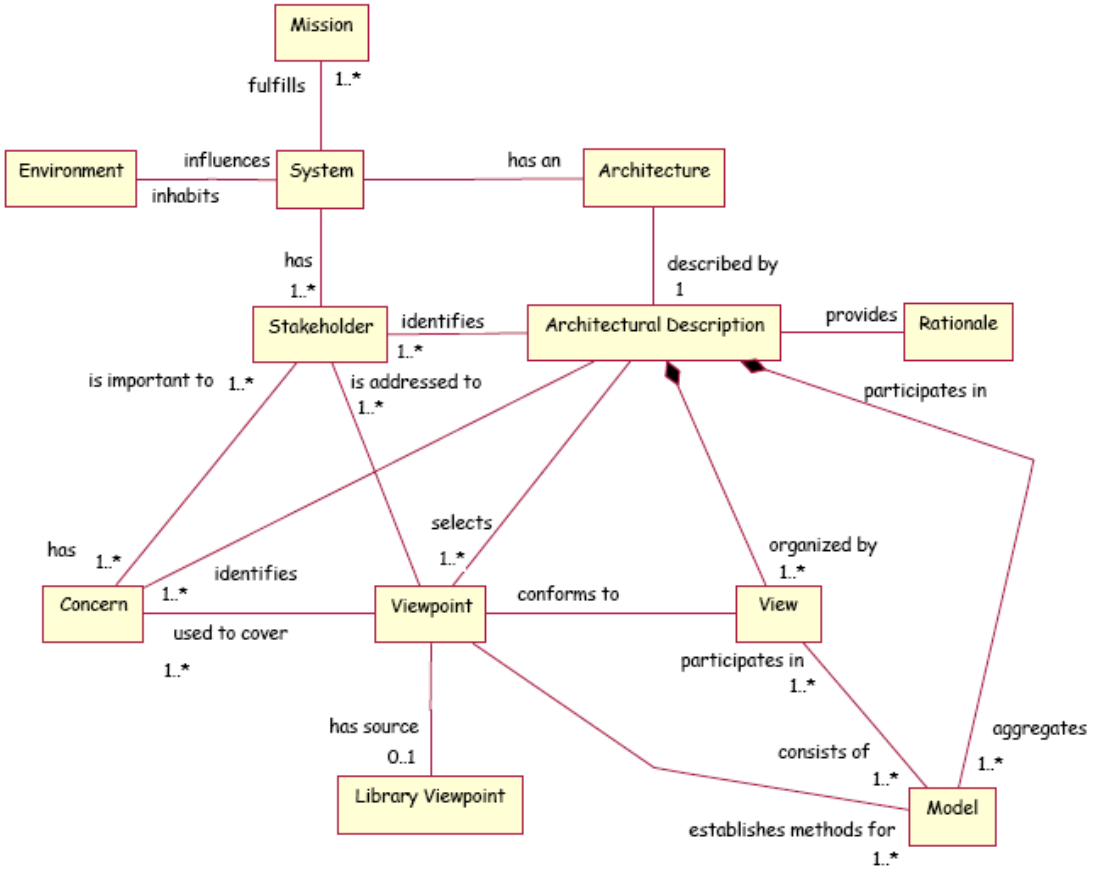
SOFTWARE ARCHITECTURE

Lecture 2: Quality Attributes

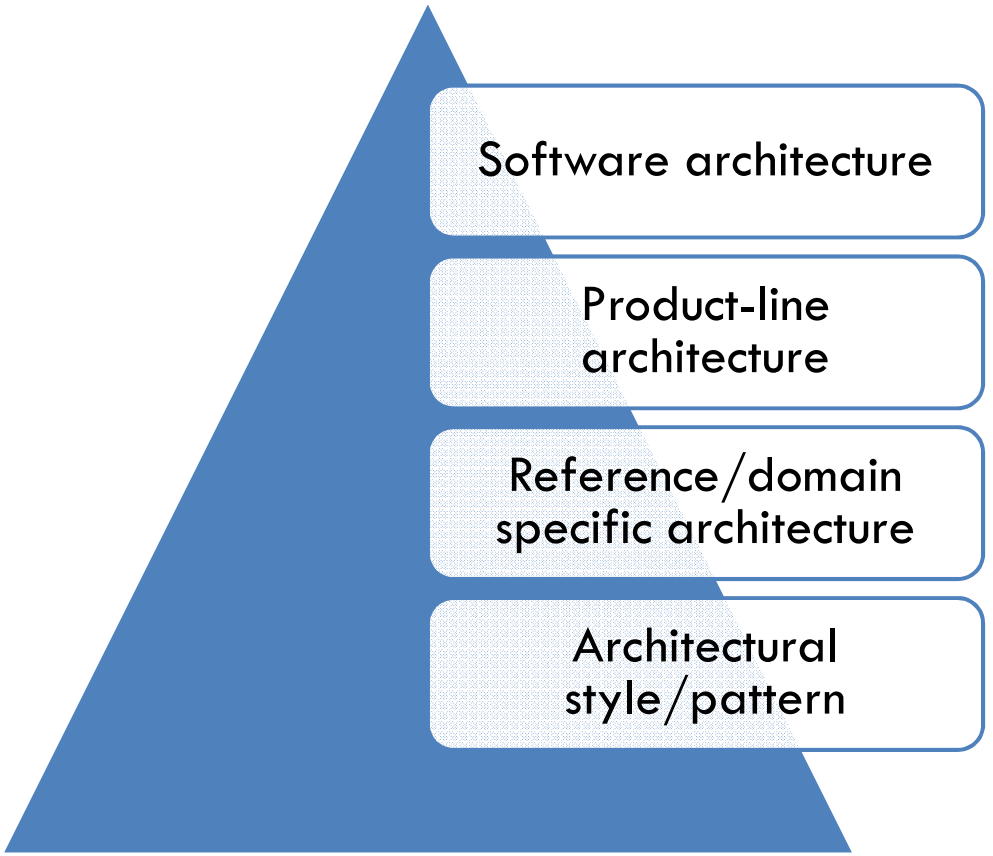
Last Lecture

- Definition 1 (Bass, Clements, Kazman, Software Architecture in Practice, 2nd edition. 2003):
  - The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationship among them
- Definition 2 (ANSI/IEEE Std 1471, ISO/IEC 42010):
  - The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

What?


Mission
fulfills
1..*
Environment
influences
inhabits
System
has an
Architecture
has
1..*
described by
1
Stakeholder
identifies
1..*
Architectural Description
provides
Rationale
is important to
1..*
is addressed to
1..*
participates in
selects
1..*
has
1..*
identifies
1..*
organized by
1..*
Concern
Viewpoint
conforms to
View
used to cover
1..*
participates in
1..*
has source
0..1
aggregates
1..*
consists of
1..*
Library Viewpoint
Model
establishes methods for
1..*

How?


Software architecture
Product-line architecture
Reference/domain specific architecture
Architectural style/pattern

Why?

- Cope with complexity of software systems
  - need "standard" solutions to common problem types - **reuse**
  - need some way to talk about them - **understanding, communication**
  - need some way to organize them - **construction, evolution**
  - need some way to reason about them - **analysis, management**
  - **large scale decisions** are more important than data structures and algorithms

This Lecture

- What means a “good architecture”?
- Which are the “quality attributes” of a good architecture?

Functional and Non-Functional Requirements

- Requirements analysis (or engineering) needs to separate functional from non-functional concerns
- Both functional and non-functional concerns influence architectural decision … and both are hard to define
- **Quality attributes** of an architecture are non-functional requirements
- Functional requirements and **quality attributes** are *orthogonal*
- As all the other requirements, **quality attributes** are stakeholders' dependent

Stakeholders' Concerns

- Architectures influenced by stakeholders
  - **Management:** low cost, keep people employed
  - **Marketing:** short time to market, low cost
  - **Customer:** timely delivery, not changed often
  - **End user:** behavior, performance, security
  - **Maintainer:** scalability
  - …

Architecture and Quality Attributes

- Quality attributes considered during
  - Design, implementation, deployment
  - No attribute is dependent only on one phase
- Architecture critical for realizing many quality attributes
  - These qualities designed and evaluated at architectural level
- Architecture does not achieve these qualities
  - They are achieved via details (implementation)

Quality Attributes

- System qualities
  - Availability, modifiability, performance, security, testability, usability
- Business qualities
  - Time to market, cost and benefit, projected lifetime of system, targeted market, rollout schedule, integration with legacy systems
- Architecture qualities
  - Conceptual integrity, correctness, completeness, buildability

Quality Attribute Scenario

- Quality attribute specific requirement, containing:
  - Source of stimulus
  - Stimulus
  - Environment
  - Artifact
  - Response
  - Response measure
- General QA scenarios
  - System independent
  - Can potentially pertain to any system
- Concrete QA scenarios
  - Specific to the particular system under consideration
  - Same role as use cases for specifying functional requirements

Quality Attribute Scenario Parts

- Source of stimulus
  - Human/computer system/other actor generating the stimulus
- Stimulus
  - Condition to be evaluated when arrives at the system
- Environment
  - The conditions the system is in (overload/running/failed etc)
- Artifact
  - Part of the system that is stimulated
- Response
  - Activity undertaken upon stimulus arrival
- Response measure
  - Response should be measurable in some manner so that the requirement can be tested

Quality Attributes Presentation

- Presented as a collection of general scenarios
  - Table giving possible system-independent values for each of the six parts
  - General scenario: choose one value per element
- As requirements for a particular system
  - The relevant general scenarios are made system specific

Quality Attribute Scenario Generation

- We need to generate meaningful QA requirements for a system
  - Requirements gathering phase
- Starting point, but not disciplined enough recording
- We generate concrete QA scenarios:
  - First create general scenarios from tables
  - From them derive system-specific scenarios

Availability

- Readiness of usage
- Concerned with system failures and consequences
- Failure
  - Deviation from intended functional behavior
  - Observable by system users
- Fault vs. Failure
  - Fault: event which may cause an error
  - Failure: incorrect internal system state

Availability Concerns

- How system failure is detected?
- How frequently system failure may occur?
- What happens when a failure occurs?
- How long is a system allowed to be inoperable?
- When can failures occur safely?
- How to prevent failures?
- What kind of notifications are required when a failure occurs?

Repairment and Maintenance

- Time to repair:
  - Time until failure is no longer observable
- Automatic repair
- Maintenance: scheduled downtimes
- Probability
  - Mean time to fail/(mean time to fail + mean time to repair)

Availability General Scenarios

Source	Internal/external to system
Stimulus	Fault: omission, crash, timing, response etc
Artifact	System's processors, communication channels, persistent storage, processes etc
Environment	Normal operation or degraded mode
Response	Detect event and record it/notify appropriate parties/disable event sources causing faults/failures/ be unavailable for an interval/ continue
Response measure	Time interval of available system, availability time, time interval of degraded mode, repair time

Modifiability

- Concerns cost of change
  - What can change the artifact?
  - When and by whom is the change made?
- Upon specifying a change
  - New implementation must be designed, implemented, tested, deployed
  - All these cost time and money
  - Time and money can be measured

What Can Change the Artifact?

- Any aspect of a system: add/ delete/ modify
  - The functions the system computes
  - The platform the system exists on (=> portability)
    - HW, OS, MW
  - System environment
    - Systems to interoperate with, communication protocols
  - System qualities
    - Reliability, performance, modifiability
  - Capacity
    - Number of users supported, of supported operations

When and by Whom is the Change Made?

- Implementation
  - Modifying source code
- Build time
- Configuration setup
- Execution
- By:
  - Developers, end users, system administrators

Modifiability General Scenarios

Source	End user, developer, system administrator
Stimulus	Wishes to add/delete/modify/vary functionality, QA, capacity etc
Artifact	System UI, platform, environment, systems that interoperates with target system
Environment	Design time, implementation integration , build time, runtime
Response	Locates place in architecture to modify, makes modification w/o affecting other functions, tests modifications, deploys modifications
Response measure	Costs in terms of number of elements affected, effort, money; extent to which this affects other QAs, functions

Performance

- Concerned with timing
  - How long it takes a system to respond when an event occurs
- Events
  - Interrupts, messages, requests from users, passage of time
- Complication
  - Number of event sources and arrival patterns

Arrival Pattern for Events

- Periodic
  - E.g., every10 ms
  - Most often seen in real-time systems
- Stochastic
  - Events arrive according to some probabilistic distribution
- Sporadic

System Responses

- Latency
  - Time between stimulus arrival and system's response to it
- Deadlines in processing
- Throughput in the system
  - Number of transactions system can process in a second
- Jitter of the response
  - Variation in latency
- Number of events not processed
  - Because system too busy to respond
- Lost data
  - Because system too busy

Performance General Scenarios

Source	Independent sources
Stimulus	Periodic or stochastic or sporadic events occur
Artifact	System
Environment	Normal mode, overload mode
Response	Processes stimuli; changes level of service
Response measure	Latency, deadline, throughput, jitter, miss rate, data loss

Security

- Measure of system's ability
  - To resist unauthorized usage
  - To provide services to legitimate users
- Attack: attempt to breach security
  - Unauthorized attempt to access data/services
  - Unauthorized attempt to modify data
  - Attempt to deny services to legitimate users

Example of Attacks

- Theft of money by electronic means
- Theft of credit card numbers
- Destruction of files on computer systems
- Denial-of-service, attacks by worms, viruses

Security Components

- Non repudiation
  - Transaction cannot be denied by any of its parties
- Confidentiality
  - Data/service protected from unauthorized access
- Integrity
  - Data/services delivered as intended
- Assurance
  - Parties in a transactions are who they say they are
- Availability
  - System ready for legitimate use
- Auditing
  - System tracks activities within it to be able to reconstruct them

Security General Scenarios

Source	Individual/system: identity, internal/external, authorization, access
Stimulus	Try to: display data, change/delete data, access system services, reduce availability
Artifact	System services, data within system
Environment	On/offline, (dis)connected, firewalled or open
Response	Various
Response measure	Various

Security Scenario Responses

- Authenticates user
- Hides identity of user
- Blocks/allows acces to data/services
- Grants/withdraws permission to access data/services
- Records access/modification or attempts
- Stores data in certain formats
- Recognizes access/usage roles
- Informs users on other systems
- Restricts availability of services

Security Scenario Response Measure

- Time/effort/resources for circumventing security measures successfully
- Probability of detecting attack, identifying attacker
- Percentage of services still available under DoS attack
- Restore data/services
- Extent to which data/services damaged or legitimate access denied

Testability

- Easiness with which SW can demonstrate its faults through testing
  - Testing: 40% costs of developing good systems
- Probability that SW will fail on its next test execution
  - Assuming the software has at least one fault
- Response measures
  - Effectiveness of tests (in finding faults)
  - How long do satisfactory tests last

Testable System

- It must be possible
  - To control internal state and inputs of each component
  - To observe the outputs
- Test harness
- Specialized software designed for exercising the SW under test

Who and What?

- Who does it
  - Various developers, testers, verifiers, users
- Last step of SW development cycle
- What to test
  - Portions of code
  - Design
  - Complete system

Testability General Scenarios

Source	Unit developer, increment integrator, system verifier, client acceptance tester, system user
Stimulus	Analysis, architecture, design, class, subsystem integration completed, system delivered
Artifact	Design part, code part, complete application
Environment	At design time, at development time, at compile time, at deployment time
Response	Provides access to state values; provides computed values; prepares test environment
Response measure	Percent executable statements executed, probability of failure if fault exists, time to perform tests, length of longest dependency chain in a test, length of time to prepare test environment

Usability

- Concerned with
  - How easy it is for the user to accomplish a desired task
  - User support type the system provides
- Usability problems are usually discovered during prototype building and user testing
- Later in process and deeper in architecture the repair needs to go: the more expensive

Usability Areas

- Learning system features
- Using a system efficiently
- Minimizing the impact of errors
- Adapting system to user needs
- Increasing confidence and satisfaction

Usability General Scenarios

Source	End user
Stimulus	Wants to learn system features, use system efficiently, minimize impact of errors, adapt system, feel comfortable
Artifact	System
Environment	At runtime and configure time
Response	Various
Response measure	Task time, number of errors, number of problems solved, user satisfaction, user knowledge gain, ratio of successful operations to total operations, amount of time/data lost

Usability Scenario Responses

- System provides responses to support
  - Learn system features
  - Use system efficiently
  - Minimize impact of errors
  - Adapt system
  - Feel comfortable

Stimuli

Availability	Unexpected event, non occurrence of expected event
Modifiability	Request to add/delete/modify/vary functionality, QA, capacity, platform, etc
Performance	Periodic or stochastic or sporadic events occur
Security	Tries to: display data, change/delete data, access system services, reduce availability
Testability	Analysis, architecture, design, class, subsystem integration completed, system delivered
Usability	Wants to learn system features, use system efficiently, minimize impact of errors, adapt system, feel comfortable

Communicating Concepts

- General scenarios
  - Need to make stakeholders communicate
- Each attribute community has own vocabulary
  - Different terms can mean the same thing
- Stimuli can occur during runtime or before
- Architect's job:
  - Understand which stimuli represent the same occurrence or are aggregates of other stimuli or are independent
- When stimuli relations are clear
  - Communicate them to stakeholders
  - Use appropriate language for each stakeholder category

Other System Qualities

- Scalability
  - Captured as modifiability of capacity (e.g.: number of supported users)
- Portability
  - Captured as modifiability of platform

Business Qualities

- Goals centering on considerations of:
  - Cost
  - Schedule
  - Market
  - Marketing
- Examples:
  - Time to market
  - Cost and benefit
  - Projected lifetime of system
  - Targeted market
  - Rollout schedule
  - Integration with legacy systems

Time to Market

- Important when
  - Competitive pressure
  - Window of opportunity for system/product is short
- Pressure to buy/reuse existing elements
  - COTS
  - Previous projects elements
- Decomposition of system into elements

Cost and Benefit

- Every project has a budget
- Different architectures yield different development costs
  - Architecture relying on out-of-house technology: more expensive
  - Highly flexible architecture: more expensive to build than rigid architecture
    - Less costly to maintain and modify though

Projected Lifetime

- Long projected time:
  - Portability, scalability, modifiability important
  - Building extra layers for these: increases time to market
  - In many cases this is worthy

Targeted Market

- General purpose software
  - Platform and feature set determine size of potential market
  - Portability and functionality: key
- Attract large market with collection of related products
  - Design product-line approach
  - Core system common (and portable)
  - Layers of specific software constructed around core

Rollout Schedule

- Product introduced as base functionality
- Many features scheduled to release later
  - Flexibility
  - Customizability
  - System constructed with expansion in mind

Integration with Legacy Systems

- Appropriate integration mechanisms
- Substantial architectural implications

Architecture Qualities

- Qualities related directly to the architecture itself:
  - Conceptual integrity
  - Correctness
  - Completeness
  - Buildability

Buildability

- Allows a system
  - To be completed by available team in timely manner
  - To be open to certain changes as development progresses
- Refers to ease of constructing desired system
- Knowledge about problem to be solved important

Buildability Architecturally Achieved

- By paying attention to
  - Decomposition into modules
  - Assigning modules to development teams
  - Limiting dependencies between modules
  - Limiting number of teams
- Goal
  - Maximize development parallelism