview from a business perspective
- The presentation should describe
  - important functions
  - relevant technical, managerial, economic, or political constraints
  - business goals and context
  - major stakeholders
  - architectural drivers (architecturally significant requirements)

Step 3: Present the Architecture

- The **lead architect** makes a presentation describing the architecture
  - **technical constraints** such as operating system, hardware, or middleware prescribed for use, and other systems with which the system must interact.
  - **architectural approaches**, i.e., patterns & tactics used to meet the requirements
- Should convey the essence of the architecture and NOT go too deeply into the details

Step 4: Identify Architectural Approaches

- Understand its architectural approaches, especially patterns and tactics.
- The evaluation team catalogs (分类) the patterns and tactics that have been identified.
- The list is publicly captured and will serve as the basis for later analysis

Step 5: Generate Utility Tree

- The evaluation team works with the project decision makers to identify, prioritize, and refine the system’s most important quality attribute goals
- The quality attribute goals are articulated (阐述) in detail via a quality attribute utility tree.
- The scenarios are assigned a rank of importance (High, Medium, Low)

Step 6: Analyze Architectural Approaches

- The evaluation team examines the highest-ranked scenarios one at a time; the architect is asked to explain how the architecture supports each one
- Evaluation team members probe (探究) for the architectural approaches used to carry out the scenario
- the evaluation team documents the relevant architectural decisions and identifies and catalogs their risks, nonrisks, and tradeoff points. Examples:
  - **Risk**: The frequency of heartbeats affects the time in which the system can detect a failed component.
  - **Tradeoff**: The heartbeat frequency determines the time for detecting a fault. Higher frequency leads to better availability but consumes more processing time and communication bandwidth (potentially reducing performance).

Step 7: Brainstorm and Prioritize Scenarios

- The purpose of scenario brainstorming is to understand what system success means for stakeholders
- Once the scenarios have been collected, they are prioritized by voting.
- The list of prioritized scenarios is compared with those from the utility tree exercise.
  - If they agree, it indicates good alignment between the architect had in mind and what the stakeholders actually wanted.
  - If additional driving scenarios are discovered, this may itself be a risk. some disagreement between the stakeholders and the architect.

Step 8: Analyze Architectural Approaches

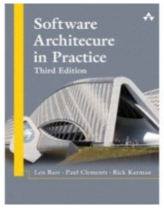
- In this step the evaluation team performs the same activities as in step 6
- using the highest-ranked, newly generated scenarios

Step 9: Present Results

- The evaluation team group risks into risk themes, based on some systemic deficiency.
  - For example, a group of risks about the system's inability to function in the face of various hardware and/or software failures might lead to a risk theme about insufficient attention to providing high availability.
- For each risk theme, the evaluation team identifies which of the business drivers listed in step 2 are affected.

Step 9: Present Results

- The collected information from the evaluation is summarized and presented to stakeholders.
- The following outputs are presented:
  - The architectural approaches documented
  - The set of scenarios and their prioritization from the brainstorming
  - The utility tree
  - The risks discovered
  - The nonrisks documented
  - The sensitivity points and tradeoff points found
  - Risk themes and the business drivers threatened by each one

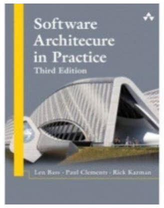


Phases of the ATAM

Phase	Activity	Participants	Typical duration
0	Partnership and preparation: Logistics, planning, stakeholder recruitment, team formation	Evaluation team leadership and key project decision-makers	Proceeds informally as required, perhaps over a few weeks
1	Evaluation: Steps 1-6	Evaluation team and project decision-makers	1-2 days followed by a hiatus of 2-3 weeks
2	Evaluation: Steps 7-9	Evaluation team, project decision makers, stakeholders	2 days
3	Follow-up: Report generation and delivery, process improvement	Evaluation team and evaluation client	1 week

Lightweight Architectural Evaluation

- An ATAM is a substantial undertaking
  - It requires some 20 to 30 person-days of effort from an evaluation team, plus even more for the architect and stakeholders.
  - Investing this amount of time makes sense on a large and costly project
- A Lightweight Architecture Evaluation method for smaller, less risky projects.
  - May take place in a single day, or even a half-day meeting.
  - May be carried out entirely by members internal to the organization.
  - Of course this may not probe the architecture as deeply.



Typical Agenda: 4-6 Hours

Step	Time	Notes
1. Present the ATAM	0 hours	Participants already familiar with process.
2. Present business drivers	0.25 hours	The participants are expected to understand the system and its business goals and their priorities. A brief review ensures that these are fresh in everyone's mind and that there are no surprises.
3. Present architecture	0.5 hours	All participants are expected to be familiar with the system. A brief overview of the architecture, using at least module and C&C views, is presented. 1-2 scenarios are traced through these views.
4. Identify architectural approaches	0.25 hours	The architecture approaches for specific quality attribute concerns are identified by the architect. This may be done as a portion of step 3.
5. Generate QA utility tree	0.5- 1.5 hours	Scenarios might exist: part of previous evaluations, part of design, part of requirements elicitation. Put these in a tree. Or, a utility tree may already exist.
6. Analyze architectural approaches	2-3 hours	This step—mapping the highly ranked scenarios onto the architecture—consumes the bulk of the time and can be expanded or contracted as needed.
7. Brainstorm scenarios	0 hours	This step can be omitted as the assembled (internal) stakeholders are expected to contribute scenarios expressing their concerns in step 5.
8. Analyze architectural approaches	0 hours	This step is also omitted, since all analysis is done in step 6.
9. Present results	0.5 hours	At the end of an evaluation, the team reviews the existing and newly discovered risks, nonrisks, sensitivities, and tradeoffs and discusses whether any new risk themes have arisen.

Summary

- If a system is important enough for you, then that architecture should be evaluated.
- The **ATAM** is a comprehensive method for evaluating software architectures
- **Lightweight Architecture Evaluation**, based on the ATAM, provides an inexpensive architecture evaluation that can be carried out in several hours
- The number of evaluations and the extent of each evaluation may vary from project to project
  - A designer should perform an evaluation during the process of making an important decision
  - Lightweight evaluations can be performed several times during a project as a peer review exercise

11.25体系结构

微信扫码签到