Architectural Skills

CSSE6400

Richard Thomas

May 30, 2022

Quote

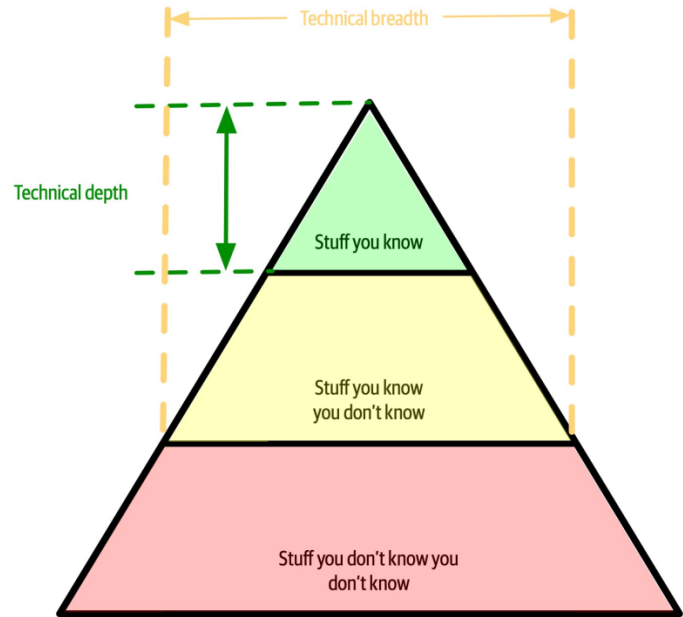Architecture is the stuff you can't Google.
    – Mark Richards [1]

Quote

There are no right or wrong answers in architecture—only trade-offs.
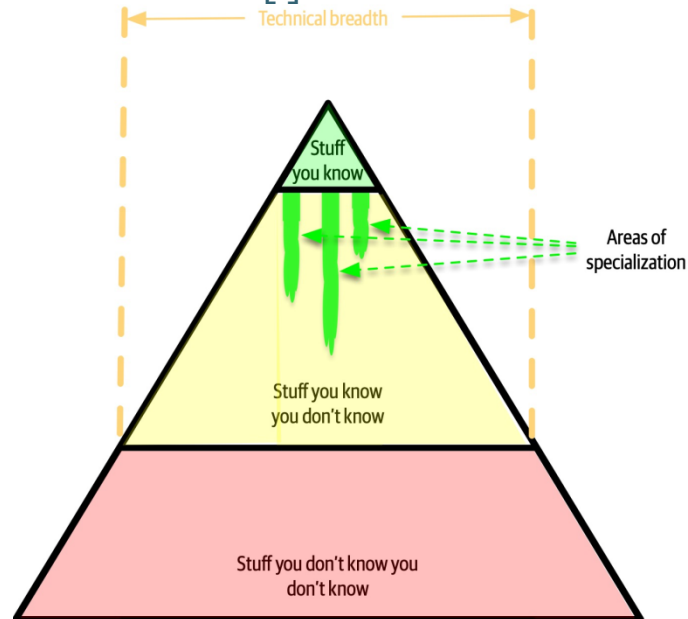
　　— Neal Ford [1]

Architectural Design

Architects use knowledge and experience to analyse trade-offs to design architectures appropriate to the system context.

Developers – Technical Depth [1]

Architects – Technical Breadth [1]

- Architects need greater technical breadth than depth.
- Breadth allows better consideration of trade-offs.
- Avoid trying to become an expert across many areas – you'll fail.
- Don't stop learning – increase your breadth – don't let your knowledge become stale.

What are the benefits of a monolith architecture?

What are the benefits of a monolith architecture?

- Simple deployment

What are the benefits of a monolith architecture?

- Simple deployment
- Simple communication between modules

What are the benefits of a monolith architecture?

- Simple deployment
- Simple communication between modules
- Simple system testing & debugging

Why do monoliths have a bad name?

Why do monoliths have a bad name?

- Many legacy system nightmares were monoliths

Question

Why do monoliths have a bad name?

Answer

- Many legacy system nightmares were monoliths
- Easy to defeat modularity

Why do monoliths have a bad name?

- Many legacy system nightmares were monoliths
- Easy to defeat modularity
- Cannot scale components of system

Why do monoliths have a bad name?

- Many legacy system nightmares were monoliths
- Easy to defeat modularity
- Cannot scale components of system
- Monolith databases scale poorly

What can be done if a monolith architecture is no longer suitable?

What can be done if a monolith architecture is no longer suitable?

- Greenfields replacement

What can be done if a monolith architecture is no longer suitable?

- Greenfields replacement
- Migrate to another architecture

- Replacement: Can choose any suitable architecture. Risky, as you're developing a new system and maintaining existing.
- Migration: Adaptive maintenance, changing architecture slowly. Some limitation on choice of architecture, but most sophisticated architecture can be used.

How do I migrate a monolith to a new architecture?

How do I migrate a monolith to a new architecture?

Implies a service-based or microservices architecture.
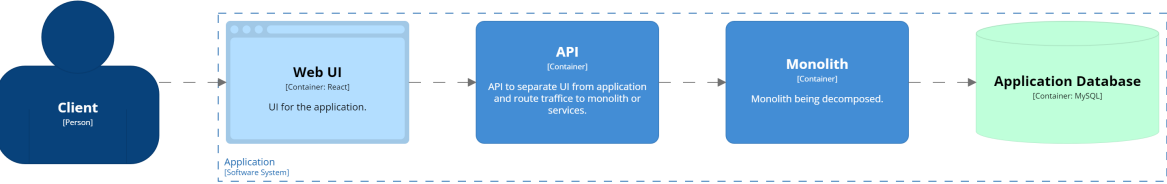
Decompose the monolith into services.

Strangler Fig Pattern

- Develop API for application's UI
- Proxy intercepts API calls
  - Proxy directs calls to application or new services
- Implement a service
  - Redirect calls to service
- Progressively replace monolith
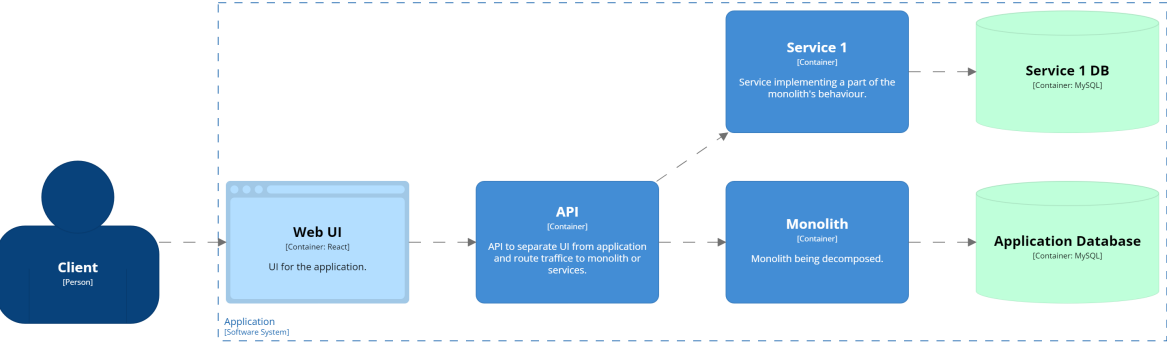- Shadow & Blue-Green Deployment

- May already have an API if the UI is a web or mobile app.
- Initially deploy proxy and new interface into production, with only existing monolith. Test it works as expected.
- Use shadow deployment to test service with application, before making available to end users.

Monolith Deployment

Client
[Person]

Web UI
[Container: React]

UI for the application.

API
[Container]

API to separate UI from application and route traffice to monolith or services.

Monolith
[Container]

Monolith being decomposed.

Application Database
[Container: MySQL]

Application
[Software System]

Monolith Decompose: Step 1

Monolith Decompose: Step 2

Client
[Person]

Web UI
[Container: React]

UI for the application.

API
[Container]

API to separate UI from application and route traffic to monolith or services.

Service 1
[Container]

Service implementing a part of the monolith's behaviour.

Service 1 DB
[Container: MySQL]

Monolith
[Container]

Monolith being decomposed.

Application Database
[Container: MySQL]

Service 2
[Container]

Service implementing a part of the monolith's behaviour.

Service 2 DB
[Container: MySQL]

Application
[Software System]

- Identify bounded-contexts
- Simple first service
  - e.g. Authentication
- Minimise dependency from services to monolith
  - Monolith may use services

- Use first service (or first few) to validate approach and deployment infrastructure.
- Minimise changes required in monolith.

Decomposition Process

- Reduce coupling between bounded-contexts
  - e.g. Customer account management
    - Profile, Wish List, Payment Preferences – separate services
- Decouple vertically
  - Service delivers entire bounded-context
    - Data is decoupled from monolith

- Account management may be tightly coupled in monolith. Separate each aspect (context), one at a time.
- Do not focus only on UI or internal components, service needs to implement all parts of the business process.
- Data management needs to be decentralised.

Decomposition Process

- **Focus on pain points**
  - Bottlenecks
  - Frequently changing behaviour

- **Rewrite, don't reuse**
  - Redesign for new infrastructure
  - Reuse complex logic
    - e.g. Discounts based on customer loyalty and behaviour, bundle offers, …

- Extract services that deliver highest value.
- What contexts may need to scale more than others?
- What contexts change more frequently and benefit from separate deployment?
- Services deliver capabilities provided by monolith.
- Most often it is better to rewrite the capability to take advantage of new infrastructure.
- Only reuse code that has complex logic that will be difficult to duplicate and test fully.

- **Refactor monolith**
  - Use service to deliver application functionality
    - Monolith may need to invoke service
  - Remove service logic from monolith

- Atomic replacement of monolith behaviour by service's behaviour.
- Don't deploy production code with service behaviour left in monolith. Leads to a maintenance nightmare determining where behaviour is used, or it may be used in both the monolith and service.

Stepwise Decomposition

Replace application functionality one service at a time.

Definition 1. Macroservice

Separate service, but may span more than one domain or share a database with the monolith or other services.

- Similar scalability and deployment issues to a monolith, but grouped by clusters of macroservices if they share a database.
- Interim step to build microservices.

> **Definition 2. Conway's Law**
>
> Organisations design systems whose structure is inevitably a copy of the organisation's communication structure [2] [3].

- First citation is original article.
- Second citation is one of several about MIT and Harvard research into the phenomenon, calling it the "mirroring hypothesis".
- Elaborate on this point and Coplien's research into organisational sociology.

- Business Process Management
- Microservices to reflect organisation structure
- Teams formed around services

- BPM: Redesign organisation structure to reflect system you want.
- Microservices: Design system to reflect your organisation.
- Elaborate on benefits of both approaches.
- Comment on benefits of small focussed teams.

Team insularity – more loyal to team than organisation.

- Amazon example from week 11, negotiation difficulties with other teams.
- Need to ensure inter-team cooperation.
- Possibly move people between teams.
- Cloud platforms, and microservices, also support this and with larger teams.
- Intra-team communication becomes more difficult with large teams.

Conway's Law Issues

- Cross-cutting concerns
  - e.g. Security
- Organisation structure should align with market structure
- Physical location of teams

- Cross-cutting concerns span services, and consequently teams.
- Can't have a "security" service. It has to be part of every service.
- Teams solely based around Conway's law and services may not deliver some cross-cutting concerns.
- Cooperation, documentation and audits may be necessary.
- Market structure may complement team structure to place teams closer to their end users.
- Global development and outsourcing mean different teams are likely to be in different locations.
- Requires additional overhead and documentation for cooperation between teams.

Don't follow fads, seek evidence for good practice.

Elaborate on finding reliable sources of information and confirming facts yourself.

Software Engineering's Greatest Hits

**what we actually know about software development
and why we believe it's true**

Greg Wilson

https://youtu.be/HrVtA-ue-x0

References

[1] Mark Richards and Neal Ford.
Fundamentals of Software Architecture: An Engineering Approach.
O'Reilly Media, Inc., January 2020.

[2] Melvin E. Conway.
How do committees invent?
Datamation, April 1968.

[3] Alan MacCormack, Carliss Baldwin, and John Rusnak.
Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis.
Research Policy, 41(8):1309–1324, 2012.