ISR Institute for Software Research
UNIVERSITY OF CALIFORNIA, IRVINE

Primacy of Place: The Reorientation of Software Engineering Demanded by Software Architecture

Richard N. Taylor
University of California, Irvine

Acknowledgments: My key collaborators

- Nenad Medvidovic
  - University of Southern California (USC)
- Eric Dashofy
  - University of California, Irvine (UCI)

Summary

- Software Architecture is a *really* powerful concept
  - It provides the key intellectual lever for the whole of development
- It has enjoyed considerable success
  - Whether in small projects or large, commercial or F/OSS
- BUT: it is frequently undervalued and misunderstood
  - Widespread practice is in ignorance of a substantive understanding
  - We don't teach it well
- We have contributed to the problem
  - By ignoring the legitimate claims of other key system stakeholders
  - By pursuing limited research goals
- Architecture rightly belongs at the center of software engineering: because it is the concept that can tie development & evolution together
- SE must change to give architecture its proper place
  - An issue for academics and those who promulgate standards

Outline

- A few success stories: software architecture has already had a big impact
- Why there's still work to be done
  - Expanding the agenda and the scope
- Software architecture and the development process
  - Requirements, Design, Implementation, Evolution, Project management
- Thinking about what's going on, using a new visualization
- A little summary of some of the tasks facing us

Example Success #1: REST

- REpresentational State Transfer
  - An architectural style for decentralized network-based applications
- The underlying and guiding style of the post-1994 WWW
- Now finding impact in other applications
  - Other levels of the network stack; multi-media; Subversion, Web services

Google on "REST": hits 2 through 7

Building Web Services the REST Way
Brief article on REST as an architectural style. (Roger L. Costello, xfront.com)
www.xfront.com/REST-Web-Services.html

Representational State Transfer - Wikipedia, the free encyclopedia
Representational State Transfer (REST) is a software architectural style for ... REST strictly refers to a collection of architectural principles (described ...
en.wikipedia.org/wiki/Representational_State_Transfer

REST - Representational State Transfer
Roy Thomas Fielding's PhD dissertation "Architectural Styles and the Design of Network-based Software Architectures".
www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

RESTwiki
Site dedicated to all things related to the REST architectural style; includes a list of REST resources.
rest.blueoxen.net/

Second Generation Web Services
Comment on this article Does the REST model make sense, or is SOAP enabling ... Implementing REST Web Services: Best Practices and Guidelines (122 tags) ...
www.xml.com/pub/a/2002/02/06/rest.html

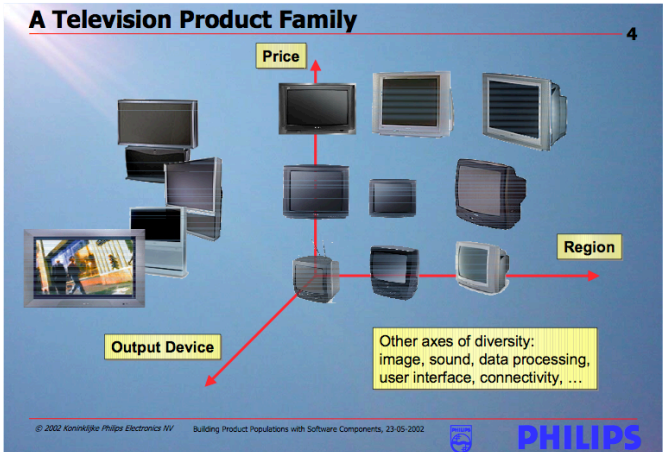
REST and the Real World
Following on from his Next Generation Web Services article, Paul Prescod shows how the REST model for web services meets real world demands such as security ...
www.xml.com/pub/a/2002/02/20/rest.html

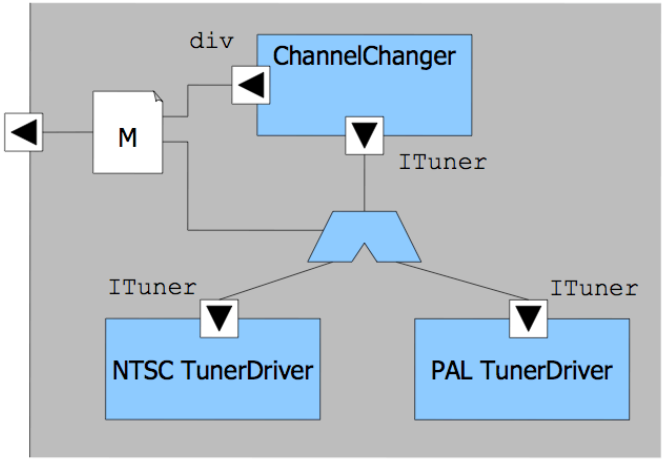
An Informal Summary of REST

- The Web is a collection of resources, each of which has a unique name known as a uniform resource locator, or “URL”.
- Each resource denotes, informally, some information. “Any information that can be named can be a resource: a document or image, a temporal service (e.g., “today’s weather in Los Angeles”), a collection of other resources, a nonvirtual object (e.g., a person), and so on.”
- URI’s can be used to determine the identity of a machine on the Internet, known as an origin server, where the value of the resource may be ascertained.
- Communication is initiated by clients, known as user agents who make requests of servers. Web browsers are common instances of user agents.
- Resources can be manipulated through their representations. For instance, a resource may be updated by a user agent sending a new representation of that resource to the origin server that holds that information. Similarly a resource may be viewed by a user agent obtaining a representation from an origin server and displaying that representation on a monitor. HTML is a very common representation language used on the Web.
- All communication between user agents and origin servers must be performed by a simple, generic protocol (HTTP), which offers the command methods GET, POST, and a few others.
- All communication between user agents and origin servers must be fully self-contained. That is, an origin server must be able to respond correctly to a user agent’s request based solely on information contained in the request, and not require maintenance of a history of interactions between the user agent and the origin server.

Example Success #2: Philips Consumer Electronics

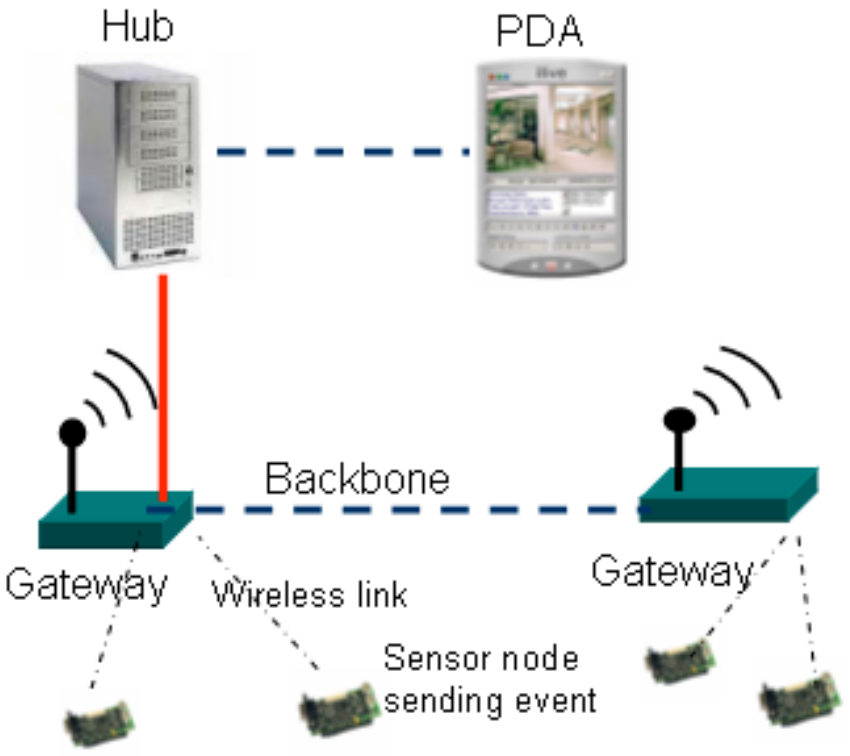
- Efficient exploitation of product families
- Key is creation, use, specialization, and evolution of common architectures
- Koala as the means of representation





Other Successes

- Bosch: Wireless, embedded, sensor applications
  - Architecture-focused middleware
- Boeing: Software-defined Radio Architectures
  - Modeling at multiple levels: network, unit, board, software
  - Hardware/software combinations



Outline

- A few success stories: architecture has already had a big impact
- Why there's still work to be done
  - Expanding the agenda and the scope
- Architecture and the development process
  - Requirements, Design, Implementation, Evolution, Project management
- Thinking about what's going on, using a new visualization
- A little summary of some of the tasks facing us

If the Idea is so Cool, what's Wrong?

- "Most of the time" "architecture" is confined to a development **phase**
- "Most of the time" "architecture" is equated with "high-level design"
- "Most of the time" "architecture" **follows** requirements
- "Most of the time" "architecture" and implementation act like divorcees
  - Still related somehow -- but mostly in legal terms
  - Often at odds with each other
  - Trying to live independent lives
  - Both think they are the most important

Rather than…

- … confined to a development **phase**
  - Is used, changed, and developed through most, if not all of a development process
- … equated with "high-level design"
  - Entails all principal design decisions
  - From multiple stakeholder perspectives
- … **following** requirements
  - Is used and developed throughout application conception
- … acting like divorcees
  - Is a key partner throughout implementation (and evolution)

Primacy of Place

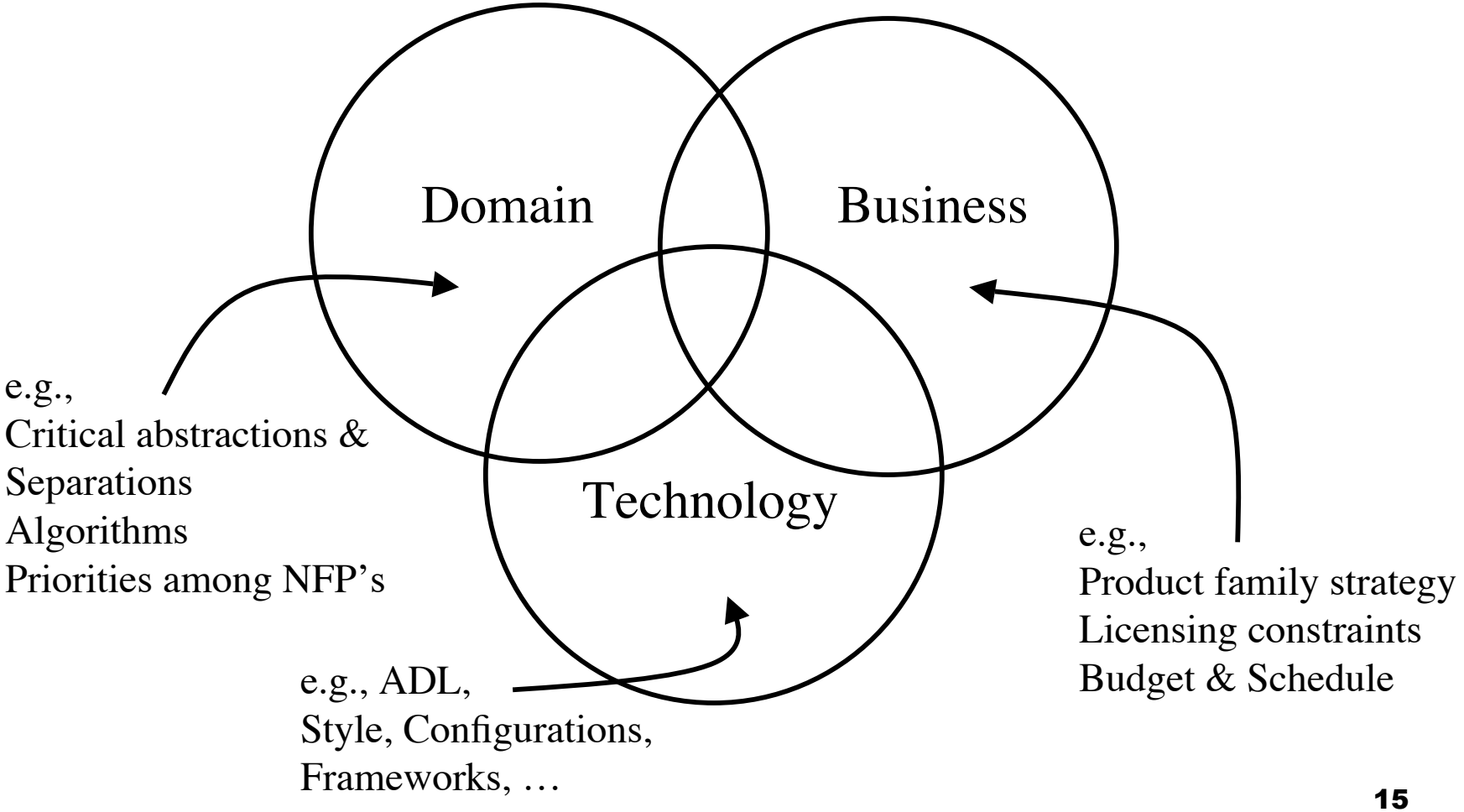
- Until architecture assumes the dominant conceptual role in software development, it cannot yield its potential benefits
- To assume that role, several things must change

> The very character of key software engineering activities, such as requirements analysis and programming, are altered and the technical approaches taken during development activities are necessarily changed.

But What is "Architecture"?

A software system's architecture is the set of *principal design decisions* about the system.

A Stakeholder-Centric Perspective on the Sources of "Principal" Design Decisions



Outline

- A few success stories: architecture has already had a big impact
- Why there's still work to be done
  - Expanding the agenda and the scope
- Architecture and the development process
  - Requirements, Design, Implementation, Evolution, Project management
- Thinking about what's going on, using a new visualization
- A little summary of some of the tasks facing us

Architecture and Development Processes

- Principal decision decisions may emerge at many points in the development process
  - E.g., choice of implementation framework (may effect whole implementation, and approach thereto); choice of specific algorithm, e.g. for signal processing applications; decision as to whether application should be developed as part of a (new or existing) product line; choice of security/trust model (what do you believe about the user community?)
- The architecture should be an accurate characterization of all representations of the application… including the deployed object code
  - It is the *foundational characterization* of the application

The Restructuring of "Requirements Engineering"

- The conventional teaching of software engineering is that RE precedes design.
- Typical practice shows that to be almost never the case
  - "The idea of completing requirements specification before doing design work is nonsense." -- *Anonymous*
- *It is not because we are idiots or badly trained engineers*
  - *But why is it so?*

Solution and structure are equal partners with requirements in a conversation about needs (1)

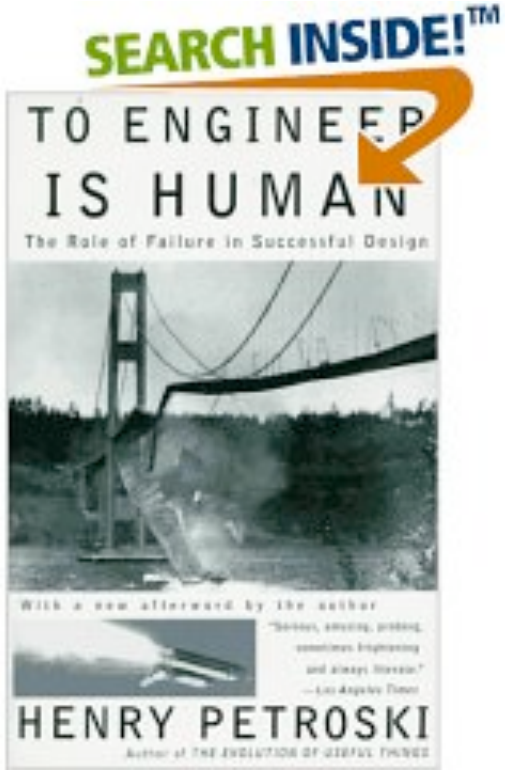
- Existing designs and architectures provide the vocabulary to talk about what might be;
- Our understanding of what works now, and how it works, affects our wants and perceived needs, typically in very solution-focused terms;

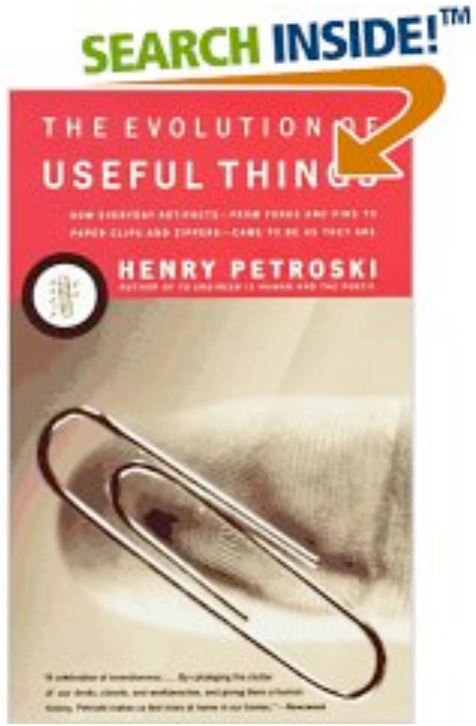
Solution and structure are equal partners with requirements in a conversation about needs (2)

- The insights from our experiences with existing systems helps us imagine what might work and enables us to assess, at a very early stage, how long we must be willing to wait for it, and how much we will need to pay for it.
- The simple conclusion then, is that analysis of requirements and consideration of design – concerning oneself with the decisions of architecture – must be pursued **cooperatively and contemporaneously.**

This is Not New in Engineering

- Henry Petroski: failure as a key engine of design innovation





The Evolution of Zippers

(No Model.)

W. L. JUDSON.
SHOE FASTENING.

No. 504,037. Patented Aug. 29, 1893.

Fig. 1.

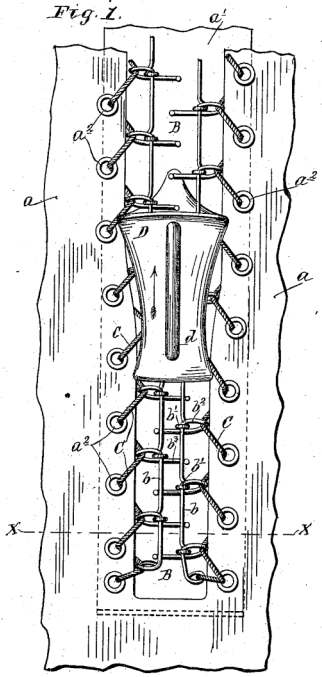


Fig. 3.

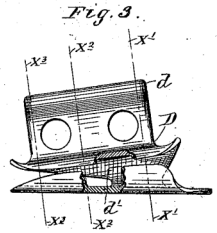


Fig. 4.

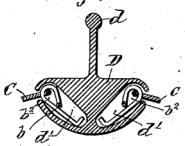


Fig. 5.

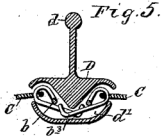


Fig. 6.

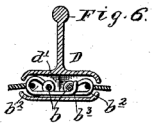


Fig. 2

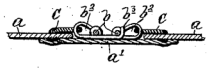


Fig. 7.

Witnesses
A. H. Upsaht.
E. F. Elmore.

Inventor
Whitcomb L. Judson
By his Attorney.

G. SUNDBACK.
SEPARABLE FASTENER.
APPLICATION FILED AUG. 27, 1914.

1,219,881. Patented Mar. 20, 1917.

Fig. 1.

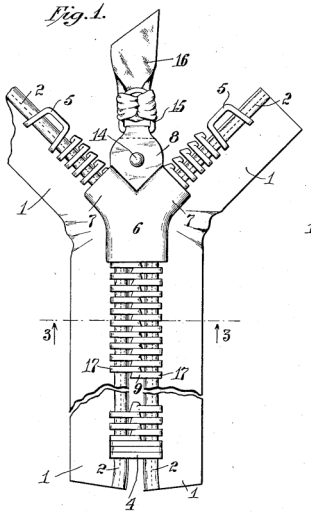


Fig. 2.

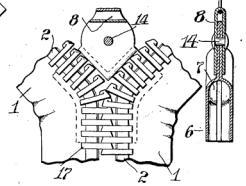


Fig. 8.

Fig. 4.

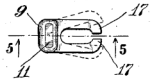


Fig. 5.

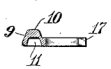


Fig. 3.

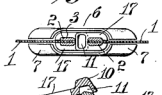


Fig. 6.

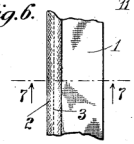
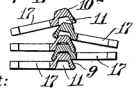


Fig. 7.

Fig. 9.



Attest:
J. D. Connelly, Jr.

Inventor:
Gideon Sundback,
by Edwards, Sager & Wooster
Attys.

This is Not New in Software Engineering

E.g.

- Twin Peaks (Nusibeh)
- Agile methods
- … but somehow we have let developers merrily carry along under the fiction of requirements engineering as first

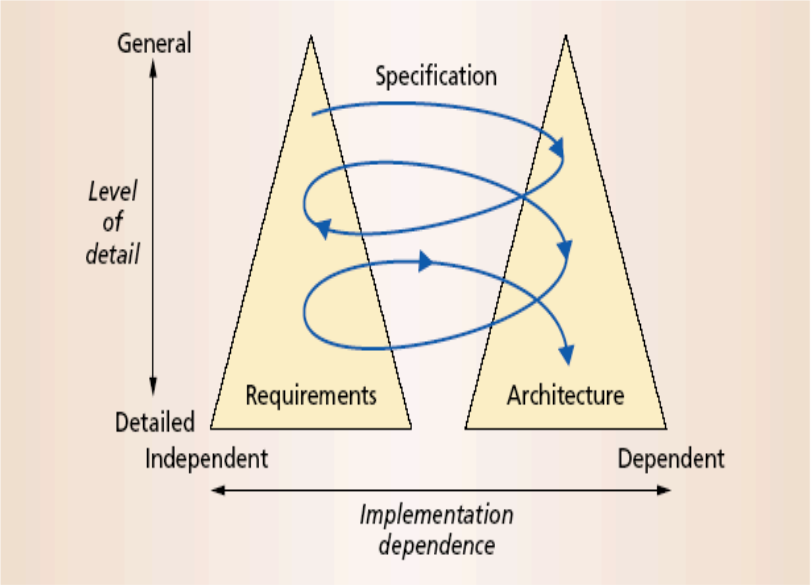


Figure 1. The Twin Peaks model develops progressively more detailed requirements and architectural specifications concurrently. This is an adaptation of the model first published in Paul Ward and Stephen Mellor's Structured Development for Real-Time Systems: Introduction and Tools, *vol. 1, Prentice Hall, Upper Saddle River, N.J., 1985, and subsequently adapted by Andrew Vickers in his student lecture notes at the University of York, UK.*

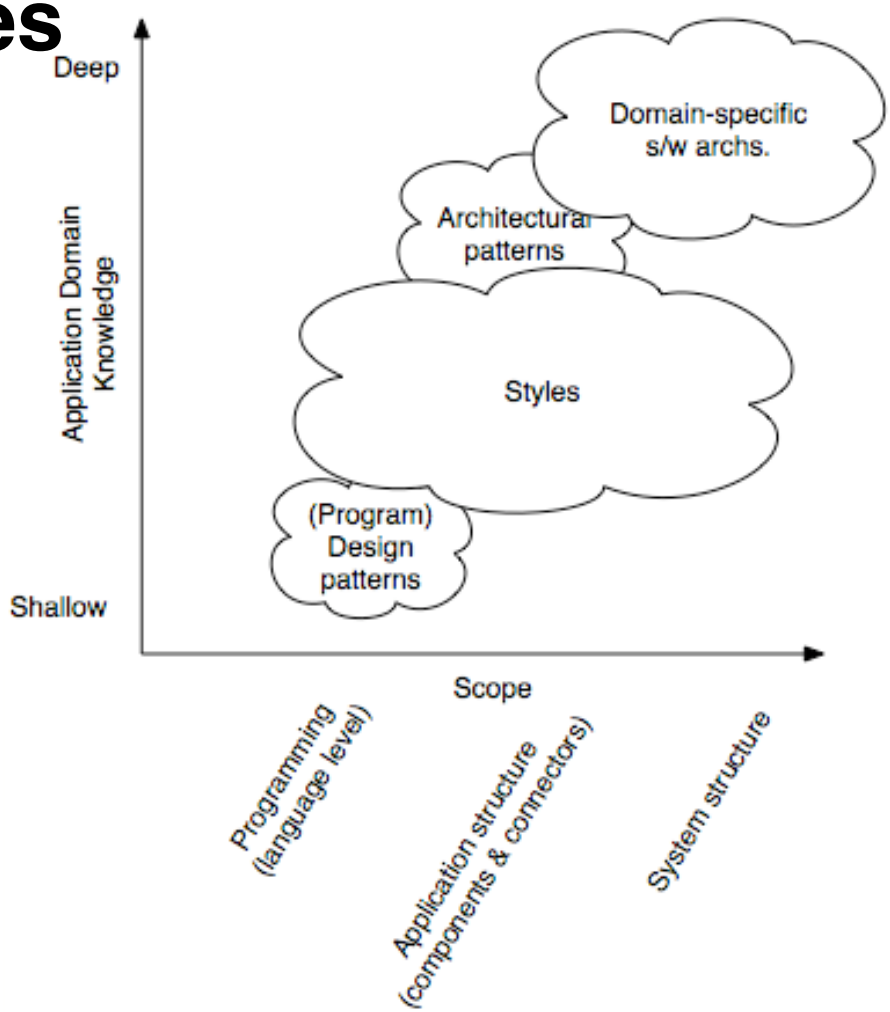
So Let's Call this Activity "Conceptualization"

- It is the joint exploration of what to build and how to build it.
- Use our full range of experience (i.e. existing architecture), intuition, and analytical abilities to make the key initial decisions about an application.
- The principal design decisions resulting from this activity are critical elements of the application's architecture

Design

- Architecture is naturally a centerpiece of the design activity
  - But designing proceeds throughout most of the lifetime of an application -- it is not a one-shot deal
- Design is a rich activity -- much richer than we usually teach it

Domain Knowledge, Scope, and Design Techniques



Design, what's different

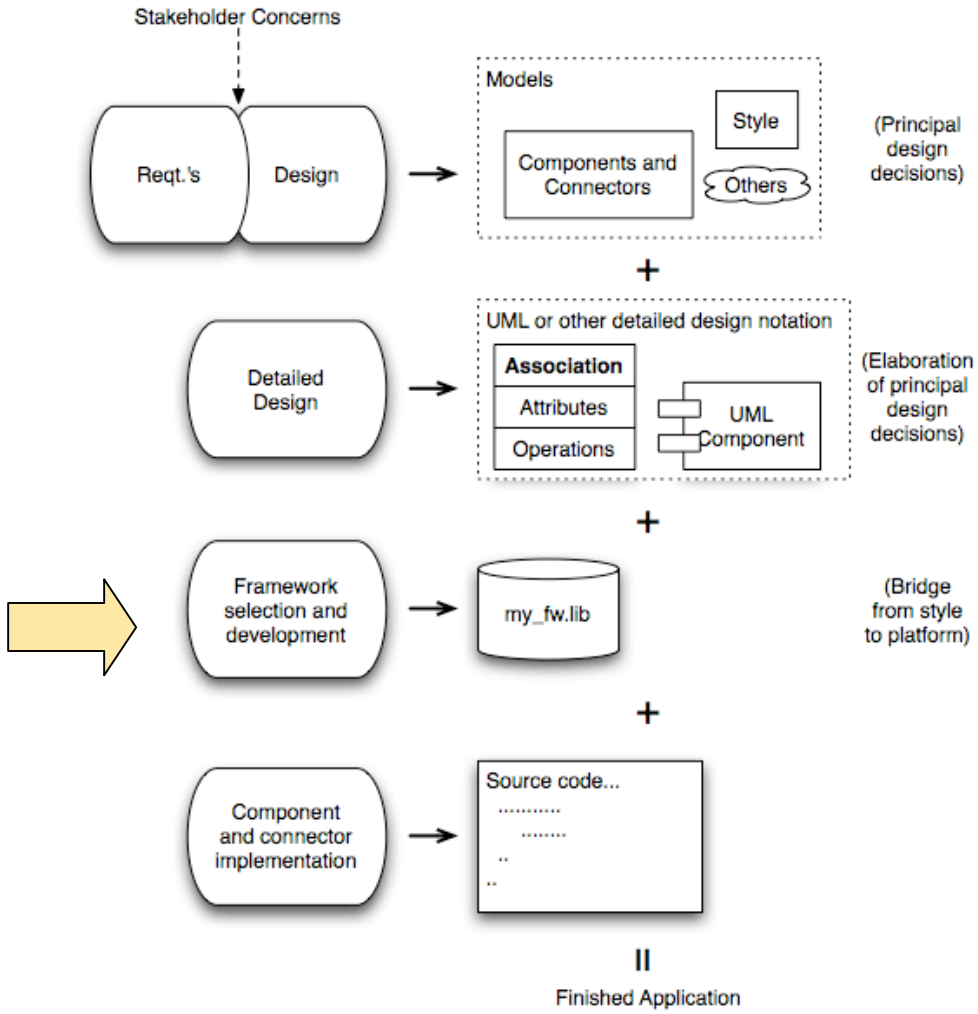
- Teaching and using a fuller range of techniques, experience, intuition, and analytical abilities to make key choices
- Addressing design more holistically: all stakeholder perspectives
- Recognizing that designing is a pervasive activity

Implementation: What's the Role of Architecture Here?

- Goal: create, or co-create, an implementation faithful to the architecture (as it exists at the time implementation begins)
  - The principal design decisions must not be violated
  - The structural architecture should provide prime guidance
- Architecture provides the basis for the use of:
  - Generative techniques
  - Reuse-based techniques, including F/OSS
  - Implementation frameworks
  - … and last of all, new source code
- The architecture is the context for evaluating implementation alternatives
  - E.g. what are the consequences of reusing a package that doesn't quite fit the existing architecture?

What We Need to Do is Help Bridge the Gap

Frameworks: technical aids in implementing the architectural styles of the system's architecture



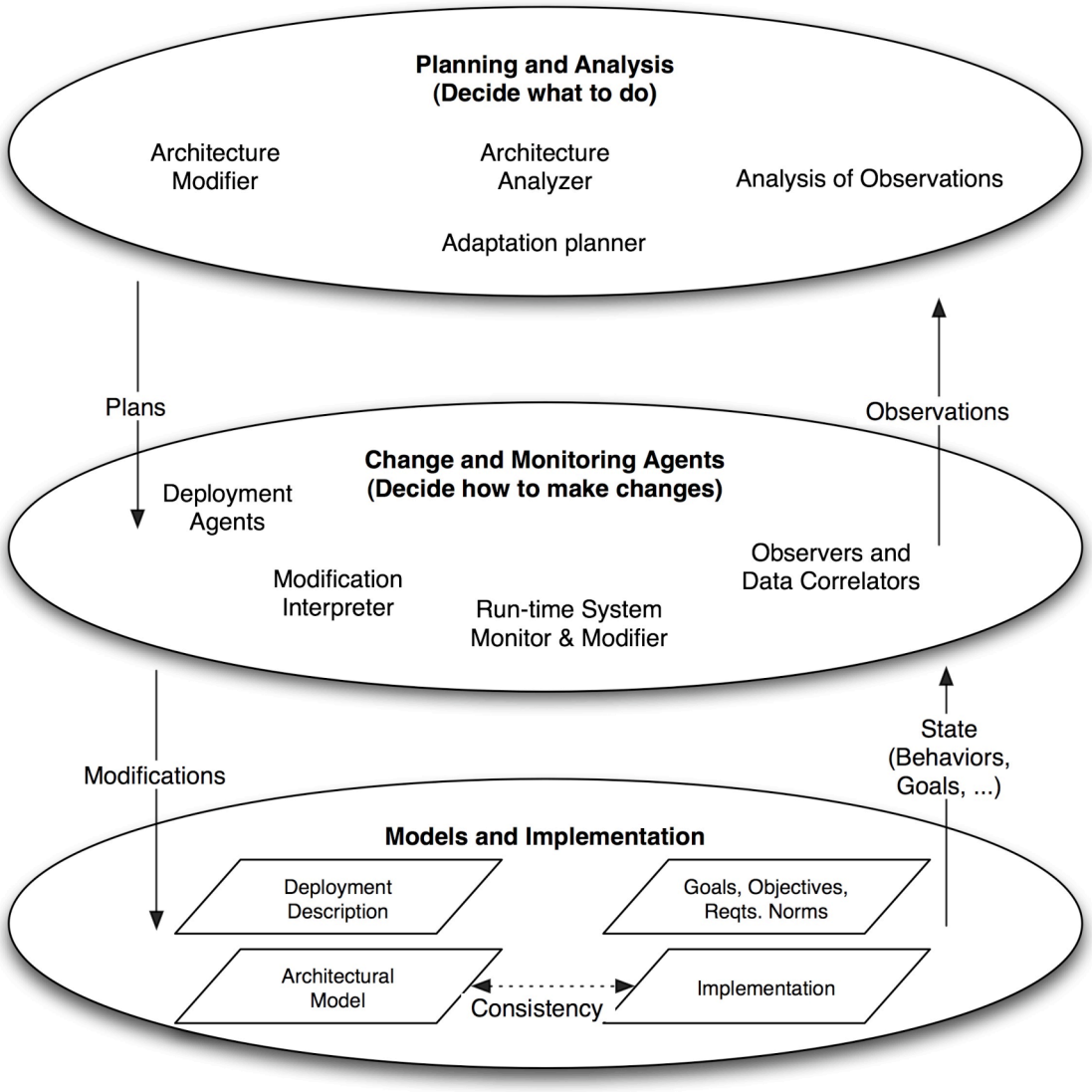
Instead of Divorcees, We need a Happily Married, Mature Couple

- The design provides useful, substantive, invaluable guidance for the implementation.
- The implementation faithfully implements the design.
- Changes to either one are reflected accurately in the other.

Analysis and Testing: Better, Faster, and Cheaper

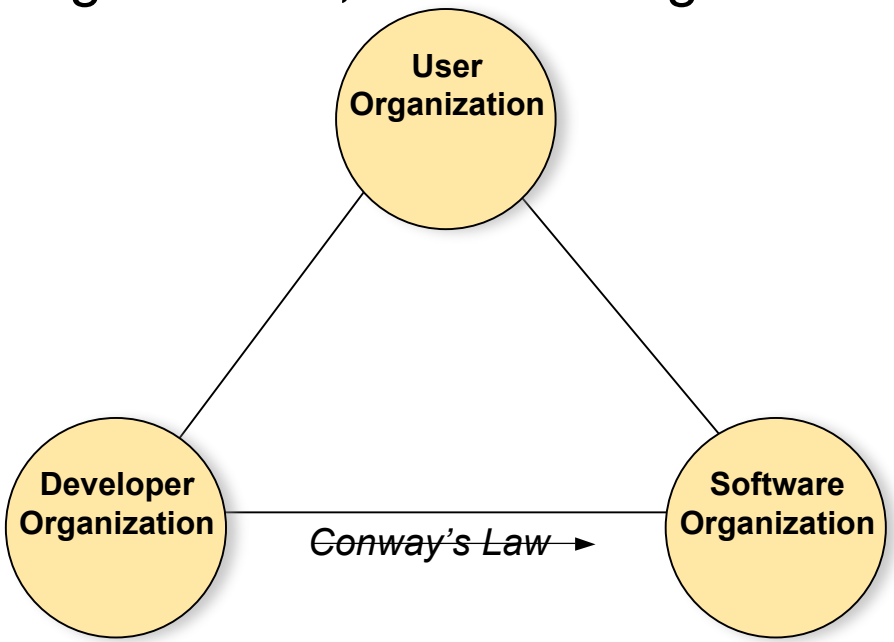
- Explicit architecture implies opportunity for analysis
- Consistency checking: with requirements and with the implementation
  - Or with a model derived from the implementation
- Prioritization of A&T activities: core elements of product families, e.g.
- Carry-forward of prior A&T work: architecture provides the context for determining what needs to be done … or re-done

Evolution


Planning and Analysis
(Decide what to do)
Architecture Modifier
Architecture Analyzer
Analysis of Observations
Adaptation planner
Plans
Observations
Change and Monitoring Agents
(Decide how to make changes)
Deployment Agents
Observers and Data Correlators
Modification Interpreter
Run-time System Monitor & Modifier
Modifications
State (Behaviors, Goals, ...)
Models and Implementation
Deployment Description
Goals, Objectives, Reqts. Norms
Architectural Model
Consistency
Implementation

Corporate Structure and Management

- Product line development requires investment in, and management of, the product line, not just individual products
- Defeating Conway's Law: Alignment of system architecture, development organization, and user organizations



The Interactions Between the Activities: Principal Design Decisions Can be Made in Many Contexts

For instance:

- Conceptualization: the concurrent activities of design and requirements
- Design and Implementation
- Analysis, Re-design, Re-implementation
- User interface design, implementation, and field trial

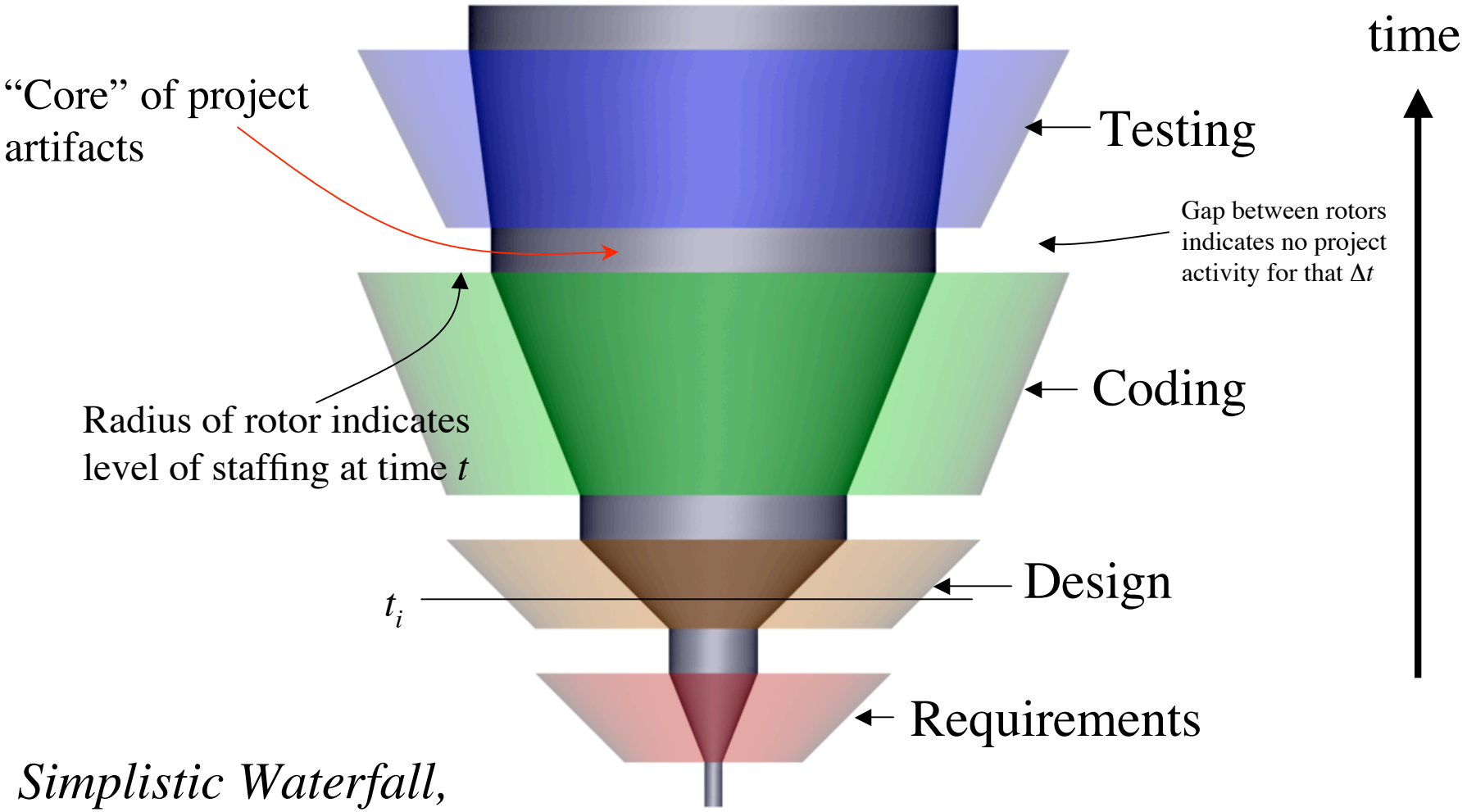
Outline

- A few success stories: architecture has already had a big impact
- Why there's still work to be done
  - Expanding the agenda and the scope
- Architecture and the development process
  - Requirements, Design, Implementation, Evolution, Project management
- Thinking about what's going on, using a new visualization
- A little summary of some of the tasks facing us

A New Visualization/Illustration Model

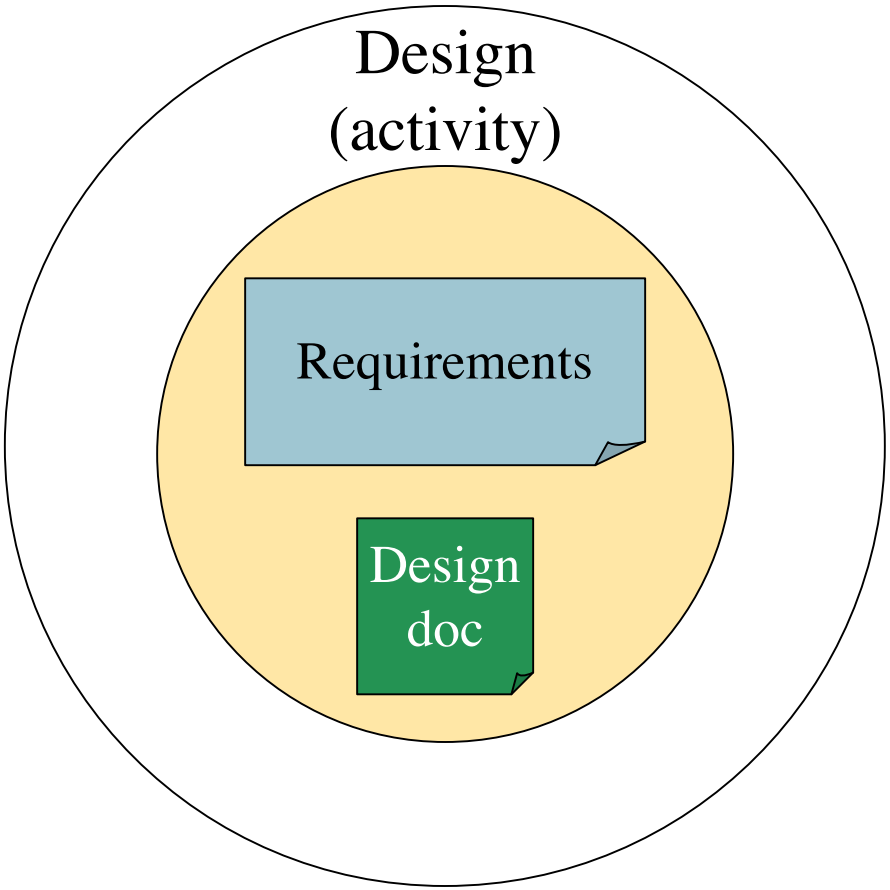
- Goals of the visualization
  - Provide an intuitive sense of
    - Project activities at any given time
      - Including concurrency of types of development activities
    - The “information space” of the project
  - Show centrality of the products
    - (Hopefully) Growing body of artifacts
    - 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

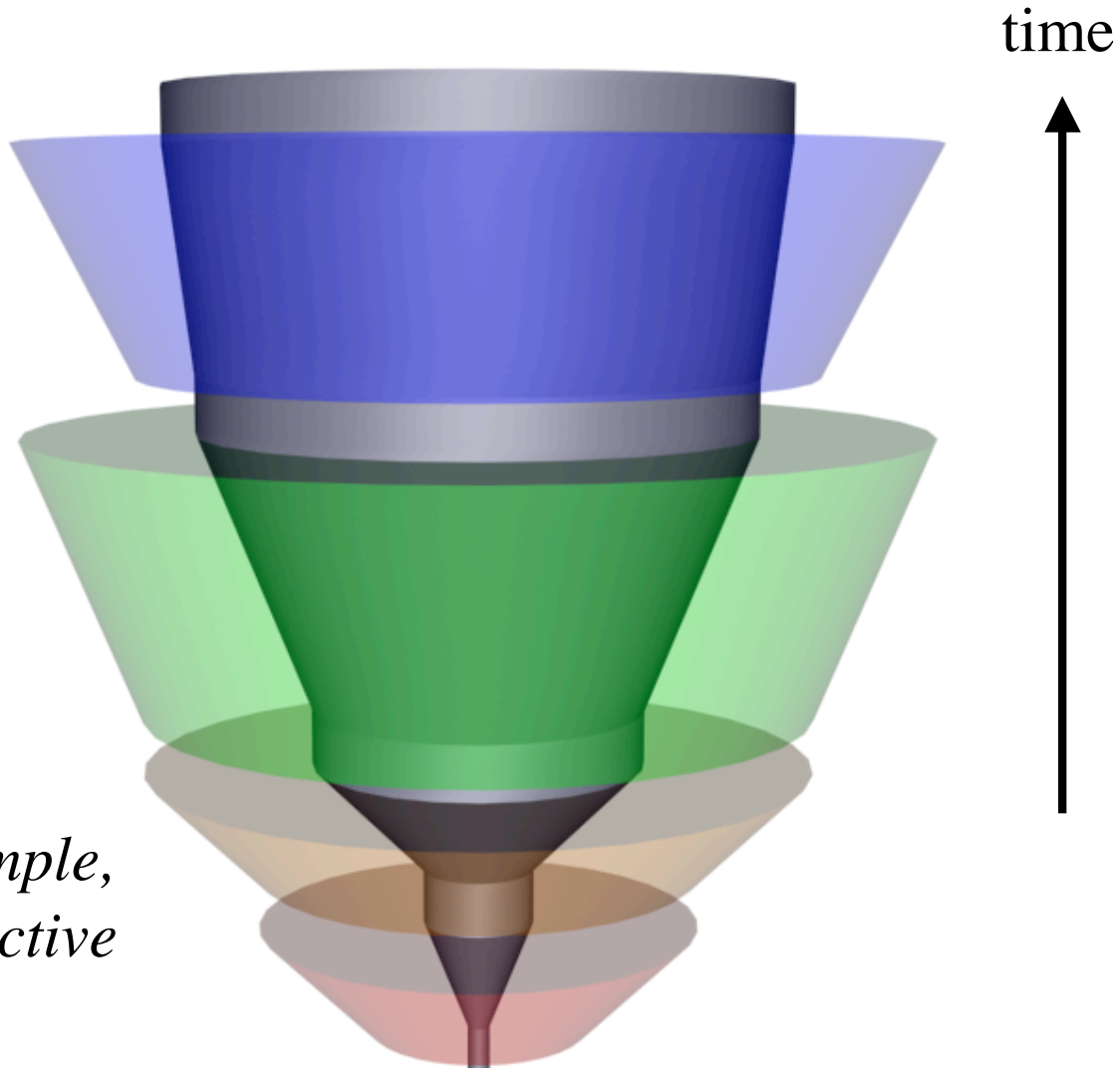


Simplistic Waterfall, Side perspective

Cross-section at time t_i

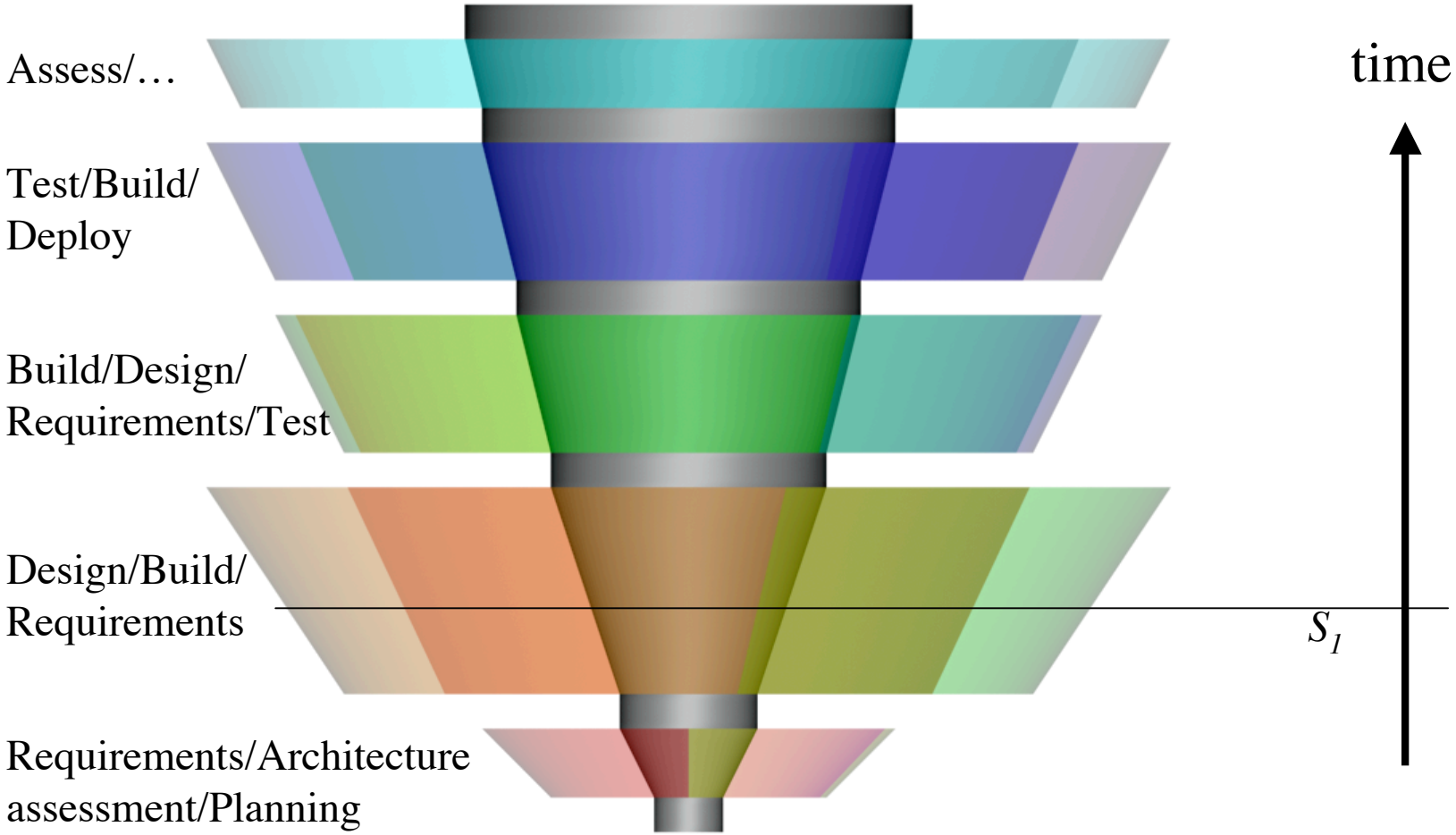


The Turbine Model

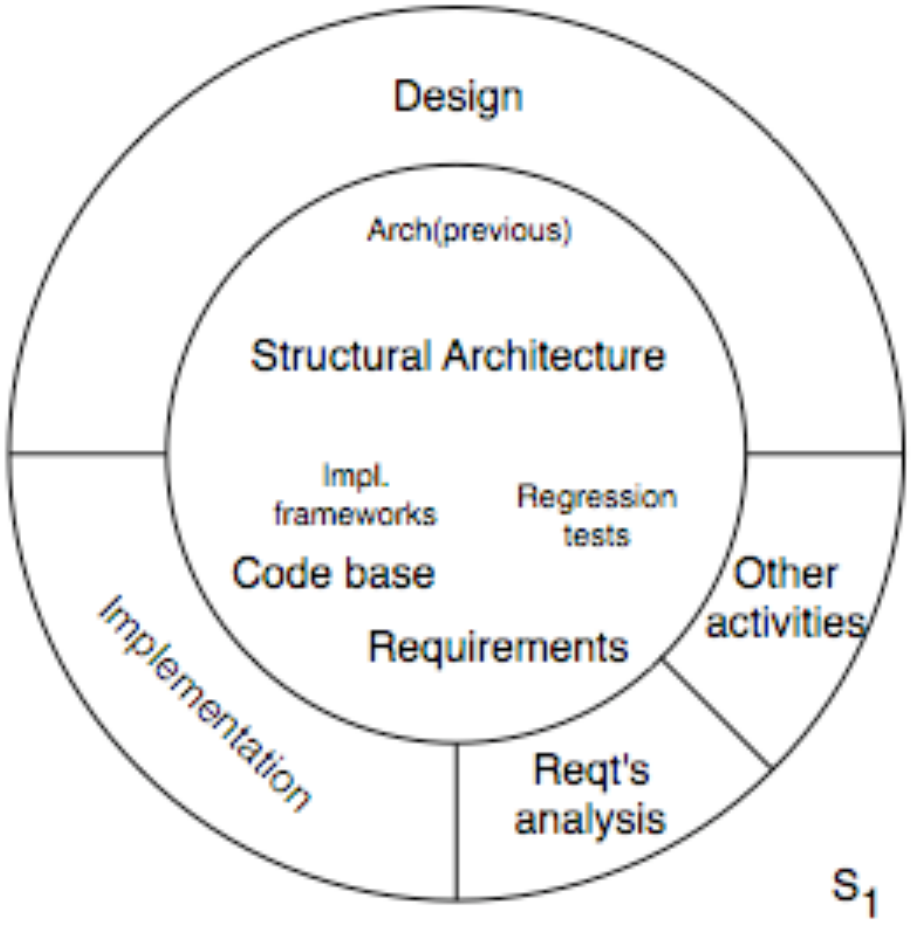


Waterfall example,
Angled perspective

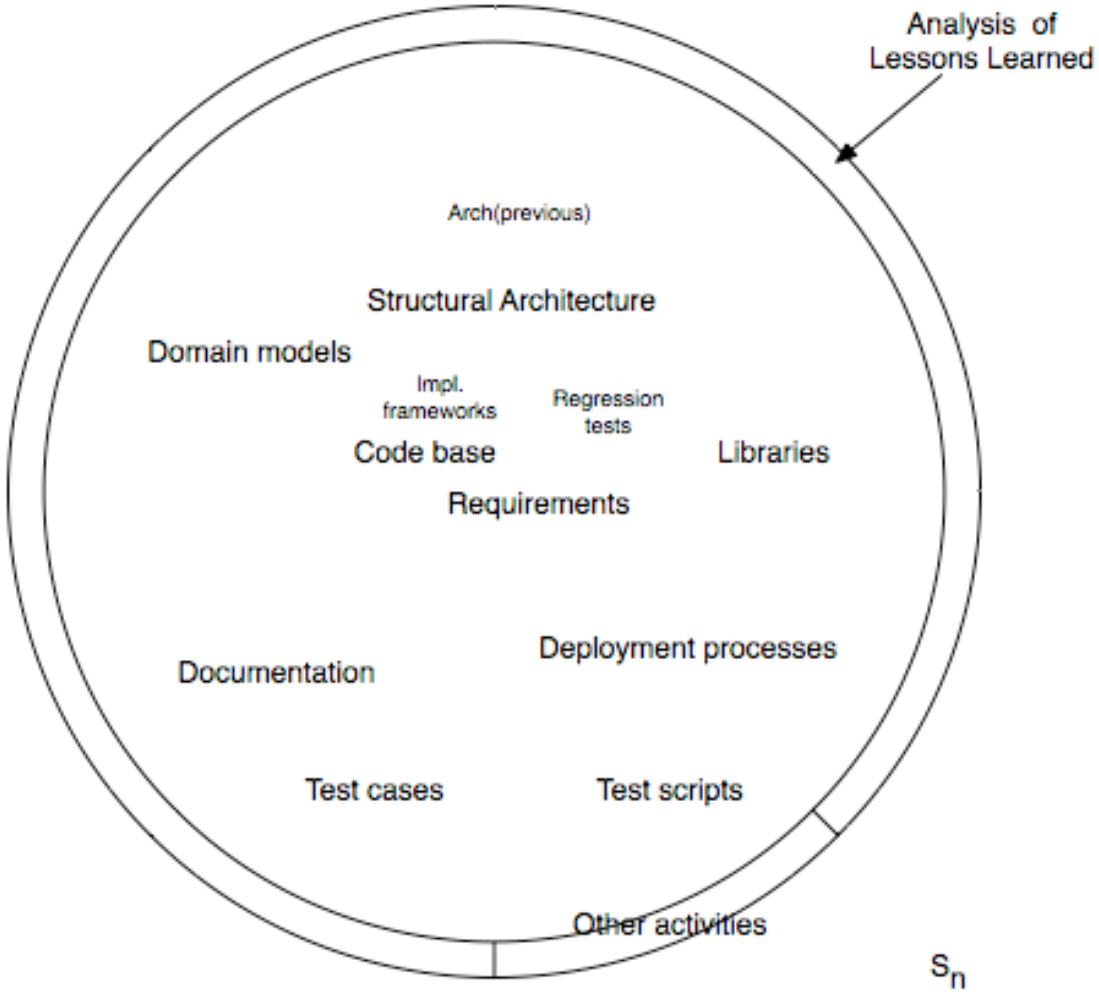
A Richer Example



A Sample Cross-Section



A Cross-Section at Project End



Volume Indicates Where Time was Spent

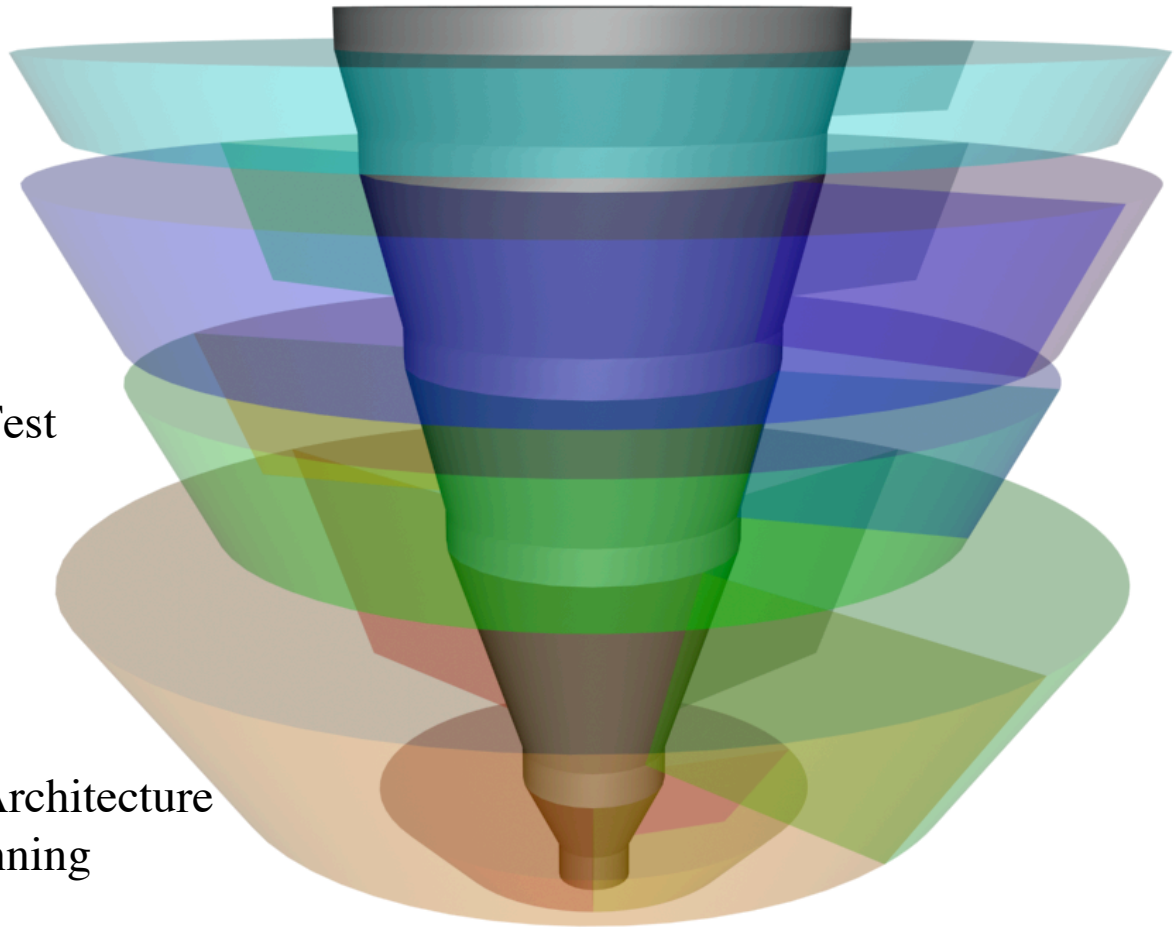
Assess/…

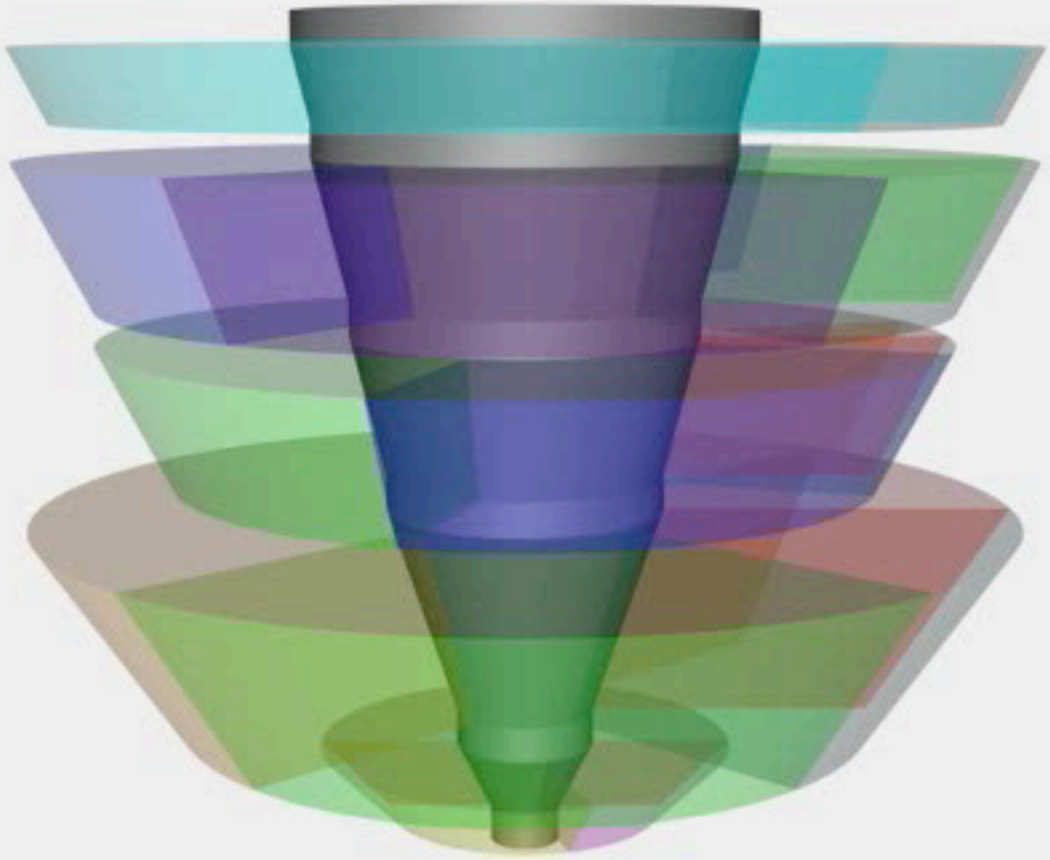
Test/Build/
Deploy

Build/Design/
Requirements/Test

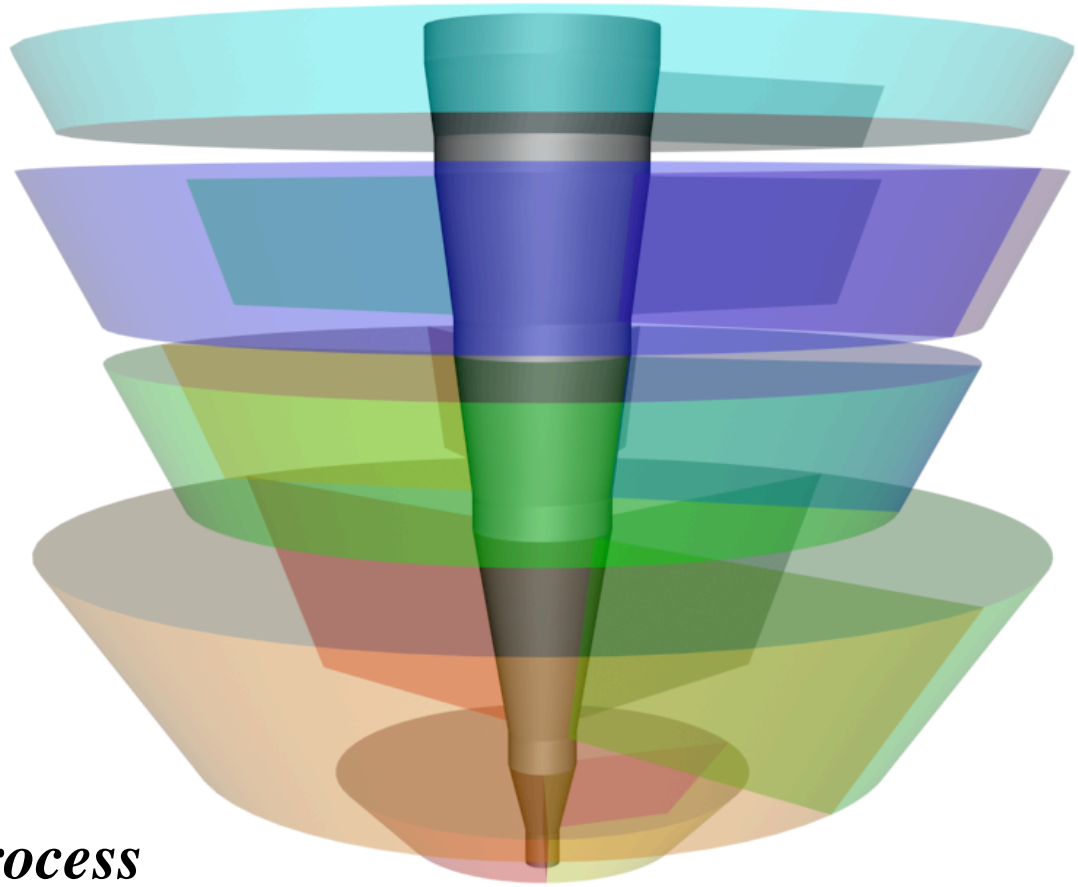
Design/Build/
Requirements

Requirements/Architecture
assessment/Planning



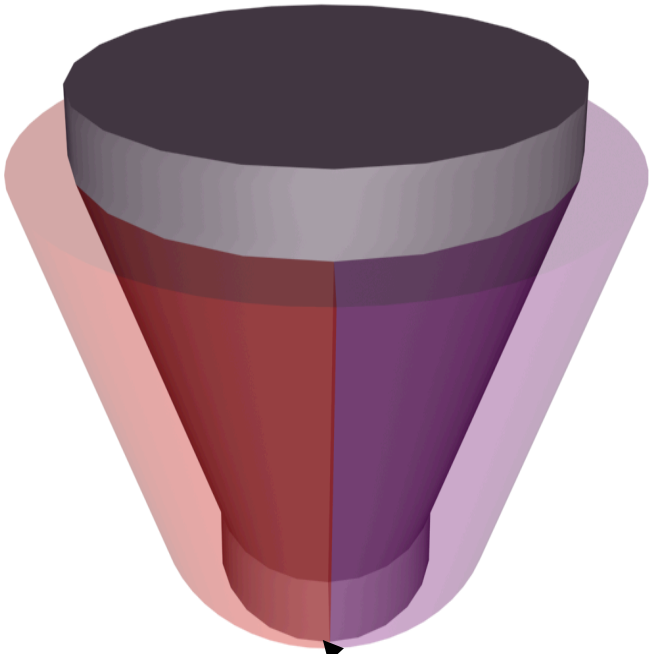


Cool. But what does this have to do with architecture-centric development?



An "Agile" Process

What happens during the follow-on project?

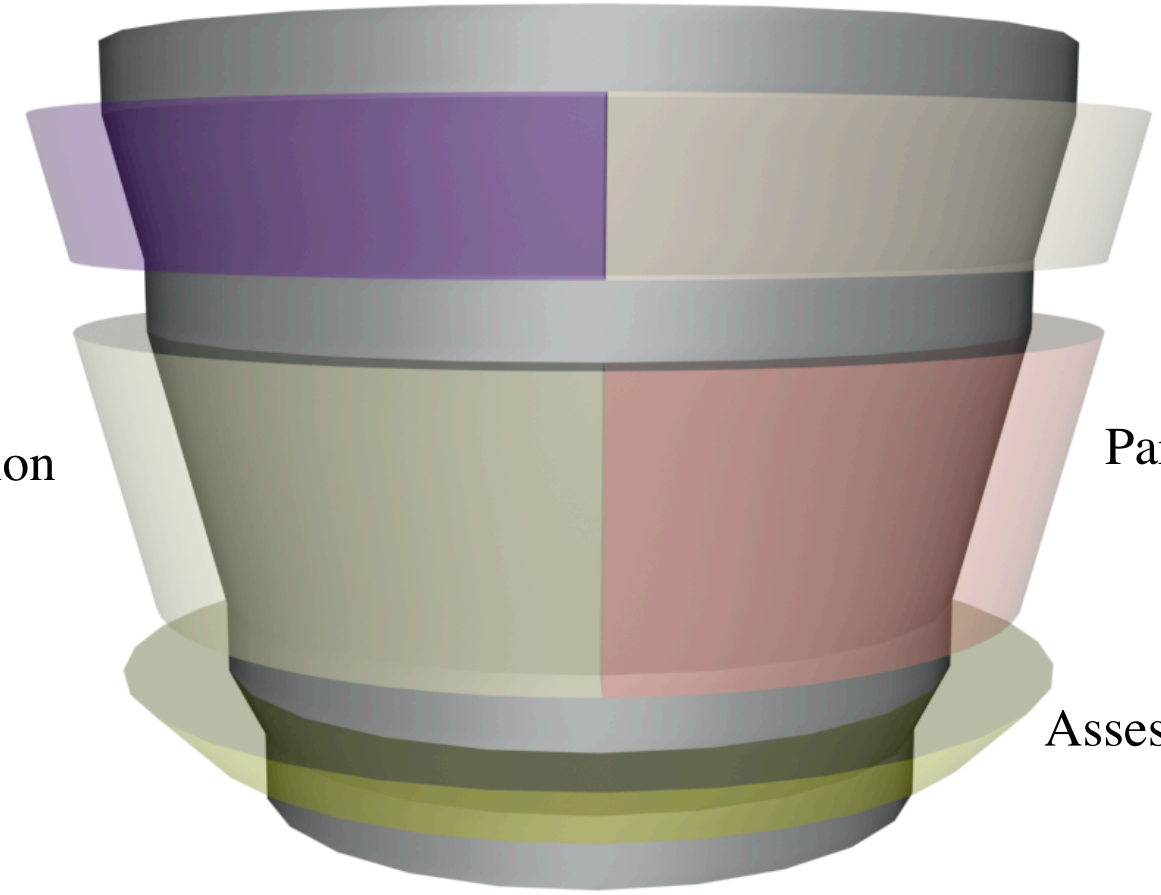


50% architecture recovery

50% RA and assessment

The project artifacts produced by the previous project

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)
  - Presentation here focused primarily on the development activities

Summary

- Software Architecture is a *really* powerful concept
  - It provides the key intellectual lever for the whole of development
- It has enjoyed considerable success
  - Whether in small projects or large, commercial or F/OSS
- BUT: it is largely undervalued and misunderstood
  - Widespread practice is in ignorance of a substantive understanding
- We have contributed to the problem
  - By ignoring the legitimate claims of other key system stakeholders
  - By pursuing limited research goals
- Architecture rightly belongs at the center of software engineering: because it is the concept that can tie it all together
- Development practice must change to give architecture its proper place
  - An issue for academics and those who promulgate standards

Lots of Opportunities for Architecture Researchers

For instance:

- ADLs: extensible to capture stakeholder concerns (technology, domain, business)
  - In general, how can we capture all principal design decisions?
- Consistency: internal and between representations
  - Allow inconsistency
  - But always enable checking of same… between all representations
- Own up to the role of architecture in all other aspects of development… and support them
  - Especially implementation
  - Traceability

Shameless Promotion

Software Architecture: *Foundations, Theory, and Practice*

Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy

To appear: 2007
(I hope)