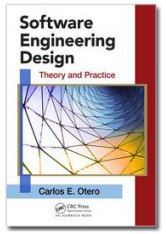
CHAPTER 3: PRINCIPLES OF SOFTWARE ARCHITECTURE

SESSION I: FUNDAMENTALS OF SOFTWARE ARCHITECTURE

Software Engineering Design: Theory and Practice
by Carlos E. Otero



SESSION'S AGENDA

- Fundamentals of software architecture
  - ✓ What is software architecture?
  - ✓ Why is it important?
- Key tasks in software architecture
  - ✓ Stakeholders concerns
  - ✓ Identify architectural views, styles, and patterns
  - ✓ Identify major component's and interfaces
  - ✓ Evaluating and validating the Architecture
  - ✓ Introducing policies for design synchronicity
- Problem-solving in architecture
- What's next…

FUNDAMENTALS OF SOFTWARE ARCHITECTURE

- Let's get straight to the point, formally, we define software architecture as
  - ✓ The *foundational software design* activity that evaluates and translates *software requirements* (both functional and non-functional) into a *collection of design elements* that specify structural and behavioral aspects of the *major components of the system*, together with their provided quality and interrelationships required to *support the detailed design and construction* of software systems.
  - ✓ The product resulting from such activity.

- From this definition, a few things are of interest and need further explanation:
  - ✓ Foundational design
  - ✓ Transforming requirements
  - ✓ Collection of design elements
  - ✓ Major system components together with their provided quality
  - ✓ Support detailed design and construction

- Let's take a more detailed look at these…

FUNDAMENTALS OF SOFTWARE ARCHITECTURE

- On " *… foundational software design…*"
  - ✓ Software architecture provides the groundwork essential for meeting requirements
    - This applies to both functional and non-functional requirements.
  - ✓ This suggest that architecture is not an optional activity or activity performed as a means of documenting software systems long after they are implemented.
    - New development efforts should approach software architecture as a forward engineering activity that leads to the implementation of systems and not as a reverse engineering mechanism for documentation.
  - ✓ As foundational design, it is where designing for quality begins. Not considering quality attributes of the system during the software architecture activity can be a grave mistake!

- On "*…translates software requirements…*"
  - ✓ Requirements is a tricky business! Inexperienced engineers tend not to see the many traps behind the requirements effort.
  - ✓ Assuming that every requirement is captured and well understood, design elements need to be created so that there is a mapping between requirements and design element.
    - One design element (e.g., UML component) can be assigned one or more requirements.
    - When we do this, we transform one or more requirements from textual form into a graphical form that represents (in the design domain) the services that need to be provided by the system.
    - This allows us to map requirements to design elements and provide the means for tracing requirements through the development life-cycle.

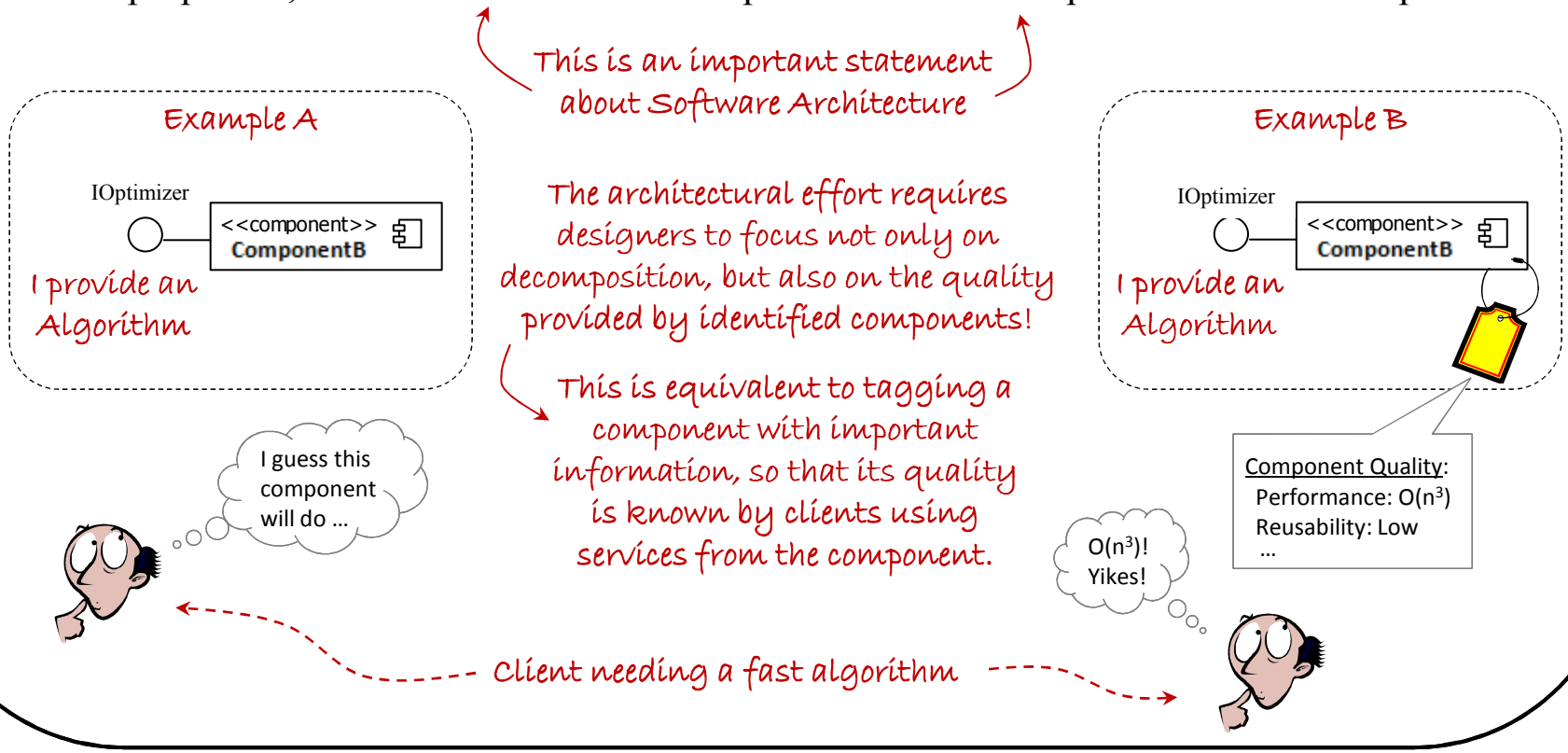
FUNDAMENTALS OF SOFTWARE ARCHITECTURE

- More on "*…translates software requirements…*"
  - ✓ To create design elements from requirements, it is assumed that requirements are understood. Sometimes this is not the case.
    - For example, some may think that "*The system shall perform fast.*" specifies a requirement that can be used to create design elements.
    - Such statements create problems for designers. These problems need to be resolved before we can translate from requirement to design domain.
    - This suggest that architects must be proficient in activities related to requirements engineering. We will cover such situations later on in the course.

- On "*…collection of design elements…*"
  - ✓ This suggest that no one structure or diagram can fully describe the software architecture.
    - Think about this: can you evaluate a system's usability and performance with one diagram?
  - ✓ This suggest that architectures are composed of multiple structures.
  - ✓ We will see examples of this later on…

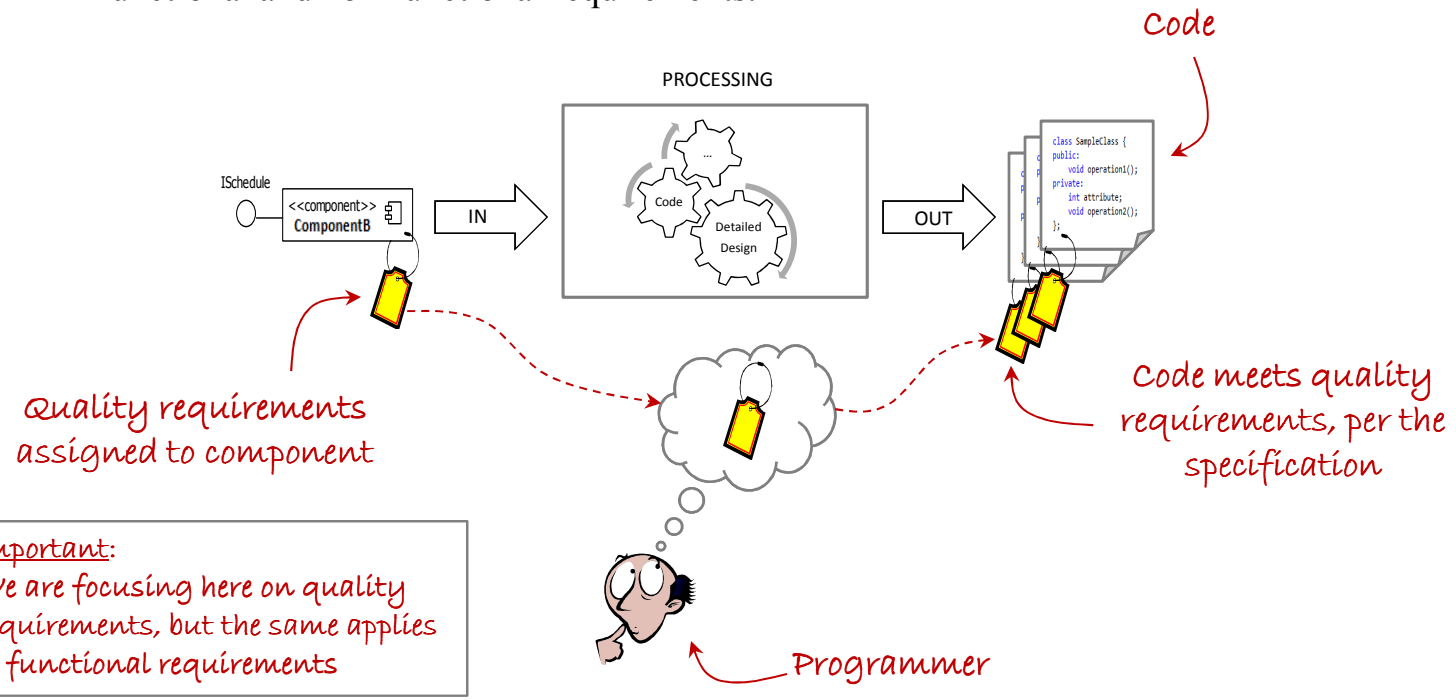
FUNDAMENTALS OF SOFTWARE ARCHITECTURE

- On "*…major components of the system, together with their provided quality…*"
  - ✓ This suggest that software architecture works at a distinct level of abstraction that differs from other forms of design, such as detailed design.
  - ✓ This means that architectural work focuses on the major components, the quality properties, and services that these components exhibit and provide to other components.



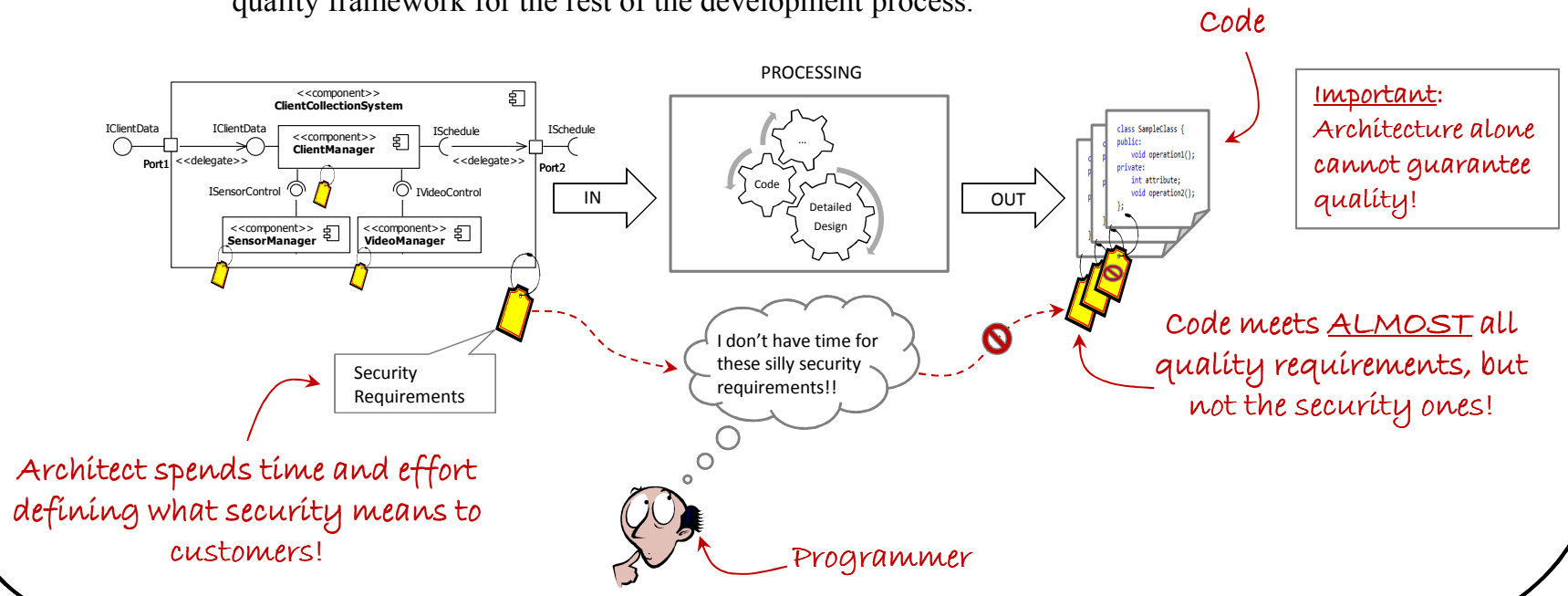
FUNDAMENTALS OF SOFTWARE ARCHITECTURE

- More on "*...major components of the system, together with their provided quality...*"
  - ✓ Expected quality requirements identified during architecture trickles down all the way to the implementation of components.
    - This provides developers with enough information to produce code that meets the system's functional and non-functional requirements.



FUNDAMENTALS OF SOFTWARE ARCHITECTURE

- On "*...support detailed design and construction...*"
  - ✓ Although architecture focuses on the quality properties of the system, it must also result in a design that supports efficient detailed design and construction of the system.
    - Even though Architects do not need to be proficient in a particular programming language, they benefit greatly from having proficiency in general programming design concepts.
  - ✓ This suggests that architecture alone *cannot guarantee the quality of the system*!
    - Since work performed during subsequent activities and phases significantly shapes the system's quality, software architecture can only play the initial (indispensible) role of establishing the design quality framework for the rest of the development process.



KEY TASKS IN ARCHITECTURAL DESIGN

- From the software architecture definition, we have been able to derive key tasks that need to be performed during software architecture.
  - ✓ However, defining the structure of software systems requires consideration of many other project-specific aspects and how those aspects relate to the organization's goals.
  - ✓ Formally, key tasks that need to be performed during the software architecture design effort include:
    - Identifying stakeholders concern
    - Identifying appropriate architectural views
    - Identifying architectural styles and patterns
    - Identifying influences of architectural decisions in organizations
    - Identifying the system's major components and interfaces
    - Evaluating and validating the architecture
    - Establishing policies for ensuring architectural design synchronicity

- Let's discuss these in more detail…

KEY TASKS IN ARCHITECTURAL DESIGN

- Identifying stakeholders concerns
  - ✓ Stakeholders are persons, groups, or organizations that have a direct or indirect stake in the system.
    - They include systems engineers, software engineers, hardware engineers, project management, customers, testing teams, quality assurance teams, members of the configuration management team, etc.
  - ✓ A stakeholder's concern provides high-level information about desired characteristics of the software system.
    - The software architect must ensure that the software to be developed addresses the concerns of all stakeholders.
    - Stakeholders' concerns serve as driving force behind architectural decisions
  - ✓ The software architect must identify and understand the different ways stakeholders influence the system.
    - These need to be elicited before any design effort can begin.

Important:
Stakeholders' concerns serve as driving force for architectural decisions

KEY TASKS IN ARCHITECTURAL DESIGN

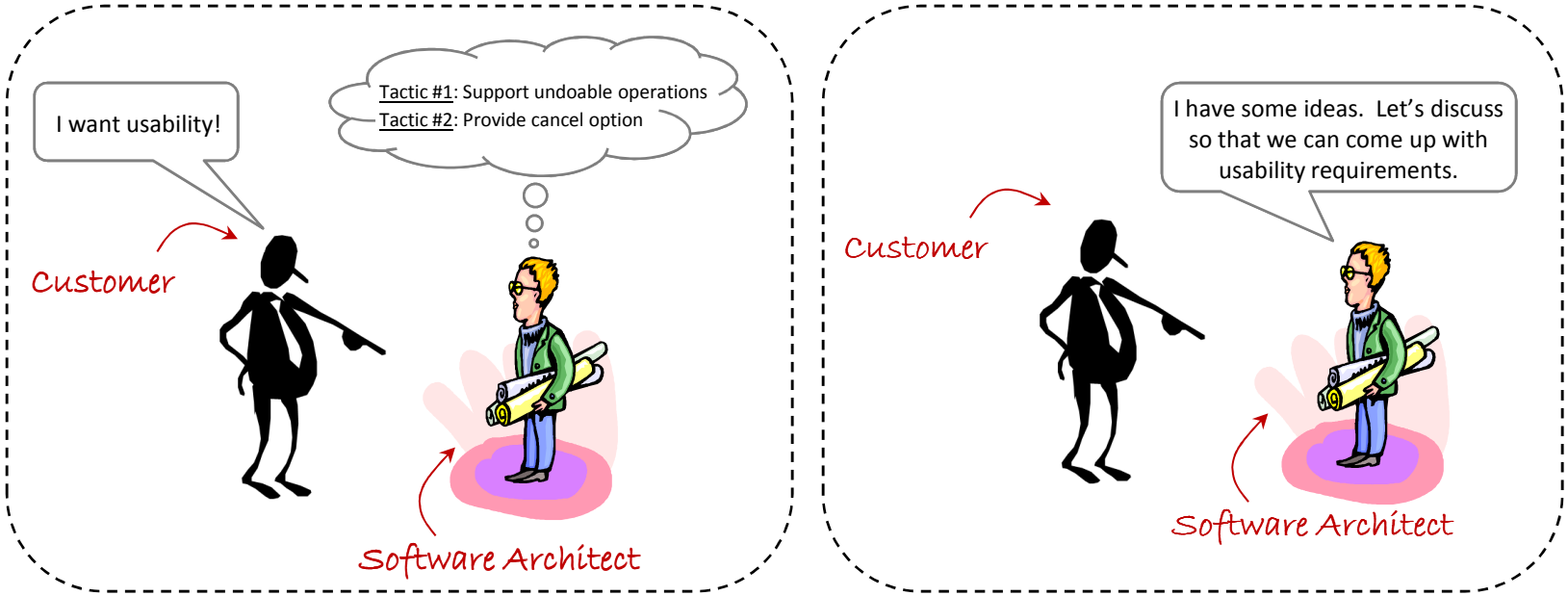
- So far, we've vaguely mentioned the concepts of **quality** and stakeholders' **concerns**.
  - ✓ These high-level concerns are often related to the desired quality of the system.
- Let's formally define some important quality attributes of software systems.

Quality	Description
Usability	The degree of complexity involved when learning or using the system.
Modifiability	The degree of complexity involved when changing the system to fit current or future needs.
Security	The system's ability to protect and defend its information or information system.
Performance	The system's capacity to accomplish useful work under time and resource constraints.
Reliability	The system's failure rate.
Portability	The degree of complexity involved when adapting the system to other software or hardware environments.
Testability	The degree of complexity involved when verifying and validating the system's required functions.
Availability	The system's uptime.
Interoperability	The system's ability to collaborate with other software or hardware systems.

- Notice that these quality attributes also describe high-level information about desired characteristics of the software system.
  - ✓ In their current form, they are not sufficient to develop the system.
  - ✓ For a system to exhibit any of these qualities, design decisions must be made to support the achievement of these qualities. These design decisions are referred by Bass, Clements, and Kazman as *Tactics* [1].

KEY TASKS IN ARCHITECTURAL DESIGN

- According to Bass, Clements, and Kazman, a *tactic* is a design decision that influences the control of a quality attribute response [1].



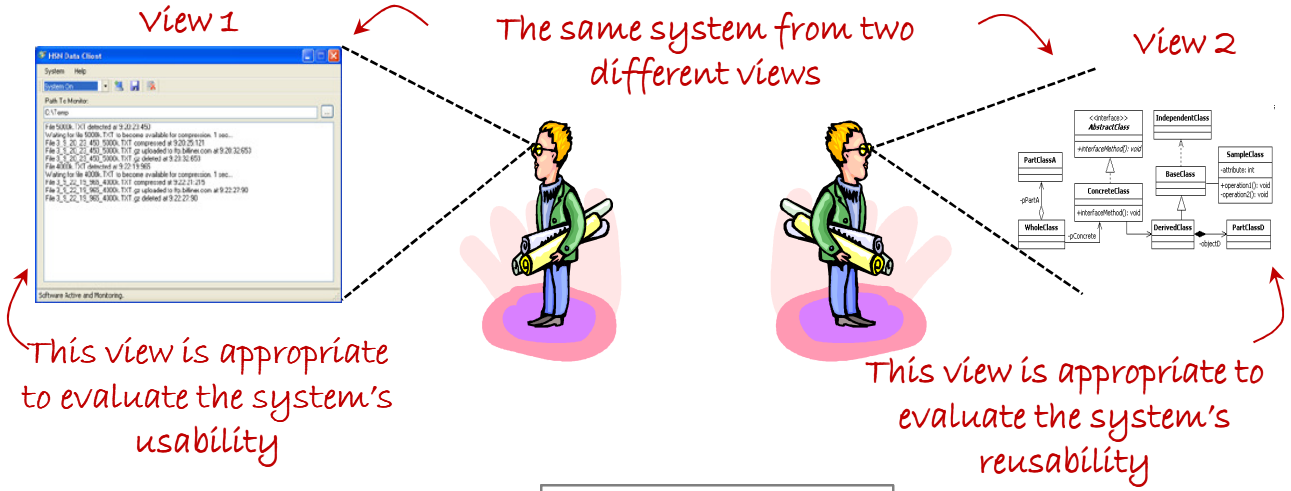
Important:
In many cases, these quality goals fall through the requirement phase, Leaving the architect responsible for specifying requirements to meet these quality attributes. During this process, tactics are identified for each desired quality attribute.

KEY TASKS IN ARCHITECTURAL DESIGN

- Tactics for *Security* [1]
  - ✓ Resisting Attacks
    - Authenticating users, Limit exposure, Limit access, only on need-to-know basis , etc.
  - ✓ Detecting Attacks
    - Intrusion detection
- Tactics for *Testability*
  - ✓ Event logging
    - Log data and operations throughout the system. Allow testers to enable/disable this feature so that when enabled, events are displayed in the console to give insight into the system's operations and data.
- Tactics for *Modifiability* [1]
  - ✓ Localize changes
    - Modularization, abstraction, encapsulation
  - ✓ Prevention of ripple effects
    - Encapsulation, reduce coupling
- Tactics for *Availability*
  - ✓ Redundancy, Task monitor, Watchdog timer , etc.
- Tactics for *Performance* [1]
  - ✓ Increase computation efficiency, reduce computational overhead, introduce concurrency, etc.

KEY TASKS IN ARCHITECTURAL DESIGN

- Identifying appropriate *architectural views*
  - ✓ In complex software systems, there can be numerous stakeholders with different backgrounds.
    - These stakeholders have different perception about the system, which influence the way they evaluate the system's design
  - ✓ For this reason, architectural designs must support different architectural views used to evaluate the design from a particular stakeholder's perspective.
  - ✓ An architectural *view* is a representation of the system.
    - Different representations are required to evaluate certain properties of the system.



KEY TASKS IN ARCHITECTURAL DESIGN

- Identifying Architectural Styles and Patterns
  - ✓ The concept of architectural styles and patterns are fundamental to the efficient creation of software architectures.
  - ✓ They provide an overall strategy for designing a family of software systems.
  - ✓ They provide reusable architectural solutions, documented in a way that is easily understood and applied.
    - For this reason, from the logical perspective, this is one of the first tasks performed during architecture.
  - ✓ Decisions based on styles and patterns benefit from years of documented experience.
  - ✓ Today, numerous styles and patterns exist so architects must be aware of these so that they can identify and determine the appropriateness of a particular style or pattern for their system's design.

> Important:
> Don't get stuck on terminology! As you will see later on, Architectural Styles and Architectural Patterns refer to similar concept. During Detailed Design, you will also hear about Design Patterns!

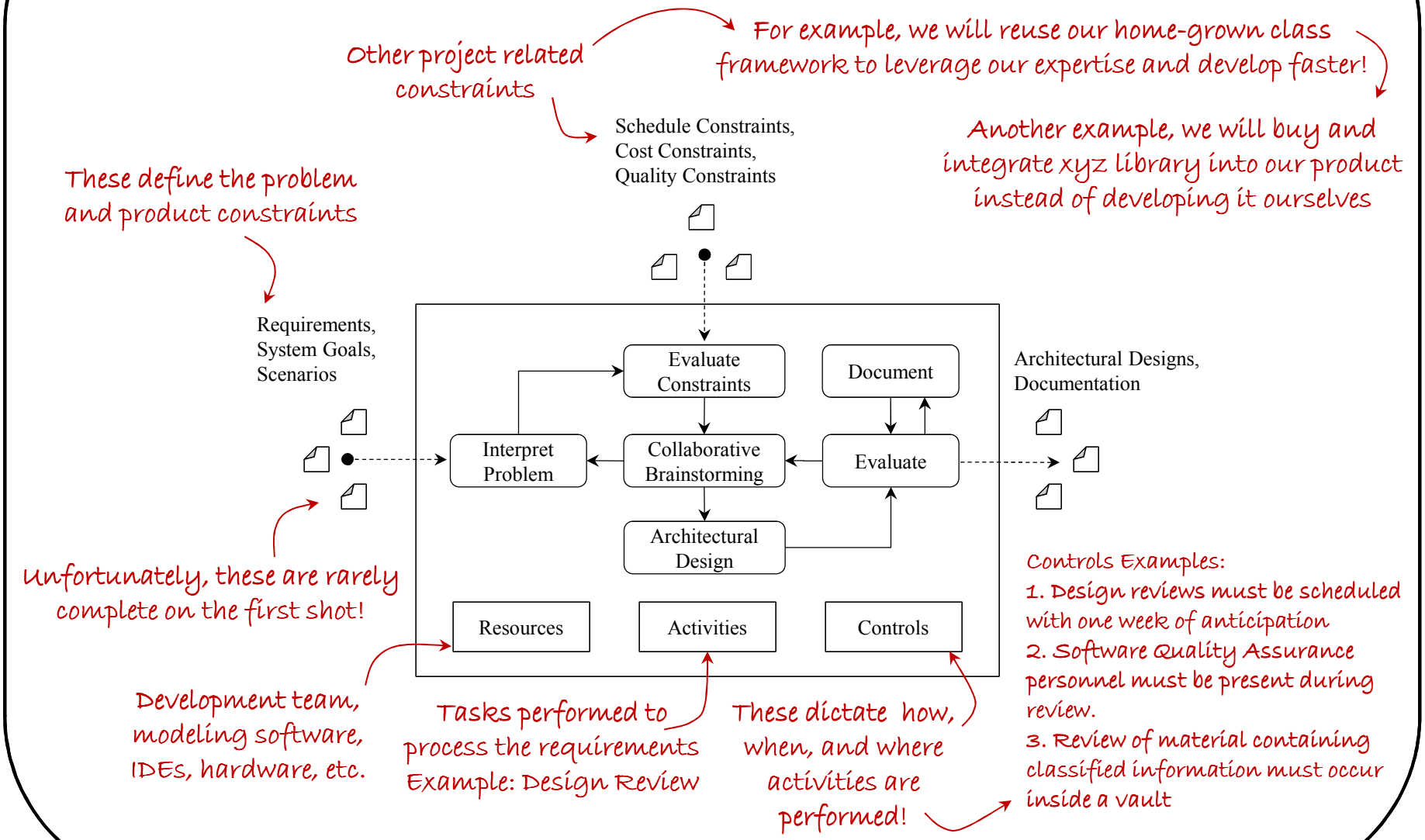
- Identifying System Interfaces
  - ✓ Interfaces are defined for components residing within single physical nodes within a single or different process space, or for components residing on different physical nodes.

- Identifying Impact of Architectural Decisions in Organization
  - ✓ Impact on customer base
  - ✓ Impact on budget and schedule
  - ✓ Impact on resource availability

KEY TASKS IN ARCHITECTURAL DESIGN

- Evaluating and Validating the Architecture
  - ✓ Long and iterative process
  - ✓ Failure to do so can have significant impact in effort and cost incurred to develop the system.
    - It is well known that defects found earlier on in the development process take much less effort to correct than if found at later stages.
  - ✓ The result should determine if the software architecture is *sufficiently* complete to support the development of the system.
  - ✓ We'll have more to say about this later on in the course…
- Introduce Policies for Design Synchronicity
  - ✓ Design synchronicity is a measurement of the degree of how well the software implementation reflects its design, both in architectural form and detailed design form.
  - ✓ Obviously, we want high synchronicity, but this is not always the case.
    - It is very easy to deviate from design, especially on multi-year efforts.
  - ✓ For any software architecture effort to result in successful implementation, all subsequent phases and activities must be synchronized with the architecture.
    - To do this, a well-defined and understood process must be in place.
    - This includes the maintenance phase, which can go on for years after a software has been deployed!

Problem Solving During Architecture



WHAT'S NEXT…

- In this session, we presented fundamentals concepts of software architecture and key tasks that need to be performed during software architecture, including:
  - ✓ Identifying stakeholders concerns
  - ✓ Identify architectural views, styles, and patterns
  - ✓ Identify major component's and interfaces
  - ✓ Evaluating and validating the Architecture
  - ✓ Introducing policies for design synchronicity

- We've also mentioned issues with software quality and requirements, but we have not presented the details involved in actually generating good requirements in case that (as architects) we are presented with this problem.
  - ✓ Next session will focus on requirements engineering, providing enough information to successfully create good requirements.

REFERENCES

- [1] Bass, Len, Paul Clements, and Rick Kazman. Software Architecture in Practice, 2d ed. Boston: Addison-Wesley, 2003.