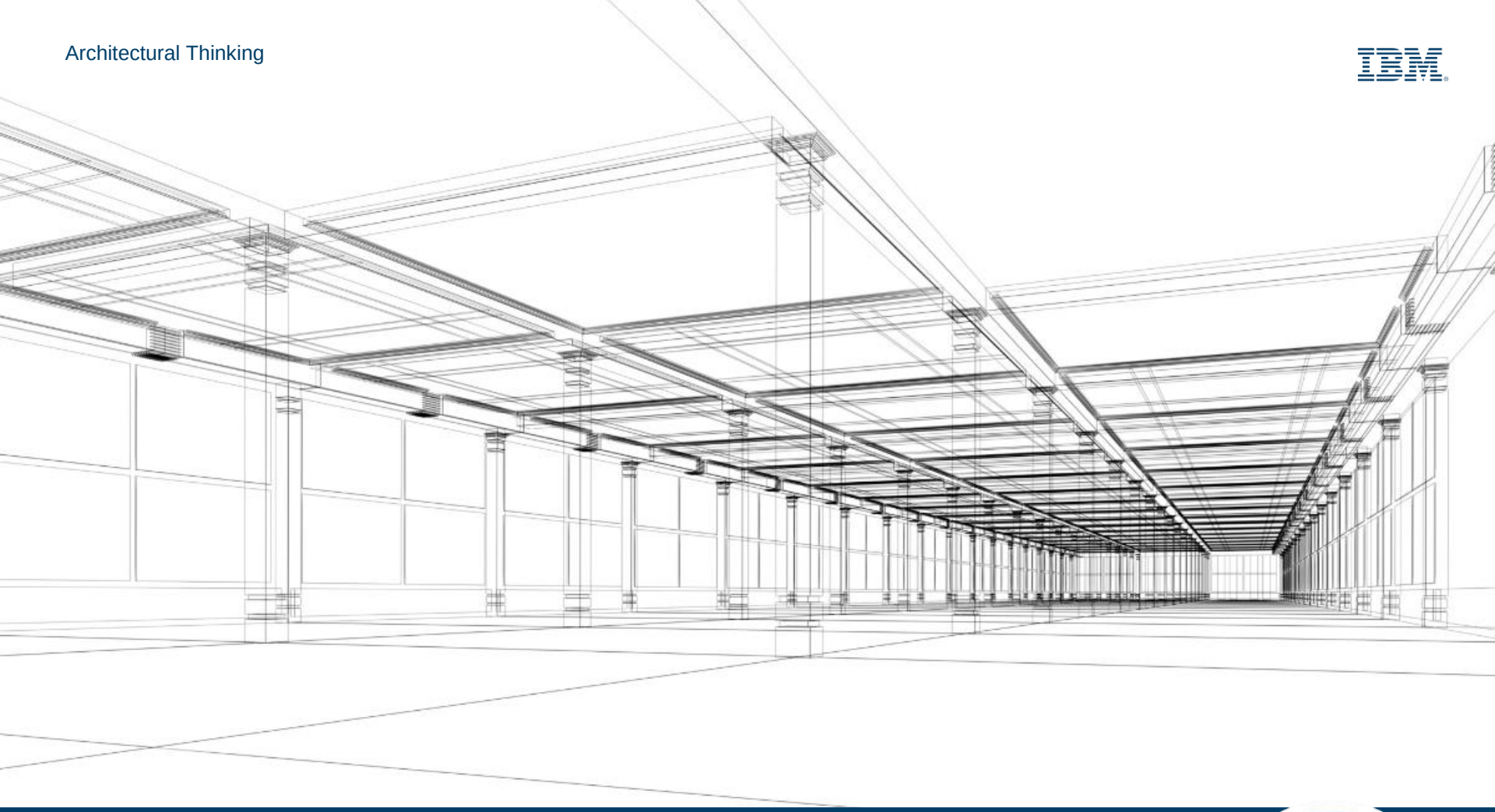


Architectural decisions and principles

Module outline

- **Introduction and objectives**

What are architectural decisions and principles?

Pragmatic architecture decision making

Summary

Learning objectives

At the end of this module, you should be able to:

- Define architectural decisions and architectural principles
- Describe how Architects use decisions and principles
- Understand best practices (e.g. alternatives should be well researched) and things to avoid (e.g. single alternative decisions) when making architectural decisions
- Understand why solution proposals are not always acceptable to clients
- Describe how to apply proven optimization patterns to make your solutions simpler, cheaper, and faster to deliver

Module outline

- Introduction and objectives
- **What are architectural decisions and principles?**
- Pragmatic architecture decision making
- Summary

What is an architectural decision?

More often than not, there is more than one possible solution option to a given architectural problem. This includes components and their relationships, choice of technology, assigning functionality to various components, making placement decisions for components hosted within various infrastructure nodes and so on. The options may have different costs associated with them, the extent they satisfy various requirements and may present different ways of balancing competing stakeholders' concerns.

Architectural decisions enable architects to:

- Formally document the critical choices that they make in their creation of the solution;
- Understand and agree why the solution looks the way it does.

Architectural decisions:

- Make explicit the rationale and justification of the decisions made
- Help ensure the solution satisfies both functional and non-functional requirements and if it does not, help making it explicit to the stakeholders
- Prevent unnecessary rework throughout the solution delivery life cycle

Architectural decisions should be formally documented and managed

Architectural Decisions must be properly documented, preferably using a standard template. It's elements are as follows:

- **The architectural decision:** Written as a statement; for example: "The system will use the Messaging integration pattern to communicate with the XYZ application."
- **Unique identifier:** A unique code that unambiguously identifies the decision; for example: "AD-065."
- **Issue or problem:** A description of the situation over which a choice has to be made; for example: "How should the system integrate with the XYZ application?"
- **Assumptions:** What is believed to be true about the context of the problem, constraints on the solution, and so on; for example: "The XYZ application can support a messaging interface."
- **Alternatives:** If there are no alternatives, then it can't be an architectural decision. An enumerated list of alternatives and explanations; for example: "1 – File transfer, 2 – Messaging."
- **Decision:** The decision taken; for example: "Alternative 2 is chosen."
- **Justification:** Why the decision was made; for example: "Messaging provides reliable, asynchronous program-to-program communication."
- **Implications:** The consequences and impacts of the decision taken or architectural option chosen on other elements or aspects of the solution; for example: 1. An integration server node will be required, 2. An additional layer will exist that must be accounted for in performance modeling.

Architectural decision best practices

#1 State the to-be decision as a question

#2 Alternatives should be well researched keeping immediate and future requirements. In some instances a Proof of Concept may be recommended

#3 Decision must be in line with the Architecture Principle

2.2 AD02: Choice of Point of Sale (PoS) Client

Subject Area	PoS Client Technology	**Topic**	Client
Design Decision	What should be the underlying technology to develop the PoS Client?	**Id.**	AD02
Issue or Problem Statement	▪ Client infrastructure constraint includes the continued use of the legacy PoS. Legacy PoS models have limited system resources e.g. 32 MB of RAM. ▪ The current PoS Client is in VC++ and is tightly coupled with other component.		
Assumptions	▪ Legacy PoS model 4614-111 will not be supported in the future. ▪ All other legacy PoS models can be upgraded to standard configuration		
Motivation	▪ *Performance*: PoS client technology should support standard and faster communication between the PoS and Application Server ▪ *Integration:* PoS client should be able to integrate with the transport.		
Alternatives	1. ***Thick Client (Custom Development)*** ▪ *Java Swing* - Common technology for the development of PoS clients o System Requirements: Runtime environment requires 32MB RAM. o Uses RMI protocol that will enable fast communication ▪ *VC++* - Expedites development & runs efficiently on legacy PoS models Enables realization of a rich GUI experience. o Support has been withdrawn by ISV Microsoft in the year 2005. ▪ .NET Based (C#) - Fast development technology and Rich GUI o Integrates with transport (WMQ) and via Web Services 2. ***IBM Lotus Expeditor*** ▪ Will expedite development of the PoS client however has high system configuration needs. 3. ***Thin Client (Browser)*** ▪ Provides a thin interface to PoS operations and simplest client to maintain. ▪ OS constraints limit leveraging recent browser versions that could support AJAX. Client side validations will be restricted to Javascript and will be cumbersome to maintain. ▪ Integration with peripherals would need to be verified.		
Decision	Alternative 1 - .NET Based (C#) client for all PoS models.		
Justification	▪ Thin browser based clients will not support the usability and speed that the PoS operators are used to. Additionally, does not provide constructs to customize the PoS keys ▪ IBM Lotus Expeditor will require higher system resources to perform, which will not be feasible in the legacy PoS that can support a maximum of 128 MB of RAM		
Implications	System Resources on legacy PoS will have to be upgraded to standard configuration.		
Parties Agreeing to the Decision	Person 1 (Representing the solution architecture team) Person 2 (representing the ARB) Person 3 (Representing Enterprise Architecture Standards team)		

#4 Clearly state the assumptions which are influencing the decision

#5 Implications affecting other architectural decisions must be documented

#6 Key stakeholders and approvers documented as agreeing

Architectural Decisions are grouped together in an *Architectural Decisions Register* work product

Decisions and principles: What's the difference?

Decisions and principles have much the same effect on a solution's architecture: They both represent some design constraint to which the solution must conform

Their source is very different:

- Architectural decisions are made within the context of the development of the solution's architecture.
- Architectural principles are imposed from outside the context of the solution architecture; they are often part of the enterprise's Enterprise Architecture.

For any given solution architecture, architectural decisions and architectural principles often carry equal weight.

- It can be the case that they are subsequently in conflict, requiring architectural governance processes to ensure that the correct resolution is found.

Principles are defined by the enterprise as a whole, as a powerful means of ensuring that all their otherwise independent solutions fit together.

If an enterprise recognizes a project's architectural decision as being generally applicable, then the decision may be adopted as a principle.

Architectural principles are part of a “guidance framework” that also include policies and guidelines (1 of 2)

Principles, policies, and guidelines* provide a framework within which an enterprise or program can direct the program or project team, so that the target IT system fits within the broader context, conforming to legal and “best practice” requirements.

- **Policy**
  - An enterprise-wide statement of *how things will be done*. Often legally binding and imposed from outside the enterprise. Non-conformance is likely to be a legal offence, or have equivalent consequences such as dismissal or financial punishment.
    - Data protection, health and safety, and industry regulations
  - Exceptions cannot be granted: alternative must be found to ensure policy is adhered to.
- **Principle**
  - An enterprise-wide rule which an organization will use to make architectural decisions. Usually determined from within the enterprise or at the divisional or some other “supra-program” level.
    - Data distribution that affects operations, “Buy not Make” that affects Application Development, and Single sign-on that affects security management
  - Exceptions can be granted: While usually written “you must…,” principles may conflict or be at variance with requirements.

*This vocabulary may not be the same for your client, but the three “levels” are usually there.

Architectural principles are part of a “guidance framework” that also include policies and guidelines (2 of 2)

- **Guideline**
  - Rarely binding, an enterprise-wide statement of best practice, giving solution architects the opportunity to adopt consistent approaches to architectural challenges.
  - Could be the same examples as principles but would be missing explicit motivation and implication statements
  - Formal exceptions are not required.

Principles are often “hard” to conform to; they usually focus on the needs of the enterprise and not the project

Principles are designed to change behavior, often away from the narrow needs of the project “live by New Year’s day” toward the needs of the enterprise “profitable at the end of next year”.

They, therefore, have a clear structure aimed at explaining why they are the way they are and their consequences for the Architect:

- Name
  - A short name or title to identify the principle
- Statement
  - What the principle is
- Motivation
  - Why the principle is the way it is; the rationale or impetus behind the enterprise wanting its systems to be architected this way
- Implication
  - The consequences of the principle both on the enterprise which is always positive and on the program sometimes implying additional cost

    d.1. This will increase total cost of acquisition but it will reduce the total cost of ownership.

Common mistakes (or pitfalls) defining Architectural principles and decisions

Their source is very different:

- Undocumented, unclear, or ambiguous decision
- Single alternative decision
- Inclusion of design guidance and code snippets
- Missing dependencies or reference between two or more related decisions
- Decisions scattered in design documents
- Not following a standard approval process for decisions, especially decisions requiring exception approval

Principles:

- Not knowing what the enterprise's standards and principles are, when creating the solution
- Not discussing deviation from an architecture principle with the governance board, for exception approval process
- Not following a standard approval process for proposal and adoption of new principles

Module outline

- Introduction and objectives
- What are architectural decisions and principles?
- **Pragmatic architecture decision making**
- Summary

Pragmatic and fit for purpose Architecture Decision making (1 of 2)

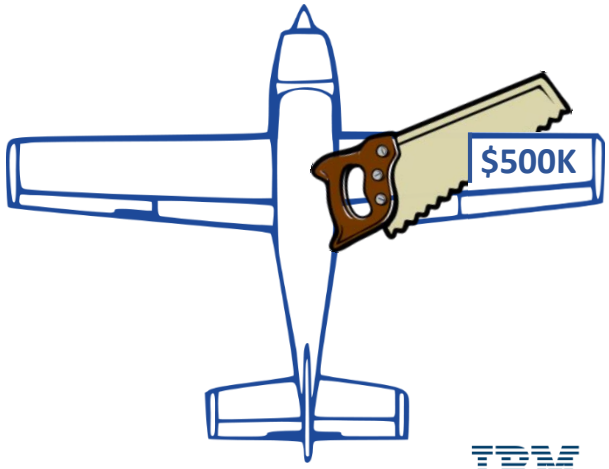
Too often, our solution proposals would work – but they don't win…

- IBM is good at solutioning but we don't always win the deal
- Often we take a broad interpretation of scope
- We assume a good solution will trump a cheaper one
- And then we lose on price
- Other bidders find a way to get to the right cost by making different assumptions

During solution design, the architect should "start with the winning price and see what can go in to the solution" rather than "start with a winning solution and see what can be slimmed to bring cost down."

Late cost cutting can turn good solutions bad

- Price often appears for the first time at the end of the bid
- And comes as a nasty surprise
- Then there is a tendency to chop important pieces out
- And we end up with something unworkable



Pragmatic and fit for purpose Architecture Decision making (2 of 2)

As you define your solution, you should apply proven optimization patterns to help identify alternative approaches or re-use opportunities to ensure that your proposed solution:

- Will work – and is priced to win
- Uses appropriate assets and accelerators to speed up delivery and improve quality
- Is not over-engineered
- Follows solid architectural patterns and approaches
- Addresses the appropriate scope and requirements
- Differentiates IBM in the eyes of the client

Shape your solution to fit a target cost

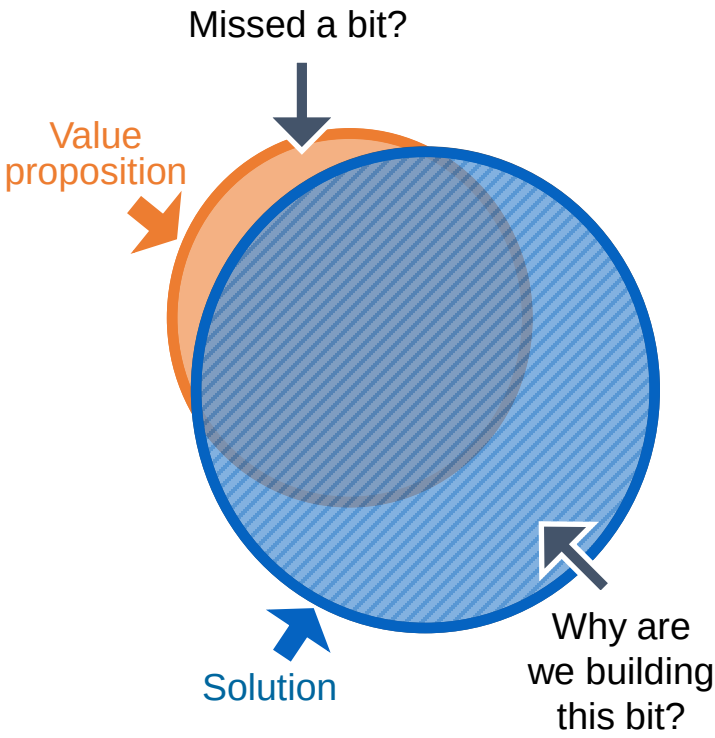
- Set a target cost
- Do basic solution shaping
- Allocate cost targets to major solution parts
- Map key requirements and fully-loaded costs to the solution outline as it is created
- You need to lead on this … don't expect it to be BAU or in IBM process: it isn't

Applying solution patterns improve our chance of winning

Pattern Area	#	Pattern Name	Description
Value Proposition	1	**Check Value Proposition**	Use Value Proposition Workshop technique. Align value to delivery schedule/capability.
Simplification	2	**Simplify the Architecture**	Is the solution over-engineered? Reduce architectural elegance where it isn't really needed or adds little value. Simplify complex architectures and excessive infrastructure layers.
	3	**Breakthrough the Design**	Take a different view of the applications and operational designs: apply Osborne's checklist, e.g.: Positive transactions to negative ones, physical updates to logically updated, etc.
	4	**Integrate the Facilities**	Integrate Service Delivery organization, reduce management overhead. Consolidate technology to make best use of the hardware, sharing services, reduce duplication of functionality
	5	**Rightshape the Platforms and Processors**	Downsize processors, change platforms or introduce different platforms. Challenge accepted dogma around platform selection. Select platforms to minimize Application Development or Service Delivery TCO.
Assets	6	**Re-use Existing Assets**	Utilize existing assets including architecture components, tools, processes, components and others including external assets.
	7	**Recycle Existing Code**	Use existing code or legacy applications through wrappering or re-engineering rather than rewriting
	8	**Apply Frameworks or Skeletons**	Apply a framework or a defined rigid development processes with work-product templates or a skeleton program approaches.
Delivery	9	**Optimize the Application Development environment**	Categorize functions into critical ones and non-critical ones, and apply the easiest Application Development environment to the latter. If suitable, use Agile approaches to deliver value sooner
	10	**Optimize the Delivery Model**	Utilize a 3-tier delivery model (onsite, local support centers & global support centers), exploit use of low cost resources.
	11	**Rationalize SW/HW License Lifecycle**	Ensure that software (and hardware) acquisition is timed appropriately through the project lifecycle.

Pattern 1: Check the value proposition

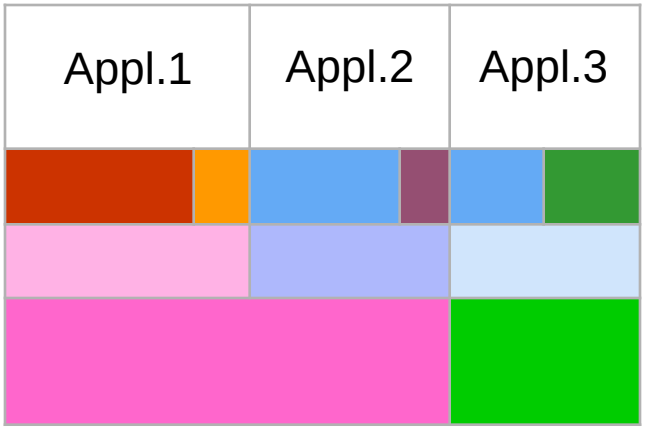
- We engineer solutions based on value propositions
- Running a separate value proposition workshop may be appropriate for large deals, but to develop value propositions for smaller deals, consider:
  - Confirming the customer wants and needs
  - Identifying value propositions – with at least one value proposition addressing each ‘wants and needs’ statement
  - Identifying IBM differentiators (“Wow!” factors)
  - Defining resources and capabilities required - leveraging all appropriate IBM brands and capabilities
- Set the high-level value proposition early and align the solution to it
- Aligning the solution and value proposition will take out cost and improve the value proposition
- The earlier the client can start realizing major components of their business case, the greater the perceived value of the overall solution is likely to be.



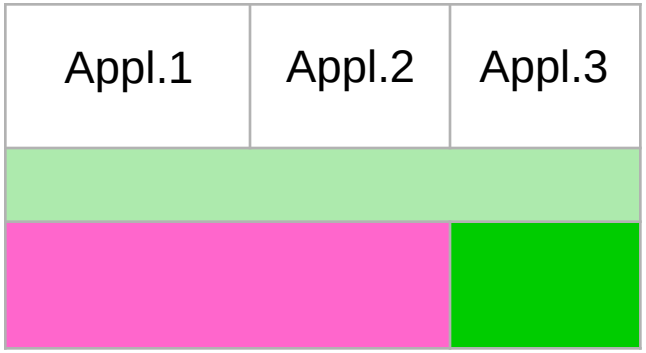
Pattern 2: Simplify the Architecture

Reduce architectural elegance where it isn't really needed or adds little value. Simplify complex architectures and excessive infrastructure layers.

Complex Architecture



Simple Architecture



Try to simplify

And compare required efforts

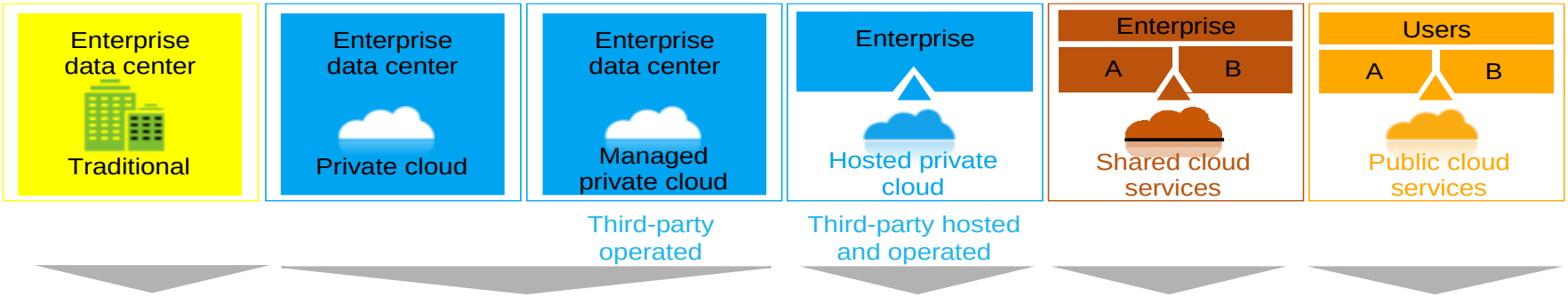
Select the best-fit simple architecture to satisfy the needs and service-levels. Avoid accommodating excessive layering.

Pattern 5: Rightshape the platforms and processors

- Consider upsizing as well as downsizing for traditional on-premises hosting. Downsizing does not always result in the lowest TCO.
- Consider cloud to support all or parts of the workload: dev/test, production, “cloudburst” of production using hybrid cloud, etc

Private
IT capabilities are provided “as a service,” over an intranet, within the enterprise, and behind the firewall

Public
IT activities / functions are provided “as a service,” over the Internet



Operator	Enterprise	Provider		
Ownership / Location	Enterprise	Provider		
Pricing model	Time & materials, fixed price, etc.	Time & materials, fixed price, variable/mixed	Pay-as-you-go	
Access	Internal enterprise network		Access through public internet or VPN	Public internet
Consumer	Single enterprise	Single enterprise or hybrid	Multiple enterprises	Any enterprise/user
Asset use	Dedicated (single tenant)	(There are a set of assets that could be flexibly allocated on a dedicated basis to multiple enterprise depending on demand)	Mixed	Multi-tenant

Module outline

- Introduction and objectives
- What are architectural decisions and principles?
- Pragmatic architecture decision making
- **Summary**

Summary

Architectural decisions help the Solution Architect communicate why the solution architecture is the way it is, across the scope of the solution:

- To facilitate design reviews and sign-off
- To minimize architectural rework during the program life cycle
- To ensure that the solution can be maintained and enhanced by architects in the future

Architectural principles help the Enterprise Architect communicate enterprise-wide decisions on the architectures of the enterprise's systems, across the scope of the enterprise:

- To maximize the usefulness of the enterprise's IT investments
- To minimize the cost of operating and maintaining these systems
- To optimize the fit between business requirements and IT solutions

Solution "rightsizing" adds value for our clients by helping to find the least costly means to deliver on the client's needs

- Encourage designing to a cost point as opposed to cost scrubbing after the design is complete
- Applies proven optimization patterns to help identify alternative approaches or re-use opportunities

References

Solution optimization

- Global Cross Brand Solution OPtimization (SOP) Technique Wiki
- Global Solution OPtimization (SOP) Technique Education
- Cross Brand Solution Optimization (SOP) Technique Paper (TP)
- Solutioning to a Price Point

THANK YOU