# Software Architecture

## Modules 2 (Pre-recorded)

### Understanding Quality Attributes

This 3 things is required while taking requirement or capture 1. Fucntionality 2. Sofware quality attributes 3. Design decisions

#### Functionality

Ability of the system to fulfill its responsibilities

#### Sofwware Quality Attributes

* It is non functionality properties
* Orthogonal to functionality ()
* is a constraint that the system must satisfy while delivering its functionality

Note: Eg: television set - its functionality is to essentially display pictures, images, videos coming from different sources that’s the functionality but the quality attributes might be something very different - it might be the quality of resolution, quality of picture, ease of how good it is to integrate with other components, how good is a sound quality. For sound recording system or a music player - then its basic functionality is to basically play some audio but it’s quality attributes are again how good the is a sound quality how good is a music quality how easy it is to use similarly for software also there is a

Note: Functionality is not about how many it can process is about processing a record and non-functional activities that it must be able to complete 5,000 record at least per second

#### Design Decisions

A constraint driven by external factors (use of a programming language, making everything service oriented)

Note: In an organization, there could be a design decision that’s already taken \* that a programming language of choice should be something \* or the platform should be something \* or the infrastructure should be from a particular vendor \* or a network topology should be like this \* or every single reusable component should be service-oriented so these are all the design decisions that are already taken you cannot compromise you cannot even question those are all something that you must adhere to

#### Quality requiements to be consider or followed

* 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.

Eg 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

To address issues, this to be addressed in initial stages

* 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
* To achieve modify ability you may have to sacrifice performance
* all other attributes actually demand more code but performance actually demands the least amount of code so that it can run faster so there is always going to be a trade-off and that will be the role of the architect

### Defining and understanding system quality attributes

* Defining a quality attribute for a system
  + System should be modifiable — vague, ambiguous

Note: statement like system should be modifiable is quite vague and ambiguous it needs to be clear because unless it is clear the architect cannot really design a system to address this and it cannot even communicate this to the stakeholders so it should be so clear that everybody should be able to understand it in an unambiguous manner so system should be modifiable is really not a right requirement if someone gives a requirement like this as an architect you should reject this requirement and you should say that the let’s have more specific more clear and unambiguous

* How to associate a failure to a quality attribute
  + Is it an availability problem, performance problem or security or all of them?

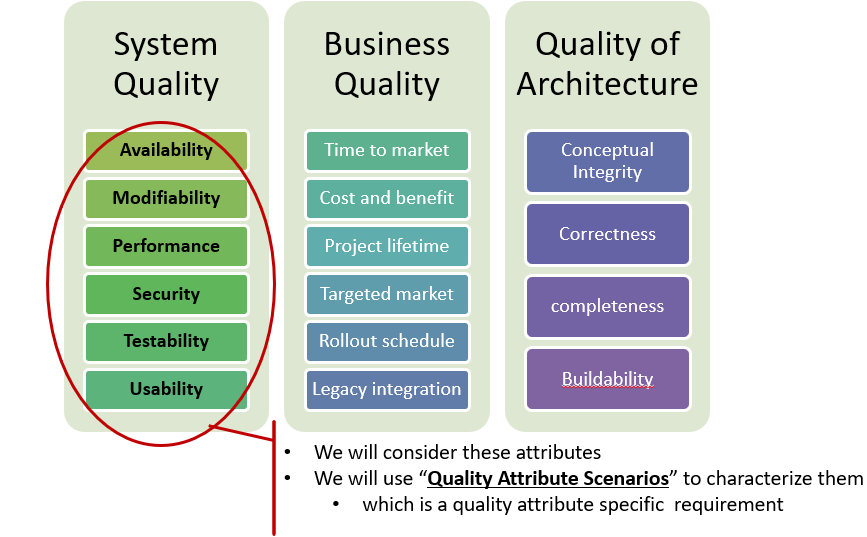
Note: when a system fail is it because of availability or is it because of performance or is it because security or is it because of all

* Everyone has his own vocabulary of quality
* ISO 9126 and ISO 25000 attempts to create a framework to define quality attributes

Note: we understand unambiguously what an availability means what a performance means what a security means in this course

### Three Quality Classes

Image



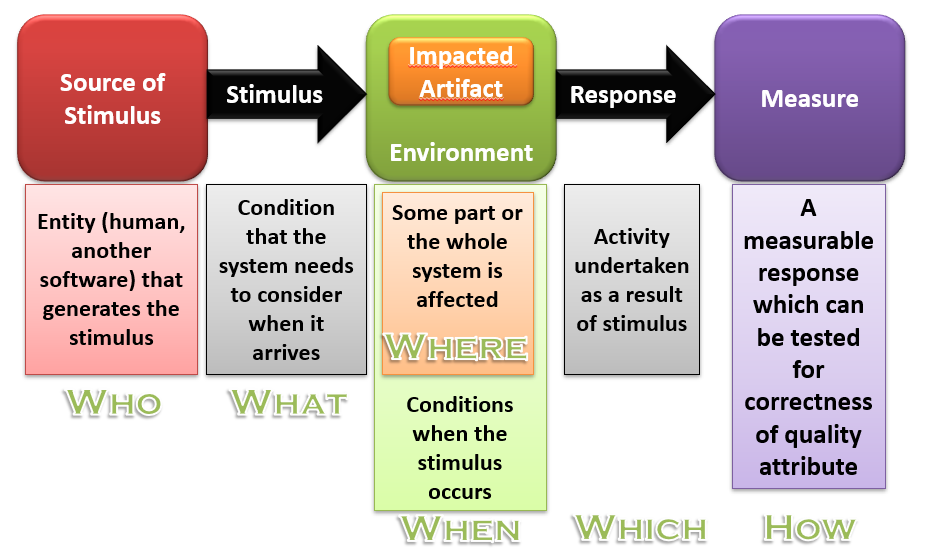
Note:

* System quality - quality of the system that you are going to build like modifiability, availability, performance, security, testability, usability.
* Business quality is time to market, cost and benefit, project lifetime, schedules etc some of them do have an impact

In this course will concentrate on system quality only

### Quality Attribute scenario

Image



* Who - is basically create scenario
* What - are the condition that system needs to be consider
* Where - is part some part of whole system is affected
* When - stimulus actually occurs, i.e., what was the state of the system when stimulus occurs
* Which - what activity should be taken as result of sitmulus
* How -measurable response which can be tested for correctness of the quality attribute

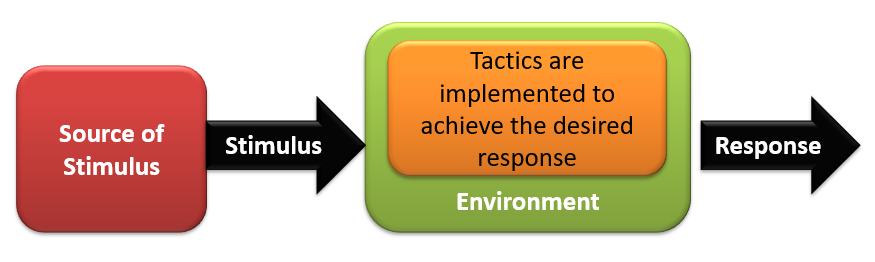
Note:

DOS attack can happen when the system is in running condition. It can impact maybe module X Y & Z of the entire system. The response is that the X Y & Z should be able to know counter and the measure could be that response time of the overall system should not be impacted and even if it is impacted should not fall below certain threshold cyber security can be key together as a requirement and then as an architect you can capture the requirement in this format and then you can use architectural tactics to address

### Architectural Tactics

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

Image



Note:

A tactic is a design decision that influences the achievement of a quality attribute response—tactics directly affect the system’s response to some stimulus

### 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
* 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
* 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

* 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 that needs 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

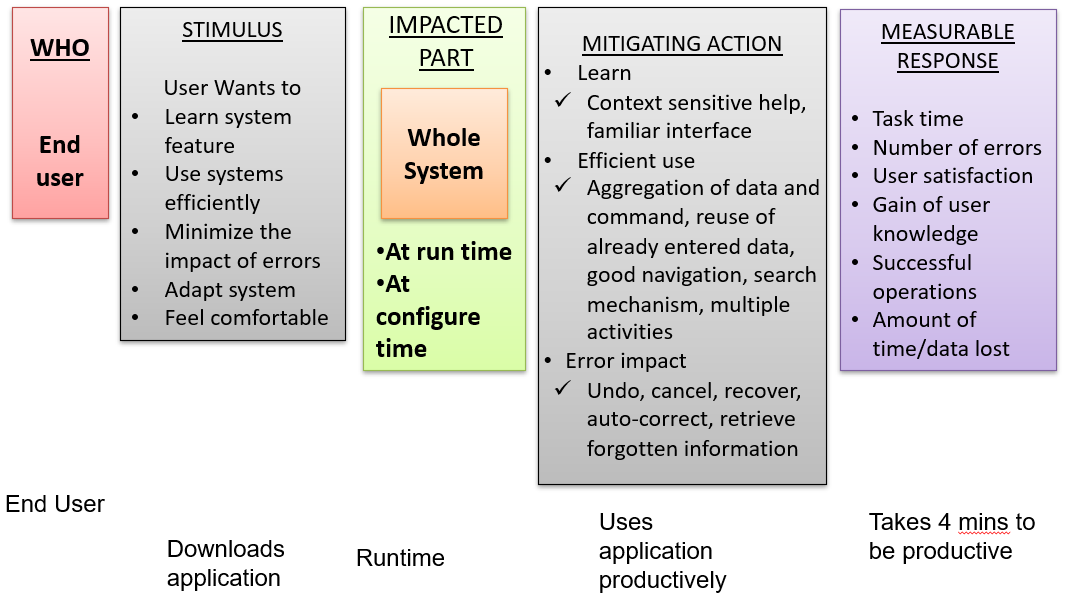
* Conceptual Integrity
  + Architecture should do similar things in similar ways
  + Unify the design at all levels
* Correctness and Completeness
  + Essential to ensure system’s requirements and run time constraints are met
* Build ability
  + Implemented by the available team in a timely manner with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

### Usability

* How easy it is for the user to accomplish a desired task and user support the system provides
* Learnability: what does the system do to make a user familiar
  + Operability:
    - Minimizing the impact of user errors
    - Adopting to user needs
    - Giving confidence to the user that the correct action is being taken?

Note: Eg: iphone mobile - it is very user friendly and easy to learn with minimum error

Image



### Usability Tactics

image

#### User Initiative and System Response

* Cancel
  + When the user issues cancel, the system must listen to it (in a separate thread)
  + Cancel action must clean the memory, release other resources and send cancel command to the collaborating components
* Undo
  + System needs to maintain a history of earlier states which can be restored
  + This information can be stored as snapshots
* Pause/resume
  + Should implement the mechanism to temporarily stop a running activity, take its snapshot and then release the resource for other’s use
* Aggregate (change font of the entire paragraph)
  + For an operation to be applied to a large number of objects
  + Provide facility to group these objects and apply the operation to the group

Note: Aggregate - Grouping feature after you draw box, lines, etc Cancel, Undo, Pause/Resume - Deploying any packages, in case if you want to revert then Undo need to maintain history for it

#### System Initiated and Its category

* Task model
  + Determine the current runtime context, guess what user is attempting, and then help
  + Correct spelling during typing but not during password entry
* System model
  + Maintains its own model and provide feedback of some internal activities
  + Time needed to complete the current activity
* User model
  + Captures user’s knowledge of the system, behavioral pattern and provide help
  + Adjust scrolling speed, user specific customization, locale specific adjustment

Note: Task model - while writing document, correct spelling System model - While download file gives percentage of completion of download User model - understand users behaviour and provide help

#### Usability Tactics and Patterns

* Design time tactics- UI is often revised during testing. It is best to separate UI from the rest of the application
  + Model view controller architecture pattern
  + Presentation abstraction control
  + Command Pattern
  + Arch/Slinky - Similar to Model view controller

#### Design Checklist

* Allocation of Responsibilities
  + Identify the modules/components responsible for
    - Providing assistance, on-line help
    - Adapt and configure based on user choice
    - Recover from user error
* Coordination Model
  + Check if the system needs to respond to
    - User actions (mouse movement) and give feedback
    - Can long running events be canceled?
* Data model
  + data structures needed for undo, cancel
  + Design of transaction granularity to support undo and cancel
* Resource mgmt can configure system’s use of resource
* Technology selection
  + To achieve usability

### Failure classification

* Transient - only occurs with certain inputs
* Permanent - occurs on all inputs
* Recoverable - system can recover without operator help
* Unrecoverable - operator has to help
* Non-corrupting - failure does not corrupt system state or data
* Corrupting - system state or data are altered

### Availability

Readiness of the software to carry out its task 100% available (which is actually impossible) means it is always ready to perform the intended task A related concept is Reliability Ability to “continuously provide” correct service without failure Availability vs Reliability A software is said to be available even when it fails but recovers immediately Such a software will NOT be called Reliable

Thus, Availability measures the fraction of time system is really available for use Takes repair and restart times into account Relevant for non-stop continuously running systems (e.g. traffic signal)

Note:

* Need to understand fault and failure because availability is about eliminating fault handling fault
* Eg: In case, word document software- if we do something or after some writing text if it gets crashes then it is availability but not reliability
* Eg: In case of health care system - if it fails after some time or with some operation then it is availability but not highly reliabile
* There is no such software is where it is reliable all the time