SS ZG653: Software Architecture

Lecture 2: Software Structure and Quality Attributes

Instructor: Prof. Santonu Sarkar

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

Jan 11, 2015

Views and Architectural Structure- Recap..

- Architecture is a set of Views
  - Each view represents certain architectural aspects of the system, created for a stakeholder
  - All the views combined together form the consistent whole
- A Structure is the underlying part of a view- essentially the set of elements, and their properties
  - A view corresponding to a structure is created by using these elements and their inter-relationships

Many Views exist

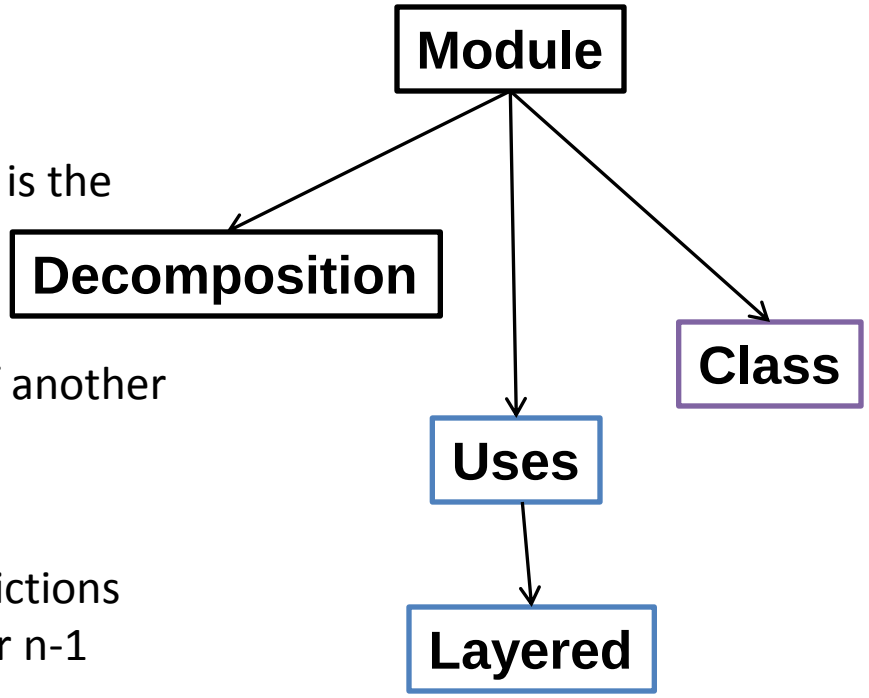
- Rational Unified Process/Kruchten 4+1 view (uses UML notations to describe these views)
- Siemens architecture framework- Conceptual, Module, Code, Execution views
- C4ISR framework – Operational, system and technical
- Classical approach – Data flow and control flow views
- RM-ODP (suitable for distributed system development) – 5 viewpoints

Three Structures will be covered

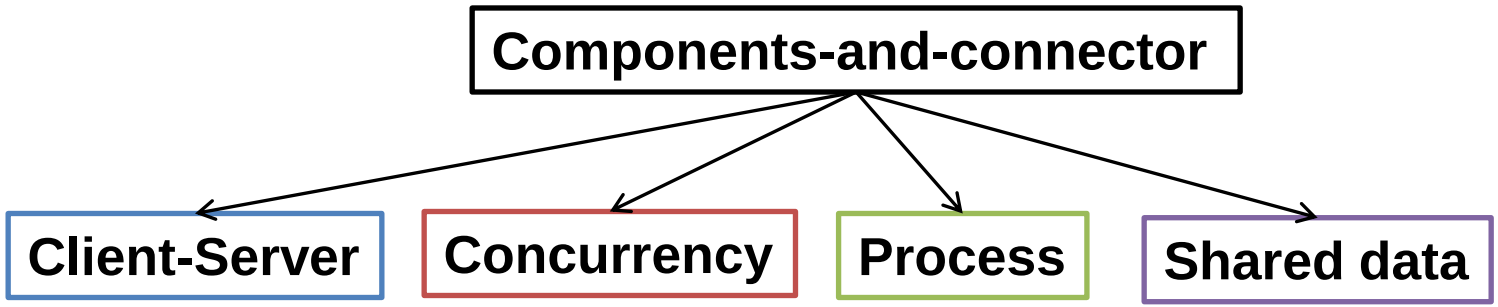
- Module Structure
  - How is the system to be structured as a set of functional units (modules)?
- Component-and-connector structures
  - Here component means a computation unit at runtime
  - Connector is the communication channel between the components
  - Models parallel execution
- Allocation structures
  - How is the system to relate to non-software structures in it's environment (CPU or cluster of CPUs, File Systems, Networks, Development Teams …)

Software Structures

- Module Structure includes
  - Decomposition
    - Sub-modules
    - All sub-mobiles combined together is the module
  - Uses
    - A modules uses the functionality of another module for its behavior
  - Layered
    - Hierarchical organization with restrictions that layer n uses the service of layer n-1
  - Class or generalization
    - Similar to OO concept

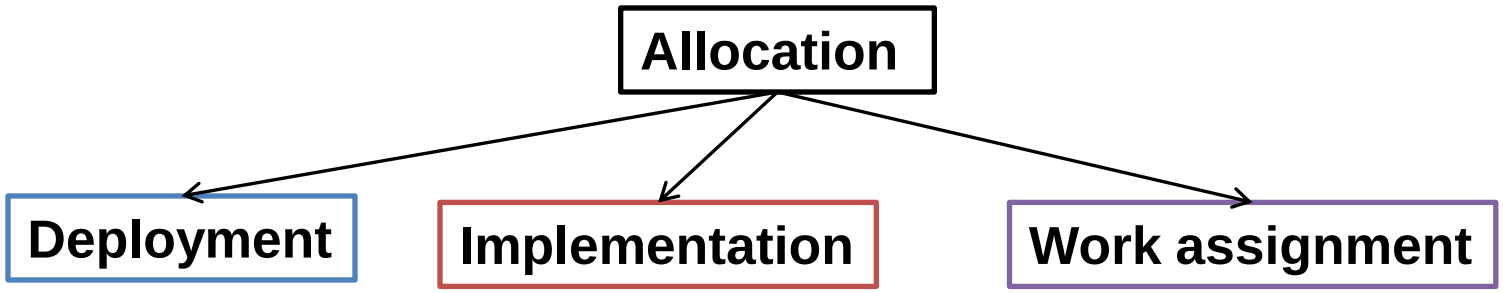


Component and Connectors



- Client-Server
  - Components are clients and servers and connectors are how they interact
- Concurrency
  - Opportunities of parallelism, where connectors are logical thread of execution dependency
- Process, or communicating processes
  - Components that are processes and connectors are how they communicate
- Shared data, or repository
  - Components have data store, and connectors describe how data is created, stored, retrieved

Allocation



- Deployment
  - Units are software (processes from component-connector) and hardware processors
  - Relation means how a software is allocated or migrated to a hardware
- Implementation
  - Units are modules (from module view) and connectors denote how they are mapped to files, folders
- Work assignment
  - Assigns responsibility for implementing and integrating the modules to people or team

Architectural Structures

Software Structure	Relations	Useful For
Decomposition	Is a sub-module of	Resource allocation and project structuring; information hiding, encapsulation; configuration control
Uses	Requires the correct presence of	Engineering subsets; engineering extensions
Layered	Requires the correct presence of; uses the services of; provides abstraction to	Incremental development; implementing systems on top of "virtual machines" portability
Class	Is an instance of; shares access methods of	In object-oriented design systems, producing rapid most-alike implementations from a common template

Architectural Structures

Software Structure	Relations	Useful For
Client-Server	Communicates with; depends on	Distributed operation; separation of concerns; performance analysis; load balancing
Process	Runs concurrently with; may run concurrently with; excludes; precedes; etc.	Scheduling analysis; performance analysis
Concurrency	Runs on the same logical thread	Identifying locations where resource contention exists, where threads may fork, join, be created or be killed
Shared Data	Produces data; consumes data	Performance; data integrity; modifiability

Architectural Structures

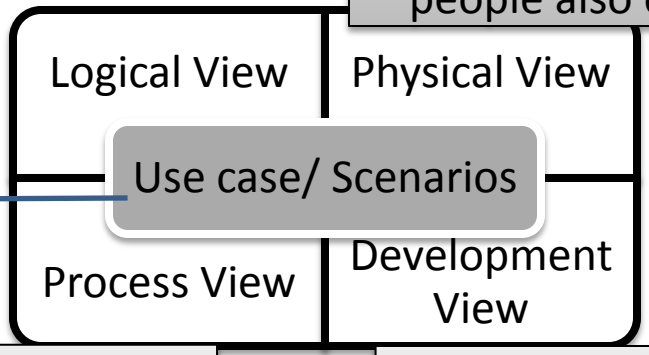
Software Structure	Relations	Useful For
Deployment	Allocated to; migrates to	Performance, availability, security analysis
Implementation	Stored in	Configuration control, integration, test activities
Work assignment	Assigned to	Project management, best use of expertise, management of commonality

Which Structure to Choose?

- Many opinions exist
- We will consider 4+1 view. This has been institutionalized as Rational Unified Process of Architecture description

End-user
- concerned with functionality
- essentially module structure

Infrastructure team
- concerned with topology, communication, security
- essentially allocation (deployment). Other people also call it as deployment view



Use-case- Binds all the four views together and describes the behavior of the system in a semi-structured text

Architect/Integrator
- concerned with performance, scalability..
- essentially component-connector

Development team
- concerned with software management
- essentially allocation (implementation and work-assignment)

SOFTWARE QUALITY ATTRIBUTES

A step back

- What is functionality?
  - Ability of the ability of the system to do the work for which it is intended
  - The structures and views we discussed so far, are meant for achieving functionality (mostly)
- Software Quality Attributes- also called non-functional properties
  - Orthogonal to functionality
  - is a constraint that the system must satisfy while delivering its functionality

Examples of Quality Attributes

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

- 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
- However, the success 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

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 is all-compassing and overlapping

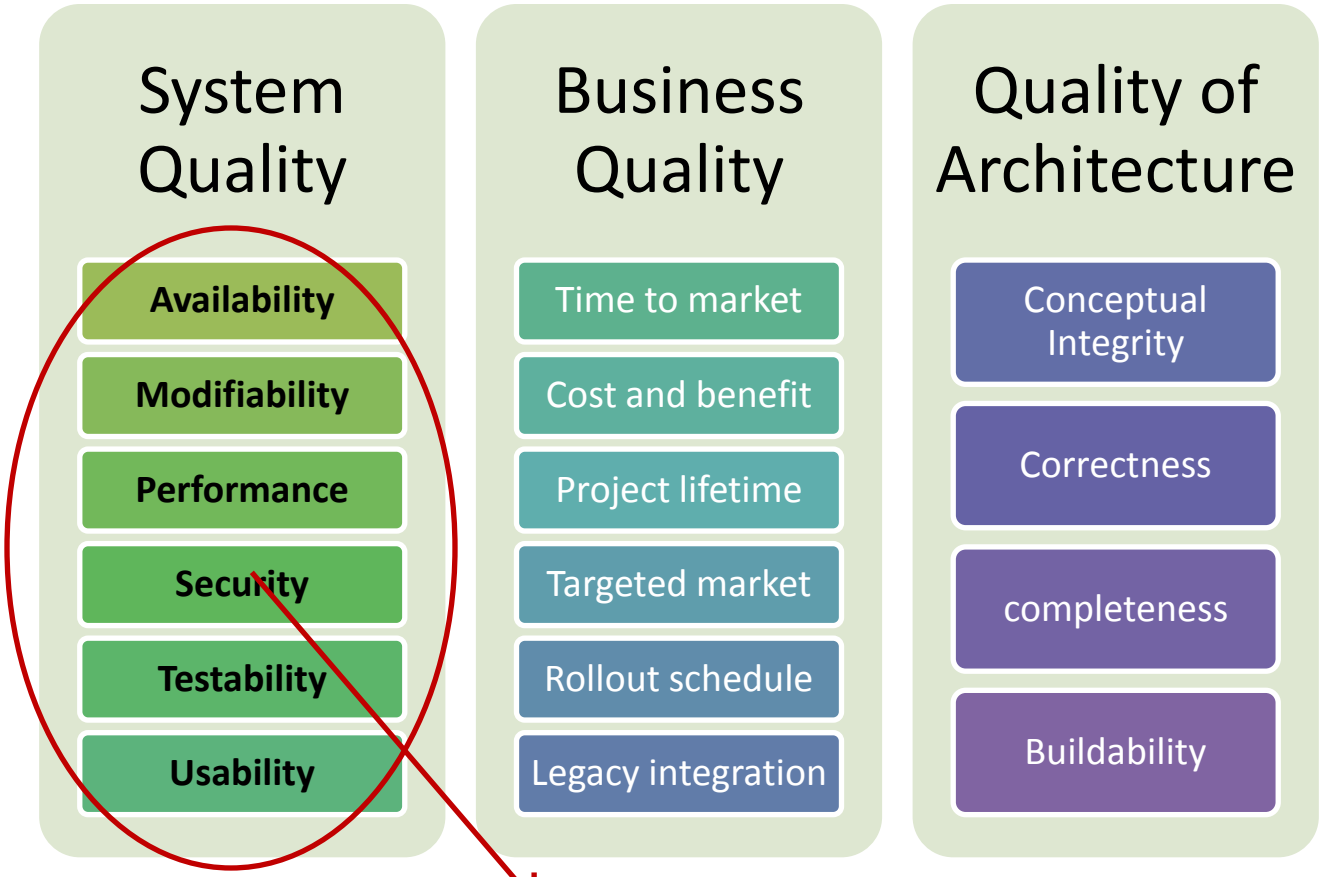
- Quality needs to be achieved throughout the design, implementation and deployment
- Big picture is important
- Architecture is critical
  - Qualities should be designed in and also evaluated at the architectural level
- Architecture alone is not sufficient
  - it is the foundation; however details are equally important
- 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

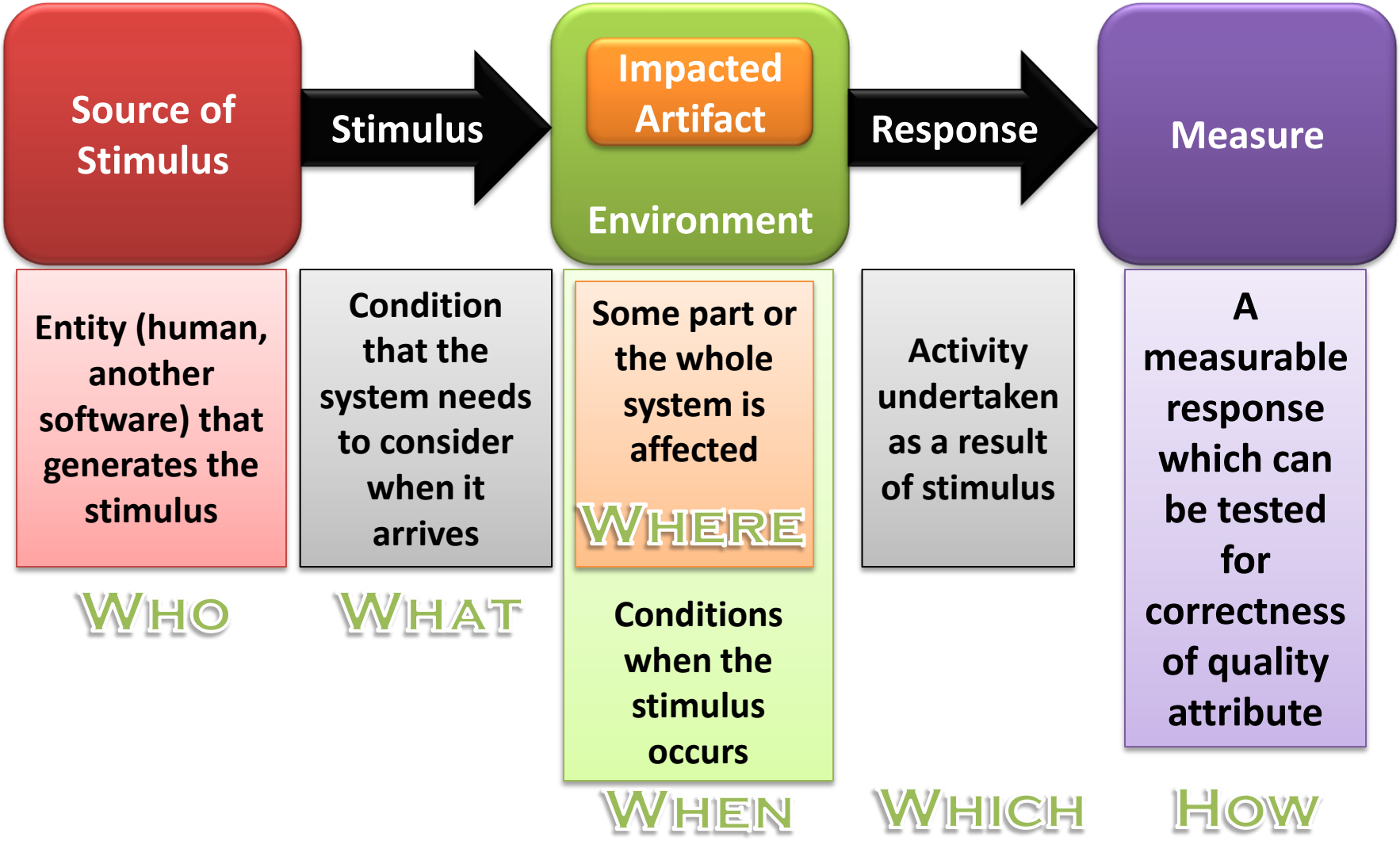
- ISO 9126 and ISO 25000 attempts to create a framework to define quality attributes

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

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	•System/Product which 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 customisability 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

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

End user

STIMULUS

User Wants to

- Learn system feature
- Use systems efficiently
- Minimize the impact of errors
- Adapt system
- Feel comfortable

IMPACTED PART

Whole System

- **At run time**
- **At configure time**

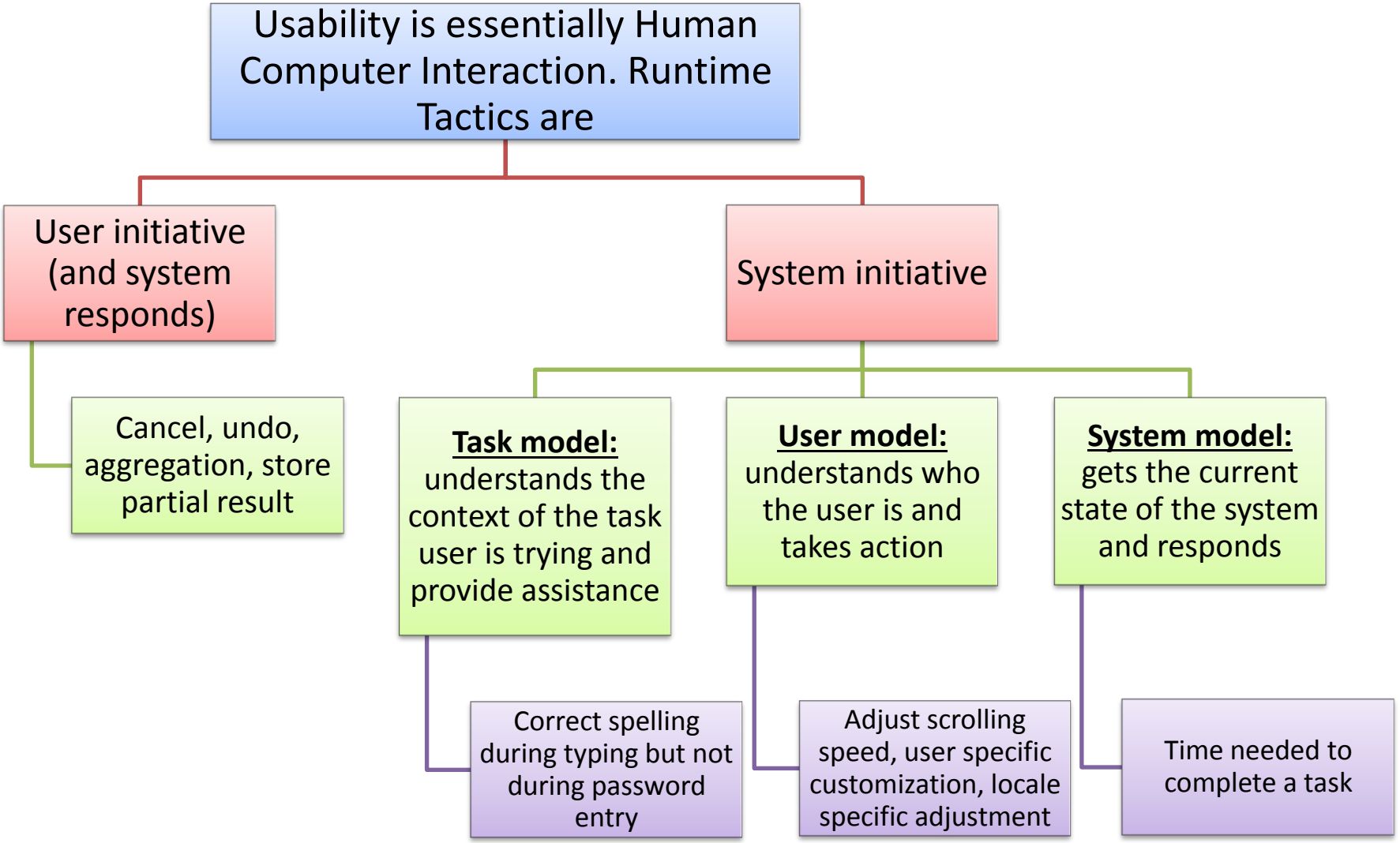
MITIGATING ACTION

- 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

MEASURABLE RESPONSE

- Task time
- Number of errors
- User satisfaction
- Gain of user knowledge
- Successful operations
- Amount of time/data lost

Usability Tactics



Usability Tactics….

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

Thank You