The Future of Software Engineering
1. Scale Out $\Rightarrow$ Scale Up
Business Software:

- Monolithic & Slow to change
- ERP Database ⇔ Integration
- On a Single Server

Why One Server?

1996

CAP Theorem:

- Partitioned databases must choose between:
  - Availability
  - Consistency
How do you store 1.3 billion web pages on one server?

[c. 2001]

Search 1,326,920,000 web pages

Google Search

Software is Eating the World
Option 1: Scale UP

Get a bigger computer.

If one ox could not do the job they did not try to grow a bigger ox, but used two oxen. When we need greater computer power, the answer is not to get a bigger computer, but...to build systems of computers and operate them in parallel.

(Grace Hopper)

Option 2: Scale Out

Get more computers.

Software is Eating the World
Scale Out: Files

Google’s Problem:
- Search the entire Internet
- Instantly

Response:
- Horizontal Scaling
  - Break files into small pieces
  - Make three copies of each piece
  - Spread the pieces across many servers
  - Manage the pieces with software
Scale Out: Architecture

Amazon ~ 2001
- Handle a gazillion transactions
- All at once

What Amazon Did:
- Break transactions into services
  - Replicate bottleneck services
- Each service owned by a semi-autonomous “two pizza” team

Option 1: Scale Up
Option 2: Scale Out
2. Infrastructure as Code
Chris Pinkham (VP Infrastructure)
- Proposes self-service deployment for development teams
- Maybe sell the capability?

Autonomous Services ⇒ Autonomous Service Teams ≡ Independent Deployment ⇒ Much Stress in Operations

Time Passes....
- Pinkham moves to South Africa
- Asked to pursue project there
- Assembles and leads a team
- Develops EC2 in 2 years
- EC2 Launches in 2006
- 10 Years Later:

⇒ $10 billion/yr.
There is a Cloud in Your Future

**Economics**

The Cloud is cheaper, more stable, more secure, and more expandable than most on-premises data centers.

**Technology**

Applications designed for traditional data architectures do not work well in the cloud without a lot of recoding.*

*Arthur Cole - IBM
Infrastructure as Code

Containers
As Process

Serverless
Architectures

Software
Defined
Networks

docker

Software is Eating the World
3. The New Technology Stack
The Evolution of IT

1996

Thick, client-server app on thick client

Well-defined stack:
- O/S
- Runtime
- Middleware

Monolithic Physical Infrastructure

2016

Thin app on mobile, tablet

Assembled by developers using best available services

Running on any available set of physical resources (public/private/virtualized)

www.docker.io
Enterprise Architecture

The Dependency Problem

Scale Up

Business processes and activities use...

Data that must be collected, organized, safeguarded, and distributed using...

Applications such as custom or off-the-shelf software tools that run on...

Technology such as computer system and telephone networks.

Scale Out

Federated Architecture

Software is Eating the World
## Software Engineering with API’s

### API’s as Architecture
- Enable Service Architectures
- Lower Integration Friction
- Local Persistence

### API’s as Product
- Owned by a Responsible Team
- Focus on API Consumers
- Evolving Capabilities

### API’s Replace SQL

### Enable the Internet of Things

---

*Software is Eating the World*
Software Engineering for Big Data Systems

Distribution, Data, Deployment: Software Architecture Convergence in Big Data Systems
Ian Gorton, John Klein, Software Engineering Institute Carnegie Mellon University, May 2014

“Big data applications are pushing the limits of software engineering .... It is essential that the body of software architecture knowledge evolves to capture this advanced design knowledge for big data systems.”

Designing for scale:
- Big Data systems are inherently distributed; their architectures must explicitly handle:
  - partial failures
  - concurrency
  - consistency
  - replication
  - communications latencies

Resilient Architectures:
- Replicate data to ensure availability in the case of failure.
- Design components to be stateless
- replicated
- tolerant of failures of dependent services
Resilience Engineering

Three Kinds of Systems

Fragile  Robust  Anti-Fragile

Nassim Nicholas Taleb:
Antifragile – Things that gain from Disorder

Acm Queue, September 2012:
“Resilience Engineering: Learning to Embrace Failure”
Software Engineering in the Cloud

- Monolithic Architecture
- Database as Integrator
- Stateful Protocols
- Consecutive Execution
- Synchronous Communication
- Procedural Programming
- Defect Free / Fragile

The Enterprise Legacy

- Federated Architecture
- API’s as Connectors
- Stateless Protocols
- Concurrent Execution
- Asynchronous Communication
- Event Driven Programming
- Fault Tolerant / Antifragile

The Internet of Things

Software is Eating the World
The Three Ways

Flow
Feedback
Experimentation & Learning
One Thing We Know for Sure

For Complex Systems
This does not work

Smash!
One Thing We Know for Sure

For Complex Systems

This works

Continuous Flow
4. Build a Deployment Pipeline

Faster – Safer – Better

- Acceptance test driven development process
- Cross-functional teams include Product, QA and Ops
- Automated build, testing, db migration, and deployment
- Incremental development **on the trunk** with continuous integration
- Software always production ready, or everything stops until it is
- Deploy constantly, release by switch

*Continuous Flow*

Slide content credit: Jez Humble
5. End-to-end Feedback

Full Stack Teams

Photograph © Tom Poppendieck
Don’t Outsource Design

Two Thirds of the features and functions in a specification are unnecessary.

End-to-end Feedback
From Delivery Teams To Problem Solving Teams

Delivery

Problem Solving

End-to-end Feedback
Start with Signals
Focus on Problems
Plan with Hypotheses

Signals / Patterns

Analysis & Conclusion
Problem Statement

Experiment
Hypothesis

Do Multiple Experiments
Use Data to Decide

Not Requirements
Not Features
Not Estimates

Not a Backlog of Stories
Don’t Guess at a Solution

The Process

Experimentation and Learning
Google I/O 2014  Design Sprint
A process for prototyping and testing any product in 5 days.

Sketch: Jake Knapp
Leveraging Diversity: Individual Idea Generation

1. **NOTES**
   - Gather key info
   - 20 min.

2. **IDEAS**
   - Doodle rough solutions
   - 20 min.

3. **CRAZY 8s**
   - Try rapid variations
   - 8 min.

4. **SOLUTION SKETCH**
   - Figure out the details
   - 30+ min.

Sketch: Jake Knapp
Stop Voting

1. Explore multiple ideas, including outliers.
2. Pursue a variety of ideas with champions and volunteers.
3. Gradually narrow the ideas to those that will work.
4. Maintain multiple options as long as possible.

Conformity Bias: Those who think they are in the minority self-silence
The Future of Software Engineering

Software Engineering in the Cloud

1. Scale Out >> Scale Up
2. Infrastructure as Code
3. The New Technology Stack

The Three Ways

4. Continuous Deployment Pipeline
5. Full-Stack Problem-Solving Teams
6. Experimentation and Learning
The Future has Arrived

Thank You!

Mary Poppendieck
mary@poppendieck.com
www.poppendieck.com