# Automation & Testing Saved a Project from the Brink of Collapse

Jonathan Solórzano-Hamilton
Sr. Site Reliability Engineer
Procore Tecnologies

## About Me

- Senior Site Reliability Engineer at Procore since June
- 7 years in DevOps @ UCLA
- 3 years @ Stanford University
- ~5 years assorted consulting, internships
- Studied & worked as an experimental physicist

# About Procore

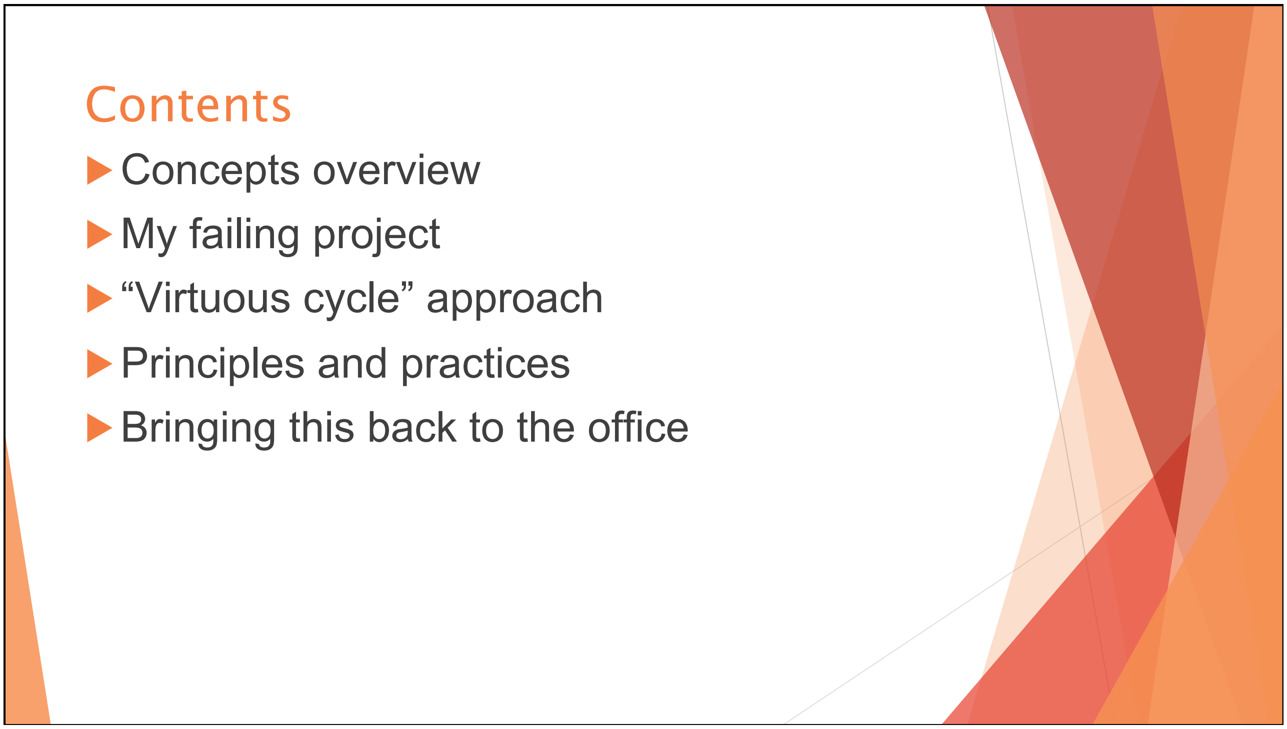
## What we are

- SaaS for the $10 trillion construction market
- Over 1,200 employees
- 11 locations globally, HQ in Carpinteria

## How we work

- Rails + PostgreSQL, mobile native, Elixir, JS, C, …
- CI with 100’s of production deploys per day across ~2000 servers
- R&D supported by robust QA organization
- Currently hiring QA at all levels: Senior Manager to Entry-level

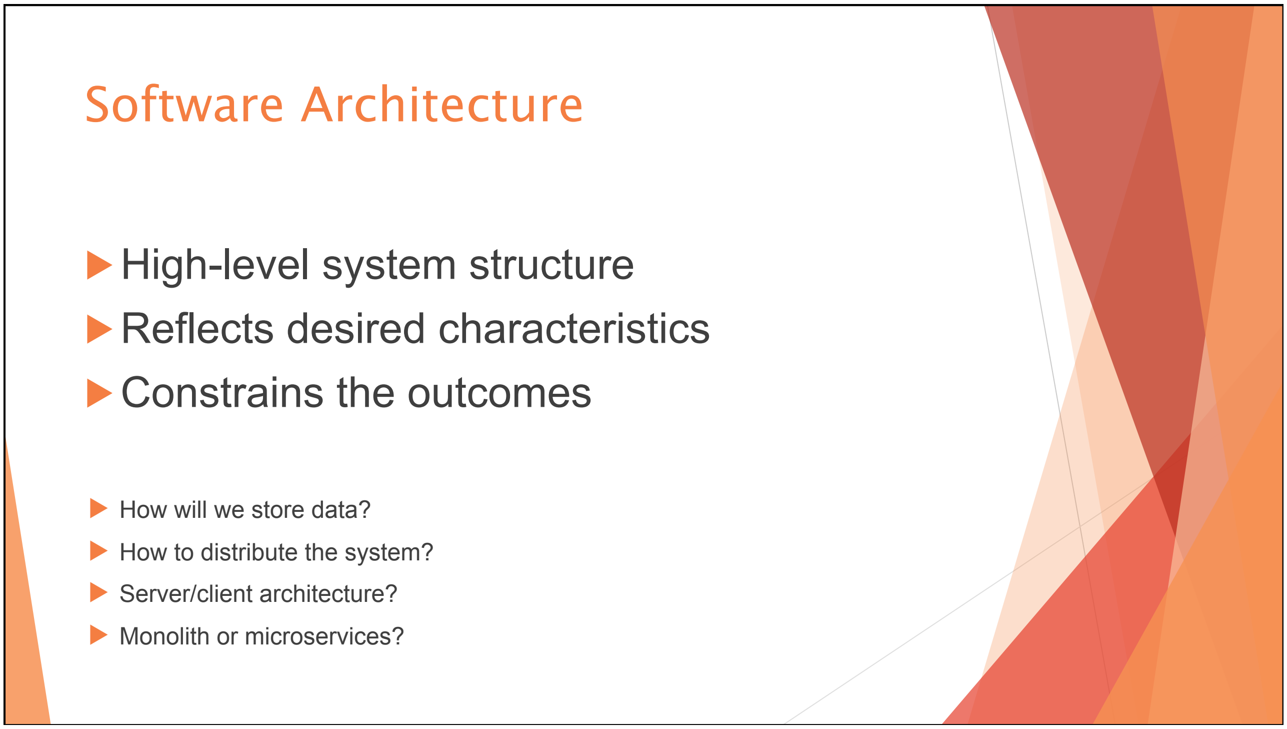
# Conceptual Overview



## Contents

- Concepts overview
- My failing project
- “Virtuous cycle” approach
- Principles and practices
- Bringing this back to the office

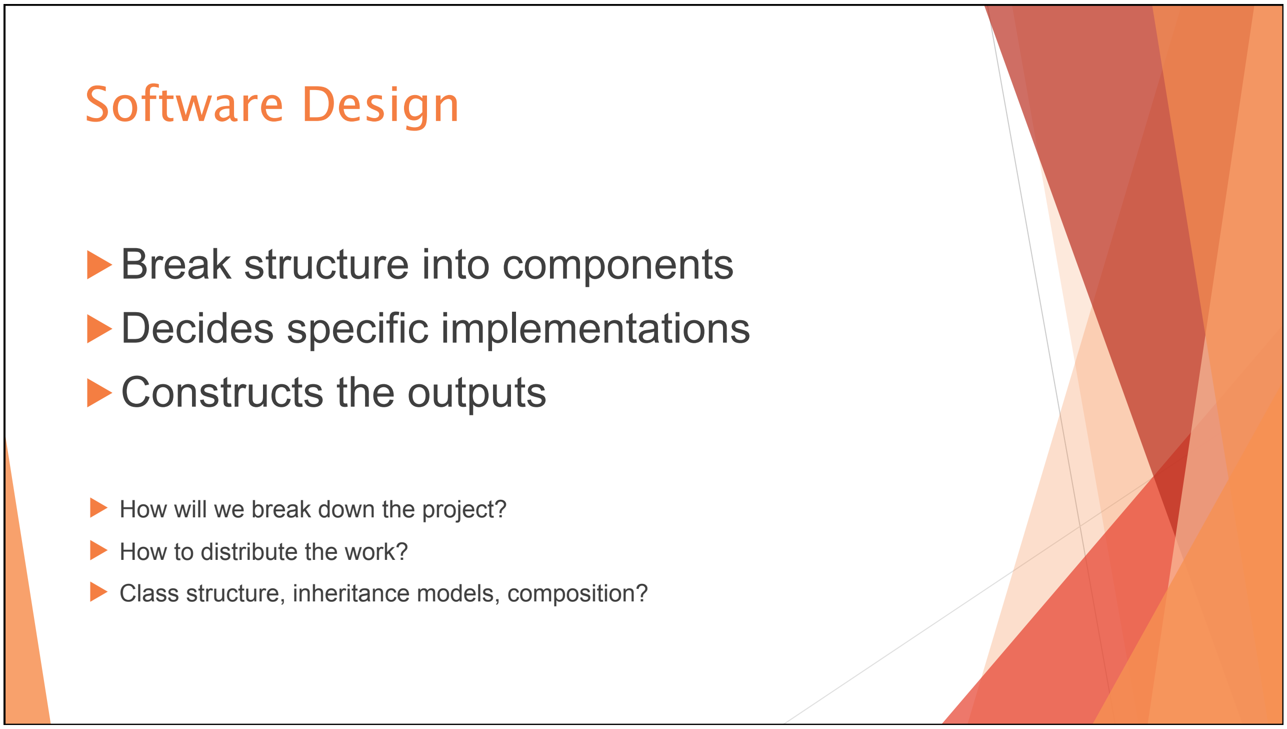
## Software: Architectre & Design



## Software Architecture

- High-level system structure
- Reflects desired characteristics
- Constrains the outcomes

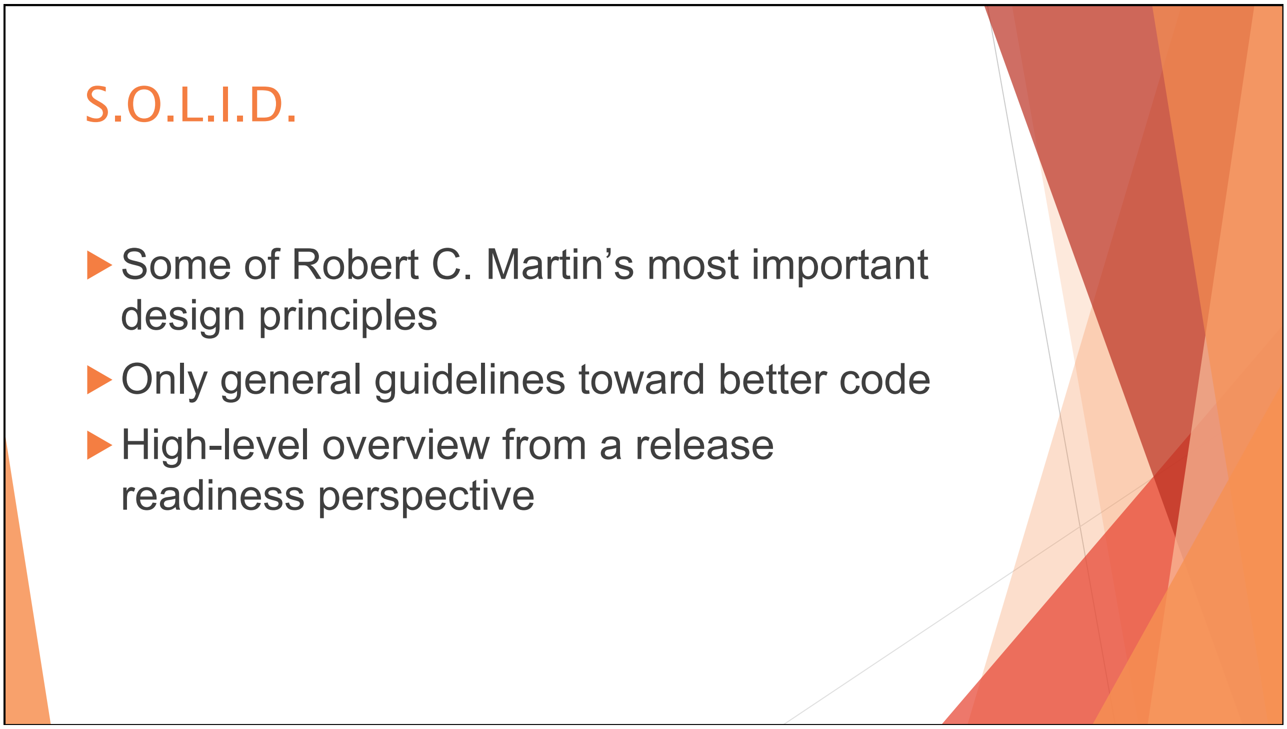
- How will we store data?
- How to distribute the system?
- Server/client architecture?
- Monolith or microservices?



## Software Design

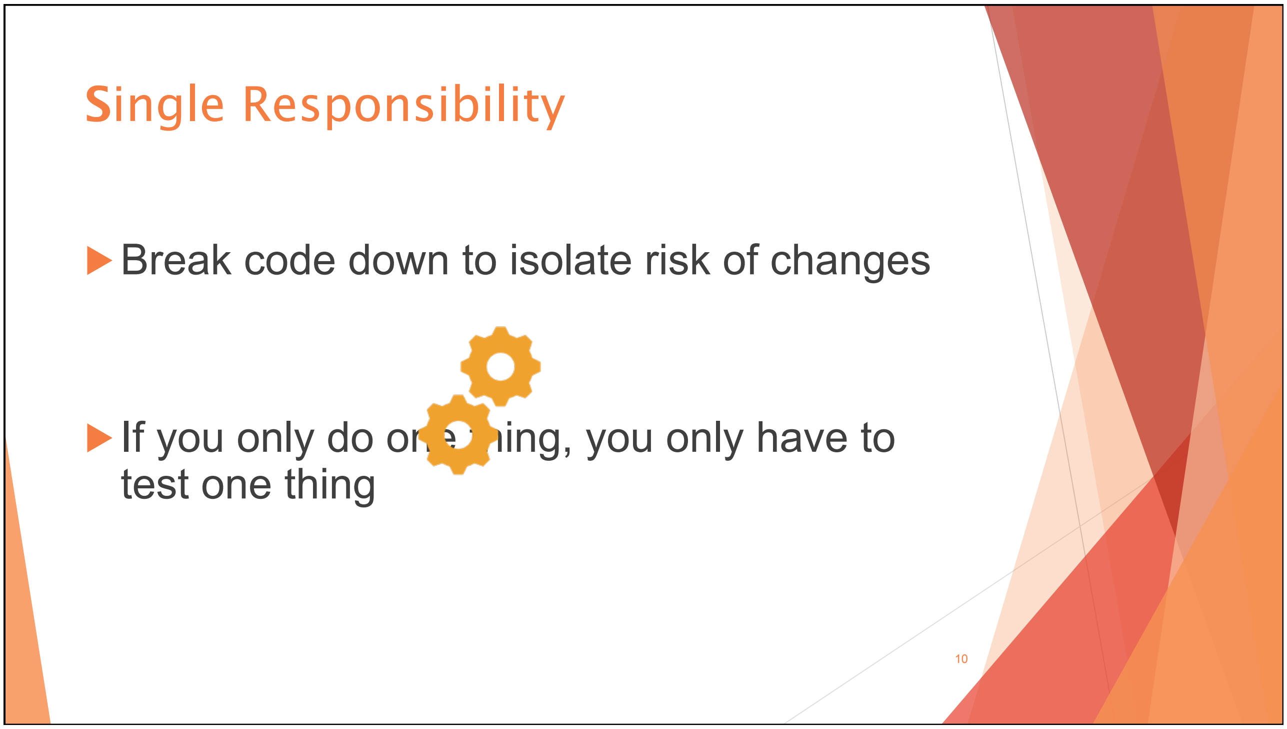
- Break structure into components
- Decides specific implementations
- Constructs the outputs

- How will we break down the project?
- How to distribute the work?
- Class structure, inheritance models, composition?



## S.O.L.I.D.

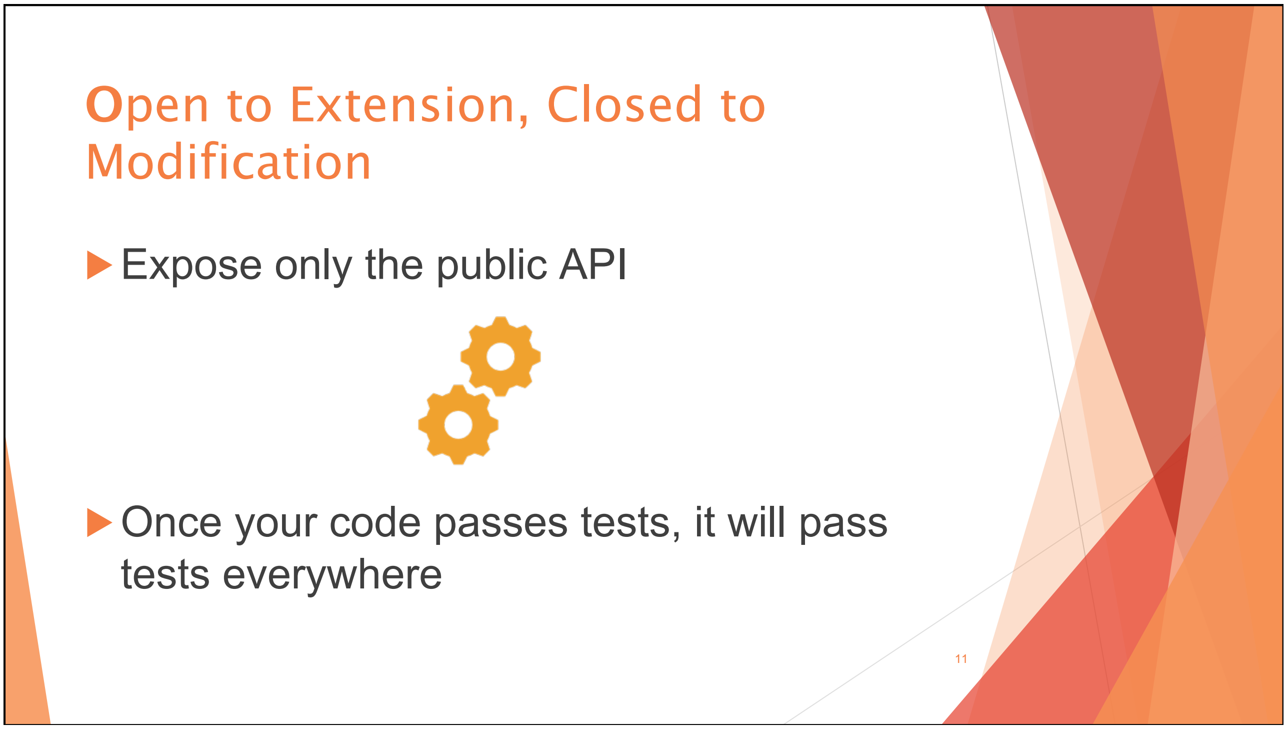
- Some of Robert C. Martin's most important design principles
- Only general guidelines toward better code
- High-level overview from a release readiness perspective



## Single Responsibility

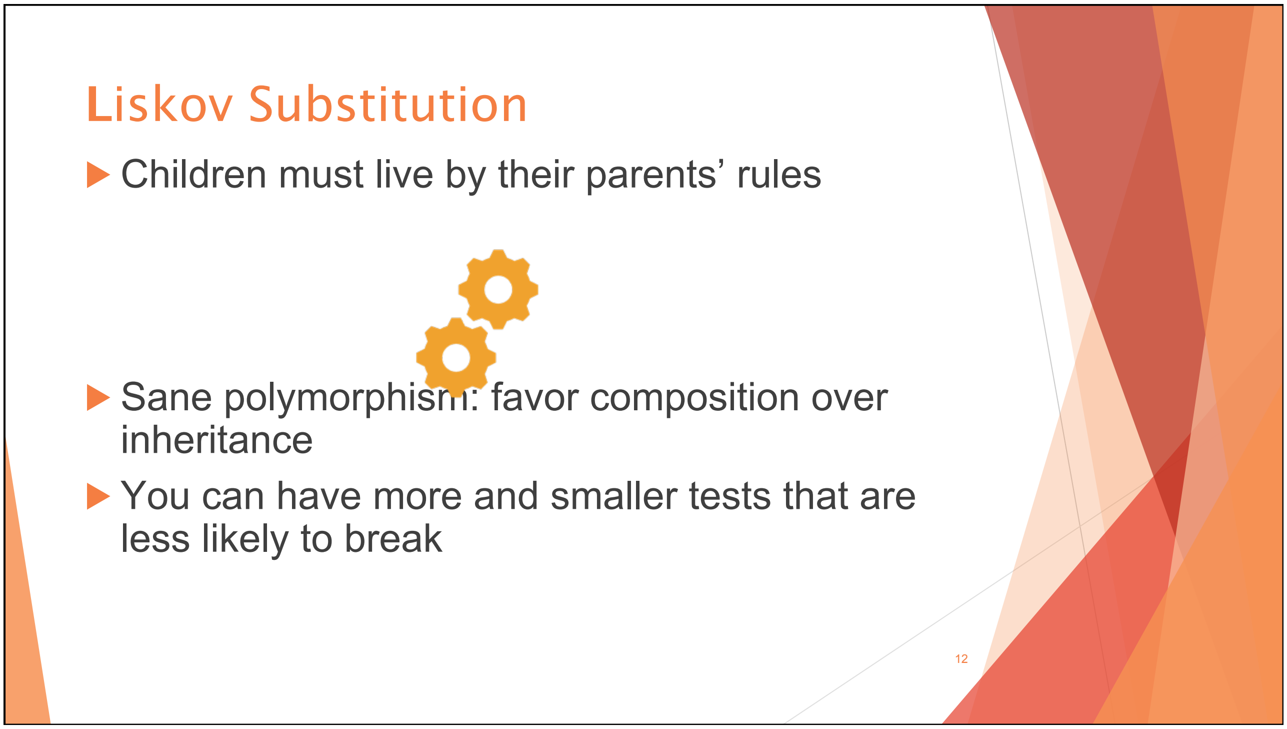
- Break code down to isolate risk of changes
- If you only do one thing, you only have to test one thing

10



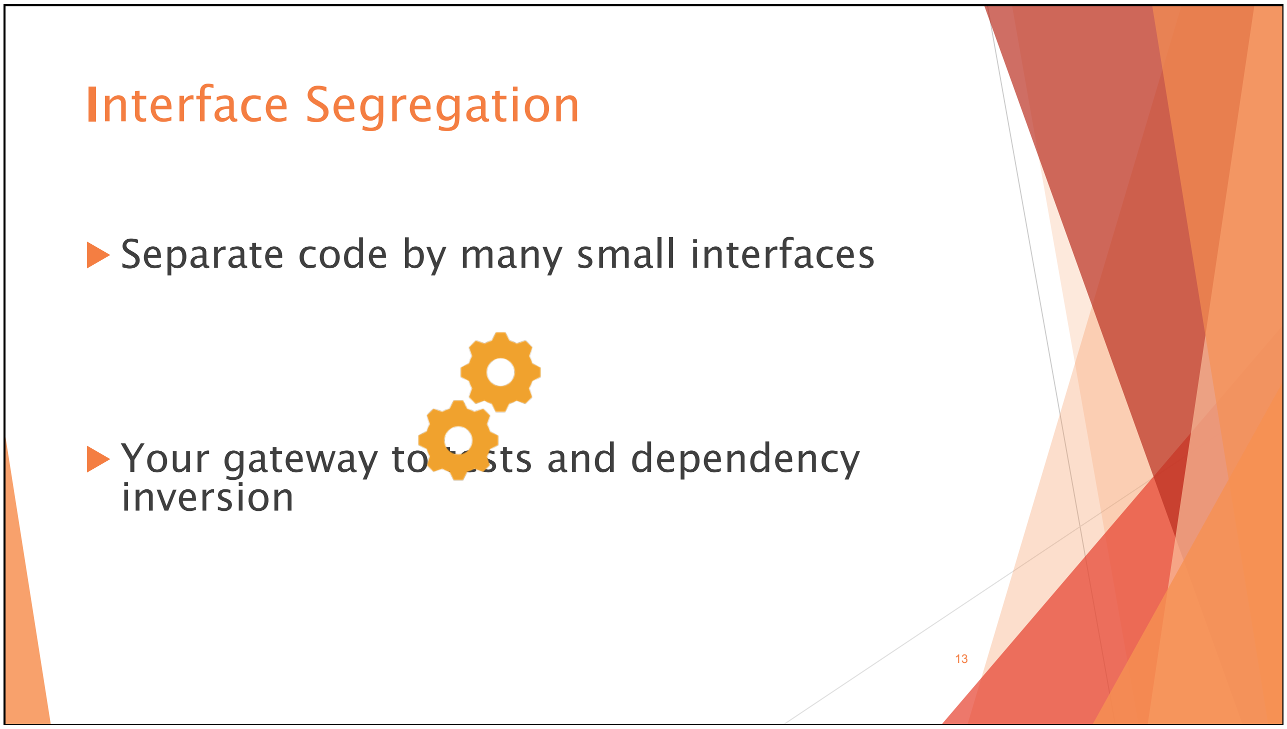
## Open to Extension, Closed to Modification

- Expose only the public API
- Once your code passes tests, it will pass tests everywhere



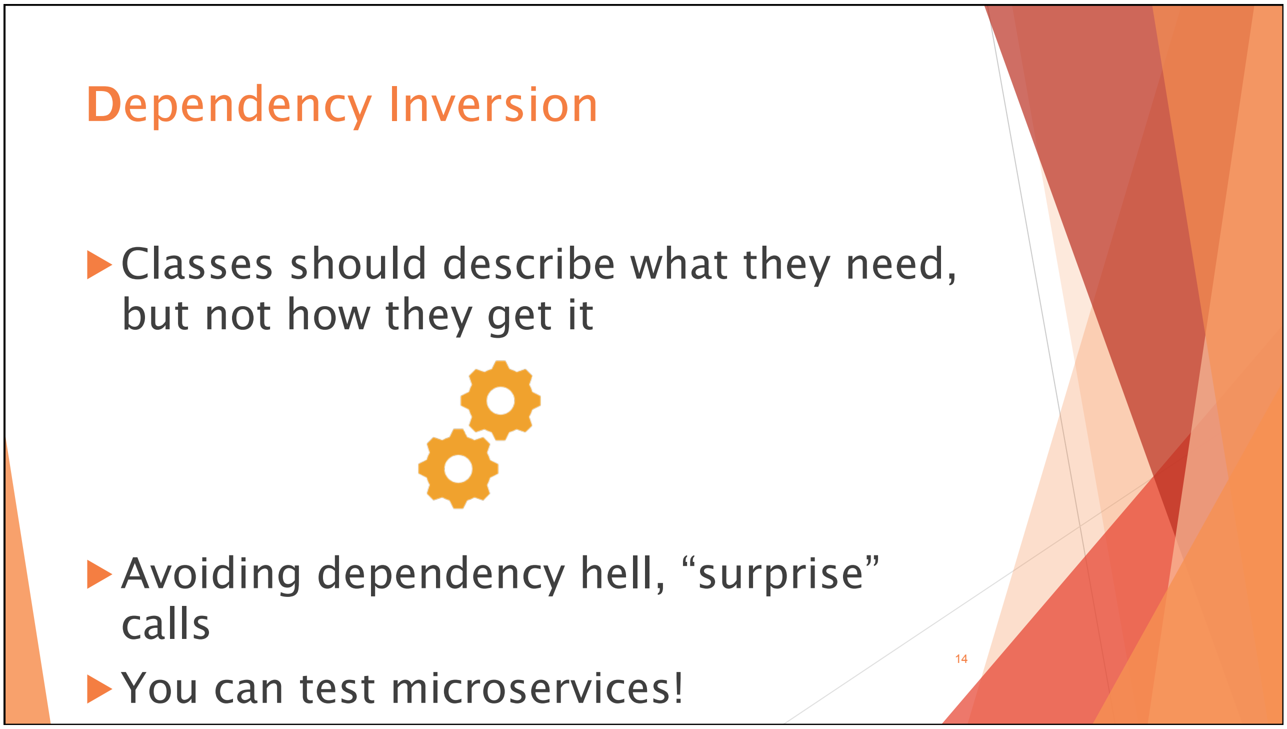
## Liskov Substitution

- Children must live by their parents’ rules
- Sane polymorphism: favor composition over inheritance
- You can have more and smaller tests that are less likely to break



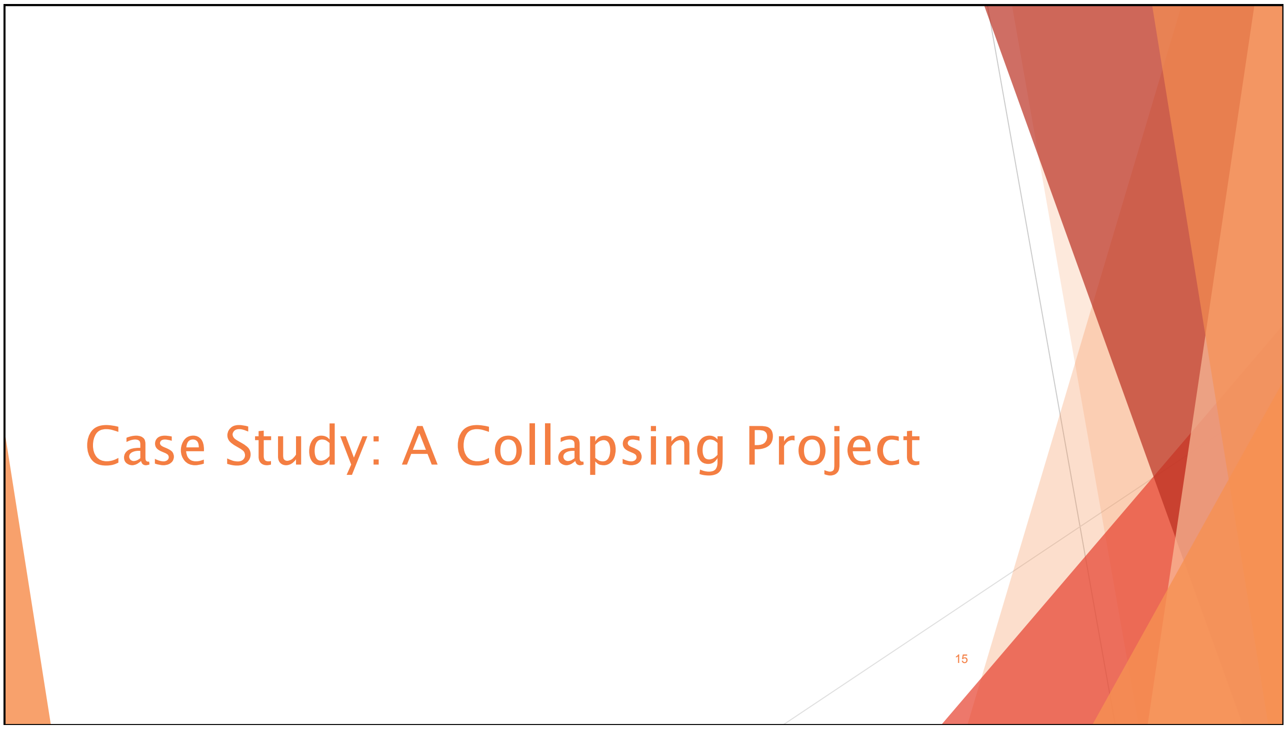
## Interface Segregation

- Separate code by many small interfaces
- Your gateway to tests and dependency inversion

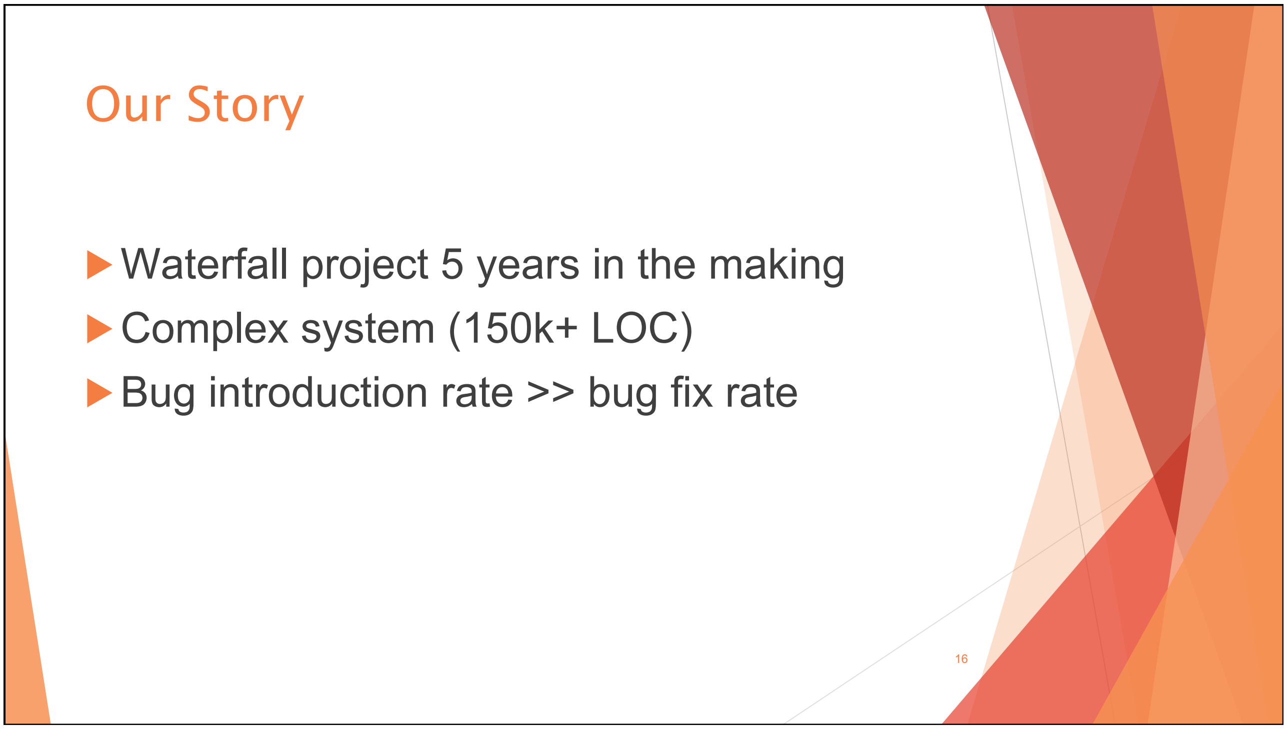


## Dependency Inversion

- Classes should describe what they need, but not how they get it
- Avoiding dependency hell, “surprise” calls
- You can test microservices!

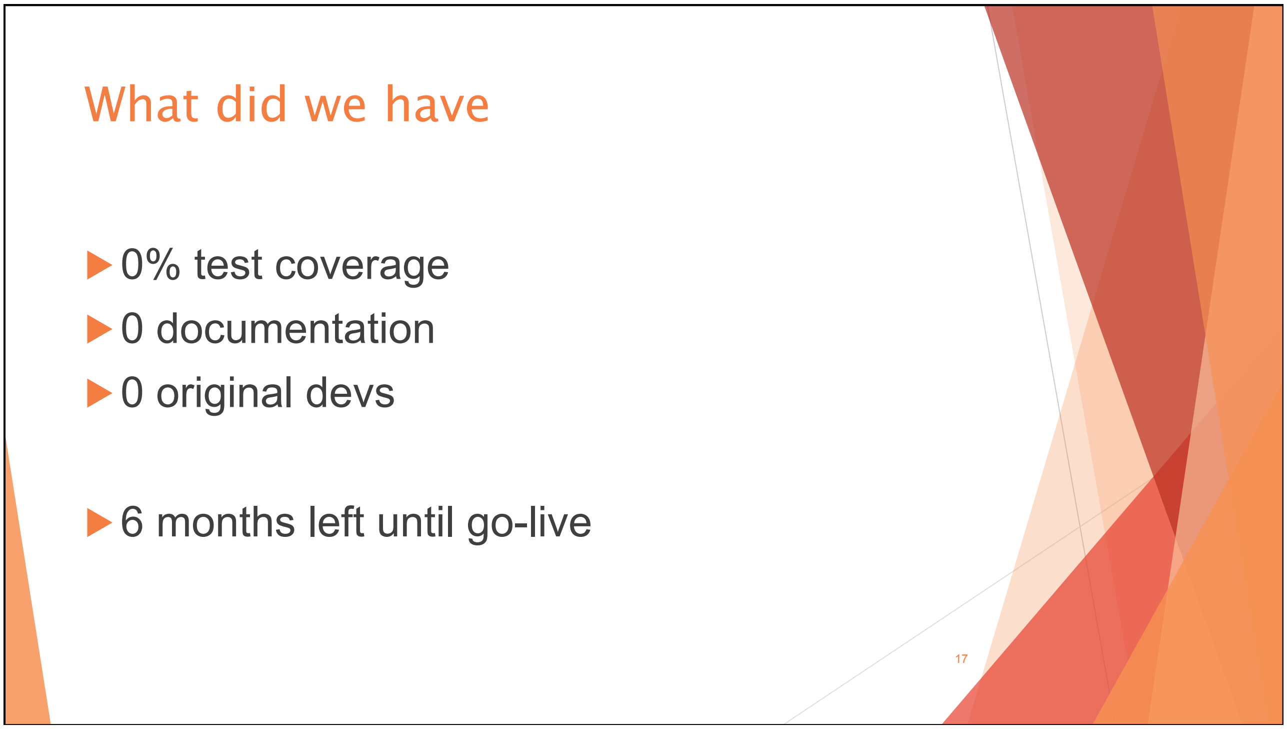


## Case Study: A Collapsing Project



## Our Story

- Waterfall project 5 years in the making
- Complex system (150k+ LOC)
- Bug introduction rate >> bug fix rate

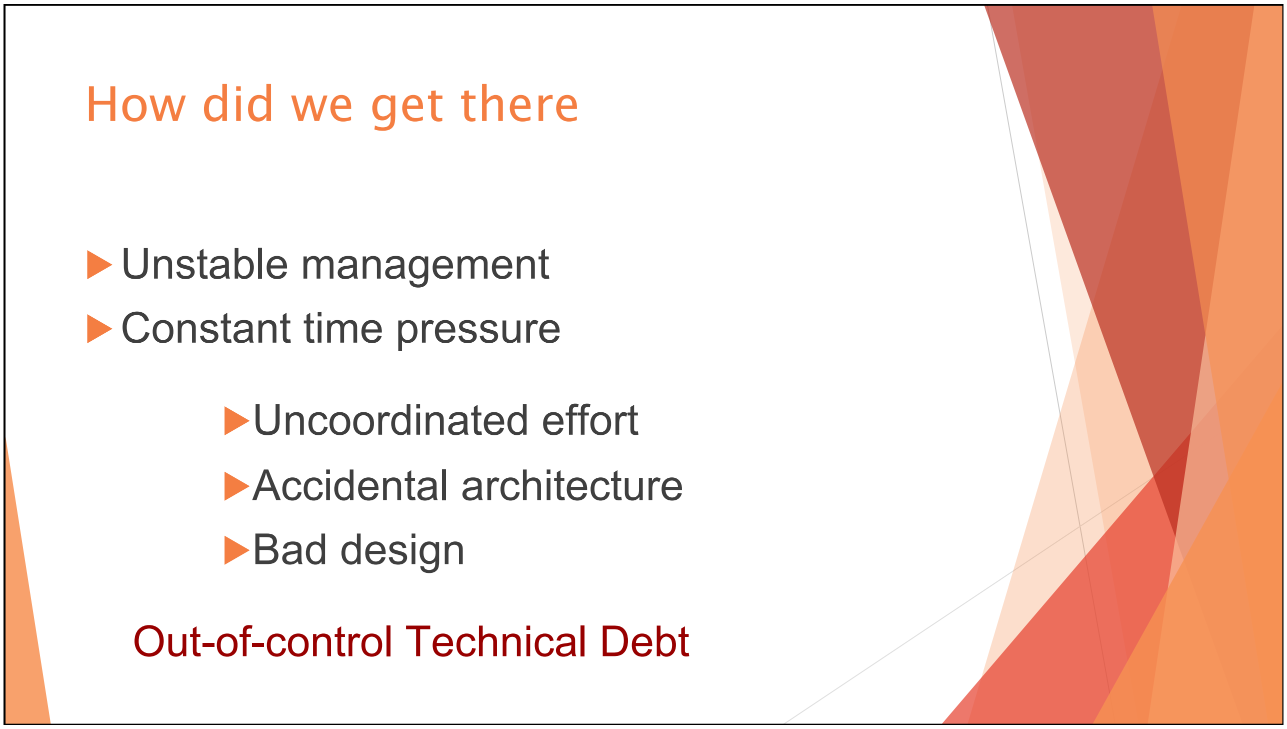


## What did we have

- 0% test coverage
- 0 documentation
- 0 original devs

- 6 months left until go-live

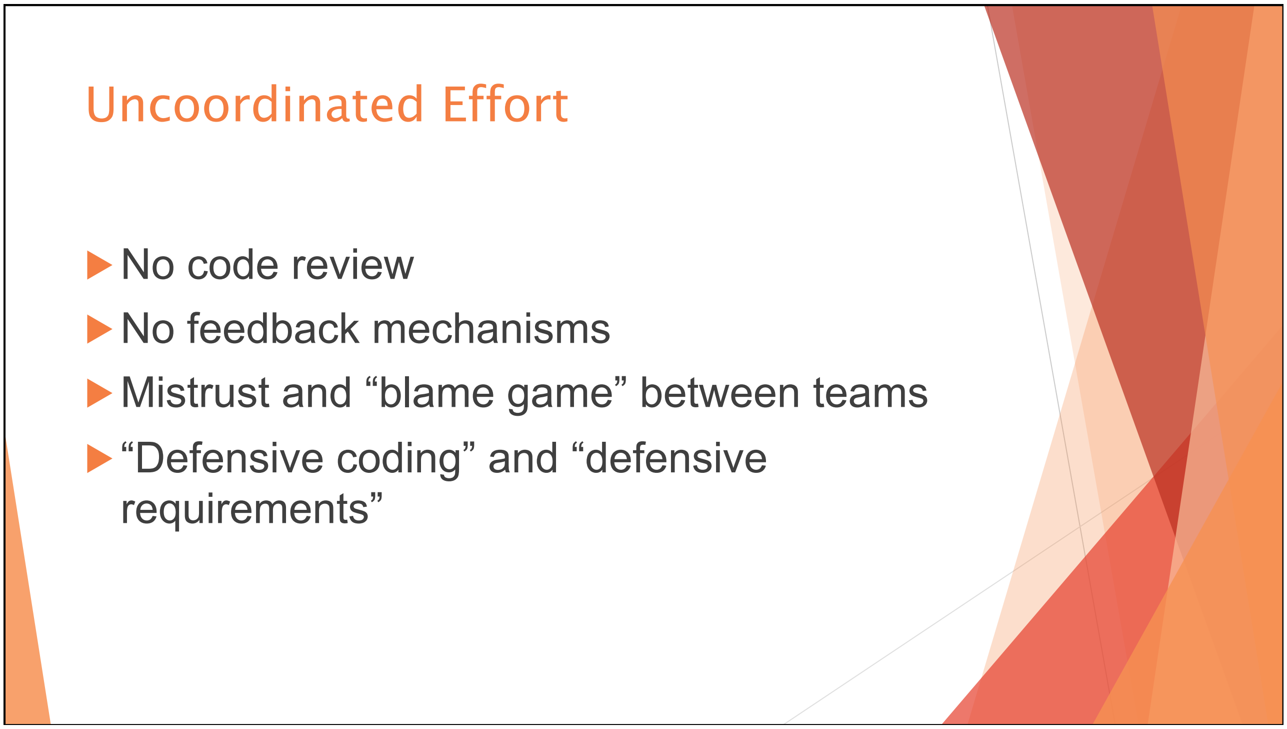
17



## How did we get there

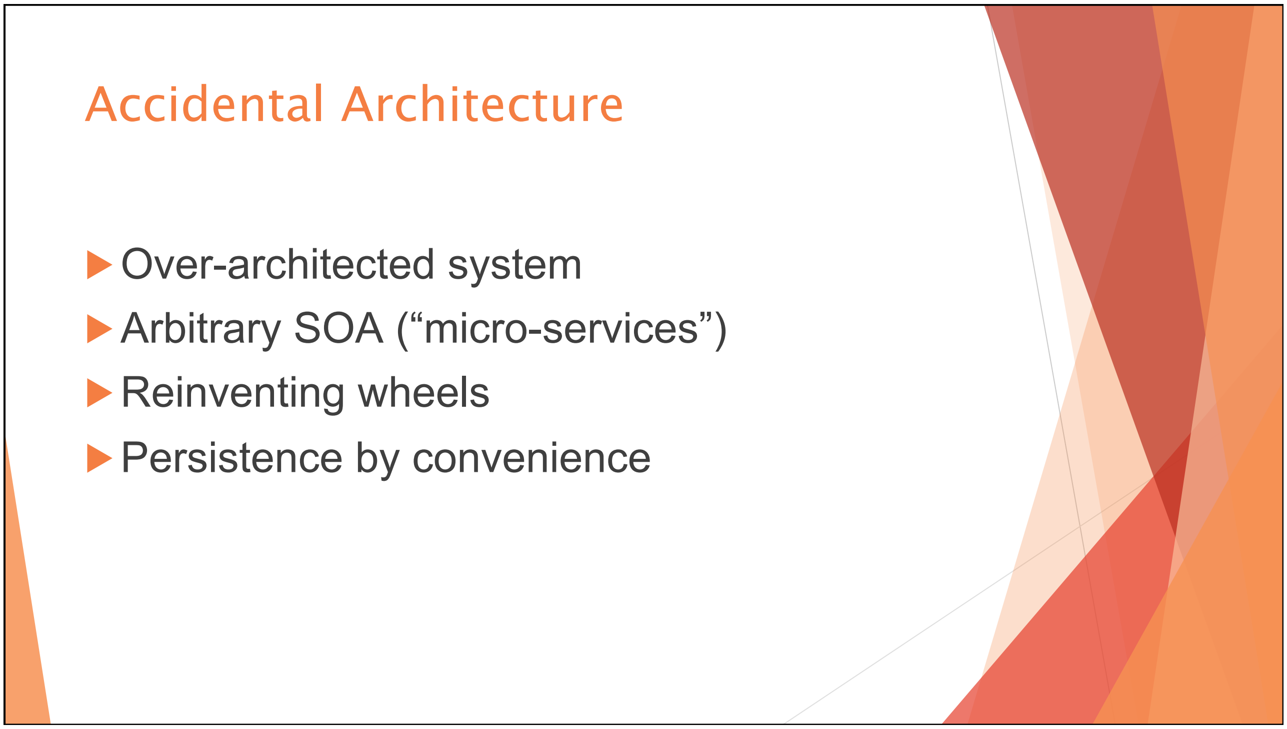
- Unstable management
- Constant time pressure
  - Uncoordinated effort
  - Accidental architecture
  - Bad design

Out-of-control Technical Debt



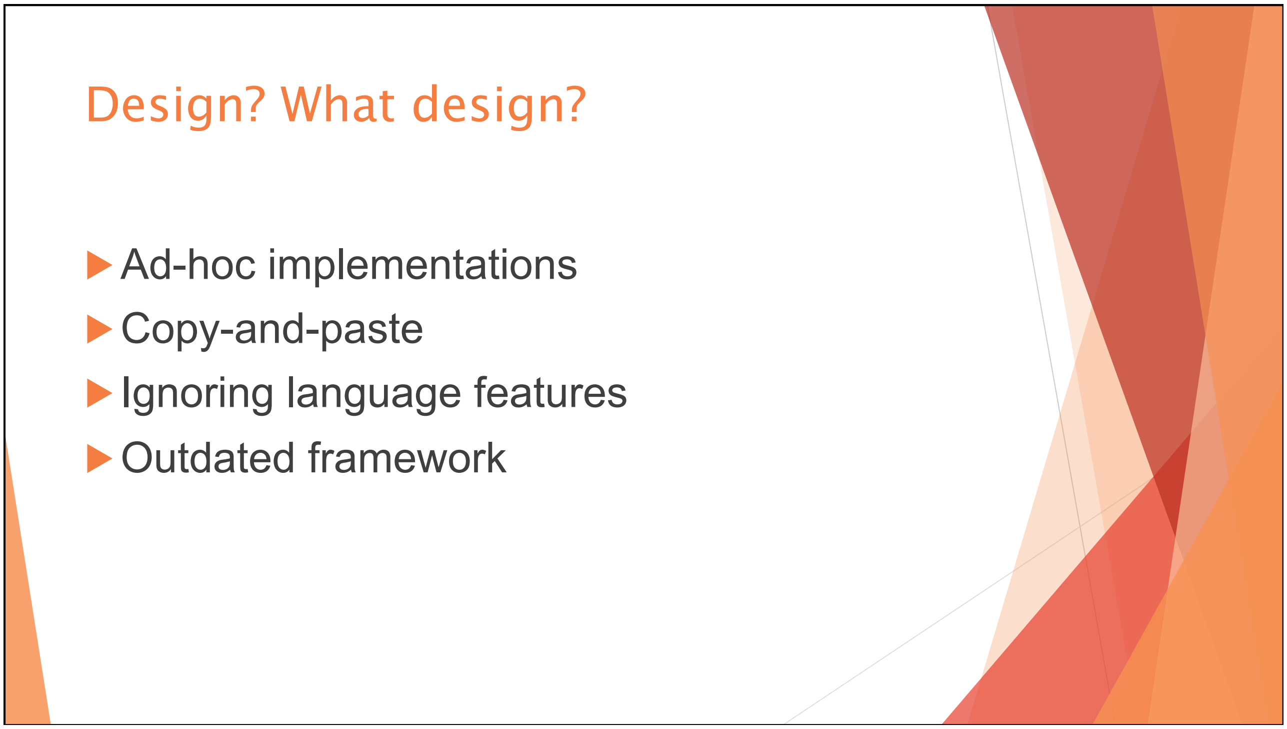
## Uncoordinated Effort

- No code review
- No feedback mechanisms
- Mistrust and “blame game” between teams
- “Defensive coding” and “defensive requirements”



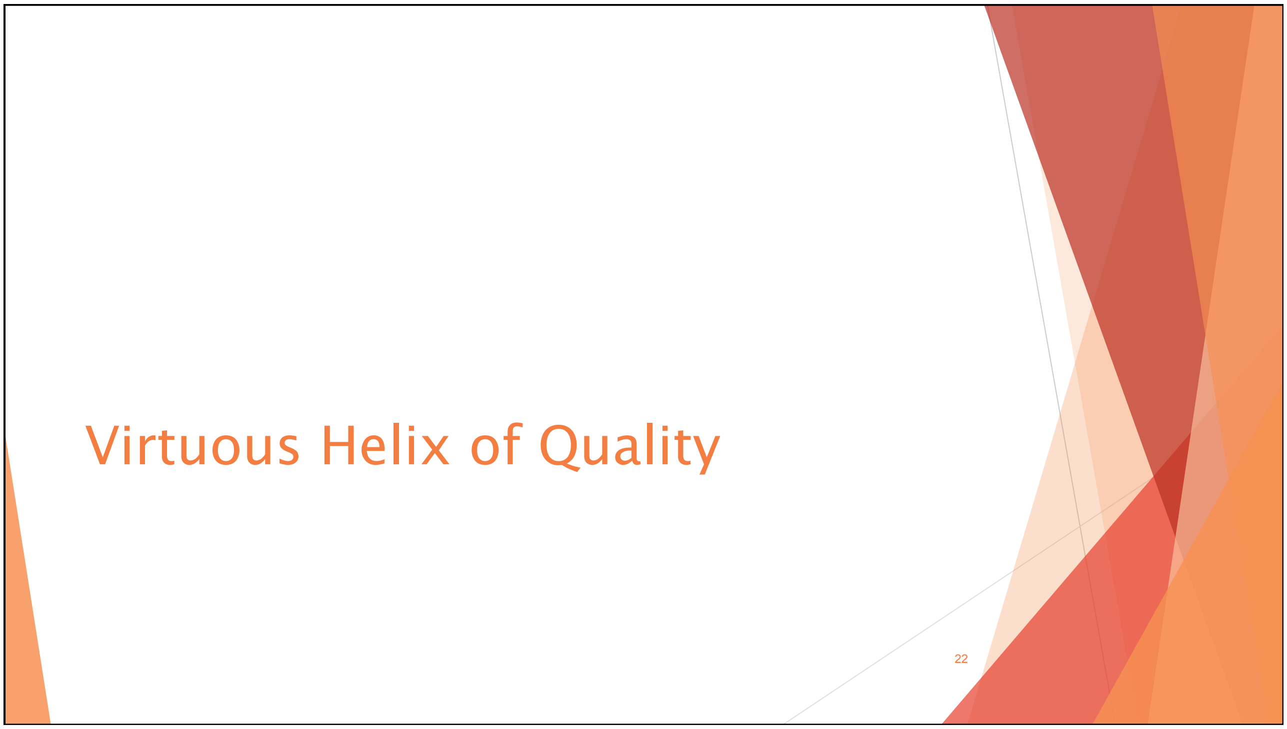
## Accidental Architecture

- Over-architected system
- Arbitrary SOA (“micro-services”)
- Reinventing wheels
- Persistence by convenience

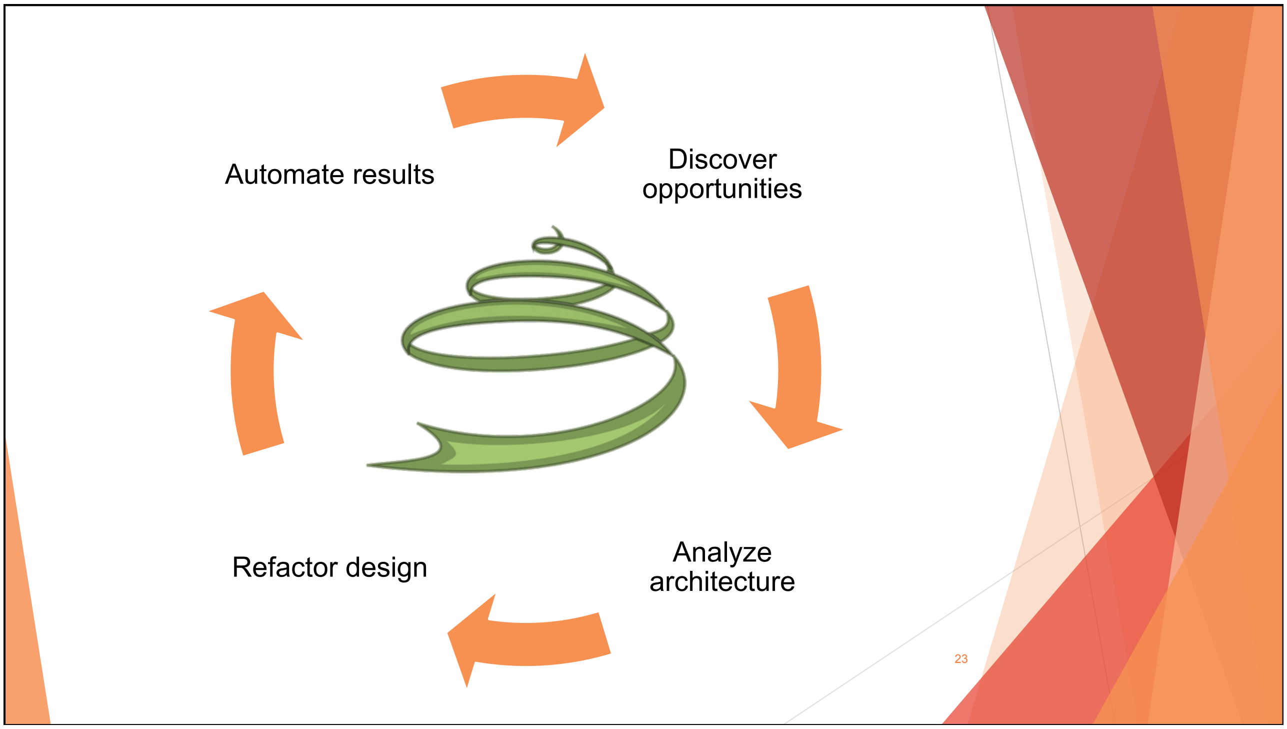


## Design? What design?

- Ad-hoc implementations
- Copy-and-paste
- Ignoring language features
- Outdated framework



## Virtuous Helix of Quality

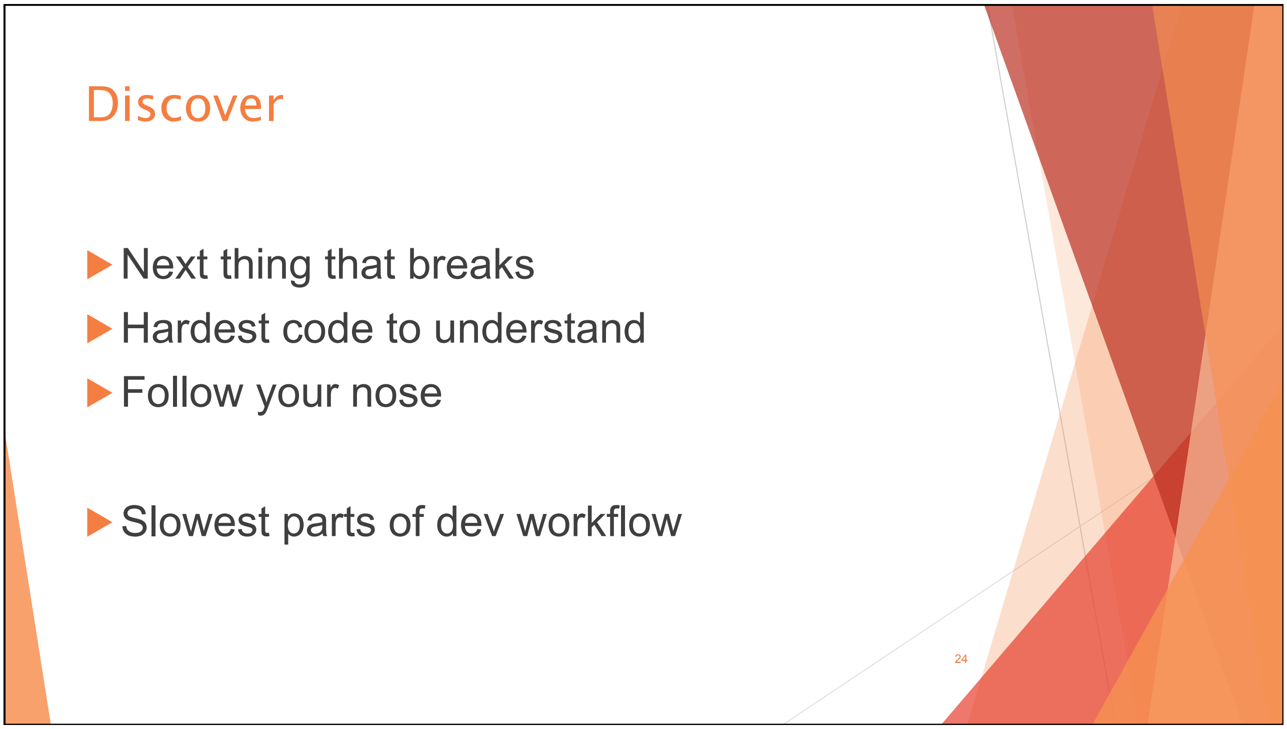


Discover opportunities

Analyze architecture

Refactor design

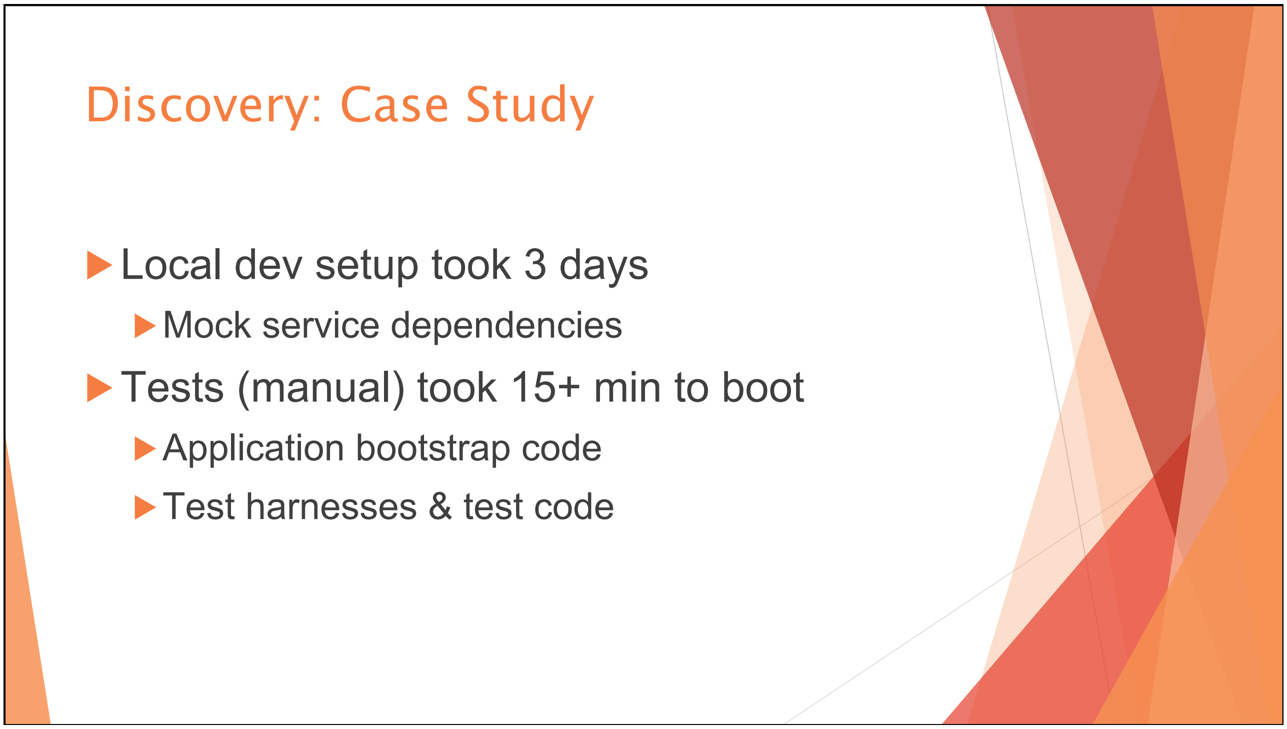
Automate results



## Discover

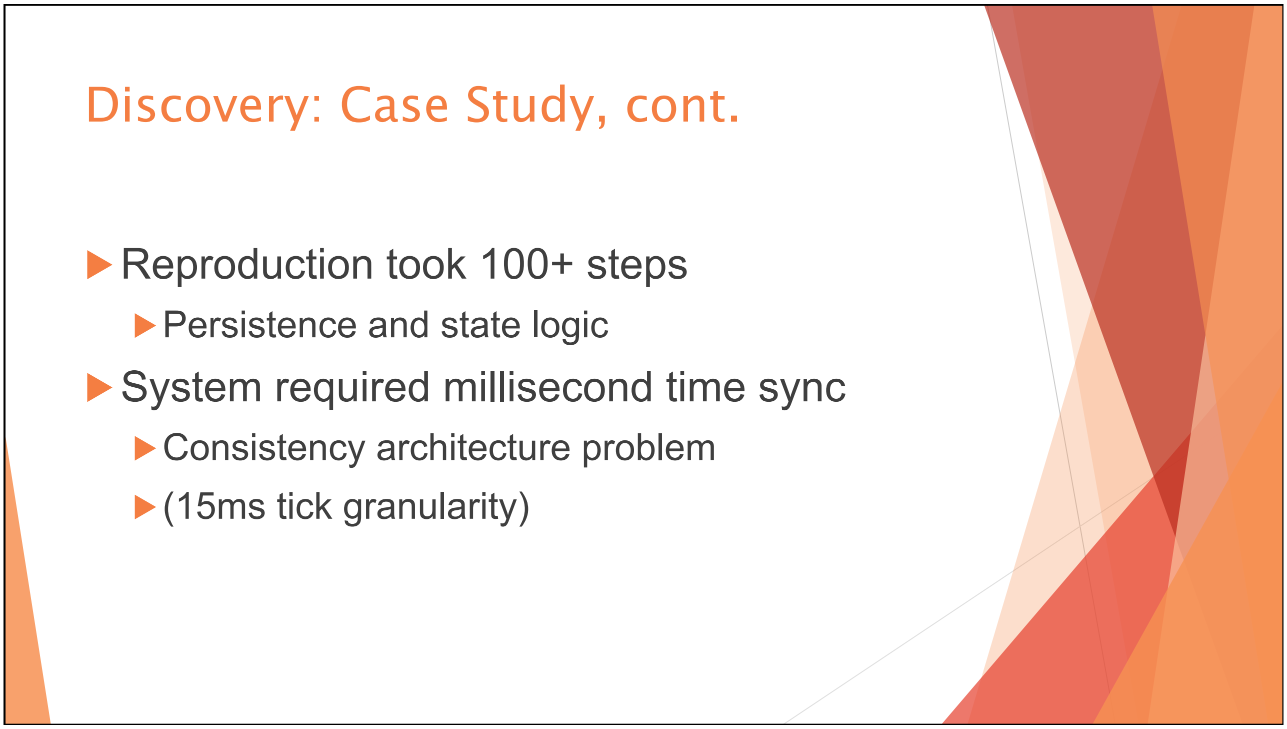
- Next thing that breaks
- Hardest code to understand
- Follow your nose

- Slowest parts of dev workflow



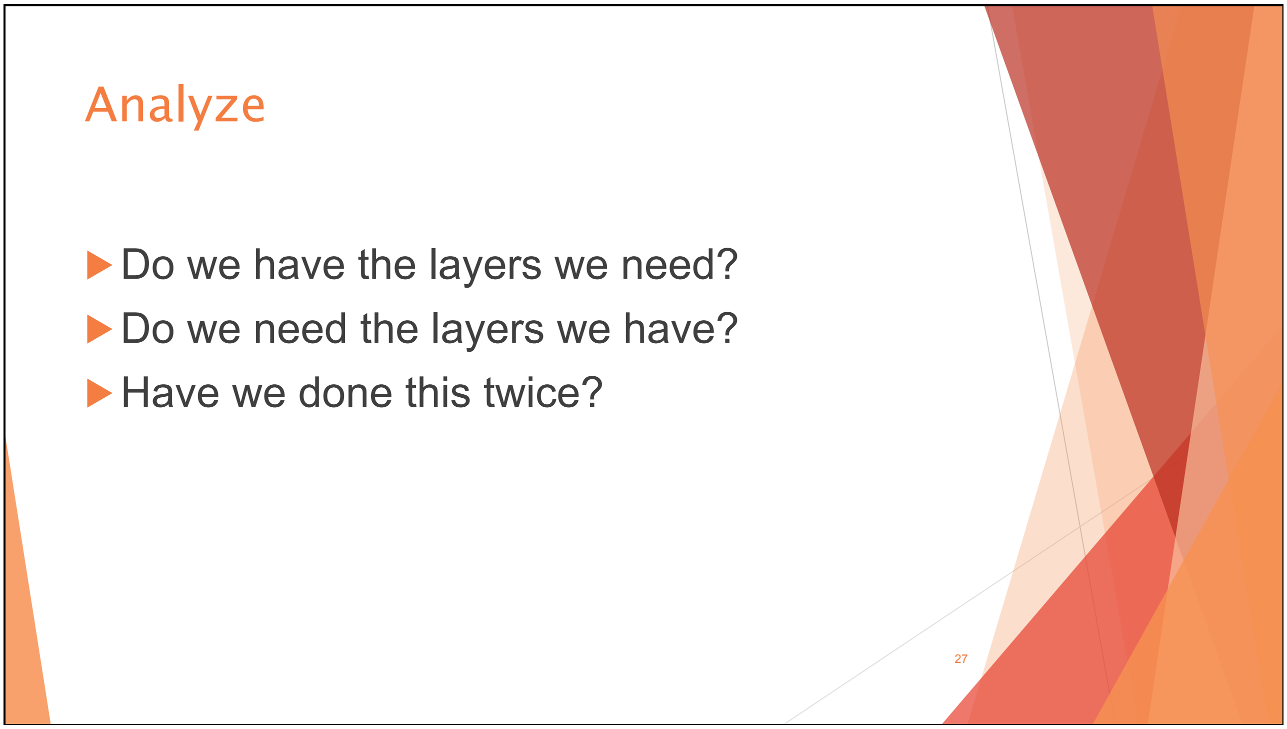
## Discovery: Case Study

- Local dev setup took 3 days
  - Mock service dependencies
- Tests (manual) took 15+ min to boot
  - Application bootstrap code
  - Test harnesses & test code



## Discovery: Case Study, cont.

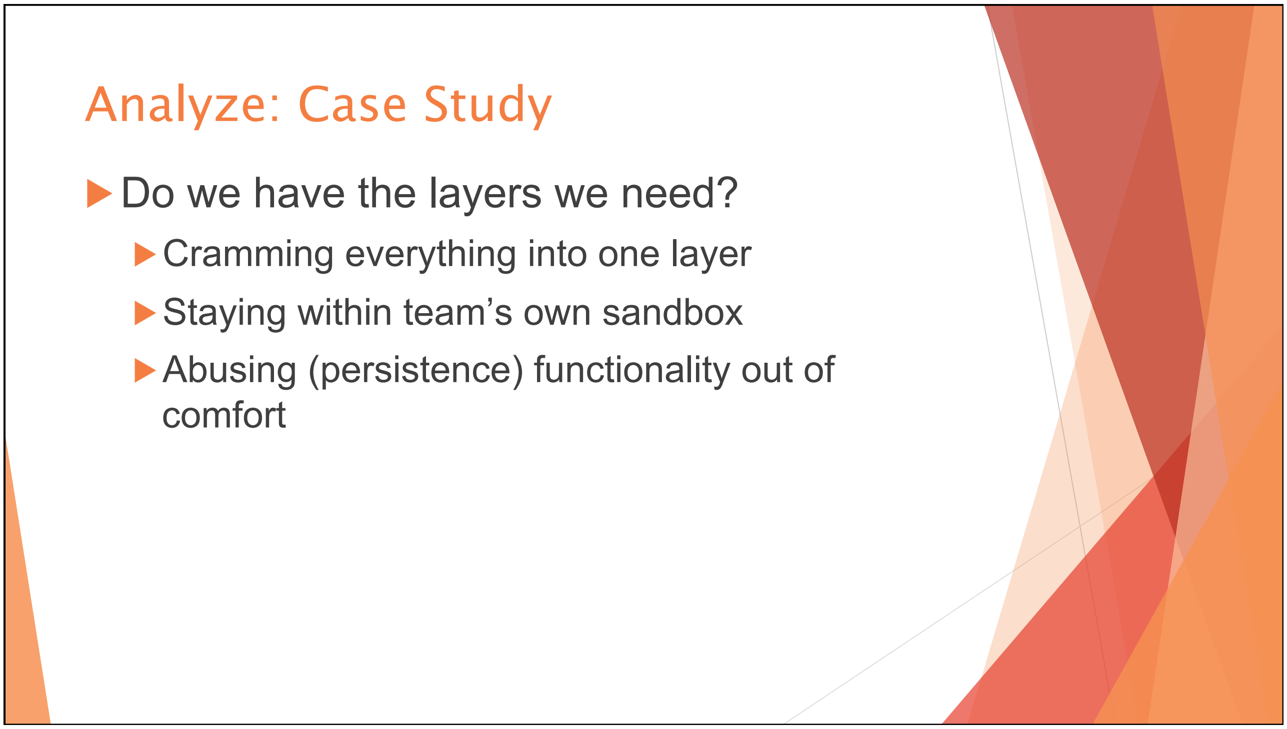
- Reproduction took 100+ steps
  - Persistence and state logic
- System required millisecond time sync
  - Consistency architecture problem
  - (15ms tick granularity)



## Analyze

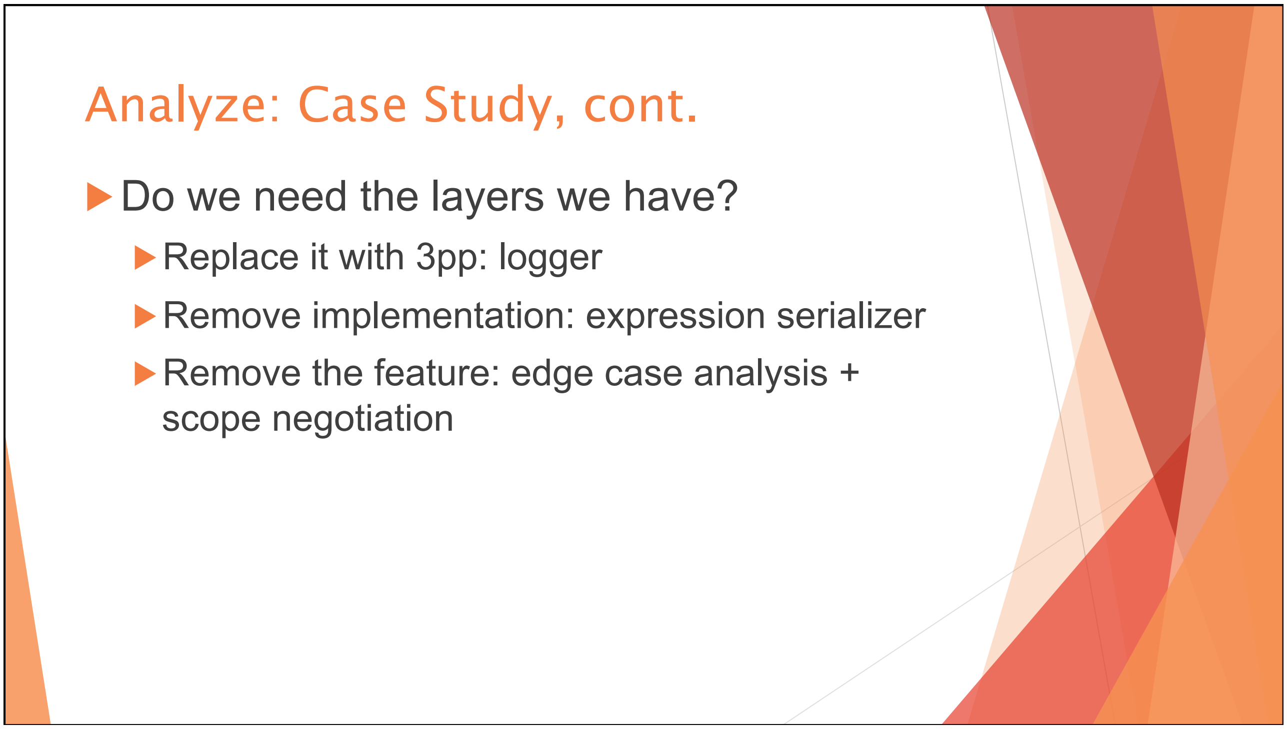
- Do we have the layers we need?
- Do we need the layers we have?
- Have we done this twice?

27



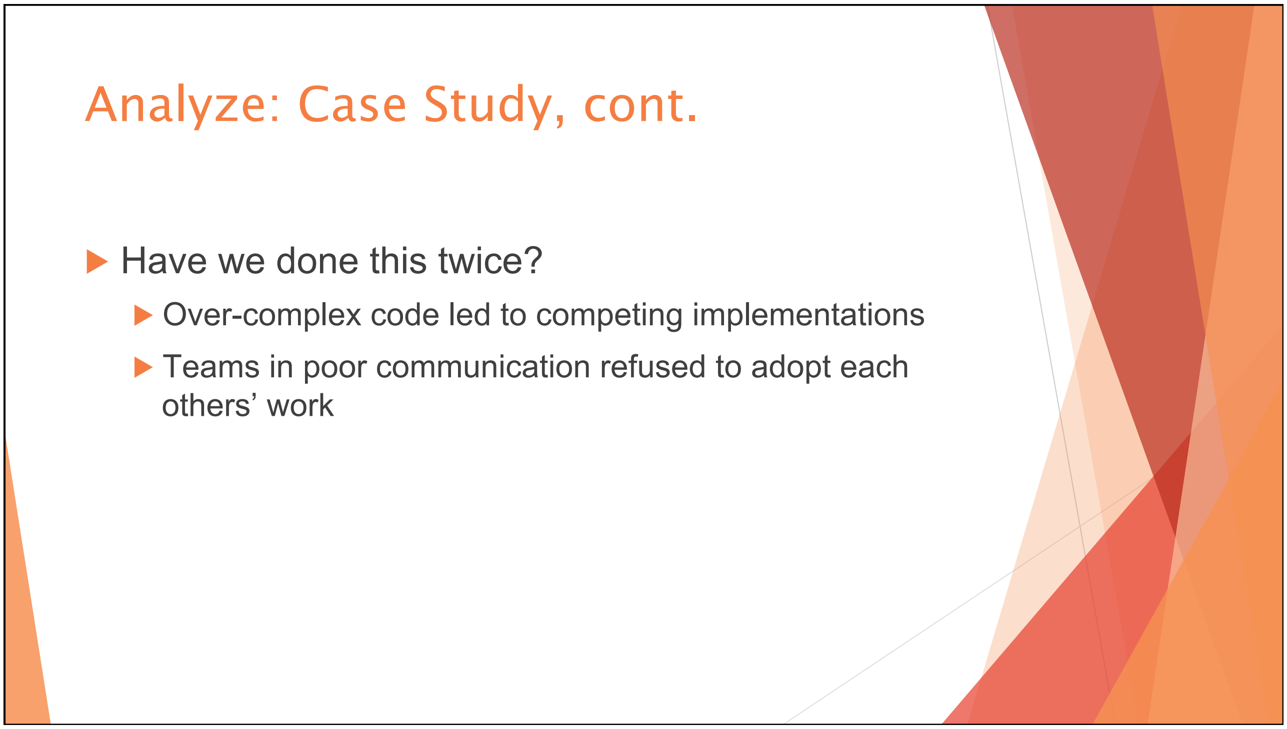
## Analyze: Case Study

- Do we have the layers we need?
  - Cramming everything into one layer
  - Staying within team’s own sandbox
  - Abusing (persistence) functionality out of comfort



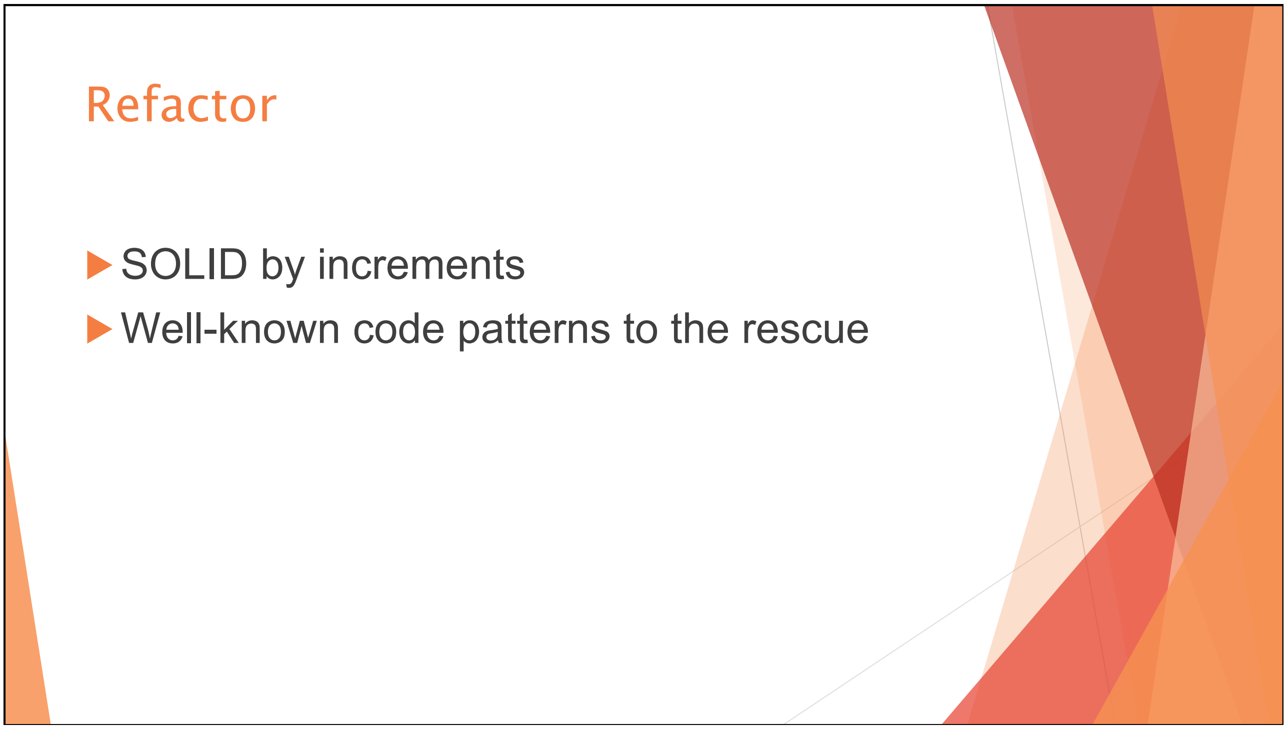
## Analyze: Case Study, cont.

- Do we need the layers we have?
  - Replace it with 3pp: logger
  - Remove implementation: expression serializer
  - Remove the feature: edge case analysis + scope negotiation



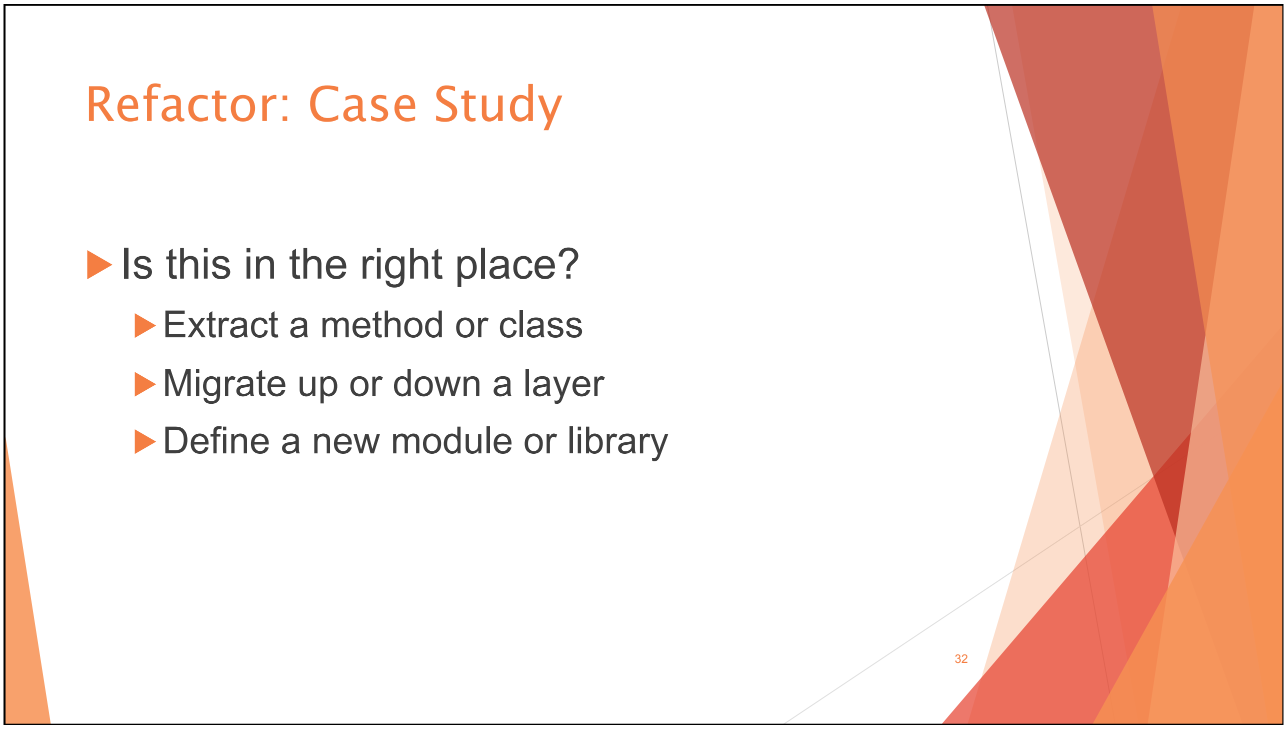
## Analyze: Case Study, cont.

- Have we done this twice?
  - Over-complex code led to competing implementations
  - Teams in poor communication refused to adopt each others’ work



## Refactor

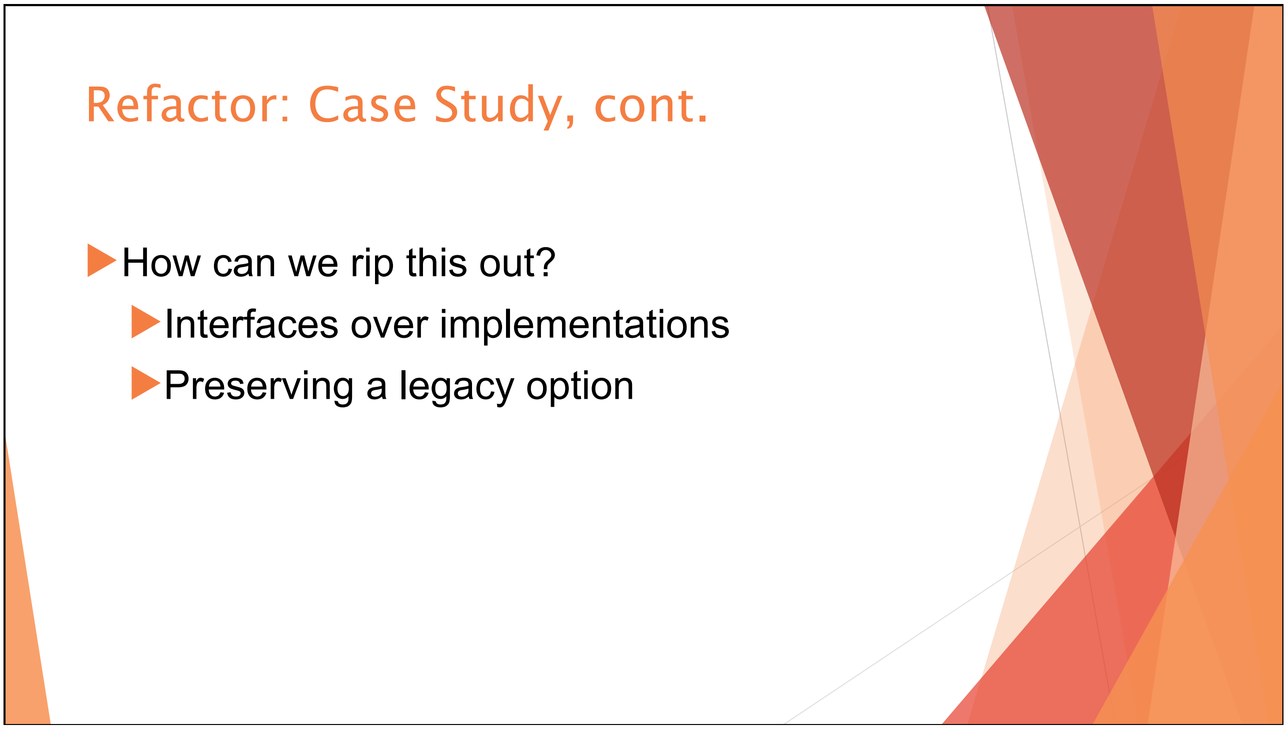
- SOLID by increments
- Well-known code patterns to the rescue



## Refactor: Case Study

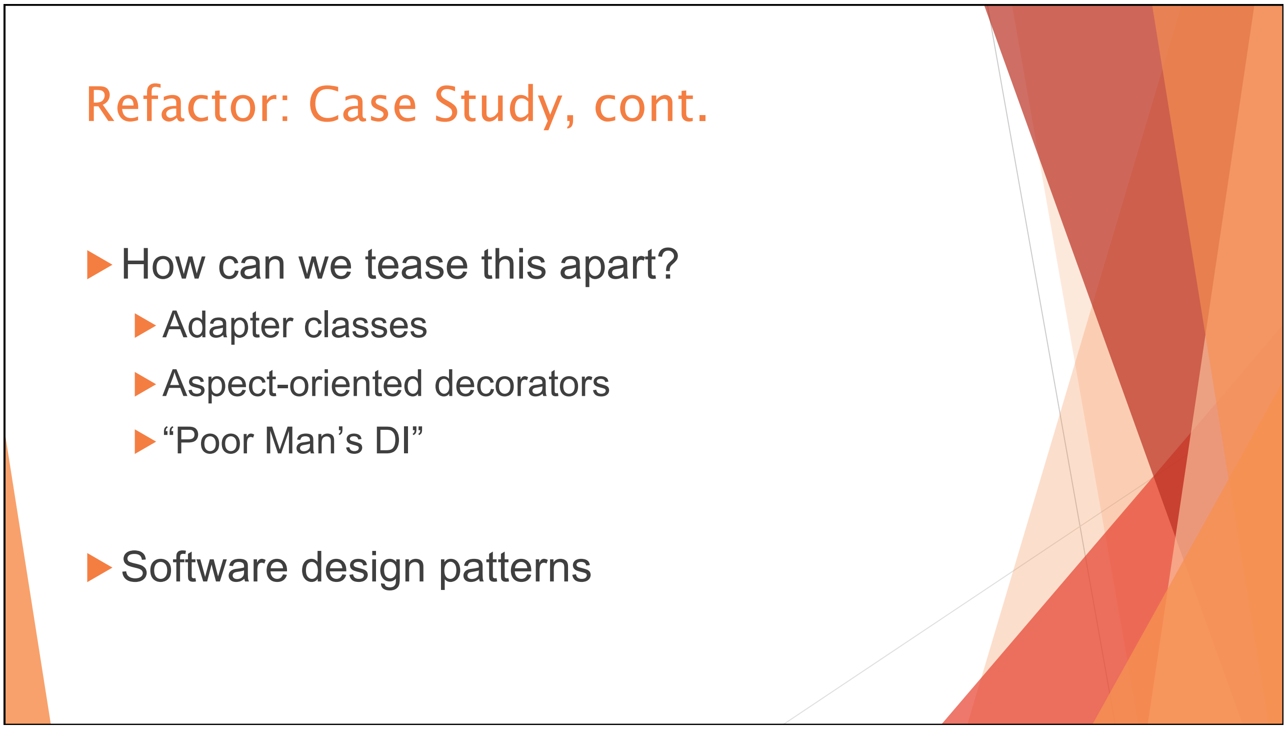
- Is this in the right place?
  - Extract a method or class
  - Migrate up or down a layer
  - Define a new module or library

32



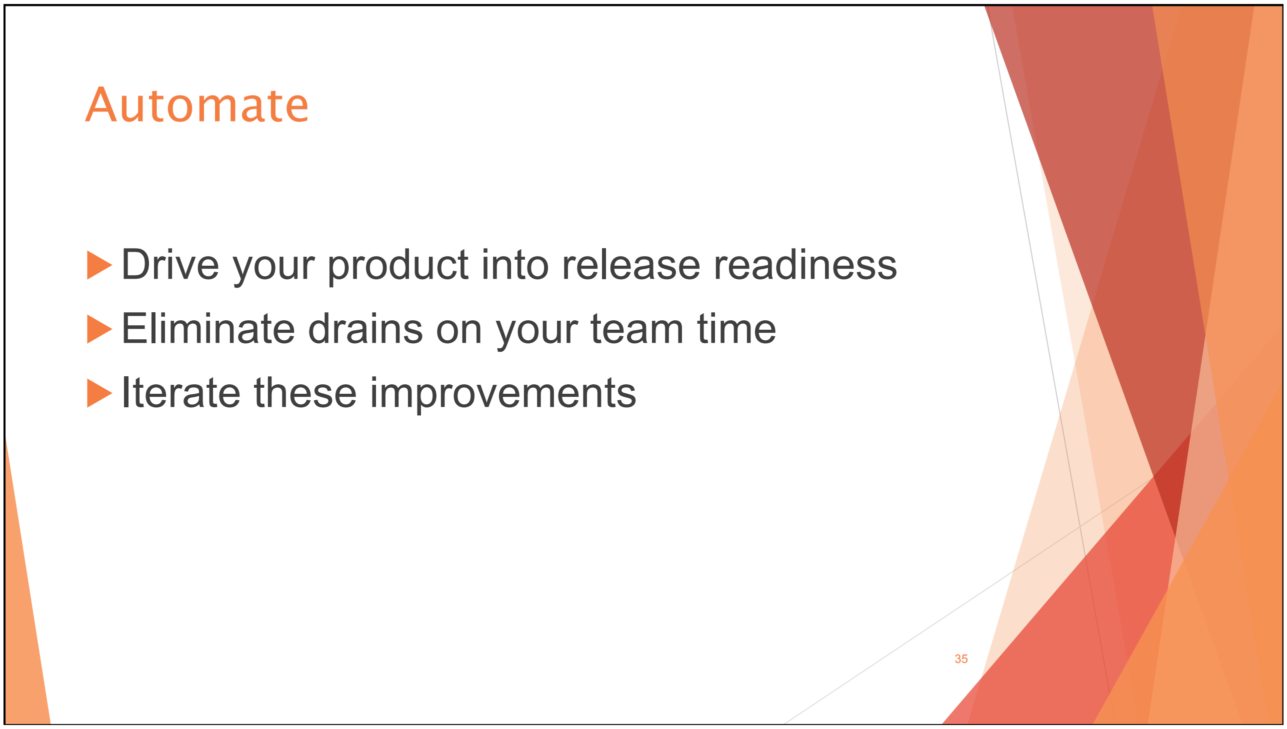
## Refactor: Case Study, cont.

- How can we rip this out?
  - Interfaces over implementations
  - Preserving a legacy option



## Refactor: Case Study, cont.

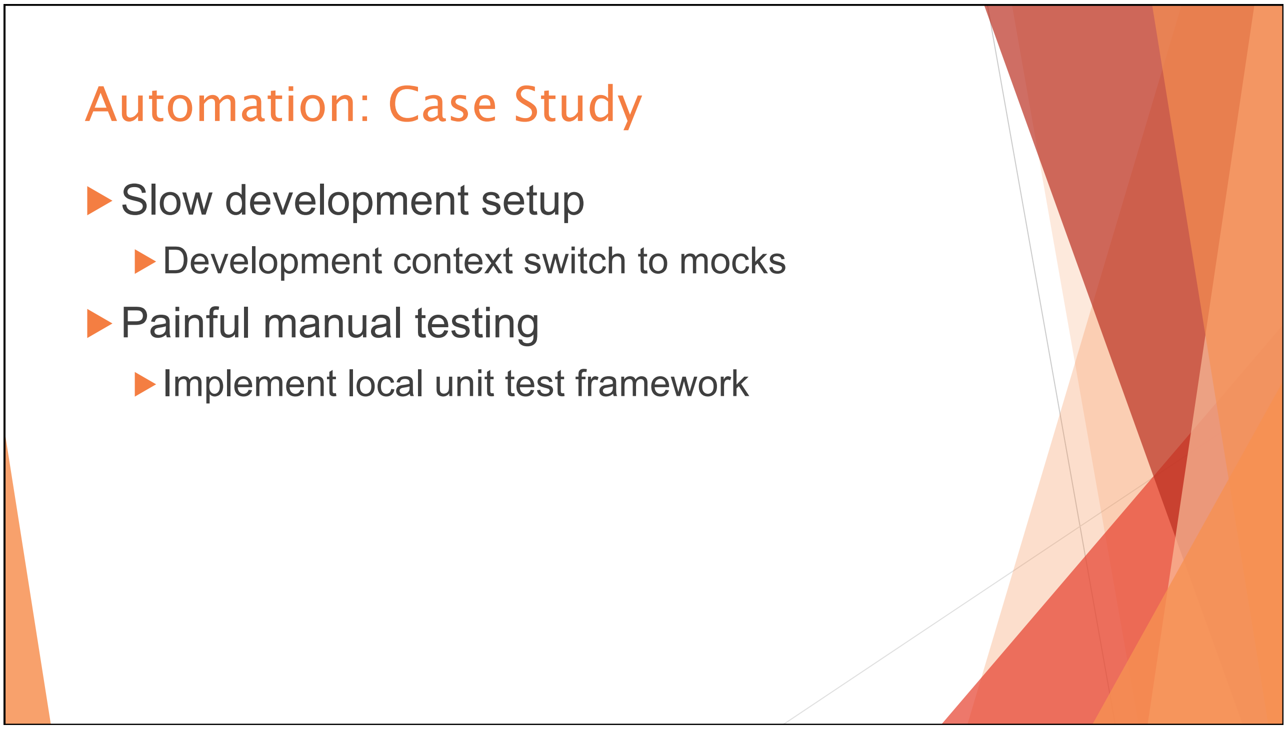
- How can we tease this apart?
  - Adapter classes
  - Aspect-oriented decorators
  - “Poor Man’s DI”
- Software design patterns



## Automate

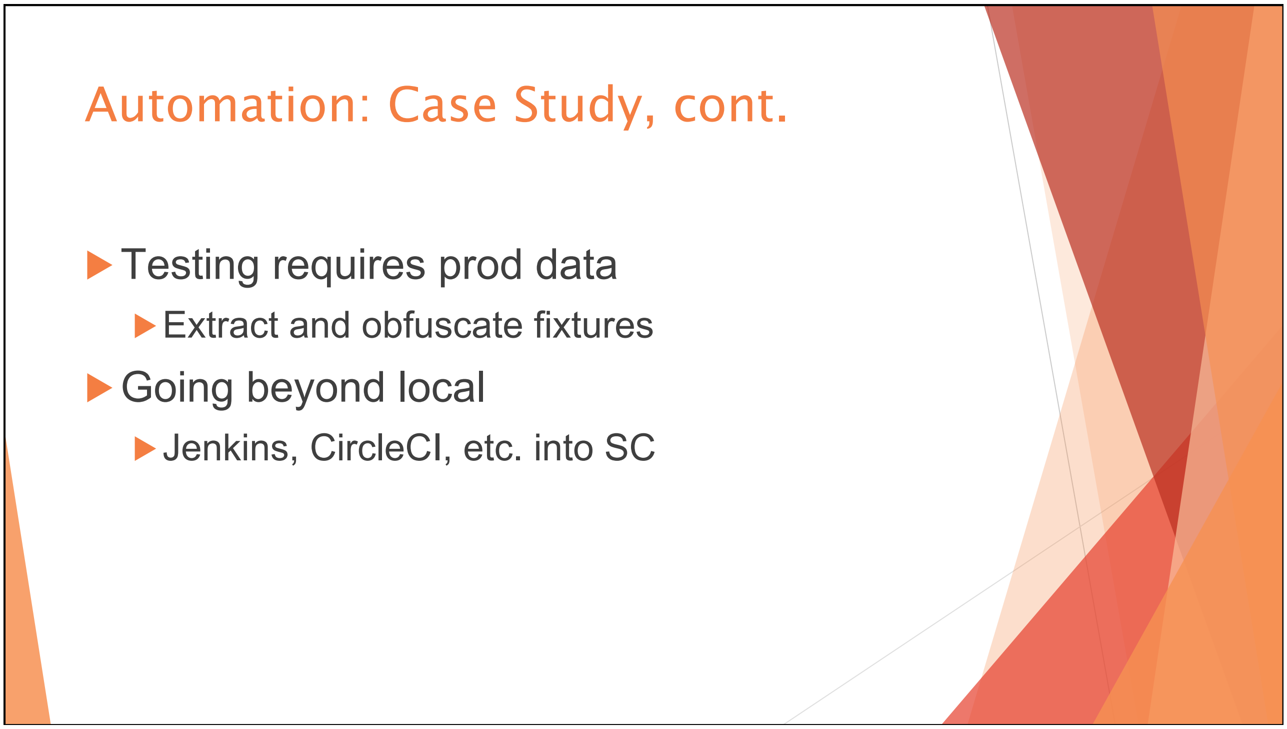
- Drive your product into release readiness
- Eliminate drains on your team time
- Iterate these improvements

35



## Automation: Case Study

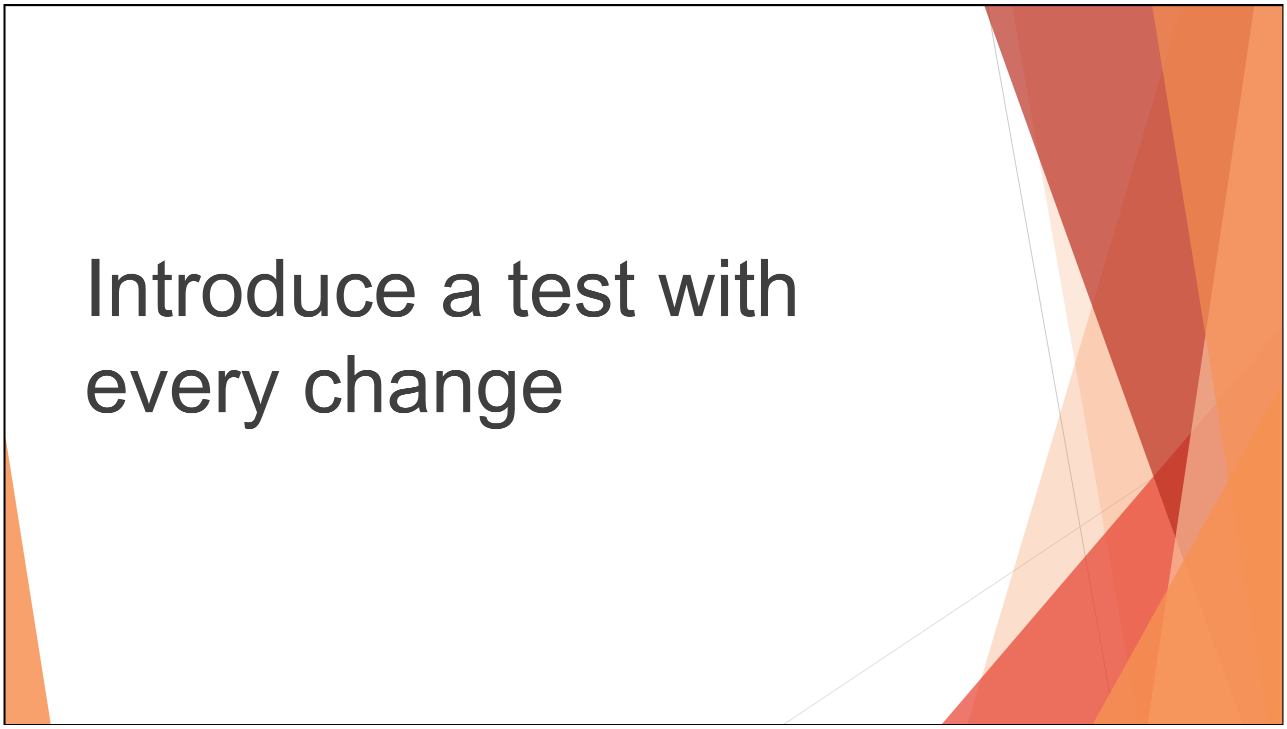
- Slow development setup
  - Development context switch to mocks
- Painful manual testing
  - Implement local unit test framework



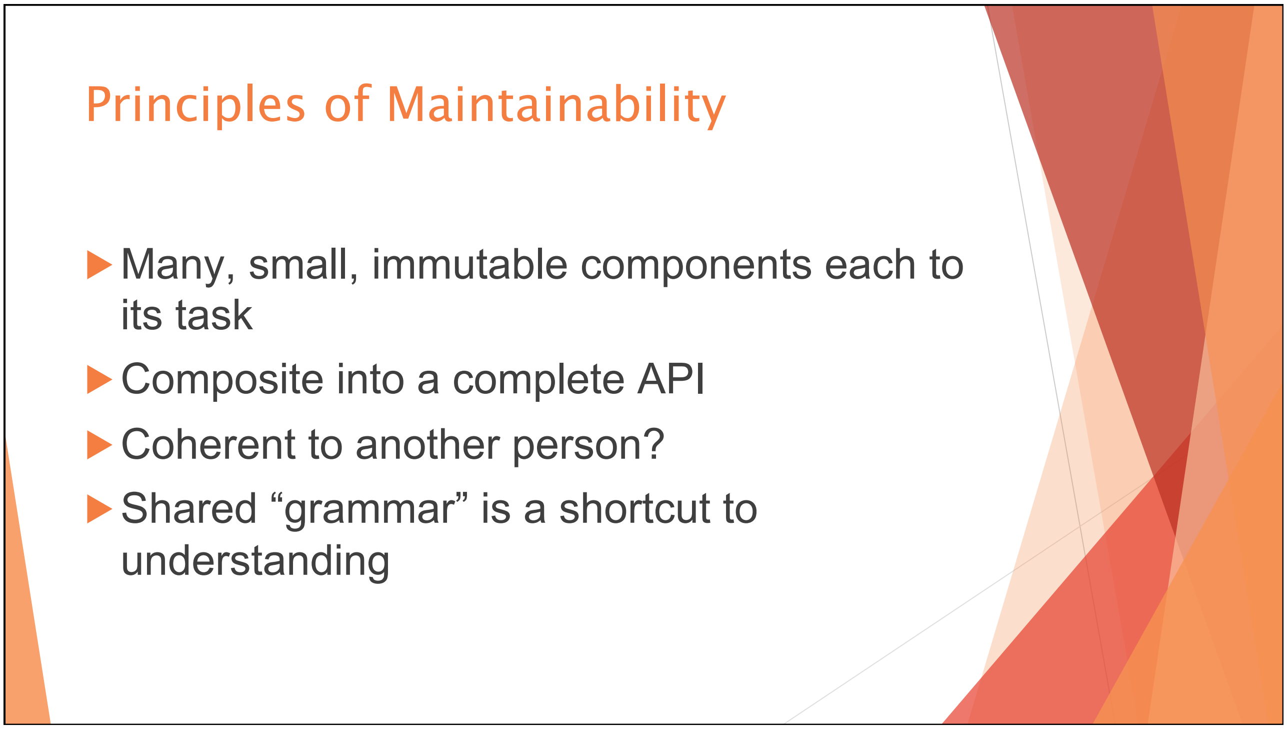
## Automation: Case Study, cont.

- Testing requires prod data
  - Extract and obfuscate fixtures
- Going beyond local
  - Jenkins, CircleCI, etc. into SC

## Principles and Practices

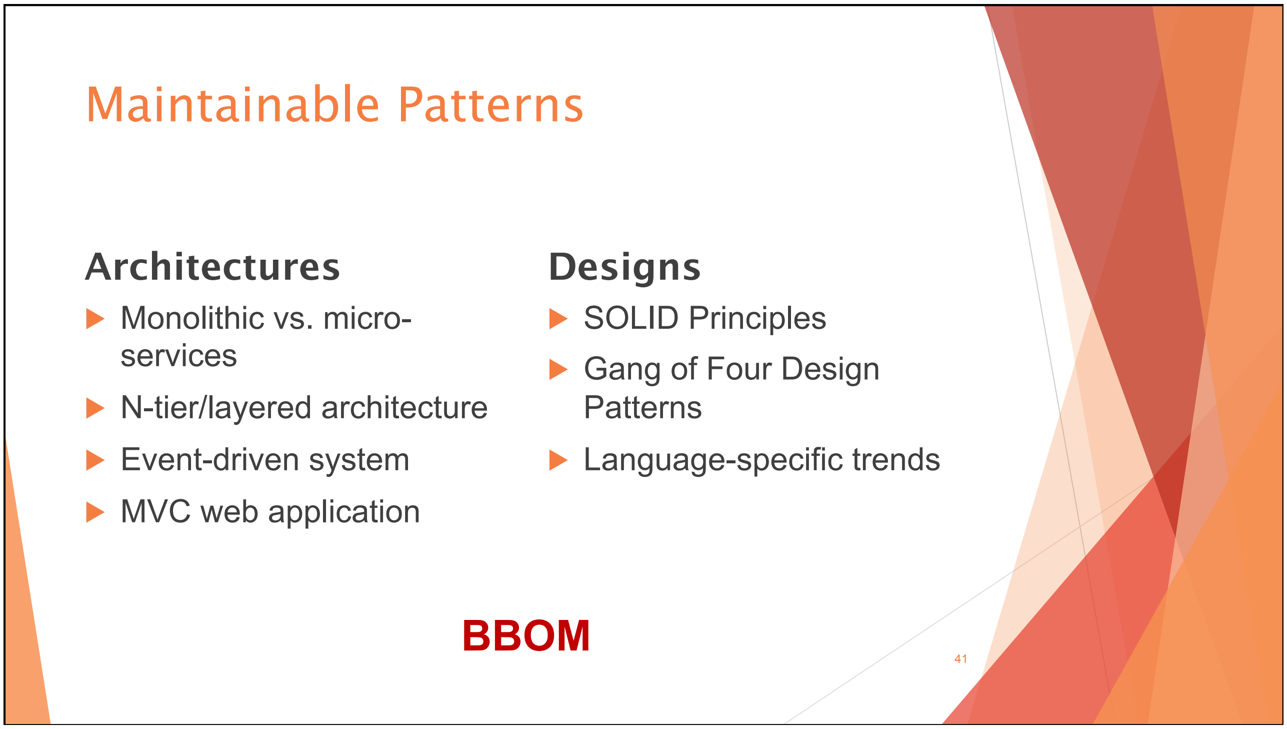


# Introduce a test with every change



## Principles of Maintainability

- Many, small, immutable components each to its task
- Composite into a complete API
- Coherent to another person?
- Shared “grammar” is a shortcut to understanding



# Maintainable Patterns

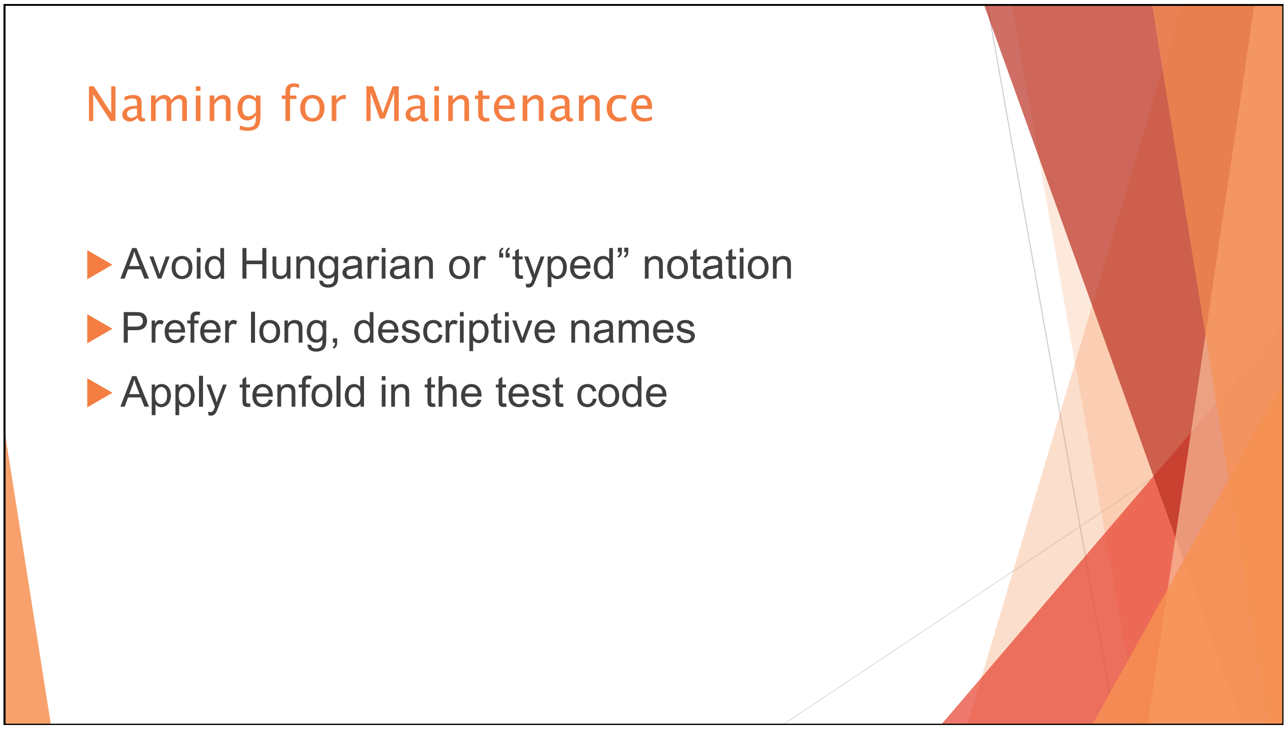
### Architectures

- Monolithic vs. micro-services
- N-tier/layered architecture
- Event-driven system
- MVC web application

### Designs

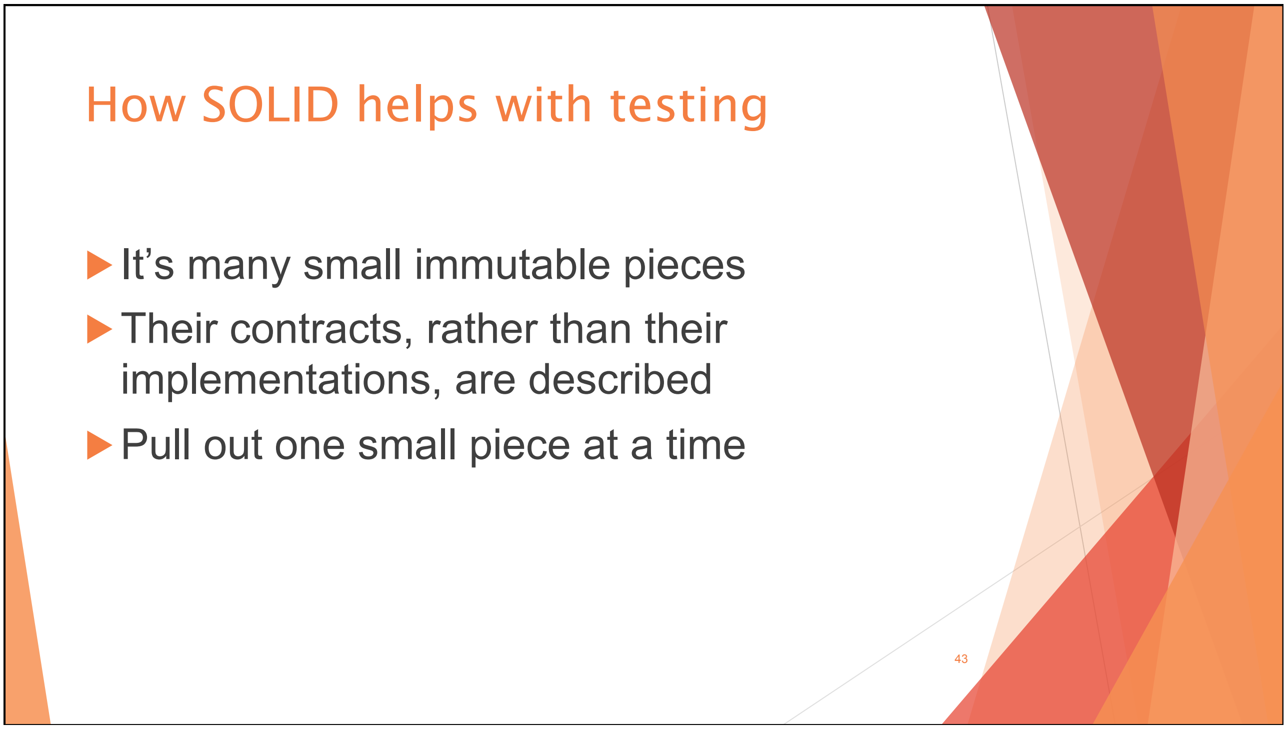
- SOLID Principles
- Gang of Four Design Patterns
- Language-specific trends

**BBOM**



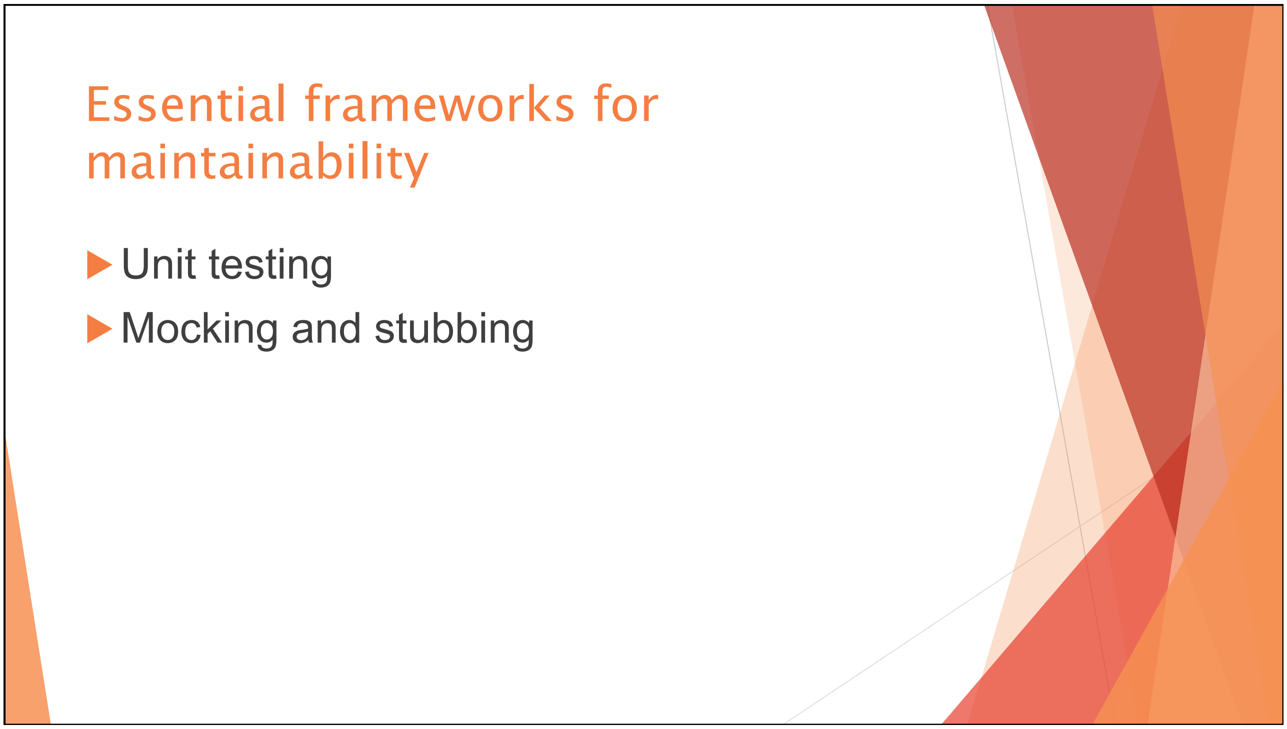
# Naming for Maintenance

- Avoid Hungarian or “typed” notation
- Prefer long, descriptive names
- Apply tenfold in the test code



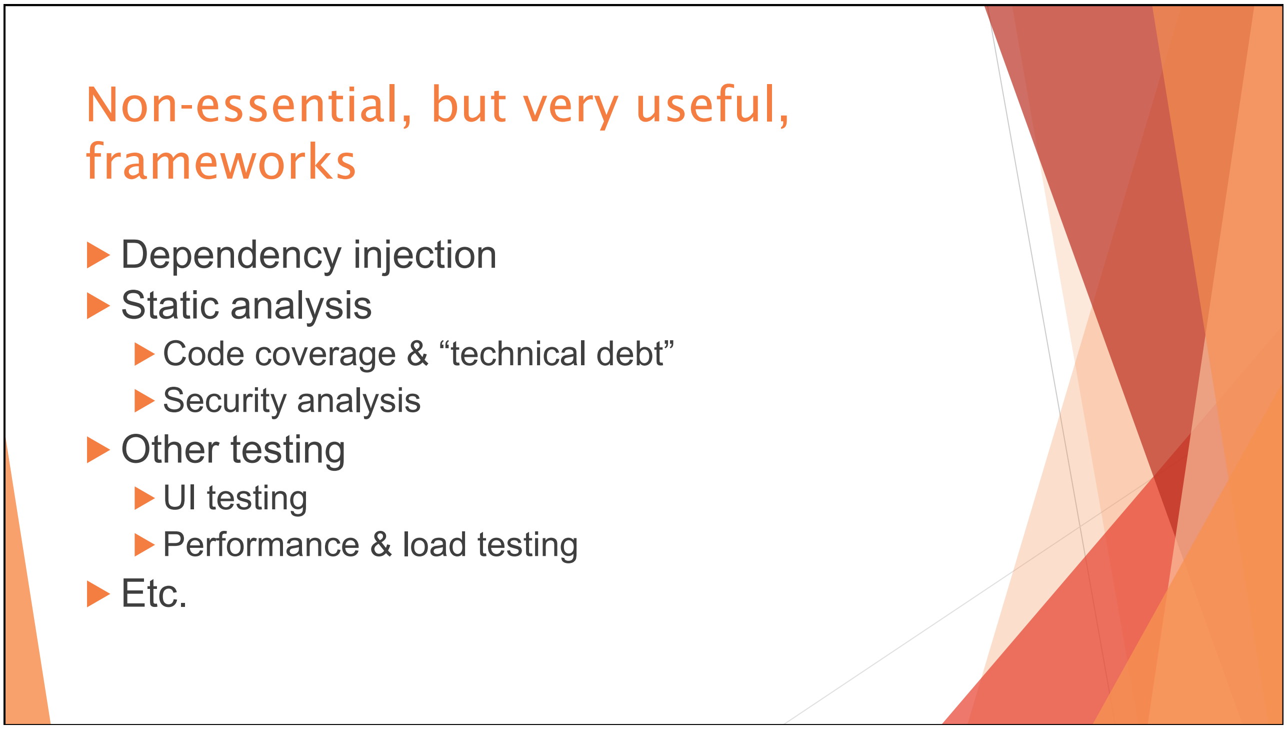
## How SOLID helps with testing

- It’s many small immutable pieces
- Their contracts, rather than their implementations, are described
- Pull out one small piece at a time



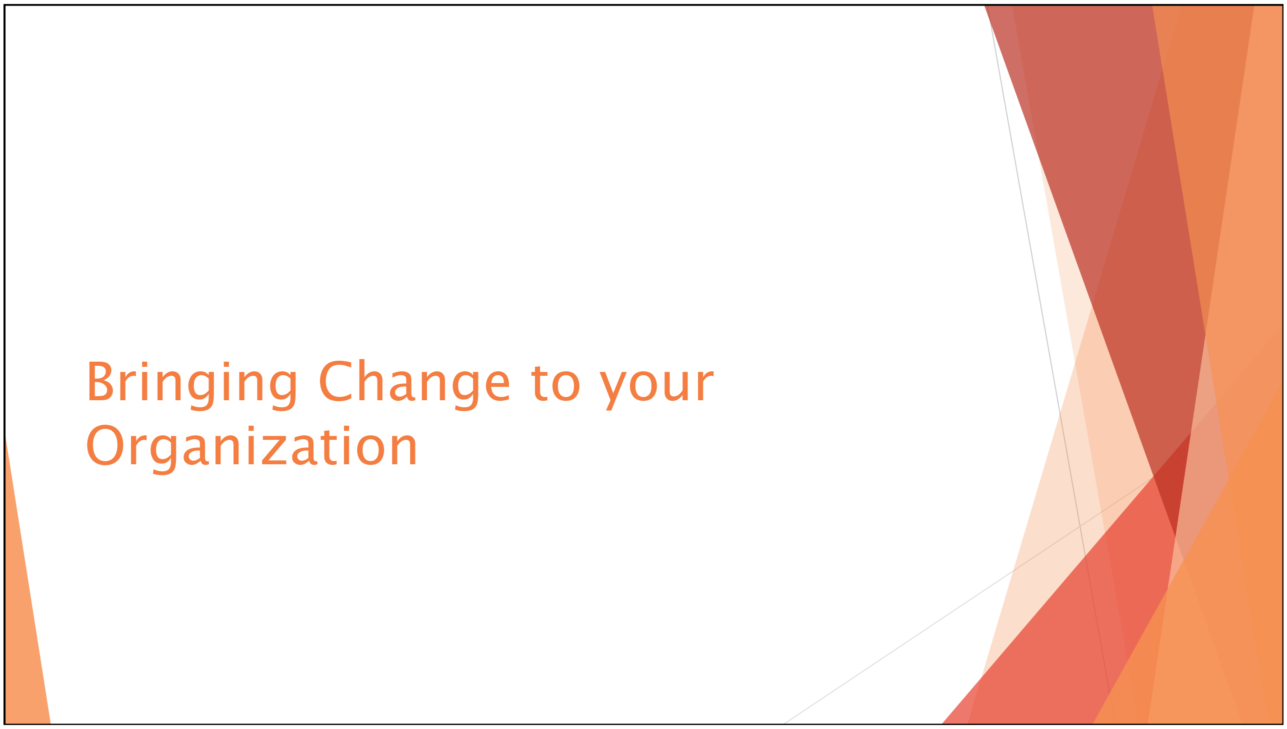
## Essential frameworks for maintainability

- Unit testing
- Mocking and stubbing

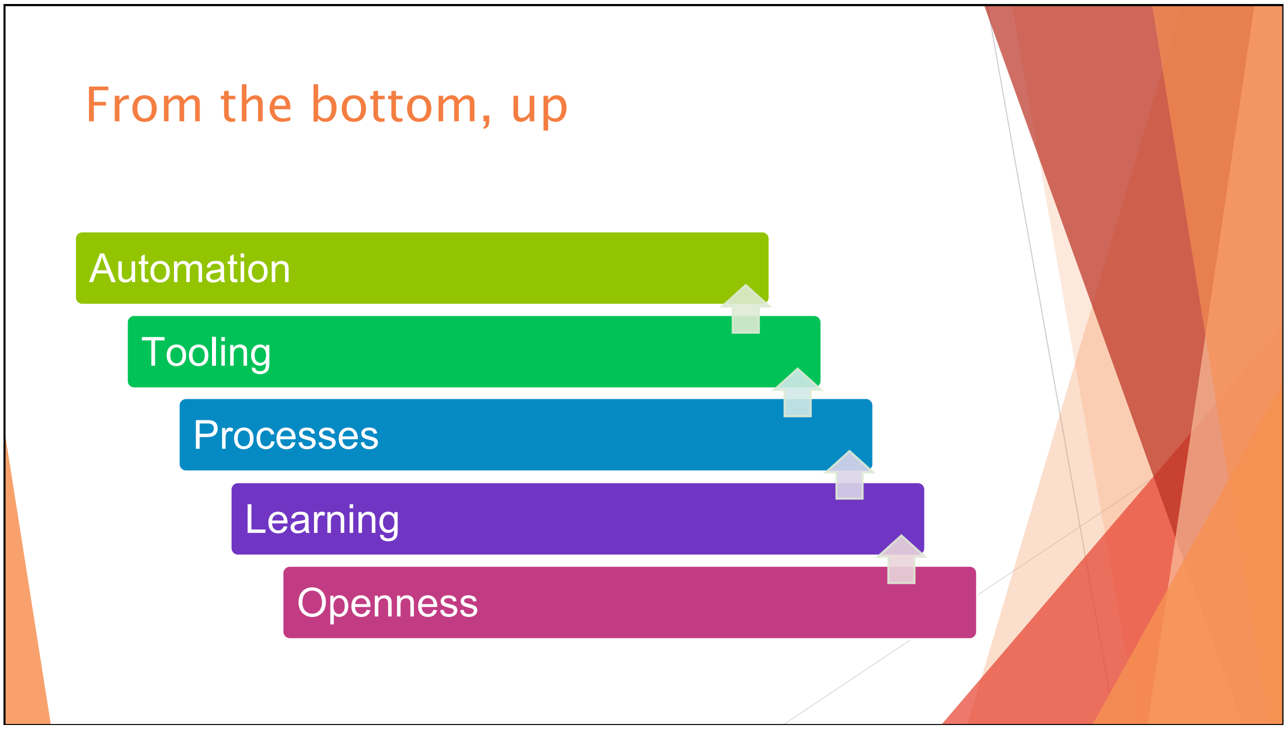


## Non-essential, but very useful, frameworks

- Dependency injection
- Static analysis
  - Code coverage & “technical debt”
  - Security analysis
- Other testing
  - UI testing
  - Performance & load testing
- Etc.



## Bringing Change to your Organization



## From the bottom, up

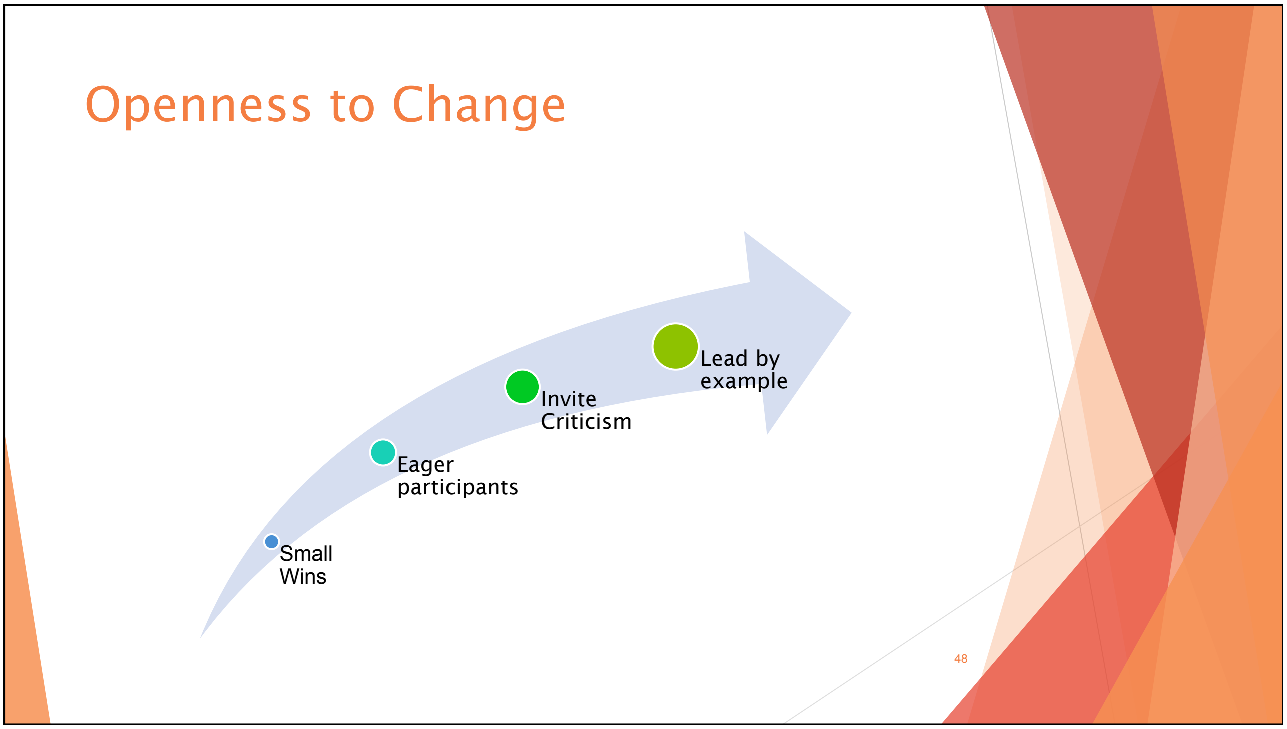
Automation

Tooling

Processes

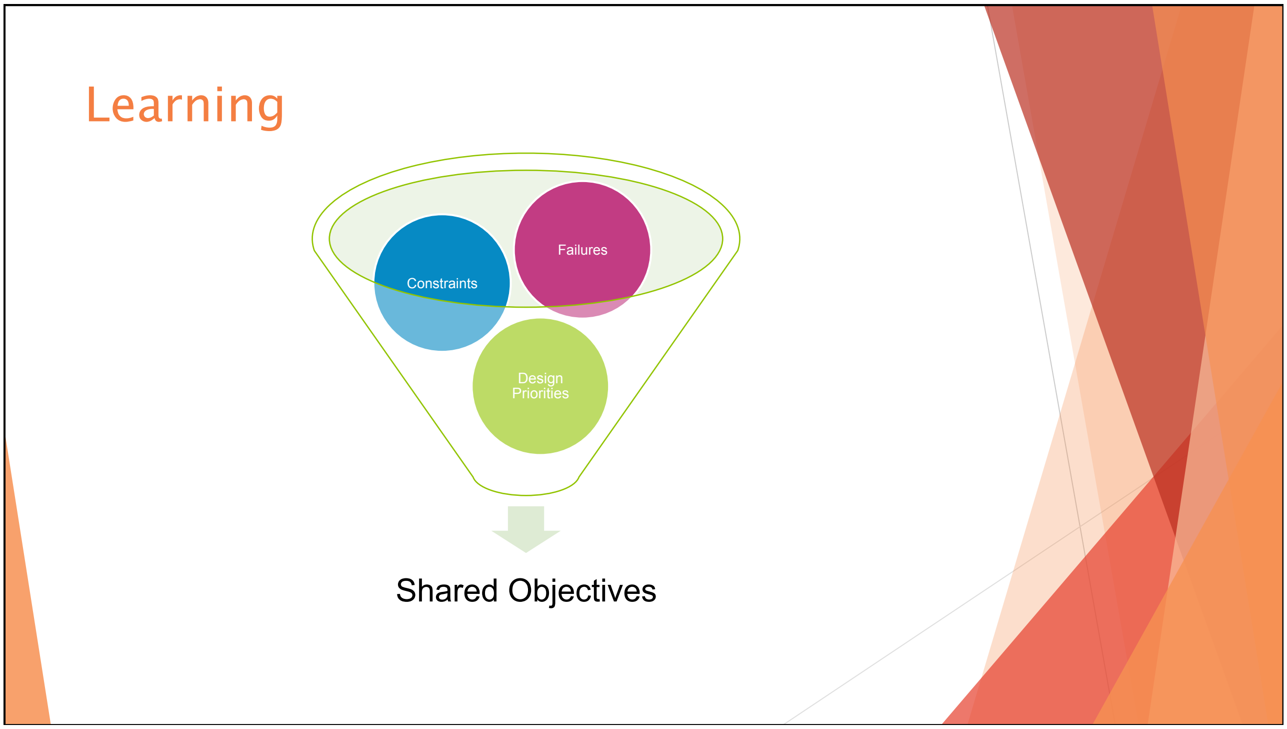
Learning

Openness



## Openness to Change

48



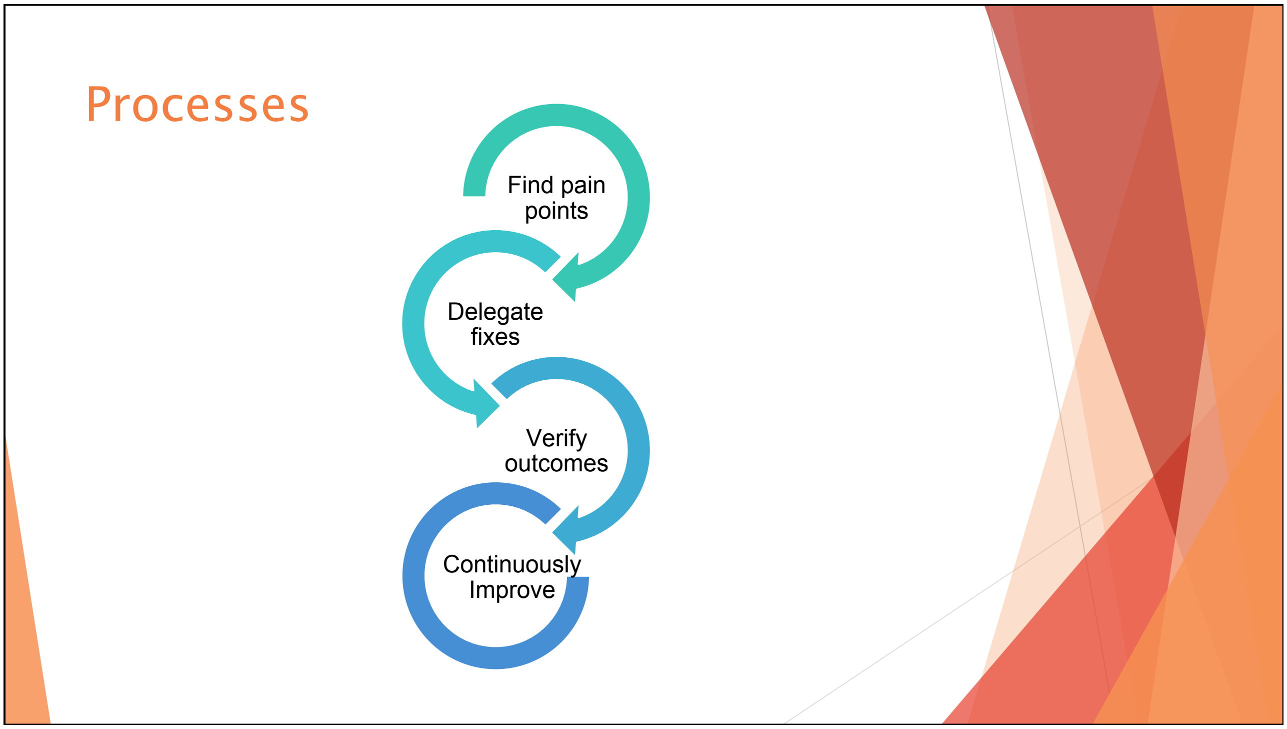
## Learning

Constraints

Failures

Design Priorities

Shared Objectives



## Processes

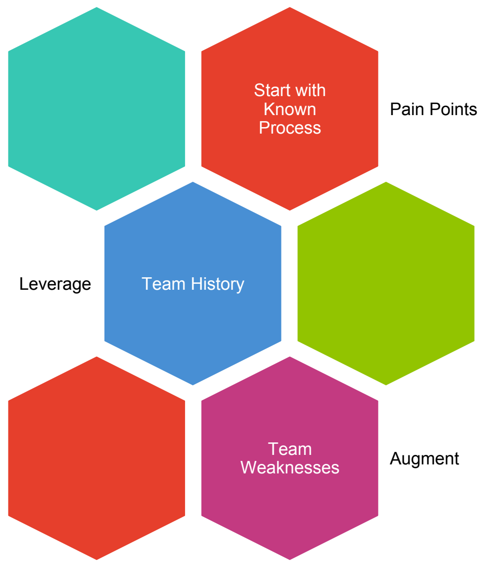
Find pain points

Delegate fixes

Verify outcomes

Continuously Improve

# Tooling



Start with Known Process

Pain Points

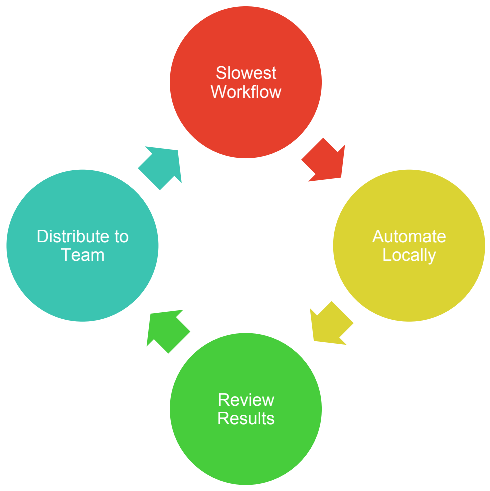
Leverage

Team History

Team Weaknesses

Augment

# Automation



Slowest Workflow

Automate Locally

Review Results

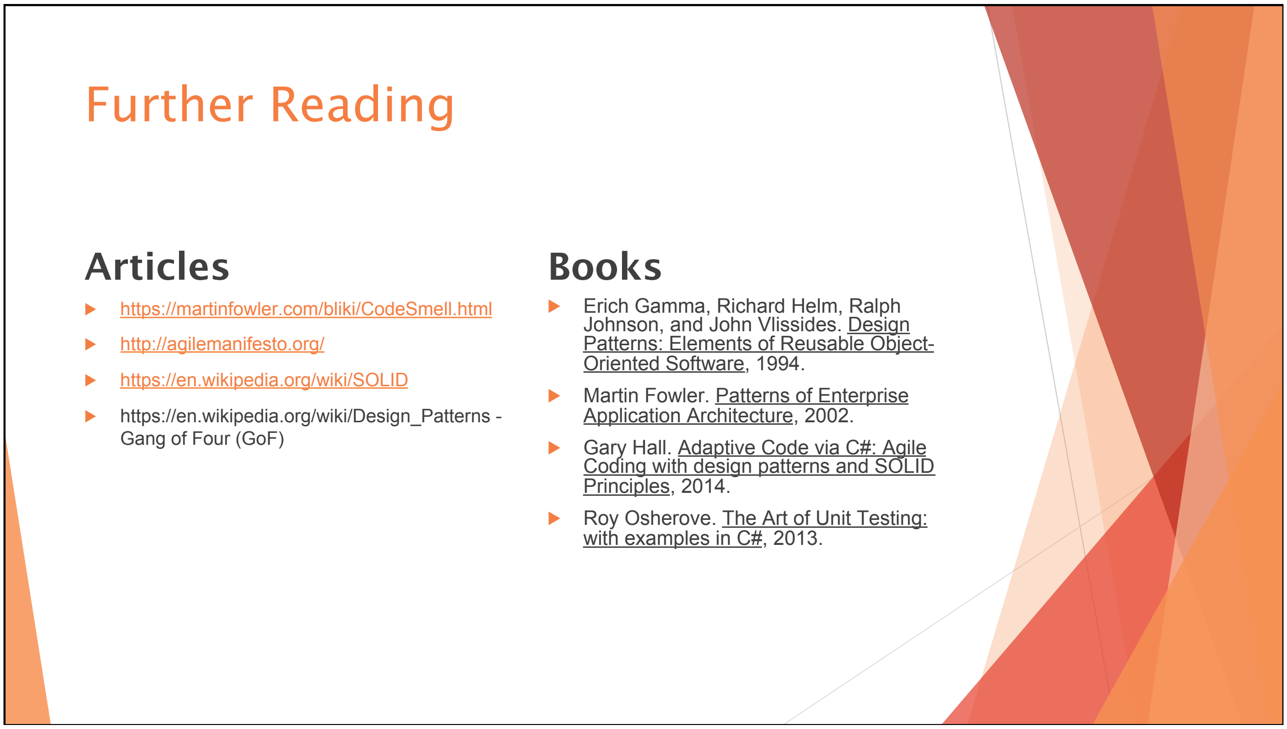
Distribute to Team

# Q&A

## Contact Information

- @jhsolor
- in/peachpie
- @peachpie
- jhsolor

- @ProcoreTech
- @ProcoreJobs
- procore.com/jobs/openings



# Further Reading

## Articles

- https://martinfowler.com/bliki/CodeSmell.html
- http://agilemanifesto.org/
- https://en.wikipedia.org/wiki/SOLID
- https://en.wikipedia.org/wiki/Design_Patterns - Gang of Four (GoF)

## Books

- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, 1994.
- Martin Fowler. Patterns of Enterprise Application Architecture, 2002.
- Gary Hall. Adaptive Code via C#: Agile Coding with design patterns and SOLID Principles, 2014.
- Roy Osherove. The Art of Unit Testing: with examples in C#, 2013.