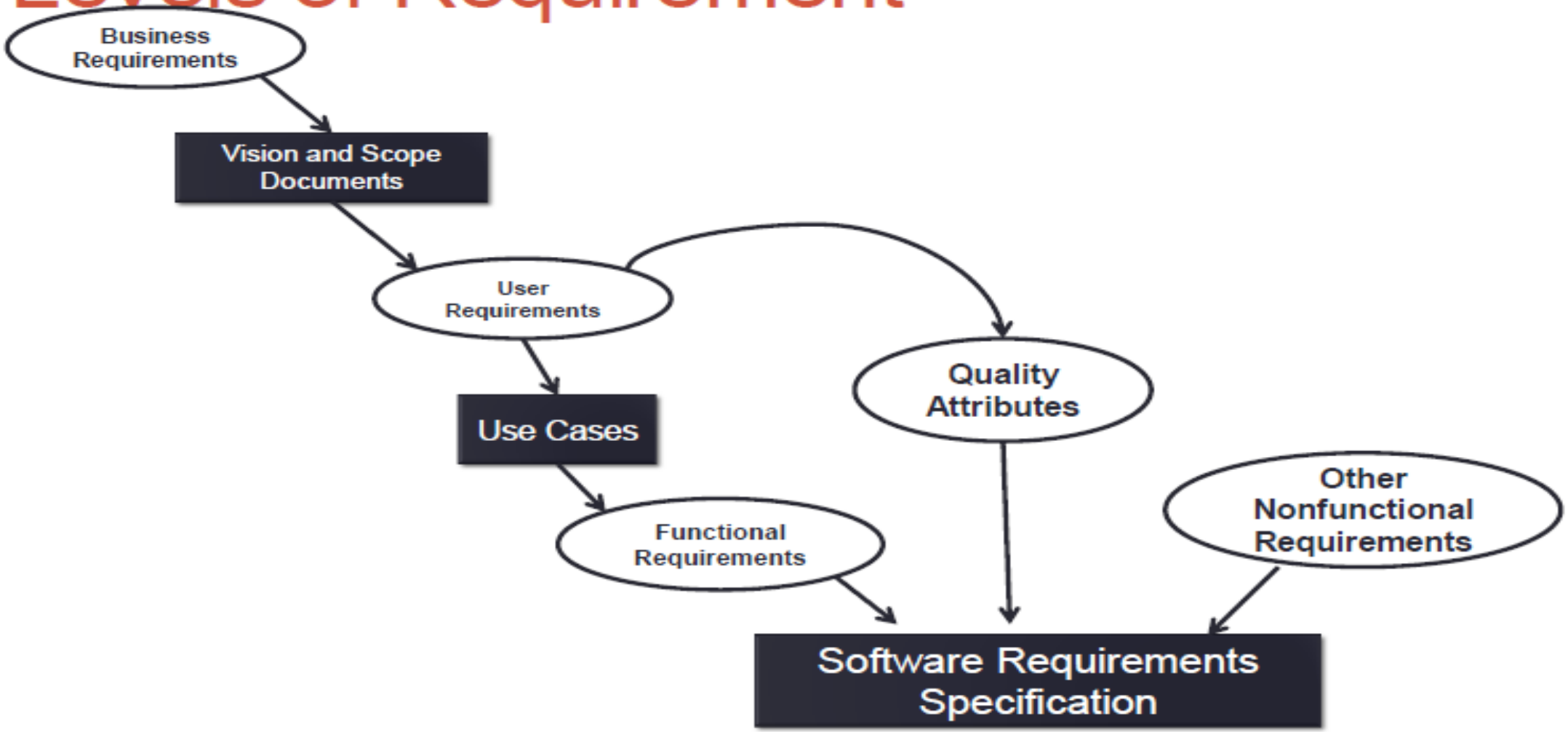
QAW

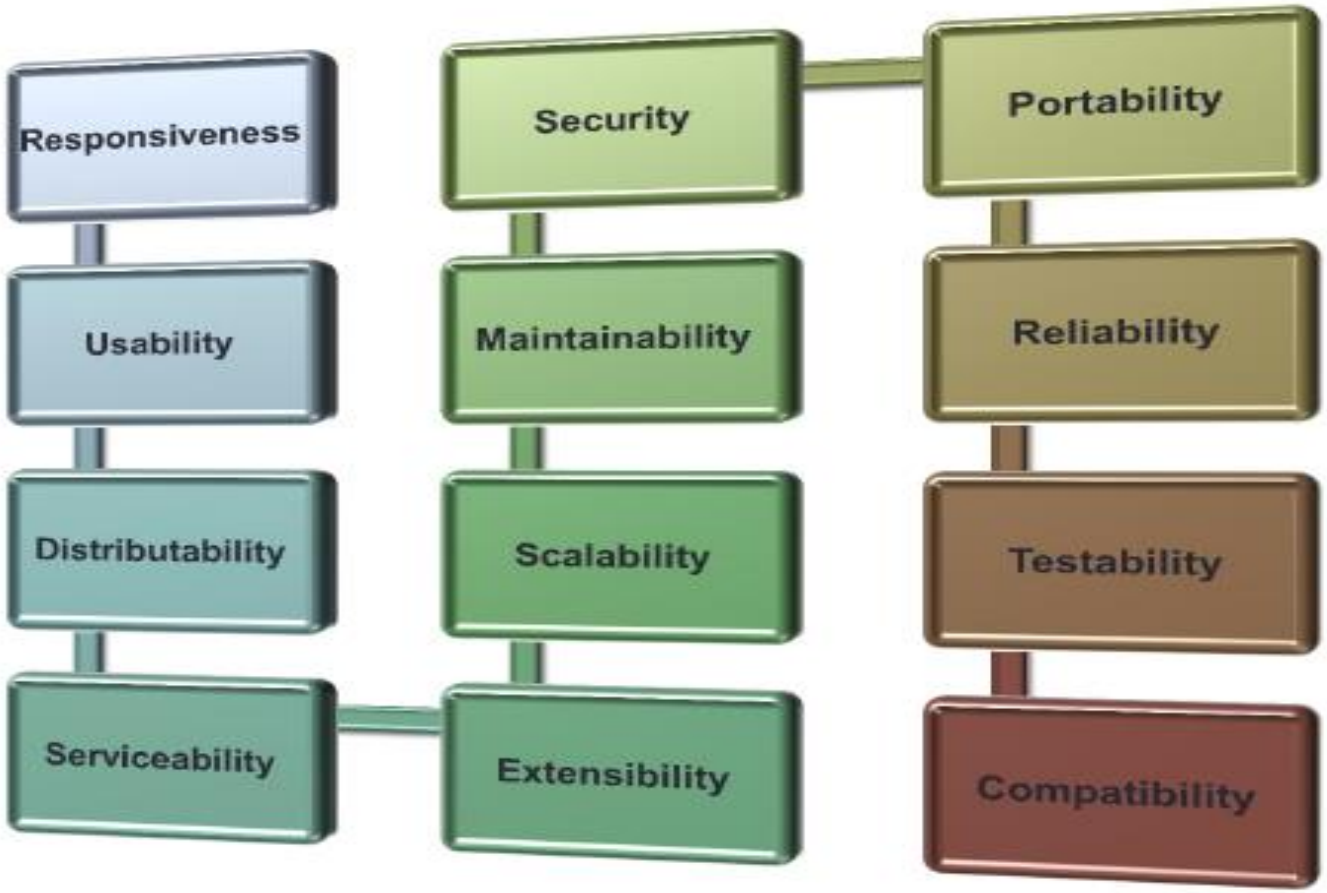
Presented by
Venugopal Shastri

Quality Attributes in Software Architecture

Levels of Requirement



Quality Attributes

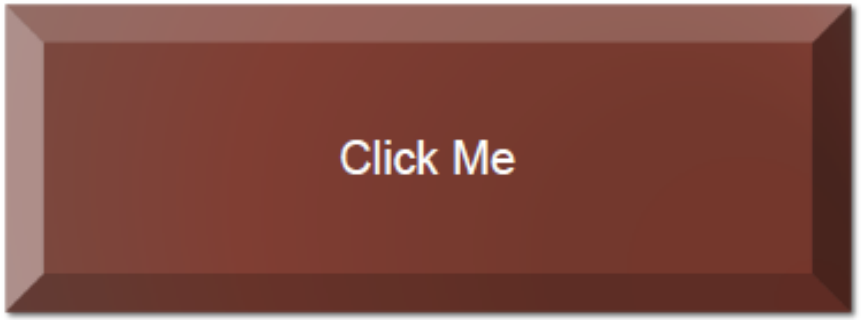


Responsiveness

- Responsiveness is defined as how quickly a system responds to user input.
  - Long delays can be a major cause of user frustration, or let the user believe the system is broken or that a command or input has been ignored.
- Responsiveness is not same as performance.

Responsiveness

- ***All Operations Must Be Responsive***



Under normal IO load, any operation must return to the user within 3 seconds.

What causes the performance issue?

- Complexity
- Indirection
- Repetitions
- Bad Design
- I/O
- … …

SAP Standard Reference

Req.	Description	Influence on
PERF-1	A test system and appropriate test cases shall be in place.	
PERF-2	Average and maximum throughput requirements shall be provided.	
PERF-3	A sizing procedure shall be defined for required CPU, memory and disk space.	CPU, Disk space Memory
PERF-4	The average end-to-end dialog response time shall be competitive (default 1000 ms), and run times for batch processing shall be appropriate.	Response time
PERF-5	The number of accesses to the persistence layer shall be recorded.	CPU, Disk I/O
PERF-6	The data volume transferred between persistence layer and application layer shall be recorded in kilo bytes.	CPU, Disk I/O
PERF-7	The physical memory consumption in MB on all application stacks shall be recorded.	Memory
PERF-8	The total CPU time of all application stacks in ms shall be recorded.	CPU
PERF-9	The avg. number of synchronous roundtrips between front end and application layer per user interaction step shall be recorded.	Response time Network Bandwidth
PERF-10	The avg. data volume in KB per interaction step transferred between the front end and the application layer step shall be recorded.	Network Bandwidth, Response time
PERF-11	For identical functions the resource consumption between two subsequent software versions shall not exceed 3%. The data basis for this non-regression requirement is the data recorded with PERF-5 to PERF-10.	Non-regression
PERF-12	The load on the persistence layer shall be kept at a minimum: Unnecessary accesses to the persistence layer shall be removed.	Response time, CPU, Disk I/O
PERF-13	The access times to the persistence shall be independent on the amount of data persisted.	Response time, CPU, memory
PERF-14	Parallel processing through efficient lock design and proper splitting and size of work packages shall be enabled.	CPU, Disk I/O Memory
PERF-15	Memory consumption shall be linear at most to the amount of processed data. There shall be no memory leaks.	Memory
PERF-16	Processing time shall be linear to the amount of processed data at most.	CPU
PERF-17	There shall be no more than 2 synchronous round trips between front end and application layer per user interaction step.	Network Bandwidth, Response time
PERF-18	There shall be no more than 2 synchronous round trips between two servers/instances per user interaction step.	Network Bandwidth, Response time

Scalability

- Scalability refers to a systems ability to handle increased adversity in its environment in a manner that is graceful and predictable.
- Adversity comes in two dimensions:
  - Increased managed content
  - Decreased system resources

Usability

- Usability is the customer's ability to get work done with the system in an efficient and pleasing manner.
- A usable system should build on skills a user already has and not require new or unique knowledge to use the system.
- Each new function the user encounters should follow a similar pattern so that once a user has learned one function others are intuitive to learn.

Maintainability

- Maintainability means the ease with which a system can be modified in order to correct defects, meet new functionality, make maintenance easier, or cope with a changing environment.
  - ***Adhere to Coding Standards***
  - ***Avoid duplication***
  - *Story Telling with your code*
  - ***Keep*** *your code simple*

Portability

- Portability is defined as the ability to reuse features and utility source code, across multiple Operating systems.
  - By having an OS abstraction layer that implements platform specific utilities, such as locks and shared memory.
  - Can also be achieved by compiling different segments of code, depending on the platform the software is to be run on.

Reliability (Availability)

- Reliability is defined as the ability of a system, to function correctly, under any reasonable circumstance.
  - Reasonable can include adverse situations, in which unexpected (but supported) usage occurs.
  - For Storage systems, this includes putting extreme I/O load on the system, but expecting the system to be manageable.

$$A = \frac{E[\text{Uptime}]}{E[\text{Uptime}] + E[\text{Downtime}]}$$

MTBF – Mean Time Between Failure
MTTR – Mean Time to Recovery
Availability= MTBF/(MTBF+MTTR)

Testability

- **Software testability** is the degree to which a software artifact (i.e. a software system, software module, requirements- or design document) supports testing in a given test context.

Testability of Software Components

- **Controllability**: The degree to which it is possible to control the state of the component under test (CUT) as required for testing.
- **Observability**: The degree to which it is possible to observe (intermediate and final) test results.
- **Isolateability**: The degree to which the component under test (CUT) can be tested in isolation.
- **Separation of concerns**: The degree to which the component under test has a single, well defined responsibility.
- **Understandability**: The degree to which the component under test is documented or self-explaining.
- **Automatability**: The degree to which it is possible to automate testing of the component under test.
- **Heterogeneity**: The degree to which the use of diverse technologies requires to use diverse test methods and tools in paralle

Testability of Requirements

- **consistent**
- **complete**
- **unambiguous**
- **quantitative**
- **verifiable in practice**

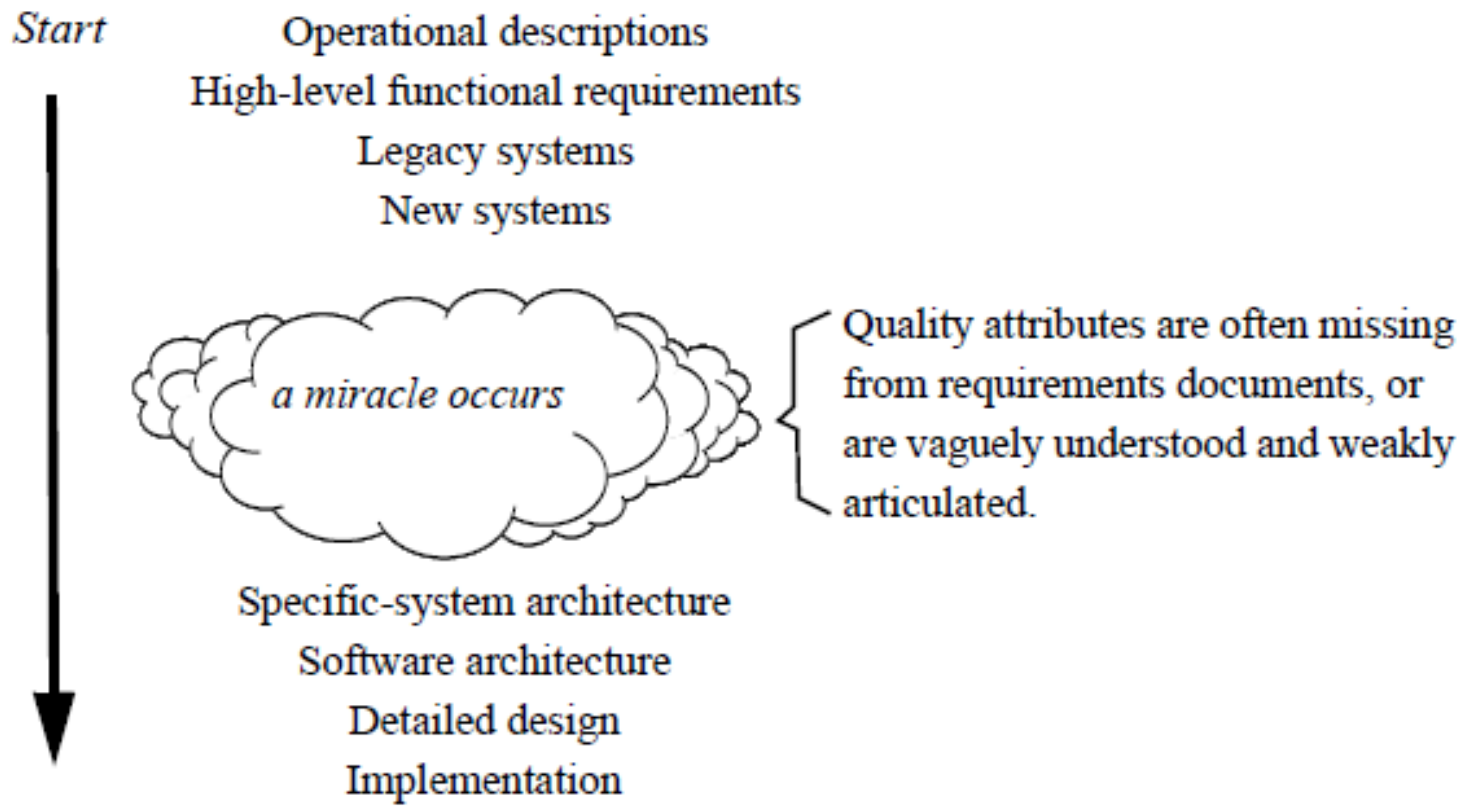
Business Qualities

- Time to Market
- Cost and Benefit
- Project Lifetime
- Target Market
- Rollout schedule
- Integration with Legacy System

Symptoms of Bad Architecture

- Rigidity
  - the system is hard to change because every change forces many other changes.
- Fragility
  - changes cause the system to break in conceptually unrelated places.
- Immobility
  - it's hard to disentangle the system into reusable components.
- Viscosity
  - doing things correctly is harder than doing things incorrectly.
- Opacity
  - the code is hard to read and understand. It does not express its intent well.

Traditional System Development Approach



QAW

- The QAW is a “lightweight” (i.e., non-intrusive) version of the Architecture Tradeoff Analysis Method(ATAM) developed by the Software Engineering Institute (SEI).
- The Architecture Tradeoff Analysis Method (ATAM) is a technique for analyzing a software architecture with respect to the quality attributes of the system
  - The ATAM can detect areas of potential risk within the architecture of a complex software–intensive system
  - It reveals how well an architecture satisfies goals and provides insight into how these quality goals interact with each other
  - It also allows engineering tradeoffs to be made among possibly conflicting quality goals.

When ATAM Applied

- before architectural decisions have been completely determined
- after architectural decisions have been determined, but before detailed design and coding activities have started or have been completed
- after system deployment, when modernization is being considered
- before system development, when multiple candidate architectures are being considered
- ATAM Evaluation can be performed quickly and inexpensively And it does not require detailed analyses of measurable quality attributes, such as mean time to failure or latency, to succeed.

QAW

- The QAW is one way to discover, document, and prioritize a system's quality attributes early in its life cycle
- Although an architecture cannot guarantee that an implementation will
- meet its quality attribute goals, the wrong architecture will surely spell disaster
- As an example,
  - consider security. It is difficult, maybe even impossible, to add effective security to a system as an afterthought. Components as well as communication mechanisms and paths must bedesigned or selected early in the life cycle to satisfy security requirements.
- QAW does not aim at an absolute measure of architectural quality. Rather, the objective is to identify

QAW – Primary Purpose

- What is the precise meaning of quality attributes—such as modifiability, security, performance, and reliability—in the context of the system being built?
- How can you discover, characterize, and prioritize the key quality attributes before the system is built?
- How can geographically dispersed communities of system stakeholders be engaged in a disciplined and repeatable way in the discovery and characterization of quality attributes?
- How can all this information be used?

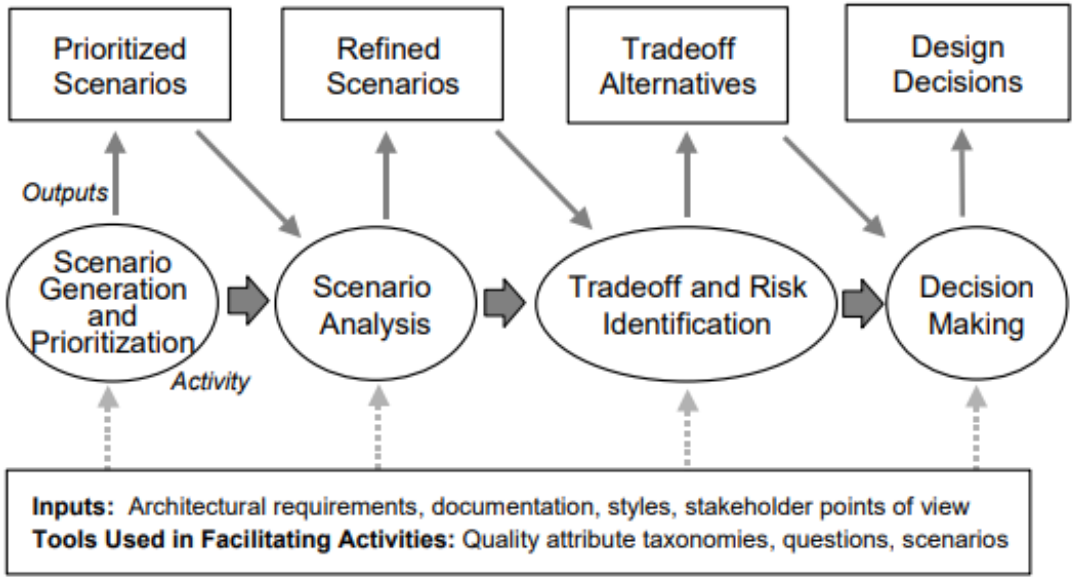
Introduction

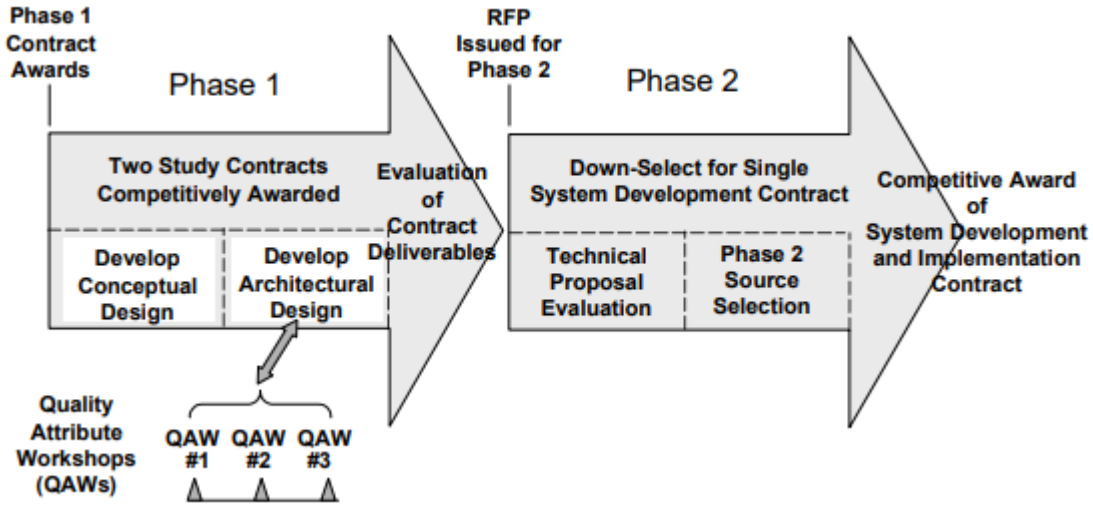
- To develop software for a system the requirements must be known.
- Several methods are available to elicit these requirements from stakeholders with varying degrees of success.
- QAW is a systematic approach to elicit the needed requirements to ensure that all needed quality attributes are included in the final design.
  - A quality attribute is a desired feature of a software system.

Goals Of QAW

- Quality Attribute Workshops are designed to gather stakeholders together, and get them to discuss the features of the system that they want.
- Scenarios are created for the attributes so that their purpose is better understood.
- This allows the developer to better understand what the attributes of the software need to be, and how they are supposed to be used.
- This also allows the stakeholders to better understand the system as a whole.
- The final goal is to produce documentation that includes as many as possible of the quality attributes specified by the stakeholders.

QAW Methodology





- A QAW has eight steps:
  - QAW Presentation and Introductions
  - Business/Mission Presentation
  - Architectural Plan Presentation
  - Identification of Architectural Drivers
  - Scenario Brainstorming
  - Scenario Consolidation
  - Scenario Prioritization
  - Scenario Refinement

Presentation and Introduction

- To begin the QAW presenters give a brief overview of the why the workshop is being done, and how it works.
- The presenters introduce themselves, and then each stakeholder does the same
  - Information given by each stakeholder should include
    - Name
    - Organization
    - Background
    - Role within the organization
    - Relationship to the system

Business Mission / Presentation

- In this step one of the stakeholders will give about an hour long presentation on the drivers of the system.
- Depending on the organization that requested the software the drivers could be business oriented or mission oriented
- During the presentation the workshop leaders will gather information from the presentation about quality attribute drivers.

- the system's business/mission context
- high-level functional requirements, constraints, and quality attribute requirements

Architectural Plan Presentation

• Since the system architecture is still in the design phase detailed technical documents are probably not available.
•A technical stakeholder must present whatever information is available.
•Information that may be useful includes:
  • Plans for how key goals are going to be met
  • Major technical requirements and constraints
  • Any existing diagrams and other descriptions of the system
•Again the workshop leaders role is to obtain more information about quality attribute drivers.

- plans and strategies for how key business/mission requirements will be satisfied
- key technical requirements and constraints—such as mandated operating systems, hardware, middleware, and standards—that will drive architectural decisions
- presentation of existing context diagrams, high-level system diagrams, and other written descriptions

Identification Of Architectural Drivers

- During this time the stakeholders should be given about a 15 minute break.
- This gives the workshop leaders time to compare notes, and consolidate the architecture drivers they have found.
- Information they list may include:
  - High-level requirements
  - business/mission concerns
  - Goals and objectives
  - Quality attributes
- After the list is compiled the stakeholders are asked to return, and the entire group goes over the list for refinement and any corrections that need to be made.

Scenario Brainstorming

- Workshop leaders have stakeholders create scenarios in a round-robin fashion.
- Each scenario must have a stimulus, environment, and response, and must deal with one of the drivers identified earlier.
- Stakeholders are given at least two chances to generate scenarios that address their individual concerns and needs.
- Workshop leaders role in this stage is to ensure that stakeholders create well formed scenarios.

Scenario Generation

- Scenario generation is a key step in the QAW method and must be carried out with care
- Facilitators should help stakeholders create well-formed scenarios
  - stakeholders to recite requirements such as “The system shall produce reports for users.”
  - *“A remote user requests a database report via the Web during peak usage and receives the report within five seconds.”*
- Facilitators need to remember that there are three general types of scenarios and to ensure that each type is covered during the QAW:
  - use case scenarios - involving anticipated uses of the system
  - growth scenarios - involving anticipated changes to the system
  - exploratory scenarios - involving unanticipated stresses to the system that can include uses and/or changes

Scenario Consolidation

- Several scenarios generated may be similar.
  - Similar scenarios are reviewed, and if possible merged.
- In order to merge two or more scenarios the stakeholders must agree that important goals are not being omitted after the merger.
- Workshop leaders should help to make compromises, and ensure that the majority of the stakeholders are accommodated

Scenario Prioritization

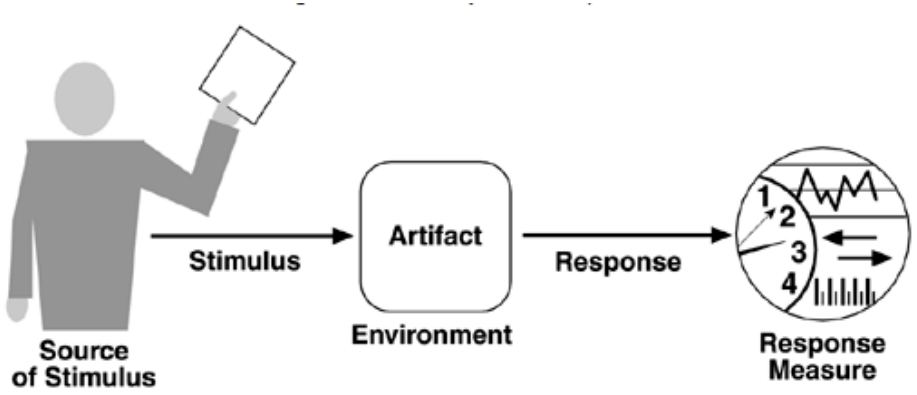
For example, if 30 scenarios were generated, each stakeholder gets 30 x 0.3, or 9, votes rounded up to 10. Voting is done in round-robin fashion, in two passes. During each pass, stakeholders allocate half of their votes. Stakeholders can allocate any number of their votes to any scenario or combination of scenarios. The votes are counted, and the scenarios are prioritized accordingly

- Each stakeholder is given votes equalling 30% of the total number of scenarios.
- Workshop leaders may round the number of votes to at their discretion if they do not come out in whole numbers.
  - Each stakeholder must have an even number of votes though.
- Voting occurs in round-robin fashion for two passes.
  - During each pass stakeholders must cast half of their votes.
  - Stakeholders may allocate that half of the votes in any way they wish.
- Once voting is completed the votes are tallied, and the scenarios are prioritized accordingly.

Scenario Refinement

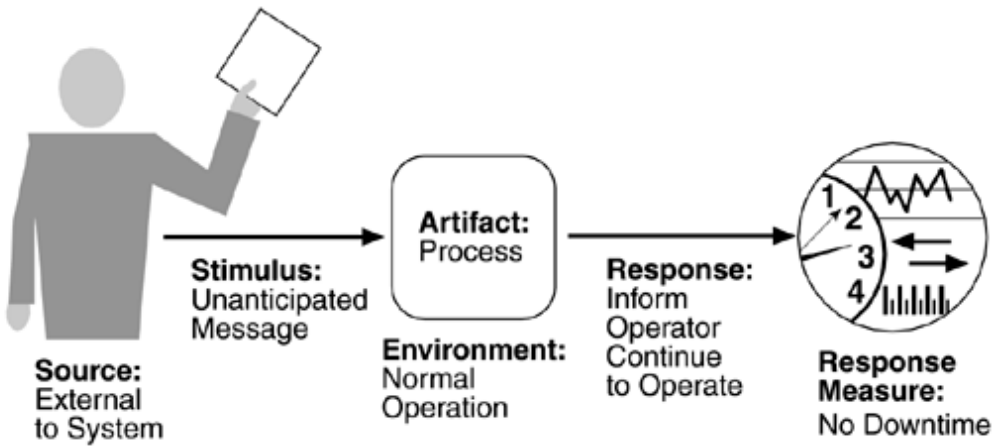
- If time remains the top four or five scenarios are refined.
- Further information on each scenario is gathered, and the workshop leaders must document the following:
  - Further clarify the scenario
    - Stimulus
    - Response
    - Source of stimulus
    - Environment
    - Artifact stimulated
    - Response measure
  - business/mission goals affected by scenario
  - Descriptions of relevant quality attributes
  - Any questions or issues stakeholders may have with the scenario.

QUALITY ATTRIBUTE SCENARIOS



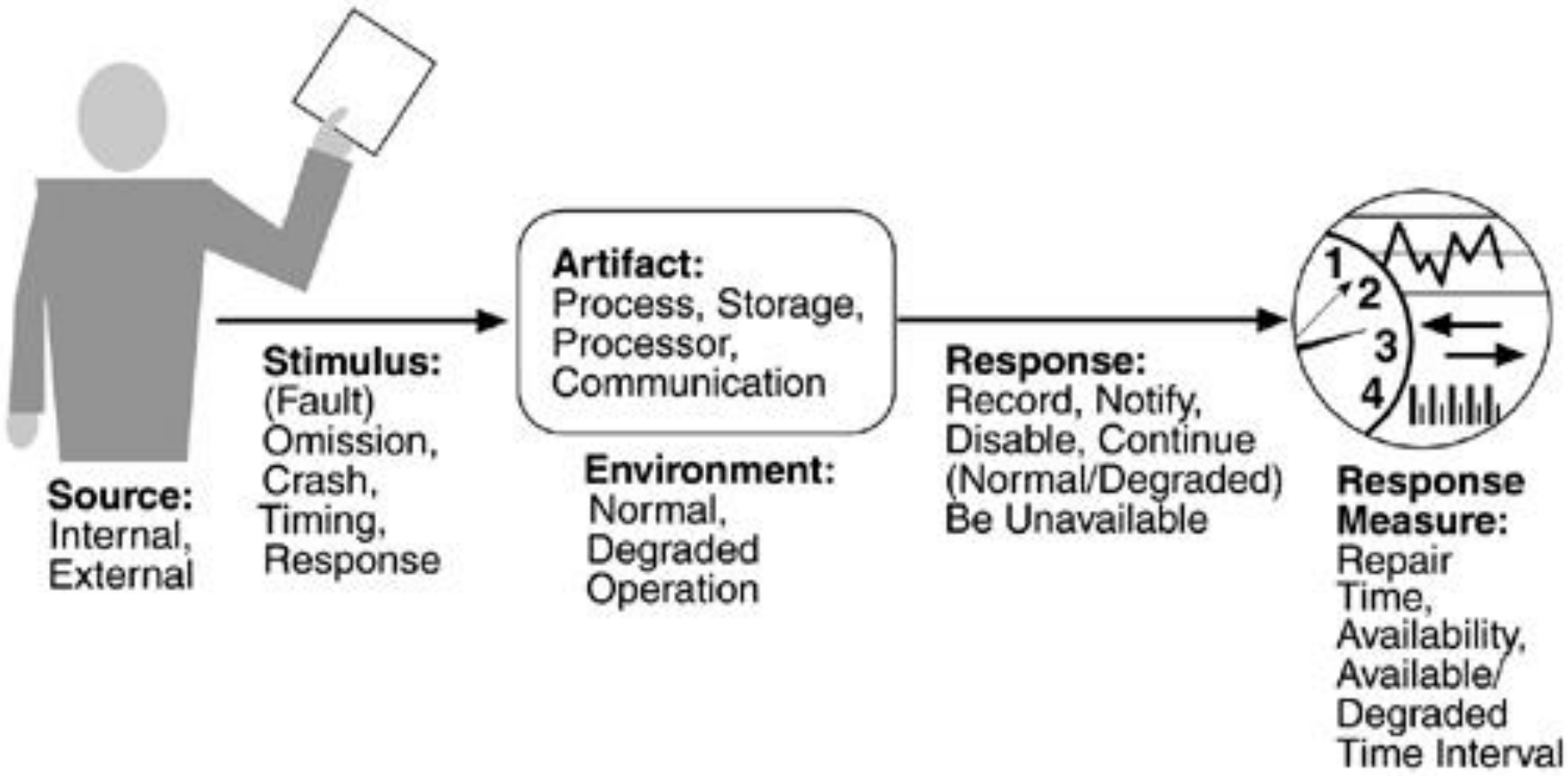
- A quality attribute scenario is a quality-attribute-specific requirement. It consists of six parts.
- **Source of stimulus**. This is some entity (a human, a computer system, or any other actuator) that generated the stimulus.
- **Stimulus**. The stimulus is a condition that needs to be considered when it arrives at a system.
- **Environment**. The stimulus occurs within certain conditions. The system may be in an overload condition or may be running when the stimulus occurs, or some other condition may be true.
- **Artifact**. Some artifact is stimulated. This may be the whole system or some pieces of it.
- **Response**. The response is the activity undertaken after the arrival of the stimulus.
- **Response measure**. When the response occurs, it should be measurable in some fashion so that the requirement can be tested.

Sample availability scenario

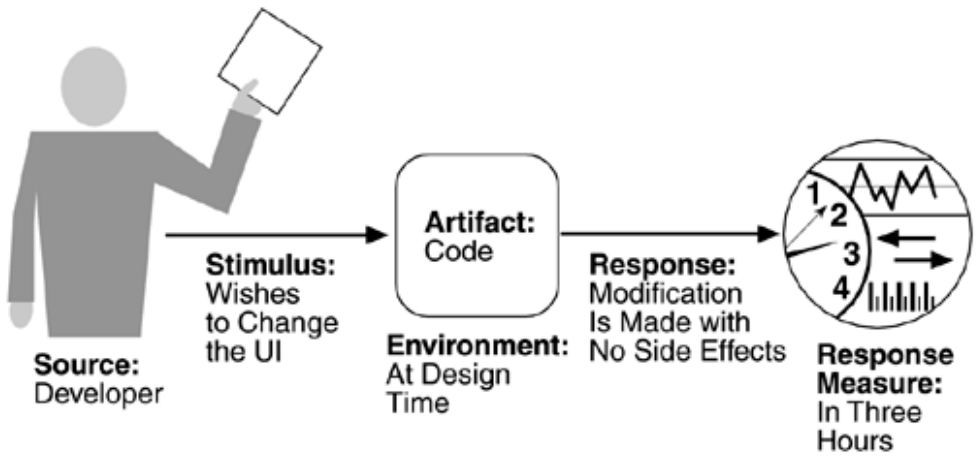


- "An unanticipated external message is received by a process during normal operation. The process informs the operator of the receipt of the message and continues to operate with no downtime."

Availability General Scenario



Modifiability Scenario



A sample modifiability scenario is "A developer wishes to change the user interface to make a screen's background color blue. This change will be made to the code at design time. It will take less than three hours to make and test the change and no side effect changes will occur in the behavior."