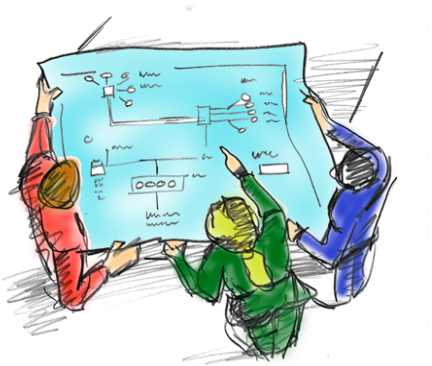


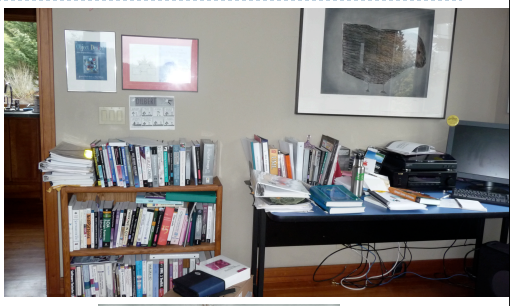
# Lessons Learned From Architecture Reviews

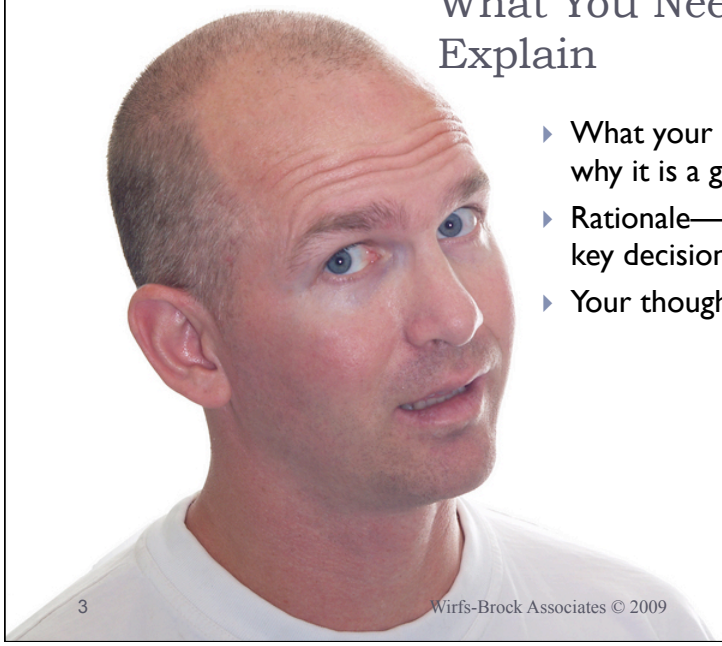
Rebecca Wirfs-Brock




## Two Perspectives

- An outsider evaluating strengths and weakness of products, enterprise applications, and systems
- As an insider with recognized communication skills





## What You Need to Explain

- What your design is and why it is a good solution
- Rationale—why you made a key decision
- Your thought process




## Collaborate

To work together, especially in a joint intellectual effort

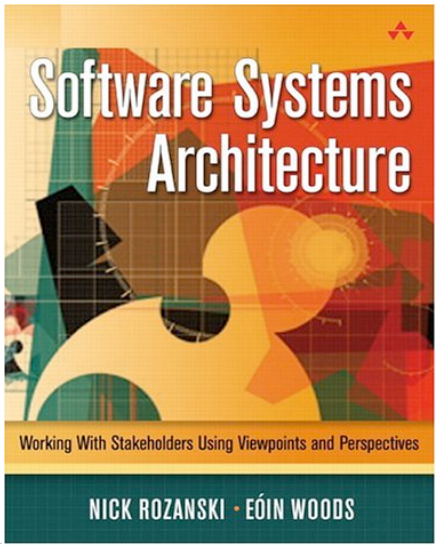
## Collaborate

To cooperate treasonably, as with an enemy occupation force



## Explaining and Defending Architecture

- Decision
- Constraints
- Alternatives: Options considered and reasons for ruling them out
- Effects: What the does decision imply
- Evidence: Confirmation the decision is good



# Explaining to a Constructive Reviewer

- Design idea
- Requirements
- Advantages
- Disadvantages
- Limitations
- Design notes
- Issues, uncertainties



**Resource Managers**

A ResourceManager is responsible for relating Resources to the external applications that own those Resources. Therefore, a ResourceManager identifies the external application that should be the owner of a newly created Resource and a ResourceManager is able to locate an existing Resource based on the names by which the Resource is known in external applications.

**Business Requirement**

A ResourceManager has to assign a new Resource to an external application that should own the data for that Resource. For example, when a new Account is created, the AccountManager might assign it to a specific billing system based on the billing address of the account.

**Design Notes**

The prototype has four ResourceManagers:

- AccountManager
- CustomerManager
- ProductManager
- ServiceManager

Each ResourceManager creates the corresponding type of Resource for a given ExternalReference and PropertyList. For example, the AccountManager creates Accounts and the ProductManager creates Products. After creating the Resource, the ResourceManager maintains a mapping of ExternalReference to Resource. For example, any object that needs to resolve an ExternalReference to an Account will ask the AccountManager to get the Account for a given ExternalReference.

**Advantages**

A ResourceManager encapsulates the association of ExternalReferences to Resources, so that the mapping is centralized in one location.

A ResourceManager encapsulates the business rules that associate Resources with external applications, so that the rules are localized in one central location.

**Limitations**

In the prototype, we assume that an ExternalReference maps to a unique Resource. In the future, it might be useful to locate a set of Resources that match a particular set of Qualifiers.

Each Resource must be fully owned by a single external application. It is not possible for some of a Resource's data to be owned by one application, while the rest of that Resource's data is owned by another application. For example, in the prototype, we have a service definition called Calling Card Plastic. It builds two tasks: Emboss Calling Card and Mail Calling Card. For the prototype, both of these tasks are being handled by the Manual adapter but the chances of both of these tasks being accomplished by the same system is not likely. Either the Task Strategy or the resource needs to be able to point to multiple adapters for getting work accomplished. This however does not mean that one system does not own the data concerning a given service.

**Issues**

An ExternalReferenceManager provides a local cache of the mapping of ExternalReferences to Resources. Some testing is required to determine whether such a cache provides a significant enhancement in performance compared to accessing a ResourceManager directly. It is also necessary to verify that the cache cannot get out of synch with the mapping held by the ResourceManager.

8

## Advice and Its Impact

A “triage” mentality can help you as a reviewer focus your energy and efforts




## Advice: Key Findings +

Recommendations Suggestions Observations

## Advice Example

**Key Finding:** The business object model and logical data model need to be developed concurrently with the technical architecture. The initial Visio diagram of the business object diagram is just a start. It simply names objects and shows associations between them.

**Recommendation:** *Add detail to the preliminary business object model: define key concepts, relationships between them, major responsibilities, and their lifecycles.*

**Recommendation:** *Assign someone to work on the logical database and schema design.*

A business object model is different than a logical database design, and they both are needed. In the current products, the logical database design is non-existent. Any original intent behind the current physical database design has been obscured by years of product customizations. Development of logical data model is a big effort that shouldn't lag other elements of the architecture.

**Key Finding:** The project needs to produce architecture and design documents which have value to developers and new staff.

Much of the design for the existing system is "in the heads" of the developer or buried in code that is poorly understood. There isn't a culture for producing design documentation. Developers are used to reading code, not design documents.

**Recommendation:** *Develop design documentation and models for key concepts and design ideas.*

**Suggestion:** *Consider documenting the existing xxx product architecture as an aid to fleshing out the integrated functional architecture.*




## Observations Example

Observation #2: Database locking or single-threaded access to the Provisionable Database might be a perceived rather than a real problem.

We say this for a number of reasons. It is our understanding that the length of time any process should hold a lock on the Provisioning Database should be very small. In fact, the design intent is for ManagementRequests to be implemented so as to acquire a lock for the smallest unit of work. So the design intent is to: do a small amount of work, unlock, then try to get another lock to do some more work, etc. So, if they are designed right, a complex Management Request that affects multiple provisionable entities will be broken down into multiple transactions. *In theory, there is an opportunity* that xxx queries and updates could be interleaved with any longstanding xxx command, for example, that is "in progress."

Furthermore, if a process needs to affect or read the status of a number of entities in the Provisionable Database, it can perform multiple operations within the same transaction. So, if xxx needs to perform a number of queries before it can determine what resources to provision, database access should be guaranteed (and be relatively fast).

## Lesson: Comment on Good Decisions Too

Observed Best Practice: Use of interfaces.

Java interfaces are being used wherever possible with Spring's dependency injection, so that XXX and YYY objects depend on an API, rather than using implementation classes directly. This allows the implementation to change more easily without affecting client objects. It also provides a mechanism for supplying mock objects for unit tests.

Observed Best Practice: Use of JMS.

The Java Message Service (JMS) API is being used to initiate heavyweight processes. Even though the asynchronous services are currently running in the application server JVM, use of JMS for calling these services, such as attachment processing, will make it easy to distribute those services to separate JVMs at a future time.

In summary, we are impressed by the thoughtful discussions we had with the architects. They clearly articulated rationale behind their technology choices and the integrating new technologies. The team seems to have made significant progress and we expect them to continue. From our perspective, no unresolved issue seems insurmountable.




## Lesson: Scale the Review to the Size of the Project

"As you gain experience, consider whether you might want to organize questions according to whether a review is "bronze", "silver", or "gold" and/or whether it is early or late in the process."

## Lesson: Agile Development Has Architectural Impacts

- Enterprise Architecture can be accessed for agility:
  - Does it support automation of acceptance tests? How much automation is possible at what cost?
  - How to encode domain rules and knowledge to be easily testable (potentially by analysts rather than developers)?
  - How easy is it configure? Reconfigure?
  - Can it be delivered and deployed incrementally?




## Agility Assessment: A COTS Component

"Testing practices and automatic improvements can and should be made, but they are all feasible. Unit testing suggestions:

- Write xxx formulas in a modular decomposed fashion.
- Write tests in Java to call formulas and/or sub-formulas to see if they are correctly implemented, perform, and work against correct tabular data.
- Exploit ability to run tests locally and remotely.
- Acceptance testing should be done via Java with the system under test either locally or remote. These could be driven from Fitnesse when that makes sense and also from scripts."

## Agility Assessment Example: SOA

"Build out SOA patterns, interfaces, and collaborations incrementally:

- Start sending messages between components as soon as possible. Early on, the only messages available may be "heartbeat" and similar messages. Use those messages to work out "baseline" integration problems.
- Start performance and load testing soon after. Over time this testing will reveal emerging problems so they can be addressed early. Also, implementing this testing early will avoid having to add it under pressure. Constant monitoring will provide useful feedback on optimal service partitioning, and reducing excessive message passing.
- Early on, select and implement features that work like tracer bullets through the entire system, touching as many of the major components as possible.
- Flesh out the details of orchestrated service design patterns with simple, realistic and concrete scenarios. Then, if desired, write up more generic patterns. Documentation should lag (not drive) proven practice."




## Lesson: Beware of the Technical Stack

## Lesson: Merging Existing Systems is really^Hard

"Many hidden requirements are in the heads of support or buried in custom code.

There is no migration strategy.

The core of the architecture team is in CA while needed expertise is in P…"




## Lesson: Risks Compound

"While no one particular technology choice stands out as being highly risky, the overall project risk is high due to the fact that the team is using new technologies, building an extensible platform, and implementing a new software and system architecture

While there is significant technical risk, we feel the architecture team has been judicious in their technology selection. The technology is not unproven. The challenge is that the team needs to acquire expertise and work through detailed design issues."

## Lesson: Get the Right People Involved

"We suggest that several, realistic scenarios be written down, and agreed upon as representative by product marketing."




## Focused Questions

"There are separate sets of questions for each reviewer, as well as a set of questions to be considered by all. These questions are intended as a guide for reviewing. However, we welcome all comments and suggestions."

**All Reviewers**

1. Should we have a framework running at each installation? Or should it be one system? What about federated systems?

2. What kind of problems will we encounter trying to build the relationships between resource objects? What about strategies for refreshing the framework's view of the data in applications?

3. Should we keep some functionality in the framework (e.g. task management) or would we be better to push it out to external systems, even if we write them?

4. Do you think that tasks are structured such that failure recovery can be worked in or are there design flaws? Where should recovery occur? What about canceling?

5. Is our document consistent? What do you find useful? Is there too much or too little?

**Brian**

1. Critique our current transaction boundaries between queues.

2. What is the cost of passing too much data? If we use CORBA? If we use JavaBeans? Adapters only referent to external references. Was this a good decision?

3. Comment on properties and how they might be constructed using a tool.

**Order App Vendor**

1. What limit should be set on how many threads run in one process?

2. Comment on our product catalog design, and your thoughts about what you keep track of products.

## Lesson: Ask the Right Questions





## Lesson: Ask the Right Questions

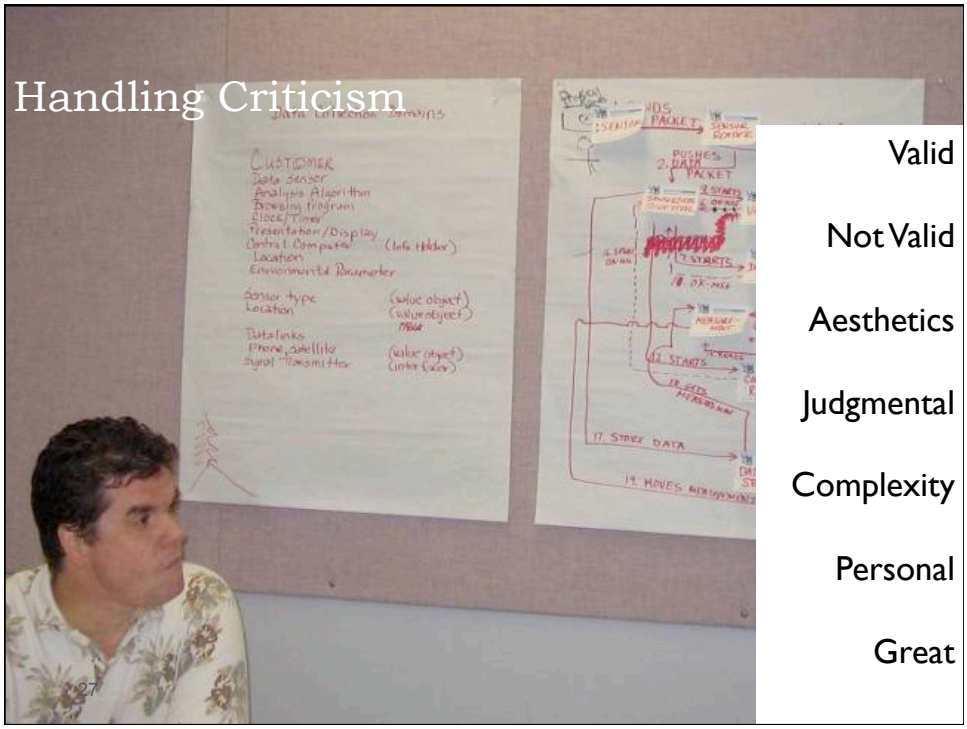
## Probing Questions

- Evaluation…how good do you think it will be
- Accuracy…how did you come up with those numbers
- Completeness…is that all
- Relevance…does this apply here
- Purpose…why did you suggest that
- Extension…tell me more




## Clarifying Questions

- Get them to think:
  - Why do you say that?
  - What exactly do you mean?
  - How does this relate to what we discussed earlier?
  - Can you give me an example?
  - Are you saying ... or ... ?
  - Can you restate your concern?




| Type of criticism | Characteristics of criticism | Appropriate Tactic |
|---|---|---|
| Valid | Info indicates a flaw or weakness in idea | Refine your idea—but don't lose its advantages |
| Not valid | Clear misfit between your idea and criticism | Improve your ability to explain |
| Aesthetic | Negative reaction reflecting form vs. substance | Acknowledge, defuse by explaining your position |
| Judgmental | Negative reaction with/without enough info to indicate a problem | Ask critic for more specific info |
| Complexity | Value judgment with implicit assumption that a simpler solution exists | Explore. May need to educate about inherent complexity |
| Great! | May or may not be judgmental/ specific | Optionally, probe behind the praise |

## Graham's Disagreement Hierarchy

- 0 Name calling
- 1 Ad hominem
- 2 Responding to tone
- 3 Contradiction
- 4 Counterargument
- 5 Refutation
- 6 Refuting the central point
- www.paulgraham.com/disagree.html

## Lesson: Increase Information Availability

- People decide based on what they remember
- To increase information availability make it
  - Recent
  - Vivid
  - Easy to imagine
- To decrease, make it
  - Complex
  - Uncomfortable

## Presenting Tradeoffs: Version 1

**Option 1: One Large Transaction**

- – Can't handle optimistic lock exception
- + Can batch updates
- + Can handle validation business logic
- – Can only rollback entire transaction

**Option 2: Split into many smaller transactions**

- – Can't batch updates
- – Slower performance
- + One set of code
- + Partial failure easier
- + Rollback code could update db
- – Cannot use first level Hibernate cache
- + Could run small transactions in parallel but…
  - – Added complexity getting partial results and setting up txns




## Presenting Tradeoffs: Version 2

**Option 1: One Large Batch Transaction**

- + 20 times faster than split transactions
- + Can use Hibernate cache
- – Can only rollback entire transaction

**Option 2: Split into many smaller transactions**

- – Slow performance
- – Can't batch updates
- – Cannot use first level Hibernate cache
- + Partial failure possible
- + Optimization possible

  Could run small transactions in parallel but…adds complexity of handling partial results and setting up txns

## Presenting Tradeoffs: Version 3

**Option 1: One Large Batch Transaction**

- \+ 20X faster than split transactions
- \+ Simpler batch code
- – Can only rollback entire transaction
- \+ Can use Hibernate cache

**Bottom line: Significantly greater batch performance with simple txn logic.**

**Option 2: Split into many smaller transactions**

- – 20x slower
- \+ Optimization possible by parallelizing txns
- – Optimization complex
- ± Can support partial failure (but recovery actions unclear)
- – Cannot use first level Hibernate cache

**Bottom line: Performance is significantly slower. Some optimization possible with extra dev. time**




## Lesson: Recognize Cognitive Biases

- Cognitive biases are distortions in how people naturally tend to process and interpret information
- Not every one shares the same biases
- They cause us to "react blindly" rather than "think and behave logically"

## Contrast Effect

People can't avoid comparing items against each other rather than against a fixed standard

35

## Confirmation Bias

- The tendency to
  - Seek and interpret information in a way that confirms preconceptions
  - Avoid things that will disconfirm beliefs


36

## Hyperbolic Discounting

- People prefer smaller, more immediate rewards over larger rewards promised in the future
- Tough to counteract





## Common appeals…

- Emotion
- Fear
- Novelty
- Standard practice
- Authority

## Persuasion

- To be persuaded a person must:
  - Listen to your advice
  - Compare to previously held views
  - Reconcile it with contrary ones
  - Agree with it