BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Software Architectures

SECLZG651/SSCLZG653

Parthasarathy

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

SECLZG651/SSCLZG653 – CS#2
Getting Started with Quality Attributes

Agenda for CS #2

1) Introduction to Software Architecture
    - What is Software Architecture?
    - Definitions of Software architecture
    - Architecture structure and patterns
    - Good architecture
2) Quick check, Q&A
3) Importance of software architecture
4) Contexts of Software architecture
5) Getting started with Quality attributes
6) Usability and its tactics
7) Q&A!

Structures and Views

- A *view* is a representation of a coherent set of architectural elements, as written by and read by system stakeholders.
  - A view consists of a representation of a set of elements and the relations among them.
- A *structure* is the set of elements itself, as they exist in software or hardware.
- In short, a view is a representation of a structure.
  - For example, a module *structure* is the set of the system's modules and their organization.
  - A module *view* is the representation of that structure, documented according to a template in a chosen notation, and used by some system stakeholders.
- Architects design structures. They document views of those structures.

Structures Provide Insight

- Structures play such an important role in our perspective on software architecture because of the analytical and engineering power they hold.
- Each structure provides a perspective for reasoning about some of the relevant quality attributes.
- For example:
  - The module structure, which embodies what modules use what other modules, is strongly tied to the ease with which a system can be extended or contracted.
  - The concurrency structure, which embodies parallelism within the system, is strongly tied to the ease with which a system can be made free of deadlock and performance bottlenecks.
  - The deployment structure is strongly tied to the achievement of performance, availability, and security goals.
  - And so forth.

Architectural Patterns

- Architectural elements can be composed in ways that solve particular problems.
  - The compositions have been found useful over time, and over many different domains
  - They have been documented and disseminated.
  - These compositions of architectural elements, called architectural patterns.
  - Patterns provide packaged strategies for solving some of the problems facing a system.
- An architectural pattern delineates the element types and their forms of interaction used in solving the problem.
- A common module type pattern is the Layered pattern.
  - When the uses relation among software elements is strictly unidirectional, a system of layers emerges.
  - A layer is a coherent set of related functionality.
  - Many variations of this pattern, lessening the structural restriction, occur in practice.

Architectural Patterns

- Common component-and-connector type patterns:
- Shared-data (or repository) pattern.
  - This pattern comprises components and connectors that create, store, and access persistent data.
  - The repository usually takes the form of a (commercial)
  - database.
  - The connectors are protocols for managing the data, such as SQL.
- Client-server pattern.
  - The components are the clients and the servers.
  - The connectors are protocols and messages they share among each other to carry out the system's work.

Architectural Patterns

- Common allocation patterns:
- Multi-tier pattern
  - Describes how to distribute and allocate the components of a system in distinct subsets of hardware and software, connected by some communication medium.
  - This pattern specializes the generic deployment (software-to-hardware allocation) structure.
- Competence center pattern and platform pattern
  - These patterns specialize a software system's work assignment structure.
  - In competence center, work is allocated to sites depending on the technical or domain expertise located at a site.
  - In platform, one site is tasked with developing reusable core assets of a software product line, and other sites develop applications that use the core assets.

What Makes a “Good” Architecture?

- There is no such thing as an inherently good or bad architecture.
- Architectures are either more or less fit for some purpose
- Architectures can be evaluated but only in the context of specific stated goals.
- There are, however, good rules of thumb.

Process "Rules of Thumb"

- The architecture should be the product of a single architect or a small
- group of architects with an identified technical leader.
  - This approach gives the architecture its conceptual integrity and technical consistency.
  - This recommendation holds for Agile and open source projects as well as "traditional" ones.
  - There should be a strong connection between the architect(s) and the development team.
- The architect (or architecture team) should base the architecture on a prioritized list of well-specified quality attribute requirements.
- The architecture should be documented using views. The views should address the concerns of the most important stakeholders in support of the project timeline.
- The architecture should be evaluated for its ability to deliver the system's important quality attributes.
  - This should occur early in the life cycle and repeated as appropriate.
- The architecture should lend itself to incremental implementation,
  - Create a "skeletal" system in which the communication paths are exercised but which at first has minimal functionality.

Structural "Rules of Thumb"

- The architecture should feature well-defined modules whose functional responsibilities are assigned on the principles of information hiding and separation of concerns.
  - The information-hiding modules should encapsulate things likely to change
  - Each module should have a well-defined interface that encapsulates or "hides" the changeable aspects from other software
- Unless your requirements are unprecedented your quality attributes should be achieved using well-known architectural patterns and tactics specific to each attribute.
- The architecture should never depend on a particular version of a commercial product or tool. If it must, it should be structured so that changing to a different version is straightforward and inexpensive.
- Modules that produce data should be separate from modules that consume data.
  - This tends to increase modifiability
  - Changes are frequently confined to either the production or the consumption side of data.

Why is Software Architecture Important?

1. An architecture will inhibit or enable a system's driving quality attributes.
2. The decisions made in an architecture allow you to reason about and manage change as the system evolves.
3. The analysis of an architecture enables early prediction of a system's qualities.
4. A documented architecture enhances communication among stakeholders.
5. The architecture is a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
6. An architecture defines a set of constraints on subsequent implementation.
7. The architecture dictates the structure of an organization, or vice versa.
8. An architecture can provide the basis for evolutionary prototyping.
9. An architecture is the key artifact that allows the architect and project manager to reason about cost and schedule.
10. An architecture can be created as a transferable, reusable model that form the heart of a product line.
11. Architecture-based development focuses attention on the assembly of components, rather than simply on their creation.
12. By restricting design alternatives, architecture channels the creativity of developers, reducing design and system complexity.
13. An architecture can be the foundation for training a new team member.

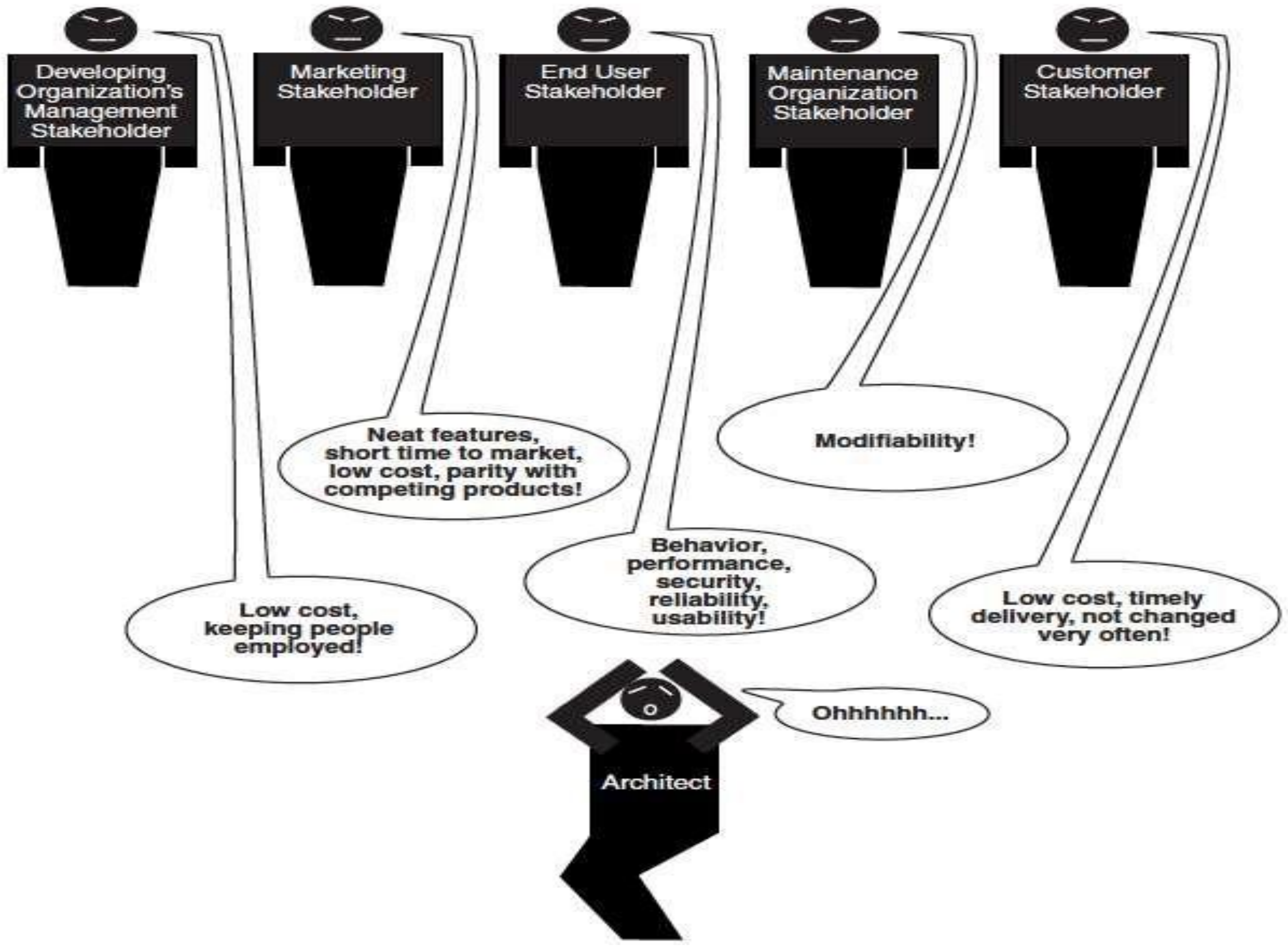
The Many Contexts of Software Architecture

- Architecture in a Technical Context
- Architecture in a Project Life-Cycle Context
- Architecture in a Business Context
- Architecture in a Professional Context
- Stakeholders
- How Is Architecture Influenced?
- What Do Architectures Influence?

Contexts of Software Architecture

- Sometimes we consider software architecture the center of the universe!
- Here, though, we put it in its place relative to four contexts:
  - **Technical**:What technical role does the software architecture play in the system or systems of which it's a part?
  - **Project life cycle**: How does a software architecture relate to the other phases of a software development life cycle?
  - **Business**:How does the presence of a software architecture affect an organization's business environment?
  - **Professional.**:What is the role of a software architect in an organization or a development project?

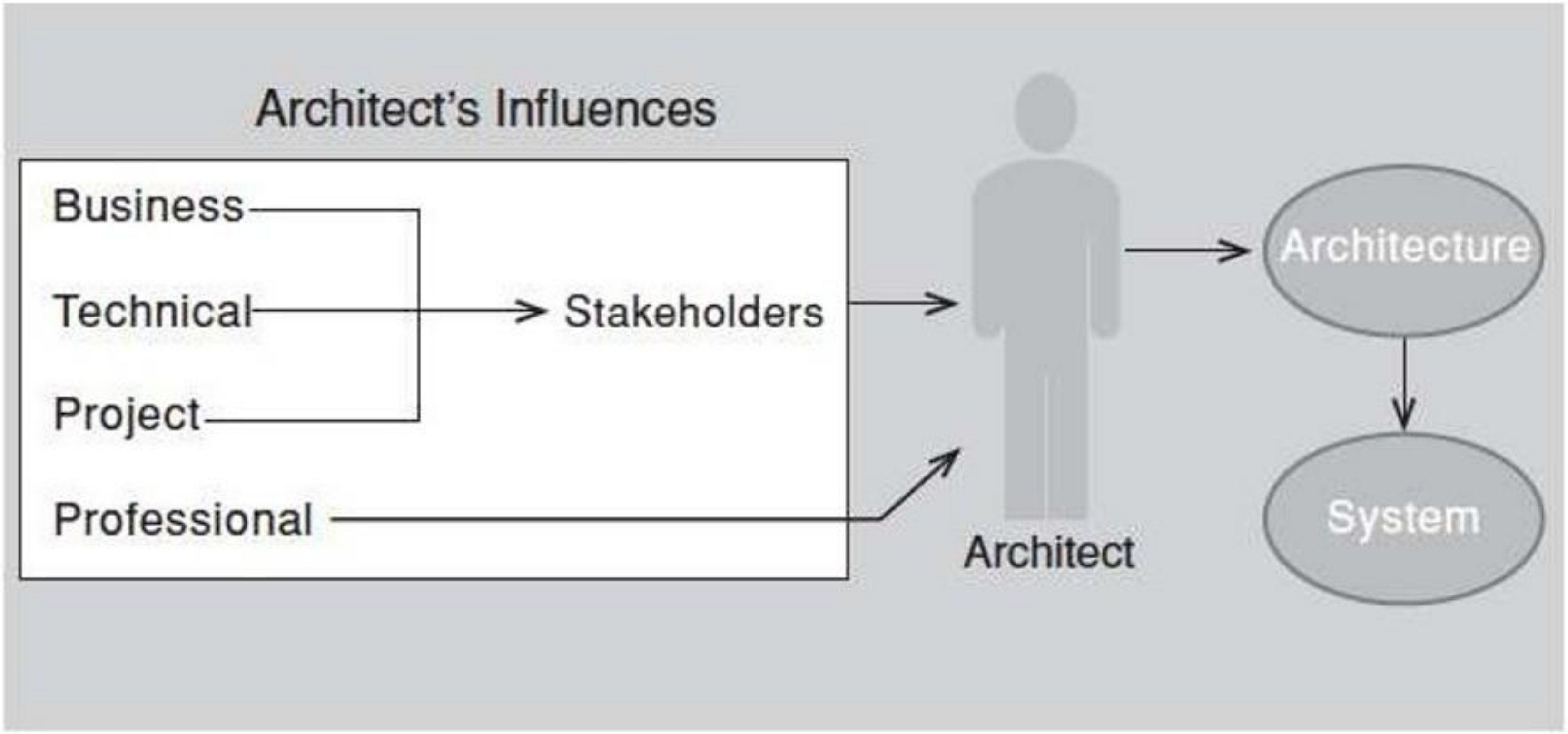
Professional Context


Developing Organization's Management Stakeholder
Marketing Stakeholder
End User Stakeholder
Maintenance Organization Stakeholder
Customer Stakeholder
Neat features, short time to market, low cost, parity with competing products!
Modifiability!
Behavior, performance, security, reliability, usability!
Low cost, keeping people employed!
Low cost, timely delivery, not changed very often!
Ohhhhhh...
Architect

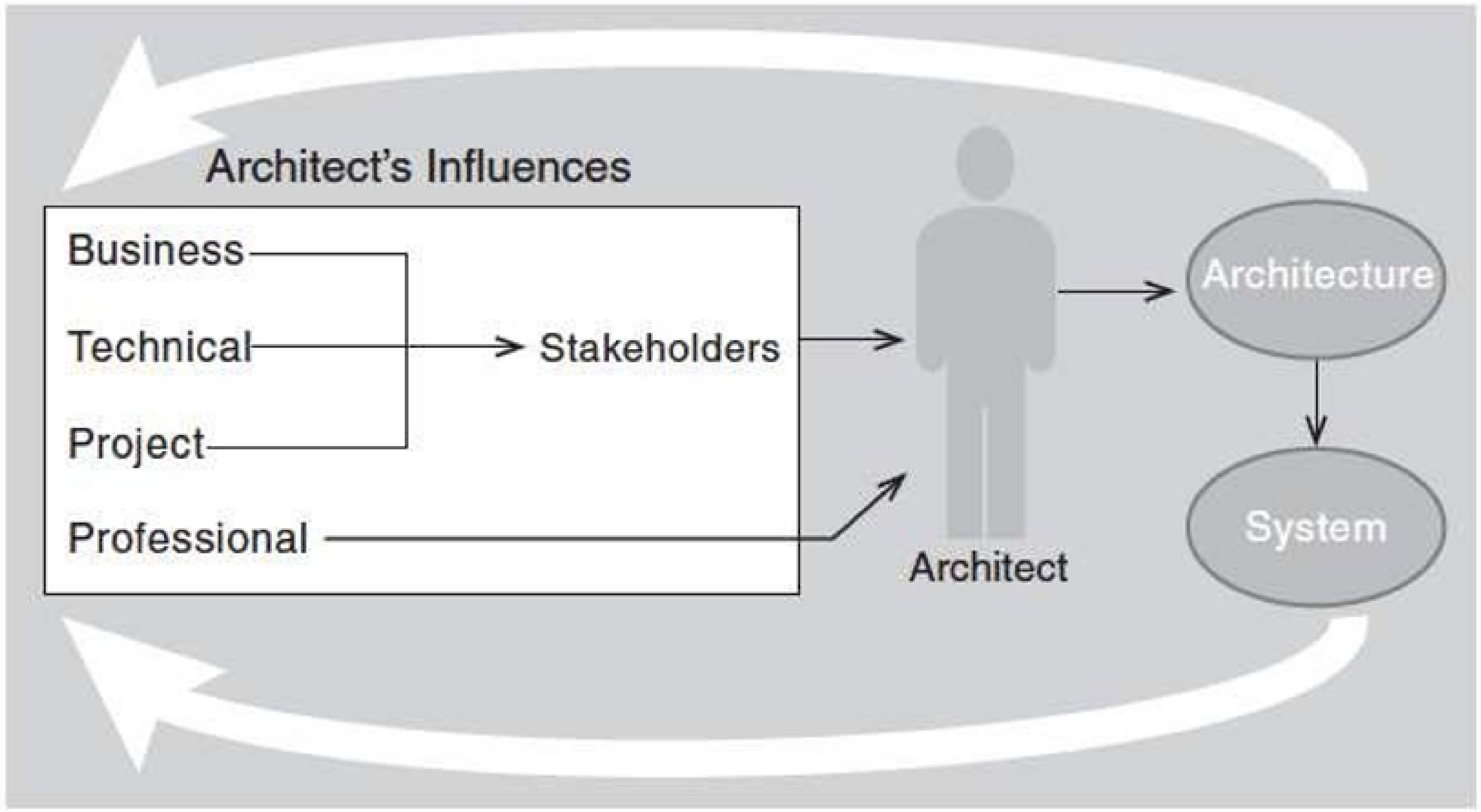
Professional Context

- You will perform many *duties* beyond directly producing an architecture.
  - You will need to be involved in supporting management and dealing with customers.
- Architects need more than just technical *skills*.
  - Architects need to explain to one stakeholder or another the chosen priorities of different properties, and why particular stakeholders are not having all of their expectations fulfilled.
  - Architects need diplomatic, negotiation, and communication skills.
  - Architects need the ability to communicate ideas clearly
  - You will need to manage a diverse workload and be able to switch contexts frequently.
  - You will need to be a leader in the eyes of developers and management.
- Architects need up-to-date *knowledge*.
  - You will need to know about (for example) patterns, or database
  - platforms, or web services standards.
  - You will need to know business considerations.

How is Architecture Influenced?



Architecture Influence Cycle



BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Quality classes and attribute, quality attribute scenario and architectural tactics

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

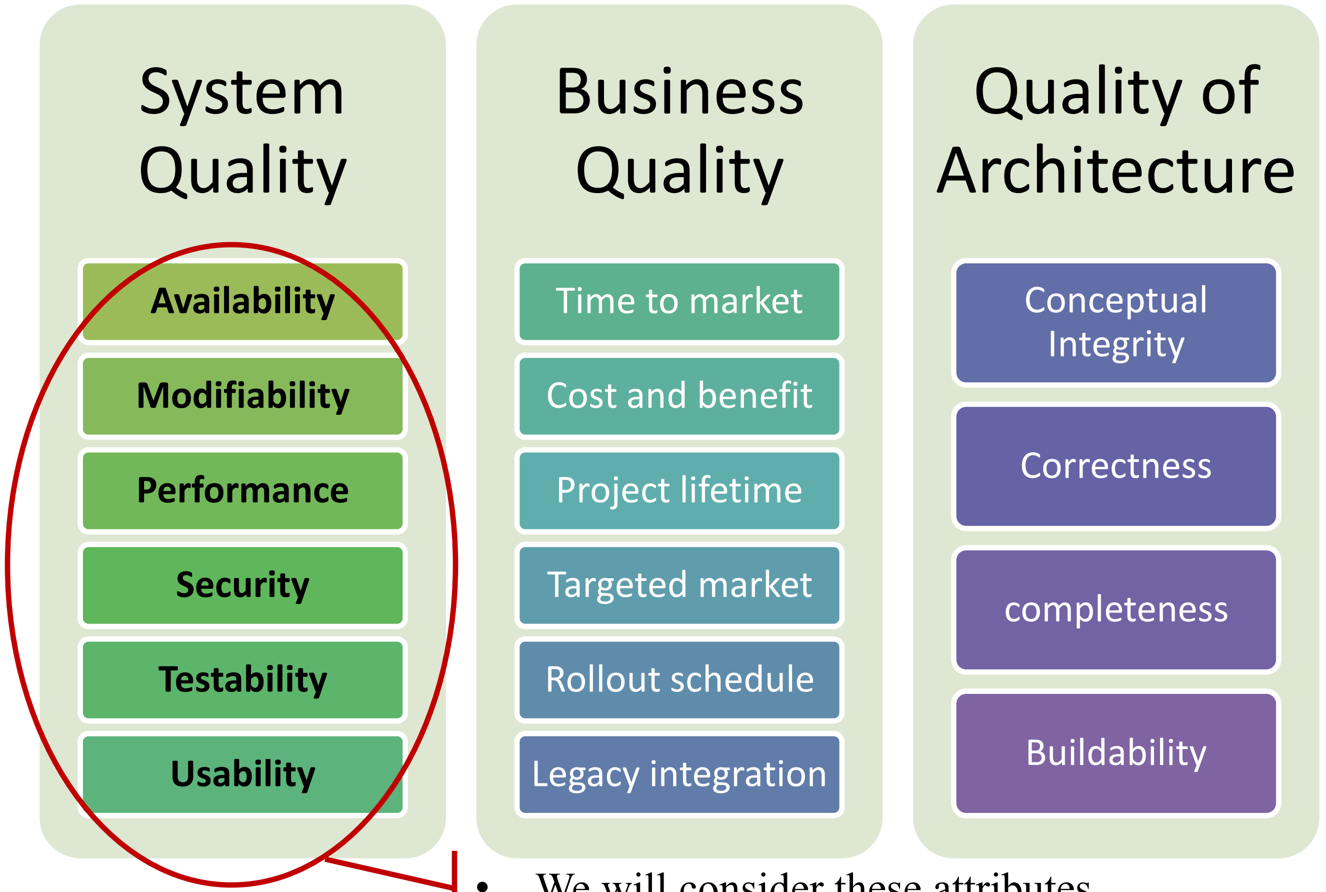
- The success of a product will ultimately rest on its Quality attributes
  - "Too slow!"-- performance
  - "Keeps crashing!" --- availability
  - "So many security holes!" --- security
  - "Reboot every time a feature is changed!" --- modifiability
  - "Does not work with my home theater!" --- integrability

- 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

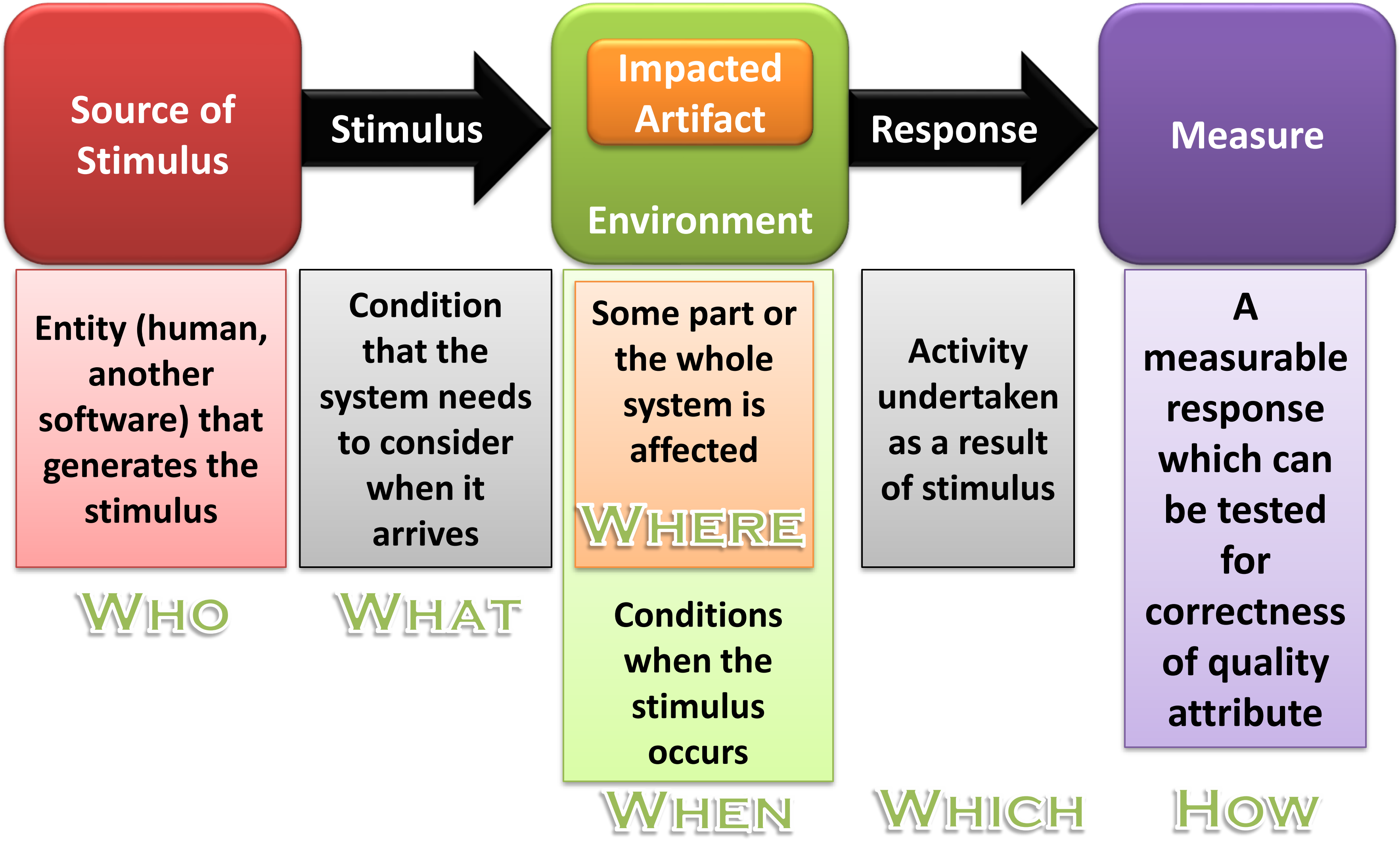
- Defining a quality attribute for a system
  - System should be modifiable --- vague, ambiguous
- How to associate a failure to a quality attribute
  - Is it an availability problem, performance problem or security or all of them?
- Everyone has his own vocabulary of quality
- ISO 9126 and ISO 25000 attempts to create a framework to define quality attributes

Three Quality Classes



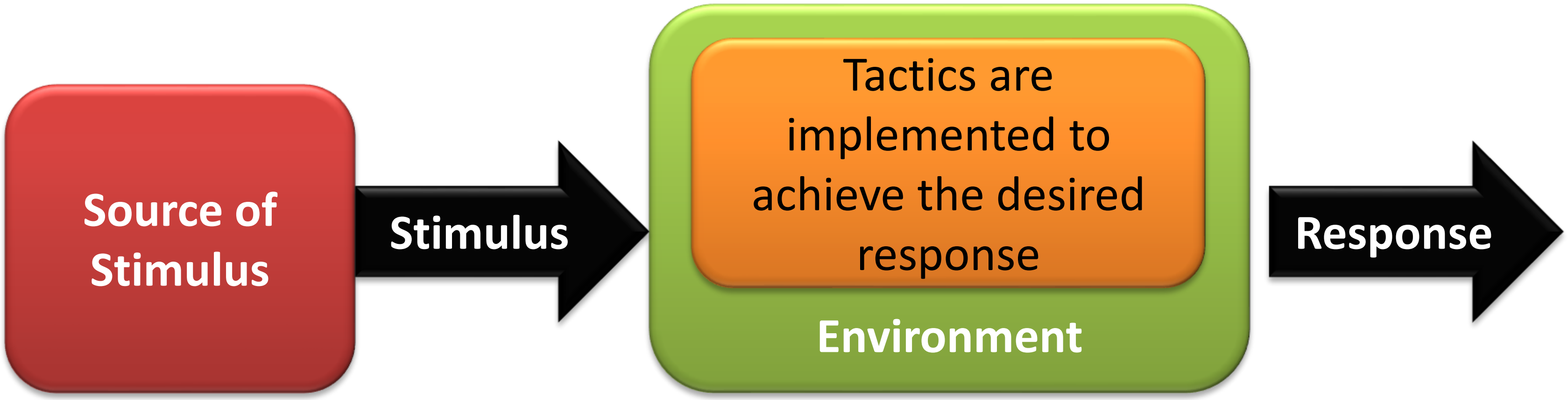
- We will consider these attributes
- We will use "**<u>Quality Attribute Scenarios</u>**" to characterize them
  - which is a quality attribute specific requirement

Quality Attribute Scenario


Source of Stimulus
Stimulus
Impacted Artifact
Environment
Response
Measure
Entity (human, another software) that generates the stimulus
Condition that the system needs to consider when it arrives
Some part or the whole system is affected
WHERE
Conditions when the stimulus occurs
Activity undertaken as a result of stimulus
A measurable response which can be tested for correctness of quality attribute
WHO
WHAT
WHEN
WHICH
HOW

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 6 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

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Usability and Its Tactics

Jan 11, 2015

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?

Usability Scenario Example

WHO	STIMULUS	IMPACTED PART	MITIGATING ACTION	MEASURABLE RESPONSE
End user	User Wants to • Learn system feature • Use systems efficiently • Minimize the impact of errors • Adapt system • Feel comfortable	**Whole System** **•At run time** **•At configure time**	• Learn ✓ Context sensitive help, familiar interface • Efficient use ✓ Aggregation of data and command, reuse of already entered data, good navigation, search mechanism, multiple activities • Error impact ✓ Undo, cancel, recover, auto-correct, retrieve forgotten information	• Task time • Number of errors • User satisfaction • Gain of user knowledge • Successful operations • Amount of time/data lost
End User	Downloads application	Runtime	Uses application productively	Takes 4 mins to be productive

Usability Tactics

Usability is essentially Human Computer Interaction. Runtime Tactics are

- User initiative (and system responds)
  - Cancel, undo, aggregation, store partial result
- System initiative
  - Task model: understands the context of the task user is trying and provide assistance
  - User model: understands who the user is and takes action
  - System model: gets the current state of the system and responds

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

System Initiated

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

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

Design how user can configure system’s use of resource

Technology selection

To achieve usability

Thank You for your time & attention !

Contact : parthasarathypd@wilp.bits-pilani.ac.in