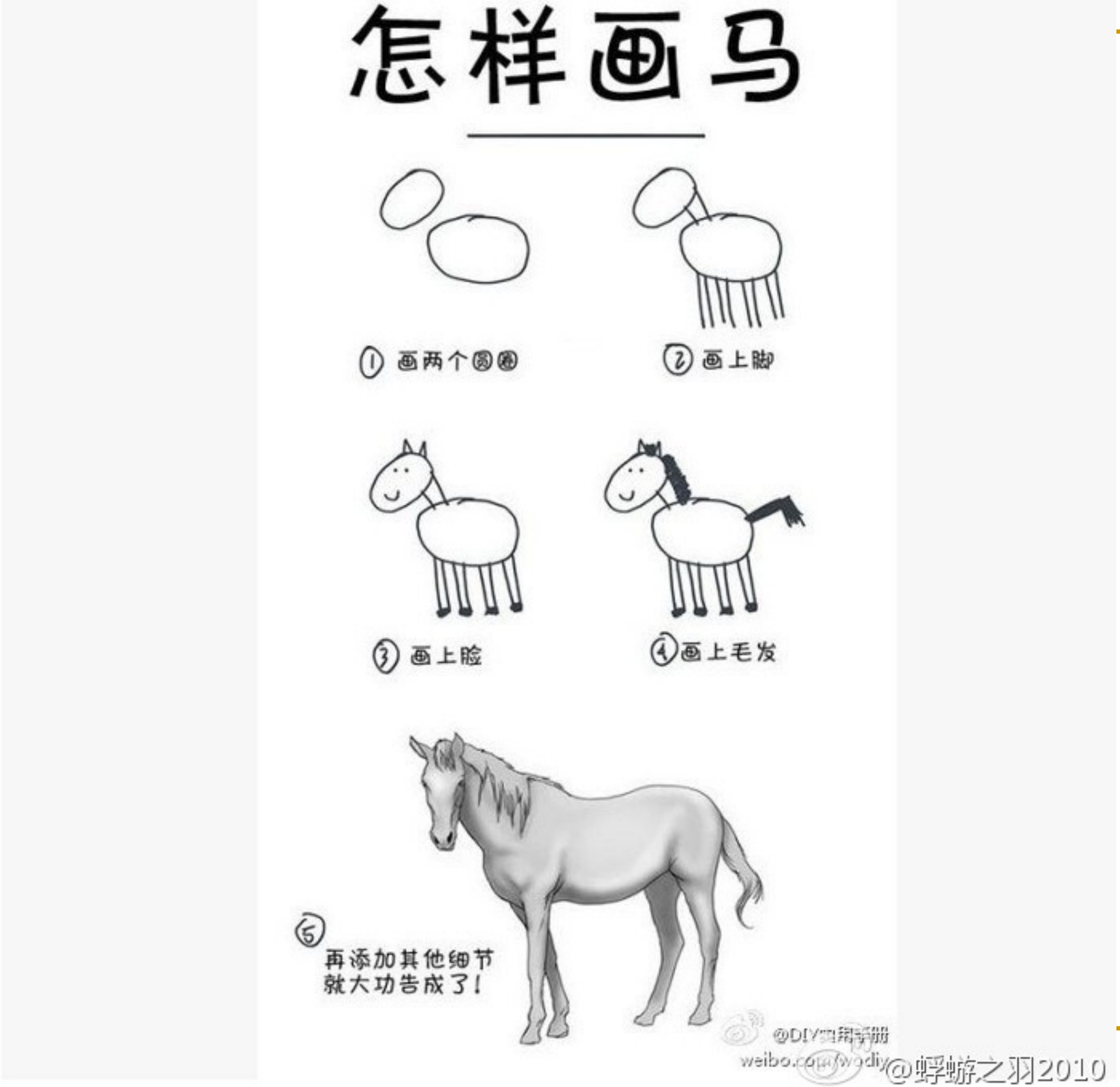
Designing an Architecture (1)

主讲教师：王灿

Email: wcan@zju.edu.cn

TA: 李奇平 liqiping1991@gmail.com

Course FTP: ftp://sa:sa@10.214.51.13



每次看论文都有这种坑爹的感觉。。。。

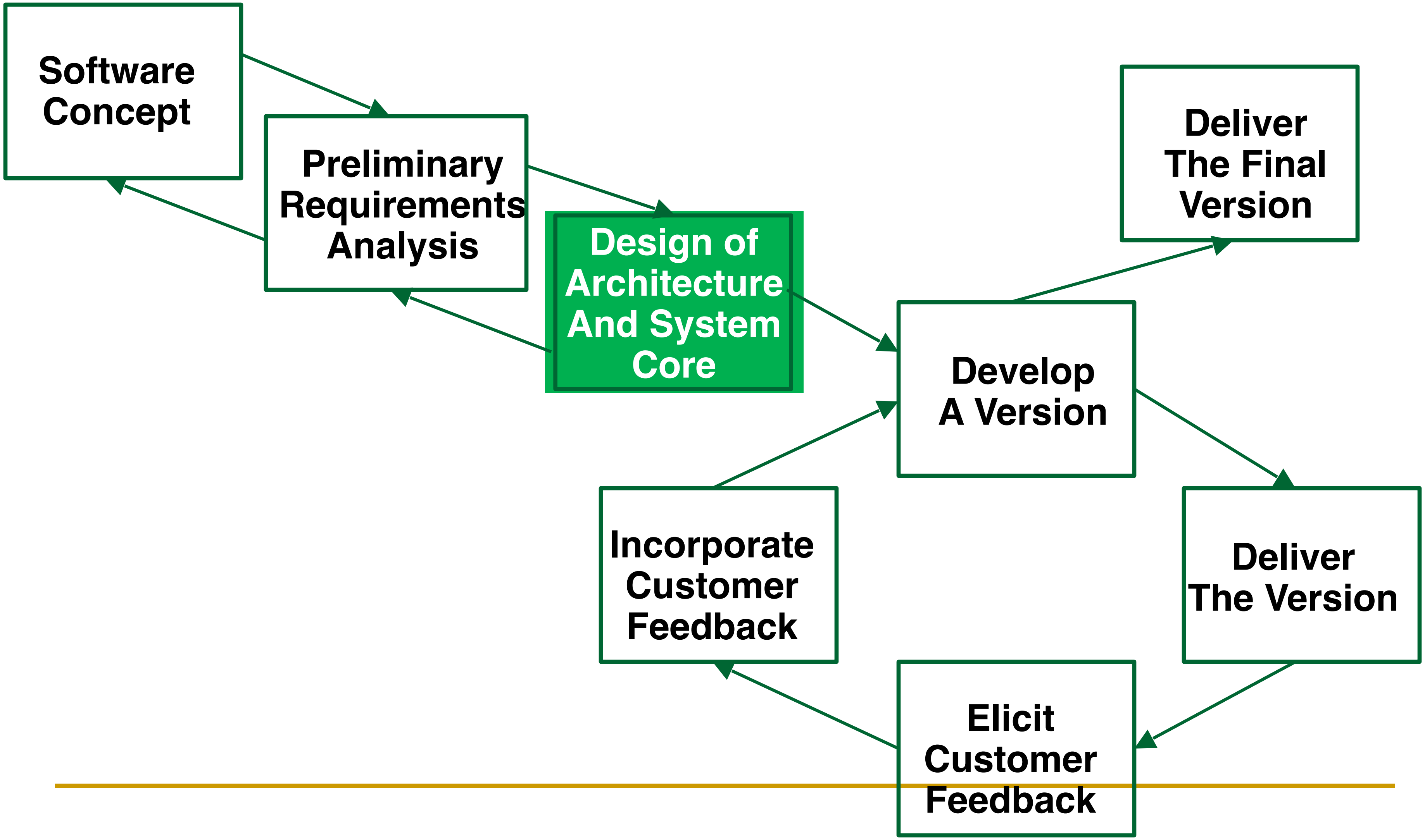
Designing an Architecture

Architecturally Significant Requirements

Design Strategy

Attribute Driven Design

Evolutionary Delivery Life Cycle



Software Concept

Preliminary Requirements Analysis

Design of Architecture And System Core

Develop A Version

Deliver The Final Version

Deliver The Version

Elicit Customer Feedback

Incorporate Customer Feedback

When Do We Start Designing the Software Architecture?

- Requirements come first
  - But not all requirements are necessary to get started
- An architecture shaped by some:
  - Functional requirements
  - Quality requirements
  - Business requirements
- We call these requirements “architecturally significant requirements (ASRs)
  - Requirements that will have profound effects on the architecture
  - Often takes the form of quality attribute requirements

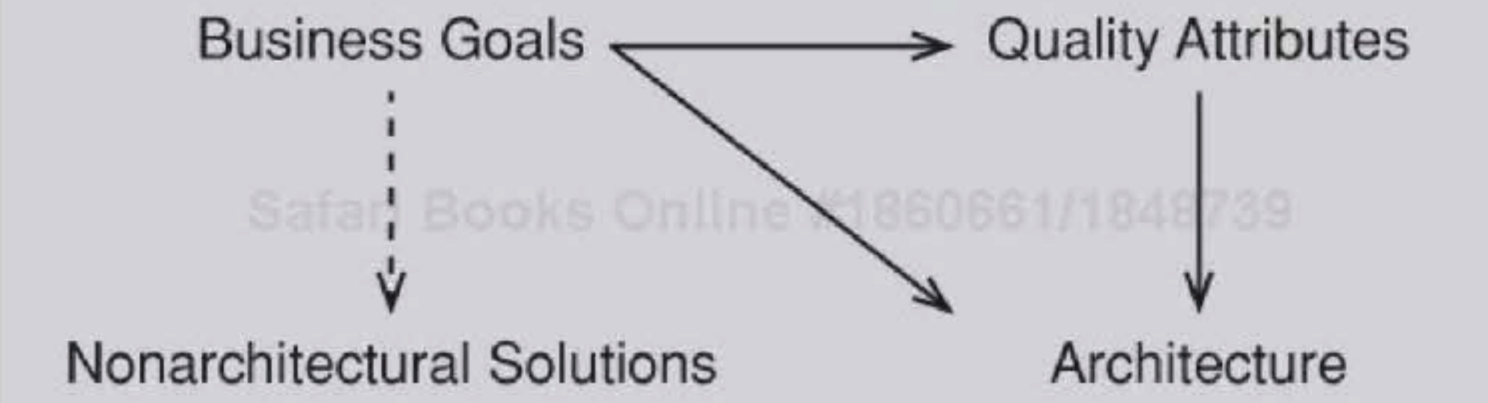
Gathering ASRs from Requirements Documents

- Requirements specification focused on the required features and functionality of a system
  - Most of the requirements does not affect the architecture
  - Architectures are mostly driven or "shaped" by quality attribute requirements
- Architects have to excavate ASRs out of requirements documents
  - Allocation of responsibilities, coordination model, data models... (Table 16.1)

Gathering ASRs by Interviewing Stakeholders

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

Gathering ASRs by Understanding the Business Goals

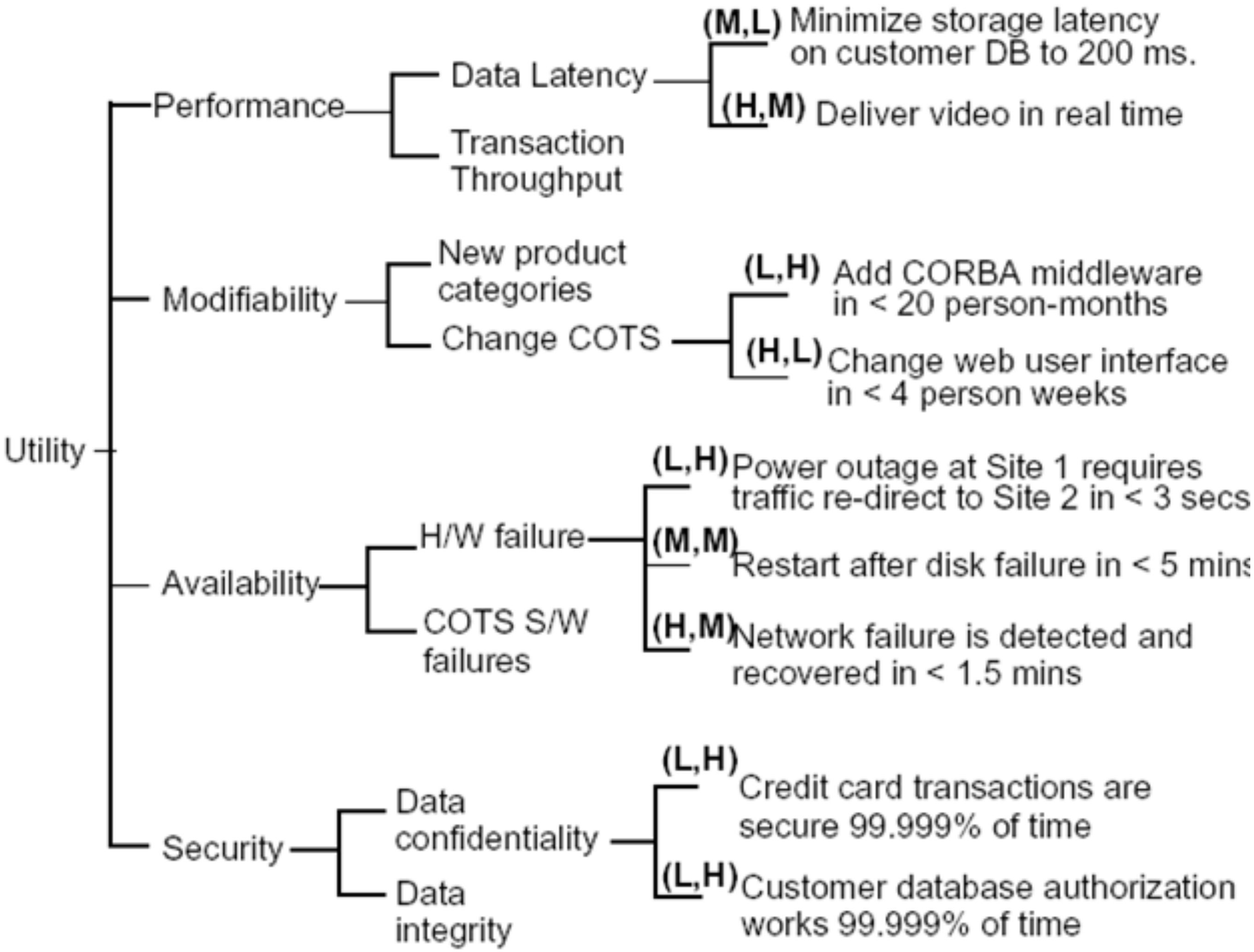


- Business goals will
  - Frequently lead directly to ASRs
    - Often lead to quality attribute requirements.
  - May directly affect the architecture without precipitating a quality attribute requirement at all

Capturing ASRs in a Utility Tree (1)

- A utility tree begins with the word "utility" as the root node
  - Utility is an expression of the overall "goodness" of the system
- Listing under the root the major quality attributes that the system is required to exhibit
- Under each quality attribute, record a specific refinement of that QA
  - E.g., performance -- "data latency" and "transaction throughput
- Under each refinement, record the appropriate ASRs (usually expressed as QA scenarios).
- Evaluate the ***business value*** of the candidate ASR and the ***architectural impact*** of including it

Capturing ASRs in a Utility Tree (2)



Utility

Performance

Data Latency

(M,L) Minimize storage latency on customer DB to 200 ms.

(H,M) Deliver video in real time

Transaction Throughput

Modifiability

New product categories

Change COTS

(L,H) Add CORBA middleware in < 20 person-months

(H,L) Change web user interface in < 4 person weeks

Availability

H/W failure

(L,H) Power outage at Site 1 requires traffic re-direct to Site 2 in < 3 secs

(M,M) Restart after disk failure in < 5 mins

(H,M) Network failure is detected and recovered in < 1.5 mins

COTS S/W failures

Security

Data confidentiality

(L,H) Credit card transactions are secure 99.999% of time

(L,H) Customer database authorization works 99.999% of time

Data integrity

Designing an Architecture

Architecturally Significant Requirements

Design Strategy

Attribute Driven Design

Decomposition

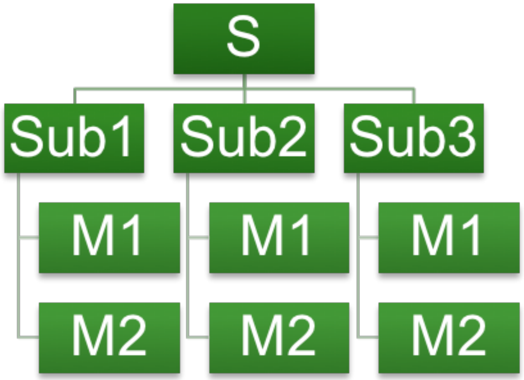
- Quality attributes refer to the system as a whole. So the design will begin with the system as a whole.
  - As it is decomposed, the quality attribute requirements can also be decomposed and assigned to the elements of the decomposition
- Constraints such as preexisting components on the design shall be accommodated by a decomposition strategy
  - The design shall accommodate the constraints and achieves the quality and business goals for the system.

Decomposition: Motivation

- Motivations for decomposition includes:
  - Semantic coherence
  - Functionality support for quality attributes
  - For achieving quality attributes
  - Existing components
  - …

Decomposition: Appropriate Level of Abstraction

System

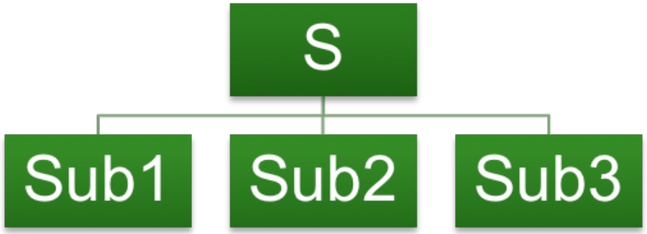


S

Sub1 Sub2 Sub3

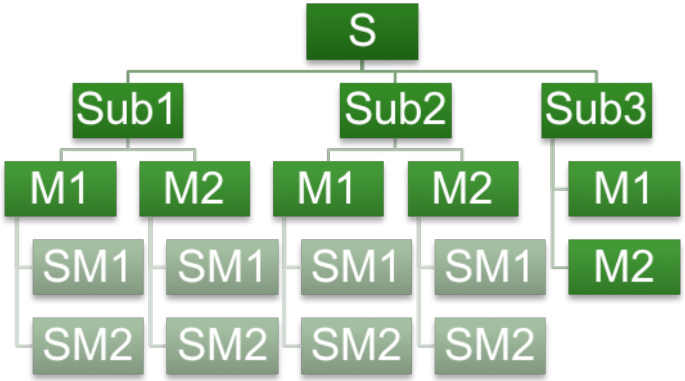
M1 M1 M1

M2 M2 M2



S

Sub1 Sub2 Sub3



S

Sub1 Sub2 Sub3

M1 M2 M1 M2 M1

SM1 SM1 SM1 SM1 M2

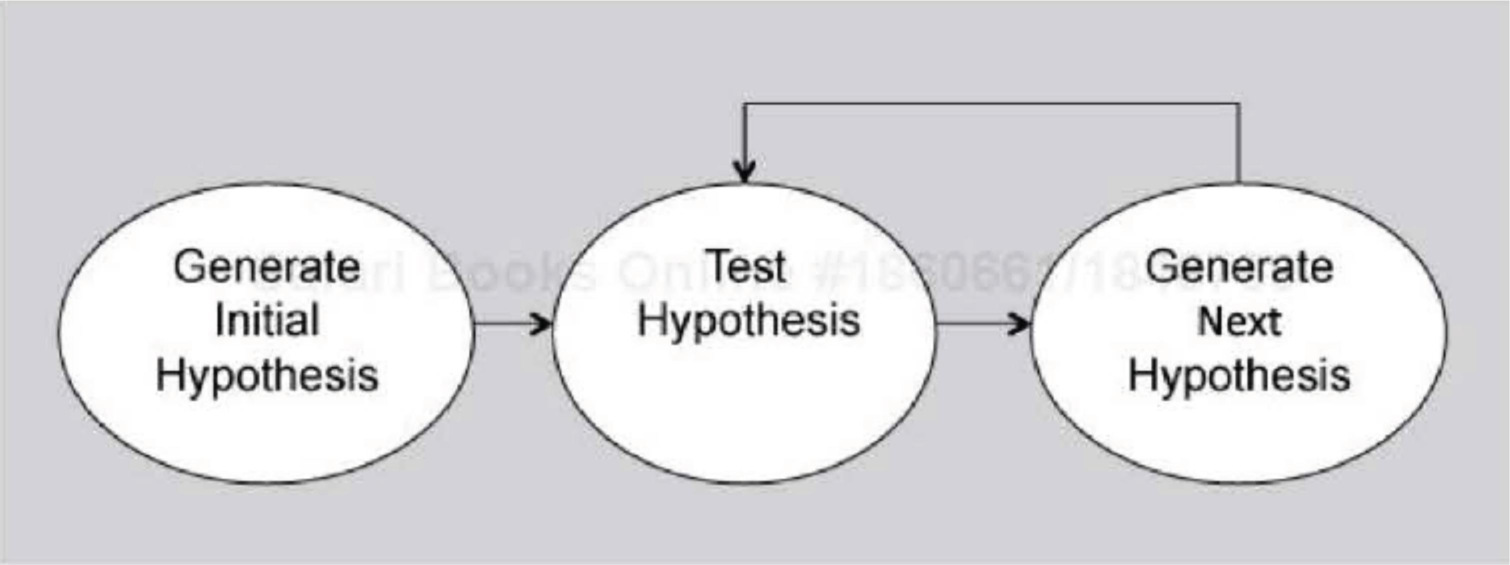
SM2 SM2 SM2 SM2

. . . . . .

Designing to ASRs

- ASRs are the requirements that drive the architectural design
  - You must design to satisfy these requirements
- What about the non-ASR requirements?
  - Met by the current design
  - Met by slightly adjusting current design
  - Not met by the current design
- Design for all of the ASRs or one at a time?
  - Experienced architects can catch multiple ASRs

Generate and Test



- The generate-and-test approach
  - Creating the initial hypothesis
  - Choosing the tests
  - Generating the next hypothesis
  - Terminating the process

Where Does the Initial Hypothesis Come From?

- Existing systems
  - It is likely that systems already exist that are similar to the system being constructed
- Frameworks available to the project
  - E.g. Web applications, decision support systems etc. Note that the framework may constrain your initial design hypothesis
- Known architectural patterns and tactics
- Domain decomposition
- Design checklists
  - The point of using a checklist is to ensure completeness

Designing an Architecture

Architecturally Significant Requirements

Design Strategy

Attribute Driven Design

Attribute Driven Design (ADD)

- An iterative, "divide and conquer” approach aims to satisfy architecturally significant requirements
- At each iteration, ADD will
  - Choose a part of the system to design
  - Marshal all the architecturally significant requirements for that part
  - Create and test a design for that part

Input to ADD

- A set of ASRs
  - Functional requirements as use cases
  - Quality requirements expressed as system-specific quality scenarios
  - Constraints
- A context description
  - What are the boundaries of the system being designed?
  - What are the external systems, devices, users, and environmental conditions with which the system being designed must interact?

Output of ADD

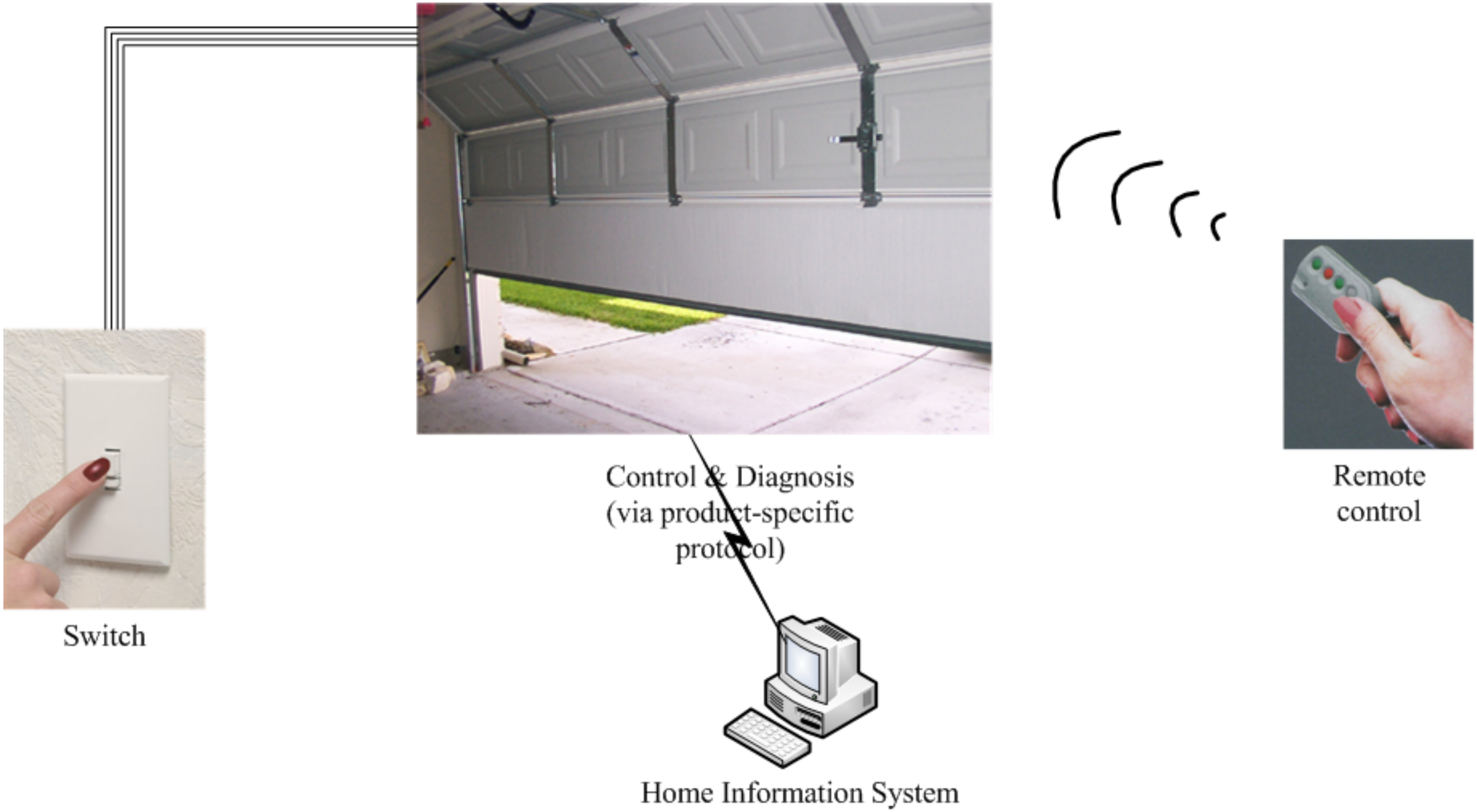
- A set of sketches of architectural views
  - Module view + other views as appropriate
  - These views together will identify a collection of architectural elements and their relationships or interactions

- The system is described as a set of containers for functionality and the interactions among them
  - The resulted design is critical for achieving the desired qualities and provides a framework for achieving the functionality

Case Study---Garage Door Opener

- Task: design *a product line architecture* for a garage door opener within a home information system

The Garage Door Opener---Product Line Design



Switch

Control & Diagnosis (via product-specific protocol)

Remote control

Home Information System

ASRs (1)

- The device and controls for opening and closing the door are different *for the various products in the product line*.
  - Various devices (various actuators, etc.)
  - Various controls
    - Controls from within a home information system
    - Various switches
    - Various remote controls

  - *The product architecture for a specific set of controls should be directly derivable from the product line architecture.*

ASRs (2)

- The ***processor*** used in different products will differ
  - *The product architecture for each specific processor should be directly derivable from the product line architecture*

ASRs (3)

- If an obstacle (person or object) is detected by the garage door during descent, it must halt (alternately re-open) ***within 0.1 second***

ASRs (4)

- The garage door opener should be accessible for diagnosis and administration from within the home information system using ***a product-specific diagnosis protocol***.
  - It should be possible to directly produce an architecture that reflects this protocol

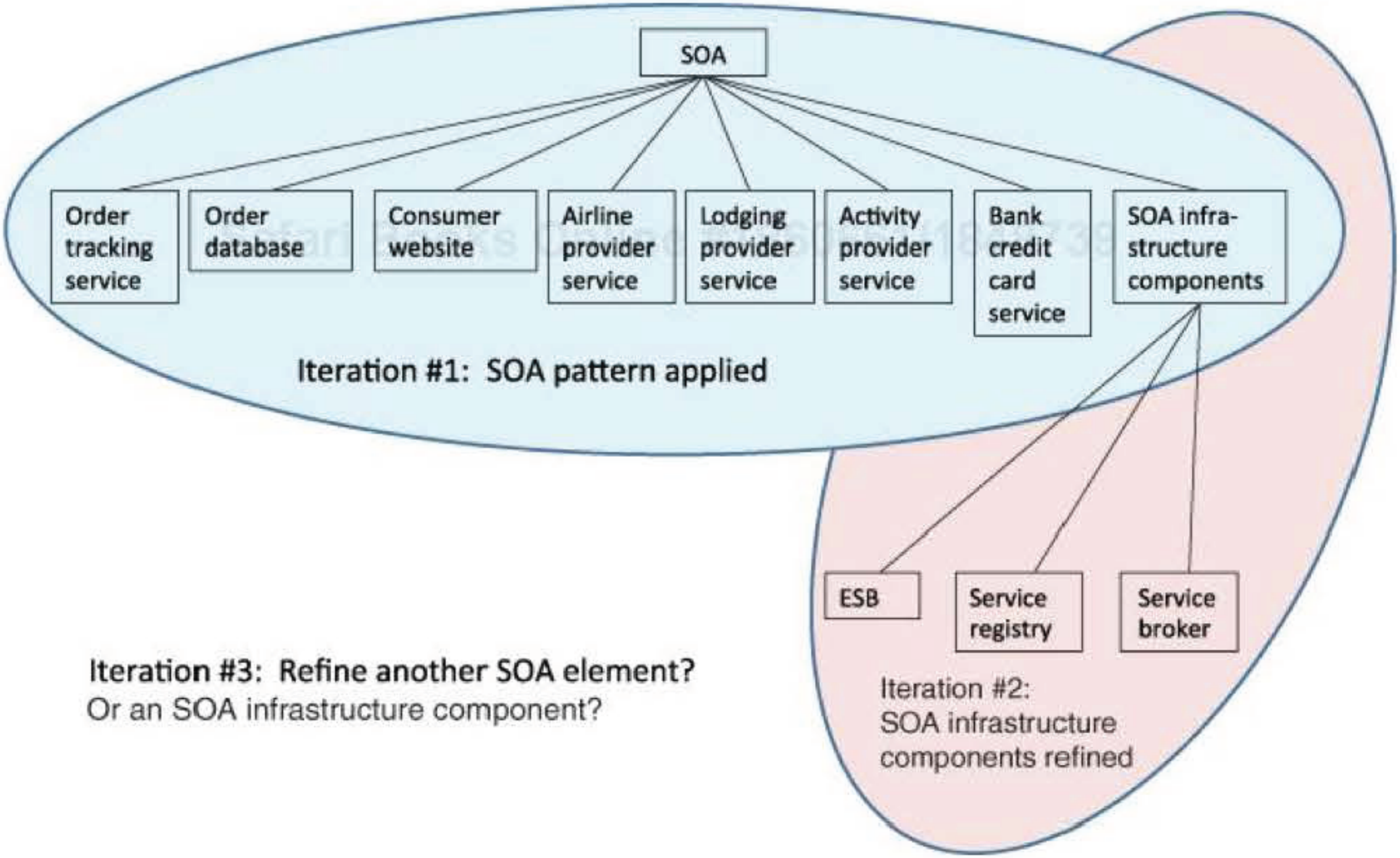
The Steps of ADD

- ADD is a five-step method:
  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 (1)

- Start with entire system
  - Inputs for this element need to be available
    - Constraints, functional and quality requirements
- 1st iteration: create a collection of elements that together constitute the entire system
- 2nd iteration: take one of these elements and design it, resulting in still finer-grained elements
- 3rd iteration: which one to choose?

Step 1: Choose an Element of the System to Design (2)



SOA

Order tracking service

Order database

Consumer website

Airline provider service

Lodging provider service

Activity provider service

Bank credit card service

SOA infra-structure components

Iteration #1: SOA pattern applied

ESB

Service registry

Service broker

Iteration #2:
SOA infrastructure
components refined

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

Step 1: Choose an Element of the System to Design (3)

- Two main refinement strategies to pursue with ADD: breadth first and depth first
- Factors affecting the order to work through ADD:
  - Personnel availability: usually depth first
  - Risk mitigation: usually depth first
    - E.g., prototyping for the risky part
  - Deferral of some functionality or quality attribute concerns
- All else being equal, a breadth-first refinement strategy is preferred for it allows for
  - apportion the most work to the most teams soonest
  - consideration of the interaction among the elements at the same level

Step 2: Identify the ASRs for This Element

- If the chosen element for design in step 1 is the whole system, then a utility tree can be a good source for the ASRs
  - Otherwise, construct a utility tree specifically focused on this chosen element, using the quality attribute requirements that apply to this element
- In our case study, the ASRs are as listed

Step 2: Identify the ASRs for This Element

- In our case, requirements to be addressed at this level of design include:
  - Real-time performance requirement *
  - Modifiability requirements
  - Support online diagnose via product-specific protocol
- Requirements are not treated as equals
  - Less important requirements are satisfied within constraints obtained by satisfying more important requirements
    - This is a difference of ADD from other SA design methods

Reading Assignment

- Read Chapter 17 of the textbook