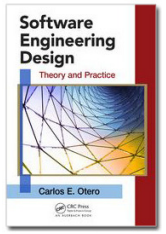
CHAPTER 3: PRINCIPLES OF SOFTWARE ARCHITECTURE

SESSION II: FUNDAMENTALS OF REQUIREMENTS ENGINEERING

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



SESSION'S AGENDA

- Software Architecture Process
  - ✓ Understand and Evaluate Requirements
  - ✓ Design the Architecture
  - ✓ Evaluate the Architecture
  - ✓ Document the Architecture
  - ✓ Monitor and control implementation
- Requirements Engineering
  - ✓ Elicitation
  - ✓ Analysis
  - ✓ Specification
  - ✓ Validation
- What's next..

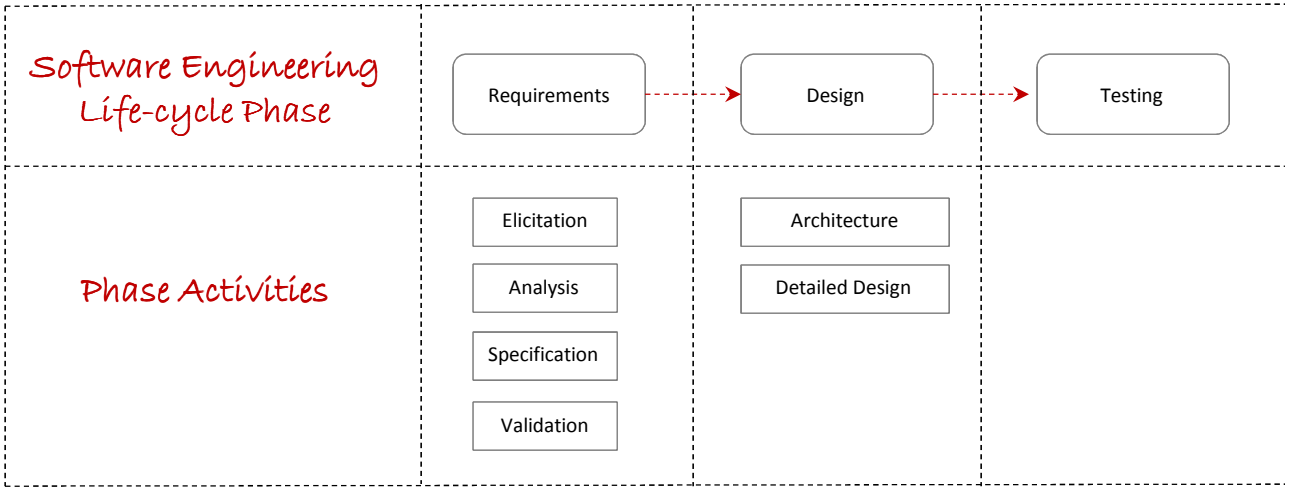
SOFTWARE ARCHITECTURE PROCESS

- On a larger scale, the process for creating software architectures can be executed using the following tasks:
  - ✓ Understand and evaluate requirements
  - ✓ Design the architecture
  - ✓ Evaluate the architecture
  - ✓ Document the architecture
  - ✓ Monitor and control implementation

- Software architects spend a great deal of time working with software requirements.
  - ✓ Even after requirements are specified, software architects find themselves going back and forth between requirements and design.
  - ✓ In some cases, architects are completely immersed in the requirements phase, playing a key role in the specification of requirements.

UNDERSTAND AND EVALUATE REQUIREMENTS

- Requirements engineering
  - ✓ The discipline within software engineering that is concerned with the systematic approach to requirements specification, mainly through the following activities:
    - Elicitation
    - Analysis
    - Specification
    - Validation
- Similar to the design phase, the requirements phase can be broken down into well defined activities



UNDERSTAND AND EVALUATE REQUIREMENTS

- Elicitation
  - ✓ Activity that deals with identifying stakeholders, uncovering what the customer needs and wants, and with determining the non-functional requirements.
  - ✓ Begins by identifying all *sources* of information that can be used to generate requirements.
    - These vary from project to project and can provide bias information to shape the system in a way that addresses their particular needs.
  - ✓ Different sources can have similar but different visions for the system.
    - Common sources of requirements include:
      - Stakeholders
      - Goals
      - Domain knowledge
      - Operational and organizational environment

- Elicitation Techniques
  - ✓ Interviews
  - ✓ Facilitated meetings
  - ✓ Observation
  - ✓ Scenarios

UNDERSTAND AND EVALUATE REQUIREMENTS

- On eliciting requirements using *Scenarios*
  - ✓ A popular approach for eliciting requirements.
  - ✓ Allow designers to present stakeholders with storylines about different behaviors that the system is expected to provide.
  - ✓ These storylines are born out of the perceived expected behavior by the designer and refined and validated through stakeholders' reviews.
  - ✓ They provide a valuable means for:
    - Establishing a framework for eliciting requirements
    - Identifying major system functions and details of the software
    - Providing initial insight into the required testing of the software.

- In UML, scenarios are grouped by use cases.
  - ✓ For each use case, one or more scenarios—one for the main flow of events and other for alternate scenarios—are created to document the expected system behavior and deviations from its main flow of events.
  - ✓ Scenarios represent instances of use cases, so there is one-to-many relationship between use case and scenarios.
  - ✓ Since there are no universally accepted method for documenting scenarios, they can be found in the following format:
    - Paragraphs, numbered list, tabular or graphical form, etc.
  - ✓ Without scenarios, use cases provide limited benefits.

Scenario Name and Number

UC-00-Search Product Main Scenario			
Description: This scenario describes the main flow of operations for requesting a product-search with the server system.			
Actors: Operator, Server System			
Pre-Conditions: The client and server system have been initialized.			
Requirements: MCR-001, MCR-002			
Alternate Scenario: UC-10-Invalid Search, UC-11-Connection Failure, UC-12-Response Timeout			
Revision History			
Date	Version	Description	Revised By
9/17/2010	1.0	Initial scenario creation.	John Doe
Description			
Step	Operator Action	System Action	
1	Operator enters valid product ID and clicks on the search button.	*Validates the data.* Retrieves server's communication information from config file.	
2		Establishes a connection with the server system and sends product request data to the server.	
3		Waits a maximum of 3 seconds for a server response.	
4		. .	
5		Response received and product information is displayed.	
6		*Save response data* in file system and ask user to search for another product.	
7	Operator clicks the cancel button to finish searching for products.		
Notes			
For details of data validation and saving response data to file system see use cases UC-05 and UC-06 respectively.			
Approval Signatures			
Software Engineer:			
Stakeholder(s):			
Quality Auditor:			

Scenario Description and other Information

Quality requirements are discovered using scenarios

Steps included in the scenario, classified as operator action and system response.

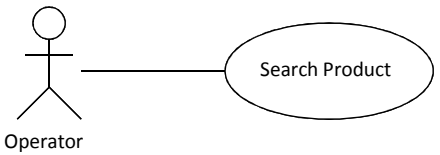
Important:
Scenarios are NOT thrown away after design! They are eventually transformed into Unit Test Cases!

Other important information regarding this scenario

Scenarios need to be approved!

Search Product's Main Scenario

Search Product Use Case



UNDERSTAND AND EVALUATE REQUIREMENTS

- Analysis
  - ✓ Requirements are analyzed in their raw form to address issues such as requirements that are contradicting, incomplete, vague, or just wrong [1].
  - ✓ Allows architects to clear the air in regard to what needs to be done before devising more detailed designs.
  - ✓ The following tasks can be performed during analysis:
    - Requirement classification
    - Requirement prioritization
    - Requirement negotiation
    - Conceptual modeling

- Requirement classification
  - ✓ Performed for identifying the nature of each requirement
    - Functional vs. non-functional
    - Product vs. process
    - Imposed vs. derived

Criteria	Description
Functional vs. Non-Functional	Classification that differentiates between requirements that specify the functional aspects of the system vs. the ones that place constraints on how the functional aspects are achieved.
Product vs. Process	Requirement placed on the system product vs. requirements placed on the process employed to build such product.
Imposed vs. Derived	Requirements imposed by stakeholders vs. requirements that are derived by the development team.

UNDERSTAND AND EVALUATE REQUIREMENTS

- Requirement prioritization and negotiation
  - ✓ Helps identify the most important functions of the software system.
  - ✓ When done properly, it can help refine the projected schedule by determining which requirements need to be processed first.
  - ✓ Can be used to determine different builds of the software
  - ✓ Can help during negotiation when conflicts between requirements are identified

- Conceptual modeling
  - ✓ Conceptual models are created to further identify the requirements by understanding their context, discovering the bounds of the software system, and conceptualizing how the system interacts with its environment.
  - ✓ In many projects, this is where architectural design begins, since system decomposition is essential to developing effective conceptual models.

UNDERSTAND AND EVALUATE REQUIREMENTS

- Specification and validation
  - ✓ Activity where the results of elicitation an analysis are formally captured and documented in an appropriate format for the use and review of all stakeholders.
  - ✓ The format of the specification varies depending on the developing organization or project; however, it is typically produced as a document, or its electronic equivalent, and is referred as the software requirements specification.

- When specifying requirements, it is important that each requirement exhibit certain characteristics desired for designing successful systems. Requirements must be:
  - ✓ Specific
  - ✓ Correct
  - ✓ Complete
  - ✓ Consistent
  - ✓ Attainable
  - ✓ Verifiable

UNDERSTAND AND EVALUATE REQUIREMENTS

- On being *specific*,
  - ✓ Requirements need to be specified in a clear, concise, and exclusive manner.
  - ✓ Clear requirements are not open to interpretation; unclear or ambiguous requirements lead to incorrect designs, incorrect implementations, and deceptive validation during test.
  - ✓ Concise requirements are brief and to the point and are therefore easier to understand.
  - ✓ Exclusive requirements specify one, and only one thing, making them easier to verify.

For example, consider the following statement:
The software needs to provide an easy-to-use interface; that is, it must be usable.

This statement provides important information to begin thinking about what the customer wants and expects from the software system.

However, in its current form, this statement is too generic to use as basis for design, construction, and verification of the software system.

For example, a console-based interface may be highly usable for developers, but not for customers, which may prefer a graphical user interface!

Designing based on different interpretations of this statement may entail sacrificing other functions that may be important to customers and users!

UNDERSTAND AND EVALUATE REQUIREMENTS

- Example of specific and nonspecific requirements:

Search the database for what?

What does this mean?

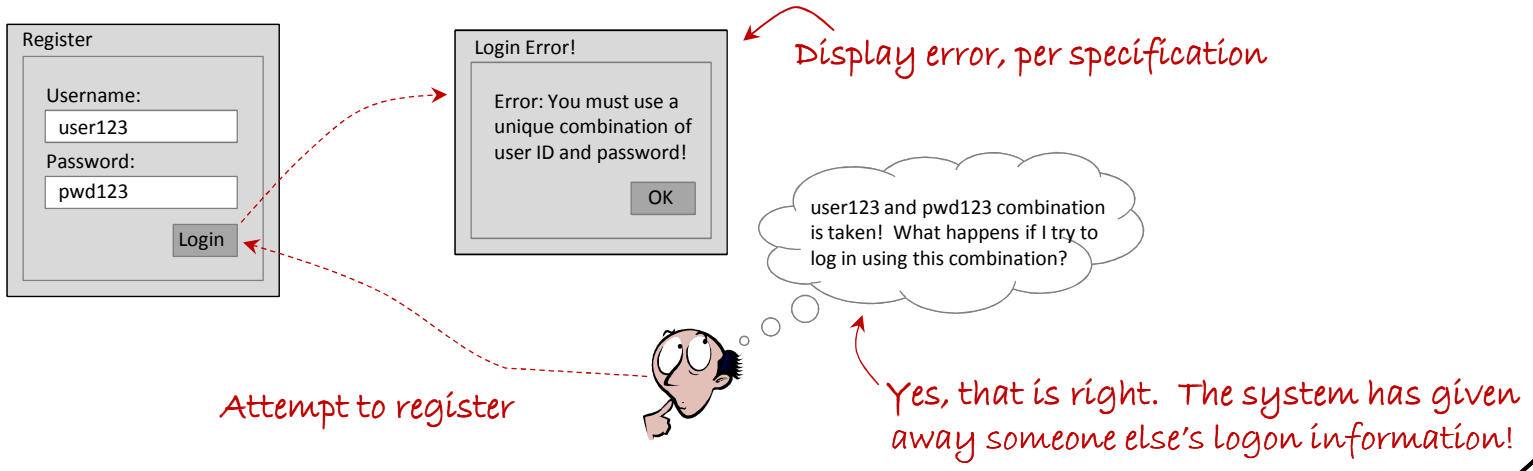
Specific	Requirement
No	The software shall search the database.
Yes	The software shall search for a product using the product ID.
No	The software shall be secure.
Yes	The software shall authenticate users with user ID and password.
No	The software shall be secure and fast.
Yes	The software shall authenticate users with user ID and password.
	Server acknowledgment message shall be sent within 1/2 second from the time a request is received.

Secure and fast?

These two requirements are specific and provide the information required to determine what secure and fast mean!

UNDERSTAND AND EVALUATE REQUIREMENTS

- On being *correct*,
  - ✓ Requirements need to be correct in the sense that they must accurately describe a desired system function.
  - ✓ In some cases, correctness of requirements is easily identified; in others, it is not.
  - ✓ Laplante [1] presents an example based on requirements for a computer security system for which it requires users to *log on using a unique combination of user ID and password*.
    - In this case, when users attempt to log on using an already existing user name or password, the system is required to reject the attempt, therefore giving insight into someone else's logon information.
  - ✓ Incorrect requirements can lead to incorrect or undesired behavior.



UNDERSTAND AND EVALUATE REQUIREMENTS

- On being *complete*,
  - ✓ Requirements must be complete both individually and as collective set.
    - This means that each requirement should be specified thoroughly so that it absolutely describes the functions required to meet some need.
    - Collectively, requirements need to provide complete specification of the software's required functionality in the software requirements specification (SRS).
  - ✓ Incomplete requirements lead to incomplete designs, which in turn leads to incomplete construction of the software system.
  - ✓ Requirements that are complete help clarify questions during construction and testing by providing information necessary to disambiguate or prevent misinterpretations of required functionality.
  - ✓ Completeness is hard because it is not always obvious or it is sometimes too difficult to determine when information is missing [1].

Notice how the original requirement is broken into two complete requirements!

This is a good requirement in the sense that it is specific and correct.

Complete	Requirement
No	The software shall generate product reports.
Yes	The software shall generate product reports consisting of product description, picture, and price.
	Product reports shall be in PDF format.

However, if we know the requirements for product reports, then a complete version of this requirement specifies this information. This help disambiguate the notion of a product report so that it can be implemented easily in code.

UNDERSTAND AND EVALUATE REQUIREMENTS

- On being *consistent*,
  - ✓ Requirements are consistent when they do not preclude the design or construction of other requirements.

- On being *attainable*,
  - ✓ Requirements that are unattainable serve no purpose.
  - ✓ Attainability can be determined for both product and process.

Can you meet this requirement today?

Attainable	Requirement
No	The software shall execute on all future operating systems.
Yes	The software shall execute on the Microsoft Windows 7 platform.

- On being *verifiable*,
  - ✓ Perhaps the most obvious desirable characteristic of requirements.
  - ✓ Requirements that cannot be verified cannot be claimed as met.
  - ✓ Inability to verify requirements point to a serious flaw early on in the development process

Verifiable	Requirement
No	The system shall maximize communication speed.
Yes	The system's data rate shall be no less than 1Mbps.

WHAT'S NEXT…

- In this session, we presented fundamentals concepts of requirements engineering, and provided enough information to successfully create good requirements.

- In the next session, we'll discuss how to design the software architecture using the popular 4+1 View Model, which includes
  - Logical view
  - Development View
  - Process View
  - Physical View
  - User View

REFERENCES

- [1] Laplante, Phillip A. *Requirements Engineering for Software and Systems*. Boca Raton, FL: Auerbach Publications, 2009.