SS ZG653 (RL 3.1): Software Architecture

Quality classes and attribute, quality attribute scenario and architectural tactics

Instructor: Prof. SantonuSarkar

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Jan 11, 2015

A step back

- What is functionality?
  - Ability of the system to fulfill its responsibilities
- Software Quality Attributes- also called non-functional properties
  - Orthogonal to functionality
  - is a constraint that the system must satisfy while delivering its functionality
- Design Decisions
  - A constraint driven by external factors (use of a programming language, making everything service oriented)

Consider the following requirements

- User interface should be easy to use
  - Radio button or check box? Clear text? Screen layout? --- NOT architectural decisions
- User interface should allow redo/undo at any level of depth
  - Architectural decision
- The system should be modifiable with least impact
  - Modular design is must – Architectural
  - Coding technique should be simple – not architectural
- Need to process 300 requests/sec
  - Interaction among components, data sharing issues--architectural
  - Choice of algorithm to handle transactions -- non architectural

Quality Attributes and Functionality

- Any product (software products included) is sold based on its functionality – which are its features
  - Mobile phone, MS-Office software
  - Providing the desired functionality is often quite challenging
    - Time to market
    - Cost and budget
    - Rollout Schedule
- Functionality DOES NOT determine the architecture. If functionality is the only thing you need
  - It is perfectly fine to create a monolithic software blob!
  - You wouldn't require modules, threads, distributed systems, etc.

Examples of Quality Attributes

- Availability
- Performance
- Security
- Usability
- Functionality
- Modifiability
- Portability
- Reusability
- Integrability
- Testability

- The success of a product will ultimately rest on its Quality attributes
  - "Too slow!"-- performance
  - "Keeps crashing!" --- availability
  - "So many security holes!" --- security
  - "Reboot every time a feature is changed!" --- modifiability
  - "Does not work with my home theater!" --- integrability

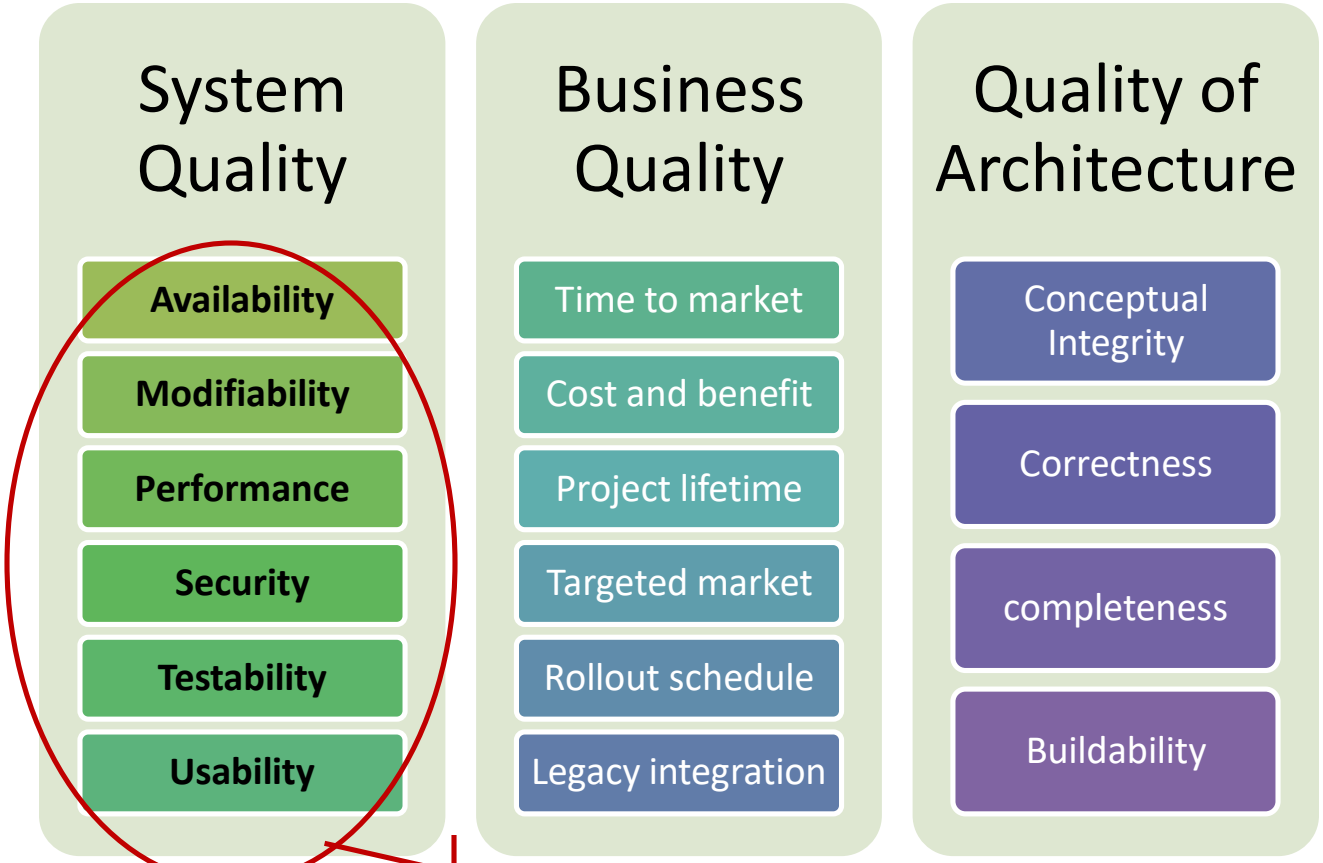
- Needs to be achieved throughout the design, implementation and deployment
- Should be designed in and also evaluated at the architectural level
- Quality attributes are NON-orthogonal
  - One can have an effect (positive or negative) on another
  - Performance is troubled by nearly all other. All other demand more code where-as performance demands the least

Defining and understanding system quality attributes

- Defining a quality attribute for a system
  - System should be modifiable --- vague, ambiguous
- How to associate a failure to a quality attribute
  - Is it an availability problem, performance problem or security or all of them?
- Everyone has his own vocabulary of quality

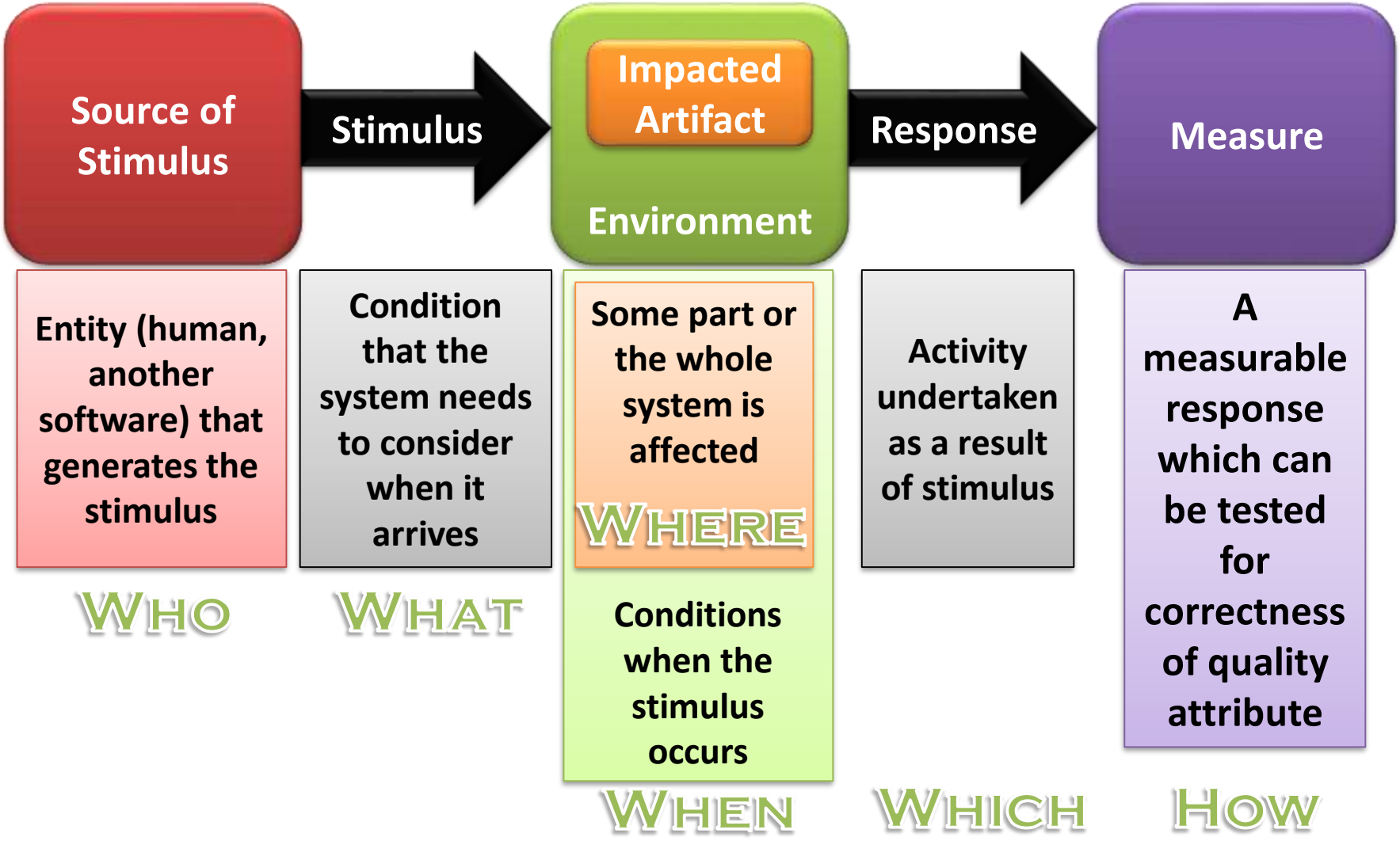
- ISO 9126 and ISO 25000 attempts to create a framework to define quality attributes

Three Quality Classes



- We will consider these attributes
- We will use "**Quality Attribute Scenarios**" to characterize them
  - which is a quality attribute specific requirement

Quality Attribute Scenario



Architectural Tactics

- To achieve a quality one needs to take a design decision- called Tactic
  - Collection of such tactics is **architectural strategy**
  - A pattern can be a collection of tactics

Quality Design Decisions

- To address a quality following 7 design decisions need to be taken
  - Allocation of responsibilities
  - Coordination
  - Data model
  - Resource Management
  - Resource Binding
  - Technology choice

Quality Design Decisions

- Responsibility Allocation
  - Identify responsibilities (features) that are necessary for this quality requirement
  - Which non-runtime (module) and runtime (components and connectors) should address the quality requirement
- Coordination
  - Mechanism (stateless, stateful…)
  - Properties of coordination (lossless, concurrent etc.)
  - Which element should and shouldn't communicate
- Data Model
  - What's the data structure, its creation, use, persistence, destruction mechanism
  - Metadata
  - Data organization
- Resource management
  - Identifying resources (CPU, I/O, memory, battery, system lock, thread pool..) and who should manage
  - Arbitration policy
  - Find impact of what happens when the threshold is exceeded
- Binding time decision
  - Use parameterized makefiles
  - Design runtime protocol negotiation during coordination
  - Runtime binding of new devices
  - Runtime download of plugins/apps
- Technology choice

Business Qualities

Business Quality	Details
Time to Market	•Competitive Pressure – short window of opportunity for the product/system •Build vs. Buy decisions •Decomposition of system – insert a subset OR deploy a subset
Cost and benefit	•Development effort is budgeted •Architecture choices lead to development effort •Use of available expertise, technology •Highly flexible architecture costs higher
Projected lifetime of the system	•The product thatneeds to survive for longer time needs to be modifiable, scalable, portable •Such systems live longer; however may not meet the time-to-market requirement
Targeted Market	•Size of potential market depends on feature set and the platform •Portability and functionality key to market share •Establish a large market; a product line approach is well suited
Rollout Schedule	•Phased rollouts; base + additional features spaced in time •Flexibility and customizability become the key
Integration with Legacy System	•Appropriate integration mechanisms •Much implications on architecture

Architectural Qualities

Architectural Quality	Details
Conceptual Integrity	•Architecture should do similar things in similar ways •Unify the design at all levels
Correctness and Completeness	•Essential to ensure system's requirements and run time constraints are met
Build ability	•Implemented by the available team in a timely manner with high quality •Open to changes or modifications as time progresses •Usually measured in cost and time •Knowledge about the problem to be solved