The Rise and Fall of Microservices

Mark Richards
Independent Consultant
Hands-on Software Architect / Published Author
Founder, DeveloperToArchitect.com

http://www.wmrichards.com
https://www.linkedin.com/in/markrichards3
@markrichardssa
Pars occidentalis

Pars orientis

Provincia
Diocesis
Provincia proconsularis

PARTITIO IMPERII
AD 395
The Rise of Microservices
Agility Essential as Businesses Face Rapid Pace of Change

Speeds are one of the few constants in an increasingly digital business world. Consumers change their preferred apps and services quickly. They look for better ways to interact with the businesses they depend on all the time. They expect companies to respond at a rapid pace when they provide feedback or interact with customer service teams. In response, businesses must change with the same quickness, creating a sense of urgency that legacy operational models cannot handle.
Agility Essential as Businesses Face Rapid Pace of Change

Speed is one of the few constants in an increasingly digital business world. Consumers change their preferred apps and services quickly. They look for better ways to interact with the businesses they depend on all the time. They expect companies to respond at a rapid pace when they provide feedback or interact with customer service teams. In response, businesses must change with the same quickness, creating a sense of urgency that legacy operational models cannot handle.
Agility Essential as Businesses Face Rapid Pace of Change

Speed is one of the few constants in an increasingly digital business world. Consumers change their preferred apps and services quickly. They look for better ways to interact with businesses to which they depend all the time. They expect companies to respond at a rapid pace when they provide feedback or interact with customer service teams. In response, businesses must change with the same quickness, creating a sense of urgency that legacy operational models cannot handle.
The Decline and Fall of Microservices
dealing with code reuse in a “share nothing” architecture
ignoring the challenges and issues associated with distributed computing
jumping on the bandwagon of microservices without really needing it
putting too much business logic in the API gateway
relying only on restful communications between services
creating services that are too fine-grained
using a monolithic user interface with backend microservices
devops became just another term for operational tools and automation
not embracing the organizational change required for microservices
focusing too much on functionality and not enough about data
dealing with code reuse in a “share nothing” architecture
the decline and fall of microservices

reuse is abuse!

please repeat yourself!

share nothing!!
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices

code replication

```java
@ServiceEntrypoint
public class PaymentServiceAPI {
    ...
}
```

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface ServiceEntrypoint {}
the decline and fall of microservices

code replication

base image
(service template)
the decline and fall of microservices

code replication

- very low code volatility (static code)
- compile-based code
- code changes due to bugs or added functionality
- difficult to expand (add additional classes)
the decline and fall of microservices
shared library

ServiceEntrypoint.java

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface ServiceEntrypoint {}

annotations.jar v1.0

ServiceEntrypoint.java
OrchestrationService.java
DataService.java
the decline and fall of microservices

shared library
the decline and fall of microservices

shared library
the decline and fall of microservices

shared library

[Diagram of a shared library]
the decline and fall of microservices

shared library
the decline and fall of microservices

shared library
the decline and fall of microservices

shared library
the decline and fall of microservices
shared library
the decline and fall of microservices

shared library

- ✓ low code volatility
- ✓ homogeneous code
- ✓ ability to version changes
- ✓ easy to expand (add additional classes)
- ✗ version adoption and deprecation
the decline and fall of microservices

shared service
the decline and fall of microservices

shared service
the decline and fall of microservices

shared service

- high code volatility
- heterogeneous (polyglot) code
- versioning is difficult (runtime changes)
- performance issues due to latency
- availability / fault tolerance issues
- scalability / throughput issues
- greater chance of breaking things
the decline and fall of microservices

service consolidation
the decline and fall of microservices

service consolidation

✓ no code sharing!
ignoring the challenges and issues associated with distributed computing
the decline and fall of microservices

driving without a distributed architecture license
the decline and fall of microservices

en.wikipedia.org/wiki/Fallacies_of_distributed_computing
the decline and fall of microservices

1. the network is reliable
the decline and fall of microservices

2. latency is zero

\[ T_{\text{remote}} > T_{\text{local}} \]
the decline and fall of microservices

3. bandwidth is infinite
the decline and fall of microservices

3. bandwidth is infinite
the decline and fall of microservices

4. the network is secure
the decline and fall of microservices

5. topology doesn’t change
the decline and fall of microservices

6. there is one administrator
the decline and fall of microservices

6. there is one administrator
the decline and fall of microservices

7. transport cost is zero
the decline and fall of microservices

8. the network is homogeneous
the decline and fall of microservices

contract creation, maintenance, versioning, and coordination

the decline and fall of microservices

distributed logging facilities to provide a holistic view of a transaction

the decline and fall of microservices

distributed transaction management

jumping on the bandwagon of microservices without really needing it
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices
the decline and fall of microservices

we should have focused on the business drivers first!
## the decline and fall of microservices

<table>
<thead>
<tr>
<th>Feature</th>
<th>Layered Monolith</th>
<th>Microkernel</th>
<th>Microservices</th>
<th>Service-Based</th>
<th>Event-Driven</th>
<th>Space-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agility</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Deployment</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Testability</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Performance</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Scalability</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Elasticity</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Simplicity</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Fault-tolerance</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Evolvability</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Total Cost</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
# The Decline and Fall of Microservices

<table>
<thead>
<tr>
<th>Feature</th>
<th>Layered Monolith</th>
<th>Microkernel</th>
<th>Microservices</th>
<th>Service-Based</th>
<th>Event-Driven</th>
<th>Space-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agility</td>
<td>★★</td>
<td>★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
</tr>
<tr>
<td>Deployment</td>
<td>★</td>
<td>★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
</tr>
<tr>
<td>Testability</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Performance</td>
<td>★★★★</td>
<td>★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
</tr>
<tr>
<td>Scalability</td>
<td>★</td>
<td>★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
</tr>
<tr>
<td>Elasticity</td>
<td>★</td>
<td>★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
</tr>
<tr>
<td>Simplicity</td>
<td>★★★★★</td>
<td>★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
</tr>
<tr>
<td>Fault-Tolerance</td>
<td>★</td>
<td>★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Evolvability</td>
<td>★</td>
<td>★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>Total Cost</td>
<td>★★★★★★</td>
<td>★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
<td>★★★★★</td>
</tr>
</tbody>
</table>
### The Decline and Fall of Microservices

<table>
<thead>
<tr>
<th>Feature</th>
<th>Layered Monolith</th>
<th>Microkernel</th>
<th>Microservices</th>
<th>Service-Based</th>
<th>Event-Driven</th>
<th>Space-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agility</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Deployment</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Testability</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Performance</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Scalability</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Elasticity</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Simplicity</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Fault-Tolerance</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Evolvability</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Total Cost</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
<td>🟠🌟🌟🌟🌟🌟</td>
</tr>
</tbody>
</table>
the decline and fall of microservices
the decline and fall of microservices

agility

testability

deployability

time to market

scalability

availability
the decline and fall of microservices

competitive advantage

agility

testability

deployability

scalability

availability

time to market
the decline and fall of microservices
the decline and fall of microservices

not every portion of an application has to be microservices
creating services that are too fine-grained
service granularity

- service functionality
- database transactions
- data dependencies
- workflow and choreography
service granularity

service functionality

database transactions

data dependencies

workflow and choreography
service granularity

service scope and function
(single-purpose function)
service granularity

- service functionality
- database transactions
- data dependencies
- workflow and choreography
service granularity

- service functionality
- database transactions
- data dependencies
- workflow and choreography
service granularity

database transactions

no acid transaction
service granularity

- service functionality
- database transactions
- data dependencies
- workflow and choreography
service granularity

service functionality

database transactions

data dependencies

workflow and choreography
service granularity
service granularity
service granularity
service granularity

function a
function b
function c

service a

service b

service c

data dependencies
service granularity
service granularity

function a
function b
function c

service component

service a
service b
service c

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

1 2 3 4 5 6

data dependencies
service granularity
service granularity

- Service component:
  - function a
  - function b
  - function c

- Service a:
  - module 1
  - module 2
  - module 3
  - module 4
  - module 5
  - module 6

- Service b/c:
  - module 3
  - module 5

Data dependencies:

1 2 3 4 5 6
service granularity
service granularity

service functionality

database transactions

data dependencies

workflow and choreography
service granularity

workflow and choreography
service granularity

fault tolerance (availability)

workflow and choreography
service granularity

workflow and choreography

performance due to latency
service granularity

workflow and choreography

reliability and data consistency
service granularity

workflow and choreography
service granularity

- service functionality
- database transactions
- data dependencies
- workflow and choreography
service granularity

database transactions

data dependencies

workflow and choreography
is there a way to stop the decline and fall of microservices?
avoid jumping on the bandwagon; know when to use microservices and why
understand and analyze the challenges of microservices and determine whether it is the right architectural fit
embrace modularity but be cautious of granularity
Developer to Architect

Training and resources for the journey from developer to software architect

Mark Richards, Software Architect and Founder

Developer To Software Architect

“The journey from developer to software architect is a difficult and uncharted path filled with lots of challenges, pitfalls, and confusion. The purpose of DeveloperToArchitect.com is to provide resources and training to help you along the journey to becoming an effective software architect”

Mark Richards, Software Architect, Founder of DeveloperToArchitect.com

Software Architecture Public Class Dates and Locations for 2019

Software Architecture Fundamentals Class
October 21-23, Denver, CO
November 13-15, Boston, MA
The Rise and Fall of Microservices

Mark Richards
Independent Consultant
Hands-on Software Architect / Published Author
Founder, DeveloperToArchitect.com

http://www.wmrichards.com
https://www.linkedin.com/in/markrichards3
@markrichardssa