CS 410: Recap

Reid Holmes

Source: [*Gamma et all, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995*]

- Program an Interface not an Implementation
- Favor Composition Versus Inheritance
- Find what varies and encapsulate it

Source: [*R. Martin, "Agile Software Development, Principles, Patterns, and Practices", Prentice-Hall, 2002*]

- Dependency-Inversion Principle
- Liskov Substitution Principle
- Open-Closed Principle
- Interface-Segregation Principle
- Reuse/Release Equivalency Principle
- Common Closure Principle
- Common Reuse Principle
- Acyclic Dependencies Principle
- Stable Dependencies Principle
- Stable Abstraction Principle

Source: [*Larman, "Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development", Prentice-Hall, 20*

- Design principles are codified in the GRASP Pattern
- GRASP (Pattern of General Principles in Assigning Responsabilities)
- Assign a responsibility to the information expert
- Assign a responsability so that coupling remains low
- Assign a responsability so that cohesion remains high
- Assign responsabilities using polymorphic operations
- Assign a highly cohesive set of responsabilities to an artificial class that does not represent anything in the problem domain (when you want to
- Don't talk to strangers (Law of Demeter)

Source: [*Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", Commununication of ACM, 1972*]

- Information Hiding
- Modularity

Source: [*Hunt, Thomas, "The Pragmatic Programmer: From Journeyman to Master", Addison-Wesley, 1999*]

- DRY – Don't Repeat yourself
- Make it easy to reuse
- Design for Orthogonality
- Eliminate effects between unrelated things
- Program close to the problem domain
- Minimize Coupling between Modules
- Design Using Services
- Always Design for Concurrency
- Abstractions Live Longer than details

Source: [*Lieberherr,Holland, "Assuring Good Style for Object-Oriented Programs", IEEE Software, September 1989*]

- Law of Demeter

Source: [*Raymond, "Art of Unix Programming", Addison-Wesley, 2003*]

Pragmatic Programmer:

Eliminate Effects Between Unrelated Things –
design components that are:
self-contained,
independent,
and have a single, well-defined purpose

What is Software Engineering?

> The establishment and application of scientific, economic, social, and practical knowledge in order to invent, design, build, maintain, research, and improve software that is reliable and works efficiently on real machines.
>
> **— WIKIPEDIA MASHUP**

[Tailor et al.]

Essential Difficulties

- Abstraction alone cannot help.
  - Complexity
    - Grows non-linearly with program size.
  - Conformity
    - System is dependent on its environment.
  - Changeability
    - Perception that software is easily modified.
  - Intangibility
    - Not constrained by physical laws.

Abstraction

- Complex problems can be approached by abstracting away unnecessary detail
- Focus on the key issues while eliding extraneous detail (some of these details will be pertinent during more detailed design activities)
- In software two classes of abstraction dominate:
  - Control abstraction
    - (e.g., structured programming)
  - Data abstraction
    - (e.g., abstract data types)

Specifications

- A specification:
  - Connects customer and engineer
  - Ensures parts of the implementation work together
  - Defines the correctness of the implementation
- Therefore, everyone must understand the spec
  - Designers, developers, testers, managers, ops, customers…
- Good specifications are essential for a project to be successful

Elicitation

- Required functionality: what the software should do
  - Record keeping, data computations / transformations, process control, query processing, commands to hardware devices, etc.
- Quality attributes: desired characteristics (NFPs)
  - Performance, efficiency, safety, security, usability, maintainability, reliability, robustness, availability
- Design constraints: customer-specified limits
  - Mandated hardware components, mandated adjacent systems, resource constraints, mandated development process, budget
- Environmental assumptions: assumed context
  - Working status of hardware / software components, assumptions about inputs (data format, rate of input, number of users), operating conditions
- Preferences
  - Priority rankings of requirements

Architectural representations

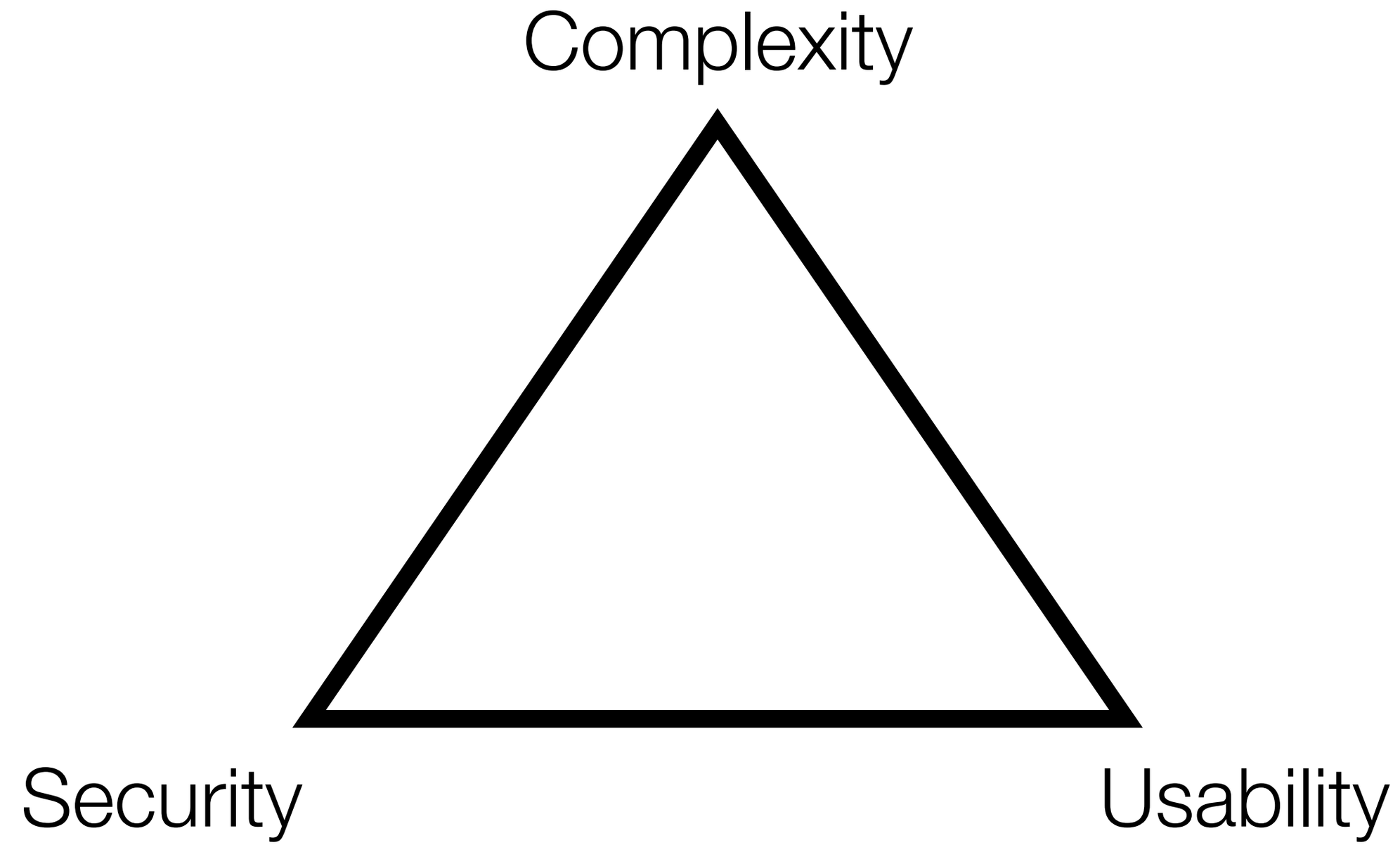
- Software architecture is fundamentally about facilitating technical communication between project stakeholders
- An opaque architecture has no value as it will not be adequately understood
- Properties of representations:
  - Ambiguity: Open to more than one interpretation?
  - Accuracy: Correct within tolerances
  - Precision: Consistent but not necessarily correct

[Tailor et al.]

NFPs

- NFPs are constraints on the manner in which the system implements and delivers its functionality.
  - E.g.,
    - Efficiency
    - Complexity
    - Scalability
    - Heterogeneity
    - Adaptability
    - Security
    - Dependability
    - Testability
    - Usability
    - Performance

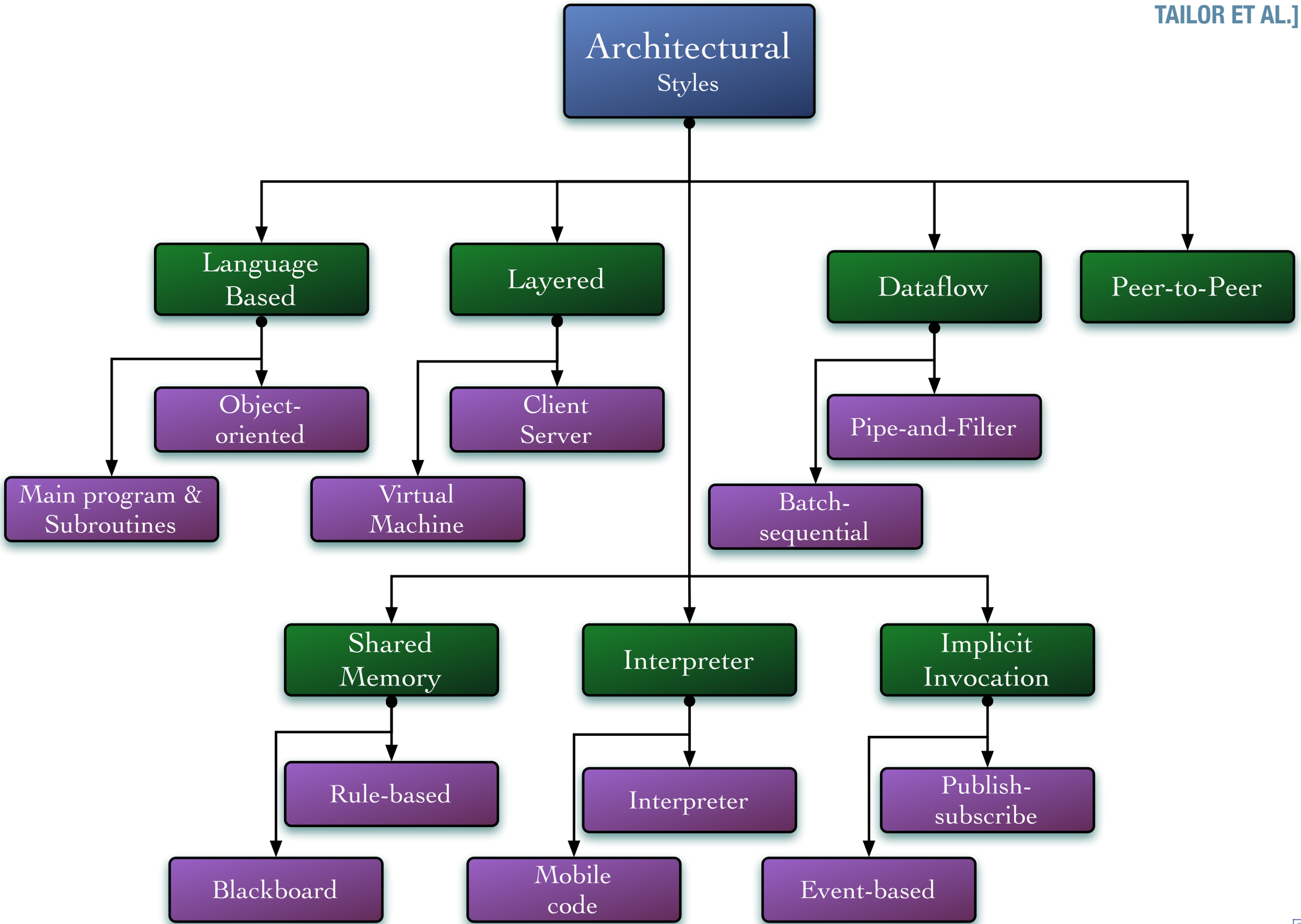
NFP Triangle



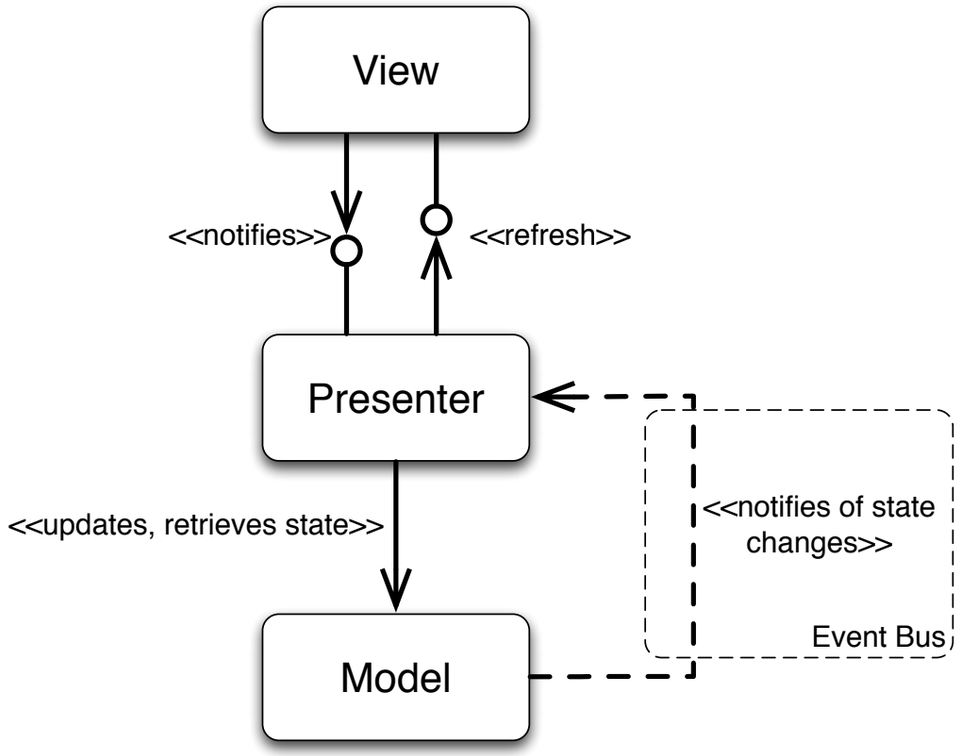
Architectural styles

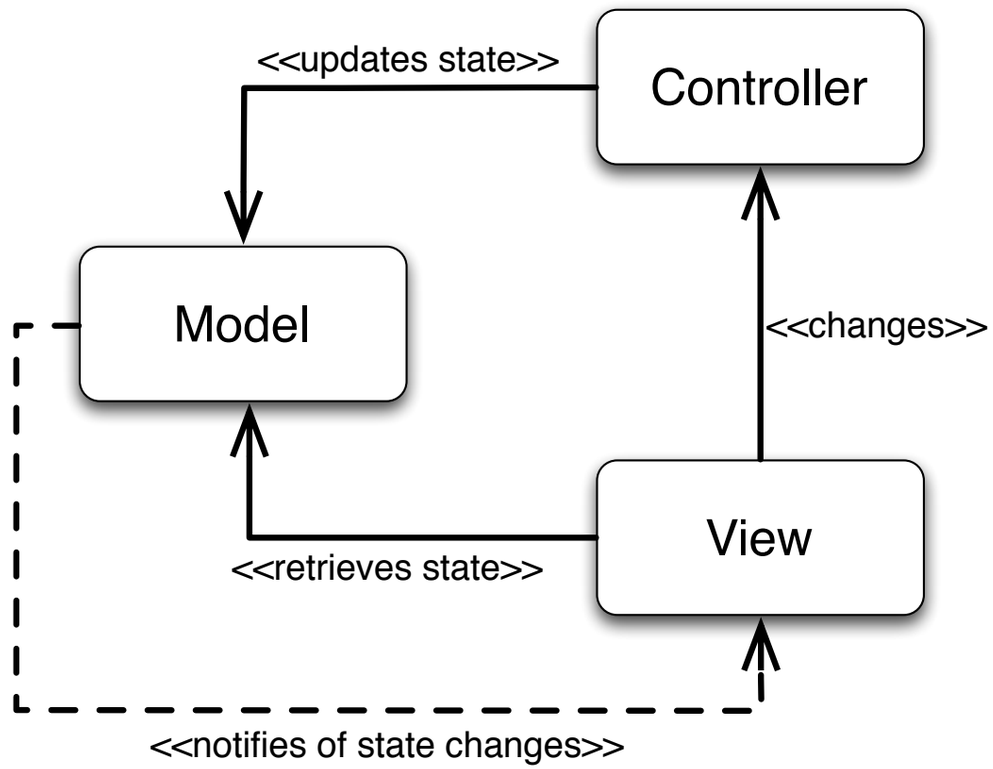
- Some design choices are better than others
  - Experience can guide us towards beneficial sets of choices (patterns) that have positive properties
- An architectural style is a named collection of architectural design decisions that:
  - Are applicable to a given context
  - Constrain design decisions
  - Elicit beneficial qualities in resulting systems

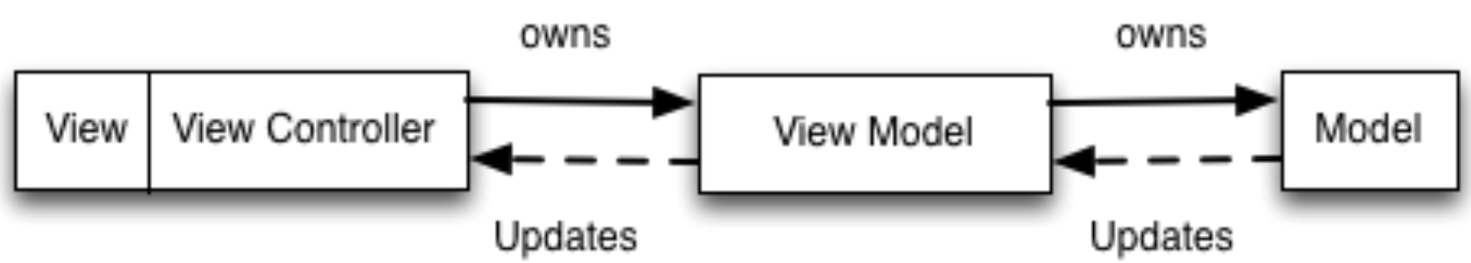
[TOPOLOGY FROM TAILOR ET AL.]



MVC/MVP/MVVM


View
<<notifies>>
<<refresh>>
Presenter
<<updates, retrieves state>>
Model
<<notifies of state changes>>
Event Bus


<<updates state>>
Controller
Model
<<changes>>
<<retrieves state>>
View
<<notifies of state changes>>

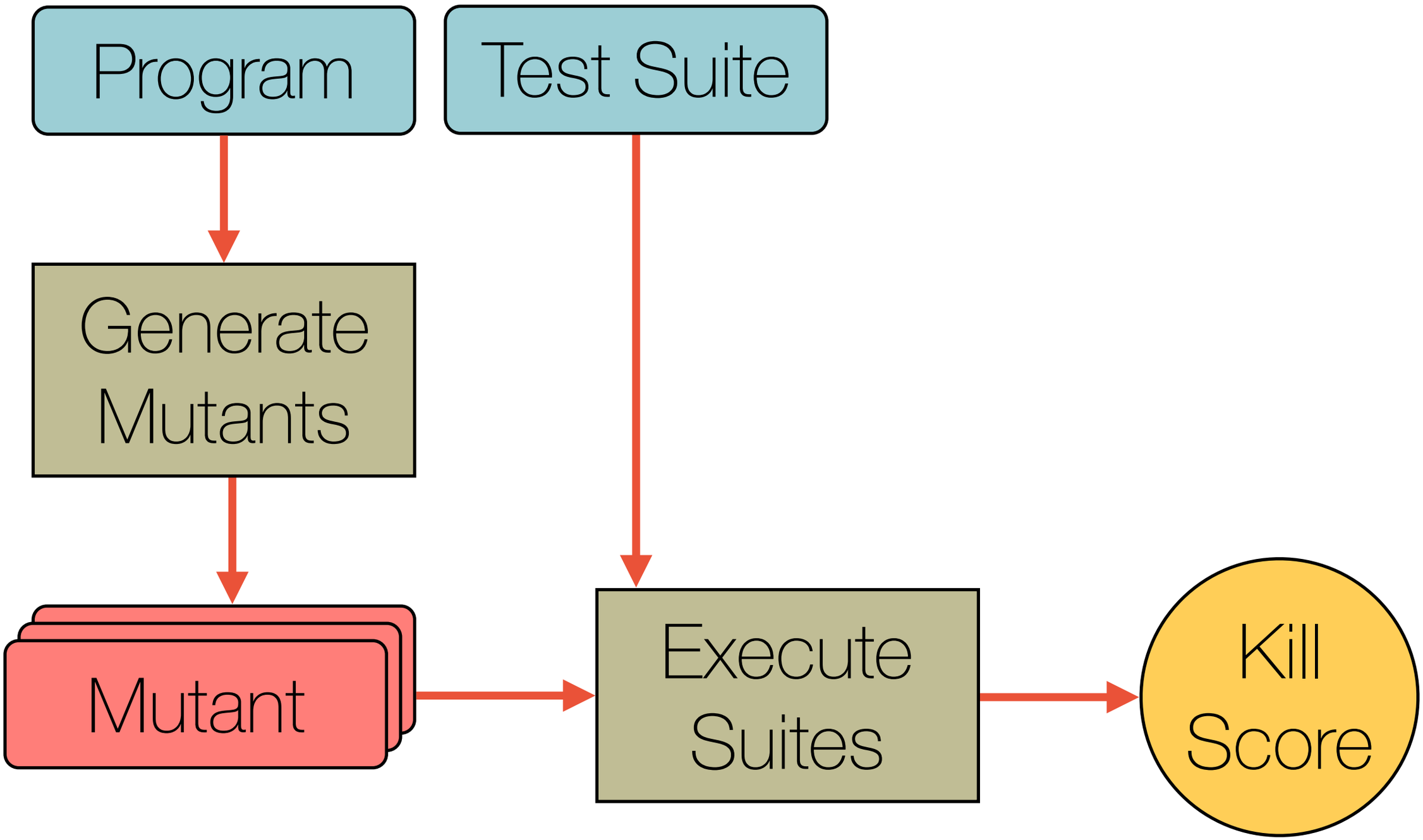

owns
owns
View
View Controller
View Model
Model
Updates
Updates

Coverage

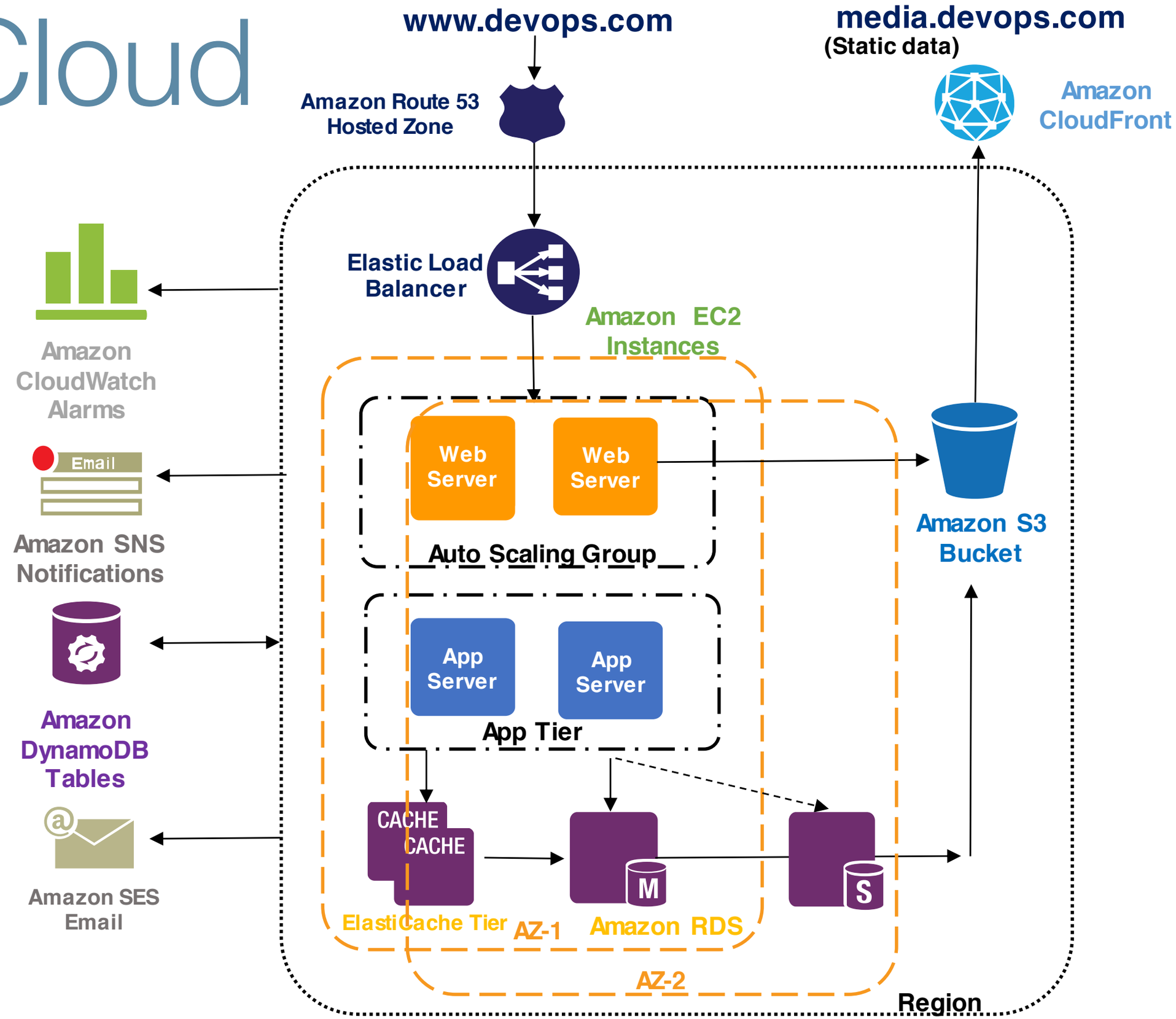
```
int eval(int x,
                boolean c1,
                boolean c2){
   if (c1)
     x++;
   if (c2)
     x--;
   return x;
}
```

```
eval(0,false,false);
```

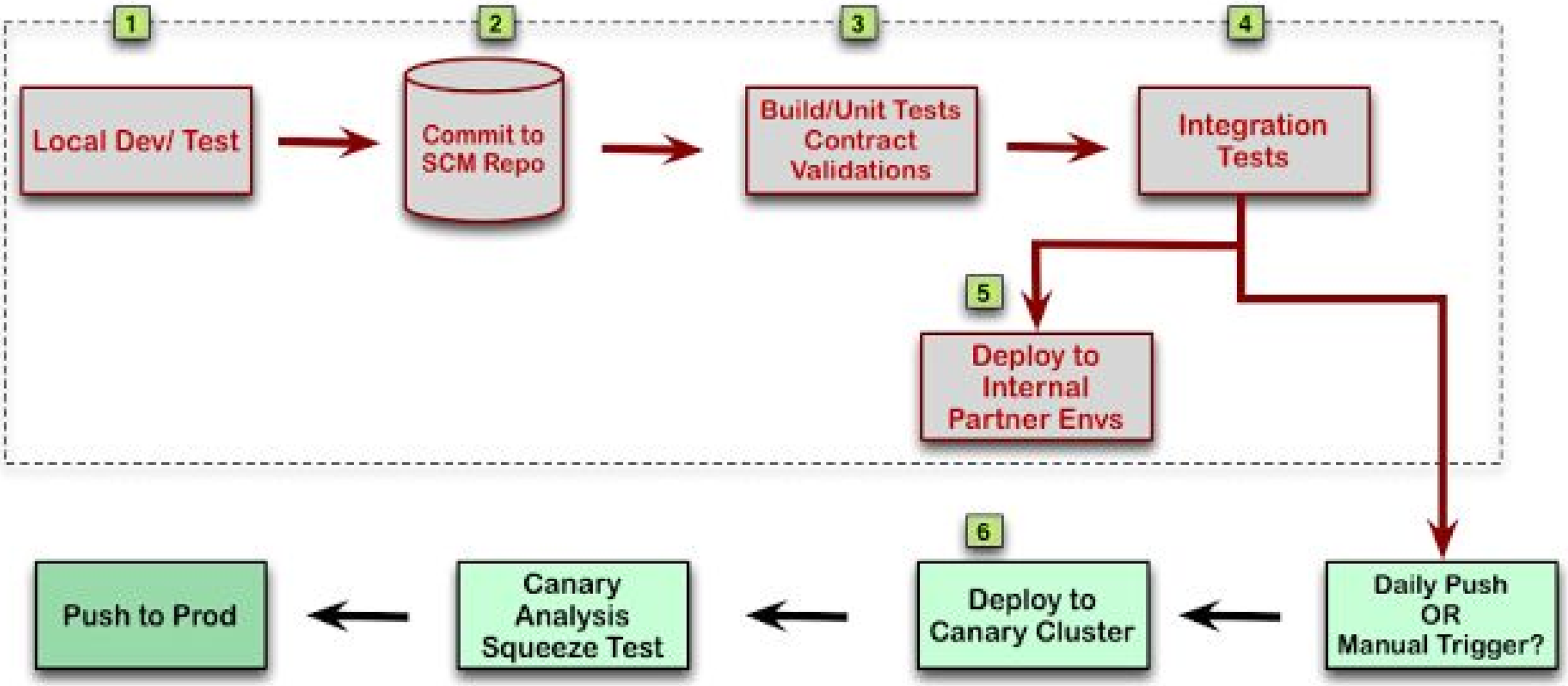
Mutation Testing



Cloud


www.devops.com
media.devops.com
(Static data)
Amazon Route 53
Hosted Zone
Amazon
CloudFront
Elastic Load
Balancer
Amazon EC2
Instances
Amazon
CloudWatch
Alarms
Email
Amazon SNS
Notifications
Web
Server
Web
Server
Amazon S3
Bucket
Auto Scaling Group
App
Server
App
Server
App Tier
Amazon
DynamoDB
Tables
CACHE
CACHE
M
S
Amazon SES
Email
ElastiCache Tier
AZ-1
Amazon RDS
AZ-2
Region

DevOps


1
2
3
4
Local Dev/ Test
Commit to SCM Repo
Build/Unit Tests Contract Validations
Integration Tests
5
Deploy to Internal Partner Envs
6
Push to Prod
Canary Analysis Squeeze Test
Deploy to Canary Cluster
Daily Push OR Manual Trigger?

Single responsibility

Open for extension (closed to modification)

Liskov substitution

Interface segregation

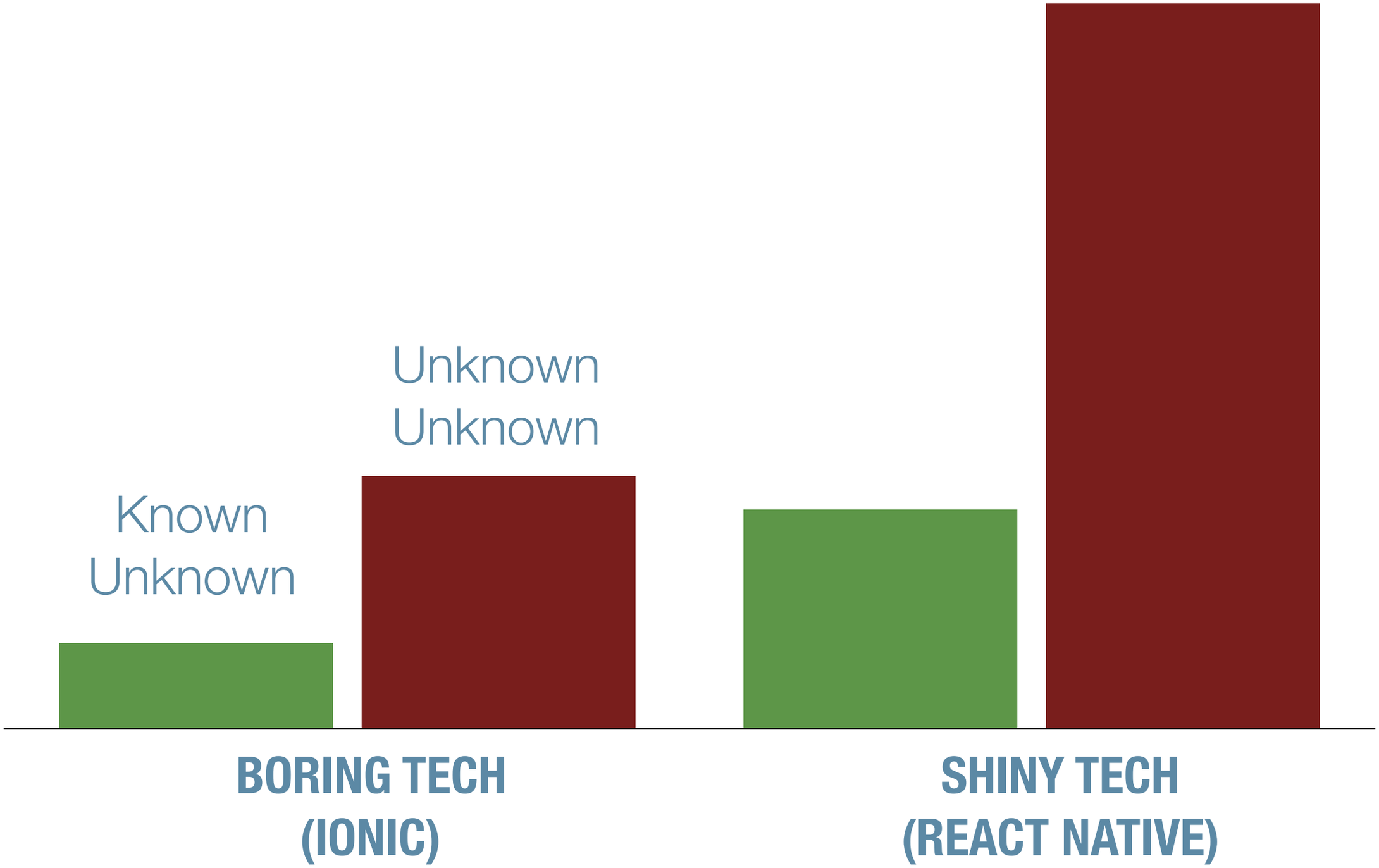
Dependency inversion

Project

Advocacy Safety Issues In The News Search

Toyota Unintended Acceleration and the Big Bowl of “Spaghetti” Code

Innovation Tokens



http://mcfunley.com/choose-boring-technology

Test Demo

90	3656	278	117
Committers	Commits	Issues	Pull Requests

Issues

Pull Requests

Changes

Weekly Commits