Architectures in Context

Software Architecture

Fundamental Understanding

- Architecture is a set of principal design decisions about a software system
- Three fundamental understandings of software architecture
  - Every application has an architecture
  - Every application has at least one architect
  - Architecture is not a phase of development

Wrong View: Architecture as a Phase

Treating architecture as a phase denies its foundational role in software development

More than "high-level design"

Architecture is also represented, e.g., by object code, source code, …

Context of Software Architecture

- Requirements
- Design
- Implementation
- Analysis and Testing
- Evolution
- Development Process

Requirements Analysis

- Traditional SE suggests requirements analysis should remain unsullied by any consideration for a design
- However, without reference to existing architectures it becomes difficult to assess practicality, schedules, or costs

  In building architecture we talk about specific rooms…

  …rather than the abstract concept "means for providing shelter"
- In engineering new products come from the observation of existing solution and their limitations

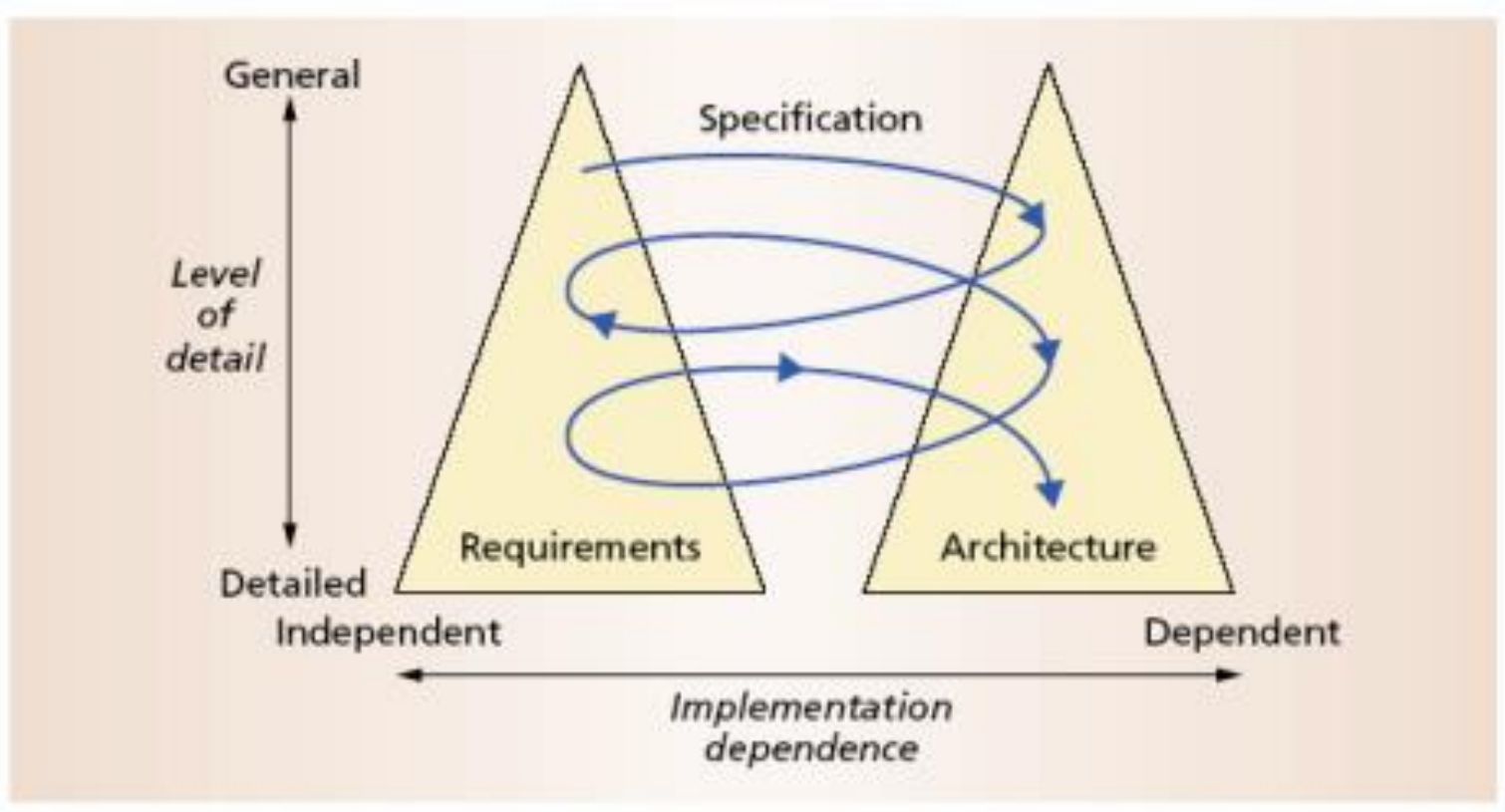
New Perspective on Requirements Analysis

- Existing designs and architectures provide the solution vocabulary
- Our understanding of what works now, and how it works, affects our wants and perceived needs
- The insights from our experiences with existing systems
  - helps us imagine what might work and
  - enables us to assess development time and costs
- Requirements analysis and consideration of design must be pursued at the same time

Non-Functional Properties (NFP)

- NFPs are the result of architectural choices
- NFP questions are raised as the result of architectural choices
- Specification of NFP might require an architectural framework to even enable their statement
- An architectural framework will be required for assessment of whether the properties are achievable

The Twin Peaks Model



Design and Architecture

- Design is an activity that pervades software development
- It is an activity that creates part of a system's architecture
- Typically in the traditional Design Phase decisions concern
  - A system's structure
  - Identification of its primary components
  - Their interconnections
- Architecture denotes the set of principal design decisions about a system
  - That is more than just structure

Architecture-Centric Design

- Traditional design phase suggests translating the requirements into algorithms, so a programmer can implement them
- Architecture-centric design
  - stakeholder issues
  - decision about use of COTS component
  - overarching style and structure
  - package and primary class structure
  - deployment issues
  - post implementation/deployment issues

Design Techniques

- Basic conceptual tools
  - Separation of concerns
  - Abstraction
  - Modularity
- Two illustrative widely adapted strategies
  - Object-oriented design
  - Domain-specific software architectures (DSSA)

Object-Oriented Design (OOD)

- Objects
  - Main abstraction entity in OOD
  - Encapsulations of state with functions for accessing and manipulating that state

Pros and Cons of OOD

- Pros
  - UML modeling notation
  - Design patterns
- Cons
  - Provides only
    - One level of encapsulation (the object)
    - One notion of interface
    - One type of explicit connector (procedure call)
      - Even message passing is realized via procedure calls
  - OO programming language might dictate important design decisions
  - OOD assumes a shared address space

DSSA

- Capturing and characterizing the best solutions and best practices from past projects within a domain
- Production of new applications can focus on the points of novel variation
- Reuse applicable parts of the architecture and implementation
- Applicable for product lines

  Recall the Philips Koala example discussed in the previous lecture

Implementation

- The objective is to create machine-executable source code
  - That code should be faithful to the architecture
    - Alternatively, it may adapt the architecture
    - How much adaptation is allowed?
    - Architecturally-relevant vs. -unimportant adaptations
  - It must fully develop all outstanding details of the application

Faithful Implementation

- All of the structural elements found in the architecture are implemented in the source code
- Source code must not utilize major new computational elements that have no corresponding elements in the architecture
- Source code must not contain new connections between architectural elements that are not found in the architecture
- Is this realistic?
  Overly constraining?
  What if we deviate from this?

Unfaithful Implementation

- The implementation does have an architecture
  - It is latent, as opposed to what is documented.
- Failure to recognize the distinction between planned and implemented architecture
  - robs one of the ability to reason about the application's architecture in the future
  - misleads all stakeholders regarding what they believe they have as opposed to what they really have
  - makes any development or evolution strategy that is based on the documented (but inaccurate) architecture doomed to failure

Implementation Strategies

- Generative techniques

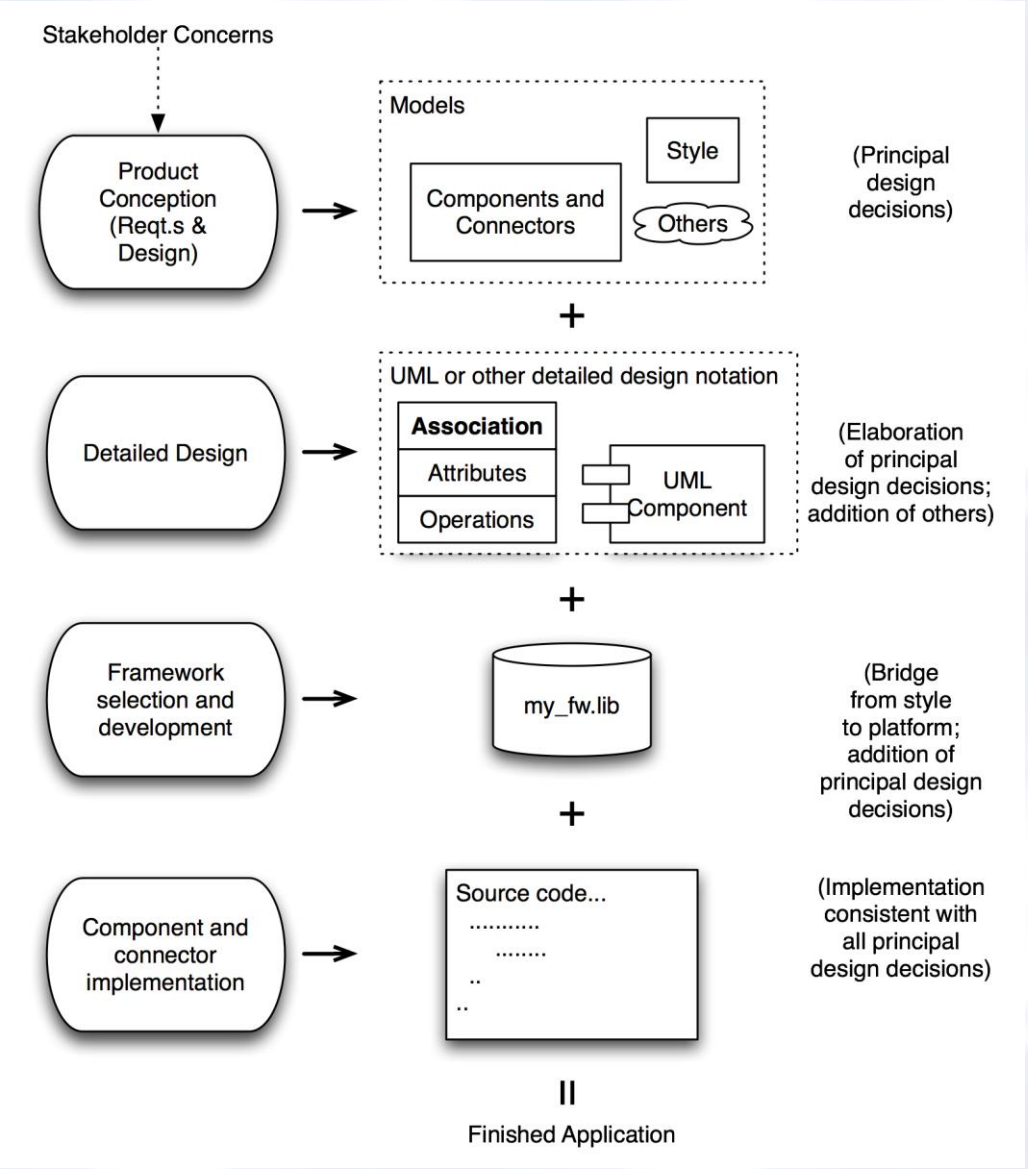
  e.g. parser generators
- Frameworks

  collections of source code with identified places where the engineer must "fill in the blanks"
- Middleware

  CORBA, DCOM, RPC, …
- Reuse-based techniques

  COTS, open-source, in-house
- Writing all code manually

How It All Fits Together



Analysis and Testing

- Analysis and testing are activities undertaken to assess the qualities of an artifact
- The earlier an error is detected and corrected the lower the aggregate cost
- Rigorous representations are required for analysis, so precise questions can be asked and answered

Analysis of Architectural Models

- Formal architectural model can be examined for internal consistency and correctness
- An analysis on a formal model can reveal
  - Component mismatch
  - Incomplete specifications
  - Undesired communication patterns
  - Deadlocks
  - Security flaws
- It can be used for size and development time estimations

Analysis of Architectural Models (cont'd)

- Architectural model
  - may be examined for consistency with requirements
  - may be used in determining analysis and testing strategies for source code
  - may be used to check if an implementation is faithful

Evolution and Maintenance

- All activities that chronologically follow the release of an application
- Software will evolve
  - Regardless of whether one is using an architecture-centric development process or not
- The traditional software engineering approach to maintenance is largely ad hoc
  - Risk of architectural decay and overall quality degradation
- Architecture-centric approach
  - Sustained focus on an explicit, substantive, modifiable, faithful architectural model

Architecture-Centric Evolution Process

- Motivation
- Evaluation or assessment
- Design and choice of approach
- Action

  includes preparation for the next round of adaptation

Processes

- Traditional software process discussions make the process activities the focal point
- In architecture-centric software engineering the product becomes the focal point
- No single "right" software process for architecture-centric software engineering exists

Turbine – A New Visualization Model

- Goals of the visualization

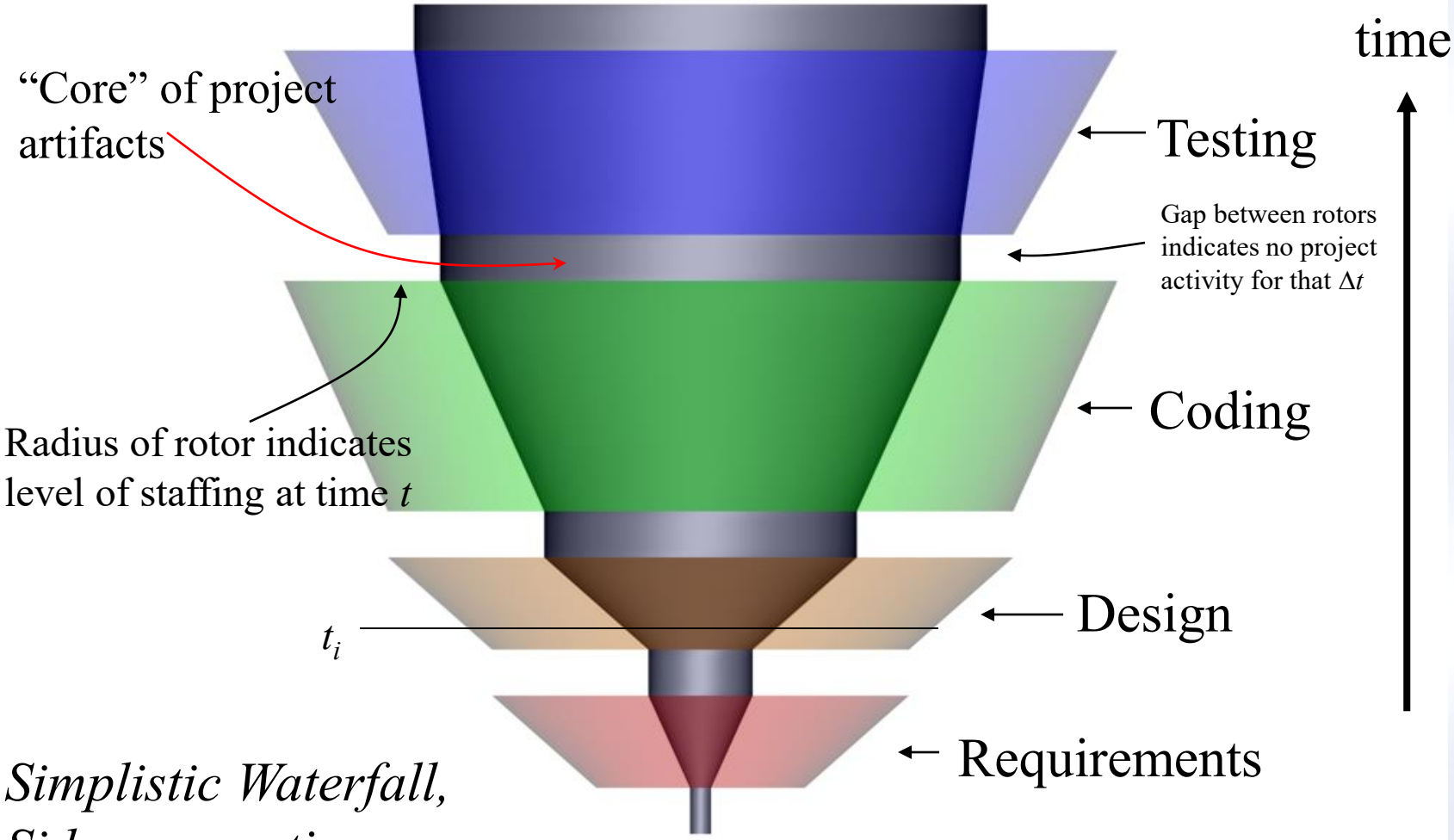
  Provide an intuitive sense of
  - Project activities at any given time
    - Including concurrency of types of development activities
  - The "information space" of the project

  Show centrality of the products
  - (Hopefully) Growing body of artifacts
  - Allow for the centrality of architecture
    - But work equally well for other approaches, including "dysfunctional" ones

  Effective for indicating time, gaps, duration of activities

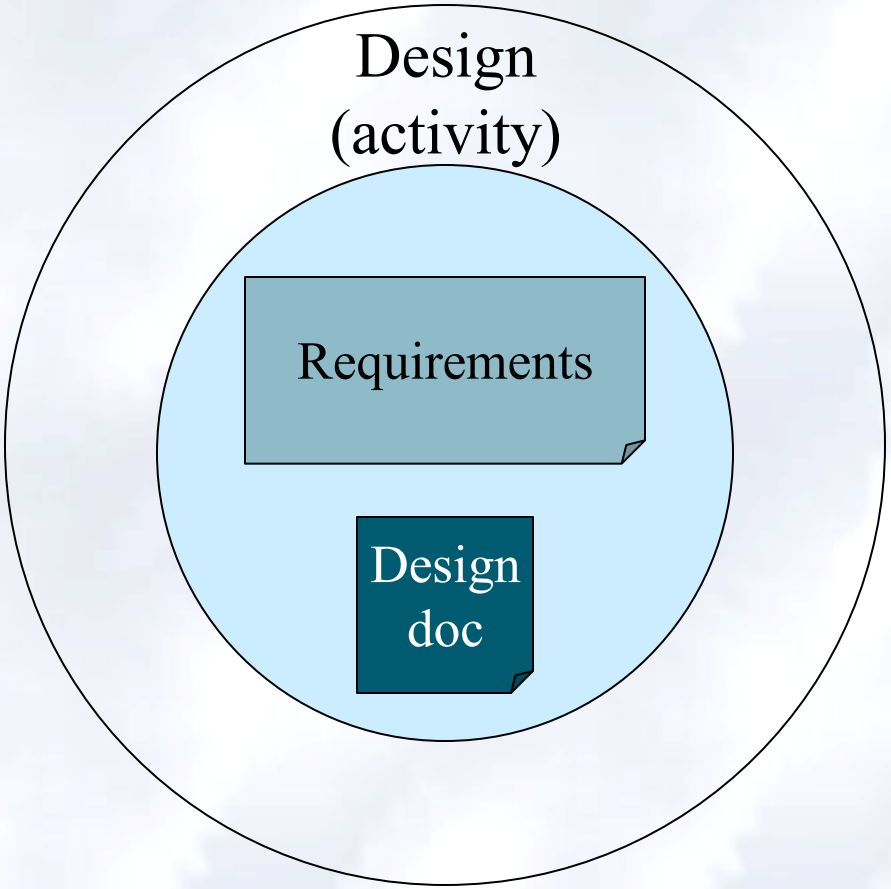
  Investment (cost) indicators

The Turbine Model

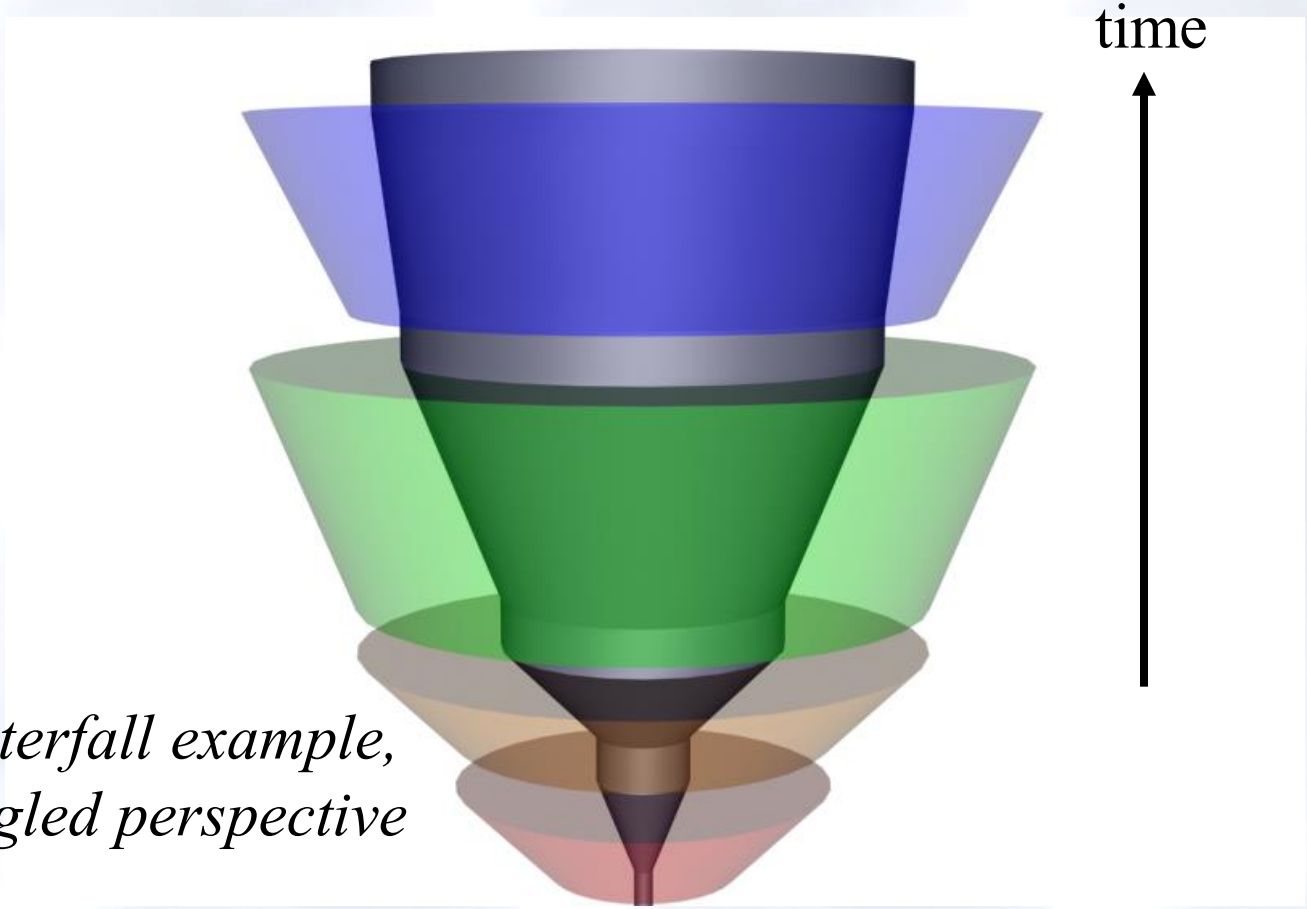


Simplistic Waterfall, Side perspective

Cross-section at time t_i

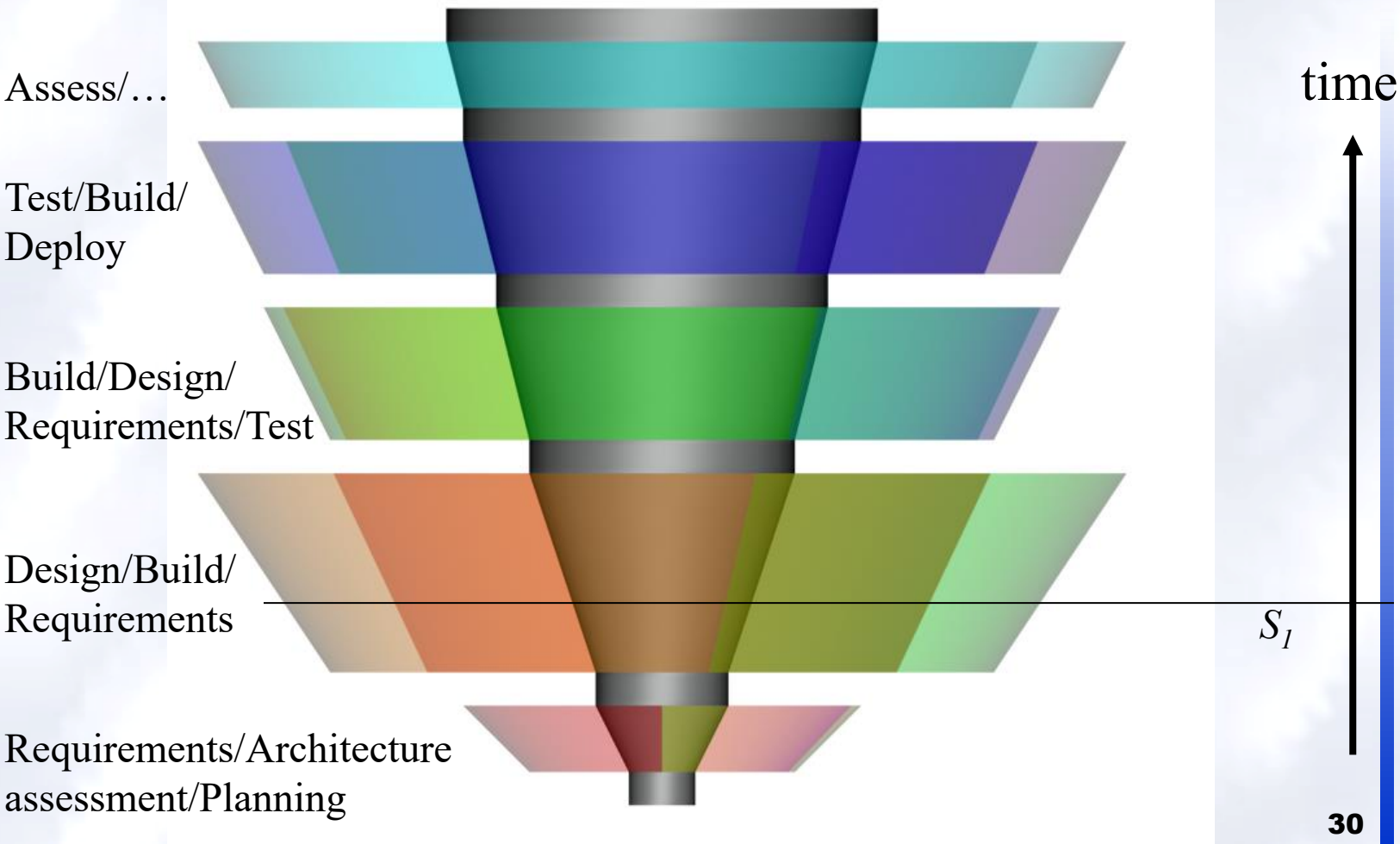


The Turbine Model

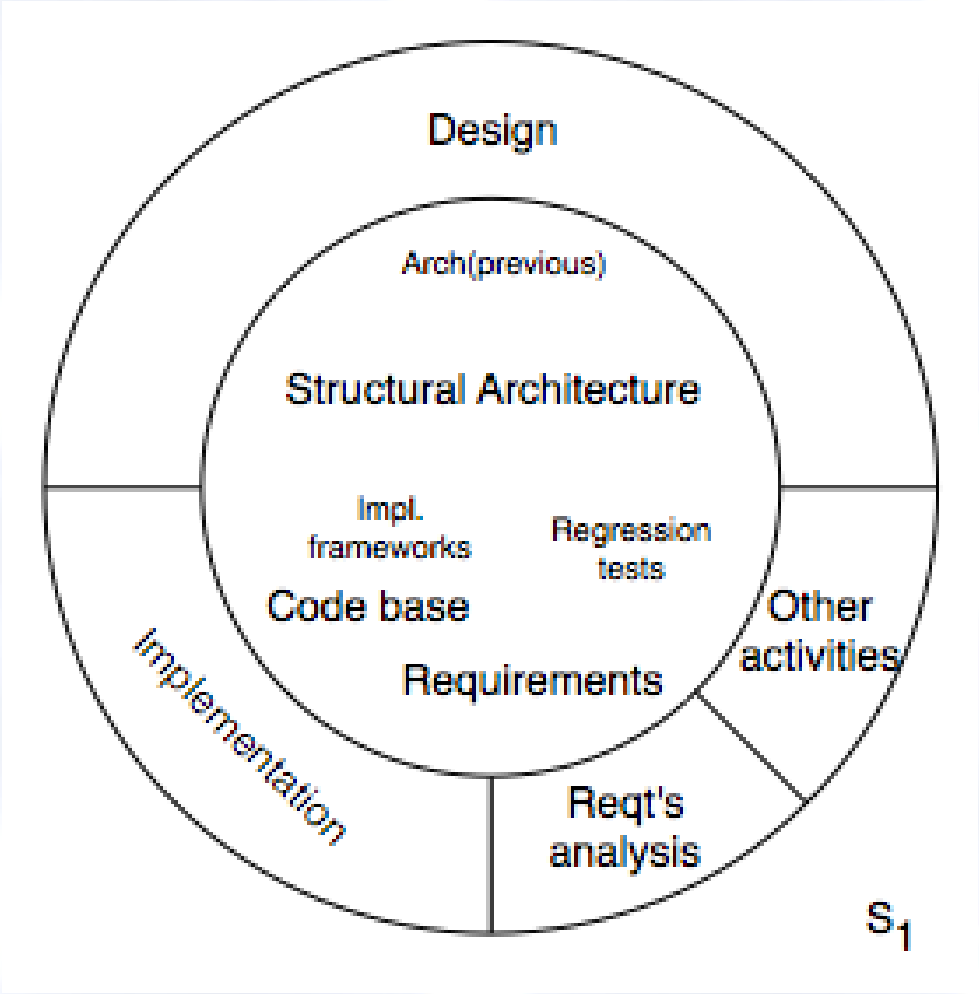


Waterfall example,
Angled perspective

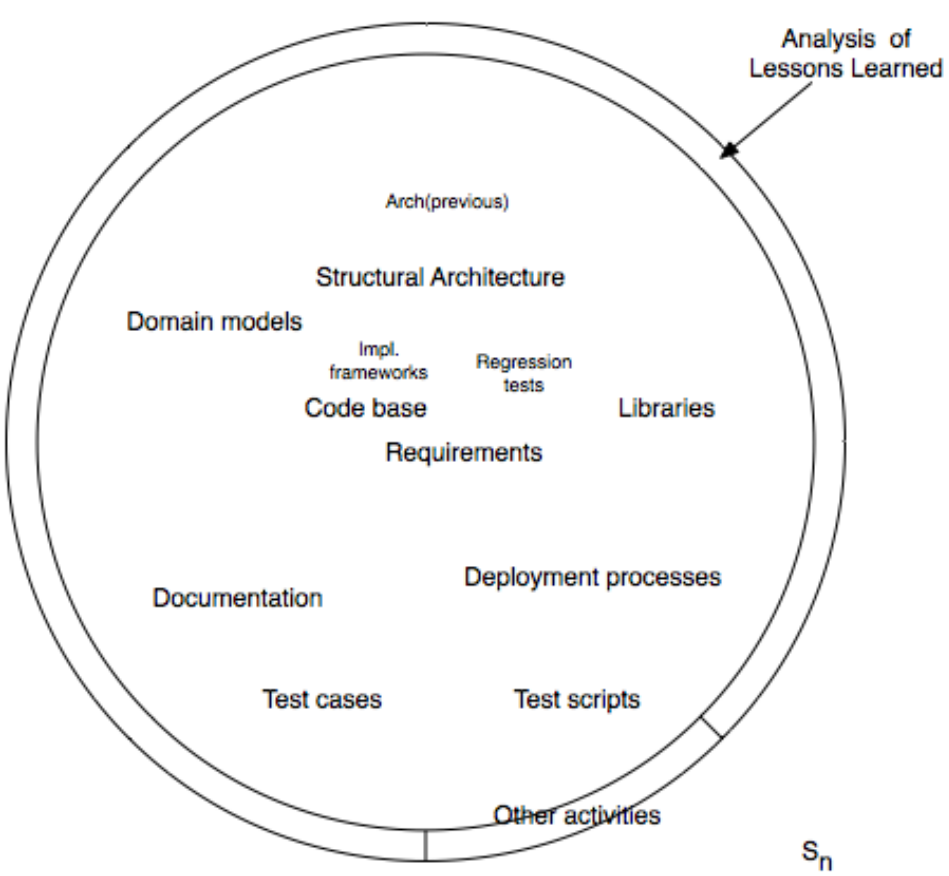
A Richer Example



A Sample Cross-Section



A Cross-Section at Project End



Volume Indicates Where Time was Spent

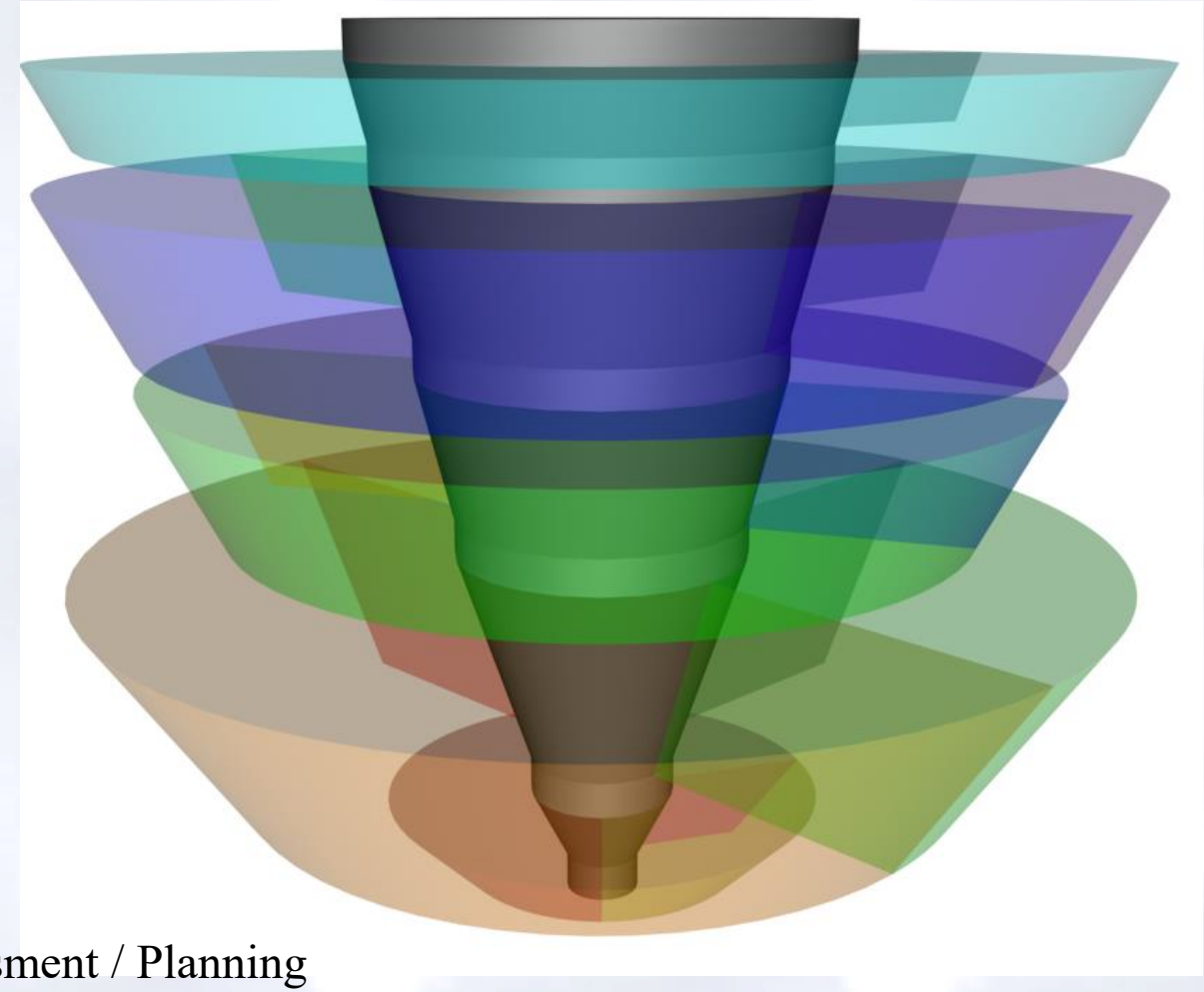
Assess/…

Test/Build/
Deploy

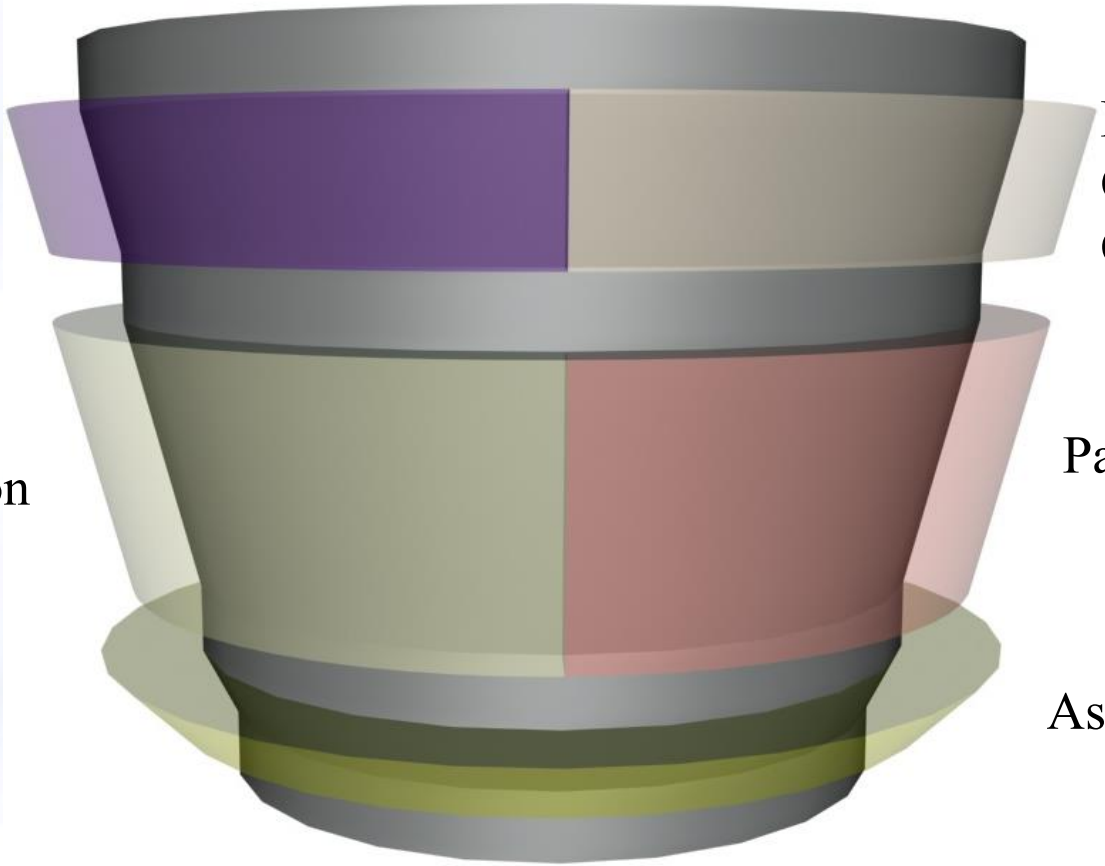
Build/Design/
Requirements/Test

Design/Build/
Requirements

Requirements/
Architecture Assessment / Planning



A Technically Strong Product-Line Project



Visualization Summary

- It is illustrative, not prescriptive
- It is an aid to thinking about what's going on in a project
- Can be automatically generated based on input of monitored project data
- Can be extended to illustrate development of the information space (artifacts)

  The preceding slides have focused primarily on the development activities

Processes Possible in this Model

- Traditional, straight-line waterfall
- Architecture-centric development
- DSSA-based project
- Agile development
- Dysfunctional process

Summary (1)

- A proper view of software architecture affects every aspect of the classical software engineering activities
- The requirements activity is a co-equal partner with design activities
- The design activity is enriched by techniques that exploit knowledge gained in previous product developments
- The implementation activity
  - is centered on creating a faithful implementation of the architecture
  - utilizes a variety of techniques to achieve this in a cost-effective manner

Summary (2)

- Analysis and testing activities can be focused on and guided by the architecture
- Evolution activities revolve around the product's architecture.
- An equal focus on process and product results from a proper understanding of the role of software architecture