MICROSERVICES
BOUNDDED CONTEXTS
AND EVERYTHING IN BETWEEN
"95% of the words are spent extolling the benefits of “modularity” and that little, if anything, is said about how to achieve it”

Glenford J. Myers
“MICROSERVICES FTW!!!!!!!1”

Me @ Internovus
“MICROSERVICES FTW!!!!!!!!1”
"95% of the words are spent extolling the benefits of "modularity" and that little, if anything, is said about how to achieve it"

Glenford J. Myers
Decomposition Strategies

Microservices

Design Heuristics
INTERNOVUS

BUSINESS DOMAIN
DECOMPOSITION STRATEGIES
Bounded Contexts
Ubiquitous Language

Software Developer

Business Domain Expert

#OREillySACon
Your Product → Marketing Strategy → Creatives → Campaigns → Sales Agents

Profits → Optimization
LEAD?

Campaign Manager

Sales Agent
Conflicting models

Different contexts

Dividing the language/model

Bounded Context
Bounded Contexts

Marketing
- Campaign
- Placement
- Agency
- Funnel
- Landing Page
- Insertion Order
- Lead

Sales
- Contact
- Conversion
- Desk
- Campaign
- Message
- Proposition
- CDR
- Lead
Decomposition Strategy #1: Bounded Contexts

Marketing Service

Sales Service
Business (Sub)Domains
Business Domain: Customer Acquisition

**Bounded Context: Marketing**
- Subdomain: Creative Catalog
- Subdomain: Campaigns Mngmt
- Subdomain: Ads Optimization
- Subdomain: Contracts Mngmt

**Bounded Context: Sales**
- Subdomain: CRM
- Subdomain: Sales Optimization
- Subdomain: Commissions
- Subdomain: Desks Mngmt
Decomposition Strategy #2: Subomains

- Campaign Management Service
- Content Catalog Service
- Desks Management Service
- CRM Service
- Users Service
- Billing Service
Business Entities and Processes
Decomposition Strategy #3: Entities

- Website
- Asset
- Target Market
- Broker
- Distribution
- Customer
Decomposition Strategies

Bounded Contexts

Subdomains

Business Entities and Processes
SHOW ME THE MICROSERVICES!
Bounded Contexts == Microservices ?
LEAD?

Campaign Manager

Sales Agent
Bounded Context

Consistency boundary of the language / model

Define the biggest valid monoliths

Required decomposition boundary
Bounded Contexts == Microservices? No
MICROSERVICES
Service

“A service is a unit of functionality exposed to the world” - Juval Lowy

“A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface” - OASIS
Service Interface

“Any mechanism for getting data in or out of the service”

- Randy Shoup
Service Interface

Synchronous:
- Request / Response
- Bulk ETL

Asynchronous:
- Producing events
- Consuming events
Service Interface

“Any mechanism for getting data in or out of the service”

- Randy Shoup
Service Interface
Services ➔ Microservice
A microservice is a service with a micro interface
Microservice = Microinterface

Reducing coupling between services

Limits reasons for change

Easier to understand

Better fault isolation

More autonomy between services
Microservice & Databases

Microservices should own its database
No external access
Access through Microservice’s API only
Database = infinite interface!
1 Method = Perfect Microservice !!??
# Backlog

- + Add Item
- + Update Item's Details
- + Archive Item
- + Set Item's Priority
- + List Backlog Items
- + Search
- + Show Item
- + Assign Item to Sprint
The **threshold** upon which a system can be **decomposed** into **microservices** is defined by the **use cases** of the **system** that the **microservices** are a part of.
Cost of Change

Monolith  Microservices  Distributed Monolith
Backlog

- Add Item
- Update Item’s Details
- Archive Item
- Set Item’s Priority
- List Backlog Items
- Search
- Show Item
- Assign Item to Sprint
"Global complexity … the complexity of the overall structure of a program or system. I.e., the degree of association or interdependence among the major pieces of a program”

Glenford J. Myers
"Global complexity … the complexity of the overall structure of a program, the degree of association or interdependence among the major pieces of a program."

Glenford J. Myers

Composite / Structured Design, 1978
When you design a system..., then if the features can be broken into ... loosely bound groups of relatively closely bound features, then that division is a good thing to be made a part of the design. This is just good engineering.

Tim Berners-Lee

Principles of design, 1998
Services + Good Engineering = Microservices
It is not only necessary to make sure your own system is designed to be made of modular parts. It is also necessary to realize that your own system ... should always be designed to be a part of another larger system.

Tim Berners-Lee

Principles of design, 1998
A major part of this paper will be concerned with the question of how good modularity can be achieved, that is, how modules can be chosen so as to minimize the connections between them.

Barbara Liskov

A design methodology for reliable software systems, 1972
A **microservice** is a service with a **micro** interface.

The **threshold** upon which a system can be **decomposed** into **microservices** is defined by the **use cases** of the **system** that the **microservices** are a part of.
A **microservice** is a service with a **micro interface**.

The *threshold* upon which the system can be decomposed is defined by the use cases of the system that the microservices are a part of.

...**how to evaluate**?
DESIGN
HEURISTICS
Heuristic #1: Decompose to Bounded Contexts

Do not implement conflicting models in the same service. **Always** decompose to **Bounded Contexts**.
Heuristic #2: Don’t

First Law of Distributed Object Design: “Don’t distribute your objects”

Martin Fowler
Heuristic #2: Don’t Addi)onal Complexity

Benefits

Additional Complexity
MONOLITH

SYSTEM DESIGN THAT UNDERMINES DELIVERY OF FUNCTIONAL OR NON-FUNCTIONAL REQUIREMENTS
Heuristic #2: Don’t

Benefits

Additional Complexity
Heuristic #2: Don’t

Benefits  

Additional Complexity
Core

Supporting

Generic
Generic Subdomains

No competitive advantage

Everybody is implementing in the same way

Complex business logic
Marketing

- Creative
- Catalog
- Billing
- Campaign
- Management
- Optimization
- Identity & Access

Sales

- CRM
- Telephony
- Optimization
- Commissions
- Desks
- Identity & Access
- Generic
Heuristic #3: Buy/Adopt Generic Subdomains

Your System

3rd party product
Heuristic #3: Buy/Adopt Generic Subdomains

Your System

Anti-Corruption Layer

3rd party product
Core Subdomains

Inventing something new

Optimizing existing practices

Competitive advantage

Complex business logic

Change often
Core Subdomains

Inventing something new

Optimizing existing paractices

Competitive advantage

Complex business logic

Change often
Heuristic #4: Core Subdomains - Don’t Rush

Adhere to subdomain’s boundaries. Decompose further only when you acquire domain knowledge.
Supporting Subdomains

No competitive advantage

Support the Core Subdomains

Can’t be bought / adopted

Simple business logic
Supporting Subdomains

No competitive advantage

Support the Core Subdomains

Can’t be bought / adopted

Simple business logic
Heuristic #5: Supporting Subdomains - Safe

Safe to decompose beyond the subdomain’s boundaries.
Generic

Core

Supporting
Heuristic #6: Evaluate Consistency Requirements
Heuristic #6: Evaluate Consistency Requirements

Concurrency control? - Same service

Service A

Method A  Method B
Heuristic #6: Evaluate Consistency Requirements

Read last write? - Two services, synchronous communication

![Diagram showing a sync call between Service A and Service B]
Heuristic #6: Evaluate Consistency Requirements

Eventual consistency? - Two services, asynchronous communication

Service A
  Method A

Service B
  Method B

Asynchronous Communication
Heuristic #6: Evaluate Consistency Requirements

Concurrency control? - Same service

Read last write? - Two services, sync communication

Eventual consistency? - Two services, async communication
Asynchronous Communication

Service A

Method A

Service B

Method B
Heuristic #7: Public / Private Events

Service

- Event Type 1
- Event Type 2
- Event Type 3
- Event Type 4
- Event Type 5
- Event Type 6
- ... 
- Event Type 1000

#OREillySACon
Heuristic #7: Public / Private Events

Service

Event Type 1
Event Type 2
Event Type 3
Event Type 4
Event Type 5
Event Type 6
...
Event Type 1000

Private event types (Implementation details)
Heuristic #7: Public / Private Events

Service

- Event Type 1
- Event Type 2
- Event Type 3
- Event Type 4
- Event Type 5
- Event Type 6
- ...  
- Event Type 1000

{ Public event types (Public interface) 

{ Private event types (Implementation details) 

#OREillySACon
Heuristic #7: Public / Private Events

- EmailChanged
- PhoneNumberChanged
- AddressChanged

Service
Heuristic #7: Public / Private Events

Service

- EmailChanged
- PhoneNumberChanged
- AddressChanged
- ContactDetailsChanged
Heuristic #7: Public / Private Events

- EmailChanged
- PhoneNumberChanged
- AddressChanged
- ContactDetailsChanged

- Private event types (Implementation details)
- Public event type (Public interface)
Heuristic #7: Public / Private Events

- Domain Events
  - Private
- Events?
- State Change Events
  - Public
Heuristic #8: Make Events Explicit

Eliminate ambiguity
Heuristic #8: Make Events Explicit

Service

AgentAssignedToLead
Heuristic #8: Make Events Explicit

Service

- AgentAssignedToLead
- AgentAssignedToLead
- AgentAssignedToLead
Heuristic #8: Make Events Explicit

Service

AgentAssignedToLead
AgentUnassignedToLead
AgentAssignedToLead
Heuristic #9: Evaluate Reasons for Change

Service A

Service B
Heuristic #9: Evaluate Reasons for Change

Same reasons for change?

Same rate of change?

Can be decoupled?

Should be the same service?
On the Criteria To Be Used in Decomposing Systems into Modules

D.L. Parnas
Carnegie-Mellon University

Introduction

A lucid statement of the philosophy of modular programming can be found in a 1970 textbook on the design of system programs by Gouthier and Pont [1, ¶10.23], which we quote below:

A well-defined segmentation of the project effort ensures system modularity. Each task forms a separate, distinct program module. At implementation time each module and its inputs and outputs are well-defined, there is no confusion in the intended interface with other system modules. At checkout time the integrity of the module is tested independently; there are few scheduling problems in synchronizing the completion of several tasks before checkout can begin. Finally, the system is maintained in modular fashion; system errors and deficiencies can be traced to specific system modules, thus limiting the scope of detailed error searching.

Usually nothing is said about the criteria to be used in dividing the system into modules. This paper will discuss that issue and, by means of examples, suggest...
Heuristic #10: Evaluate Services’ “Doors”
Heuristic #10: Evaluate Services’ “Doors”
WRAP UP
1. A service is a unit of functionality exposed to the world through its public interface.
2. A microservice is a service with micro interface
3. The “micro-interface” is relative to the use cases of the system that the service is a part of
Heuristic #1: Always decompose to Bounded Contexts

Heuristic #2: Don’t go further, unless you have to

Heuristic #3: Buy/adopt generic subdomains

Heuristic #4: Core subdomains - don’t rush

Heuristic #5: Supporting subdomains can be decomposed early
Heuristic #6: Evaluate Consistency Requirements
Heuristic #7: Private / Public Events
Heuristic #8: Explicitly Define Events
Heuristic #9: Evaluate Reasons for Change
Heuristic #10: Evaluate Services’ “Doors”
Bibliography

1. A Design Methodology for Reliable Software Systems, Barbara Liskov
2. Designing Autonomous Teams and Services, Nick Tune & Scott Millet
3. Composite/Structured Design, Glenford Myers
4. Domain-Driven Design: Tackling Complexity in the Heart of Software, Eric Evans
5. Emerging Boundaries, Mathias Veraes
6. Implementing Domain-Driven Design, Vaughn Vernon
7. Enterprise Integration Patterns, Gregory Hohpe, at al
8. Long Sad Story of Microservices, Greg Young
9. Managing Data in Microservices, Randy Shoup
10. Microservices, At Last Some Boundaries, Eric Evans
11. On the Criteria to be Used in Decomposing Systems Into Modules, David Parnas
12. Patterns of Enterprise Application Architecture, Martin Fowler, at al
13. Principles of Design, Tim Berners-Lee
14. Programming WCF Services, Juval Lowy
15. What Every Programmer Should Know About Object Oriented Design, Melir Page-Jones
THANK YOU!

@vladikk
vladikk.com


http://careers.doit-intl.com
Rate today’s session

Session page on oreillysacon.com/ny

O’Reilly Events App