Patterns in Architecture

"Building Quality Systems"

Joseph W. Yoder
The Refactory, Inc.
joe@refactory.com
http://www.refactory.com
http://www.joeyoder.com

Joe's Bio

Joseph Yoder began working with software in the mid 1980's, and has developed robust systems for many companies and organizations with global impact. He is a founder and principle of The Refactory, Inc., a company focused on software architecture, design, implementation, consulting and mentoring of all facets of software development. Joe is an international speaker and pattern author on software architecture, design, and implementation. Joseph specializes in Object-Oriented Analysis and Design, C#, Java, Smalltalk, Patterns, Agile Methods, Adaptable Systems, Refactoring, Reuse, and Frameworks. He has mentored developers on many types of software applications.

Joseph Yoder is a longstanding member and vice char of the board of The Hillside Group, a group dedicated to improving the quality of software development. Joseph has chaired the Pattern Languages of Programming Conferences (PLoP), as well as given many tutorials and talks at conferences such as JAOO, QCon, OOPSLA and ECOOP. He is co-author of the Big Ball of Mud pattern, which illuminated many fallacies in the approach to software architecture.

Joseph Yoder currently resides in Urbana, Illinois. He teaches Agile Methods such as XP, Design Patterns, Object Design, Refactoring, and Testing in industrial settings and mentors developers on these concepts. Joseph currently oversees a team of developers who have constructed an order fulfillment system based on enterprise architecture using the .NET environment. His other recent work includes working in both the Java and .NET environments, and deploying Domain-Specific Languages for clients. Joe thinks software is still too hard to change. He wants to do something about this and believes that putting the ability to change software in the hands of the people with the knowledge to change it is a promising avenue to solve this problem.

What is Architecture?

Software Architecture is commonly viewed as the structure(s) of the software which includes the software components, the externally visible properties of those components, and the relationships between them.

Lot’s of work has been examining Architectural styles which include common patterns seen in different types of architecture

Still a mix of art and science.

Software Architecture

Coarse grain components and how they interact

Considered Larger in scale than algorithms & code

Defines what “someone” sees as:
“the structure of the system”

Described with boxes, bubbles, lines, charts, layers…

Sketchy map of the system…

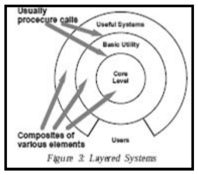
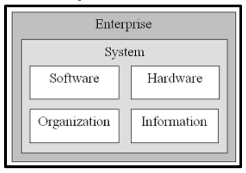


Figure 3: Layered Systems

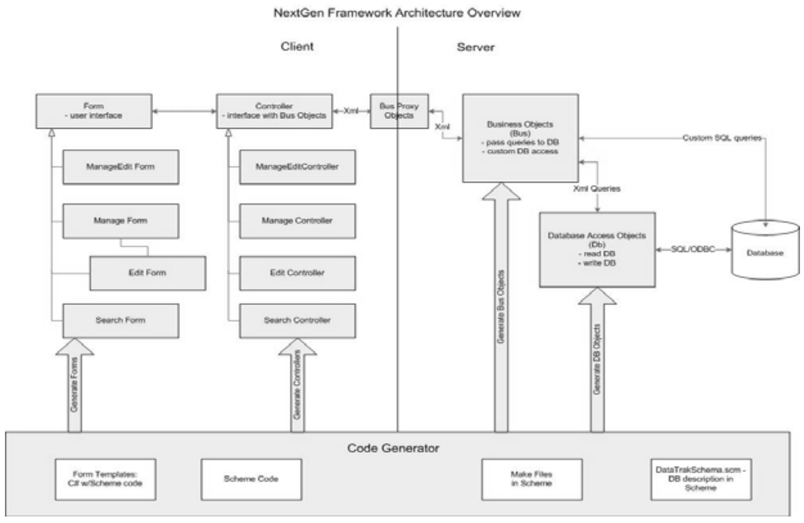


Software Architecture Views

A building architect might create wiring diagrams, floor plans, and elevations to describe different facets of a building to its different stakeholders (electricians, owners, planning officials).

A software architect might create physical and security views of an software system for the stakeholders who have concerns related to these aspects.

Software Architecture Different Views…



What Is Our Reality?

Are we only talking about architectural visions?

This is what we want the system to look like?

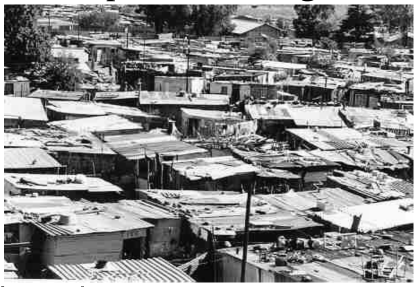
What is really under the hood?

Big Ball of Mud

Alias: Shantytown, Spaghetti Code

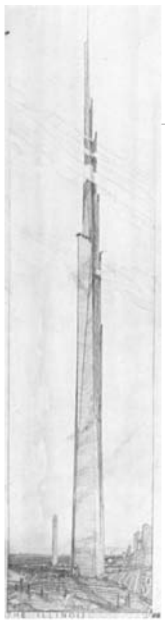
A BIG BALL OF MUD is haphazardly structured, sprawling, sloppy, duct-tape and bailing wire, spaghetti code jungle.

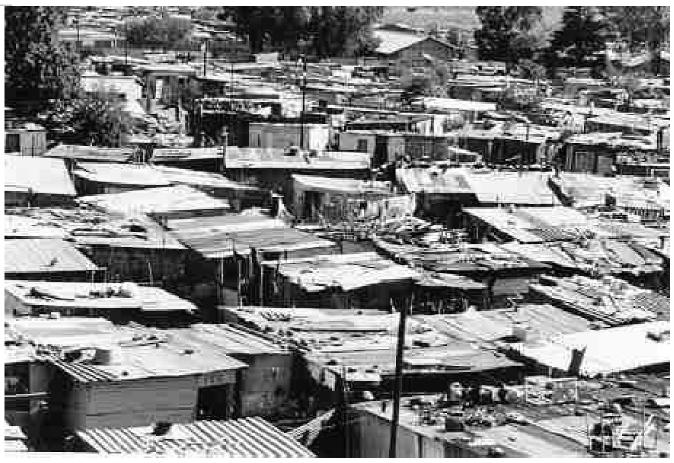
The de-facto standard software architecture. Why is the gap between what we **preach** and what we **practice** so large?



We preach we want to build high quality systems but why are BBoMs so prevalent?

Big Ball of Mud





Why do we have BBoM???

Lack of Upfront Design? Throwaway Code?

Late changes to the requirements of the system?

Continuously Evolving the Architecture?

Piecemeal Growth? Keep it Working?

Focus on Process rather than Architecture?

Isn’t this mostly what Agile preaches?

Maybe there is a good reason for it?

Worse is Better (examples)

Betamax vs VHS Format

- Why did VHS win?
- Betamax was arguably a better format

Macintosh vs Windows

- Mac was easier to use
- Far superior in many ways

MS Word/Publisher vs FrameMaker

- Lot's of people use word
- FrameMaker is better for books

The Quality Goes In

BUILT BETTER
...to last longer!
Every Zenith portable TV is Handcrafted – built better to last longer. There are no printed circuits. No production shortcuts. Every connection is carefully handwired. This kind of dedication to quality has made Zenith America's largest selling TV. It is one of the important reasons why Zenith TV gives you finer performance. Fewer service problems. Greater operating dependability. And a sharper, clearer picture, year after year. Don't settle for less than Zenith—the Handcrafted TV.
ZENITH
The quality goes in before the name goes on

Quality

- **Quality Definition:** a peculiar and essential character or nature, an inherent feature or property, a degree of excellence or grade, a distinguishing attribute or characteristic

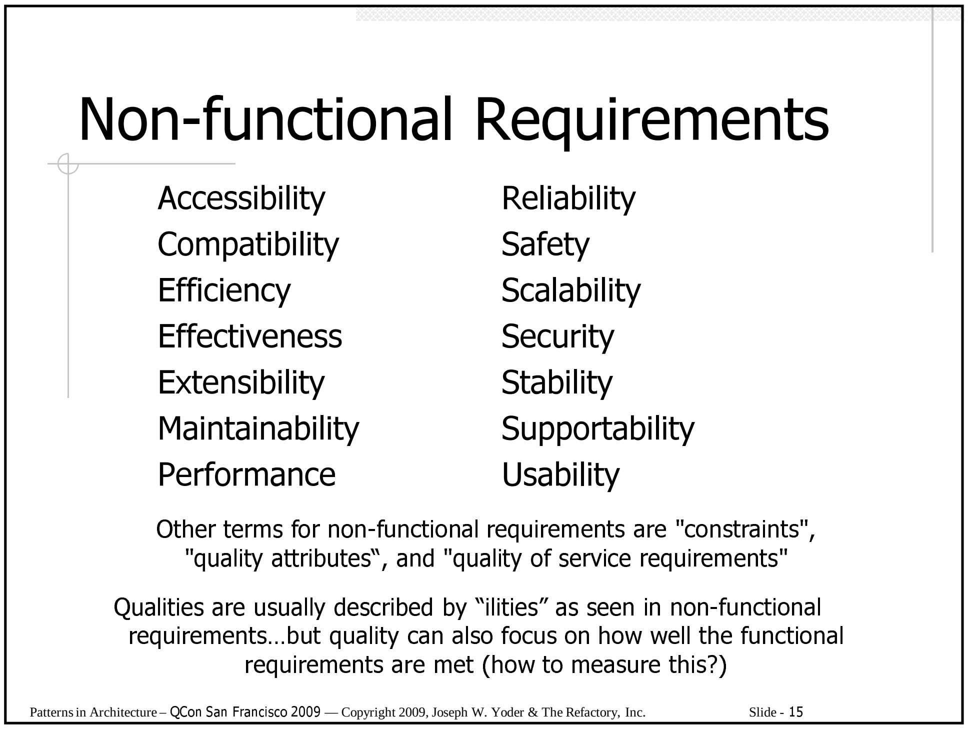
Quality (Who’s perspective)

Artist important/boring	Scientist true/false
Designer cool/uncool	Engineer good/bad

“The Four Winds of Making”…Gabriel

An architect might have perspectives from an artist to a design or an engineer!

Rich Gold "The Plenitude: Creativity, Innovation & Making Stuff (Simplicity: Design, Technology, Business, Life)“
Triggers and Practices – Richard Gabriel http://www.dreamsongs.com

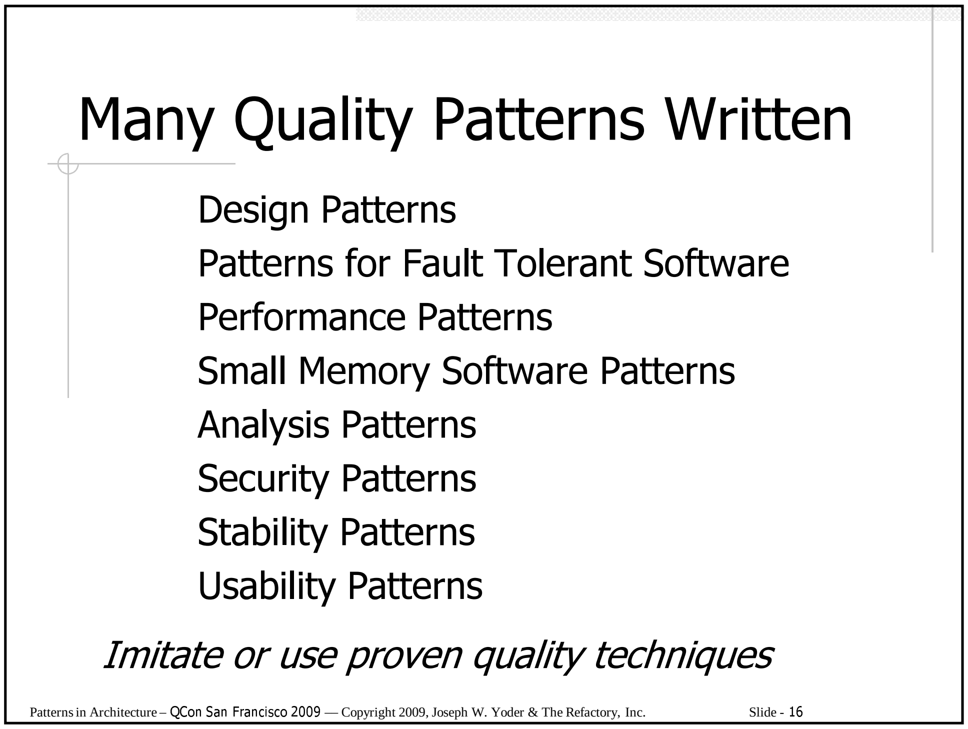


Non-functional Requirements

- Accessibility
- Compatibility
- Efficiency
- Effectiveness
- Extensibility
- Maintainability
- Performance
- Reliability
- Safety
- Scalability
- Security
- Stability
- Supportability
- Usability

Other terms for non-functional requirements are "constraints", "quality attributes‟, and "quality of service requirements"

Qualities are usually described by "ilities" as seen in non-functional requirements…but quality can also focus on how well the functional requirements are met (how to measure this?)



Many Quality Patterns Written

- Design Patterns
- Patterns for Fault Tolerant Software
- Performance Patterns
- Small Memory Software Patterns
- Analysis Patterns
- Security Patterns
- Stability Patterns
- Usability Patterns

Imitate or use proven quality techniques

Brooklyn Bridge

The **Brooklyn Bridge,** 1883, one of the oldest suspension bridges in the United States, stretches over a mile from Brooklyn to Manhattan.

On completion, it was the largest suspension bridge in the world and the first steel-wire suspension bridge.

Designed by John Augustus Roebling. His son, Washington, succeeded him, but was stricken with caisson disease (decompression sickness, commonly known as "the bends").

Brooklyn Bridge

- The occurrence of the bends caused Washington to halt construction of the Manhattan side of the tower 30 feet (10 m) short of bedrock. Today, the Manhattan tower rests only on sand.
- This is a big Change of the Design! But it has been proven that the quality was good enough. Cost of human life was too high.

Brooklyn Bridge

Over engineered. Had 6 times what it needed which proved useful over time

What happens if we tried overdesign of our systems (a language for printing hello world) or the same line of code 6 times (is this 6 times more reliable?)

```
if (x != a) x = a; if (x != a) x = a; if (x != a) x = a;
if (x != a) x = a; if (x != a) x = a; if (x != a) x = a;
```

Redundant components "can" make our systems more reliable

Being Good Enough

Quality of being good enough!!!

Does it meet the minimum requirements?

Quality has many competing forces…are we designing a system for online orders or for controlling the space shuttle, they have different qualities, thus different patterns and solutions apply!

Perfection is the enemy of "Good Enough"

Implementation Patterns

Patterns about creating quality code that communicates well, is easy to understand, and is a pleasure to read. Book is about patterns of "Quality" code.

But…Kent states, "…this book is built on a fragile premise: that good code matters. I've seen too much ugly code make too much money to believe that quality of code is either necessary or sufficient for commercial success or widespread use. However I still believe *quality* of code matters."

Patterns assist with making code more bug free and easier to maintain and extend.

Draining the Swamp

You can escape from the
"*Spaghetti Code Jungle*"

Indeed you can transform the landscape

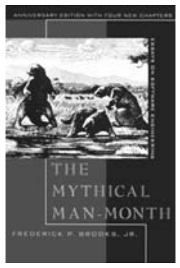
The key is not some magic bullet, but a long-term commitment to **architecture**, and to cultivating and refining "*quality*" **artifacts** for your domain (Refactoring)!

Patterns of the best practices can help!!!

Silver Buckshot

- There are no silver bullets
  …Fred Brooks
- But maybe some silver buckshot
  …promising attacks

Good Design
Frameworks
Patterns
Architecture
Process/Organization
Tools
Good People


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

What is a Pattern?

Patterns can be thought of "Best Practices"
Proven Solutions to Repeating Problems
Embody Experiences of What Works…
…and What Doesn't Work
Captures or Describes Knowledge of Experts

Embody "Quality" Attributes for
Solutions to specific Problems

Alexander on Patterns

To seek the timeless way we must first know the quality without a name.

There is a central quality which is the root criterion of life and spirit in a man, a town, a building, or a wilderness. This *"quality"* is objective and precise, but it cannot be named.

I was no longer willing to start looking at any pattern unless it presented itself to me as having the capacity to connect up with some part of this quality [the quality without a name]. Unless a particular pattern actually was capable of generating the kind of life and spirit that we are now discussing, and that it had this quality itself, my tendency was to dismiss it, even though we explored many, many patterns.

Quality Without a Name (QWAN)

- *From patterns perspective, Alexander looked at QWAN as "the quality"* that imparts incommunicable beauty and immeasurable value to a structure. It encompasses all of the following:

  Universally recognizable aesthetic beauty and order
  Recursively nested centers of symmetry and balance
  Life and wholeness
  Resilience, adaptability, and durability
  Human comfort and satisfaction
  Emotional and cognitive resonance

The Samurai Hasekura

What is a Pattern?

It must be a solution to a problem in a context

You must be able to tell the problem solver what to do, how to solve the problem

It must be a mature, proven solution (Rules of 3)

It must contribute to human comfort or "*quality* " of life

It must be something you didn't invent yourself (Buschmann's Rule)

The solution should build on the insight of the problem solver, and can be implemented a million times without ever being the same twice

It cannot be formalized or automated (if it can, do that instead of writing a pattern)

It should have a dense set of interacting forces that are independent of the forces in other patterns

Architectural Patterns

Architectural Patterns express fundamental structural organization schemas for software systems. The provide a set of predefined subsystems, specify their responsibilities, and include rules and guidelines for organizing the relationships between them.

Patterns of Best Practices!

Pattern – Oriented Software Architecture
Buschmann et al.

Patterns in Architecture

When looking a many systems, common architectural elements will appear

Common architectures have common patterns (solutions to common problems)

Architectural Styles (Garland and Shaw)

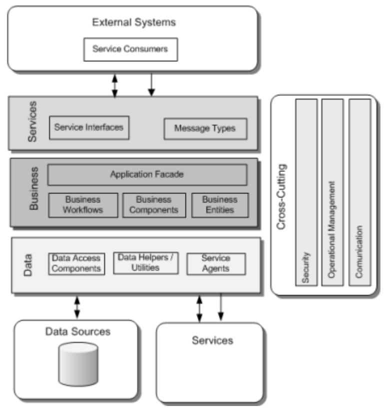
Blackboard, Client-Server, Distributed Computing, Event Driven, Pipes and Filters, Rule Based System, Service Oriented Architecture, Three-Tiered (N-Tiered) systems…

Layered Architecture Pattern

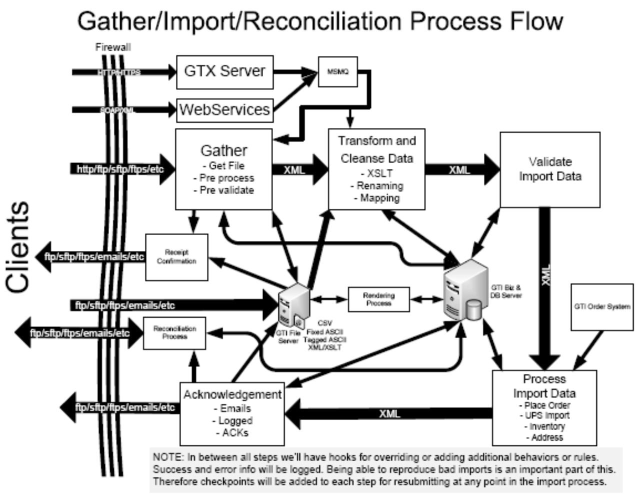
Commonly used to break system down into groups and subtasks

Decompose problem into functional layers

Related to Multi-Tiered!

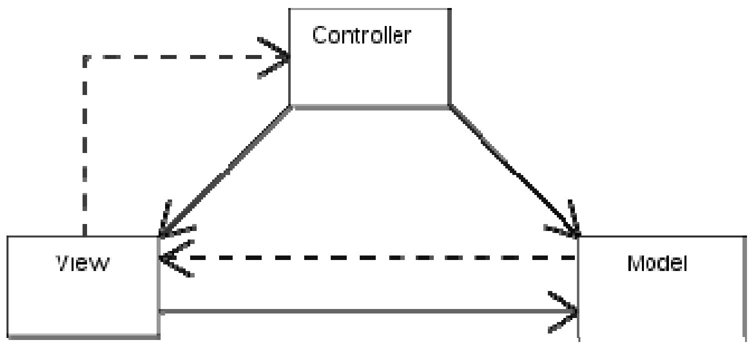


Pipes and Filters Pattern



NOTE: In between all steps we'll have hooks for overriding or adding additional behaviors or rules. Success and error info will be logged. Being able to reproduce bad imports is an important part of this. Therefore checkpoints will be added to each step for resubmitting at any point in the import process.

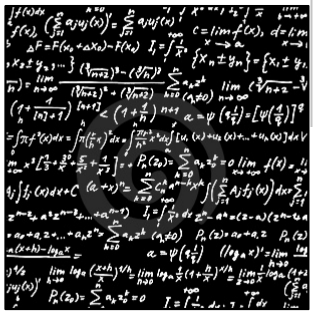
Model-View-Controller Pattern



Blackboard Pattern

"The Blackboard architectural pattern is useful for problems for which no deterministic solution strategies are known. In Blackboard several specialized subsystems assemble their knowledge to build a possibility partial or approximate solution."

Bushmann et al.



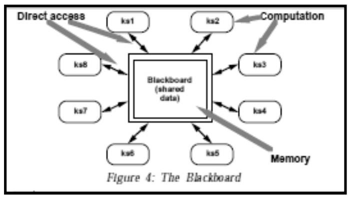


Figure 4: The Blackboard

Service Oriented Architecture (SOA)

What about SOA???

(n.) Abbreviated *SOA*, an application architecture in which all functions, or services, are defined using a description language and have invokable interfaces that are called to perform business processes. Each interaction is independent of each and every other interaction and the interconnect protocols of the communicating devices (i.e., the infrastructure components that determine the communication system do not affect the interfaces).

Service Oriented Architecture (SOA)

There is no widely-agreed upon definition of **service-oriented architecture** other than its literal translation that it is an architecture that relies on service-orientation as its fundamental design principle.

- Uses loosely coupled services.
- Resources on a network are available as independent services without knowledge of their underlying platform implementation.
- Can be applied to business, software and other types of producer/consumer systems.

Service Oriented Architecture (SOA)

Because interfaces are platform-independent, a client from any device using any operating system in any language can use the service. Though built on similar principles, SOA is not the same as Web services, which indicates a collection of technologies, such as SOAP and XML. SOA is more than a set of technologies and runs independent of any specific technologies.

SOA is an architectural style rather than a product.

Almost always see implementation using SOAP/XML.

Service Oriented Architecture

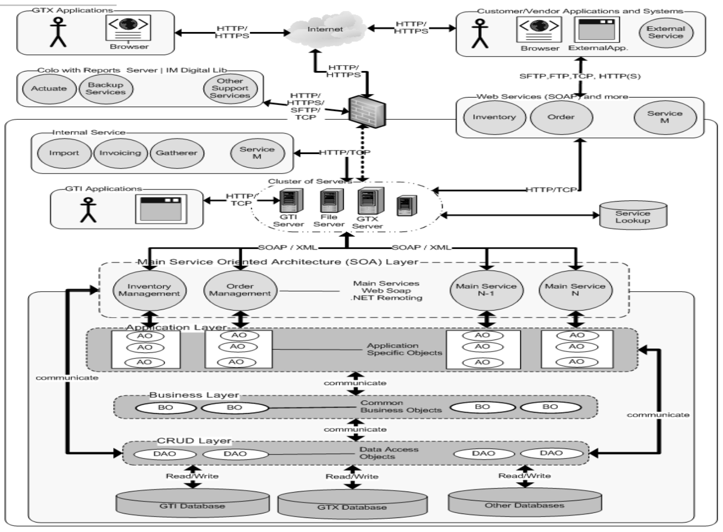
Create a set of loosely coupled services for exposing functionality to enterprise.

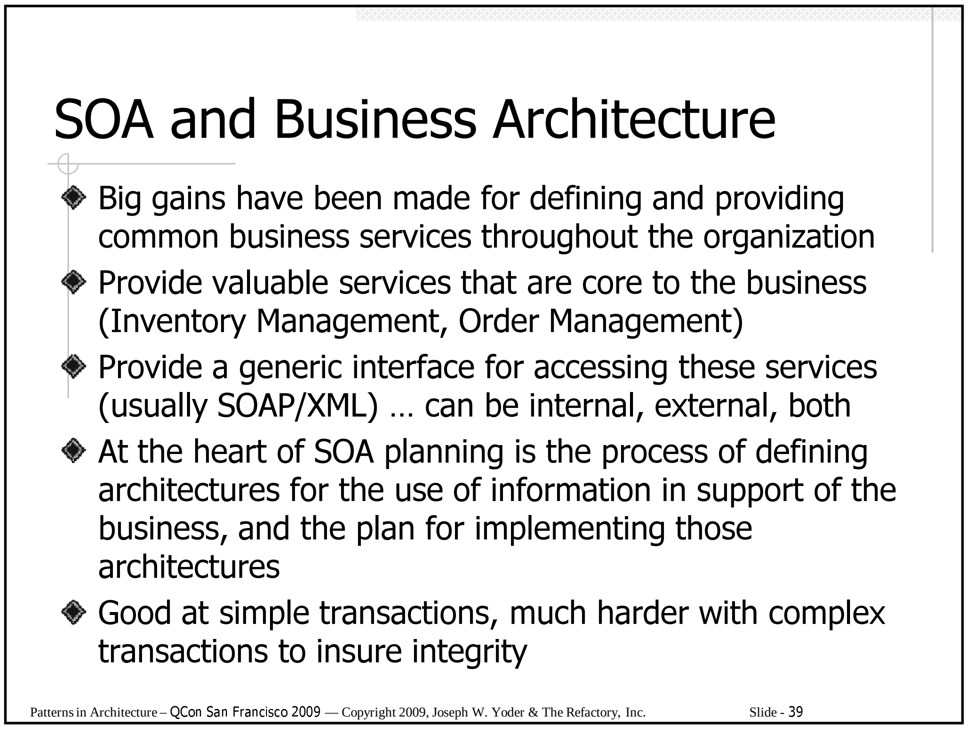
Commonly uses XML and WSDL.

Architectural Style for developing apps.

Quite a bit of hype over the last few years about SOA.

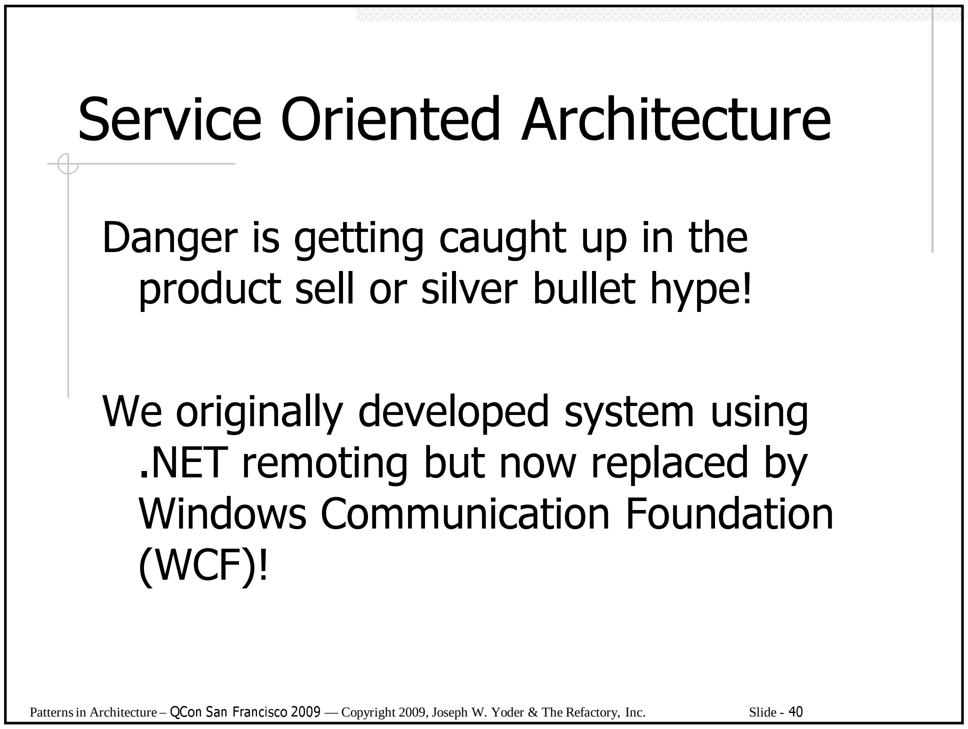
Service Oriented Architecture





SOA and Business Architecture

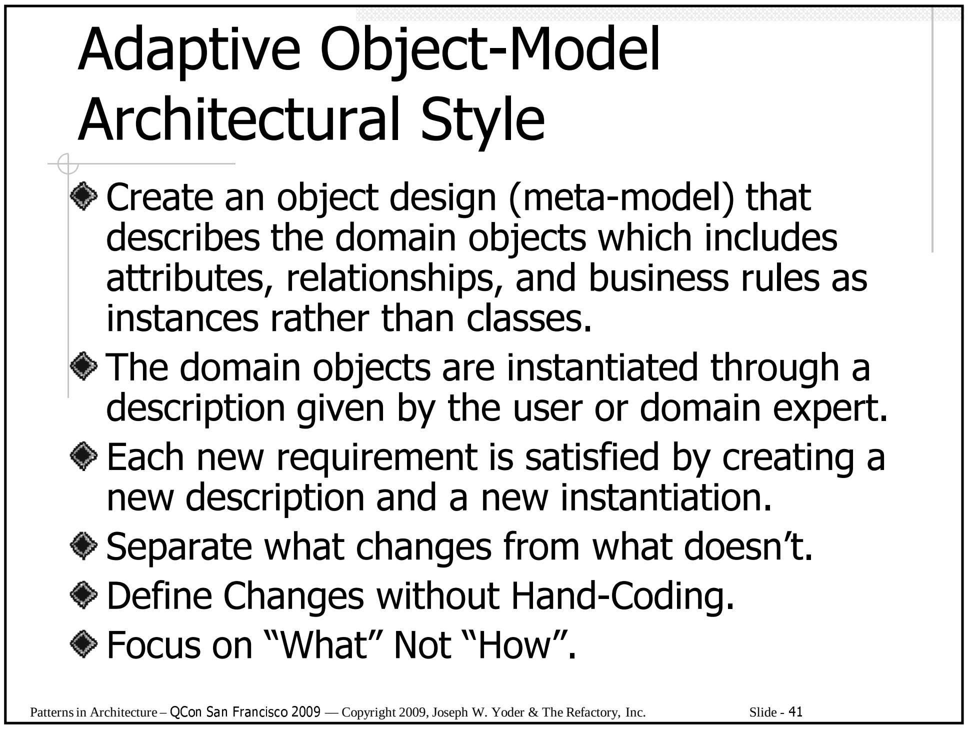
- Big gains have been made for defining and providing common business services throughout the organization
- Provide valuable services that are core to the business (Inventory Management, Order Management)
- Provide a generic interface for accessing these services (usually SOAP/XML) … can be internal, external, both
- At the heart of SOA planning is the process of defining architectures for the use of information in support of the business, and the plan for implementing those architectures
- Good at simple transactions, much harder with complex transactions to insure integrity



Service Oriented Architecture

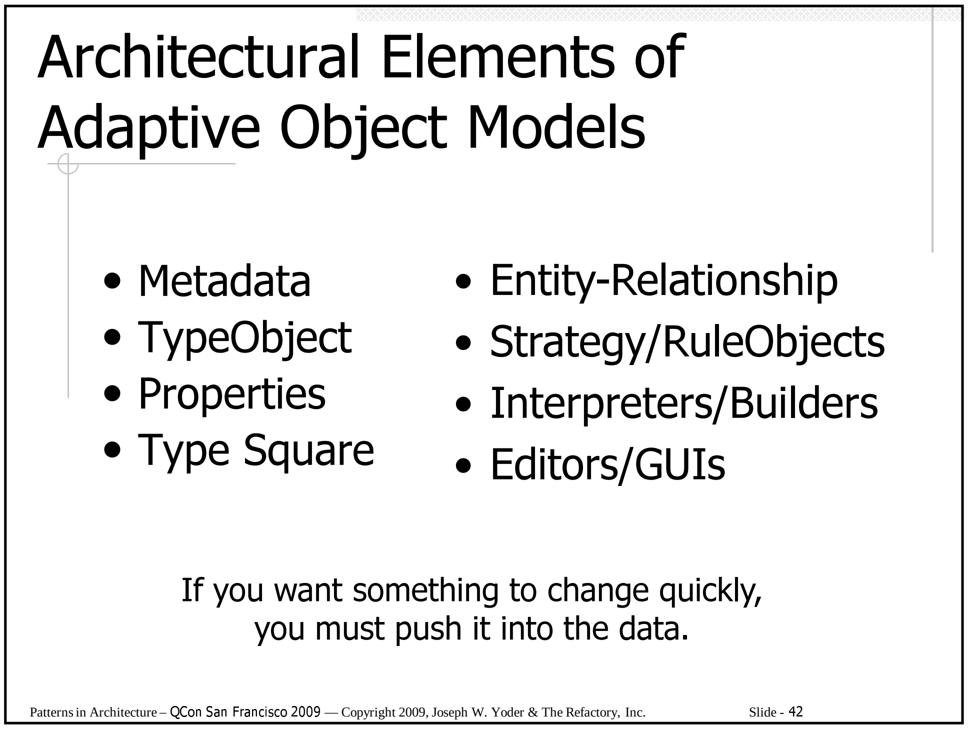
Danger is getting caught up in the product sell or silver bullet hype!

We originally developed system using .NET remoting but now replaced by Windows Communication Foundation (WCF)!



Adaptive Object-Model Architectural Style

- Create an object design (meta-model) that describes the domain objects which includes attributes, relationships, and business rules as instances rather than classes.
- The domain objects are instantiated through a description given by the user or domain expert.
- Each new requirement is satisfied by creating a new description and a new instantiation.
- Separate what changes from what doesn't.
- Define Changes without Hand-Coding.
- Focus on "What" Not "How".

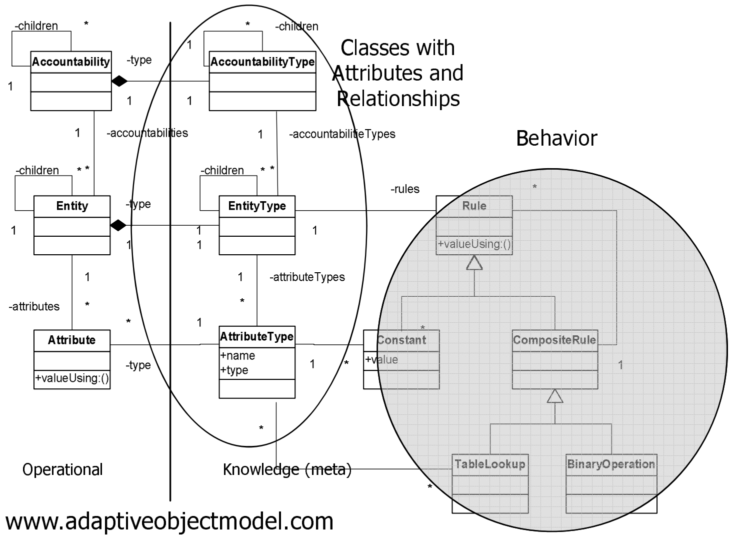


Architectural Elements of Adaptive Object Models

- Metadata
- TypeObject
- Properties
- Type Square
- Entity-Relationship
- Strategy/RuleObjects
- Interpreters/Builders
- Editors/GUIs

If you want something to change quickly,
you must push it into the data.

Adaptive Object Model "Common Architecture"



www.adaptiveobjectmodel.com

Successfully Used For:

(some can be found in papers)

www.adaptiveobjectmodel.com

- Representing Insurance Policies
- Telephone Billing Systems
- Workflow Systems
- Medical Observations
- Banking and Trading
- Validate Equipment Configuration
- Documents Management System
- Gauge Calibration Systems
- Simulation Software

Architect also Implements

Beyond advising and communicating with Developers, Architects should also participate in implementation.

The Architect should be organizationally engaged with Developers and should himself or herself write code. The Architect may implement along with a developer by pair programming!

Jim Coplien Organizational Patterns

Summary

Many proven Architectural styles and patterns are proven techniques that work well in practice.

Perfect is the enemy of the adequate, and Architecture sometimes glorifies perfection, and turns up its nose at the merely adequate or "good enough".

Patterns say go for the good stuff, do what we know works and avoid what we know doesn't work.

This is the Quality of Patterns, focus on good principles rather than trying to count defects and eliminate them, that's beside the point.

Focusing on Architectural Patterns can be a good thing!

Where to Find More Information and Acknowledgements

- ➢http://www.hillside.net
- ➢http://www.refactory.com
- ➢http://www.joeyoder.com
- ➢http://www.dreamsongs.com
- ➢http://www.adaptiveobjectmodel.com

Thanks to Ralph Johnson, Richard Gabriel, Brian Foote, Martin Fowler, and others…

That's All