Advanced SW Engineering: Design Decisions

Uwe Zdun

Software Architecture

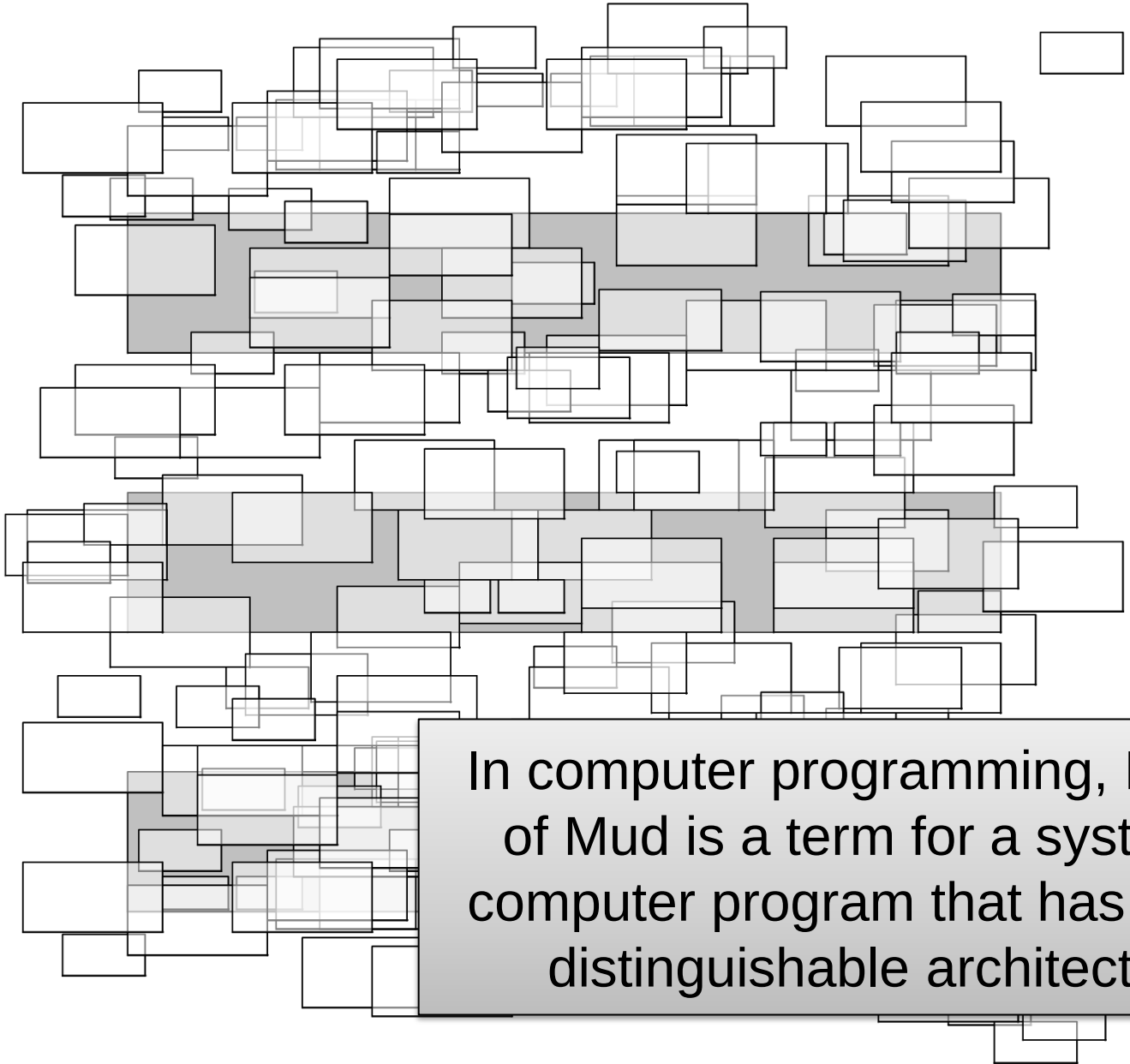
Faculty of Computer Science

University of Vienna

http://cs.univie.ac.at/swa

SOFTWARE ARCHITECTURE

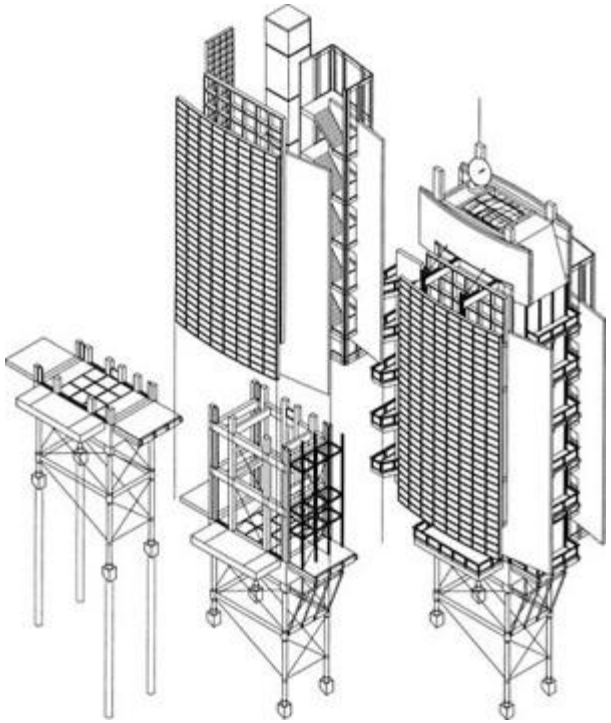
Big Ball of Mud



In computer programming, Big Ball of Mud is a term for a system or computer program that has no real distinguishable architecture.

Analogy in Civil Architecture

Inspiration from Civil Architecture



In analogy to built architecture, software architecture aims to manage the complexity of the systems we build

Defining Software Architecture

There are numerous definitions of SW architecture

See: http://www.sei.cmu.edu/architecture/definitions.html

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

– Bass, Clements, Kazman

Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled.

– Eoin Woods

DESIGN DECISIONS

Design as Decision Making

- During design, designers are faced with a number of **design issues**
- For each design issue, there are usually one or more solution alternatives (**design options**)
- The designer needs to make a **design decision** to resolve a design issue
  - The designer must choose the best option among the alternatives
- **Design Space***:* The space of possible designs that could be achieved by choosing different sets of alternatives

Informed Decision Making and Relying on Experiences

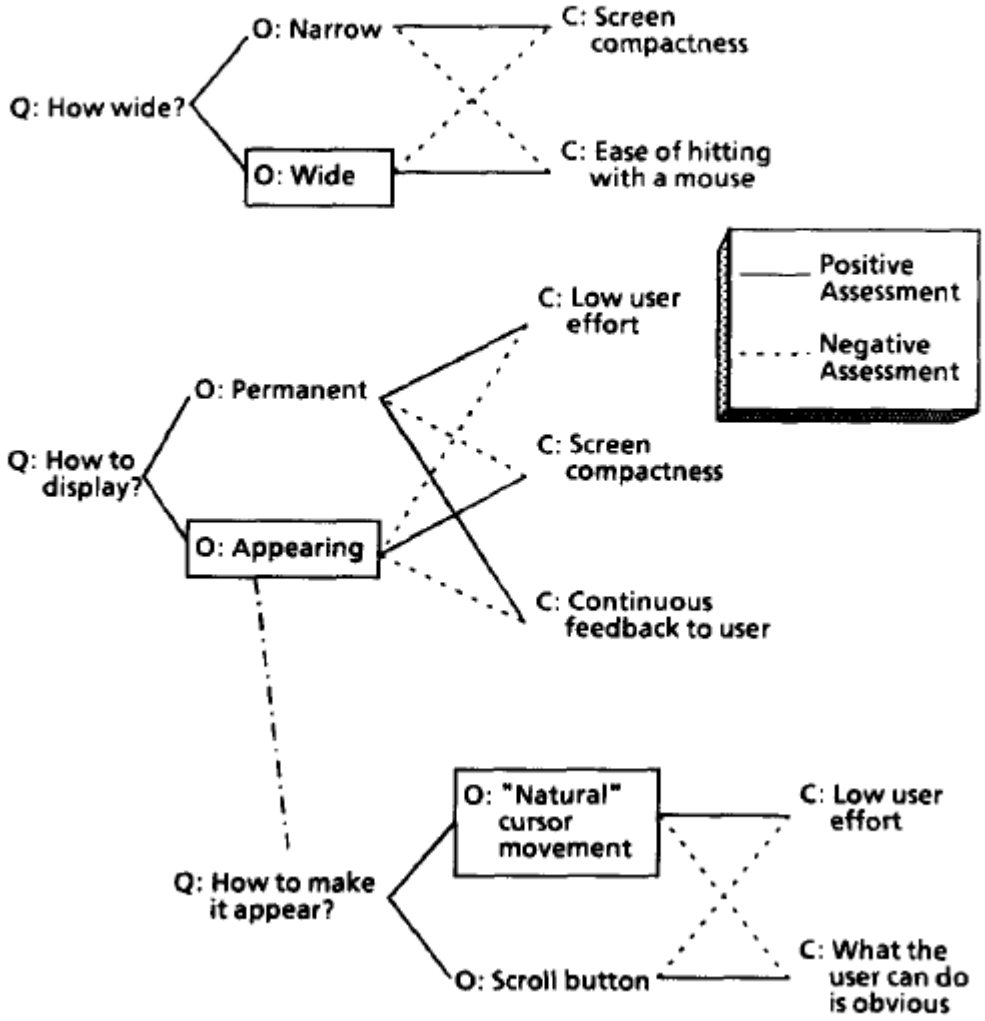
- It is important to **gather information** as a foundation for **informed decision making**
  - Difficult: **How much information is needed**?
  - Psychological research shows that **decisiveness decreases when the amount of information increases**
  - A "**critical mass**" of information should be pursued
- Often designers/architects can rely on their **experience**
  - Consider experiences of others, e.g. by using **patterns**
  - **Use cases and scenarios** can help to make the decision in the right scope

Questions, Options, and Criteria: Elements of design space analysis

- **Questions, Options, and Criteria** (QOC): Semiformal notation for a design rationale
  - **Questions**: Key issues for structuring the space of alternatives
  - **Options**: possible alternatives
  - **Criteria**: the bases for choosing among the options
- Fourth concept is also needed:
  - **Assessments**: Whether an option supports or challenges a criterion

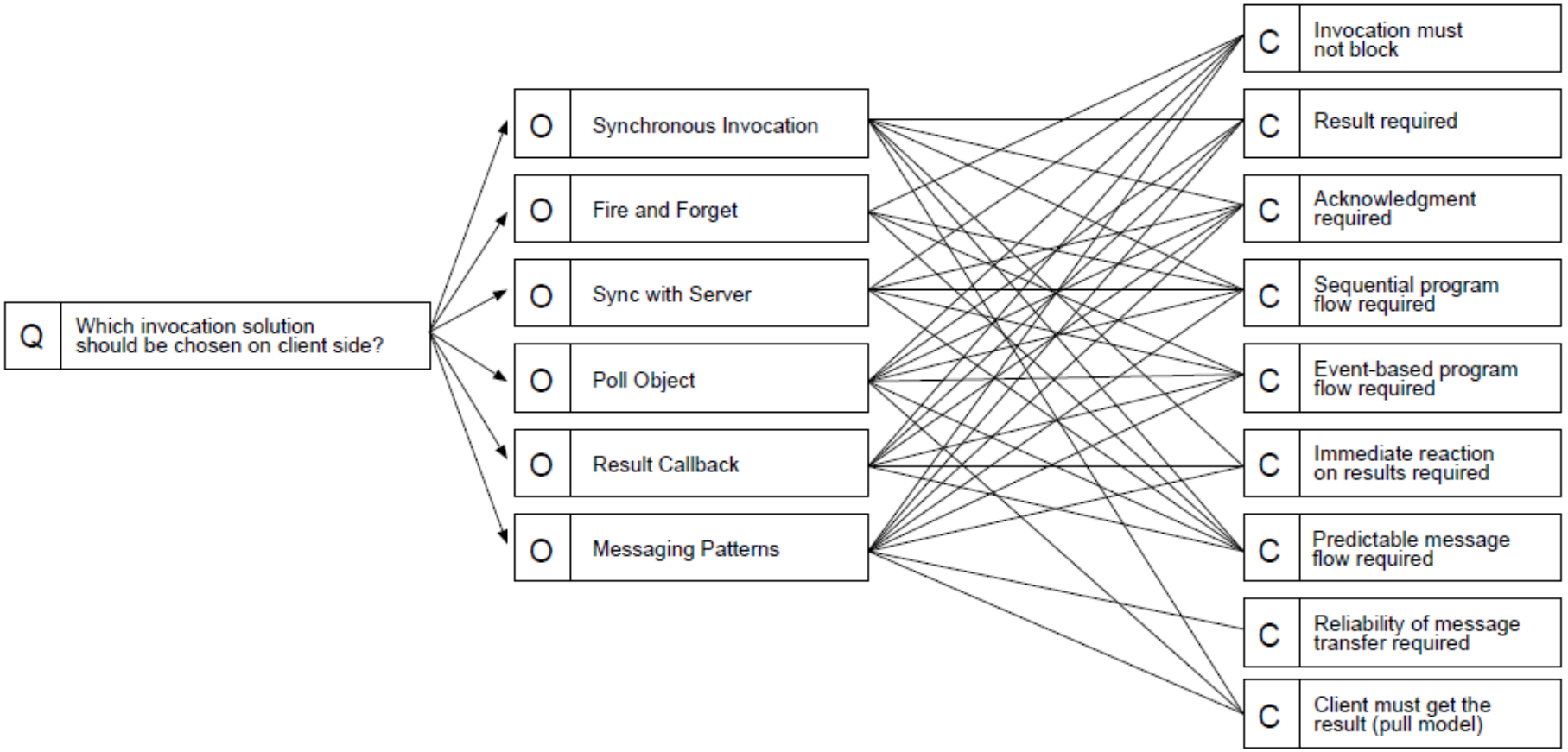
Example: QOC Diagram in HCI

Figure 3. **A QOC representation of the design space for the XCL, elaborated from Figure 2 to include Criteria and Assessments. The boxed Options are the decisions made in the design of the XCL environment.**



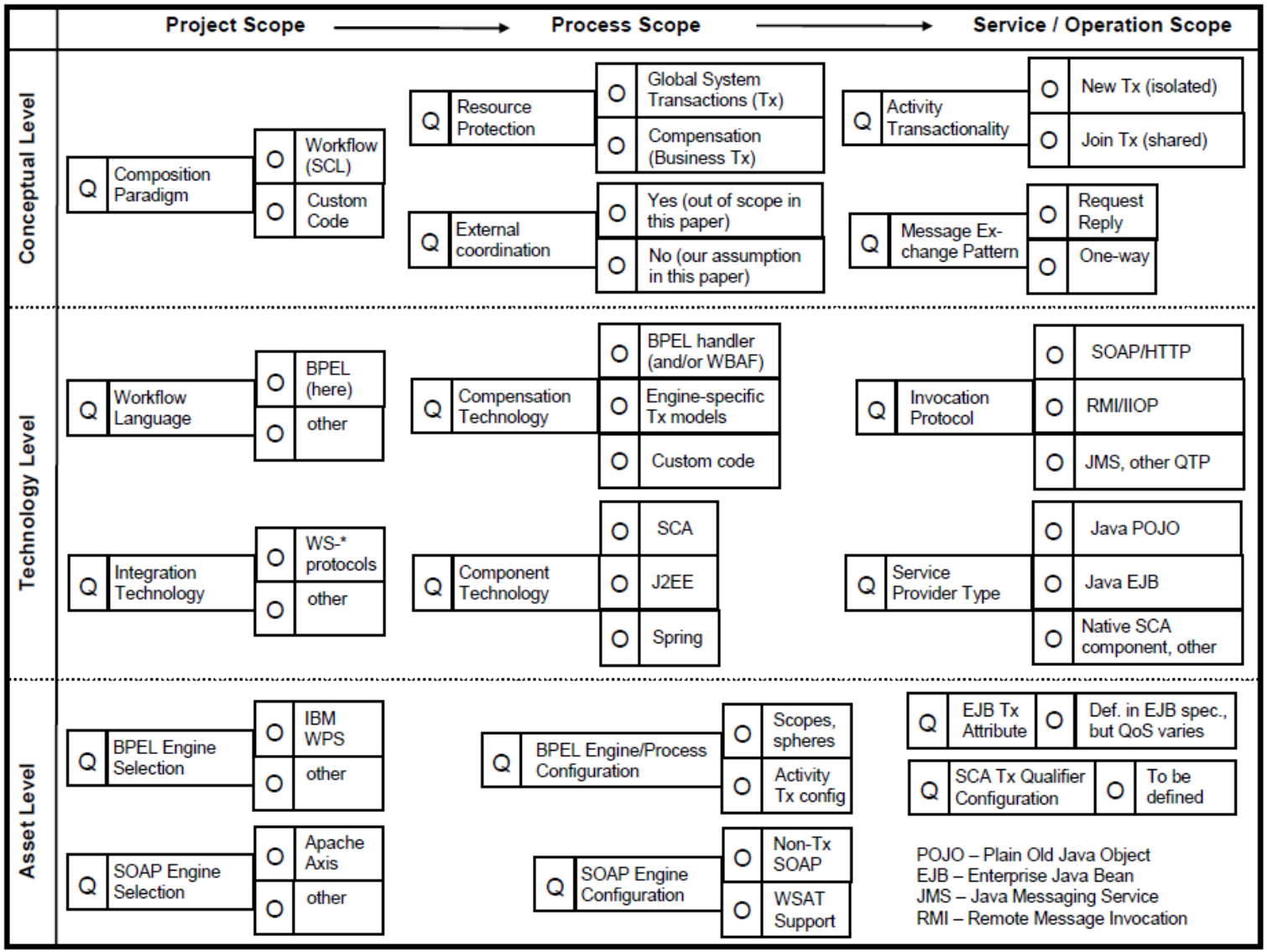
From: Allan MacLean, Richard M. Young, Victoria M.E. Bellotti, Thomas P. Moran (1991) Questions, Options, and Criteria: Elements of design space analysis

Example: QOC Diagram in Software Design



From: U. Zdun. Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis. Software: Practice & Experience, vol. 37, no. 9, pages 983-1016, Wiley, 2007.

Example: Excerpt from a SOA Design Space



From: Olaf Zimmermann. Recurring Architectural Decisions: A Context-Specific Guide through SOA and Cloud Design. Saturn 2010 Tutorial.

ARCHITECTURAL DECISIONS

Recap: Software Architecture Definitions

There are numerous definitions of SW architecture

See: http://www.sei.cmu.edu/architecture/definitions.html

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

– Bass, Clements, Kazman

Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled.

– Eoin Woods

Architectural Decisions

- Architectural decisions have a **long-term** and **significant influence** on the system
- Architecture concerns those decisions that are **costly to change**
- Architecture is the **result of a series of decisions**
  - Architect must decide which functional or non-functional requirements are **prioritized**
  - Architecture is always a **compromise**

The Vision

Architectural knowledge is shared and reused

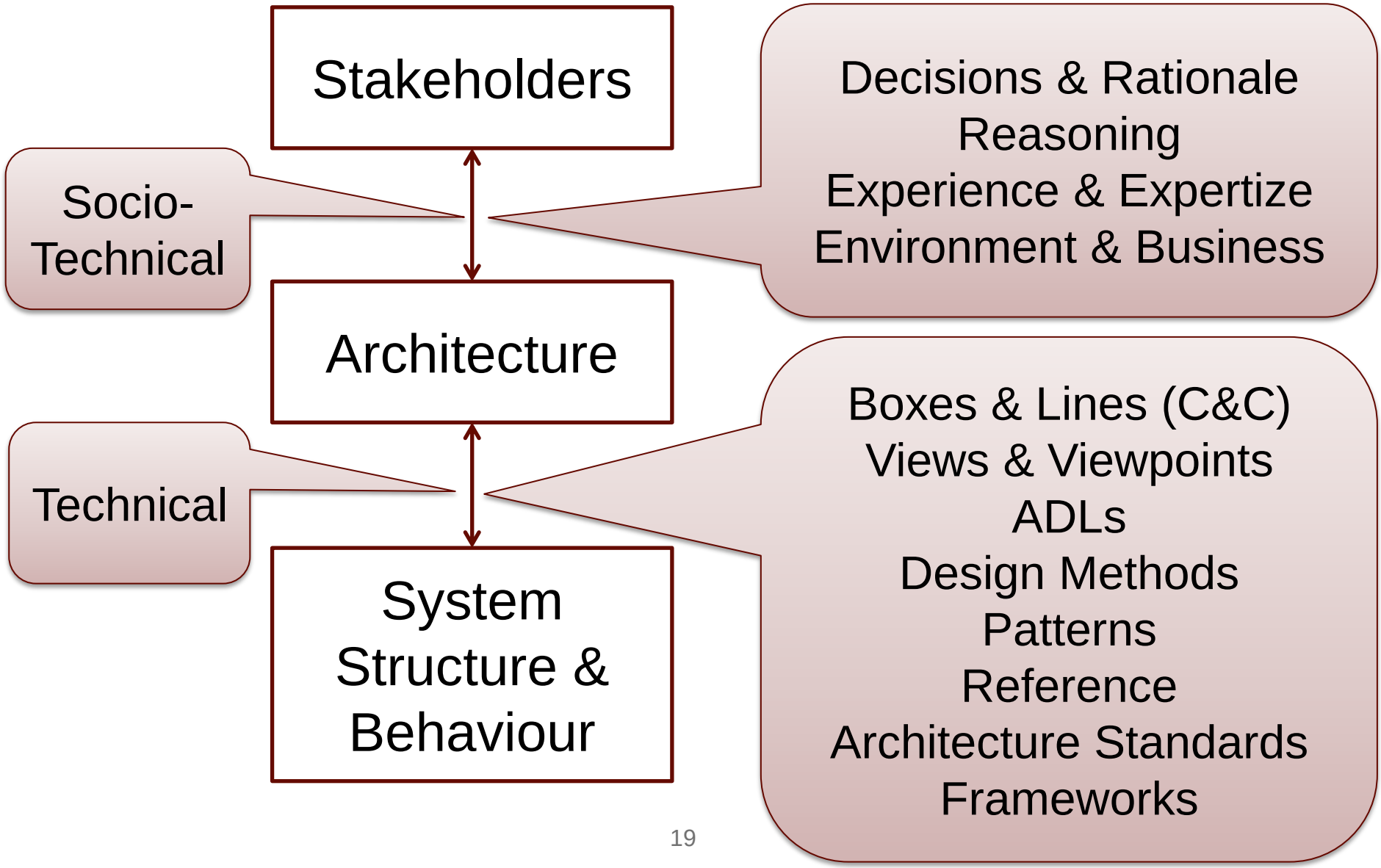
Decisions as first class entities

Paradigm Shift

2 perspectives to contrast old and new paradigm:

1. Relating architecture to systems vs. stakeholders
2. The solution vs. how we got there

1 – Relating architecture to systems vs. stakeholders



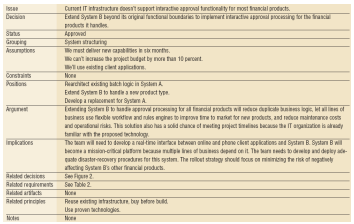
2 – The solution vs. how we got there

Design alternatives, decisions, rationale

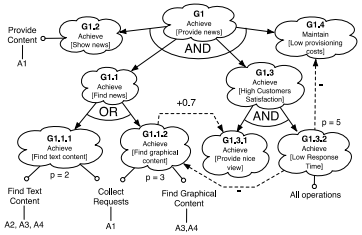
Expertise, skills, previous successes, patterns, best practices

Goals, constraints, concerns, requirements, assumptions, risks

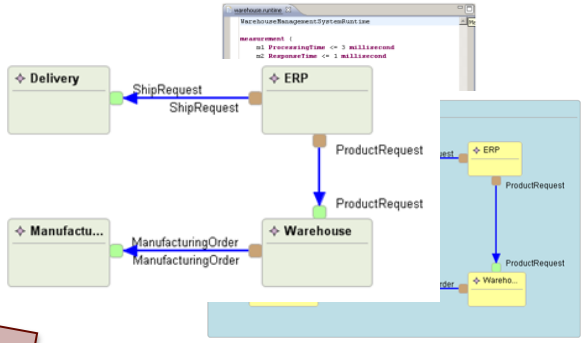
Context, business, environment, technologies, market



Issue	Current IT infrastructure doesn't support interactive approval functionality for most financial products.
Decision	Extend System B beyond its original functional boundaries to implement interactive approval processing for the financial products it handles.
Status	Approved
Grouping	System structuring
Assumptions	We must deliver new capabilities in six months. We can't increase the project budget by more than 10 percent. We'll use existing client applications.
Constraints	None
Positions	Rearchitect existing batch logic in System A. Extend System B to handle a new product type. Develop a replacement for System A.
Argument	Extending System B to handle approval processing for all financial products will reduce duplicate business logic, let all lines of business use flexible workflow and rules engines to improve time to market for new products, and reduce maintenance costs and operational risks. This solution also has a solid chance of meeting project timelines because the IT organization is already familiar with the proposed technology.
Implications	The team will need to develop a real-time interface between online and phone client applications and System B. System B will become a mission-critical platform because multiple lines of business depend on it. The team needs to develop and deploy adequate disaster-recovery procedures for this system. The rollout strategy should focus on minimizing the risk of negatively affecting System B's other financial products.
Related decisions	See Figure 2.
Related requirements	See Table 2.
Related artifacts	None
Related principles	Reuse existing infrastructure, buy before build. Use proven technologies.
Notes	None



Architecture views & models



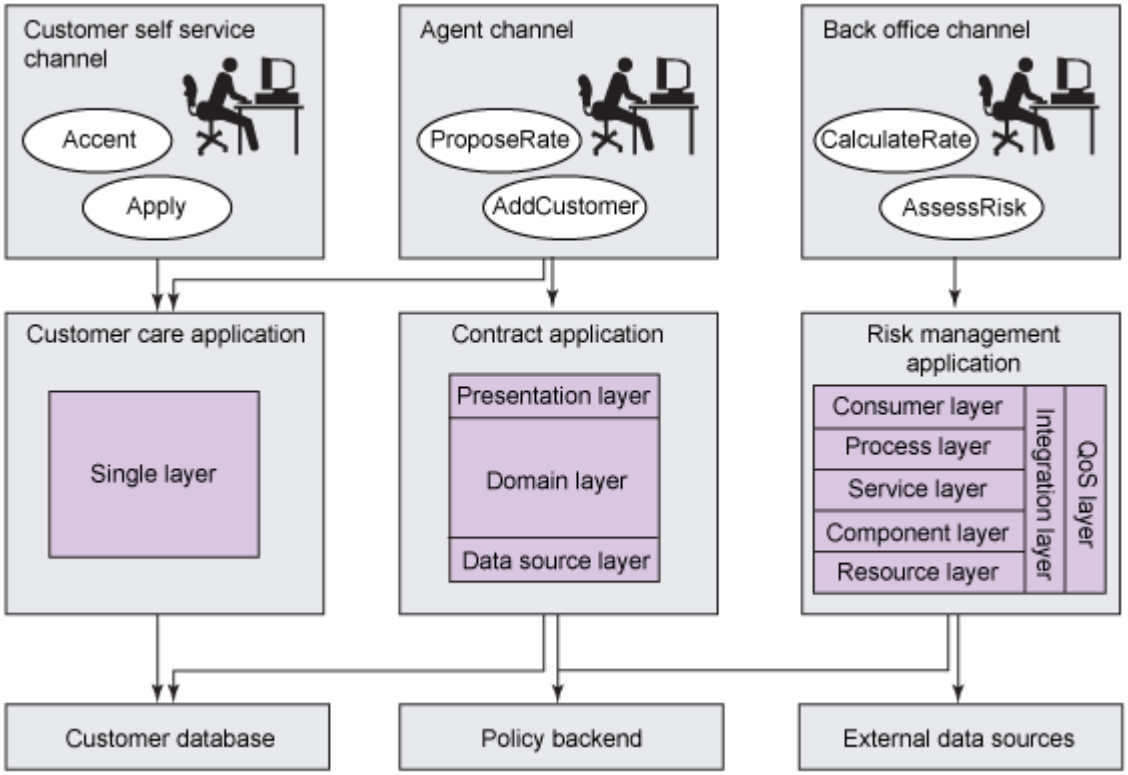
ARCHITECTURAL DECISION EXAMPLE

Example: Deciding for a Layering Scheme

- Selecting an overall **logical layering scheme** for a system under construction qualifies as an architectural decision
  - Prior Decision: Selecting the **layers pattern** is an architectural decision driven by the desire for organization and flexibility
- But how to organize the layers? Some **alternatives** are:
  - Single Layer
  - Presentation/Domain/Data Source Layering
  - More complex SOA Layering Scheme

Example: Deciding for a Layering Scheme

- Three applications in one system; **three different decision outcomes**
- Problem: **Rationale** of the decisions is not documented in this figure



ARCHITECTURAL DECISION MODELING

Problem: Decisions Get Lost in Architectural Models

- Architectural views and architecture modeling do **not focus on decisions**
  - Decisions are not documented explicitly but are **implicit** in the models the architect builds
- Developers want **clear guidance on how to proceed with a design**
- Customers want a clear understanding of the architecture in terms of **required environmental changes** and **meeting their business needs**
- Other architects want a clear understanding of the **architecture's key aspects**, including the **rationale** and **options the original architect considered**

Solution: Decision Modeling

- Model architectural decision explicitly using a **decision template** or **meta-model**
- Capture the **key design issues** and the **rationale** behind the decision
  - **Conscious design decisions** concerning the architecture
  - Consider the **impact on nonfunctional requirements** and **quality factors**

Example: Decision Template by Tyree/Ackerman

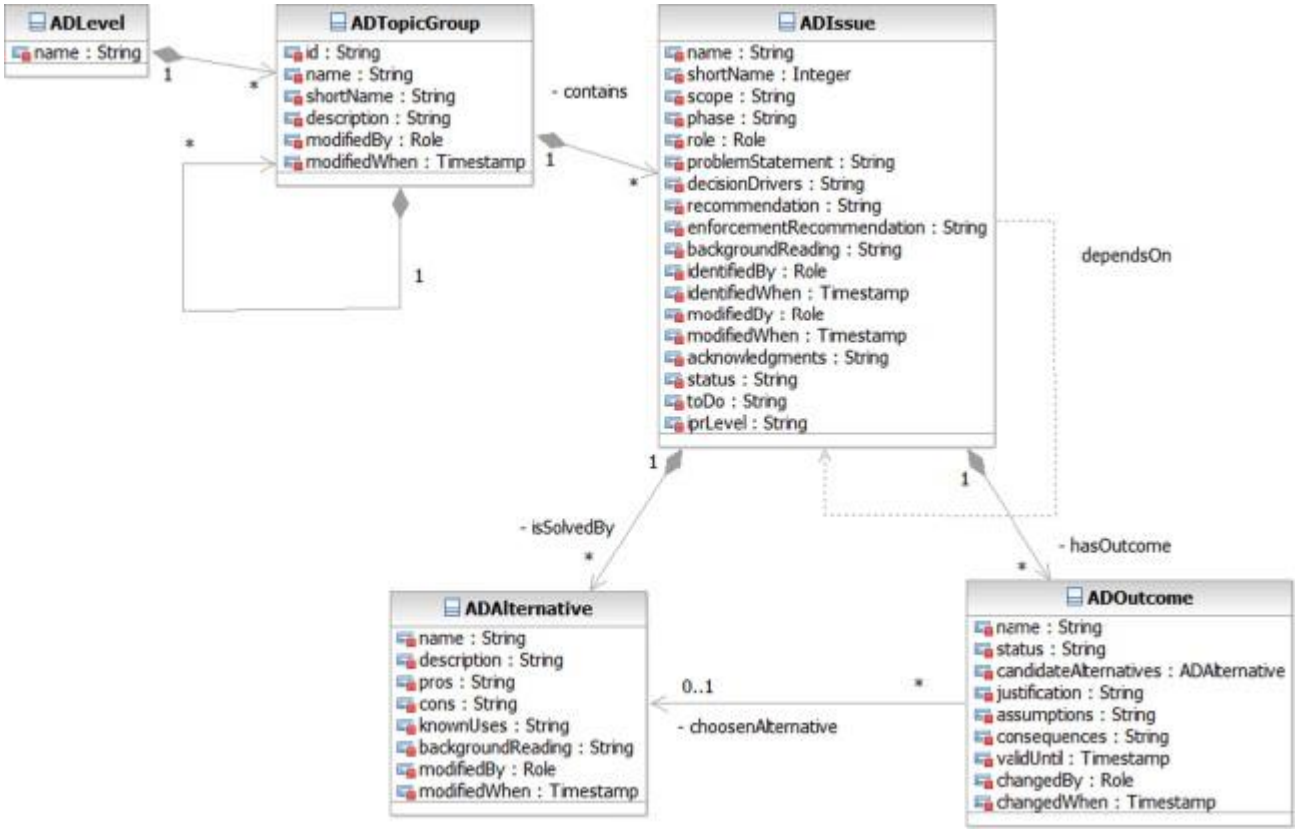
Issue	Describe the architectural design issue you're addressing, leaving no questions about why you're addressing this issue now. Following a minimalist approach, address and document only the issues that need addressing at various points in the life cycle.
Decision	Clearly state the architecture's direction—that is, the position you've selected.
Status	The decision's status, such as pending, decided, or approved.
Group	You can use a simple grouping—such as integration, presentation, data, and so on—to help organize the set of decisions. You could also use a more sophisticated architecture ontology, such as John Kyaruzi and Jan van Katwijk's, which includes more abstract categories such as event, calendar, and location.[8] For example, using this ontology, you'd group decisions that deal with occurrences where the system requires information under event.
Assumptions	Clearly describe the underlying assumptions in the environment in which you're making the decision—cost, schedule, technology, and so on. Note that environmental constraints (such as accepted technology standards, enterprise architecture, commonly employed patterns, and so on) might limit the alternatives you consider.
Constraints	Capture any additional constraints to the environment that the chosen alternative (the decision) might pose.
Positions	List the positions (viable options or alternatives) you considered. These often require long explanations, sometimes even models and diagrams. This isn't an exhaustive list. However, you don't want to hear the question "Did you think about … ?" during a final review; this leads to loss of credibility and questioning of other architectural decisions. This section also helps ensure that you heard others' opinions; explicitly stating other opinions helps enroll their advocates in your decision.
Argument	Outline why you selected a position, including items such as implementation cost, total ownership cost, time to market, and required development resources' availability. This is probably as important as the decision itself.
Implications	A decision comes with many implications, as the REMAP metamodel denotes. For example, a decision might introduce a need to make other decisions, create new requirements, or modify existing requirements; pose additional constraints to the environment; require renegotiating scope or schedule with customers; or require additional staff training. Clearly understanding and stating your decision's implications can be very effective in gaining buy-in and creating a roadmap for architecture execution.
Related decisions	It's obvious that many decisions are related; you can list them here. However, we've found that in practice, a traceability matrix, decision trees, or metamodels are more useful. Metamodels are useful for showing complex relationships diagrammatically (such as Rose models).
Related requirements	Decisions should be business driven. To show accountability, explicitly map your decisions to the objectives or requirements. You can enumerate these related requirements here, but we've found it more convenient to reference a traceability matrix. You can assess each architecture decision's contribution to meeting each requirement, and then assess how well the requirement is met across all decisions. If a decision doesn't contribute to meeting a requirement, don't make that decision.
Related artifacts	List the related architecture, design, or scope documents that this decision impacts.
Related principles	If the enterprise has an agreed-upon set of principles, make sure the decision is consistent with one or more of them. This helps ensure alignment along domains or systems.
Notes	Because the decision-making process can take weeks, we've found it useful to capture notes and issues that the team discusses during the socialization process.

From: J. Tyree, A. Akerman, Architecture Decisions: Demystifying Architecture, IEEE Software, vol. 22, no. 2, 2005

Example: Decision Modeled using the Template by Tyree/Ackerman

Issue	Current IT infrastructure doesn't support interactive approval functionality for most financial products.
Decision	Extend System B beyond its original functional boundaries to implement interactive approval processing for the financial products it handles.
Status	Approved
Grouping	System structuring
Assumptions	We must deliver new capabilities in six months. We can't increase the project budget by more than 10 percent. We'll use existing client applications.
Constraints	None
Positions	Rearchitect existing batch logic in System A. Extend System B to handle a new product type. Develop a replacement for System A.
Argument	Extending System B to handle approval processing for all financial products will reduce duplicate business logic, let all lines of business use flexible workflow and rules engines to improve time to market for new products, and reduce maintenance costs and operational risks. This solution also has a solid chance of meeting project timelines because the IT organization is already familiar with the proposed technology.
Implications	The team will need to develop a real-time interface between online and phone client applications and System B. System B will become a mission-critical platform because multiple lines of business depend on it. The team needs to develop and deploy adequate disaster-recovery procedures for this system. The rollout strategy should focus on minimizing the risk of negatively affecting System B's other financial products.
Related decisions	See Figure 2.
Related requirements	See Table 2.
Related artifacts	None
Related principles	Reuse existing infrastructure, buy before build. Use proven technologies.
Notes	None

Meta model for decision capturing and modeling by Zimmermann et al.



Dependencies between ADs and ADAlternatives

- **influences:** This dependency type expresses a generic dependency. It is bidirectional
- **decomposesInto:** A complex design problem is split into several smaller, more manageable ones for a divide-and-conquer strategy to problem solving.
- **refinedBy:** This dependency type describes dependencies between architectural decisions located in different levels of abstraction and refinement, for example, conceptual, technology, or vendor asset level. AD levels will be discussed in more detail in the next part of this series.
- **forces, isIncompatibleWith:** The selection of an alternative in one decision might force the selection of another one in another decision. A slightly weaker statement is that two alternatives might be incompatible with each other.
- **triggers:** As soon as architectural decision A is decided, architectural decision B can be decided. This influence type is used for temporal dependencies.
- **prunes:** If architectural decision A is decided for a certain alternative, architectural decision B is no longer relevant and can be marked as such.

Refinement Levels of Architectural Decisions

- *Stage 1*: Executive decisions, requirements analysis.
- *Stage 2*: Conceptual decisions including selection of architectural patterns and key technology choices.
- *Stage 3*: Detailed technology decisions, design patterns as architecture alternatives.
- *Stage 4*: Vendor asset level decisions and selection of implementation, deployment, and test patterns.

Sample decision with ADOutcome information

AD shortname	Exe-01	AD name	Logical Layering Schema		
Topic hierarchy	InsuranceSoaProject - BusinessExecutiveLevel - ExecutiveDecisions - KeyTechnologyProcessAndToolDecisions				
Scope	Project	Phase	Solution outline	Role	Lead architect
Problem statement	Which logical layering scheme, possibly defined in a public or internal reference architecture, should frame the design work? Architectural consistency starts with speaking one language—and hearing the same things.				
Decision drivers	Software engineering method and viewpoint catalog chosen. Standards for industry or application genre, and selected reference architecture. Existing client assets and enterprise architecture efforts. Vendor preferences.				
Alternatives	[1] Single layer [2] Presentation Domain Data Source Layering, introduced by Brown and Fowler [3] Service layering scheme in "Enterprise SOA" [4] IBM SOA Solution Stack (S3), superseding 5+2 SOMA layers [5] Other SOA layering schemes, such as from CBDI or SOA in Practice [6] Not applicable				
Recommendation	The S3 layering and the service type taxonomies from the Enterprise SOA and SOA in Practice are useful. The S3 layering is intuitive and can be mapped to older layering schemes, such as Presentation Domain Data Source Layering.				
Decision Outcomes	Decision instance: SOA modernization project, risk management application subproject				
	Status	decided by ArchieTekt on 2008-08-05 16:48:59.951000 Valid until July 31, 2009			
	Chosen alternative	[4] IBM SOA Solution Stack (S3), superseding 5+2 SOMA layers			
	Justification	The recommendation works for us. It's an SOA project, following the SOMA method, which also works with the S3 layers. Not aware of any mature insurance industry standard.			
	Consequences	Will need to define a mapping from "Enterprise SOA" scheme and the presentation, domain, and data source layers to S3.			
	Assumptions	For now, will only use the original 5+2 layers. Are going forward with IBM GBS.			
Background reading	"Design an SOA solution using a reference architecture" introduces the S3 layering. Layering pattern is explained in depth in pattern literature (in POSA series, for example).				
Related decisions	is influenced by Exe-00 PrimaryArchitecturalStyle influences Gov-01 TerminologyHardening influences Gov-02 StandardsAdoption				
Editorial information	Acknowledgments: the entire SOA project team was involved ModifiedWhen 2008-08-05 16:50:46.394000 Status: published, to be reviewed semi-annually ToDo: keep up to date				

(Some) Practical Problems of Decision Modeling

- Decision Documentation Effort
- Understanding the Links between Decisions and other Software Artifacts
- Avoiding Repetitive Effort
- Dealing with Invalid or Bad Justifications

JUSTIFICATION GUIDELINES

Guidelines

- Justifications must provide precise rationale and should avoid common sense statements, truisms, and killer phrases.
- Justifications should highlight the decision drivers and explain why recommendations found in guidance models or other trusted sources were followed or not.
- Justifications must refer to actual project requirements and not just generic background information found in the literature.

Example: Two example justifications

Alternative A best meets user expectations and functional requirements as documented in user stories, use cases, and business process model.

End users want it, but there is no evidence for a pressing business need.

Relevant Literature and Sources

- Zimmermann et al. Modeling and sharing architectural decisions, Part 1: Concepts. http://www.ibm.com/developerworks/architecture/library/ar-knowwiki1/, 2008.
- Vogel, O. and Arnold, I. and Chughtai, A. and Ihler, E. and Mehlig, U. and Neumann, T. and Voelter, M. and Zdun, Uwe. Software Architektur - Grundlagen, Konzepte, Praxis. Elsevier/Spektrum Verlag, http://www.software-architektur-buch.de/, 2005.
- J. Tyree and A. Ackerman, "Architecture Decisions: Demystifying Architecture," IEEE Software, Mar./Apr. 2005, pp. 19–27.
- N. Harrison, P. Avgeriou, and U. Zdun. Using Patterns to Capture Architectural Decisions. *IEEE Software*, pages 38-45, IEEE, July/Aug., 2007.
- Olaf Zimmermann. Recurring Architectural Decisions: A Context-Specific Guide through SOA and Cloud Design. Saturn 2010 Tutorial. See: http://soadecisions.org/

Many thanks for your attention!

Uwe Zdun

Software Architecture
Faculty of Computer Science
University of Vienna
http://cs.univie.ac.at/swa