Post-hoc systems architecture: An argument for microservices first

Anastas Stoyanovsky
Charles Gala
Aiton Goldman

IBM Watson
8 Aug 2019
Outline

- Introduction
- Background
- Problem Definition
- Proposed Approach
- Examples
  - Information Retrieval Architecture
  - Machine Learning Train Architecture
- Discussion
- Conclusion
Opening Question
Should I design my new product using a microservice architecture?
Opening Question

Should I design my new product using a microservice architecture?

Depends who you ask!
Rephrasing the Question
Rephrasing the Question

Under what circumstances do the benefits of starting with microservices significantly outweigh the risks?
A Starting Suggestion

Under what circumstances do the benefits of starting with microservices significantly outweigh the risks?

One scenario may be when development is happening concurrently with product design and/or with shifting external dependencies.
Background
Let’s start by reviewing the general arguments in either direction, starting with “Microservices first!”

e.g. [https://martinfowler.com/articles/dont-start-monolith.html](https://martinfowler.com/articles/dont-start-monolith.html)
Microservices First!

E.g. https://martinfowler.com/articles/dont-start-monolith.html
Microservices First!

Fast delivery of independent parts

e.g. https://martinfowler.com/articles/dont-start-monolith.html
Microservices First!

Fast delivery of independent parts
Easier to set boundaries during initial component creation
to prevent casual code coupling

e.g. https://martinfowler.com/articles/dont-start-monolith.html
Microservices First!

Fast delivery of independent parts

Easier to set boundaries during initial component creation to prevent casual code coupling

It would be harder to later break up tightly coupled code after starting with a monolith – and you will probably have tightly coupled code

e.g. https://martinfowler.com/articles/dont-start-monolith.html
Monolith First!

e.g. [https://martinfowler.com/bliki/MonolithFirst.html](https://martinfowler.com/bliki/MonolithFirst.html)
Monolith First!

Higher premium to maintain microservices

e.g. https://martinfowler.com/bliki/MonolithFirst.html
Monolith First!

Higher premium to maintain microservices

“You ain’t gonna need it”

e.g. https://martinfowler.com/bliki/MonolithFirst.html
Monolith First!

Higher premium to maintain microservices

“You ain’t gonna need it”

High risk of getting bounded contexts wrong

e.g. https://martinfowler.com/bliki/MonolithFirst.html
Monolith First!

Higher premium to maintain microservices

“You ain’t gonna need it”

High risk of getting bounded contexts wrong

e.g. [https://martinfowler.com/bliki/MonolithFirst.html](https://martinfowler.com/bliki/MonolithFirst.html)
Shift of Perspective
Shift of Perspective

If you accept that you'll get the bounded contexts wrong and budget for it, that is no longer a risk - it's just strategic technical debt.
Shift of Perspective

If you accept that you'll get the bounded contexts wrong and budget for it, that is no longer a risk - it's just strategic technical debt.

The question then becomes how to best intentionally take on that tech debt, and how to pay it off optimally.
Problem Definition
Problem Definition

*If we accept getting bounded contexts wrong as strategic technical debt:*
If we accept getting bounded contexts wrong as strategic technical debt:

How can we take advantage of microservice architecture to expedite initial release as much as possible,
Problem Definition

*If we accept getting bounded contexts wrong as strategic technical debt:*

How can we take advantage of microservice architecture to expedite initial release as much as possible, while also minimizing the long-term cost of strategic technical debt incurred along the way?
Proposed Approach
Our Proposal

• When there is high uncertainty in product design or future requirements but development needs to start, it can be an effective strategy to start with a microservice architecture.
Our Proposal

• When there is high uncertainty in product design or future requirements but development needs to start, it can be an effective strategy to start with a microservice architecture

• View incorrect bounded contexts as technical debt and not as risk
  • Do make best effort during design phase
Our Proposal

- When there is high uncertainty in product design or future requirements but development needs to start, it can be an effective strategy to start with a microservice architecture.
- View incorrect bounded contexts as technical debt and not as risk.
  - Do make best effort during design phase.
- Address new requirements and use cases with new components (but judiciously).
Our Proposal

• When there is high uncertainty in product design or future requirements but development needs to start, it can be an effective strategy to start with a microservice architecture

• View incorrect bounded contexts as technical debt and not as risk
  • Do make best effort during design phase

• Address new requirements and use cases with new components (but judiciously)

• Ship fast while tracking strategic technical debt
Our Proposal

• When there is high uncertainty in product design or future requirements but development needs to start, it can be an effective strategy to start with a microservice architecture

• View incorrect bounded contexts as technical debt and not as risk
  • Do make best effort during design phase

• Address new requirements and use cases with new components (but judiciously)

• Ship fast while tracking strategic technical debt

• After release, address only the technical debt in mission-critical microservices
  • Focus on increasing maintainability and enabling development agility
  • Ignore technical debt in isolated, non-critical components until forced to
Risk Minimization
Risk Minimization

- Avoid investing in features that see no usage but that could increase overhead
Risk Minimization

• Avoid investing in features that see no usage but that could increase overhead
• Decrease impact of unforeseen legal or platform requirements
Risk Minimization

- Avoid investing in features that see no usage but that could increase overhead
- Decrease impact of unforeseen legal or platform requirements
- Allow late phase addition of new functional requirements (though with higher debt)
Risk Minimization

• Avoid investing in features that see no usage but that could increase overhead
• Decrease impact of unforeseen legal or platform requirements
• Allow late phase addition of new functional requirements (though with higher debt)
• Avoid unintentional code coupling that can harm team velocity
How do I know what tech debt has the highest interest rate?
How do I know what tech debt has the highest interest rate?

• We unintentionally carried out our proposed formal strategy
How do I know what tech debt has the highest interest rate?

- We unintentionally carried out our proposed formal strategy
- Looking back empirically, we identify useful metrics for driving decisions on what debt to pay off
How do I know what tech debt has the highest interest rate?

• We unintentionally carried out our proposed formal strategy

• Looking back empirically, we identify useful metrics for driving decisions on what debt to pay off
  • Tech Debt
    • How much code needs to be refactored?
    • How hard is it to modify the codebase?
How do I know what tech debt has the highest interest rate?

• We unintentionally carried out our proposed formal strategy
• Looking back empirically, we identify useful metrics for driving decisions on what debt to pay off
  • Tech Debt
    • How much code needs to be refactored?
    • How hard is it to modify the codebase?
  • High versus Low Touch
    • How often does a component get modified?
Metrics
Metrics

Tech Debt Metrics

• Repository size
• Average number of lines modified per PR
• Percentage of production log output
• Test coverage
Metrics

**Tech Debt Metrics**
- Repository size
- Average number of lines modified per PR
- Percentage of production log output
- Test coverage

**High / Low Touch Metrics**
- Commits per PR
- Comments per PR
- Mean, standard deviation of time between commits
Metrics

Tech Debt Metrics

• Repository size
• Average number of lines modified per PR
• Percentage of production log output
• Test coverage

High / Low Touch Metrics

• Commits per PR
• Comments per PR
• Mean, standard deviation of time between commits
Case Study

- We will use ourselves as a case study
Case Study

- We will use ourselves as a case study
- We informally adopted the approach we suggest here, over several years
Case Study

- We will use ourselves as a case study
- We informally adopted the approach we suggest here, over several years
- We give two examples of how this informal approach manifested
Case Study

• We will use ourselves as a case study
• We informally adopted the approach we suggest here, over several years
• We give two examples of how this informal approach manifested
• We will identify commonalities across those examples
Case Study

• We will use ourselves as a case study
• We informally adopted the approach we suggest here, over several years
• We give two examples of how this informal approach manifested
• We will identify commonalities across those examples
• After each example, using benefit of hindsight, we identify basic metrics to help make this approach both formal and intentional
Example: Information Retrieval Architecture
Background

• Our team’s MVP was to improve an existing cloud search offering MVP using both machine learning and enhanced information retrieval methods
Background

• Our team’s MVP was to improve an existing cloud search offering MVP using both machine learning and enhanced information retrieval methods

• Our cloud organization was developing as we were implementing
Background

• Our team’s MVP was to improve an existing cloud search offering MVP using both machine learning and enhanced information retrieval methods

• Our cloud organization was developing as we were implementing

• New features and use cases were identified, often well after development started
Shipping Quickly
(or: Accumulating Technical Debt)
Product MVP

Legend

- □ Microservice
- 🏗️ Data Storage

A calls B

A → B  A calls B
Our Team’s Intended MVP

Legend

- Microservice
- Data Storage

A calls B
Our Team’s Actual MVP

Legend

- Microservice
- Data Storage

A calls B

A → B A calls B

Training Data

Subsystem

ML

Search

Query

Document Store

Redis

Late-Phase Functional Requirement

Differentiator
Our Team’s Actual MVP

Legend
- Microservice
- Data Storage

A calls B

Diagram:
- Training Data
- Subsystem
- ML
- Query
- Redis
- Differentiator
- Document Store

Connections:
- A calls B
- A → B
- B → A
But wait! There’s more!*
Solve it with architecture!

Legend

- Microservice
- Data Storage
- A calls B
Solve it with architecture!

Legend

- Microservice
- Data Storage

A calls B
Summary

• Differentiating use case was added late and was implemented with a new component
  • Isolated significant technical debt and shipped
Summary

• Differentiating use case was added late and was implemented with a new component
  • Isolated significant technical debt and shipped

• New platform requirements were met with a new deployable component
  • Solution was contained in one sidecar deployed in front of multiple components subject to the requirements
Summary

• Differentiating use case was added late and was implemented with a new component
  • Isolated significant technical debt and shipped

• New platform requirements were met with a new deployable component
  • Solution was contained in one sidecar deployed in front of multiple components subject to the requirements

• Shared concerns were identified between two components, Search and ML
  • One component was low-touch; we were able to ignore this problem until much later
Summary

- Differentiating use case was added late and was implemented with a new component
  - Isolated significant technical debt and shipped
- New platform requirements were met with a new deployable component
  - Solution was contained in one sidecar deployed in front of multiple components subject to the requirements
- Shared concerns were identified between two components, Search and ML
  - One component was low-touch; we were able to ignore this problem until much later
- This process can be characterized as an expansion phase
Paying off Tech Debt

• Large amount of tech debt in the mission-critical *Search* component
  • Significantly hampered further feature development
Paying off Tech Debt

• Large amount of tech debt in the mission-critical *Search* component
  • Significantly hampered further feature development

• Large amount of tech debt in the *Differentiator* component, which did not see much usage
  • Did not require changes often
Paying off Tech Debt

- Large amount of tech debt in the mission-critical Search component
  - Significantly hampered further feature development
- Large amount of tech debt in the Differentiator component, which did not see much usage
  - Did not require changes often
- Incorrect bounded contexts across Search, ML components was technical debt
  - Ignored until new feature development required similar changes to both components
  - At that time, we merged those services
Identifying Tech Debt: Rewriting

Legend
- Microservice
- Data Storage

A calls B

Diagram:
- Training Data
- Subsystem
- ML
- Proxy
- Search
- Query
- Redis
- Differentiator

Legend:
- Microservice
- Data Storage
  A calls B
Identifying Tech Debt: Rewriting
New and Improved Search!

Legend
- Microservice
- Data Storage
- A calls B
New and Improved Search!

Legend
- Microservice
- Data Storage

A calls B
A
B
A calls B
"Mistakes" in our bounded contexts

Legend

- Microservice
- Data Storage
- A calls B

A → B

Diagram:
- Training Data
- Subsystem
- ML
- Search v2
- Query
- Redis
- Differentiator
- Proxy
- Document Store
- NGX
"Mistakes" in our bounded contexts

Legend

- Microservice
- Data Storage

A calls B
Merge the two together

Legend

- Microservice
- Data Storage

A calls B
Paying off Tech Debt: Deleting

Legend

- Microservice
- Data Storage

A calls B  A calls B
Achieving Relative Stability

Legend
- Microservice
- Data Storage

A calls B
A → B  A calls B
## Metrics: High Touch, Paid Off Tech Debt

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of PRs (z-score)</th>
<th>Unit Test Coverage</th>
<th>Hours between Commits (Mean)</th>
<th>Hours Between Commits (STD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>1.09</td>
<td>0.53</td>
<td>7.70</td>
<td>39.03</td>
</tr>
<tr>
<td>ML</td>
<td>-0.11</td>
<td>0.50</td>
<td>40.44</td>
<td>317.65</td>
</tr>
<tr>
<td>Search + ML</td>
<td>0.89</td>
<td>0.70</td>
<td>8.75</td>
<td>46.14</td>
</tr>
</tbody>
</table>
Unit Test Coverage

- Search: 0.5
- ML: 0.5
- Search + ML: 0.7
Summary

- Isolation of technical debt per component allowed us to focus on technical debt in mission critical components
  - If we had shipped with a monolith, code coupling may have made paying this off much more expensive
Summary

- Isolation of technical debt per component allowed us to focus on technical debt in mission critical components
  - If we had shipped with a monolith, code coupling may have made paying this off much more expensive
  - Incorrect bounded contexts were addressed as technical debt when they became a problem
Summary

- Isolation of technical debt per component allowed us to focus on technical debt in mission critical components
  - If we had shipped with a monolith, code coupling may have made paying this off much more expensive
  - Incorrect bounded contexts were addressed as technical debt when they became a problem
  - Removal of platform requirement was addressed by simply removing a deployable component
Summary

• Isolation of technical debt per component allowed us to focus on technical debt in mission critical components
  • If we had shipped with a monolith, code coupling may have made paying this off much more expensive
  • Incorrect bounded contexts were addressed as technical debt when they became a problem
  • Removal of platform requirement was addressed by simply removing a deployable component
  • This process can be characterized as a contraction phase
Example: Machine Learning Training Architecture
Background

• Recall that part of our team’s MVP was to introduce machine learning capabilities
Background

• Recall that part of our team’s MVP was to introduce machine learning capabilities
• We inherited a legacy service that was ported from one platform (where it had been ported from another)
  • Very high technical debt and difficult to work in
  • Won’t be focusing on this, as we are discussing greenfield development and not legacy refactoring
Background

• Recall that part of our team’s MVP was to introduce machine learning capabilities

• We inherited a legacy service that was ported from one platform (where it had been ported from another)
  • Very high technical debt and difficult to work in
  • Won’t be focusing on this, as we are discussing greenfield development and not legacy refactoring

• One of our requirements was to automate the manual training process for that legacy product
Training View of the Architecture

Legend

Microservice
Data Storage

A calls B
New Legacy System to Manage

REST → Training Data

Search + ML

Document Store

Legend

Microservice
Data Storage

A → B  A calls B
Agents to Process Training Data
Agents to Process Training Data

Legend

- Microservice
- Data Storage

A calls B
Agents to Process Training Data

REST

Training Data

Search + ML

Document Store

Preparation Agents

Cleanup Agent

Monitoring Agent

Legacy

Legend

Microservice
Data Storage

A calls B
Agents to Process Training Data

REST → Training Data → Search + ML → Document Store

Legend:
- Microservice
- Data Storage

A calls B
New requirement = new service

Legend
- Microservice
- Data Storage
- A calls B
New requirement = new service

Legend

- Microservice
- Data Storage
- A calls B
Interlude

• We introduced several “agents” to automate a previously manual training process
  • We knew that had shared concerns; we shipped fast and labeled that as tech debt
We introduced several “agents” to automate a previously manual training process

- We knew that had shared concerns; we shipped fast and labeled that as tech debt

- We again addressed a late-phase requirement with a new component

- Its tech debt stayed isolated and unaddressed, years later
Opportunities for improvement

Legend
- Microservice
- Data Storage
- A calls B
Opportunities for improvement

Legend

- Microservice
- Data Storage

A calls B
Opportunities for improvement

Legend

- Microservice
- Data Storage
- A calls B
Shared codebase for training agents

Legend
- Microservice
- Data Storage
- A → B A calls B
Shared codebase for training agents

Legend
- Microservice
- Data Storage

A calls B A → B A calls B
GDPR
GDPR!

Legend
- Microservice
- Data Storage

A calls B
GDPR Managed!

Legend:
- Microservice
- Data Storage

A calls B
## Metrics

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of PRs (z-score)</th>
<th>Unit Test Coverage</th>
<th>Hours between Commits (Mean)</th>
<th>Hours between Commits (STD DEV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search +ML</td>
<td>0.89</td>
<td>0.70</td>
<td>8.75</td>
<td>46.14</td>
</tr>
<tr>
<td>Notifications</td>
<td>-0.68</td>
<td>0.28</td>
<td>52.35</td>
<td>301.40</td>
</tr>
<tr>
<td>GDPR as a Service</td>
<td>-1.25</td>
<td>0.53</td>
<td>41.92</td>
<td>249.44</td>
</tr>
</tbody>
</table>
Hours between Commits (Mean)

- Search +ML
- Notifications
- GDPR as a Service
Summary

• Met new requirements with new components (expansion phase)
  • Including GDPR
  • Isolated low-touch tech debt
Summary

• Met new requirements with new components (expansion phase)
  • Including GDPR
  • Isolated low-touch tech debt
• Paid off high touch, high-debt tech debt (contraction phase)
  • Extracted common concerns into architecturally hoisted* shared library

Themes

- New requirements met with new components
  - Strategic technical debt accepted in order to meet deadlines
- High-touch, high tech debt services were rewritten/combined
  - Increase in unit test coverage shows increased maintainability, decreased tech debt
- Low-touch, high tech debt services remained isolated
  - Large standard deviations in commit intervals shows modifications rarely necessary
- The overall process manifested in expansion/contraction cycles
More Statistics
Number of PRs

- Search
- Search + ML
- ML
- Differentiator
- REST
- Training Data
- Preparation Agent
- Training Agents
- Monitoring Agent
- Notifications
- GDPR

Number of PRs
Hours Between Commits (Mean)

- Search
- Search + ML
- ML
- Differentiator
- REST
- Training Data
- Training Agent
- Monitoring Agent
- Notifications
- GDPR
Hours Between Commits (Standard Deviation)

- Search
- Search + ML
- ML
- Differentiator
- REST
- Training Data
- Preparation Agent
- Training Agents
- Monitoring Agent
- Notifications
- GDPR
## Metrics (Comprehensive)

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of PRs (z-score)</th>
<th>Number of Commits (z-score)</th>
<th>Number of Comments (z-score)</th>
<th>Refactor Size (z-score)</th>
<th>Repository Size (z-score)</th>
<th>Percentage Code Refactored (z-score)</th>
<th>Log Count (z-score)</th>
<th>Unit Test Coverage</th>
<th>Time between Commits (Mean)</th>
<th>Time between Commits (STD DEV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>1.09</td>
<td>0.09</td>
<td>0.83</td>
<td>0.13</td>
<td>0.06</td>
<td>-0.78</td>
<td>NA</td>
<td>0.53</td>
<td>7.70</td>
<td>39.03</td>
</tr>
<tr>
<td>Search + ML</td>
<td>0.89</td>
<td>1.25</td>
<td>0.81</td>
<td>1.88</td>
<td>1.26</td>
<td>-0.87</td>
<td>2.84</td>
<td>0.70</td>
<td>8.75</td>
<td>46.14</td>
</tr>
<tr>
<td>ML</td>
<td>-0.11</td>
<td>-1.31</td>
<td>0.16</td>
<td>-0.37</td>
<td>-0.45</td>
<td>-0.34</td>
<td>NA</td>
<td>0.50</td>
<td>40.44</td>
<td>317.65</td>
</tr>
<tr>
<td>Differentiator</td>
<td>0.39</td>
<td>0.06</td>
<td>-0.38</td>
<td>0.35</td>
<td>-0.35</td>
<td>-0.21</td>
<td>-0.28</td>
<td>0.58</td>
<td>26.04</td>
<td>139.32</td>
</tr>
<tr>
<td>REST</td>
<td>0.44</td>
<td>0.24</td>
<td>0.58</td>
<td>0.43</td>
<td>0.70</td>
<td>-0.92</td>
<td>-0.33</td>
<td>0.37*</td>
<td>16.32</td>
<td>61.58</td>
</tr>
<tr>
<td>Training Data</td>
<td>2.31</td>
<td>0.15</td>
<td>0.68</td>
<td>0.07</td>
<td>2.71</td>
<td>-1.09</td>
<td>-0.15</td>
<td>0.56</td>
<td>13.26</td>
<td>51.80</td>
</tr>
<tr>
<td>Preparation Agent</td>
<td>-0.34</td>
<td>-1.08</td>
<td>0.88</td>
<td>-0.47</td>
<td>-0.53</td>
<td>-0.04</td>
<td>NA</td>
<td>0.73</td>
<td>26.38</td>
<td>109.92</td>
</tr>
<tr>
<td>Training Agents</td>
<td>-0.27</td>
<td>0.59</td>
<td>0.80</td>
<td>0.06</td>
<td>-0.46</td>
<td>0.01</td>
<td>-0.35</td>
<td>0.68</td>
<td>26.06</td>
<td>93.90</td>
</tr>
<tr>
<td>Monitoring Agent</td>
<td>-0.80</td>
<td>-1.36</td>
<td>-1.34</td>
<td>-1.12</td>
<td>-0.65</td>
<td>0.23</td>
<td>-0.35</td>
<td>0.55</td>
<td>77.12</td>
<td>282.87</td>
</tr>
<tr>
<td>Notifications</td>
<td>-0.68</td>
<td>-0.15</td>
<td>0.16</td>
<td>-0.83</td>
<td>-0.53</td>
<td>-0.38</td>
<td>-0.35</td>
<td>0.28</td>
<td>52.35</td>
<td>301.40</td>
</tr>
<tr>
<td>GDPR</td>
<td>-1.25</td>
<td>1.56</td>
<td>-0.64</td>
<td>1.84</td>
<td>-0.43</td>
<td>0.92</td>
<td>-0.35</td>
<td>0.53</td>
<td>41.92</td>
<td>249.44</td>
</tr>
</tbody>
</table>

* The majority of the REST component’s testing was in Cucumber tests, instead of unit tests, because it is a REST façade.
# Metrics (Highlighted)

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of PRs (z-score)</th>
<th>Log Output Percentage (z-score)</th>
<th>Unit Test Coverage</th>
<th>Hours between Commits (Mean)</th>
<th>Hours between Commits (STD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>1.09</td>
<td>NA</td>
<td>0.53</td>
<td>7.70</td>
<td>39.03</td>
</tr>
<tr>
<td>Search+ML</td>
<td>0.89</td>
<td>2.84</td>
<td>0.70</td>
<td>8.75</td>
<td>46.14</td>
</tr>
<tr>
<td>ML</td>
<td>-0.11</td>
<td>NA</td>
<td>0.50</td>
<td>40.44</td>
<td>317.65</td>
</tr>
<tr>
<td>Differentiator</td>
<td>0.39</td>
<td>-0.28</td>
<td>0.58</td>
<td>26.04</td>
<td>139.32</td>
</tr>
<tr>
<td>REST</td>
<td>0.44</td>
<td>-0.33</td>
<td>0.37*</td>
<td>16.32</td>
<td>61.58</td>
</tr>
<tr>
<td>Training Data</td>
<td>2.31</td>
<td>-0.15</td>
<td>0.56</td>
<td>13.26</td>
<td>51.80</td>
</tr>
<tr>
<td>Preparation Agent</td>
<td>-0.34</td>
<td>NA</td>
<td>0.73</td>
<td>26.38</td>
<td>109.92</td>
</tr>
<tr>
<td>Training Agents</td>
<td>-0.27</td>
<td>-0.35</td>
<td>0.68</td>
<td>26.06</td>
<td>93.90</td>
</tr>
<tr>
<td>Monitoring Agent</td>
<td>-0.80</td>
<td>-0.35</td>
<td>0.55</td>
<td>77.12</td>
<td>282.87</td>
</tr>
<tr>
<td>Notifications</td>
<td>-0.68</td>
<td>-0.35</td>
<td>0.28</td>
<td>52.35</td>
<td>301.40</td>
</tr>
<tr>
<td>GDPR as a Service</td>
<td>-1.25</td>
<td>-0.35</td>
<td>0.53</td>
<td>41.92</td>
<td>249.44</td>
</tr>
</tbody>
</table>

* The majority of the REST component’s testing was in Cucumber tests, instead of unit tests, because it is a REST façade.
Summary of Approach

• New requirements met with new components
• Track technical debt
• Look for Expansion/Contraction cycle to know how far along you are
• Choose where to pay off tech debt based on metrics
Mistakes Made
Mistakes Made

- Started collecting application runtime metrics far too late
  - Revealed many problems as we added more
Mistakes Made

• Started collecting application runtime metrics far too late
  • Revealed many problems as we added more
• Depended on customers to find our problems for us
  • Massive support cost
  • Especially when people started using our product
Advantages
Advantages

• Ship fast
Advantages

- Ship fast
- Isolate tech debt
Advantages

• Ship fast
• Isolate tech debt
• Keep up with evolving product design
Advantages

• Ship fast
• Isolate tech debt
• Keep up with evolving product design
• Meet new requirements quickly
Advantages

- Ship fast
- Isolate tech debt
- Keep up with evolving product design
- Meet new requirements quickly
- Quick iterations per component
Advantages

- Ship fast
- Isolate tech debt
- Keep up with evolving product design
- Meet new requirements quickly
- Quick iterations per component
- Inexperienced team members can focus on small codebases
Advantages

• Ship fast
• Isolate tech debt
• Keep up with evolving product design
• Meet new requirements quickly
• Quick iterations per component
• Inexperienced team members can focus on small codebases

Disadvantages
Advantages

• Ship fast
• Isolate tech debt
• Keep up with evolving product design
• Meet new requirements quickly
• Quick iterations per component
• Inexperienced team members can focus on small codebases

Disadvantages

• Experienced architect is probably essential
Advantages

• Ship fast
• Isolate tech debt
• Keep up with evolving product design
• Meet new requirements quickly
• Quick iterations per component
• Inexperienced team members can focus on small codebases

Disadvantages

• Experienced architect is probably essential
• Pay the microservice “premium”
• Higher deployment, management overhead
Advantages

• Ship fast
• Isolate tech debt
• Keep up with evolving product design
• Meet new requirements quickly
• Quick iterations per component
• Inexperienced team members can focus on small codebases

Disadvantages

• Experienced architect is probably essential
• Pay the microservice “premium”
• Higher deployment, management overhead
• High support costs at first
• On-call was quite unpleasant for a while
Advantages

• Ship fast
• Isolate tech debt
• Keep up with evolving product design
• Meet new requirements quickly
• Quick iterations per component
• Inexperienced team members can focus on small codebases

Disadvantages

• Experienced architect is probably essential
• Pay the microservice “premium”
• Higher deployment, management overhead
• High support costs at first
• On-call was quite unpleasant for a while
• Have to actually pay off tech debt
• Took a lot of convincing of product owner
• Functional development mostly halted
• Took us 3-4 months to get to a healthy state
Conclusion
Conclusion

• Both general perspectives on starting with microservices have correct points
Conclusion

- Both general perspectives on starting with microservices have correct points
- By viewing getting bounded contexts wrong as strategic technical debt instead of as risk, starting with microservices can allow one to keeping up with rapidly evolving product design
Conclusion

• Both general perspectives on starting with microservices have correct points
• By viewing getting bounded contexts wrong as strategic technical debt instead of as risk, starting with microservices can allow one to keeping up with rapidly evolving product design
• The key elements of this approach manifest as an expansion/contraction cycle:
  • Address new requirements with new components
  • Isolate technical debt
  • Pay off technical debt only in mission-critical components
Conclusion

- Both general perspectives on starting with microservices have correct points.
- By viewing getting bounded contexts wrong as strategic technical debt instead of as risk, starting with microservices can allow one to keep up with rapidly evolving product design.
- The key elements of this approach manifest as an expansion/contraction cycle:
  - Address new requirements with new components.
  - Isolate technical debt.
  - Pay off technical debt only in mission-critical components.
- Several simple preemptive measures could have avoided costly mistakes:
  - E.g. instrument early.
Conclusion

• Both general perspectives on starting with microservices have correct points
• By viewing getting bounded contexts wrong as strategic technical debt instead of as risk, starting with microservices can allow one to keeping up with rapidly evolving product design
• The key elements of this approach manifest as an expansion/contraction cycle:
  • Address new requirements with new components
  • Isolate technical debt
  • Pay off technical debt only in mission-critical components
• Several simple preemptive measures could have avoided costly mistakes
  • e.g. instrument early
• Actually budget to pay off the tech debt accrued in order to ship faster
Conclusion

• Both general perspectives on starting with microservices have correct points
• By viewing getting bounded contexts wrong as strategic technical debt instead of as risk, starting with microservices can allow one to keeping up with rapidly evolving product design
• The key elements of this approach manifest as an expansion/contraction cycle:
  • Address new requirements with new components
  • Isolate technical debt
  • Pay off technical debt only in mission-critical components
• Several simple preemptive measures could have avoided costly mistakes
  • e.g. instrument early
• Actually budget to pay off the tech debt accrued in order to ship faster
• We think this strategy could be more generally applied against other types of uncertainty
Thank you.
Supplementary Materials
### Metrics (Comprehensive)

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of PRs (z-score)</th>
<th>Number of Commits (z-score)</th>
<th>Number of Comments (z-score)</th>
<th>Refactor Size (z-score)</th>
<th>Repository Size (z-score)</th>
<th>Percentage Code Refactored (z-score)</th>
<th>Log Count (z-score)</th>
<th>Unit Test Coverage</th>
<th>Time between Commits (Mean)</th>
<th>Time between Commits (STD DEV)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>1.09</td>
<td>0.09</td>
<td>0.83</td>
<td>0.13</td>
<td>0.06</td>
<td>-0.78</td>
<td>NA</td>
<td>0.53</td>
<td>7.70</td>
<td>39.03</td>
</tr>
<tr>
<td>Search + ML</td>
<td>0.89</td>
<td>1.25</td>
<td>0.81</td>
<td>1.88</td>
<td>1.26</td>
<td>-0.87</td>
<td>2.84</td>
<td>0.70</td>
<td>8.75</td>
<td>46.14</td>
</tr>
<tr>
<td>ML</td>
<td>-0.11</td>
<td>-1.31</td>
<td>0.16</td>
<td>-0.37</td>
<td>-0.45</td>
<td>-0.34</td>
<td>NA</td>
<td>0.50</td>
<td>40.44</td>
<td>317.65</td>
</tr>
<tr>
<td>Differentiator</td>
<td>0.39</td>
<td>0.06</td>
<td>-0.38</td>
<td>0.35</td>
<td>-0.35</td>
<td>-0.21</td>
<td>-0.28</td>
<td>0.58</td>
<td>26.04</td>
<td>139.32</td>
</tr>
<tr>
<td>REST</td>
<td>0.44</td>
<td>0.24</td>
<td>0.58</td>
<td>0.43</td>
<td>0.70</td>
<td>-0.92</td>
<td>-0.33</td>
<td>0.37*</td>
<td>16.32</td>
<td>61.58</td>
</tr>
<tr>
<td>Training Data</td>
<td>2.31</td>
<td>0.15</td>
<td>0.68</td>
<td>0.07</td>
<td>2.71</td>
<td>-1.09</td>
<td>-0.15</td>
<td>0.56</td>
<td>13.26</td>
<td>51.80</td>
</tr>
<tr>
<td>Preparation Agent</td>
<td>-0.34</td>
<td>-1.08</td>
<td>0.88</td>
<td>-0.47</td>
<td>-0.53</td>
<td>-0.04</td>
<td>NA</td>
<td>0.73</td>
<td>26.38</td>
<td>109.92</td>
</tr>
<tr>
<td>Training Agents</td>
<td>-0.27</td>
<td>0.59</td>
<td>0.80</td>
<td>0.06</td>
<td>-0.46</td>
<td>0.01</td>
<td>-0.35</td>
<td>0.68</td>
<td>26.06</td>
<td>93.90</td>
</tr>
<tr>
<td>Monitoring Agent</td>
<td>-0.80</td>
<td>-1.36</td>
<td>-1.34</td>
<td>-1.12</td>
<td>-0.65</td>
<td>0.23</td>
<td>-0.35</td>
<td>0.55</td>
<td>77.12</td>
<td>282.87</td>
</tr>
<tr>
<td>Notifications</td>
<td>-0.68</td>
<td>-0.15</td>
<td>0.16</td>
<td>-0.83</td>
<td>-0.53</td>
<td>-0.38</td>
<td>-0.35</td>
<td>0.28</td>
<td>52.35</td>
<td>301.40</td>
</tr>
<tr>
<td>GDPR</td>
<td>-1.25</td>
<td>1.56</td>
<td>-0.64</td>
<td>1.84</td>
<td>-0.43</td>
<td>0.92</td>
<td>-0.35</td>
<td>0.53</td>
<td>41.92</td>
<td>249.44</td>
</tr>
</tbody>
</table>

* The majority of the REST component’s testing was in Cucumber tests, instead of unit tests, because it is a REST façade.
## Metrics (Highlighted)

<table>
<thead>
<tr>
<th>Component</th>
<th>Number of PRs (z-score)</th>
<th>Log Output Percentage (z-score)</th>
<th>Unit Test Coverage</th>
<th>Hours between Commits (Mean)</th>
<th>Hours between Commits (STD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>1.09</td>
<td>NA</td>
<td>0.53</td>
<td>7.70</td>
<td>39.03</td>
</tr>
<tr>
<td>Search+ML</td>
<td>0.89</td>
<td>2.84</td>
<td>0.70</td>
<td>8.75</td>
<td>46.14</td>
</tr>
<tr>
<td>ML</td>
<td>-0.11</td>
<td>NA</td>
<td>0.50</td>
<td>40.44</td>
<td>317.65</td>
</tr>
<tr>
<td>Differentiator</td>
<td>0.39</td>
<td>-0.28</td>
<td>0.58</td>
<td>26.04</td>
<td>139.32</td>
</tr>
<tr>
<td>REST</td>
<td>0.44</td>
<td>-0.33</td>
<td>0.37*</td>
<td>16.32</td>
<td>61.58</td>
</tr>
<tr>
<td>Training Data</td>
<td>2.31</td>
<td>-0.15</td>
<td>0.56</td>
<td>13.26</td>
<td>51.80</td>
</tr>
<tr>
<td>Preparation Agent</td>
<td>-0.34</td>
<td>NA</td>
<td>0.73</td>
<td>26.38</td>
<td>109.92</td>
</tr>
<tr>
<td>Training Agents</td>
<td>-0.27</td>
<td>-0.35</td>
<td>0.68</td>
<td>26.06</td>
<td>93.90</td>
</tr>
<tr>
<td>Monitoring Agent</td>
<td>-0.80</td>
<td>-0.35</td>
<td>0.55</td>
<td>77.12</td>
<td>282.87</td>
</tr>
<tr>
<td>Notifications</td>
<td>-0.68</td>
<td>-0.35</td>
<td>0.28</td>
<td>52.35</td>
<td>301.40</td>
</tr>
<tr>
<td>GDPR as a Service</td>
<td>-1.25</td>
<td>-0.35</td>
<td>0.53</td>
<td>41.92</td>
<td>249.44</td>
</tr>
</tbody>
</table>

* The majority of the REST component’s testing was in Cucumber tests, instead of unit tests, because it is a REST façade.
Unit Test Coverage

- Search
- Search + ML
- ML
- Differentiator
- REST
- Training Data
- Training Agent Preparation Agent
- Training Agents
- Monitoring Agent
- Notifications
- GDPR