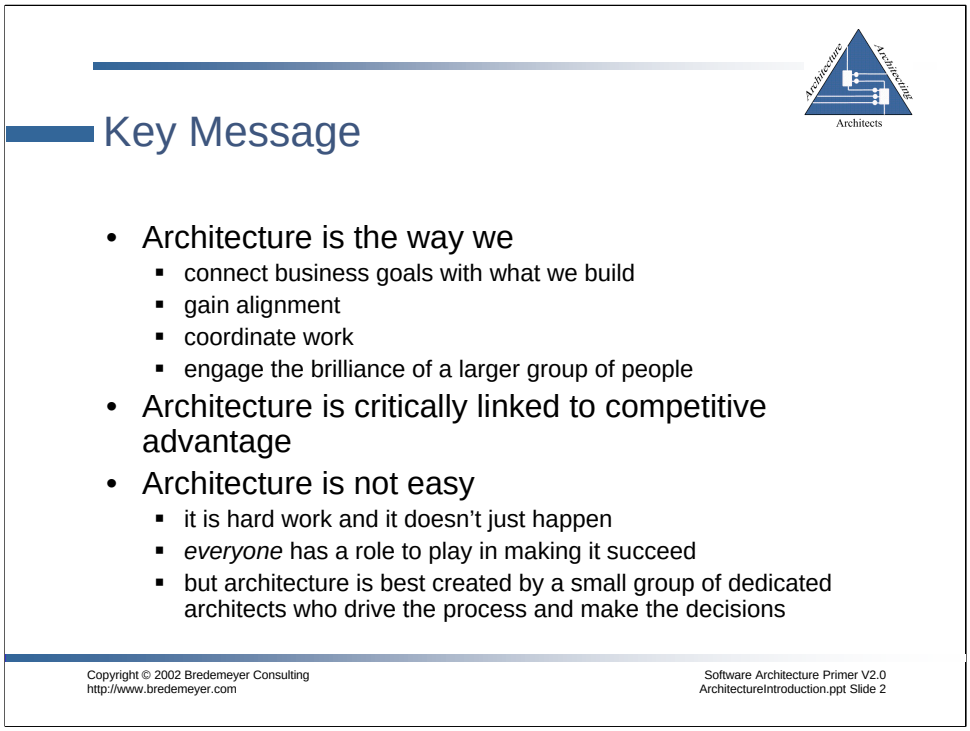
Introduction to Software Architecture

Why? What? How? Where? and Who?

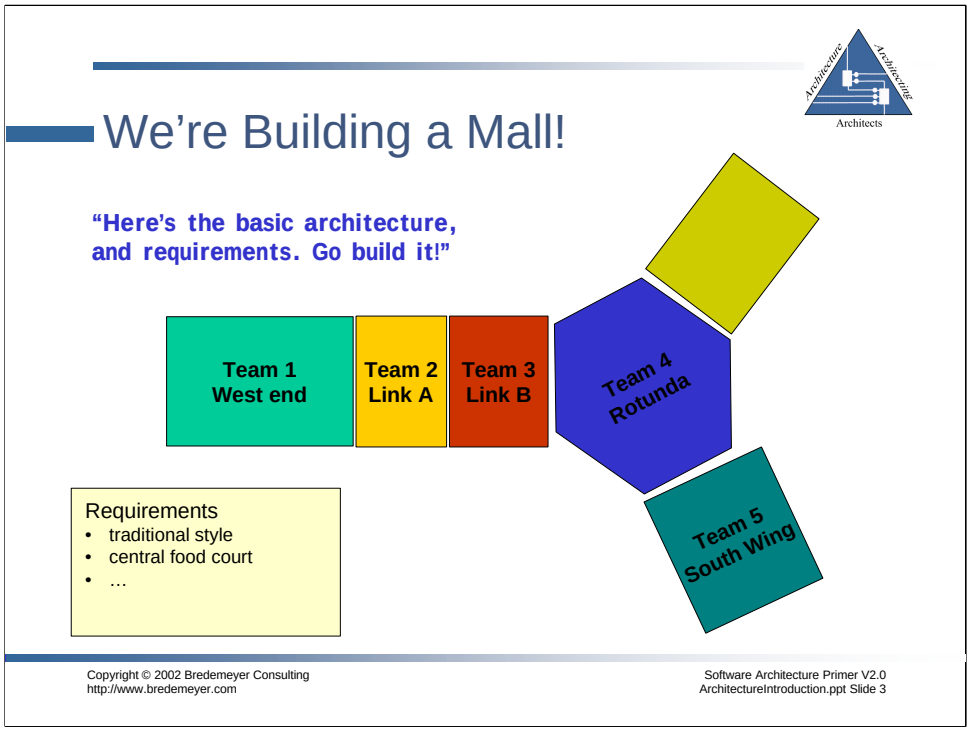
Dana Bredemeyer
Bredemeyer Consulting
Tel: (812) 335-1653
Fax: (812) 335-1652
Email: dana@bredemeyer.com
Web: http://www.bredemeyer.com

Acknowledgments

This workshop was created by Dana Bredemeyer and Ruth Malan at Bredemeyer Consulting. It has benefited from all our interactions with architects over the past several years. We would especially like to thank Bill Baddley, Bill Branson, Derek Coleman, Bill Crandall, Guy Cox, Dave Dikel, Martin Griss, Jon Gustafson, Mark Interrante, Stan Letarte, Reed Letsinger, Holt Mebane, Mike Ogush, Rob Seliger, Mark Simos, Joe Sventek, and Dean Thompson.



Architecture is inherently strategic. It should enable the achievement of the business strategy, and be the technical implementation of business strategy.



Imagine This

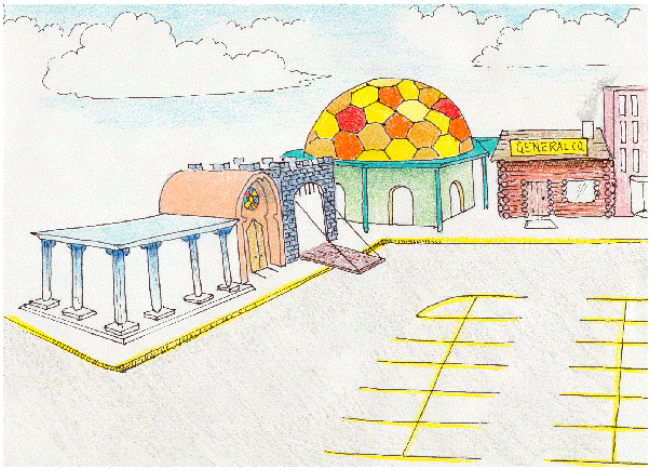
Simon Malls is building a new mall. A set of strategic guidelines have been made by the senior executive team. Now it is up to the development organization to build the mall. They want to get it done as quickly as possible, so they have several hundred builders ready to work on the site. They divide up the builders into teams. They get a good builder to outline the essential parts of the building. They assign these parts of the building to the teams, and off they go.

What do they get???

But we don't ship models…

In the software business, we ship code, not models, so we have a high discomfort with models. But if we look at the building industry, they "ship" buildings, not drawings and yet you would not begin to think of building complex physical structures without drawings.

We've Built Your Mall!



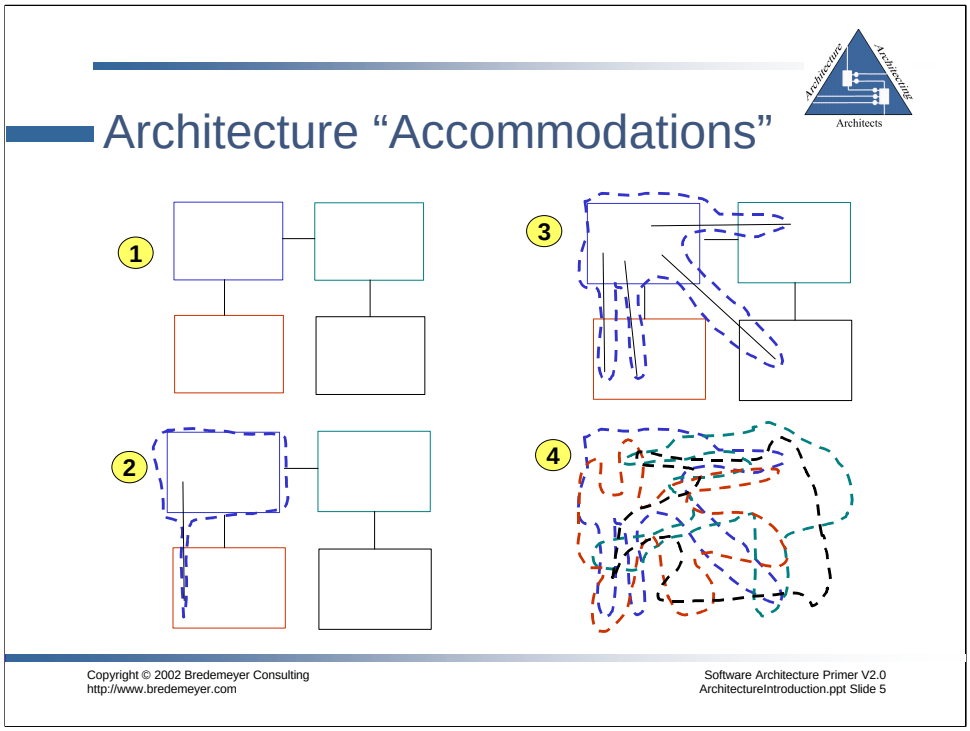
Building Architecture Analogy

When we consider complex physical buildings, it is immediately apparent that we could not do without architecture. The result is ***at best*** baroque. Along with the clumsiness, inelegance and discord in the structures, we get

- duplication, redundancy, wasted effort, rework
- gaps
- poor integration, inconsistency, mismatch

A mall is built along the ground, typically. We simply could not imagine building a skyscraper without an architecture! Now there is not just the need for integrity of the design, consistency of assumptions, and integration among the parts. Even for the inexpert, other considerations loom large:

The sequence of work has to be carefully planned. Also, structural qualities have to be ***designed***--including the building's ability to bear load, it's behavior under high-winds, the ability to move people as well as bulky heavy equipment into the building. All these normal conditions, and unusual conditions like fire, earthquake, and terrorist attacks, have to be taken into account. If they are not explicitly taken into account in creating the architecture, it is left purely to a matter of luck and who can afford this?

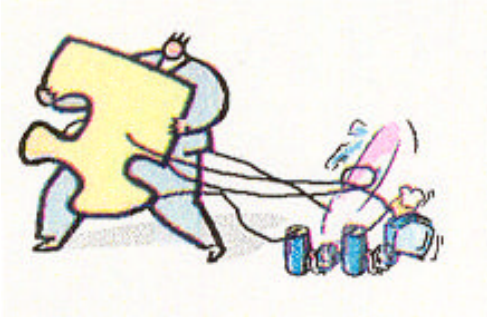


How we get to monolithic, spaghetti systems

Architectural drift begins with the first decision to short-cut the architecture, under the guise of better satisfying requirements or simply out of ignorance. As more and more decisions are made to make this or that accommodation, the coupling increases to the point where it is hard to isolate components.

“Accommodations are like rebar in concrete--the coupling turns the system into a unit”

“We typically get system devolution, not system evolution!”



The Software Problem

The typical software system is a kludge, resulting in

- unpredictability
  - in system behavior
  - in initial development and future release schedules

- poor quality
  - hard to find defects
  - hard to fix problems
  - fixes introduce new problems

- increasingly difficult to change
  - harder to respond to market shifts
  - costs more, takes longer

- hard to reuse
  - hard to isolate chunks to reuse
  - chunks are highly tuned to specific product
- morale problems

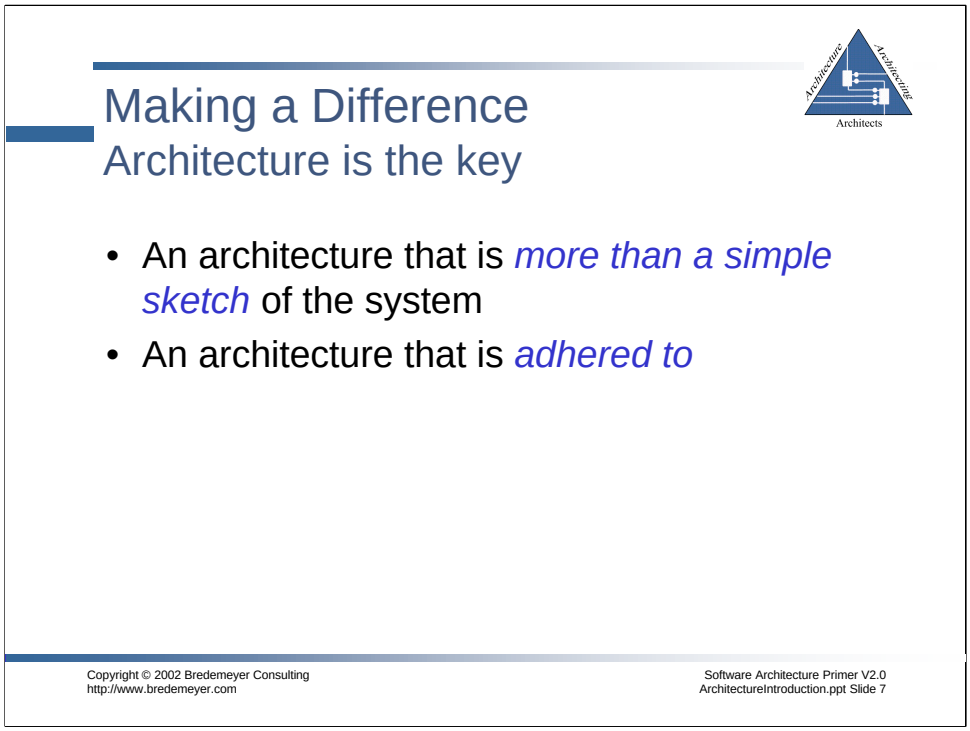
- losing ground in the market
  - competition can do more with less
  - not responsive

"Kludge"

Without architecture, we get a kludge--a word that is defined in Websters as:

1. A system, *especially a computer system*, that is constituted of poorly matched elements or of elements originally intended for other applications.
2. A clumsy or inelegant solution to a problem.

Escher's Gravitation

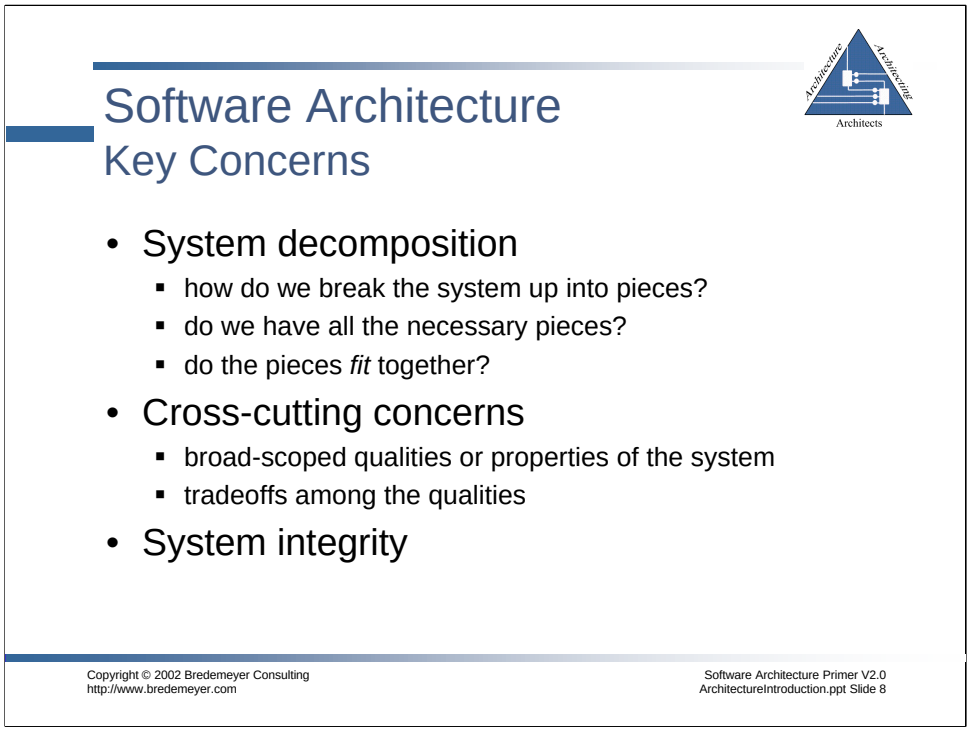


Architecture: More than a Box and Line Sketch

Just as a building's architecture is more than a sketch of the floor plan, a system's architecture is more than a box and line drawing. Just like a floor plan, the box and line drawing serves a purpose, but cannot be the extent of the architecture if you want to address cross-cutting concerns and provide a blueprint that developers can work from.

The Culture has to Change, for Architectures to "Stick"

If the architecture (no matter how good) is treated as an initial guideline for team formation, and not much more, then we will be in the same mess we're in when we don't invest in creating the architecture--or worse, because we will have only heightened the sense of failure and cynicism associated with other attempted architecture and reuse initiatives.

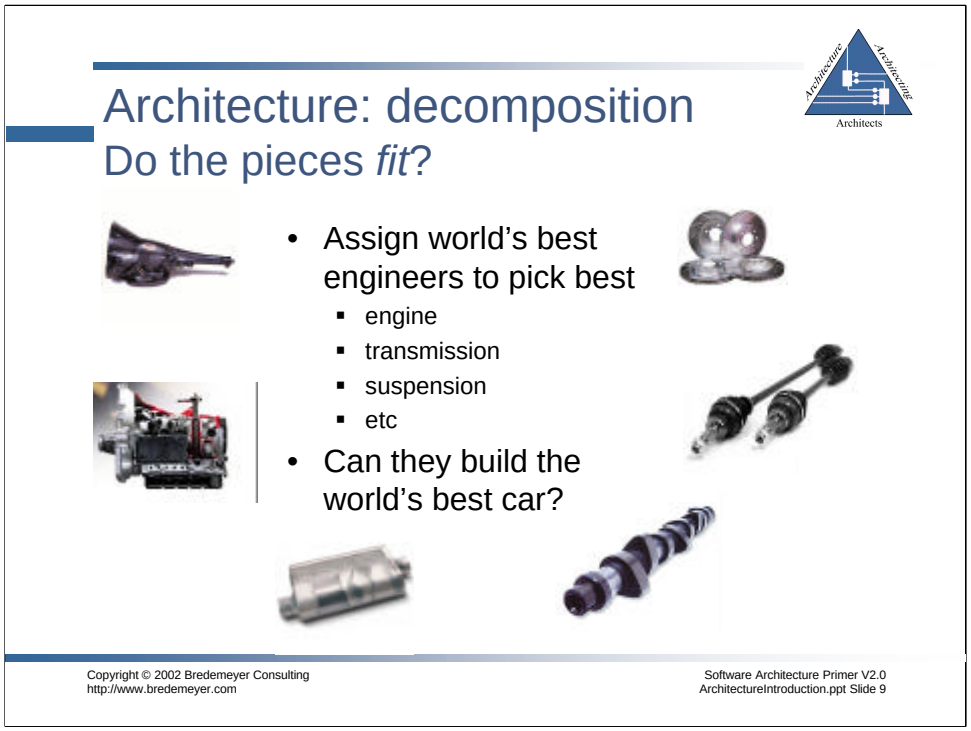


System Decomposition

In order to deal with complexity, we need to “divide and conquer”. We do this to get teams working in parallel, on the one hand, and to make the problem more intellectually tractable on the other. But when we break the system down into pieces, we need to have some confidence that the pieces can be assembled into a system.

Cross-Cutting Concerns

We refer to broad-scoped qualities or properties of the system as *cross-cutting concerns*, because their impact is diffuse or systemic. It may be a matter of preferring not to isolate these concerns because the decomposition is being driven by other concerns, or it may be that no matter how you might “slice-and-dice” the system, multiple parts are going to have to collaborate to address these cross-cutting concerns. At any rate, to effectively address cross-cutting concerns, they must be approached first at a more broad-scoped level. Many system qualities (also known as non-functional requirements or service-level agreements) are of this nature. They include performance, security and interoperability requirements. To make the picture more complicated, the system qualities often conflict, so that trade-offs have to be made among alternative solutions, taking into account the relative priorities of the system qualities.

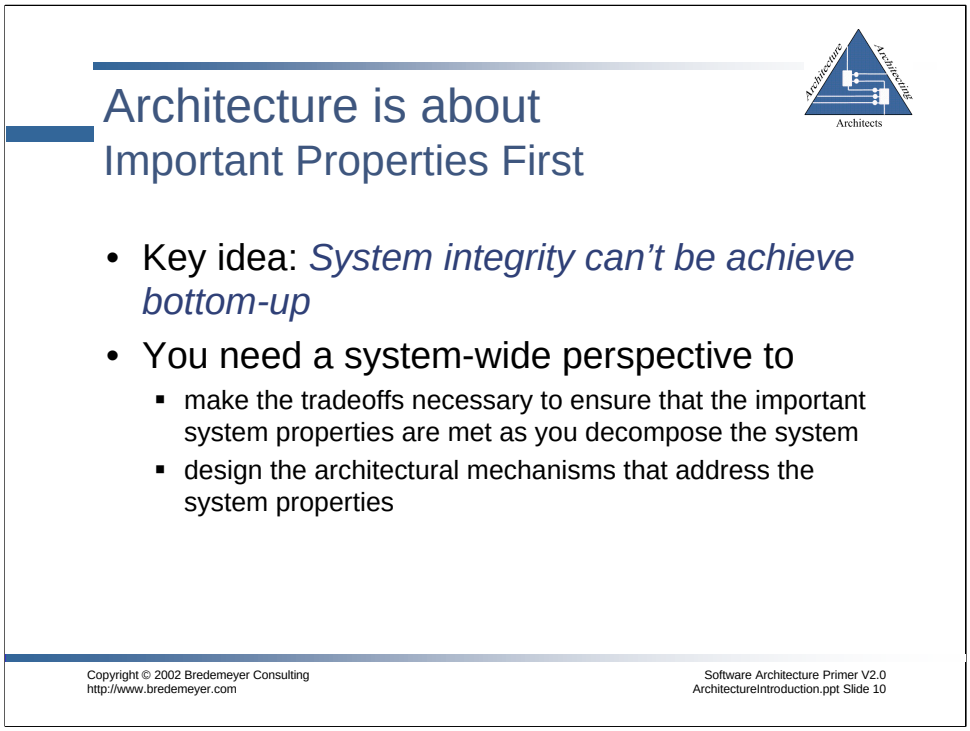


But Decomposition is Not the Only Concern

To see why consider:

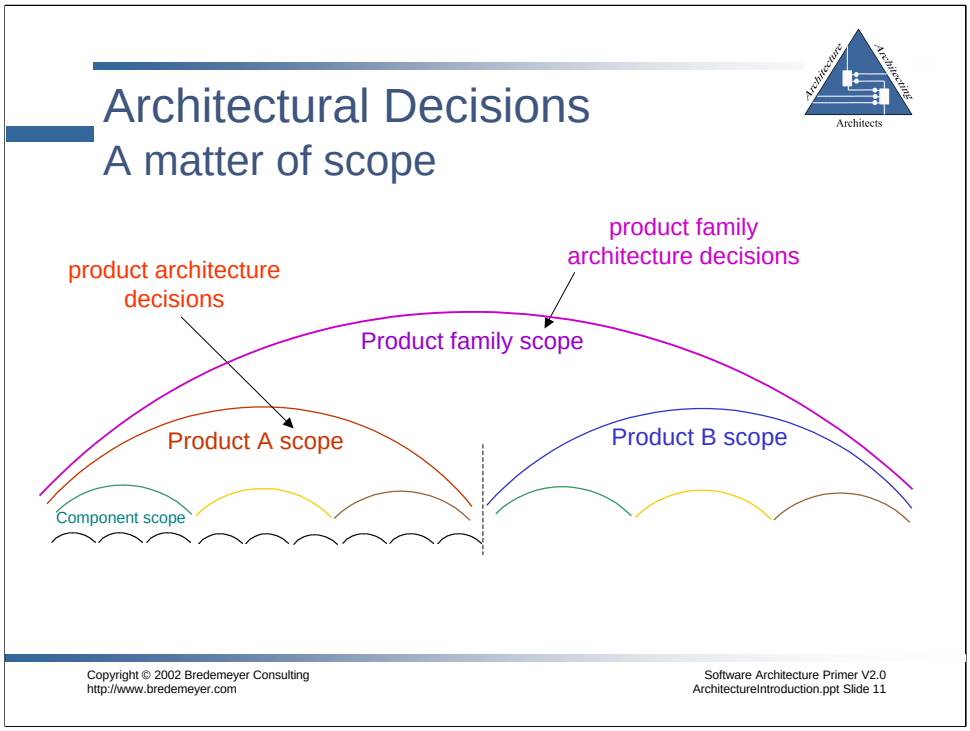
Russ Ackoff's Car Analogy

Collect together a team of the best automotive design engineers in the world. Assign them the task of selecting the best car component of each type. Will they be able to create the world's best car from these components? No! Even if they could all plug together, they are designed to optimize with respect to different sets of top priority system qualities.



Important Properties First

The ***way*** you decompose the system can make a big difference to whether or not important system properties can be met. For this reason, system properties (otherwise known as qualities or non-functional requirements) must be addressed at the architectural level. If they are not, in many cases it will be impossible, or very difficult, to meet them.

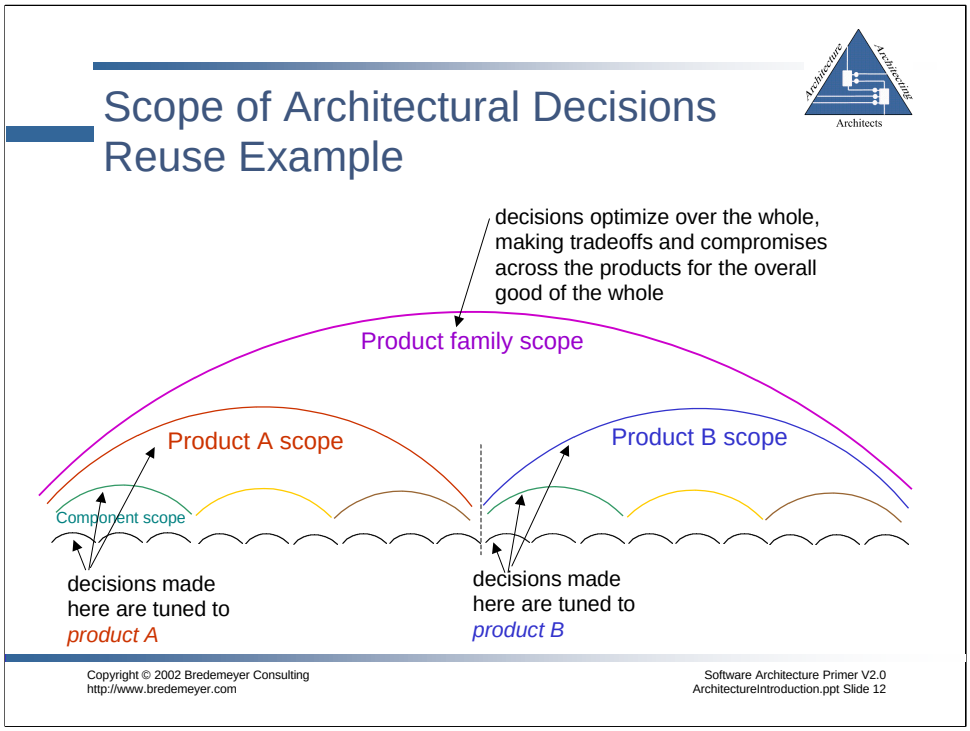


Architectural Decisions

A distinctive characteristic of architectural decisions is that they need to be made from a broad-scoped or system perspective. Any decision that could be made from a more narrowly-scoped, local perspective, is not architectural (at the current level of system scope). This allows us to distinguish between detailed design and implementation decisions on the one hand, and architectural decisions on the other—the former have local impact, and the latter have systemic impact. That is, architectural decisions impact, if not all of the system, at least different parts of the system, and a broad-scoped perspective is required to take this impact into account, and to make the necessary trade-offs across the system.

For example, if the system under consideration is an individual application, any decisions that could be made by component designers or implementers should be deferred to them and not appear as part of the architecture. If the scope of the architecture is a family of applications (or product line), then any decision that relates only to a single application (or product) should be deferred at least to the application architecture and not be part of the application family architecture.

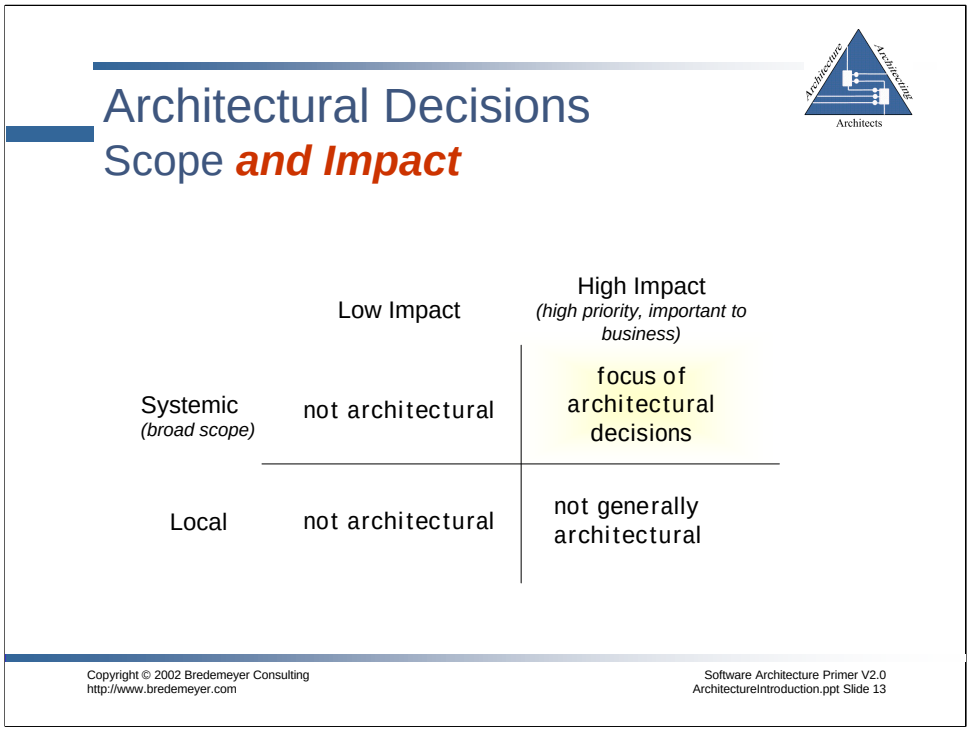
However, a decision may have systemic impact but not be very important, in which case it is also not architectural. By nature, architectural decisions should focus on high impact, high priority areas that are in strong alignment with the business strategy



Reuse Example

When we focus on one product, we do all we can to tune that product for it's customer requirements, as best we understand them. This, by nature, tends to make the components of that product less likely to be a good fit for other products, even if we have followed all the architectural principles of encapsulation and loose coupling.

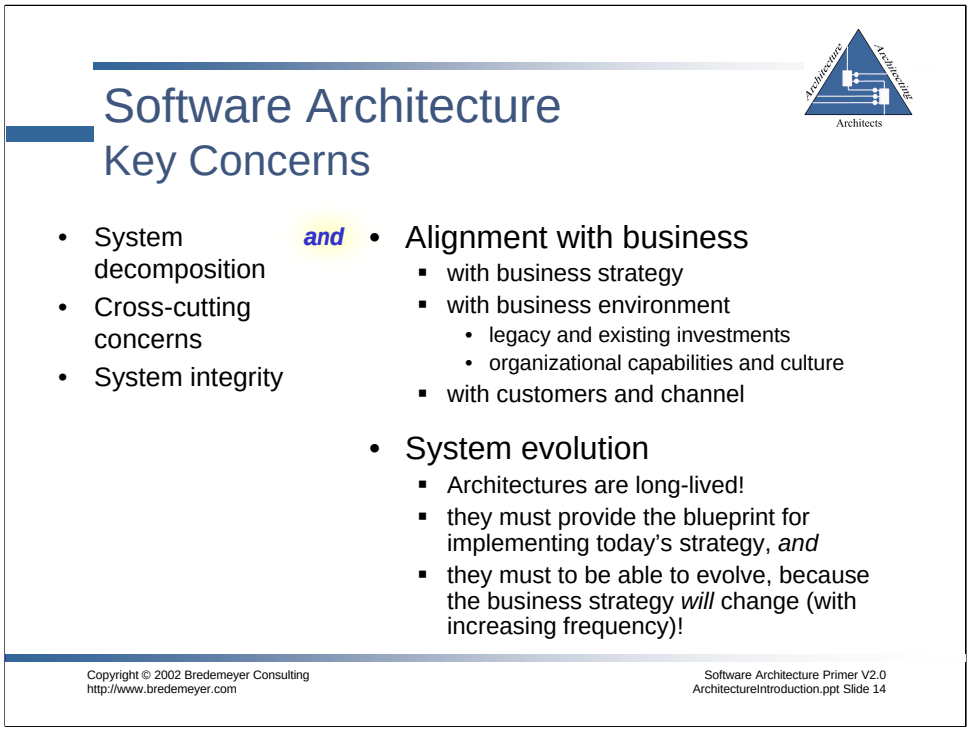
Architecture is very much about making tradeoffs and compromises in order to optimize globally across the system or systems within the scope of the architecture. Individual system priorities have to be considered in the context of the priorities for the overall system.



System Priority Setting

In the design of any complex system, one has to pick where to excel, and where to make the myriad compromises necessary to get the system built. It is essential to make priorities explicit so that attention can be focused on high-priority areas, and so that trade-offs between conflicting concerns can be made rationally, and decisions can be justified in terms of agreed priorities. Architects need to lead the priority-setting process for technical aspects of the system. This is a highly strategic process, and has to be informed by:

- the business, including business strategy and direction, core competencies and resources, and politics
- the market including customers, competitors, suppliers and channel
- technology including trends and opportunities
- constraints including existing technology investments and legacy systems
- challenges and impediments to the success of the system, the development of the system, and the business.



Bottom-Up Decisions --> Bottom-up Strategy

If developers are allowed to make any decision they wish (either because there is no architecture or the architecture is not taken seriously), then we must accept that the *real* strategy of the business will be *emergent* from their decisions.

Top-down Strategy:

Business Strategy --> Architecture Strategy --> Product Implementation

If we want strategy to drive the business, then the business strategy must be explicitly stated and shared with the architects, and translated into a technical strategy that will be the foundation for this generation of business strategy, and evolve to be the foundation for the coming generations of business strategy. This technical strategy shapes the architecture, and ultimately the products that use the architecture (now, and for the next 5 to 10 years!).

The business strategy directs what organizational capabilities must be sustained and built, and the technical strategy determines what the product capabilities are--and determines what capabilities are even feasible. If there is a disconnect between the business strategy and the technical strategy, the product capabilities will not necessarily match what the business strategy intends.

However, in the typical organization, there a disconnect between business strategy formulation and architecture strategy formulation. Business strategy is not informed by architects, and the architecture strategy is not informed by the business strategy process.

Architecture is the key to:

• making complex systems tractable, so that we can design and build ever more challenging systems without getting into a quality morass

• managing the development of complex systems; effectively managing large numbers of people

• increasing reuse and reducing waste, rework and redundancy

• increasing consistency and integration among systems

and

• implementing business strategy

• creating sustainable strategic advantage

What is Architecture? Formal Definition

- "architecture is the structure of the system, comprised of
  - components or building blocks
  - the externally visible properties of those components, and
  - the relationships among them"
    - adapted from Bass, Clements, and Kazman. *Software Architecture in Practice*, Addison-Wesley 1997

Software Architecture

Software architecture encompasses the *set of significant decisions about the organization* of a software system

• selection of the structural elements and their interfaces by which a system is composed

• behavior as specified in collaborations among those elements

• composition of these structural and behavioral elements into larger subsystems

Booch, *Presentation at Software Developers Conference* 1999

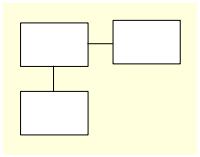
Architectural Model

An architectural model shows how significant properties of a system are distributed across its constituent parts.

Coleman and Beringer, *Tutorial Presentation at UML World 2000*

Architecture Essentials
Components and Relationships

- Components
  - high-level building blocks of the system
  - provide
    - modularity
    - separation of concerns
- Components collaborate
  - to provide services (functionality)
  - at some service level (system qualities)
- Component Interfaces
  - provide for encapsulation, with restricted access points
- Component Specifications
  - define externally visible properties
  - provide usage guidelines



Externally Visible Properties
The Purpose of Interfaces

- Interfaces
  - define the component's access points
  - enable component plug-and-play
    - decouple clients and providers
    - serve as a contract between component providers and clients
  - define externally visible properties

- A well-defined interface
  - makes it easy to identify and understand the purpose and behavior of a component, and how to use it
  - serves to increase developer productivity
    - focus on assembling and linking proven components via their interfaces

Interfaces--the "seams in the system"

An interface is a list of operations providing a coherent service. It

• names a collection of operations that can be invoked by clients

• specifies the operation signatures (at least)

• is not the implementation of any of the operations

• gives a name to a collection of operations that work together to carry out some logically interesting behavior of a system (i.e., a service). (Kruchten, 1998)

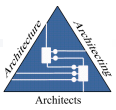
• prescribes the mechanisms for a component to interact with other components.

A component that conforms to a given interface satisfies the contract specified by that interface and may be substituted in any context within which the interface applies. (Kruchten, 1998)

Well-structured interface

"A well-structured interface provides a clear separation between the outside view and the inside view of [the component], making it possible to understand and approach [the component] without having to dive into the details of its implementation." (Booch, 1999, p. 155)

Interface Signature Syntax



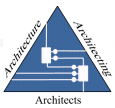
interface ContextManager {

exception UnknownParticipant { long unknownParticipant; }
exception TransactionInProgress { string instigatorName; }
exception InvalidTransaction { string reason; }
exception InvalidContextCoupon { }
exception ChangesNoteEnded { }
exception AcceptNotPossible { }
...
StartContextChanges (in long participantCoupon, out long contextCoupon) raises (UnknownParticipant, TransactionInProgress, InvalidTransaction)

EndContextChanges (in long contextCoupon, out boolean noContinue, out string[ ] responses) raises (InvalidContextCoupon, NotInTransaction, InvalidTransaction)

PublishChangesDecision (in long contextCoupon, in string decision) raises (NotInTransaction, InvalidContextCoupon, ChangesNotEnded, AcceptNotPossible)
… }

Interface Specification
Semantics



- To serve as a contract between component providers and clients, interfaces must be
  - fully documented
    - semantics, not just syntax
  - understandable, unambiguous, precise

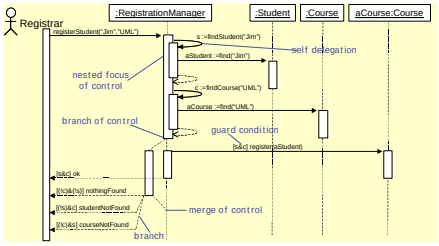
Adding semantics: informal description, models, pre/post conditions

StartContextChanges

This method enables an application to indicate that it wants to start changing the common context. The application denotes itself with its participant coupon as the value of the input *participantCoupon*. A context change transaction is initiated. Actual changes to the context data are conducted via the ContextData interface. The output *contextCoupon* is the value of the context coupon that has been assigned by the context manager to denote the change transaction.

Architecture Models

- Architecture models
  - thinking tools
    - explore alternatives and ideas (more cheaply than prototyping or trial by building the system)
    - e.g., find interface operations by exploring component collaborations
  - document the architecture
    - descriptive or prescriptive
    - help visualize the system



Unified Modeling Language (UML)

UML is the Unified Modeling Language

• It is a standard approved by the OMG

• Development of UML was initiated by Grady Booch, Jim Rumbaugh and Ivar Jacobson at Rational, but had broad industry involvement

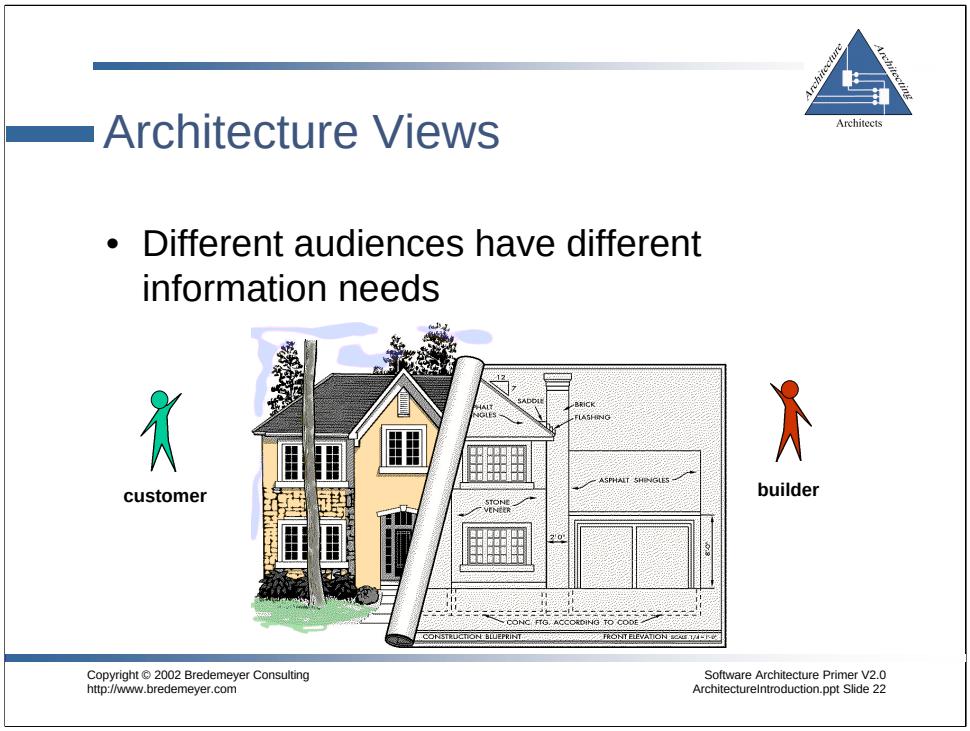
• Roots in OO methods, but ***not*** limited to object-oriented development

UML is

• A modeling *language* with vocabulary and rules, and well-defined semantics

• Primarily visual, but does have the Object Constraint Language (OCL) and allows for text annotations

• Extensible and customizable through stereotypes



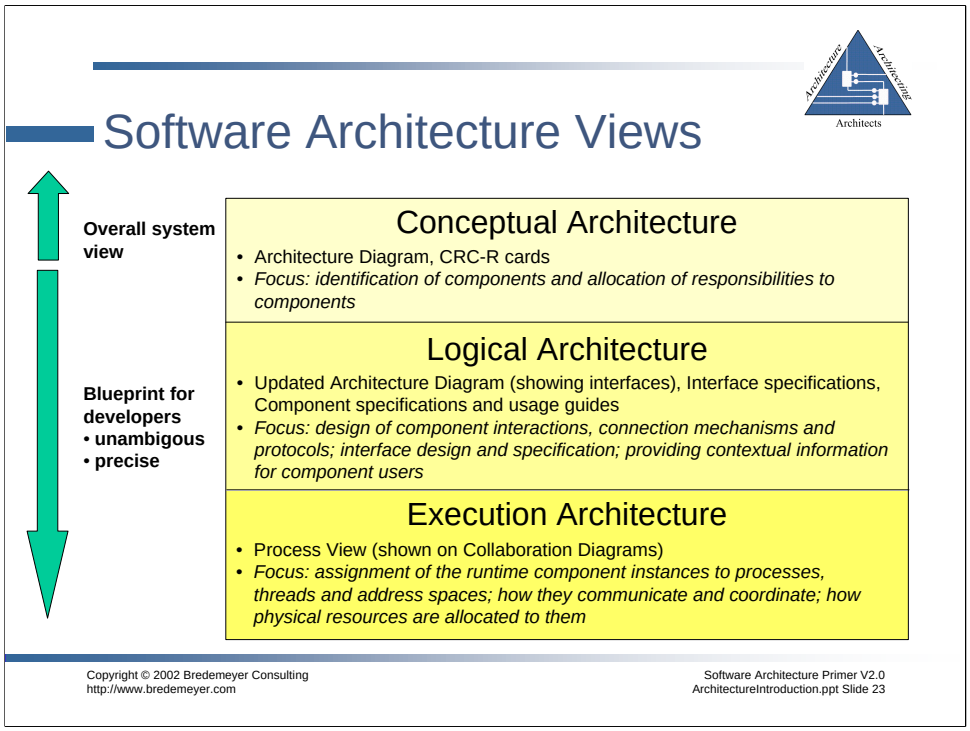
Architecture Stakeholders

Architecture stakeholders include stakeholders in the development organization, e.g.:

- management
- architects
- developers
- QA
- field support
- marketing

and stakeholders in the customer organization, e.g.:

- users
- purchasing decision makers
- system administrators



Architecture Decisions

Our layered model of software architecture reflects the different architectural decisions and areas of concern or focus and architectural thinking. We will discuss the different models or views that best support architecture decision making and documentation/ communication in each of the areas of concern.

Conceptual Architecture

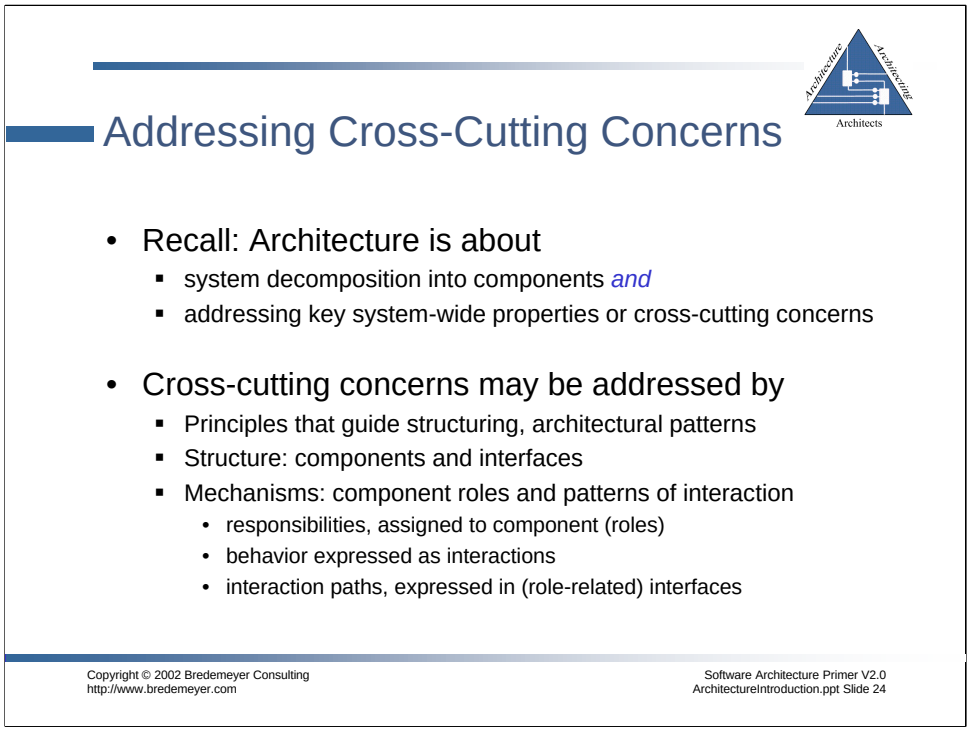
The Conceptual Architecture identifies the high-level components of the system, and the relationships among them. Its purpose is to direct attention at an appropriate decomposition of the system without delving into details. Moreover, it provides a useful vehicle for communicating the architecture to non-technical audiences, such as management, marketing, and users. It consists of the Architecture Diagram (without interface detail) and an informal component specification for each component.

Logical Architecture

In Logical Architecture, the externally visible properties of the components are made precise and unambiguous through well-defined interfaces and component specifications, and key architectural mechanisms are detailed. The Logical Architecture provides a detailed “blueprint” from which component developers and component users can work in relative independence. It incorporates the detailed Architecture Diagram (with interfaces), Component and Interface Specifications, and Component Collaboration Diagrams, along with discussion and explanations of mechanisms, rationale, etc.

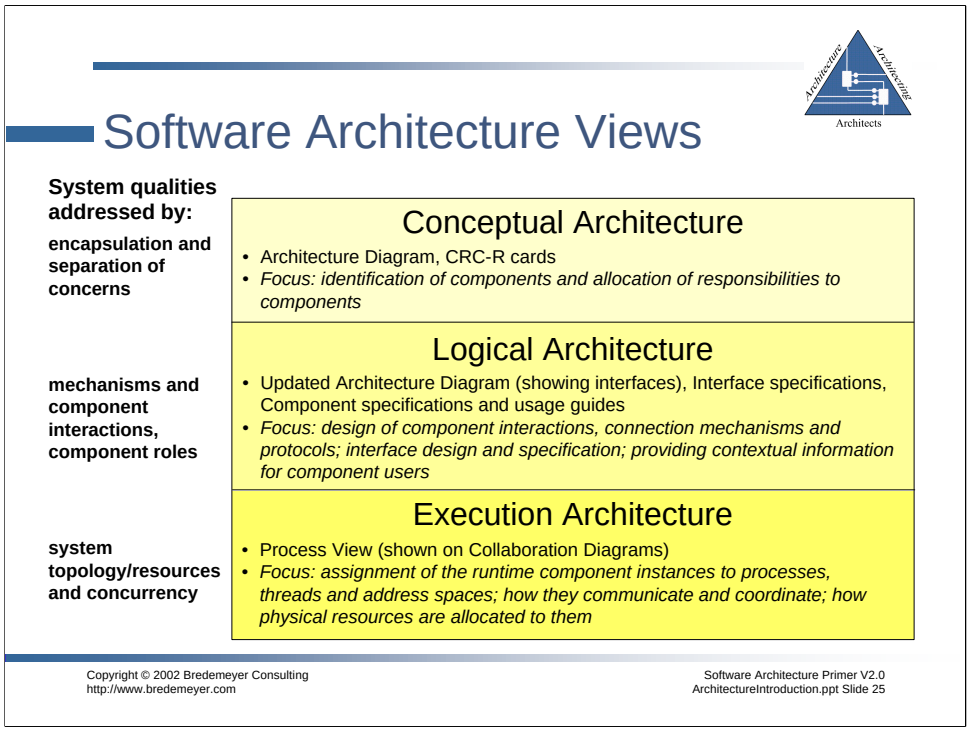
Execution Architecture

An Execution Architecture is created for distributed or concurrent systems. The process view shows the mapping of components onto the processes of the physical system, with attention being focused on such concerns as throughput and scalability. The deployment view shows the mapping of (physical) components in the executing system onto the nodes of the physical system.



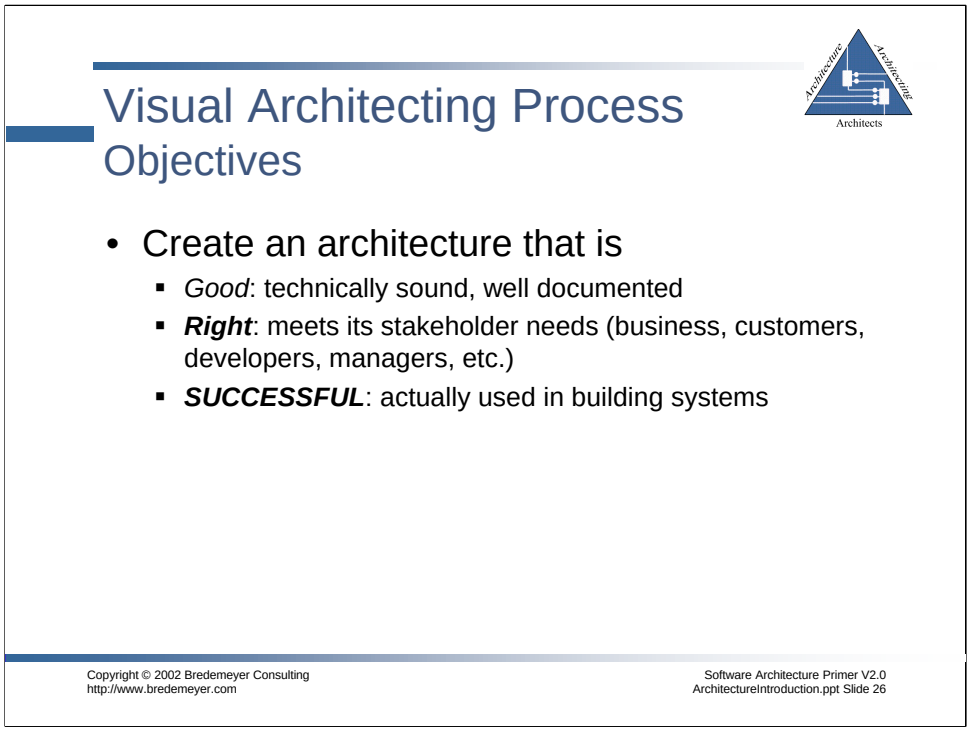
More than Components and Relationships (and Boxes and Lines)

Saying that architecture is just about components and relationships is like saying that designing a house is just about the floor plan! Qualities, like structural soundness, restrict and are restricted by some of the spatial layout choices, but also have a lot to do with structural aspects that are not evident in a floor plan at all. For example, the direction of the beams supporting the roof has something to do with width versus length of the area under the roof, as well as the roof style. Other qualities, like light, have something to do with the floor plan, but also have to do with placement and size of windows and doors, height of ceilings, even wall color and mirrors.



Architecture Decisions

Our layered model of software architecture reflects the different architectural decisions and areas of concern or focus and architectural thinking. We will discuss the different models or views that best support architecture decision making and documentation/ communication in each of the areas of concern.

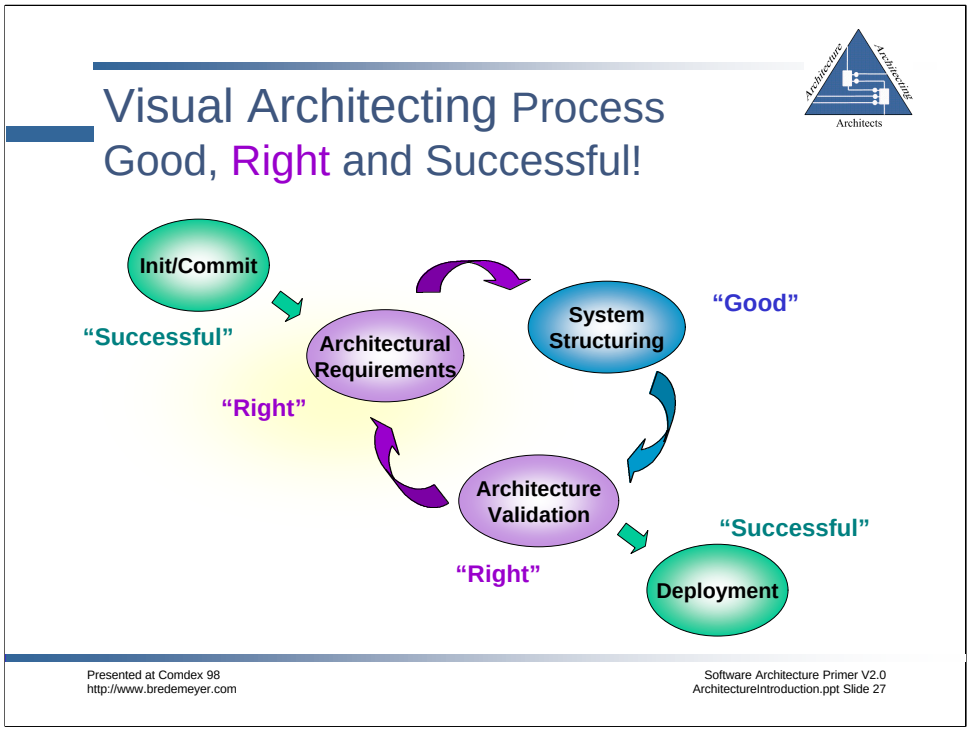


Good Architecture Documentation

Relates to the understandability, approachability, and usefulness of the documentation format and content to the various stakeholders. Expressing the architecture in terms of *views*, and tailoring documents/presentations to different stakeholders, enhances the usefulness of the documentation. Good documentation also includes *rationale* for the architectural decisions.

Bad Architecture

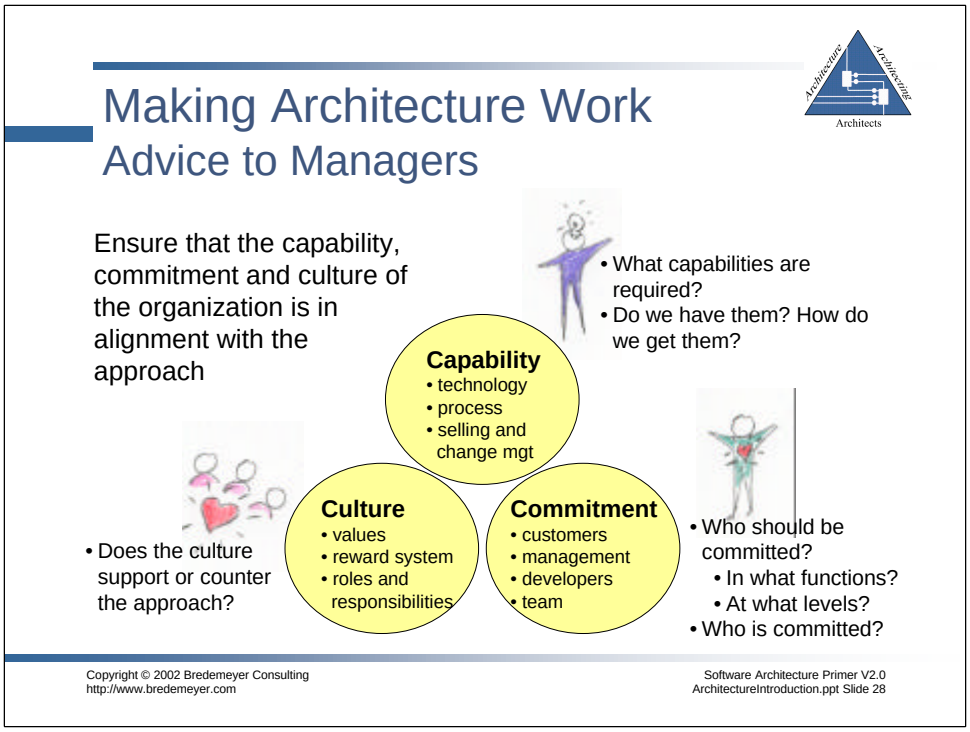
A bad “architecture” can have the following characteristics: monolithic, so even small changes are distributed through code and cause unpredictable results with long compile/link/debug cycle times; hidden implicit coupling in code; multiple inconsistent ways of doing the same thing; not documented and so has a high learning curve.



Architectural Requirements

A "good" architecture solves the problem that it solves well, but it does not necessarily solve the right problem. In order to do so, the architects cannot just rely on past experience and intuition--though these are important too. A "right" architecture is one that will enable the development organization to implement the current business strategy, and evolve to keep pace with changes in business strategy.

To create a right architecture, the architecting team translates business objectives into architectural objectives. It understands the current organizational context, and looks at trends to make assertions about the future. It considers scenarios (good and bad, likely and unlikely). It takes stakeholder goals into account in establishing function and non-functional requirements, and establishes the relative priority of these requirements. A solid understanding of current behavioral requirements, and a well-grounded projection of future behavioral requirements, drives the structuring choices that are at the heart of architecting. Tradeoffs are driven by the prioritized set of system qualities (or non-functional requirements), and mechanisms are designed to accomplish key cross-cutting concerns (system properties that cannot be localized to particular components).



3 C's

We are interested in knowing if the organization, and in particular the architecture team and lead architect, and any impacted projects, managers, etc., have the capability, commitment and culture to make the architecture effort succeed. Strengths will point to areas to build on. Weaknesses will point to areas to work on.

Rather than ignoring the forces that will likely bring the project to its knees, this approach allows us to identify risks, so that we can come up with a risk management plan.

Making Architecture Work
Advice To Architects

- The architecture must serve its stakeholders
  - it *must* address a strategic business objective of the key sponsor(s)
  - it *must* contribute immediate value to utilizers (developers, project managers, etc.) of the architecture

Reference

Role of the Architect Workshop from Bredemeyer Consulting.

Making Architecture Work
Advice To Developers

- It the architecture directly impacts you
  - follow the architecture
    - active acceptance of leadership
    - contribute, collaborate to achieve the goal
- If not, don't get in the way
- Both following and getting out of the way involve
  - goodwill, trust
  - putting your ego aside
  - leadership - your behavior affects others

Example of Following

One architecture team had three of the most senior, most talented architects at HP. They knew that to be successful, all three could not try to lead. This would cause too much division in the team. They appointed a leader, and as Joe Sventek puts it, they let him be a "benevolent dictator with a baseball bat". That is, they allowed him to set direction, make difficult decisions to break logjams, and generally lead the team. This does not mean that they were not active in debates about how to solve the challenges of the architecture, but rather that they allowed the lead architect to make choices between alternatives when there was no consensus.

"People who say it cannot be done should not interrupt those who are doing it"
Anon, quoted in Cougar, 1996

Key Message

- A “good” and “right” architecture will not “just happen”
- It requires action
  - leading the effort
  - following, supporting, contributing to the effort
  - getting out of the way
- Everyone has a role to play in the success

Resources

- Resources for Software Architects web site
  - http://www.bredemeyer.com
- Enterprise-wide IT Architecture web site
  - http://www.ewita.com
- Philips Gaudi project
  - http://www.extra.research.philips.com/natlab/sysarch/index.html

Books

- Dikel, D., D. Kane and J. Wilson, *Software Architecture: Organizational Principles and Patterns*, Prentice-Hall, 2001.
- Maier and Rechtin, *The Art of Systems Architecting, 2nd Ed*. 2001.
- Michael McGrath, *Product Strategy for High Technology Companies*, 2001
- Rechtin, E. *Systems Architecting: Creating and Building Complex Systems*. Prentice-Hall, 1991.

Papers

- Papers on http://www.bredemeyer.com/papers.htm
  - "Software Architecture: Central Concerns, Key Decisions" by Ruth Malan and Dana Bredemeyer, May 2002.
  - "Minimalist Architecture" by Ruth Malan and Dana Bredemeyer, May 2002.
  - "The Visual Architecting Process" by Ruth Malan and Dana Bredemeyer, February 2002.
  - "Architecture Teams", by Ruth Malan and Dana Bredemeyer, March 2001.
  - "Role of the Software Architect" by Dana Bredemeyer and Ruth Malan, 2002 (minor revision of 1999 paper).