Scaling Personalization
DevOps and AlgoOps at Stitch Fix

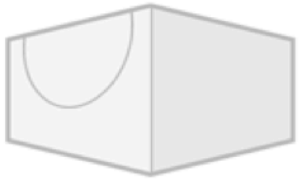
Randy Shoup
@randyshoup
linkedin.com/in/randyshoup

Background

- VP Engineering at Stitch Fix
  - Using technology and data science to revolutionize clothing retail
- Consulting "CTO as a service"
  - Helping companies move fast at scale ☺
- Director of Engineering for Google App Engine
  - World's largest Platform-as-a-Service
- Chief Engineer at eBay
  - Evolving multiple generations of eBay's infrastructure

Stitch Fix

Create Your Style Profile.

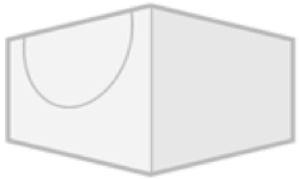


Get Five Hand-picked Items.

Keep What You Like.
Send Back the Rest.

Stitch Fix

Create Your Style Profile.



Get Five Hand-picked Items.

Keep What You Like.
Send Back the Rest.

How do you prefer clothes to fit the top half of your body?

- Mostly Tight / Form Fitting
- Prefer Fitted / Showing my Figure
- Straight
- Mostly Loose
- Oversized

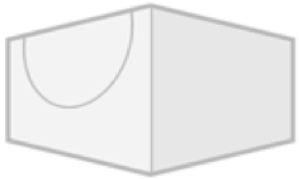
ear for pants, jeans, and skirts?

an waist

Skirts L

Stitch Fix

Create Your Style Profile.

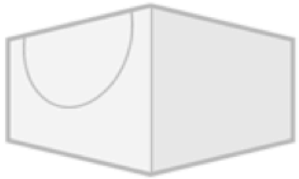


Get Five Hand-picked Items.

Keep What You Like.
Send Back the Rest.

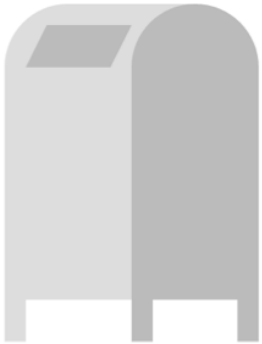
Stitch Fix

Create Your Style Profile.



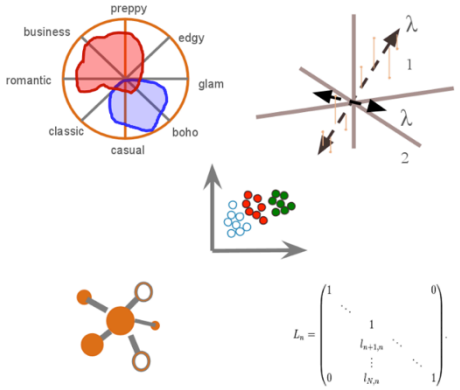
Get Five Hand-picked Items.

Keep What You Like.
Send Back the Rest.



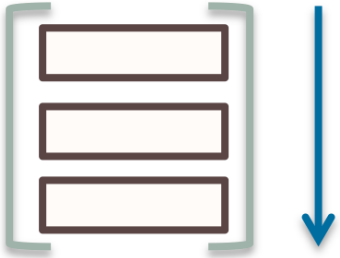
Personalized Recommendations

Inventory



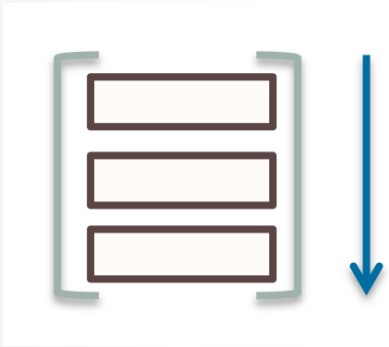
Machine learning

Algorithmic recommendations

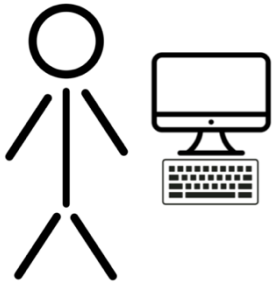


Expert Human Curation

Algorithmic recommendations



Human curation



Humans + Data Science

- 1:1 Ratio of Data Science to Engineering
  - More than 100 software engineers
  - ~80 data scientists and algorithm developers
  - Unique ratio in our industry
- Apply intelligence to *every* part of the business
  - Buying
  - Inventory management
  - Logistics optimization
  - Styling recommendations
  - Demand prediction
- Humans and machines augmenting each other

Scaling Personalization

- Organizing for Speed
- What to Build / What NOT to Build
- How to Build
- Continuous Experimentation

Scaling Personalization

- Organizing for Speed
- What to Build / What NOT to Build
- How to Build
- Continuous Experimentation

Conway's Law

- Organization determines architecture
  - Design of a system will be a reflection of the communication paths within the organization
- Modular system requires modular organization
  - Small, independent teams lead to more flexible, composable systems
  - Larger, interdependent teams lead to larger systems
- We can engineer the system we want by engineering the organization

Small "Service" Teams

- Full-Stack, "2 Pizza" Teams
  - No team should be larger than can be fed by 2 large pizzas
  - Typically 4-6 people
  - All disciplines required for the team to function
- Aligned to Business Domains
  - Clear, well-defined area of responsibility
  - Single service or set of related services
  - Deep understanding of business problems
- Growth through "cellular mitosis"

Ideally, 80% of project work should be within a team boundary.

Scaling Personalization

- Organizing for Speed
- What to Build / What NOT to Build
- How to Build
- Continuous Experimentation

"Building the wrong thing is the biggest waste in software development."

-- Mary and Tom Poppendieck,
Lean Software Development

What problem are you trying to solve?

"A problem well-stated is a problem half-solved."

-- Charles Kettering, former head of research for General Motors

What Problem Are You Trying to Solve?

- Focus on what is important for your business
- Problem might be solved without any technology at all
  - Redefine the problem
  - Change the business process
  - Implement manually for a while before automating in an application

Buy, Not Build

- Use Cloud Infrastructure
  - Faster, cheaper, better than we can do ourselves
  - Stitch Fix has no owned physical infrastructure anywhere in the world
- Prefer Open Source
  - Kubernetes, Docker, Istio
  - MySQL, Postgres, Redis, Elastic Search
  - Machine learning models
  - Etc.
  - Usually better than the commercial alternatives (!)

Buy, Not Build

- Third-Party Services
  - Stitch Fix uses >50 third party services
  - Logging, monitoring, alerting
  - Project management, bug tracking
  - Billing, fraud detection
  - Etc.

- Focus on our core competency
  - Use services for **everything else** (!)

Soon it will be just as common to run your own data center as it is to run your own electrical power generation.

Experimental Discipline

- State your hypothesis
  - What metrics do you expect to move and why
  - Understand your baseline
- Run a real A | B test
  - Sample size
  - Isolated treatment and control groups
  - No peeking or quitting early!
- Obsessively log and measure
  - Understand customer and system behavior
  - Understand why this experiment worked or did not

Experimental Discipline

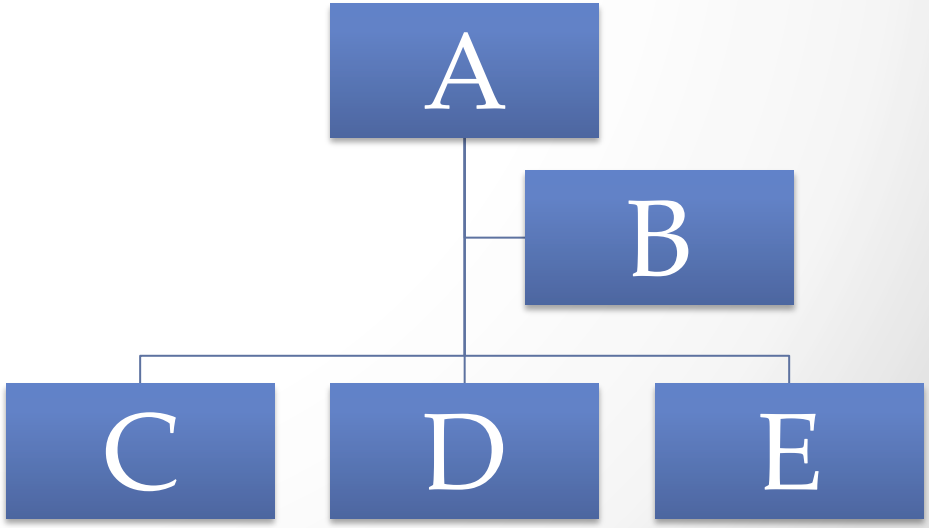
- Listen to the data
  - Data trumps hope and intuition
  - Develop insights for next experiment
- Thinking of the experiment is art; evaluating it is science
- Rinse and Repeat
  - This is a journey, not a single step

Scaling Personalization

- Organizing for Speed
- What to Build / What NOT to Build
- How to Build
- Continuous Experimentation

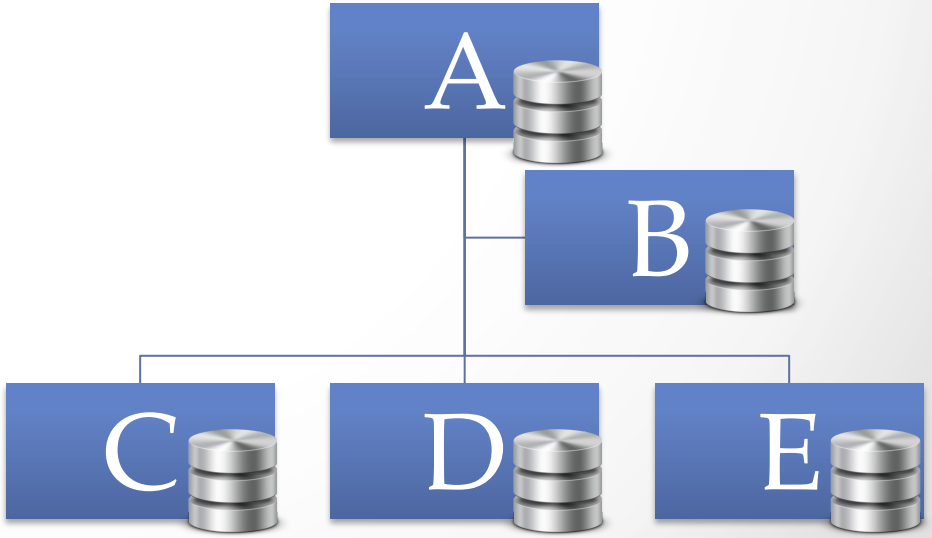
Microservices

- Single-purpose
- Simple, well-defined interface
- Modular and composable
- Independently deployable



Microservices

- Single-purpose
- Simple, well-defined interface
- Modular and composable
- Independently deployable
- **Isolated persistence (!)**



Evolution to Microservices

- eBay
  - 5th generation today
  - Monolithic Perl → Monolithic C++ → Java → microservices
- Twitter
  - 3rd generation today
  - Monolithic Rails → JS / Rails / Scala → microservices
- Amazon
  - Nth generation today
  - Monolithic Perl / C++ → Java / Scala → microservices

No one starts with microservices

…

Past a certain scale, everyone ends up with microservices

If you don't end up regretting your early technology decisions, you probably over-engineered.

Quality Discipline

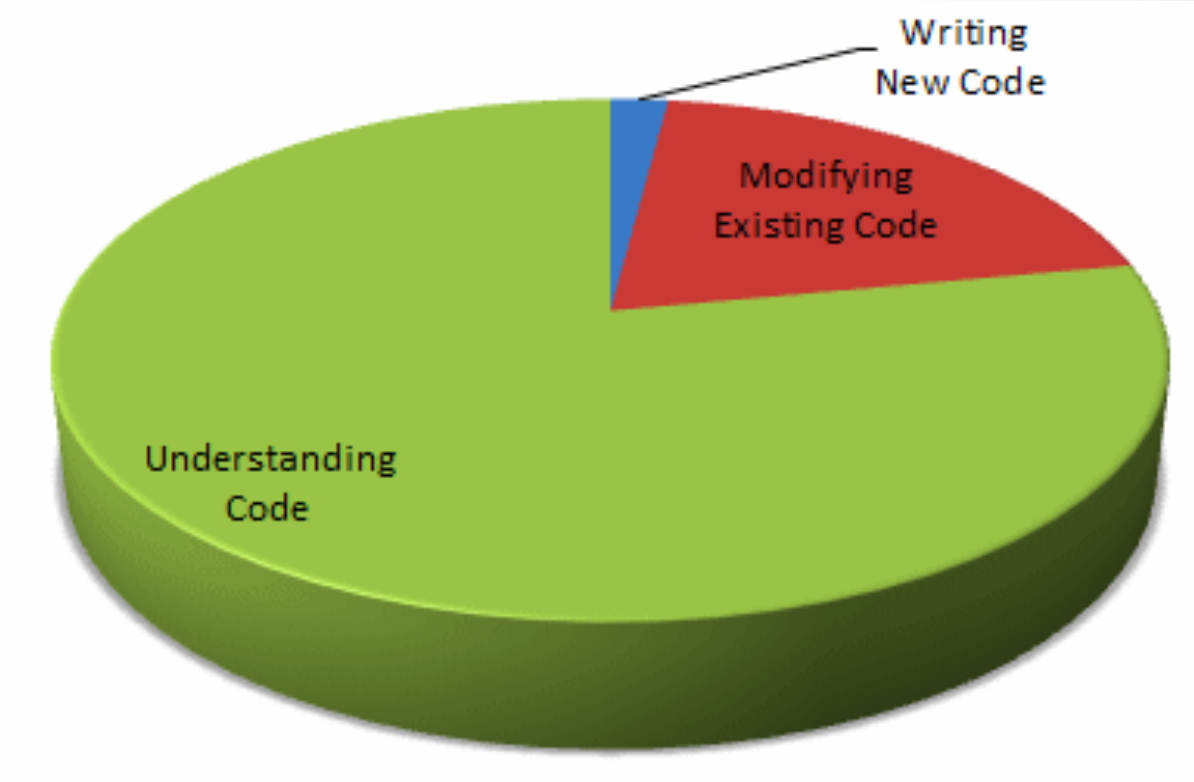
- Quality and Reliability are "Priority-0 features"
  - Equally important to users as product features and engaging user experience
- Developers responsible for
  - Features
  - Quality
  - Performance
  - Reliability
  - Manageability

Test-Driven Development

- Developers write tests and code together
  - Definition of done includes automated testing of the feature or bug fix
- Tests make better code
  - Confidence to break things
  - Courage to refactor mercilessly
  - Development velocity
- Tests make better systems
  - Catch bugs earlier, fail faster

Optimizing Developer Effort

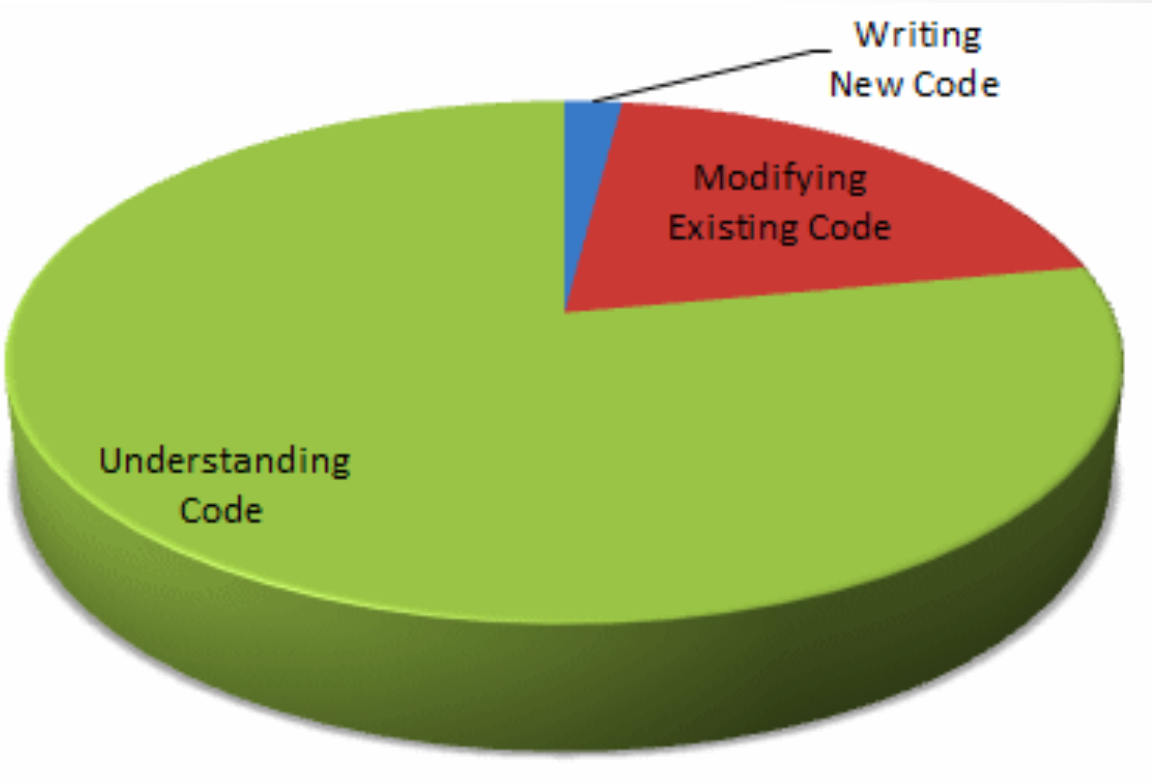
- 75% reading existing code
- 20% modifying existing code
- 5% writing new code



https://blogs.msdn.microsoft.com/peterhal/2006/01/04/what-do-programmers-really-do-anyway-aka-part-2-of-the-yardstick-saga/

Optimizing Developer Effort

- 75% reading existing code
- 20% modifying existing code
- 5% writing new code



https://blogs.msdn.microsoft.com/peterhal/2006/01/04/what-do-programmers-really-do-anyway-aka-part-2-of-the-yardstick-saga/

"We don't have time to do it right!"

"Do you have time to do it **twice**?"

The more constrained you are on time or resources, the more important it is to build it right the first time.

Build It Right (Enough) The First Time

- Build one great thing instead of two half-finished things
- Right ≠ Perfect (80 / 20 Rule)
- ➔ Basically no bug tracking system (!)
  - Bugs are fixed as they come up
  - Backlog contains features we want to build
  - Backlog contains technical debt we want to repay

Continuous Integration

- Small incremental changes
  - Faster feedback loop
  - Easy to understand, code-review, test, roll back
- Large changes are risky
  - Risk of code change is nonlinear in the size of the change
  - Ex. Initial memcache service submission
- Main code branch always shippable
  - Increased rate of change while reducing risk of change

Scaling Personalization

- Organizing for Speed
- What to Build / What NOT to Build
- How to Build
- Continuous Experimentation

You Build It, You Run It.

-- Werner Vogels

You Model It, You Run It.

-- Stitch Fix Algorithms Team

AlgoOps!

Continuous Delivery

- Repeatable Deployment Pipeline
  - Low-risk, push-button deployment
  - Rapid release cadence
  - Rapid rollback and recovery
- Most applications deployed multiple times per day
- More solid systems
  - Release smaller units of work
  - Smaller changes to roll back or roll forward
  - Faster to repair, easier to understand, simpler to diagnose

Feature Flags

- Configuration "flag" to enable / disable a feature for a particular set of users
  - Independently discovered at eBay, Facebook, Google, etc.
- More solid systems
  - Decouple feature delivery from code delivery
  - Rapid on and off
  - Develop / test / verify in production
  - Dark launches
- Enables experimentation

Service Abstraction

- All runtime Engineering <-> Algorithms interactions through RESTful service interfaces
- Loose coupling enables experimentation
  - Evolve existing algorithms without coordination
  - Introduce new algorithms without coordination
- Performance and stability improvements without coordination

Scaling Personalization

- Organizing for Speed
- What to Build / What NOT to Build
- How to Build
- Continuous Experimentation

Small Teams: Ability and Incentive to Experiment

Test-Driven Development: Build Quality In

Microservices: Make Experiments Scale

Continuous Innovation: Make Experiments Safe and Fast

Thank You!

- @randyshoup
- linkedin.com/in/randyshoup