From Monolith to Microservices
Introduction

• The Case
• Domain Driven Design
• Microservices

Discussion

• Data vs Domain Driven
• Organization and Team
• Breaking the Monolith
The Case
DBR – Drilling Reporting System

Planning and reporting of drilling operations

Began as a simple activity log

Has evolved into much more over 20 years

Client server application
- 3.5 MLOC
  - 1.5 MLOC PLSQL
  - 1.0 MLOC C#/ APS.NET/Silverlight
  - 1.5 MLOC PowerBuilder

Began as a PowerBuilder / Oracle application
- Extended with Web later
Architecture & Technology

DBR DB

Tightly coupled
+10,000 LOC procedures
Fat Windows Client / Citrix
Technological fragmented
Scripted business logic

1,5MILL LOC PLSQL

Operations
Experiences
Risk
Sections
Cost Estimates
Experiences
KPI's
Plans
The Team

Small (3-5) over very long time
- +15 years
- Now two teams 6+4, two locations

Technology segregated
- Database
- Power Builder
- Web (Microsoft Stack)

Vulnerable
- Dependent on individuals
- Number of years to retirement

Geographically segregated
- Stavanger
- Bergen
Painpoints

Long lead times for new functionality
Convoluted database model
Deployment problems (windows client on Citrix)
System level testing
All in one build bundle
Obsolete technology
Short time to retirement
Way forward

1. Make implicit concepts explicit.
2. Create functional verticals in a layered architecture.
3. How to split the database?
Domain Driven Design

Domain-Driven Design: Tackling the complexity in the heart of software
Eric Evans, 2003

http://www.domaindrivendesign.org
Domain Logic Patterns

Three main patterns for organizing the domain logic:

- Transaction Script
- Table Module
- Domain Model
Domain Driven Design – distilled

• Ubiquitous (domain based) language
  - A language that is built around the concepts of the business and that permeates every activity in the project.
  - The language used to talk about the domain model in the project
• Patterns for building a domain model
• Strategic design principles and techniques
Ubiquitous language – A Domain based language

- Technical Language
- Ubiquitous Language
- Domain Language
Building blocks: Patterns

Domain Driven Design

- Smart Database
- Smart UI
- Layered Architecture
- Services
- Entities
- Factories
- Value Objects
- Aggregates
- Repositories
Strategic design: Context maps

In large systems (or set of systems), we need a map to give us a picture of the models that are inside.
Strategic design: Integrity across systems

- Bounded context
  - The meaning of a domain concept is bound by the context it is used

- Context map
  - A map that describe the contexts and their relationships

- Relation types:
  - Shared kernel
  - Customer/supplier teams
  - Conformist
  - Anti-corruption layer
  - Separate ways
  - Open host service
  - Published language
## Strategic design: Types of relations

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared kernel</td>
<td>• Overlapping models shared among teams</td>
</tr>
<tr>
<td>Customer/supplier development teams</td>
<td>• One bounded context is maintained by one team but used by another</td>
</tr>
<tr>
<td>Conformist</td>
<td>• As C/S development teams, but the customer team strictly adheres to the supplier model, without the option to change it.</td>
</tr>
<tr>
<td>Anticorruption layer</td>
<td>• Isolation layer between models that take up the differences</td>
</tr>
<tr>
<td>Separate ways</td>
<td>• Avoid integration, let the models develop on their own</td>
</tr>
<tr>
<td>Open host service</td>
<td>• One system that has an open connection point that can be used by (many) other systems</td>
</tr>
<tr>
<td>Published language</td>
<td>• Let the integration be based on a common, well-defined language</td>
</tr>
</tbody>
</table>
Aggregates

Order aggregate

OrderRepository

+add()
+findAll()
+findMatching()

«aggregate-root»
Order

+addProduct()
+calculateCost()
+getOrderLines()
+deleteOrderLine()
+create()

OrderLine

-Price

Product aggregate

ProductRepository

+add()
+findAll()
+findMatching()

«aggregate-root»
Product

-Price
Microservices

...is a way of designing software as suites of independently deployable services
Independent

governance

web services

Team

REST

Mobility

Skills

Linux

App Engine

node.js

Kanban

Service Management

Docker

Application Development

JSON

Odata

XML

Atom Pub

Ruby

Scala

Python

Architecture

Agile

Cloud

Node.js

Kanban

Deployment

Build

Data Management

Technology Choice
Smart Endpoints
Dumb Pipes
Service Interfaces

ODATA
Open Data Protocol

{json:api}

RSS
XML
ATOM
Organized around Business capabilities
Conway’s Law

Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations

Melvin E. Conway
Borrowed from James Lewis and Martin Fowler’s article:
http://martinfowler.com/articles/microservices.html
Inverse Conway maneuver

Cross-functional teams...

... organised around capabilities
Because Conway's Law

Borrowed from James Lewis and Martin Fowler’s article:
http://martinfowler.com/articles/microservices.html
Important success criteria

- Rapid provisioning
- Basic monitoring
- Rapid Application Development
- DevOps Culture
Data driven vs Domain driven
Discussion

Domain Driven Design is advocated as the best way

Still we see that the data driven approach dominate

Object oriented languages used as script languages

1000 or even 10,000 LOC methods are still written

Why?
Data Driven Development

Has its origin in data processing

1890 US Census
Herman Hollerith
Punch cards for data storage

Entry, Validation, Sorting, Summarization, Aggregation, ...
Electro-Mechanical machines until the 1950ties...
COBOL programming language since 1959 ...
Object Oriented Development

Has its origin in simulation of dynamic systems

Interception of ICBM’s

Simula 67 language

Ole Johan Dahl / Kristen Nygaard

Encapsulation of state and behaviour in “classes”

Simplifies the modelling of real-world behaviour

Smalltalk, C++, Java, C#, Scala, ....
Thoughts on DBR and its likes

Began as a data processing systems

Record and report performed operations
  • Materials used
  • Difficulties encountered
  • Failures

With time, more and more dynamic domain’s was added

Planning (re-planning)
  • Automated planning
  • Optimisation
  • Monitoring

Scheduling
  • Cost function
  • Automatic re-scheduling
  • Optimisation

Dynamic domains are addressed with a data driven approach
Micro-services to the rescue?

Planning & Scheduling
- Automated planning
- Multi-agent

Analysis
- R for statistics

Daily reports
- Data driven

Each service can be implemented with the most suitable technology
You need skills

Knowledg

- Novice
- Advanced Novice
- Competent
- Proficient
- Expert

Creativity
Skills and productivity

Number of persons

Novice 10x Competent 10x Expert
Organization
Working in large.

- Jeff Bezos, CEO, Amazon
Our Team

Small (3-5) over very long time
• +15 years
• Now two teams 6+4, two locations

Technologically segregated
• Database
• Power Builder
• Web (Microsoft Stack)
Why have we not succeeded?
Leadership

Good software leaders are rare
• How to nurture talents?
• How to develop the needed skills?

Leading from the front or back?
• How to build trust?

How to ensure individuals pulls as a team?

It doesn’t make sense to hire smart people and then tell them what to do; we hire smart people so they can tell us what to do.

Steve Jobs
Conway’s Law

Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations

Melvin E. Conway
Our Team

Not cross functional
• Database
• Power Builder
• Web (Microsoft Stack)

Not co-located
• Stavanger
• Bergen

Vulnerable
• Dependent on individuals
• Number of years to retirement
Our company

Large enterprise organization
- Divisional structure
- Multinational
- Central IT governance

Lacking
- DevOps culture
- Infrastructure automation
Borrowed from James Lewis and Martin Fowler’s article:
http://martinfowler.com/articles/microservices.html
Actors = 4
Actors = 1
Breaking the Monolith
How do you eat an elephant?

one bite at a time
Goals

1. Make it easier to implement new features
2. Make stored data more easily available
3. Simplify build and deployment
4. Modernize technology stack

In short: Optimize delivery of new features and availability of data
Bounded contexts

DBR

Monolith

DBR DB (1,5mloc)
Risk Profile

Project Planning

DBR - Modularized

- Risk
- Daily Reports
- Experiences
- Reference Data
- Analysis
- Project Planner
- Benchmarks KPI’s
- Schedules
- Cost Estimates
- Operations History

Functionality integrated at each module level as services
- Internal bus for DBR functionality
- Services for external data
How do we get there?
Making changes
Extracting a bounded context
Database tactics

1. Duplicate databases, replicate data
2. Duplicate schemas
3. Views
Database refactoring
Master data
Summary

- Data-driven monolithic apps
- Change and adapt your organization
- Bounded contexts as Microservices
- Build domain modelling skills
- Leadership is a critical success factor
Thank you