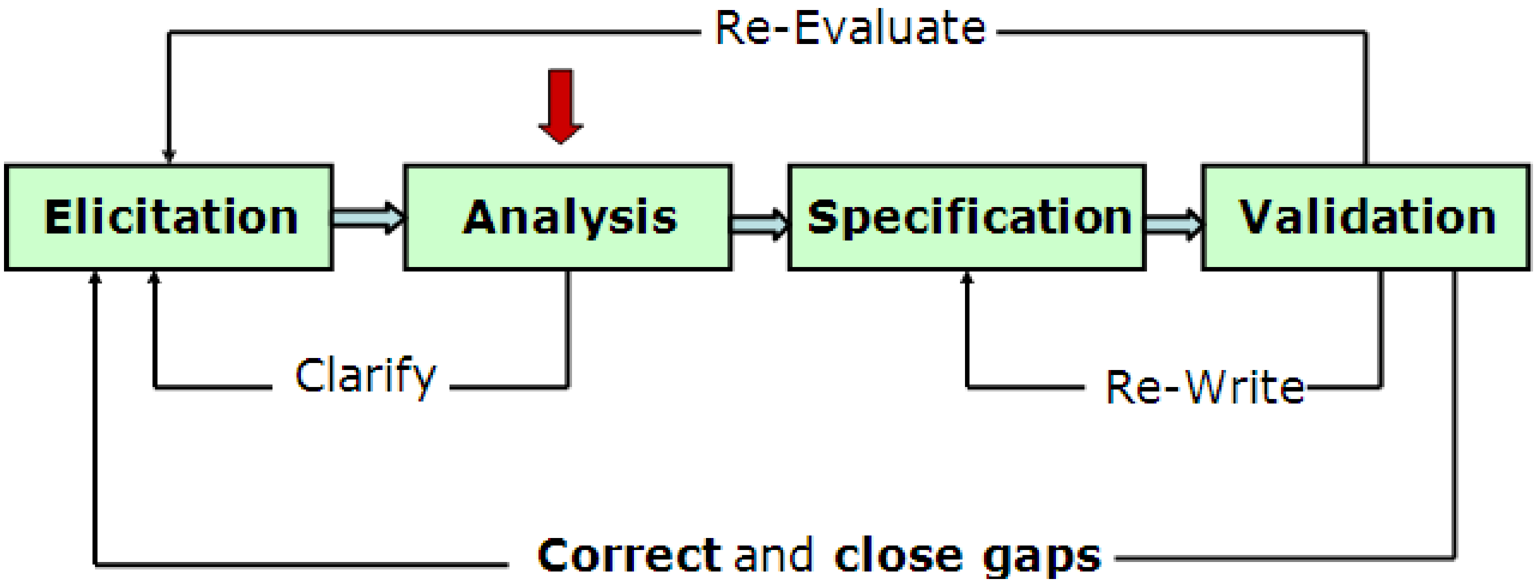
Requirement Engineering **Analysis**

References: C1.Ebook +John Vu (CMU)

Requirements Development Process



Re-Evaluate

Elicitation

Analysis

Specification

Validation

Clarify

Re-Write

Correct and **close gaps**

Iterative process - Multiple-steps, Not sequential

Requirements Analysis

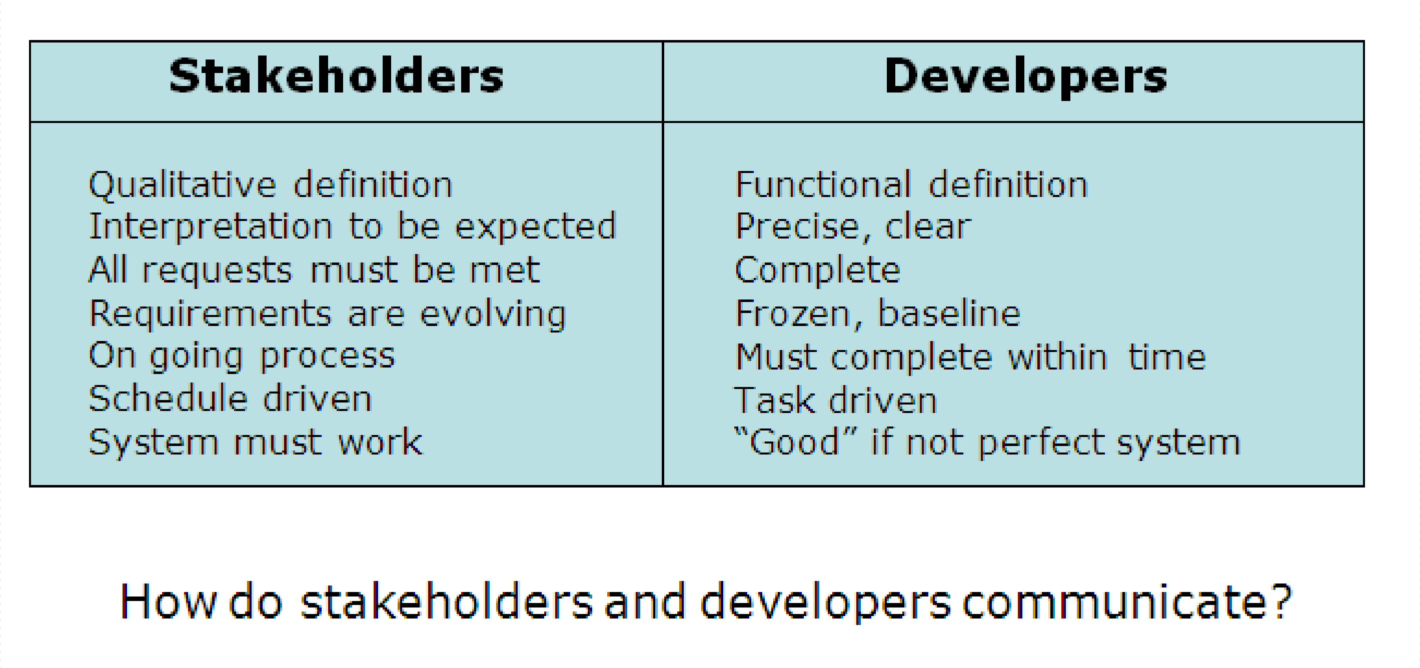
- □ The techniques of deciding which features are appropriate for the product based on stakeholders' needs.
- □ To effectively analyze requirements, Software Engineers need to understand, define and verify the requirements from the stakeholders' view so they can prioritize their needs before allocating requirements to software.
- □ Requirements elicited from stakeholders must be complete and clear to validate later

Requirements Analysis

- The process of analyzing the stakeholders' needs to prepare the definitions of system and software requirements.
- The process of transforming business needs into system descriptions with performance parameters and partitioning them into subsystems where allocation to hardware and software can take place.
- Since stakeholders and developers may have different views and expressions, to facilitate better understanding, Software Engineers should use abstract descriptions that are easy to understand and interpret

Different Views

- Objective: To define what the system will do



Stakeholders	Developers
Qualitative definition Interpretation to be expected All requests must be met Requirements are evolving On going process Schedule driven System must work	Functional definition Precise, clear Complete Frozen, baseline Must complete within time Task driven "Good" if not perfect system

How do stakeholders and developers communicate?

Requirements Analysis

- Requirements analysis results in requirements models.
- Requirements models are user requirements represented by diagrams, structured text (lists, tables, matrices) or a combination.
- Requirements analysis also focuses on trade-offs among requirements to make decisions about their relative importance to support prioritization.
- Software Engineers are responsible to analyze all requirements and collaborate with stakeholders to prioritize their needs.
- Requirements analysis is an iterative process

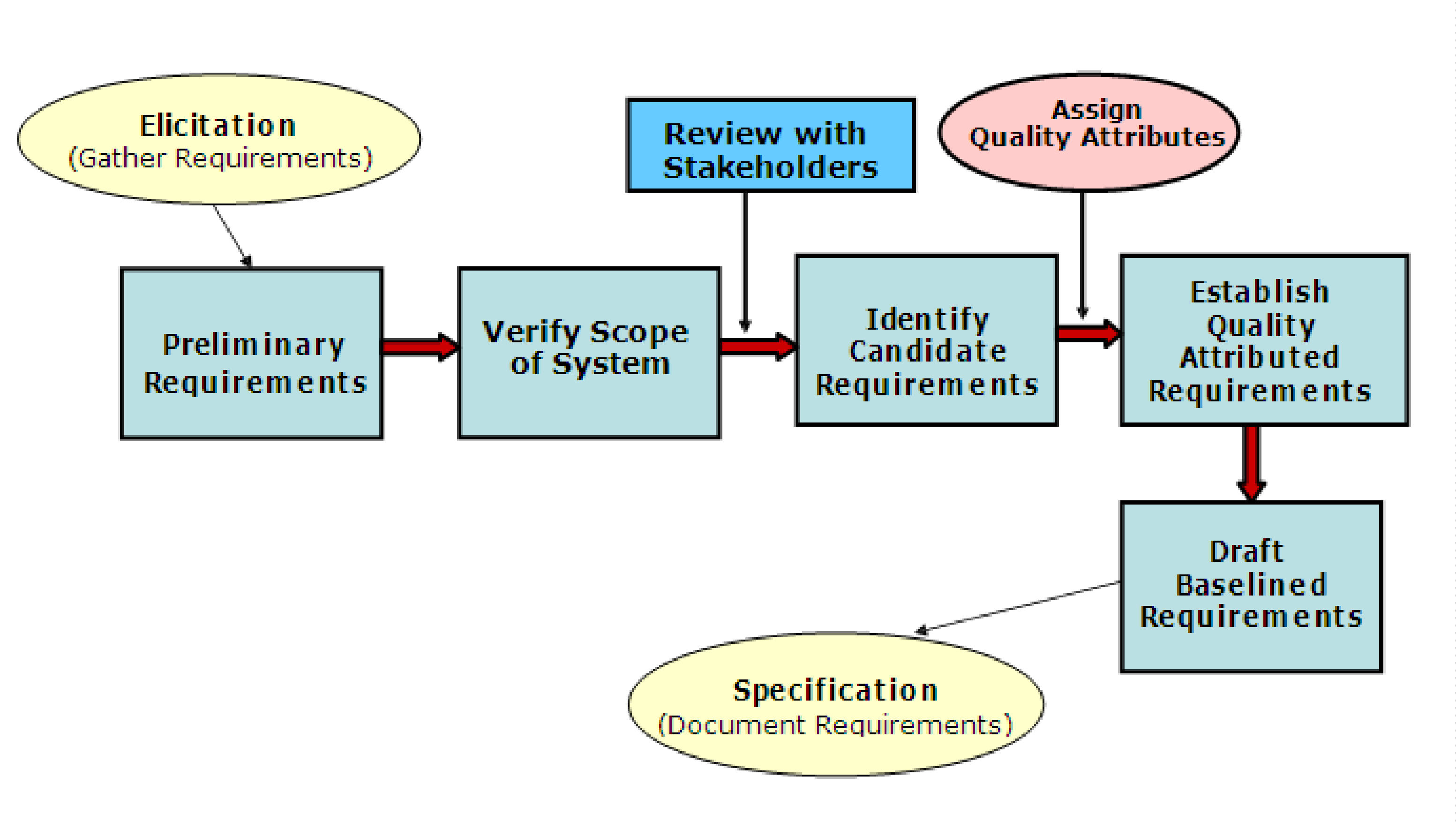
Why Requirements Models?

- Requirements models will:
  - Facilitate communication between technical and business people.
  - Allow project teams to look at different aspects of user requirements from different perspectives.
  - Uncover missing, erroneous, vague and conflicting requirements.
  - Discover interdependencies among requirements.
  - Allow stakeholders to view all requirements for better understanding.

Requirements Analysis Process

- Review all requirements to ensure they align with business goals and objectives.
- Define the project scope.
- Create detailed user requirements using multiple models that will help stakeholders better articulate their needs.
- Prioritize the requirements.
- Continue to analyze as more details emerge or are revised.
- Assign quality attributes as requirements are discovered and refined.

Requirements Analysis Process



Scope & Boundaries

- □ The scope of the requirements must be defined and all specified requirements must be relevant to the defined scope.
- □ Only take on requirements that you should and can address.
- □ Any stated 'requirement' that falls outside of the scope needs to be validated and discussed with stakeholders for including or deleting.
- □ Note: Scope states the 'overall system boundaries'.
- □ Constraints can help establish the system scope (boundaries).

Functional vs. Non-Functional

- Functional requirements describe what the system should do.
  - Things that can be captured in use cases or can be analyzed by drawing interaction diagrams, state charts, etc.
  - Functional requirements can be traced to individual module(s) of a program.
- Non-functional requirements are global constraints on a system such as development costs, operational costs, performance, reliability, maintainability, portability, robustness etc

Example Of Non-Functional

- Interface requirements
  - How will the new system interface with its environment?
  - User interfaces and 'user-friendliness'.
  - Interfaces with other systems.
- Performance requirements
  - Time/space bounds.
  - Workloads, response time, throughput and available storage space, e.g., 'the system must handle 1,000 transactions per second'

Beyond Functional Requirements

- Quality attributes (Non-Functional requirements):
  - Difficult to define.
  - Usually not mentioned, but expected by stakeholders, but different stakeholders have different preferences.
  - Important for technical considerations (during design and architectural decisions).
  - To be explored with all stakeholders during the requirements development process - not just users.
  - To be measurable and verifiable.

What Do You Mean Quality?

- Different people view quality differently:
- Tester checks for errors:
  - "Does it run without crashing?"
- Quality Assurance reviews for defects:
  - "Does it meet the specs?"
- Architect checks engineering design:
  - "Is my design reliable?"
- Senior Manager asks marketing people:
  - "How do you know that we have a quality product?"

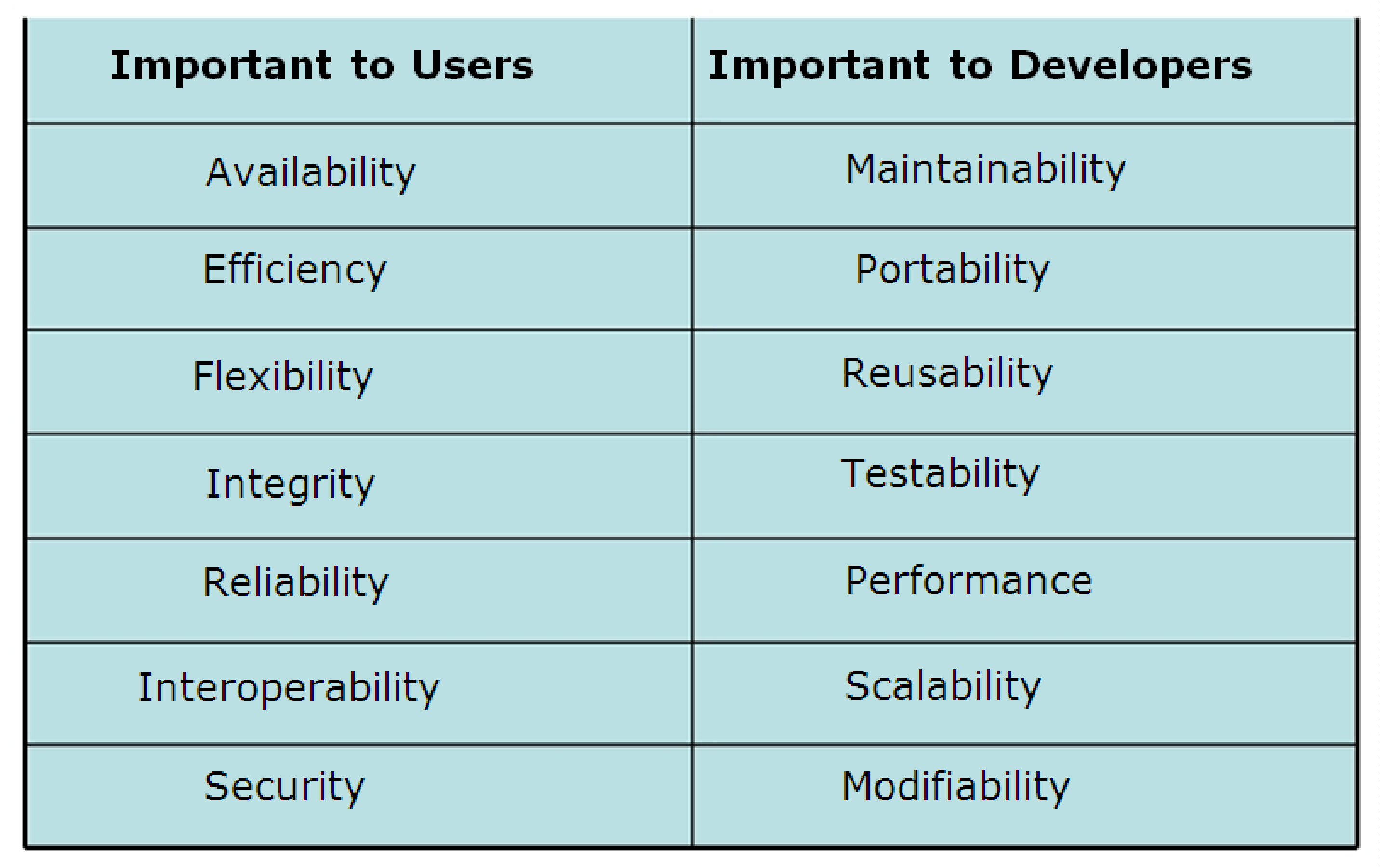
Some Quality Attributes - 1

- **Availability** : Is it available when and where I need to use it?
- **Efficiency**: How many/few system resources does it consume?
- **Flexibility:** How easy is it to add new capabilities?
- **Installability**: How easy is it to correctly install the product.
- **Integrity**: Does it protect against unauthorized access, data loss?
- **Interoperability**: How easily does it interconnect with other systems?
- **Maintainability**: How easy is it to correct defects or make changes?

Some Quality Attributes - 2

- □ **Portability:** Can it be made to work on other platforms?
- □ **Reliability:** How long does it run before experiencing a failure?
- □ **Reusability:** How easily can we use components in other systems?
- □ **Robustness:** How well does it respond to
- □ unanticipated conditions?
- □ **Safety:** How well does it protect against injury or damage?
- □ **Testability:** Can I verify that it was implemented correctly?
- □ **Usability:** How easy is it for people to learn or to use?

Quality Attributes



Important to Users	Important to Developers
Availability	Maintainability
Efficiency	Portability
Flexibility	Reusability
Integrity	Testability
Reliability	Performance
Interoperability	Scalability
Security	Modifiability

Quality Specific Scenarios

- Just like use cases are used to determine the behavior of a system (functionality), a quality driven scenario is used to determine the quality attributes of a system.
- Review specific scenarios by asking questions for each function on:
  - 1) Performance
  - 2) Scalability
  - 3) Security
  - 4) Usability
  - 5) Reliability
- For example: A threat scenario in a case of security, response time scenarios in a case of performance, and error handling scenarios in a case of reliability

Quality Attribute: Performance

- Performance is the ability of a system to allocate resources to a service request in a manner that will:
  - Satisfy timing requirements.
  - Provide various levels of service in the presence of competing requests, varying demand and varying resource availability.
- Performance is usually specified in terms of:
  - **Latency**: Time it takes to respond to a request.
  - **Throughput**: Number of events completed within observable time.
  - **Capacity**: Amount of work a system can perform and still satisfy latency and throughput requirements (no degrade).

Quality Attribute: Security

- Security can be defined as:
  - Freedom from danger, safety.
  - Protection of system data against destruction, unauthorized modification.
- Security has 3 basic types:
  - **Confidentiality**: Protection from unauthorized disclosure.
  - **Integrity**: Protection from unauthorized modification.
  - **Availability**: Protection from denial of service to authorized users.

Quality Attribute: Portability

- The degree to which software running on one platform can easily be converted to run on another.
- Considerations:
  - Portability is hard to quantify.
  - It is hard to predict all other platforms the software will be required to run.
  - Portability is strongly affected by design choices such as choice of languages, operating systems and tools that are universally available and standardized.
  - Portability requirements should be given priority for systems that may have to run on different platforms in the near future.

Quality Attribute: Modifiability

- Modifiability is about the cost of making changes.
- **Considerations:**
  - What may be changed: Function, Hardware, System, Software.
  - Who will make the change: Developer, System, User.
  - When will the change be made: During development or maintenance.
  - Changes can be classified according to:
    - Probability - How likely it will occur.
    - Frequency - How often it will occur.
    - Dependence - How does it impact others.

Quality Attribute: Reliability

- The ability of the system to behave consistently in a user-acceptable manner when operating within the environment for which it was intended. Reliability can be defined in terms of a percentage (99.999%).
- Different meanings for different applications:
  - Telephone network: the entire network can fail no more than, on average, 1hr per year, but failures of individual switches can occur much more frequently.
  - Patient monitoring system: the system may fail for up to 1hr/year, but in those cases doctors/nurses should be alerted of the failure. More frequent failure of individual components is not acceptable.
- Example of reliability requirements: 'No more than X bugs per 10KLOC may be detected during integration and testing; no more than Y bugs per 10KLOC may remain in the system after delivery

Quality Attribute: Usability

- Many systems are only technically successful (meet all functional requirements) but fall short of their expected benefits because the users find them:
  - Difficult to understand and use
  - Inconsistent
  - Needing excessive training & retraining
  - Error-prone
  - Discouraging to explore

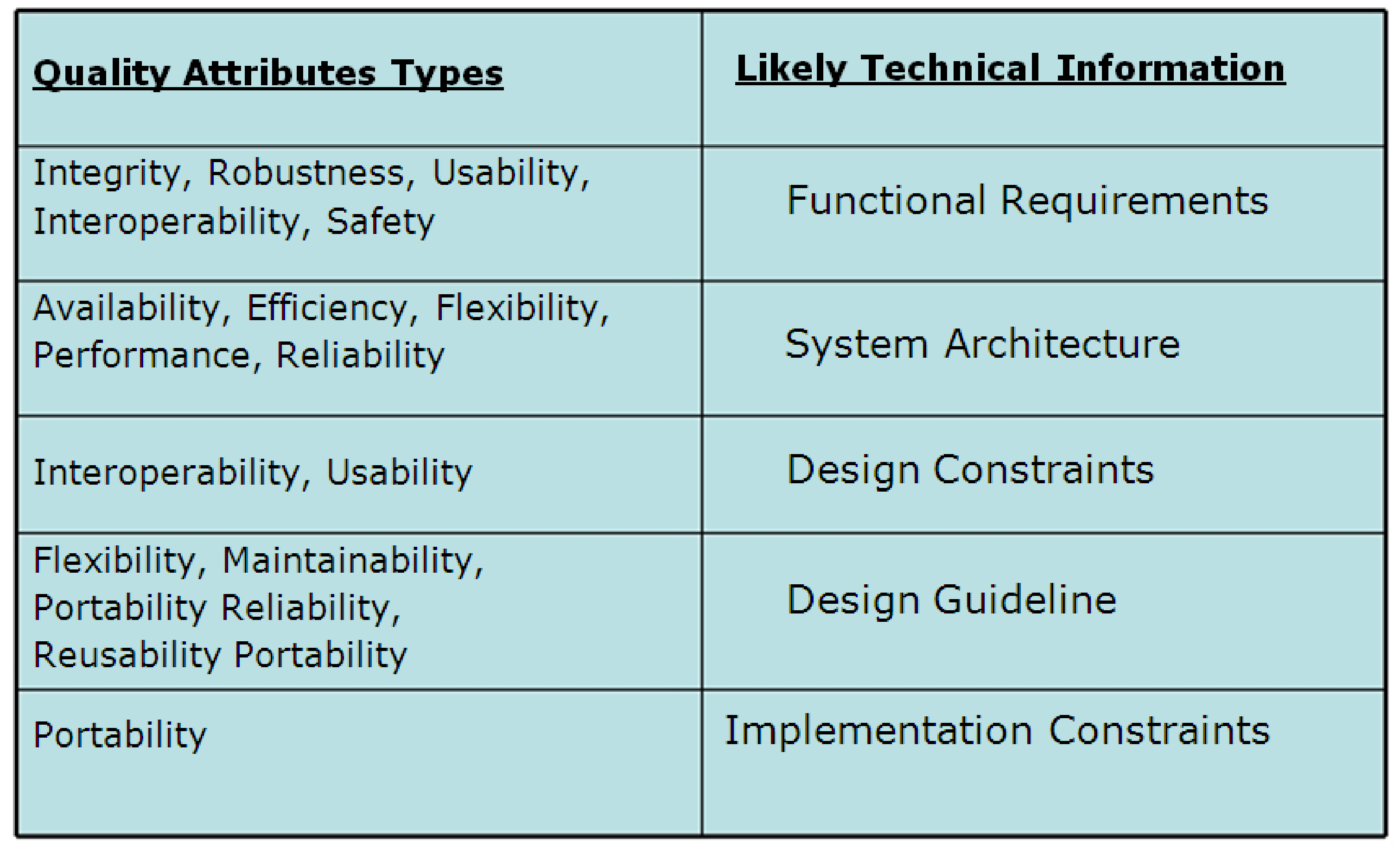
Measure Usability

- 5 areas from which to measure usability:
  - 1) Time to learn
  - 2) Speed of performance
  - 3) Error rate
  - 4) Retention over time
  - 5) Subjective satisfaction
- These criteria to some degree can be measured by observing people using software or by designing experiments

Attribute Trade-Off

- Certain attributes may have conflicting consequences, Software Engineers must conduct "attribute trade-off" to balance product characteristics.
- Software Engineers and stakeholders must decide which attributes are more important than others.
- Examples:
  - You can not expect systems to operate on multiple platforms (Portability) and also have usability.
  - Complex system (Robust) may not be fast (efficient) due to more data checking and validating.
  - It is difficult to completely test the integrity of very high security systems.

Translating To Technical Spec



Quality Attributes Types	Likely Technical Information
Integrity, Robustness, Usability, Interoperability, Safety	Functional Requirements
Availability, Efficiency, Flexibility, Performance, Reliability	System Architecture
Interoperability, Usability	Design Constraints
Flexibility, Maintainability, Portability Reliability, Reusability Portability	Design Guideline
Portability	Implementation Constraints

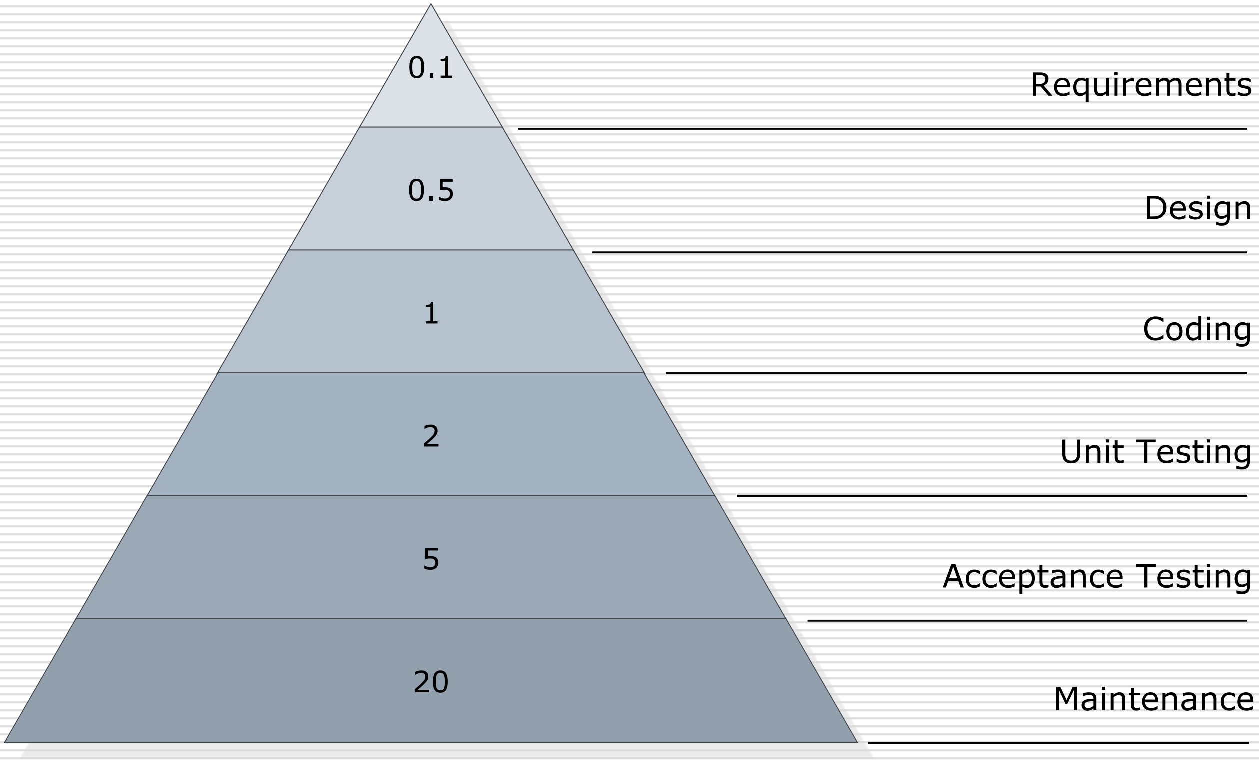
Activities

- Identify five quality attributes that you think are important to the software tool that you use today.
- Create five questions about each attribute to articulate your expectation.
- Explore the quality attribute of a computer game that you like.

The Requirements Problem

- Many systems are delivered late and over budget. They often don’t do what users wanted or are often not used to full effectiveness.
- The first step to resolving any problem is to identify the *root cause*
- The same study identified the following root causes:
  - Lack of user input
  - Incomplete requirements and specifications
  - Changing requirements and specifications

Incremental Cost of Bad/Missing Requirements



Relative Cost To Repair Defect

Effects of problems in requirements:

- Re-specification
  - Redefine requirements again
- Re-design
  - Change models and dependencies
- Re-coding
  - Re-write code
- Re-testing
  - Change test plan, test case, retest
- Lots of extra effort and lost time/resources:
  - Product Recall / Throw away product
  - Legal liability / Angry Customers
  - Increase Maintenance cost
  - Documentation changes

Good Requirements Management ?

Good Requirements Management

- Prevent:
  - Project Cost and Schedule Overruns
  - Cancellation
- Successful projects have the following "success factors":
  - User involvement
  - Executive management support
  - Clear statement of requirements

Quality of Requirements: Consistency

- Consistency
  - no contradictions
  - the program won’t work if requirements are contradictory
- Hard to guarantee
  - large sets of requirements
  - contradictions can be hidden
    - example:
      - Section 1.2 says: “no more than 128MB of memory needed”
        Section 5.8.9 says: “images should be rendered in real time”
      - => is this a contradiction???
- Formal logic: anything follows from a contradiction
- Problem:
  - often difficult to explain contradiction to customers
  - customers may want impossible things

Quality of Requirements: Manageability

- Resources must match requirements
  - can it be done in time?
  - with the money available?
  - with the skills we have?
- Risk management
  - requirements should be prioritized
  - ideally: alternatives in case it doesn't work out
  - often impossible to tell which requirements are possible
- Complexity management
  - don't do everything at once
  - iterative processes
  - prototyping

Quality of Requirements: Specificity

- Be as precise and detailed as possible
- Bad:
  - “program shall be fast”
  - “it takes a number as input”
- Good:
  - “the program shall give a response in less than 1s”
  - “it takes a 16-bit signed integer as input”
- Definitions
  - are often a good idea
  - example: “by 'Number', we always mean a 16-bit signed integer”
  - defined terms are often capitalized
- Rules about definitions
  - no circles

Quality of Requirements: No Implementation Bias

- Implementation bias:
  - giving information about the design
  - giving information about the code
- Use terminology of the domain
  - not technical terminology
  - you want to focus on WHAT
  - leave HOW for later
- Bad examples:
  - “store the checked-out books in an array”
  - “calculate the square root with Newton's algorithm”
- Good examples:
  - “the library knows which books are checked out”
  - “return the square root with 5-digit precision”

Requirements must be …

- **Correct**: They must represent the real need of customers and users.
- **Complete**: They include all the needed elements.
- Functionality, external interfaces, quality attributes and design constraints.
- **Clear**: They can be understood in the same way by all stakeholders with minimum supplementary explanation.
- **Concise**: They are stated simply, in the most minimal way possible to be understandable.
- **Consistent**: They do not conflict with other requirements.

Requirements must be …

- □ **Relevant**: They are necessary to meet a business need, goal, or objective.
- □ **Feasible**: They are possible to implement.
- □ **Verifiable**: There is a finite, cost effective technique for determining whether the requirement is satisfied.

Analysis Mechanisms

- Persistency
- Communication (IPC and RPC)
- Message routing
- Distribution
- Transaction management
- Process control and synchronization (resource contention)
- Information exchange, format conversion
- Security
- Error detection / handling / reporting
- Redundancy
- Legacy Interface

Analysis Mechanism Characteristics

- Persistency
  - Granularity
  - Volume
  - Duration
  - Access mechanism
  - Access frequency (creation/deletion, update, read)
  - Reliability
- Communication
  - Latency
  - Synchronicity
  - Message Size
  - Protocol

- Legacy interface
  - Latency
  - Duration
  - Access mechanism
  - Access frequency
- Security
  - Data granularity
  - User granularity
  - Security rules
  - Privilege types

Activities

- Identify five Analysis Mechanism that you think are important to CDIO Project.
- Create five questions about each Mechanism to articulate your expectation.