Scaling agile development across loosely coupled teams using microservice architecture

Anastas Stoyanovsky
Steven Pritko
9 August 2018

Watson
Introduction
Introduction

• IBM Watson and IBM Research are separate organizations with different goals that collaborate often

• Every time a new collaboration between organizations starts, two or more teams need to find a way of working together well

• Observation: teams from different organizations tend to have different norms and habits that align with their incentive structures

• This is a case study of one approach to establishing development process in such a new collaboration via system architecture

• With the benefit of hindsight, we extract an architecture-first approach to facilitating cross-organization collaboration
Outline

- Quality attributes and organizational incentives
- Microservice architecture and architectural hoisting
- Case study: IBM Watson / IBM Research
- Results
- Possible Improvements
- Summary
- Q&A
Quality Attributes and Organizational Incentives
Have you heard the phrase “research code”? If so, how would you define it?
“Why?”
“Research code” isn’t “bad code” – it’s code written for a specific purpose.
The quality attributes (QAs) of a system are various factors which affect its behavior, usability, and design across all its components.
Given an existing platform,
“Research Code”

Generally written for **accuracy**

Generally *not* written for **maintainability**
QA tradeoffs
A research organization generally incentivizes *publishing improvements on the state of the art* in academic journals.

A research organization generally does *not* incentivize *shipping performant, maintainable code* in a software product.
When teams from two different organizations with different incentive structures collaborate, can we design our software architecture to avoid the tension between the quality attributes that we *know* the different teams will focus on?
Microservice architecture and architectural hoisting
Microservice architecture

“Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities.”

Source: microservices.io
What is a microservice?

Source: https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4
Architectural hoisting

“Architectural hoisting is the direct ownership, management, or guarantee by the architecture of a feature, property, or quality attribute. It lets developers depend on the architecture for the hoisted feature, property, or quality.”

- George Fairbanks

Source: http://www.georgefairbanks.com/blog/architectural-hoisting-original/
When teams from two different organizations with different incentive structures collaborate, can we design our software architecture to avoid the tension between the quality attributes that we know the different teams will focus on?
Hoist certain QAs into the architecture itself
<table>
<thead>
<tr>
<th>Research Team</th>
<th>System Architecture</th>
<th>Engineering Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Reliability</td>
<td>Performance</td>
</tr>
<tr>
<td></td>
<td>Fault Tolerance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modularity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Case study:
IBM Watson / IBM Research
Background

- Existing platform to build on
- Novel task to be solved; known inputs/outputs
- Teams from IBM Watson, IBM Research picked to solve and implement
- Using architectural hoisting, engineering team iterated on system design while research iterated on algorithm development
- Algorithmic solution had not been developed but development had to begin
Initial design

A → B  A calls/submits to B
A ← B  A reads from B
Intermediate design

Proxy

REST Facade

Model Serving

Ensembler

Algorithm

Storage

Model Training

Proxy

Algorithm

Algorithm

Document Preprocessor

Message Queue

Ingestion

Data

Serve time

Train time

Ingest time
Useful Patterns
Useful patterns

• Fixed interfaces for multiple provider backends
Useful patterns

- Fixed interfaces for multiple provider backends
- Message queue for events
Useful patterns

- Fixed interfaces for multiple provider backends
- Message queue for events
- Pub/sub for state changes
A ➔ B Serve time
A ➔ B Train time
A ➔ B Ingest time

* synchronization

Ensembler

Algorithm

Algorithm

Model Server

Model Training

Storage

Proxy

Data

Ingestion

Training Database

Raw Document Store

Processed Document Store

Unsupervised Data Extraction

Message Queue

Document Preprocessor

Preprocessor

Pub* sub*

Pub* sub*

Pub* sub*

Pub* sub*
Useful patterns

• Fixed interfaces for multiple provider backends
• Message queue for events
• Pub/sub for state changes
• Proxies, facades
Results and Retrospectives
Results

- Architecture flexibility minimized the impact of changing requirements and scope creep
- Final product with a cohesive design and all desired QAs (we’d argue)
- Journal submissions made with improvements to the state of the art
- Significantly reduced communication and management overhead
- Happy teams
Retrospectives

- Decoupled *too* strongly for a bit
- Mutually exclusive solutions to the same problem from both teams
- Double edged sword: flexibility allowed loss of synchronization on requirements and design decisions
Possible Improvements
Possible Improvements

- Record architectural design decisions
  - e.g. ADRs (architectural design records)
- Encourage cross-team code review or periodic knowledge sharing
- At least some minimal formal project management role
Summary
Teams in different organizations may have different incentive structures.

Teams with different incentive structures may naturally prioritize certain QAs.

In a cross-organization collaboration, these different prioritizations may be in tension with one another.

Architectural hoisting can be an approach to resolve such a tension.

Microservice architectures can make architectural hoisting easier.

Be careful not to decouple teams too strongly – hoisting is not magic.