Creating Software Architectures

Neal Ford
ThoughtWorks
Director / Software Architect / Meme Wrangler
http://www.nealford.com
@neal4d

Mark Richards
Hands-on Software Architect
Published Author / Conference Speaker
http://www.wmrichards.com
https://www.linkedin.com/in/markrichards3
@markrichardssa
workshop agenda - day 1

course introduction

architecture characteristics

architecture patterns

architecture thinking

architecture katas

architecture tradeoffs
Workshop Agenda - Day 2

- Architecture patterns (cont)
- Architecture katas
- Component-based thinking
- Architecture katas
- Presenting architectures
- Documenting architectures
- Architecture decisions
workshop agenda - day 3

analyzing architectures

reactive architecture

evolutionary architecture

next steps

architecture katas
course slides


password: sdd
attendee introductions

your name
role or title
why are you here?
software architecture?

“the highest level concept of a system in its environment. The architecture of a software system (at a given point in time) is its organization or structure of significant components interacting through interfaces, those components being composed of successively smaller components and interfaces.”

Rational Unified Process definition, working off the IEEE definition

software architecture?

Architecture is the highest level concept of the expert developers.

“In most successful software projects, the expert developers working on that project have a shared understanding of the system design. This shared understanding is called ‘architecture.’ This understanding includes how the system is divided into components and how the components interact through interfaces. These components are usually composed of smaller components, but the architecture only includes the components and interfaces that are understood by all the developers.”

software architecture?

Architecture is about the important stuff. Whatever that is.

Ralph Johnson

Architecture is abstract until operationalized.

nealford.com/memeagora/2015/03/30/architecture_is_abstract_until_operationalized.html
expectations of an architect
expectations of an architect

Application Architect
Integration Architect
Information Architect
Data Architect
Systems Architect
Enterprise Architect
Technical Architect
Security Architect
Network Architect
Solutions Architect
Business Architect
expectations of an architect

define the architecture and design principles to guide technology decisions for the enterprise
expectations of an architect

analyze the current technology environment and recommend solutions for improvement.
expectations of an architect

analyze technology and industry trends and keep current with the latest trends
expectations of an architect

ensure compliance with the architecture
expectations of an architect

have exposure to multiple and diverse technologies, platforms, and environments
expectations of an architect

have a certain level of business
domain expertise
expectations of an architect

possess exceptional interpersonal skills, including teamwork, facilitation, and negotiation
expectations of an architect

understand the political climate of the enterprise and be able to navigate the politics
That’s what I do. I drink, and I know things.
architectural thinking
technical breadth vs. depth

- stuff you know
- technical breadth
- stuff you have to maintain
- stuff you know, you don’t know
- technical depth
- stuff you don’t know, you don’t know
where do you draw the line between architecture and design?
identifying architecture characteristics
architecture characteristics

translation skills

reliability
scalability
performance
availability
System Quality Attributes

accessibility  accountability
accuracy  adaptability
administrability
affordability
agility
auditability
autonomy
availability
compatibility
composability
configurability
correctness
credibility
customizability
debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency
evolvability  extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity
interchangeability
interoperability
learnability
maintainability
manageability
mobility
modifiability
modularity
operability
orthogonality
portability
precision
predictability
process capabilities
productibility
provability
recoverability
relevance
reliability
repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
self-sustainability
serviceability
supportability
securability
simplicity
stability
standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

architecture characteristics

“our business is constantly changing to meet new demands of the marketplace”
architecture characteristics

“due to new regulatory requirements, it is imperative that we complete end-of-day processing in time”
architecture characteristics

“we need faster time to market to remain competitive”

???
“our plan is to engage heavily in mergers and acquisitions in the next three years”
architecture characteristics

“we have a very tight timeframe and budget for this project”

???
architecture characteristics

feasibility

agility

elasticity

scalability
"we need lightning-fast response time to keep up with the backlog of calls"

"over time we are expecting the entire company to use this system"

"we are planning to acquire several businesses in the next 5 years"

"the budget and timeframe for this system is very, very tight"
architecture tradeoffs

architecture tradeoff analysis method (ATAM)

proposed architecture

business drivers

quality attributes

validated and approved architecture
architecture tradeoffs

cost-benefit analysis method (CBAM)

- business goals
- performance
- availability
- scalability

assess each and maximize the difference

cost

benefits
architecture tradeoffs

ATAM

CBAM


Software Engineering Institute Digital Library

http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=5177

http://www.sei.cmu.edu/architecture/tools/evaluate/cbam.cfm
architecture katas

familiarization and identifying architecture characteristics
software architecture patterns
architecture patterns help define the basic characteristics and behavior of the application
architecture pattern classification

monolithic

layered

microkernel

pipeline

distributed

event-driven

service-based

service-oriented

microservices

space-based
architecture pattern hybrids

- event-driven layered
- layered microkernel
- event-driven microservices
- space-based microservices
architecture pattern roadmap

- layered architecture
- microkernel architecture
- event-driven architecture
- pipeline architecture
- service-based architecture
- service-oriented architecture
- microservices architecture
- space-based architecture
architecture pattern roadmap

- layered architecture
- microkernel architecture
- event-driven architecture
- pipeline architecture
- service-based architecture
- service-oriented architecture
- microservices architecture
- space-based architecture
layered architecture

presentation layer

business layer

persistence layer

database layer
layered architecture

request

presentation layer

business layer

persistence layer

database layer
layered architecture

presentation layer

business layer

persistence layer

database layer

separation of concerns
layered architecture

presentation layer
business layer
persistence layer
database layer

layers of isolation
layered architecture
hybrids and variants

presentation layer
business layer
services layer
database layer
layered architecture
hybrids and variants
# layered architecture

<table>
<thead>
<tr>
<th></th>
<th>agility</th>
<th>deployment</th>
<th>testability</th>
<th>performance</th>
<th>scalability</th>
<th>simplicity</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image 1</td>
<td>🙅‍♂️</td>
<td>🙅‍♂️</td>
<td>👍</td>
<td>🙅‍♂️</td>
<td>🙅‍♂️</td>
<td>👍</td>
<td>$</td>
</tr>
<tr>
<td>Image 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
architecture pattern roadmap

layered architecture

microkernel architecture

event-driven architecture

pipeline architecture

service-based architecture

service-oriented architecture

microservices architecture

space-based architecture
architecture pattern roadmap

layered architecture

microkernel architecture

event-driven architecture

pipeline architecture

service-based architecture

service-oriented architecture

microservices architecture

space-based architecture
microkernel architecture
(a.k.a. plug-in architecture pattern)
microkernel architecture

architectural components

**core system**
- minimal functionality to run system
- general business rules and logic
- no custom processing

**plug-in module**
- standalone independent module
- specific additional rules or logic
microkernel architecture
microkernel architecture

claims processing
microkernel architecture

registry

registry
1: <location>, <contract>
2: <location>, <contract>
3: <location>, <contract>
4: <location>, <contract>

core system

plug-in component 1
plug-in component 2
plug-in component 3
plug-in component 4
microkernel architecture

plug-in contracts

core system

plug-in component 1

std

plug-in component 2

std

plug-in component 3

std

plug-in component 4

std
### microkernel architecture

<table>
<thead>
<tr>
<th>agility</th>
<th>deployment</th>
<th>testability</th>
<th>performance</th>
<th>scalability</th>
<th>simplicity</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td>$</td>
</tr>
<tr>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
<td>$$</td>
</tr>
</tbody>
</table>
architecture pattern roadmap

layered architecture

microkernel architecture

event-driven architecture

pipeline architecture

service-based architecture

service-oriented architecture

microservices architecture

space-based architecture
architecture pattern roadmap

layered architecture

microkernel architecture

event-driven architecture

pipeline architecture

service-based architecture

service-oriented architecture

microservices architecture

space-based architecture
event-driven architecture

broker topology

mediator topology
event-driven architecture

broker topology
event-driven architecture
event-driven architecture

mediator topology

- event
- event queue
- event channel
- event mediator
- event channel
- event channel
- event processor
  - module
  - module
- event processor
  - module
  - module
- event processor
  - module
  - module
- event processor
  - module
  - module
- event processor
  - module
  - module
- event processor
  - module
  - module
event-driven architecture

you move...

process engine

change address

recalc quote

update claims

adjust claims

notify insured

customer process

quote process

claims process

adjustment process

notification process
### Event-Driven Architecture

<table>
<thead>
<tr>
<th></th>
<th>Agility</th>
<th>Deployment</th>
<th>Testability</th>
<th>Performance</th>
<th>Scalability</th>
<th>Simplicity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td><img src="image7.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td><img src="image8.png" alt="Image" /></td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
<td><img src="image13.png" alt="Image" /></td>
<td><img src="image14.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td><img src="image15.png" alt="Image" /></td>
<td><img src="image16.png" alt="Image" /></td>
<td><img src="image17.png" alt="Image" /></td>
<td><img src="image18.png" alt="Image" /></td>
<td><img src="image19.png" alt="Image" /></td>
<td><img src="image20.png" alt="Image" /></td>
<td><img src="image21.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td><img src="image22.png" alt="Image" /></td>
<td><img src="image23.png" alt="Image" /></td>
<td><img src="image24.png" alt="Image" /></td>
<td><img src="image25.png" alt="Image" /></td>
<td><img src="image26.png" alt="Image" /></td>
<td><img src="image27.png" alt="Image" /></td>
<td><img src="image28.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td><img src="image29.png" alt="Image" /></td>
<td><img src="image30.png" alt="Image" /></td>
<td><img src="image31.png" alt="Image" /></td>
<td><img src="image32.png" alt="Image" /></td>
<td><img src="image33.png" alt="Image" /></td>
<td><img src="image34.png" alt="Image" /></td>
<td><img src="image35.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td><img src="image36.png" alt="Image" /></td>
<td><img src="image37.png" alt="Image" /></td>
<td><img src="image38.png" alt="Image" /></td>
<td><img src="image39.png" alt="Image" /></td>
<td><img src="image40.png" alt="Image" /></td>
<td><img src="image41.png" alt="Image" /></td>
<td><img src="image42.png" alt="Image" /></td>
</tr>
</tbody>
</table>
architecture pattern roadmap

layered architecture → microkernel architecture → event-driven architecture → pipeline architecture

service-based architecture ← service-oriented architecture ← microservices architecture ← space-based architecture
architecture pattern roadmap

layered architecture

microkernel architecture

event-driven architecture

pipeline architecture

service-based architecture

service-oriented architecture

microservices architecture

space-based architecture
pipeline architecture

(a.k.a. pipe and filter architecture)
pipeline architecture

pipes

uni-directional only

usually point-to-point for high performance, but could be message-based for scalability

payload can be any type (text, bytes, object, etc.)
pipeline architecture

filters

self-contained and independent from other filters
usually designed to perform a single specific task
four filter types (producer, consumer, transformer, and tester)
pipeline architecture

filters

producer -> starting point, outbound only

transformer -> input, processing, output

tester -> input, discard or pass-thru

consumer -> ending point, inbound only
pipeline architecture

e.example: capture data in multiple formats, process the data, and send to multiple outputs
<table>
<thead>
<tr>
<th>agility</th>
<th>deployment</th>
<th>testability</th>
<th>performance</th>
<th>scalability</th>
<th>simplicity</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Diagram 1]</td>
<td>![Diagram 1]</td>
<td>![Diagram 1]</td>
<td>![Diagram 1]</td>
<td>![Diagram 1]</td>
<td>![Diagram 1]</td>
<td>![Diagram 1]</td>
</tr>
</tbody>
</table>
architecture pattern roadmap

- layered architecture
- microkernel architecture
- event-driven architecture
- pipeline architecture
- service-based architecture
- service-oriented architecture
- microservices architecture
- space-based architecture
architecture pattern roadmap

- layered architecture
- microkernel architecture
- event-driven architecture
- pipeline architecture
- service-based architecture
- service-oriented architecture
- microservices architecture
- space-based architecture
space-based architecture

let's talk about scalability for a moment...
space-based architecture
space-based architecture

processing unit

processing unit

module ↔ module ↔ module

in memory data grid

data replication engine
space-based architecture

middleware

- messaging grid
- data grid
- processing grid
- deployment manager
space-based architecture
middleware
manages input request and session

messages grid
data grid
processing grid
deployment manager
space-based architecture

middleware

manages data replication between processing units
space-based architecture

text

middleware

manages distributed request processing

- messaging grid
- data grid
- processing grid
- deployment manager
space-based architecture

middleware

manages dynamic processing unit deployment
space-based architecture

product implementations

javaspaces
gigaspaces
ibm object grid
gemfire
ncache
oracle coherence
space-based architecture

it's all about variable scalability...

good for applications that have variable load or inconsistent peak times

not a good fit for traditional large-scale relational database systems

relatively complex and expensive pattern to implement
<table>
<thead>
<tr>
<th>agility</th>
<th>deployment</th>
<th>testability</th>
<th>performance</th>
<th>scalability</th>
<th>simplicity</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
<td><img src="image3.png" alt="Diagram" /></td>
<td><img src="image4.png" alt="Diagram" /></td>
<td><img src="image5.png" alt="Diagram" /></td>
<td><img src="image6.png" alt="Diagram" /></td>
<td>$</td>
</tr>
<tr>
<td><img src="image7.png" alt="Diagram" /></td>
<td><img src="image8.png" alt="Diagram" /></td>
<td><img src="image9.png" alt="Diagram" /></td>
<td><img src="image10.png" alt="Diagram" /></td>
<td><img src="image11.png" alt="Diagram" /></td>
<td><img src="image12.png" alt="Diagram" /></td>
<td>$$</td>
</tr>
<tr>
<td><img src="image13.png" alt="Diagram" /></td>
<td><img src="image14.png" alt="Diagram" /></td>
<td><img src="image15.png" alt="Diagram" /></td>
<td><img src="image16.png" alt="Diagram" /></td>
<td><img src="image17.png" alt="Diagram" /></td>
<td><img src="image18.png" alt="Diagram" /></td>
<td>$$$</td>
</tr>
<tr>
<td><img src="image19.png" alt="Diagram" /></td>
<td><img src="image20.png" alt="Diagram" /></td>
<td><img src="image21.png" alt="Diagram" /></td>
<td><img src="image22.png" alt="Diagram" /></td>
<td><img src="image23.png" alt="Diagram" /></td>
<td><img src="image24.png" alt="Diagram" /></td>
<td>$</td>
</tr>
<tr>
<td><img src="image25.png" alt="Diagram" /></td>
<td><img src="image26.png" alt="Diagram" /></td>
<td><img src="image27.png" alt="Diagram" /></td>
<td><img src="image28.png" alt="Diagram" /></td>
<td><img src="image29.png" alt="Diagram" /></td>
<td><img src="image30.png" alt="Diagram" /></td>
<td>$$$$</td>
</tr>
</tbody>
</table>
architecture pattern roadmap

layered architecture
microkernel architecture
event-driven architecture
pipeline architecture

service-based architecture
service-oriented architecture
microservices architecture
space-based architecture
architecture pattern roadmap

- layered architecture
- microkernel architecture
- event-driven architecture
- pipeline architecture
- service-based architecture
- service-oriented architecture
- microservices architecture
- space-based architecture
microservices architecture
microservices architecture
microservices architecture

protocol-aware heterogeneous interoperability

![Diagram showing service consumers and services communicating through REST](image-url)
microservices architecture

distributed

separately deployed
microservices architecture
### microservices architecture

<table>
<thead>
<tr>
<th>Service Registry and Discovery</th>
<th>Security and Compliance</th>
<th>Load Balancing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deployment</strong></td>
<td><strong>Inter-Service Communication</strong></td>
<td><strong>Network</strong></td>
</tr>
<tr>
<td><strong>Monitoring</strong></td>
<td><strong>Infrastructure Automation</strong></td>
<td><strong>Platform Management</strong></td>
</tr>
<tr>
<td><strong>Continuous Integration</strong></td>
<td><strong>API Management</strong></td>
<td><strong>Service Optimization</strong></td>
</tr>
<tr>
<td><strong>Container Registry</strong></td>
<td><strong>Operating System</strong></td>
<td><strong>Database Management</strong></td>
</tr>
<tr>
<td><strong>Cloud Management</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
microservices architecture

distributed

separately deployed

service component
microservices architecture

client requests

api layer

client requests

functional services

infrastructure services

messaging services
what is the right size for a microservice?

microservices architecture

purpose

transactions

choreography
microservices architecture

purpose

service scope and function
(single-purpose function)
microservices architecture

transactions

no acid transaction
microservices architecture

choreography
microservices architecture

distributed

separately deployed

service component

bounded context
microservices architecture
microservices architecture

client requests

api layer

client requests

client requests

data replication
microservices architecture

distributed

separately deployed

service component

bounded context

data domains
microservices architecture
microservices architecture
microservices architecture

assumes low rate of schema changes (or use of noSQL)
increases performance and overall reliability
reduces data duplication
microservices architecture

distributed

separately deployed

service component

bounded context

data domains

api layer

api layer

client requests

client requests

client requests
api layer

hides the actual endpoint of the service, exposing only those services available for public consumption
api layer
api layer

internal and external client requests

feature toggle
api layer

internal and external client requests
api layer

internal and external client requests

user interface layer

service component module module
service component module module
service component module module
service component module module
service component module module
service component module module
service component module module
service component module module
microservices architecture
api layer
endpoint proxy
api layer

load balancer
api layer

gateway (integration hub)
microservices architecture

distributed

separately deployed

service component

bounded context

data domains

api layer

event driven
microservices architecture

client requests → api layer → service components → module(s) → client requests
microservices architecture
service orchestration

front orchestrator
microservices architecture
service orchestration
microservices architecture
service orchestration
microservices architecture

service orchestration

process claims

module
module

process quotes

module
module

process adjustments

module
module

update customer

module
module

notification

module
module
microservices architecture

service orchestration

process claims

process quotes

process adjustments

update customer

notification
microservices architecture

- reporting techniques
- eventual consistency patterns
- performance tuning
- reactive architecture patterns
## microservices architecture

<table>
<thead>
<tr>
<th>agility</th>
<th>deployment</th>
<th>testability</th>
<th>performance</th>
<th>scalability</th>
<th>simplicity</th>
<th>cost</th>
</tr>
</thead>
</table>

137
pipeline architecture

pipeline vs. event-driven

<table>
<thead>
<tr>
<th>synchronous data filtering</th>
<th>asynchronous event processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>always unidirectional</td>
<td>can be request/reply</td>
</tr>
<tr>
<td>simple single purpose filters</td>
<td>complex multi-purpose processors</td>
</tr>
</tbody>
</table>
architecture pattern roadmap

layered architecture → microkernel architecture → event-driven architecture → pipeline architecture

service-based architecture ← service-oriented architecture ← microservices architecture ← space-based architecture
architecture pattern roadmap

layered architecture

microkernel architecture

event-driven architecture

pipeline architecture

service-based architecture

service-oriented architecture

microservices architecture

space-based architecture
service-oriented architecture

- Business services (BS)
- Message bus
  - Process choreographer
  - Service orchestrator
- Enterprise services (ES)
- Application services (AS)
- Infrastructure services (IS)
service-oriented architecture
service-oriented architecture
service-oriented architecture

- Business services (BS)
- Message bus
- Process choreographer
- Service orchestrator
- Enterprise services (ES)
- Application services (AS)
- Infrastructure services (IS)
service-oriented architecture

business services

message bus

process choreographer

service orchestrator

enterprise services

application services

infrastructure services
service-oriented architecture

enterprise services

business services

message bus

process choreographer

service orchestrator

enterprise services

application services

infrastructure services
service-oriented architecture

application services

business services
message bus
process choreographer
service orchestrator
enterprise services
application services
infrastructure services
service-oriented architecture

infrastructure services

business services

message bus

process choreographer

service orchestrator

enterprise services

application services

infrastructure services
service-oriented architecture

- enterprise scope
- service taxonomy
- shared resources
service-oriented architecture

shared resources

customer

auto and homeowners insurance division

customer

commercial insurance division

customer

casualty insurance division

customer

life insurance division

customer

disability insurance division

customer

travel insurance division

customer
service-oriented architecture

enterprise scope

service taxonomy

shared resources

heterogeneous integration
service-oriented architecture

protocol-agnostic heterogeneous interoperability

service consumer (C#/.NET)  service consumer (Java)  service consumer (Java)

REST  AMQP  REST

messaging  middleware

RMI  ATMI  SOAP

service (EJB3)  service (C++/Tux)  service (Java)
service-oriented architecture

enterprise scope

service taxonomy

shared resources

heterogeneous integration

contract decoupling
service-oriented architecture
service-oriented architecture
contract decoupling
service-oriented architecture

- enterprise scope
- abstraction (api layers)
- contract decoupling
- heterogeneous integration
- shared resources
- service taxonomy
service-oriented architecture

client requests

integration hub

api layer

ES
ES
ES
ES

api layer

api layer
## Service-Oriented Architecture

<table>
<thead>
<tr>
<th></th>
<th>Agility</th>
<th>Deployment</th>
<th>Testability</th>
<th>Performance</th>
<th>Scalability</th>
<th>Simplicity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔵</td>
<td>$</td>
</tr>
<tr>
<td>2</td>
<td>🔵</td>
<td>🔵</td>
<td>🔵</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>$$</td>
</tr>
<tr>
<td>3</td>
<td>🔵</td>
<td>🔵</td>
<td>🔵</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>$$$</td>
</tr>
<tr>
<td>4</td>
<td>🔵</td>
<td>🔴</td>
<td>🔵</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>$</td>
</tr>
<tr>
<td>5</td>
<td>🔵</td>
<td>🔵</td>
<td>🔵</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>$$$$</td>
</tr>
<tr>
<td>6</td>
<td>🔵</td>
<td>🔵</td>
<td>🔵</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>$$</td>
</tr>
<tr>
<td>7</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>$$$$</td>
</tr>
<tr>
<td>8</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>$$$$</td>
</tr>
<tr>
<td>9</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>🔴</td>
<td>$$$$</td>
</tr>
</tbody>
</table>

*Images represent different service-oriented architectures.*
architecture pattern roadmap

layered architecture

microkernel architecture

event-driven architecture

pipeline architecture

service-based architecture

service-oriented architecture

microservices architecture

space-based architecture
architecture pattern roadmap

layered architecture

microkernel architecture

event-driven architecture

pipeline architecture

service-based architecture

service-oriented architecture

microservices architecture

space-based architecture
service-based architecture

is there a middle ground?
service-based architecture

business applications?
service-based architecture
service-based architecture

service granularity

database scope

deployment pipeline
service-based architecture

service granularity
service-based architecture

service granularity

user interface layer

client requests

client requests

client requests
service-based architecture

service granularity

food stamp service
emergency cash service
utility assistance service
child care assist service
health care assist service
nursing facility care service
...

benefit service
service-based architecture

service granularity

advantages

👍 unit of work transactional context
👍 performance and robustness
👍 domain scope
👍 shared resources
service-based architecture

service granularity

tradeoffs

- services development and testing
- deployment pipeline planning
- change control
service-based architecture

database scope

client requests

api layer

client requests

client requests
service-based architecture

database scope

client requests

user interface layer

client requests

client requests

service component

module module module module module
module module module module module
module module module module module
module module module module module
module module module module module

service component

module module module module module
module module module module module
module module module module module
module module module module module
module module module module module

service component

module module module module module
module module module module module
module module module module module
module module module module module
module module module module module
service-based architecture

database scope

client requests

client requests

client requests

user interface layer

service component

module module module module
module module module module
module module module module
module module module module

service component

module module module module
module module module module
module module module module
module module module module

service component

module module module module
module module module module
module module module module
module module module module

database

database
service-based architecture

database scope

food stamp db
emergency cash db
utility assistance db
child care assist db
health care assist db
nursing facility care db
...

shared common db
service-based architecture

database scope

advantages

👍 performance (joins, orchestration, choreography)

👍 feasibility
service-based architecture

database scope

tradeoffs

🚫 bounded context
🚫 service coupling based on schema
🚫 schema changes
service-based architecture

deployment pipeline
service-based architecture
deployment pipeline
service-based architecture

deployment pipeline

advantages

👍 no devops complexity

👍 minimal organizational change
service-based architecture

deployment pipeline

tradeoffs

- lack of quick and effective deployments
- additional risk and coordination needed
- poor continuous delivery model
electronics recycling application

- public requests
- receiving department
- recycling and accounting

kiosk

user interface:
- quote
- receiving
- accounting
- item status
- assessment
- recycling
- reporting

database

180
electronics recycling application

kiosk
public ui
receiving ui
recycling ui

quote service
item status service
receiving service
assessment service
recycling service
accounting service
reporting service
service-based architecture
adding microservices
service-based architecture

adding microservices

client requests

user interface layer

service component

api layer
service-based architecture
adding microservices
## Service-Based Architecture

<table>
<thead>
<tr>
<th>Agility</th>
<th>Deployment</th>
<th>Testability</th>
<th>Performance</th>
<th>Scalability</th>
<th>Simplicity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td><img src="image7.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image8.png" alt="Image" /></td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
<td><img src="image13.png" alt="Image" /></td>
<td><img src="image14.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image15.png" alt="Image" /></td>
<td><img src="image16.png" alt="Image" /></td>
<td><img src="image17.png" alt="Image" /></td>
<td><img src="image18.png" alt="Image" /></td>
<td><img src="image19.png" alt="Image" /></td>
<td><img src="image20.png" alt="Image" /></td>
<td><img src="image21.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image22.png" alt="Image" /></td>
<td><img src="image23.png" alt="Image" /></td>
<td><img src="image24.png" alt="Image" /></td>
<td><img src="image25.png" alt="Image" /></td>
<td><img src="image26.png" alt="Image" /></td>
<td><img src="image27.png" alt="Image" /></td>
<td><img src="image28.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image29.png" alt="Image" /></td>
<td><img src="image30.png" alt="Image" /></td>
<td><img src="image31.png" alt="Image" /></td>
<td><img src="image32.png" alt="Image" /></td>
<td><img src="image33.png" alt="Image" /></td>
<td><img src="image34.png" alt="Image" /></td>
<td><img src="image35.png" alt="Image" /></td>
</tr>
<tr>
<td><img src="image36.png" alt="Image" /></td>
<td><img src="image37.png" alt="Image" /></td>
<td><img src="image38.png" alt="Image" /></td>
<td><img src="image39.png" alt="Image" /></td>
<td><img src="image40.png" alt="Image" /></td>
<td><img src="image41.png" alt="Image" /></td>
<td><img src="image42.png" alt="Image" /></td>
</tr>
</tbody>
</table>
traditional
CQRS
Command Query Responsibility Separation
reporting in cqrs
CQRS natural fits

task-based user interface

meshes well with event sourcing

eventual consistency
eventual consistency

“Building reliable distributed systems at a worldwide scale demands trade-offs between consistency and availability.”

http://www.allthingsdistributed.com/2008/12/eventually_consistent.html
CQRS natural fits

task-based user interface
meshes well with event sourcing
eventual consistency consistency or availability (but never both)
complex or granular domains
JVM-based retail financial trading platform centers on Business Logic Processor handling 6,000,000 orders/sec on 1 thread surrounded by Disruptors, network of lock-less queues.
overall structure

- single-threaded Java app
- relies only on JVM
- easy to test
business logic processor

in-memory

event sourcing via input disruptor

snapshots (full restart—JVM + snapshots — less than 1 min)

multiple instances running

each event processed by multiple processors but only one result used
input/output disruptors
disruptors

custom concurrency component
multi-cast graph of queues where producers enqueue objects and consumers dequeue in parallel
ring buffer with sequence counters
20x10^6 slots for input buffer
4x10^6 slots for output buffer
“mechanical sympathy”

started with transactions

switched to Actor-based concurrency

*hypothesized & measured results*

CPU caching is key ➔ single writer principle
architecture katas

identifying architecture patterns
component-based thinking
component identification and granularity
component identification

as an architect, you should think about the artifacts within the architecture in terms of components

component:
an encapsulated unit of software that has a well defined interface and a clear and concise role and responsibility statement
component identification

component scope

subsystem, layer, or service

component

module

module

component

module

module

component

module

module
component identification

roles and responsibility model
component identification

stock trade order validation
component identification

stock trade order validation

message dispatcher - compliance controller

responsible for dispatching the trade to the next available controller.
component identification

stock trade order validation

responsible for orchestrating the trade order validation process by calling specific compliance processors.
component identification

stock trade order validation

responsible for making sure the trader isn't exceeding assigned trader limits for the trade being placed.
component identification

stock trade order validation

responsible for making sure the trade order symbol isn't on the restricted stock list.
Who should be responsible for retrieving and caching all of the data needed by the compliance processors?
component identification

stock trade order validation

responsible for orchestrating the trade order validation process by calling specific compliance processors. also responsible for retrieving and caching all data needed by the compliance processors
component identification

stock trade order validation

who should be responsible for persisting trade validation errors when they occur?
component identification

stock trade order validation

responsible for orchestrating the trade order validation process by calling specific compliance processors. also responsible for retrieving and caching all data needed by the compliance processors and persisting all validation errors.
component identification

stock trade order validation
component identification

stock trade order validation
component identification

stock trade order validation

responsible for retrieving and caching all data needed by the compliance processors and persisting all validation errors.
Message Structure

message dispatcher

compliance controller

stock trade order validation

data manager

trader limits

restriction

responsible for orchestrating the trade order validation process by calling specific compliance processors.
component identification

identify initial core components

assign user stories to components

refactor components

analyze roles and responsibility statements

analyze non-functional aspects
component identification

identify initial components using core functionality

place orders → order placement

ship orders → order shipment

track orders → order tracking
component identification

assign requirements, use cases, or user stories to a component

user story: check inventory → order placement
user story: validate order → order placement
user story: notify customer → customer notification
service identification

identify coarse-grained functional areas

create coarse-grained microservices

map services to data tables
service identification

1. Identify coarse-grained functional areas
2. Identify data overlaps and dependencies
3. Refine services and data
component identification

component granularity

order manager

responsible for creating, deleting, and updating orders. also responsible for shipping the order and tracking the order once it has been shipped. this component is also responsible for notifying the customer each time the order status changes.
component identification

component granularity

order manager

- order maintenance
  - responsible for creating, deleting, and updating orders.

- order shipment
  - responsible for shipping and tracking orders

- customer notification
  - responsible for notifying the customer when the order status changes.
component coupling
component coupling

the extent to which components know about each other
**component coupling**

**afferent coupling**

the degree to which other components are dependent on the target component

![Diagram showing component coupling]

- component b
- component c
- component d
- component a
component coupling

efferent coupling

the degree to which the target component is dependent on other components
component coupling

temporal coupling

functionality is grouped into one component due to timing dependencies (e.g. transactions)
component coupling

tight coupling

pathological coupling

external coupling

control coupling

data coupling

loose coupling
component coupling
pathological coupling
one component relies on the inner workings of another component
component coupling

external coupling

multiple components share an externally imposed protocol or data format
component coupling

control coupling

one component passes information to another component on what to do

component a

component b
component coupling
data coupling
the degree to which components are bound to a shared data context
component coupling
consequences of ignoring...
component cohesion
component cohesion

the degree and manner to which the operations of a component are related to one another
component cohesion

the degree and manner to which the operations of a component are related to one another

customer maintenance

- add customer
- update customer
- get customer
- notify customer

order maintenance

- get customer orders
- cancel customer orders
architecture katas

identifying major architecture components
Documenting & Presenting Software Architecture
documenting software
<table>
<thead>
<tr>
<th>Title</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dinner Plans</td>
<td>Athena</td>
</tr>
<tr>
<td>Muffin</td>
<td></td>
</tr>
<tr>
<td>Popover</td>
<td></td>
</tr>
<tr>
<td>Quiche</td>
<td></td>
</tr>
<tr>
<td>(Kale Bake)</td>
<td></td>
</tr>
</tbody>
</table>
DO NOT ERASE
documentation
weight in bytes
motivation
agile architecture

Not all decisions made up front.

project timeline
Architectural Diagramming Techniques
https://en.wikipedia.org/wiki/4+1_architectural_view_model
http://www.codingthearchitecture.com
The C4 model

System Context
The system plus users and system dependencies

Containers
The overall shape of the architecture and technology choices

Components
Components and their interactions within a container

Classes (or Code)
Component implementation details
The C4 model

System Context
The system plus users and system dependencies

Containers
The overall shape of the architecture and technology choices

Components
Components and their interactions within a container

Classes (or Code)
Component implementation details
context

container

component

\[\text{Class}\]
\[\text{Attribute}\]

\[\text{Class Name}\]
\[\text{Attribute}\]
The C4 model

System Context
The system plus users and system dependencies

Containers
The overall shape of the architecture and technology choices

Components
Components and their interactions within a container

Classes (or Code)
Components' implementation details
Person = User | A user of my software system. | 277,674
SoftwareSystem = Software System | My software system. | 1552,674

Relationship = User | Uses | Software System |

Diagram = System Context | Software System | A description of this diagram. | A5_Landscape

ElementStyle = Element | 650 | 400 | #ffffff | 36
ElementStyle = Software System | #1168bd |
ElementStyle = Person | #08427b |

Structurizr
Workspace workspace = new Workspace("My model", "This is a model of my software system.");
Model model = workspace.getModel();

Person user = model.addPerson("User", "A user of my software system.");
SoftwareSystem softwareSystem = model.addSoftwareSystem("Software System", "My software system.");
user.uses(softwareSystem, "Uses");

ViewSet viewSet = workspace.getViews();
SystemContextView contextView = viewSet.createSystemContextView(softwareSystem, "context", "A simple example...");
contextView.addAllSoftwareSystems();
contextView.addAllPeople();

Styles styles = viewSet.getConfiguration().getStyles();
styles.addElementStyle(Tags.SOFTWARE_SYSTEM).background("#1168bd").color("#ffffff");
styles.addElementStyle(Tags.PERSON).background("#08427b").color("#ffffff");

StructurizrClient structurizrClient = new StructurizrClient("key", "secret");
structurizrClient.putWorkspace(1234, workspace);
```csharp
static void Main(string[] args) {
    Workspace workspace = new Workspace("Financial Risk System", "A simple example C4 model based upon the financial risk system architecture");
    Model model = workspace.Model;

    // create the basic model
    Person businessUser = model.AddPerson(location.Internal, "Business User", "A regular business user");
    businessUser.Uses(financeRiskSystem, "Views reports using");

    Person configurationUser = model.AddPerson(location.Internal, "Configuration User", "A regular business user who can also configure the system");
    configurationUser.Uses(financeRiskSystem, "Configures parameters using");

    SoftwareSystem tradeDataSystem = model.AddSoftwareSystem(location.Internal, "Trade Data System", "The system of record for trade data");
    tradeDataSystem.Uses(financeRiskSystem, "Gets trade data from");

    referenceDataSystem.Uses(financeRiskSystem, "Gets counterparty data from");

    SoftwareSystem emailSystem = model.AddSoftwareSystem(location.Internal, "E-mail system", "Microsoft Exchange");
    emailSystem.Uses(financeRiskSystem, "Sends a notification that a report is ready to");
    emailSystem.Delivers(businessUser, "Sends a notification that a report is ready to", "E-mail message", InteractionStyle.Asyncronous);

    centralMonitoringService.Uses(financeRiskSystem, "Sends critical failure reports to", "SNMP", InteractionStyle.Asyncronous);

}
```
Navigate from diagram to source code
Diagrams are maps
Titles

Short and meaningful, numbered if diagram order is
Lines: add descriptive text to provide additional information.

Favor unidirectional arrows.
Sticky notes and index cards make a great substitute for drawn
Color

Ensure that color coding is made explicit; watch out for color-
Color

Ensure that color coding is made explicit; watch out for color-
Orientation

Most important thing in the middle;
Shapes

Don’t assume that people will understand what different shapes
Keys

Explain shapes, lines, colors, borders, acronyms, etc
Representational Consistency

Don’t abruptly change scale on diagrams; provide context for
Representational Consistency

Don’t abruptly change scale on diagrams; provide context for
Comprehensive Diagram

Don’t try to capture the entirety of software architecture in a single
Comprehensive Diagram

Don’t try to capture the entirety of software architecture in a single
Decisions
Architecture Decision

We will keep a collection of records for "architecturally significant" decisions: those that affect the structure, non-functional characteristics, dependencies, interfaces, or construction techniques.

http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions
Architecture Decision

Textile

asciidoc
short text
file

reversed ADR
kept—
marked
superseded

doc/arch/adr-NNN.md

numbered

semantic

no number
Architecture Decision

**Title**: short noun phrase

**Context**: forces at play

**Decision**: response to forces

**Status**: proposed, accepted, superseded
https://github.com/npryce/adr-tools
Justifying Decisions
the scenario

internal client → integration hub → ActiveMQ → internal application

JMS destination
Justifying Decisions

the requirement: you need to federate the hub

extern
interna
b2b

integration hub

JMS destination

internal application

ActiveMQ
Justifying Decisions

the decision: dedicated broker instances?

integration hub  →  ActiveMQ

extern

internal

b2b

internal application

integration hub

JMS destination
Justifying Decisions
the decision: centralized broker
# Reducing the Hub

## Content

The architecture application currently utilizes an integration hub to allow internal applications to connect to it. Figure 1 illustrates the existing scenario:

![Figure 1: Before new broker added]

New requirements require developers to add two new access types: `external` and `app`.

### Considerations

- Broker only used for hub access
- Low transaction volume expected
- Application logic may be shared between different types of client applications (e.g., internal and external)

Two options exist:

### Dedicated Broker Instances

Using dedicated broker instance creates the architecture shown in Figure 1.

![Figure 2: Dedicated broker instance]

**Identified issues to address:*

- **Throughput**: Dedicated broker instances provide better throughput because no message contention exists.
- **Stateful Application Mapping**: This approach requires changes to the internal client application.

**Advantages**:

- No change to client
- Additional brokers added require minimal changes to client
- Single point of failure is removed
- Multiple broker instances should provide against aggregated performance problems

### Centralized Broker

Using dedicated broker instances creates the architecture shown in Figure 1.

![Figure 3: Centralized broker]

**Identified issues to address:*

- **Throughput**: Centralized broker potentially creates a throughput bottleneck. However, developers analyzed past and expected future usage, and this shouldn’t create problem.
- **Deterministic Application Mapping**: Some, with only one connection, app doesn’t know about broker
- **Integrity**: The internal application doesn’t need to know where the request originated.
- **Changes to client**: Additional brokers do not require changes to client.
- **Single point of failure**: Mitigated by clustering and failover

### Conclusion

We chose a **Dedicated Broker**.

### Status

- **Proposed**

**Implementation**

The internal application should not have to know from which broker instance the request came from.

Only a single broker connection is needed, allowing for the expansion of additional hub instance with no application changes.

Due to low usage volume the performance bottleneck is not an issue.

A single point of failure can be addressed through failover mode of clustering.
Federating the Hub

## Context
The AcmeWidgets application currently utilizes an integration hub to allow internal applications to connect to it.

Figure 1 illustrates the existing scenario:

![Before](adr-001-before.png)

*Figure 1: Before new clients added*

New requirements require developers to add two new access types: _external_ and _b2b_.

### Considerations:
* broker only used for hub access
* low transaction volumes expected
* application logic may be shared between different types of client applications (e.g., internal and external)

Two options exist:

### Dedicated Broker Instances

Using dedicated broker instances creates the architecture shown in Figure 2.

![dedicated](adr-001-dedicated.png)

*Figure 2: dedicated broker instances*

Identified issues to address:

* **throughput**: Dedicated broker instances provide better throughput because no message contention exists.
* **internal application coupling**: This approach requires changes to the internal client application to "understand" the broker. Internal app must now connect to three brokers and know context of request.
* **changes to client**: additional brokers added requires additional changes to client.
* **single point of failure**: redundancy prevents a single failure from disabling all integration architecture.
* **performance**: multiple broker instances should protect against aggregated performance problems.

### Centralized Broker

Using dedicated broker instances creates the architecture shown in Figure 3.
Federating the Hub

Context

The AcmeCorps application currently utilizes an integration hub to allow internal applications to connect to it. Figure 1 illustrates the current connection:

![Diagram of current connection]

Figure 1: Current connection

New requirements require developers to add two new access types: external and web.

Considerations:
- Broker only used for high volume
- One transaction instance per web
- Application logic may be shared between different types of clients (e.g., internal and external)

Two options exist:

Dedicated Broker Instances

Using dedicated broker instances creates the architecture shown in Figure 2:

![Diagram of dedicated broker instances]

Figure 2: Dedicated broker instances

Identified issues to address:
- Throughput: Dedicated broker instances provide better throughput because no message conversion

Centralized Broker

Using a centralized broker results in the architecture shown in Figure 3:

![Diagram of centralized broker]

Figure 3: Centralized broker

Identified issues to address:
- Throughput: Centralized broker potentially causes throughput issues. However, concurrency analysis and resource management can address these concerns.
- Internal application coupling: Users with only one connection, app fails. Without access to broker, users have to know where to connect externally.
- Changes in external access points can impact volume in the broker.
- Single point of failure: Degrades with downtime.

Solution: Performance because we expect low transaction volume, performance would be sufficient with a single queue.

Decision

We chose a centralized broker.

Status

In progress

Consequences

The internal applications should not have a need for any broker instance because of the centralized approach.

Only one broker instance is modeled, allowing for the expansion of additional hub instances with no application change.
The Case for

http://asciidoc.org

http://asciidoctor.org
The Case for

http://asciidoc.org/docs/asciidoc-diagram/
The Case for
Federating the Hub

Context
The AcmeWidgets application currently utilizes an integration hub to allow internal applications to connect to it. Figure 1 illustrates the existing scenario:

New requirements require developers to add two new access types: internal and b2b.

Considerations:
- broker only used for hub access
- low transaction volume expected
- application logic may be shared between different types of client applications (e.g., internal and external)

Two options exist:

Dedicated Broker Instances
Using dedicated broker instances creates the architecture shown in Figure 2.

Centralized Broker
Using dedicated broker instances creates the architecture shown in Figure 3.

Identified Issues to address:
- **throughput**: Dedicated broker instances provide better throughput because no message correlation exists.
- **internal application coupling**: This approach requires changes to the internal client application to "understand" the broker. Internal app must now connect to three brokers and know context of request.
- **changes to client**: Additional brokers added requires additional changes to client.
- **single point of failure**: Redundancy prevents a single failure from disabling all integration architecture.
- **performance**: Multiple broker instances should protect against aggregated performance problems.
Considerations:
- High-level, well-designed and secure.
- Exploit existing services.
- Use transactional interface transparent to the client application.
- The transaction engine may be utilized to ensure a consistent image of system state and transaction execution.

The returned values:

**Dedicated Broker Instances**

Using dedicated broker instances creates the architecture shown in Figure 1.

...
Archeology

docs

root for all documents

current

useful enough to update

archeology

interesting historical artifacts

317
Rules for Documentation

1. useful now

2. as little as possible

3. always accurate
Documenting & Presenting Software Architecture

@neal4d
nealford.co

ThoughtWorks

NEAL FORD
Director / Software Architect / Genre Wrangler

@neal4d
nealford.co
Presenting Software Architecture
Presentation PATTERNS

Techniques for Crafting Better Presentations

NEAL FORD | MATTHEW MCCULLOUGH | NATHANIEL SCHUTTA
building blocks
building blocks

transition  movement between slides

animation  movement on slides
building blocks

transition  movement between slides

animation  movement on slides
animations

you move...

process engine

change address
recalc quote
update claims
adjust claims
notify insured

customer
quote process
claims process
adjustment
notification
building blocks
building blocks

transition  movement between slides

animation  movement on slides

Magic Move
building blocks
Magic Move for tools that don't have Magic Move
Magic Move Version
Red Shirt

- Fact 1
- Another Fact
- Fact 3
- Yet Another Fact
- Fact 5
Magic Move Version
Non-Magic Move Version
Non-Magic Move Version

- Fact 1
- Another Fact
- Fact 3
- Yet Another Fact
- Fact 5
Non-Magic Move Version
Non-Magic Move Version
Non-Magic Move Version
Non-Magic Move Version

Lipsync
creativity
Presentation Patterns
Building Blocks for Perfect Presentations

Creativity Patterns:

Narrative Arc
Presentations are a form of ...
“Get your protagonist up a tree. Throw rocks at him. Then get him down.”

—Syd Fields
Narrative Arc

Tension

Introduction & exposition

Complication

Story progression

Climax

Resolution
Why So Many Superheroes?
Technical Narrative?
Presentation Patterns
Building Blocks for Perfect Presentations

Creativity Patterns:
Concurrent Creation
Don’t feel compelled to create your presentation materials in the same order as the presentation itself.
Concurrent Creation

Don’t feel compelled to create your presentation materials in the same order as the presentation itself.

When creating a presentation as a group, follow certain practices to retain sanity.
When creating a presentation as a group, follow certain practices to retain sanity.
Sanity-saving Practices

slide wrangler
Sanity-saving Practices

unify on the same theme/template
Sanity-saving Practices

no one can change it without ensuring everyone has the update
Sanity-saving Practices

slide wrangler
Set a Deadline (with Teeth)

slide wrangler
Known Uses

every corporation everywhere!
Expansion Joints

Also know as:
Goldilocks; Short, Medium, Long
Implicit versus Explicit
Implicit Expansion Joints practice skipping gracefully
Implicit versus Explicit
Explicit Expansion Joints
makes sense within the Narrative Arc

same (or closely related) Unifying Visual Theme
multi-purpose executives
management engineering
representational
representational
representational
creation
some ideas > 1 slide

auto-size text is evil!

don't allow the tool to alter the message

evil!
git magic
git magic

1. undo disastrous checkout
2. save changes to local stash
3. create local branch
4. push stash to local branch
git magic

git server
5. push local branch to remote branch

6. you broke it —
git magic

7. stash recent changes
8. checkout remote branch
9. fix it!
10. check into main
11. unstash & get back to work
Communities: Significant Initiatives Needed to Achieve 2008 Vision

- Recruit technical writers
  - Turn over capabilities content writing and sales material writing to technical writers
  - Full time positions
- Global leads and smaller group of experts full time
  - In support of communities, initiatives, external branding
  - In support of mentoring at specific projects by lead/inst
- Support Global Leads to become external facing
  - Development, opportunities, etc
  - Find billable advisory work (high value consultancy services)
- Special non billable time given to specific community initiatives on TBD basis
- Introduce ‘Mentor’ role for projects
  - PMs, CRs, Testers, Analysts, Architects, etc to visit at project’s request for advice / mentoring / direction
  - In support of Delivery Assurance, introduction of consistent practices, knowledge increase & role growth
  - Should have immediate impact on delivery risk
- Move to model of ‘critical path’ staffing versus long term staffing
  - Pool of experts in each category to staff long term on projects
  - They’re staffed for ‘critical periods’ and then move to different projects
  - Frees them to access mentors on multiple projects during critical periods
  - Moves us away from ‘usual suspect’ staffing (only X and Z can do this type of thing)
  - People that can do it aren’t being challenged – we just bring on the experts full time – not working
Why So Many Bullets?

• Both presenters and audience expect it.

• Title + Bullets is often the default template.

• Inexperienced speaker’s rely on bullets as speaker notes

• Easy to bang together in a conference room while > 4 people are talking
When a Slide Full of Text Appears

• Everyone in the audience

• Reads the entire thing right away

• You can’t help it

• Now, the presenter spends the next five minutes

• slowly reading what you’ve already read
When showing a slide, pause for a moment to allow the audience to read it before you continue.

– Multiple terrible presentation guides
Cookie Cutter

most ideas > 1

auto-size text evil!

2 slide

backtracking
a Context Keeper technique that enables you to reestablish a narrative context by purposefully repeating slides.

Backtracking

Introduction & exposition

Complication

Climax

Resolution

Tension

Story progression

digressio

digressio

digressio
Comedians

callback
Presentation Patterns
Building Blocks for Perfect Presentations

Slide Creation Patterns:

Floodmarks

Also know as:
Death by Advertising, Marketing Mania, Kudzu Log
Floodmarks

Floodmarks represent extraneous background imagery featured on every slide.
Building Web Services with Java

Neal Ford

CTO, The DSW Group, Ltd.

Tuesday, September 10, 2002
Floodmarks as Constraint
Communities:
Significant Initiatives Needed 
to Achieve 2008 Vision

- Recruit technical writers
  - Turn over capabilities content writing and sales material writing to technical writers
  - Full time positions
- Global leads and smaller group of experts full time
  - In support of communities, initiatives, external branding
  - In support of mentoring at specific projects by request
- Support Global Leads to become external facing
  - Development, opportunities, etc
  - Find billable advisory work (high value consultancy services)
- Special non billable time given to specific community initiatives on TBD basis
- Introduce 'Mentor' role for projects
  - PMs, DBs, Testers, Analysts, Architects, etc to visit at project's request for advice / mentoring / direction
  - In support of Delivery Assurance, introduction of consistent practices, knowledge increase & role growth
  - Should have immediate positive impact on delivery risk
- Move to model of 'critical path' staffing versus long term staffing
  - Pool of experts in each region aren't staffed long term on projects
  - They're staffed for 'critical periods' and then move to different projects
  - Frees them to act as mentors on multiple projects during critical periods
  - Moves us away from 'usual suspect' staffing (only X and Z can do this type of thing)
  - People that can do it aren't being challenged – we just bring on the experts full time – not working
- Floodmark artificially compresses the headline.
- Lines at top and bottom artificially constrain information space.
  - What happens to large images?
  - Makes slide transitions obvious
- Copyright and company logo on each slide negates intended effect.
Use Interfaces for Decoupling

Floodmarks + Images
Should You Ever Floodmark?

Bookend slides

Anytime you want to *remind* your audience of branding

At the end
temporal
Presentation

Presenter controls exposition rate.

Slideument

Infodeck

Reader controls exposition rate.
Testing Strategies in a Microservice Architecture

There has been a shift in service based architectures over the last few years towards smaller, more focussed "micro" services. There are many benefits with this approach such as the ability to independently deploy, scale and maintain each component and parallelize development across multiple teams. However, once these additional network partitions have been introduced, the testing strategies that applied for monolithic in process applications need to be reconsidered.

Here, we plan to discuss a number of approaches for managing the additional testing complexity of multiple independently deployable components as well as how to have tests and the application remain correct despite having multiple teams each acting as guardians for different services.

18 November 2014

Toby Clemson is a developer at ThoughtWorks with a passion for building large scale distributed business systems. He has worked on projects in four continents and is currently based in New York.

My thanks to Martin Fowler for his continued support in compiling this infodeck. Thanks also to Danilo Sato, Dan Coffman, Steven Lowe, Chris Ford, Mark Taylor, Praful Todkar, Sam Newman and Marcos Matos for their feedback and contributions.

http://martinfowler.com/articles/microservice-testing
Best Compromise Slideument

Problems:

- mechanically difficult to write prose in speaker’s notes
- people tends towards Bullet Riddled Corpse summaries

Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and
demos vs presentations
traveling highlights
traveling highlights
traveling highlights
traveling highlights

client requests

api layer

client requests

client requests
dimensions
dimensions
dimensions

opacity shift
dimensions
representational
representational
representational
performance anti-patterns
Going Meta

To bore your audience (at best) and annoy it (at worst), talk about your presentation within the presentation.
Shortchanging useful topic time

Talking about something only you care about

Negative foreshadowing
architecture katas

documenting and presenting your architecture
Reactive Architecture
source code

https://github.com/wmr513/reactive
reactive architecture

reactive manifesto

- responsive
- elastic
- resilient
- message driven
reactive architecture

reactive manifesto

the system responds in a consistent, rapid, and timely manner whenever possible

how the system reacts to users
reactive architecture

reactive manifesto

the system stays responsive after a failure through replication, containment, isolation, and delegation

*how the system reacts to failures*
reactive architecture

reactive manifesto

the system stays responsive under varying workload

how the system reacts to load
reactive architecture

reactive manifesto

the system relies on asynchronous messaging to ensure loose coupling, isolation, location transparency, and error delegation

how the system reacts to events
reactive patterns for self-healing systems
Thread Delegate Pattern
thread delegate pattern

how can you ensure timely and consistent response time as your system grows?
thread delegate pattern

how can you ensure timely and consistent response time as your system grows?
thread delegate pattern

let’s see the issue...
thread delegate pattern

thread delegate vs. consumer supervisor

scalability
consistent consumers
decoupled event processors
near-linear performance

elasticity
variable consumers
coupled event processors
diminishing performance
thread delegate pattern

thread delegate vs. consumer supervisor

Concurrent Consumers (zoomed in)
(500 Messages / 100ms Processing Time)
thread delegate pattern

thread delegate vs. consumer supervisor

scalability
do coupled event processors
near-linear performance
can preserve message order

elasticity
variable consumers
coupled event processors
diminishing performance
message order not preserved
thread delegate pattern

preserving message order
thread delegate pattern

preserving message order

premise: not every message must be ordered, but rather messages within a context must be ordered

1. PLACE AAPL A-136 2,000,000.00
2. CANCEL AAPL A-136 2,000,000.00
3. REBOOK AAPL A-136 1,800,000.00

1. PLACE AAPL A-136 2,000,000.00
2. PLACE GOOG V-976 650,000.00
3. CANCEL GOOG V-976 650,000.00
4. CANCEL AAPL A-136 2,000,000.00
5. REBOOK AAPL A-136 1,800,000.00
6. REBOOK GOOG V-976 600,000.00
thread delegate pattern

preserving message order

event producer

event delegate

GOOG

AAPL
thread delegate pattern

preserving message order
while (true) {
    //get the next message from the queue
    //get next available thread
    //send message to thread (or start new thread)
}
thread delegate pattern

let’s see the result...
Consumer Supervisor Pattern
consumer supervisor pattern

how can you react to varying changes in load to event consumers to ensure consistent response time?
consumer supervisor pattern

how can you react to varying changes in load to event consumers to ensure consistent response time?
consumer supervisor pattern

let’s see the issue....
consumer supervisor pattern

periodically monitor queue depth
determine consumers needed (e.g., depth/2)
apply max threshold
add or remove consumers
consumer supervisor pattern

```java
List<MyConsumer> consumers = new ArrayList<MyConsumer>();

private void startConsumer() {
    MyConsumer consumer = new MyConsumer();
    consumers.add(consumer);
    new Thread() {
        public void run() {
            consumer.startup(connection);
        }
    }.start();
}

private void stopConsumer() {
    if (consumers.size() > 1) {
        AMQPConsumer consumer = consumers.get(0);
        consumer.shutdown();
        consumers.remove(consumer);
    }
}
```
public void execute() throws Exception {
    startConsumer();
    while (true) {
        Thread.sleep(1000);
        long queueDepth = getMessageCount("trade.eq.q");
        long consumersNeeded = queueDepth/2;
        long diff = Math.abs(consumersNeeded - consumers.size());
        for (int i=0;i<diff;i++) {
            if (consumersNeeded > consumers.size())
                startConsumer();
            else
                stopConsumer();
        }
    }
}
consumer supervisor pattern

let’s see the result...
Workflow Event Pattern
how can you handle error conditions without failing the transaction?
how can you handle error conditions without failing the transaction?
workflow event pattern

let’s see the issue...
workflow event pattern

example

while asynchronously processing trades an error occurs with one of the trade orders
workflow event pattern

trading client → send error to queue

trade processor → programmatically fix error

→ resubmit to processing queue

→ send error to dashboard

human fixes error → resubmit to processing queue
//get next message from queue
String newMsg = msg.substring(0, msg.indexOf(" shares"));
//resubmit message
consumer supervisor pattern

let’s see the result...
Producer Control Flow Pattern
producer control flow pattern

how can you slow down message producers when the messaging system becomes overwhelmed?
producer control flow pattern

how can you slow down message producers when the messaging system becomes overwhelmed?

event producer \rightarrow \text{event channel} \rightarrow \text{flow monitor} \rightarrow \text{event consumer}

\text{slow down!}
producer control flow pattern

how can you slow down message producers when the messaging system becomes overwhelmed?

shutdown (broker) vs. slowdown (pattern)
producer control flow pattern

let's see the issue....
producer control flow pattern

wait for upper threshold
tell producers to slow down
wait for lower threshold
tell producers to resume
public void execute() throws Exception {
    long threshold = 10;
    boolean controlFlow = false;
    while (true) {
        Thread.sleep(3000);
        long queueDepth = getMessageCount("trade.eq.q");
        if (queueDepth > threshold && !controlFlow) {
            controlFlow = enableControlFlow(channel);
        } else if (queueDepth <= (threshold/2) && controlFlow) {
            controlFlow = disableControlFlow(channel);
        }
    }
}
private boolean enableControlFlow(Channel channel) {
    byte[] msg = String.valueOf(true).getBytes();
    // send message to producer
    return true;
}

private boolean disableControlFlow(Channel channel) {
    byte[] msg = String.valueOf(false).getBytes();
    // send message to producer
    return false;
}
public void startListener() {
    new Thread() {
        public void run() {
            while (true) {
                //wait for message from flow monitor
                boolean controlFlow =
                    new Boolean(new String(msg.getBody())).booleanValue();
                synchronized(delay) { delay = controlFlow ? 3000 : 0; }
            }
        }
    }.start();
}

private void produceMessages() {
    Thread.sleep(delay);
    //send trade to queue...
}
producer control flow pattern

let’s see the result…
reactive patterns for self-healing systems

- **Event Producer**
- **Thread Delegate Pattern**
- **Supervisor**
- **Event Dispatcher**
- **Thread Delegate**
- **Flow Monitor**
- **Producer Control Flow Pattern**
- **Workflow Processor**
- **Workflow Event Pattern**

The diagram illustrates the relationships and interactions between these components in a self-healing system.
Evolutionary Architectures

with Rebecca Parsons & Pat Kua
Dynamic Equilibrium
Definition:

An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.
Dimensions of Architecture:

Technical: The implementation parts of the architecture

Data: Database schemas, table layouts, optimization planning, etc.

Security: Defines security policies, guidelines, and specifies tools to help uncover deficiencies.

Domain:
Domain Driven Design
Bounded Context
Evolvability of Architectures
Big Ball of Mud

coupling connections

classes

dimensions: 479
Layered Architecture

presentation layer

business layer

persistence layer

database layer

opportunities: 4

dimensions: 1
Layered Architecture

request

presentation layer

business layer

persistence layer

database layer
opportunities for evolution = L - (2 x L°)
L : # of layers
L° : # of open layers
Microkernel

plug-in component

plug-in component

plug-in component

plug-in component

plug-in component

plug-in component

plug-in component

plug-in component

core system

dimensions : 1
REST

dimensions: 1
Domain Shift

presentation layer
business layer
persistence layer
database layer

domain dimensions:
Microservices

Evolutionary architecture dimensions:
Definition:

Evolutionary architecture

An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.
Composability
Composability
Composability
Definition:

An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.
Incremental Change

Components are deployed.

Features are released.

Applications consist of routing.
Incremental Change
Definition:

An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.
Architecture Fitness Function

metrics
tests
Scope

application

integration

process
Scope

application

integration

process
/**
 * Tests that a package dependency cycle does not exist for any of the analyzed packages.
 */
public void testAllPackages() {
    Collection packages = jdepend.analyze();
    assertEquals("Cycles exist",
                 false, jdepend.containsCycles());
}
Coupling Fitness Function

```java
protected void setUp() throws IOException {
    jdepend = new JDepend();
    jdepend.addDirectory("/path/to/project/util/classes");
    jdepend.addDirectory("/path/to/project/ejb/classes");
    jdepend.addDirectory("/path/to/project/web/classes");
}

public void testMatch() {
    DependencyConstraint constraint = new DependencyConstraint();
    JavaPackage ejb = constraint.addPackage("com.xyz.ejb");
    JavaPackage web = constraint.addPackage("com.xyz.web");
    JavaPackage util = constraint.addPackage("com.xyz.util");

    ejb.dependsUpon(util);
    web.dependsUpon(util);

    jdepend.analyze();

    assertEquals("Dependency mismatch",
                 true, jdepend.dependencyMatch(constraint));
}
```
Fitness Function Fit

![Fitness Function Diagram]

- Auditability
- Usability
- Configurability
- Monitorability
- Data Security
- Low Latency
- High Availability
- Legal Requirements
- Internationalization
- High Throughput

Blue line indicates the fit level for each criterion.
Guided Evolution
Definition:

An evolutionary architecture supports incremental, guided change as a first principle across multiple dimensions.
utilizing evolutionary architecture
1. choose dimensions

"-ilities"

evolutionary
change

testable
2. identify fitness functions

atomic / holistic

automated / manual
3. apply incremental change
initial & continual
architecture katas

identifying evolutionary architecture factors
next steps
next steps

technical breadth

focus here!

stuff you know

stuff you know you don't know

stuff you don't know you don't know

certification

nomination packet

personal radar
next steps

www.infoq.com

www.dzone.com

https://www.thoughtworks.com/radar
next steps
O’Reilly Free Reports

http://www.wmrichards.com/publications
Mark Richards
Boston Area Independent Consultant, Hands-on Software Architect, Published Author

I am a hands-on software architect with over 30 years experience in the industry, 20 of those years having played the role of an application architect, integration architect, and enterprise architect. I have experience creating and delivering Microservices Architectures, Service-Based Architectures, and...

Architecture Training
I teach a highly interactive 3 day architecture fundamentals training
Creating Software Architectures

**Neal Ford**  
*ThoughtWorks*  
Director / Software Architect / Meme Wrangler  
http://www.nealford.com  
@neal4d

**Mark Richards**  
*Hands-on Software Architect*  
Published Author / Conference Speaker  
http://www.wmrichards.com  
https://www.linkedin.com/in/markrichards3  
@markrichardssa