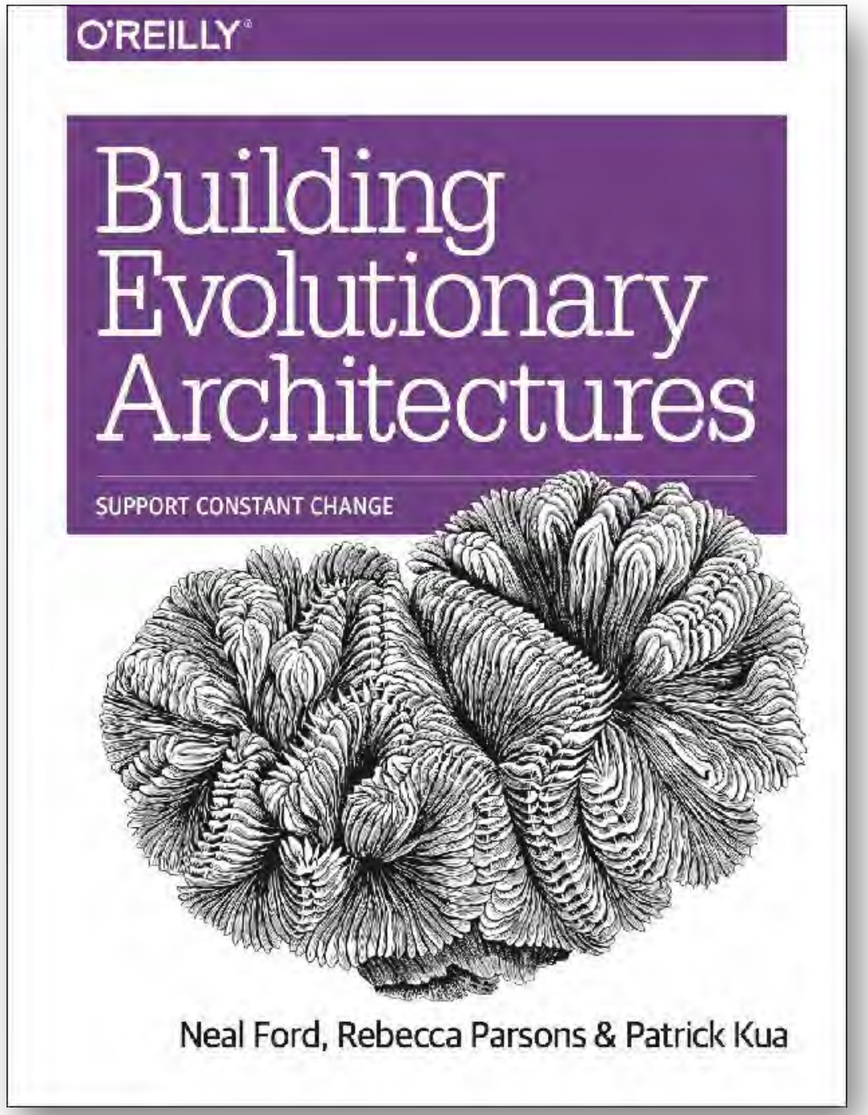
#OSCON

Building Evolutionary Architectures

Support Constant Change



Mike Mason

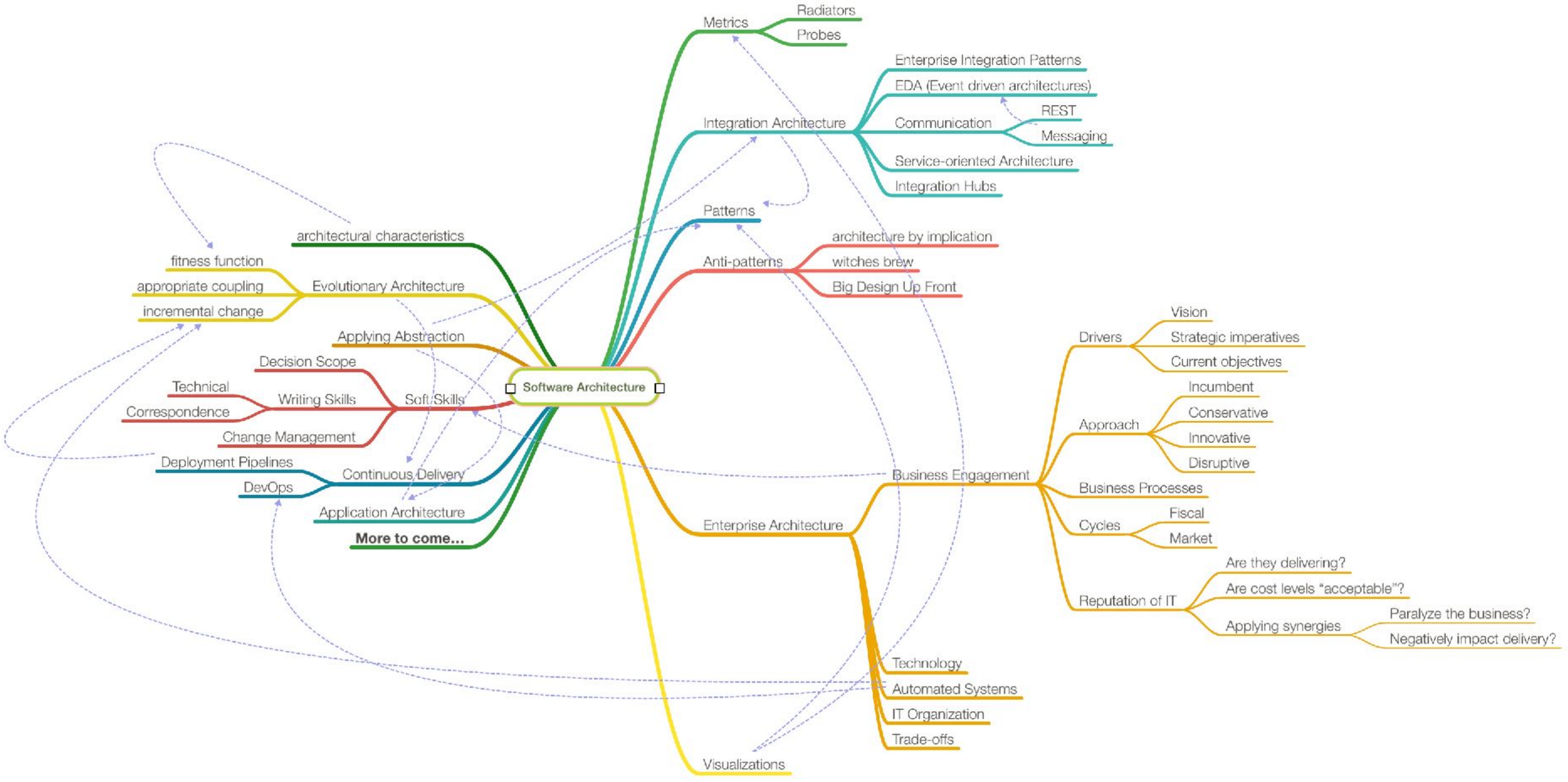
@mikemasonca

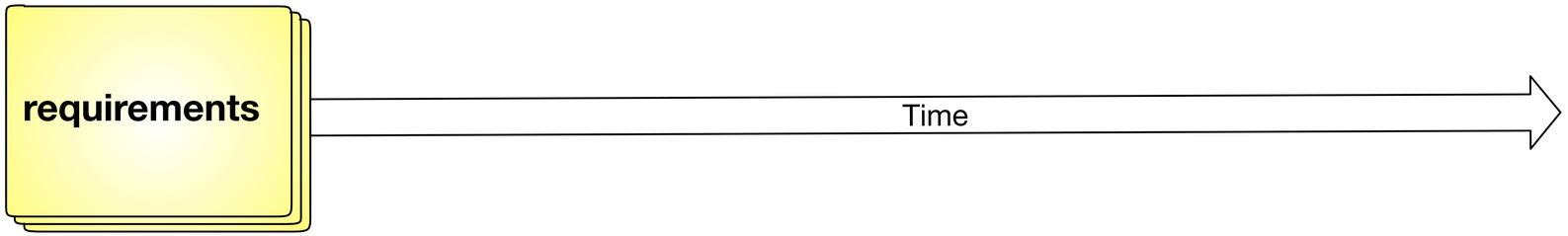
Zhamak Dehghani

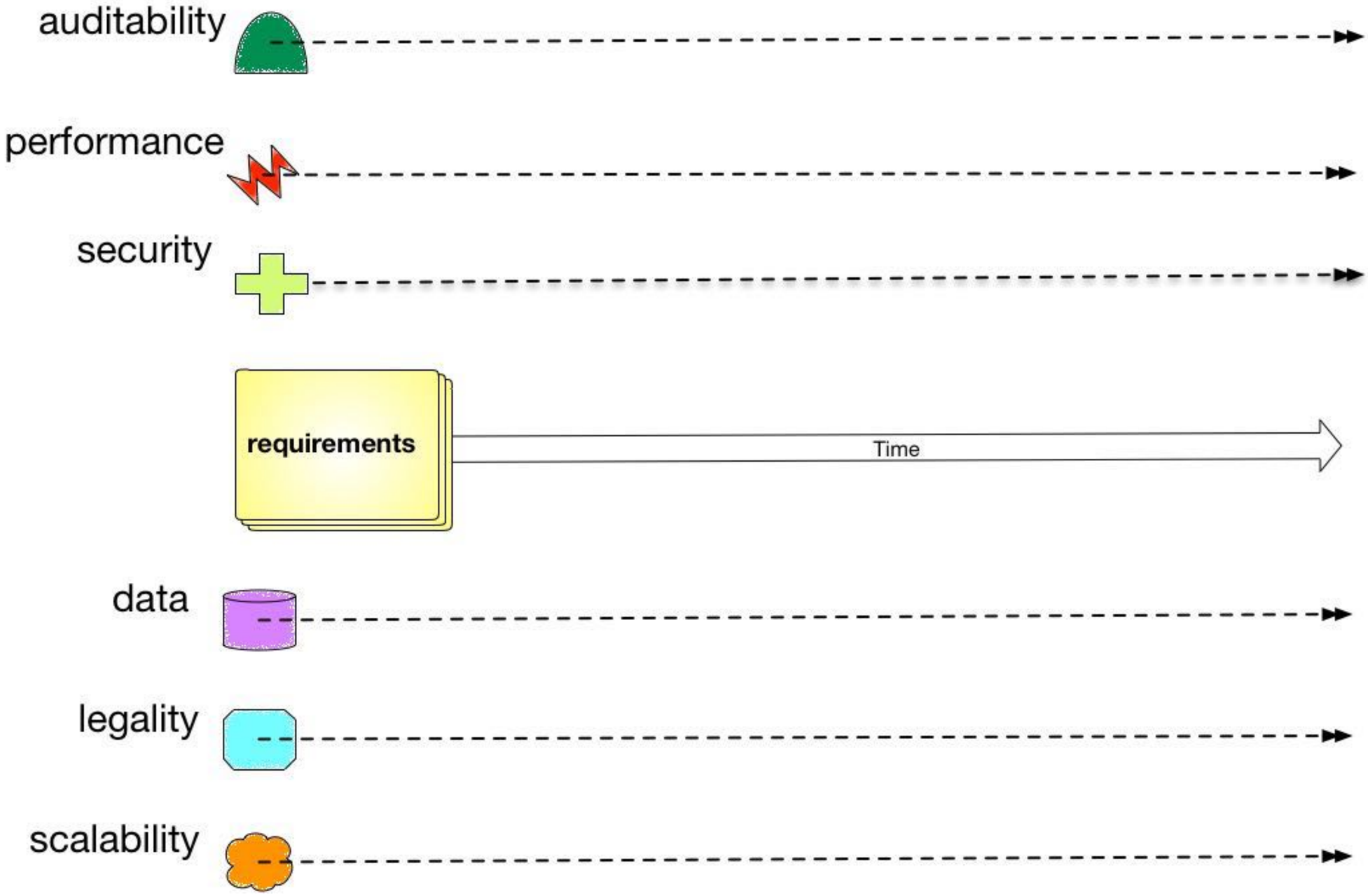
@zhamakd

OSCON Networking Opportunity

What is Software Architecture?


Software Architecture
Metrics
Radiators
Probes
Integration Architecture
Enterprise Integration Patterns
EDA (Event driven architectures)
Communication
REST
Messaging
Service-oriented Architecture
Integration Hubs
Patterns
Anti-patterns
architecture by implication
witches brew
Big Design Up Front
Enterprise Architecture
Business Engagement
Drivers
Vision
Strategic imperatives
Current objectives
Approach
Incumbent
Conservative
Innovative
Disruptive
Business Processes
Cycles
Fiscal
Market
Reputation of IT
Are they delivering?
Are cost levels "acceptable"?
Applying synergies
Paralyze the business?
Negatively impact delivery?
Technology
Automated Systems
IT Organization
Trade-offs
Visualizations
architectural characteristics
Evolutionary Architecture
fitness function
appropriate coupling
incremental change
Applying Abstraction
Soft Skills
Decision Scope
Writing Skills
Technical
Correspondence
Change Management
Continuous Delivery
Deployment Pipelines
DevOps
Application Architecture
More to come...


requirements
Time


auditability
performance
security
requirements
Time
data
legality
scalability

ility

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy
availability
compatibility
composability
configurability
correctness
credibility
customizability
debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency

reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity
interchangeability
interoperability
learnability
maintainability
manageability
mobility
modifiability
modularity
operability
orthogonality
portability
precision
predictability
process capabilities
producibility
provability
recoverability
relevance

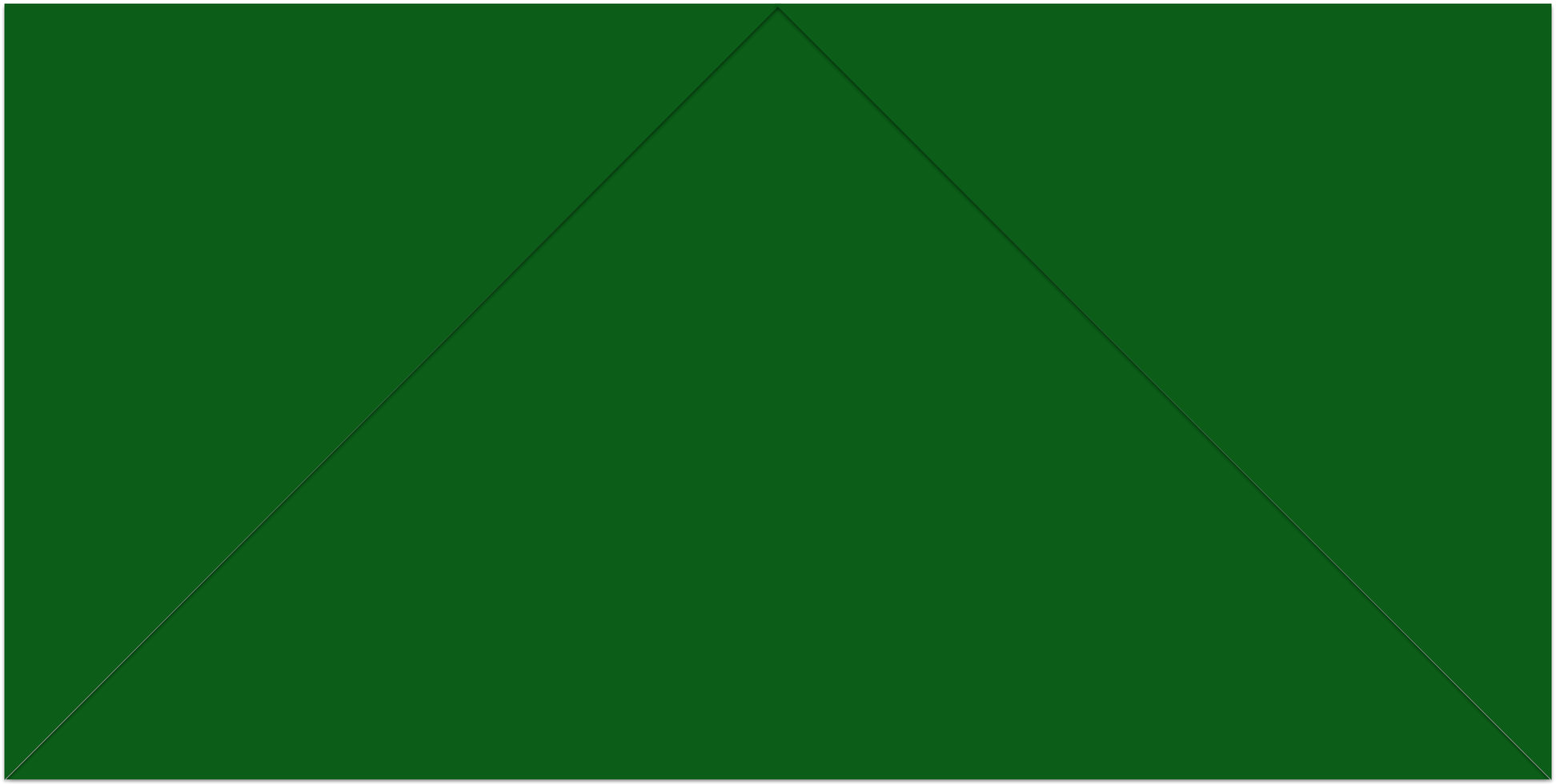
repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
self-sustainability
serviceability
supportability
securability
simplicity
stability
standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

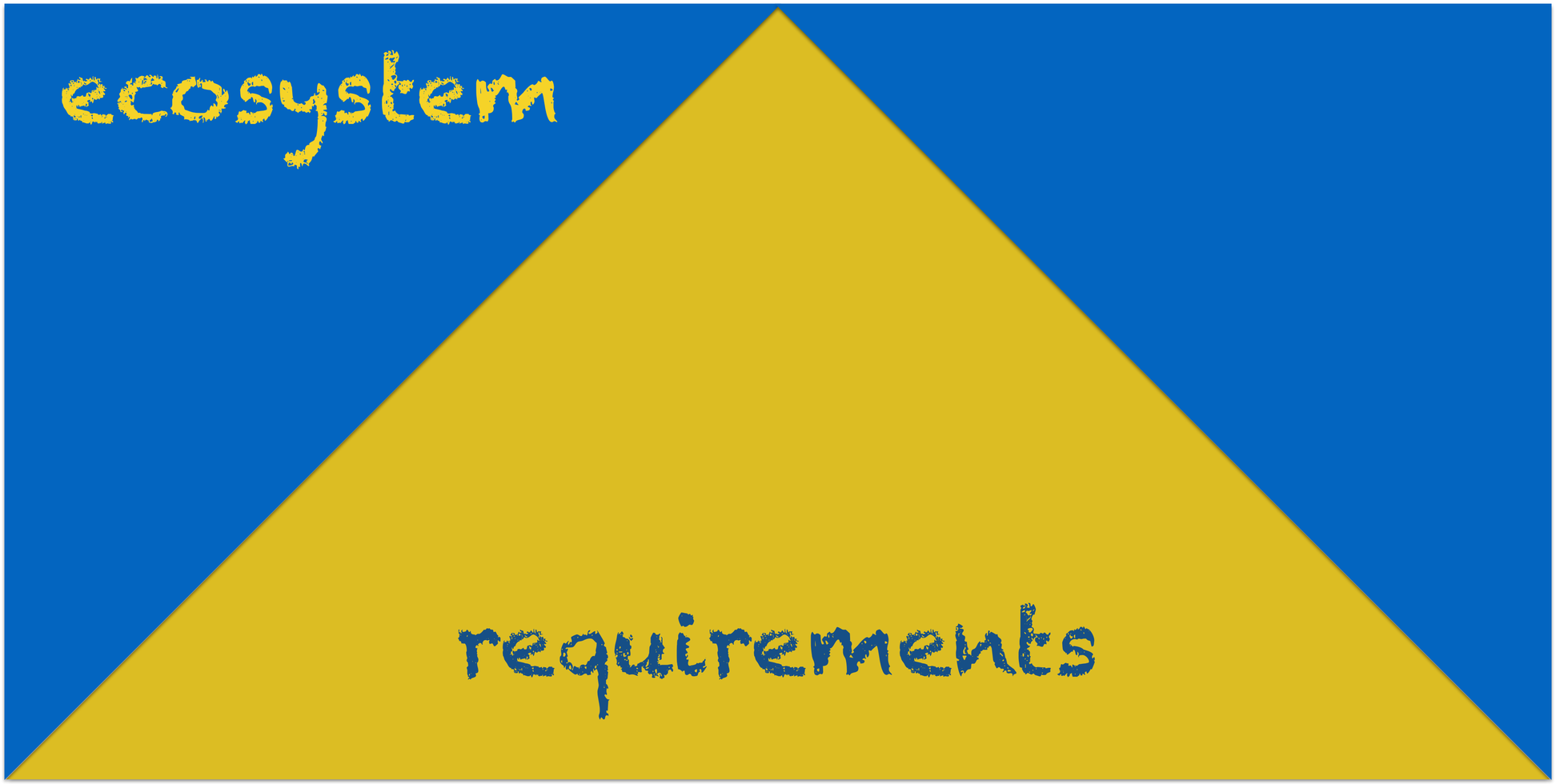
accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy
availability
compatibility
composability
configurability
correctness
credibility
customizability
debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency
reliability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity
interchangeability
interoperability
learnability
maintainability
manageability
mobility
modifiability
modularity
operability
orthogonality
portability
precision
predictability
process capabilities
producibility
provability
recoverability
relevance
repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
self-sustainability
serviceability
supportability
securability
simplicity
stability
standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

evolvability

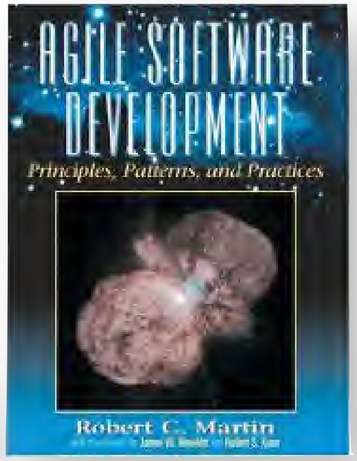
Change

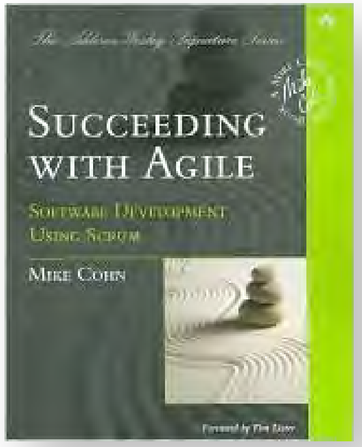


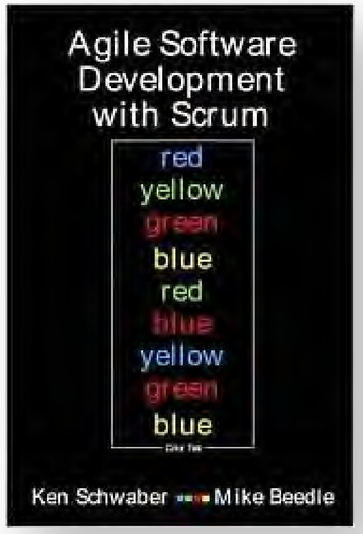
Change

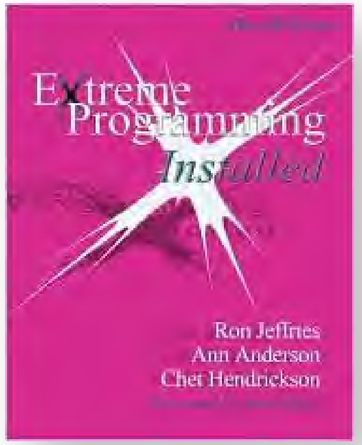


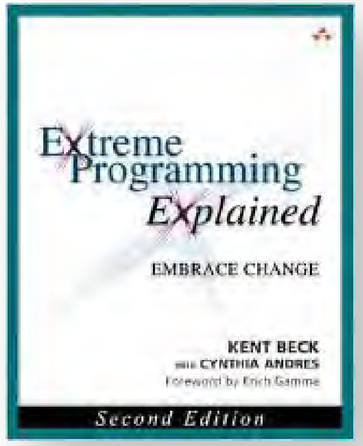
Business Change



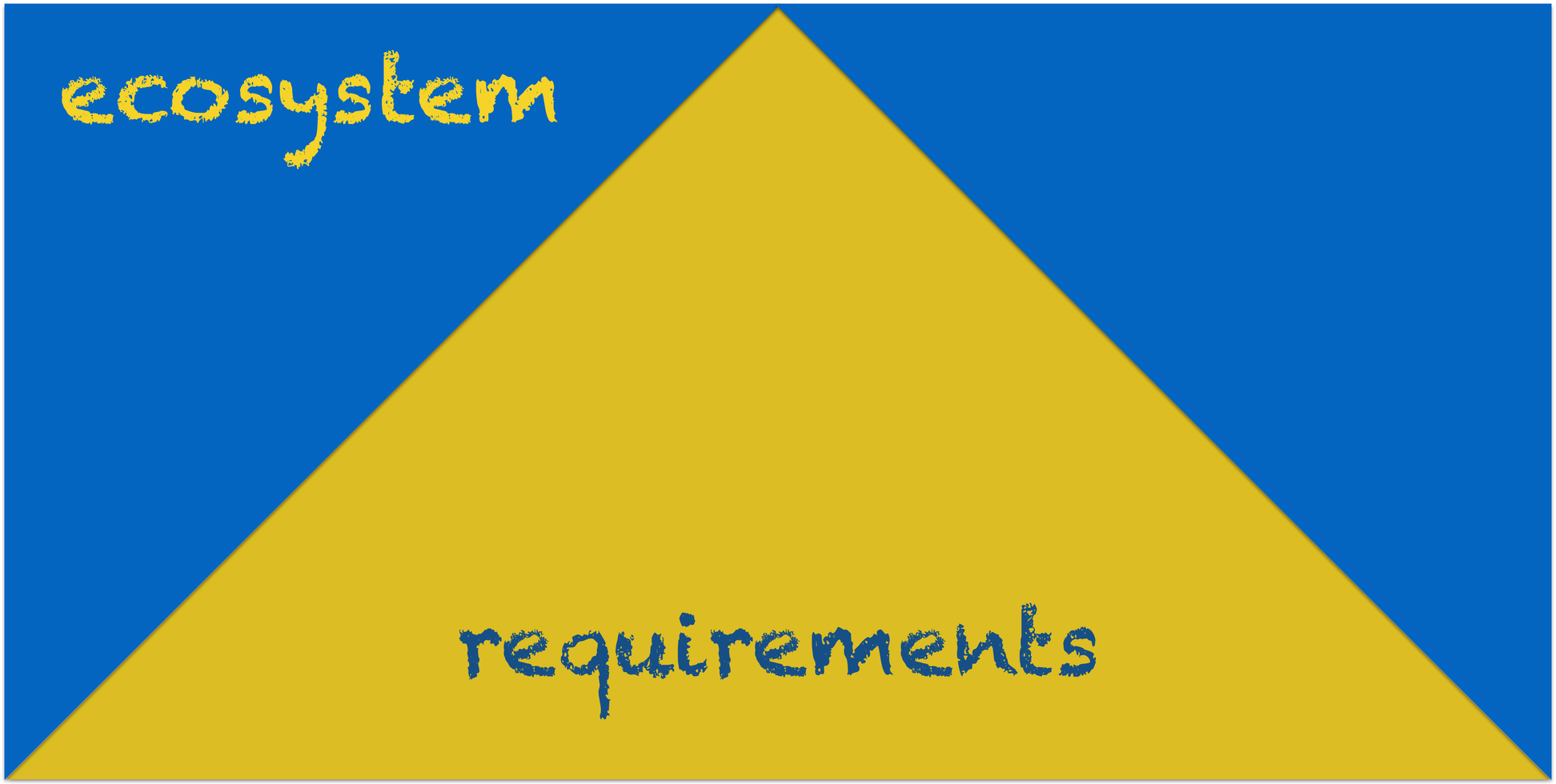






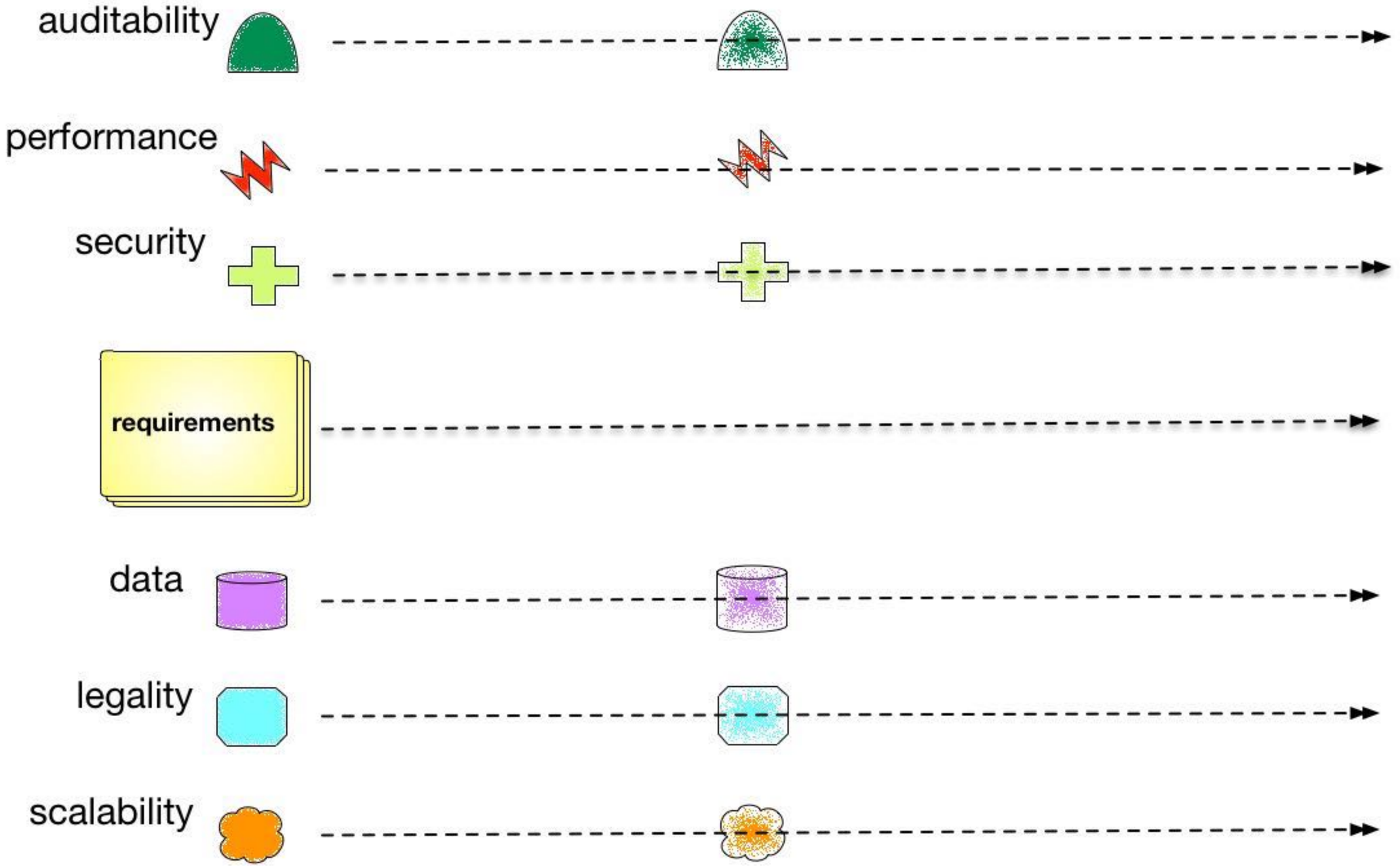


Change



Dynamic Equilibrium

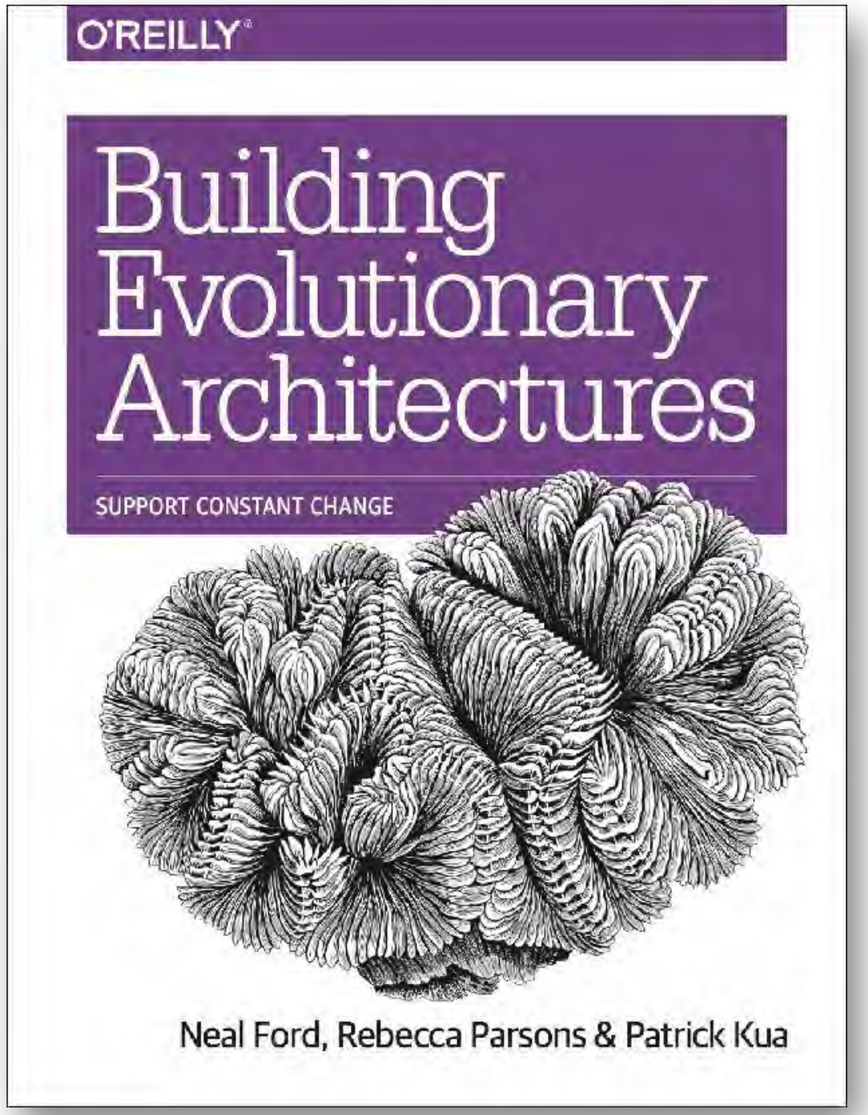
How is long term planning possible when things constantly change in unexpected ways?


auditability
performance
security
requirements
data
legality
scalability

Once I’ve built an architecture, how can I prevent it from gradually degrading over time?

#OSCON

Building Evolutionary Architectures



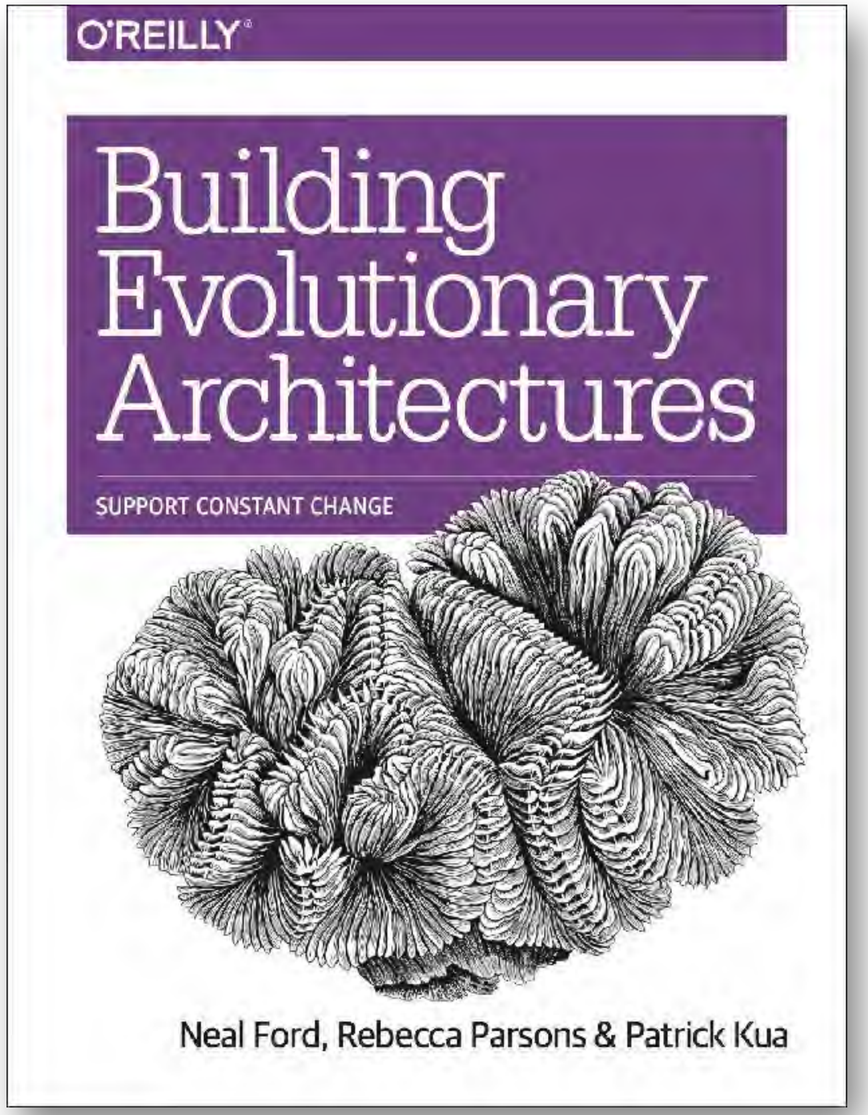
Group Icebreaker - 5 mins

In each group, choose a system that one of your members has worked on. Explain the architecture to the rest of the group — we'll use this to anchor today's workshop exercises.

Be ready to share at the end!

#OSCON

Building Evolutionary Architectures



Defining Evolutionary Architecture

Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions.

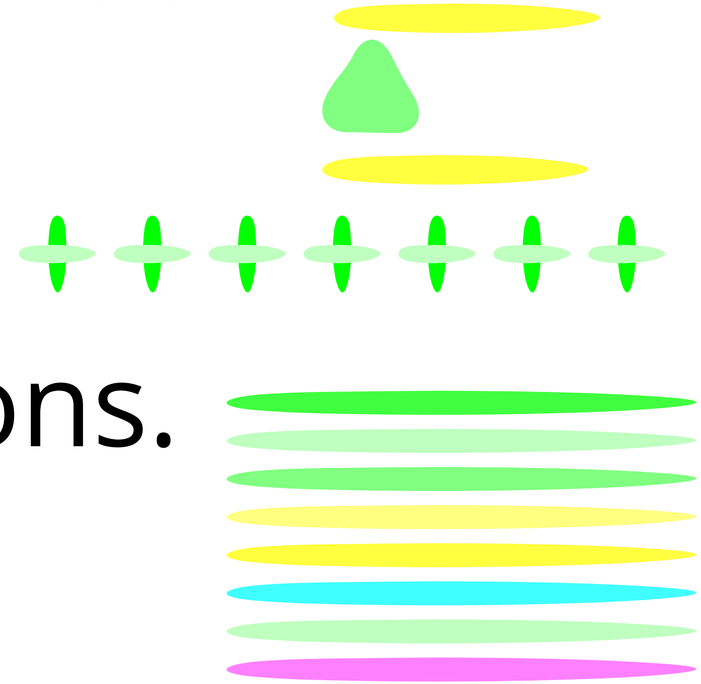
Evolutionary Architecture

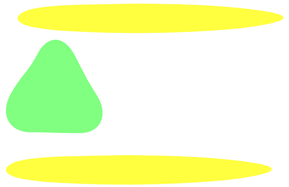
An evolutionary architecture supports

guided

incremental change

across multiple dimensions.





guided

evolutionary computing fitness function:
a particular type of objective function that is used to summarize…how close a given design solution is to achieving the set aims.


guided

An architectural fitness function provides an objective integrity assessment of some architectural characteristic(s).

Evolutionary Architecture

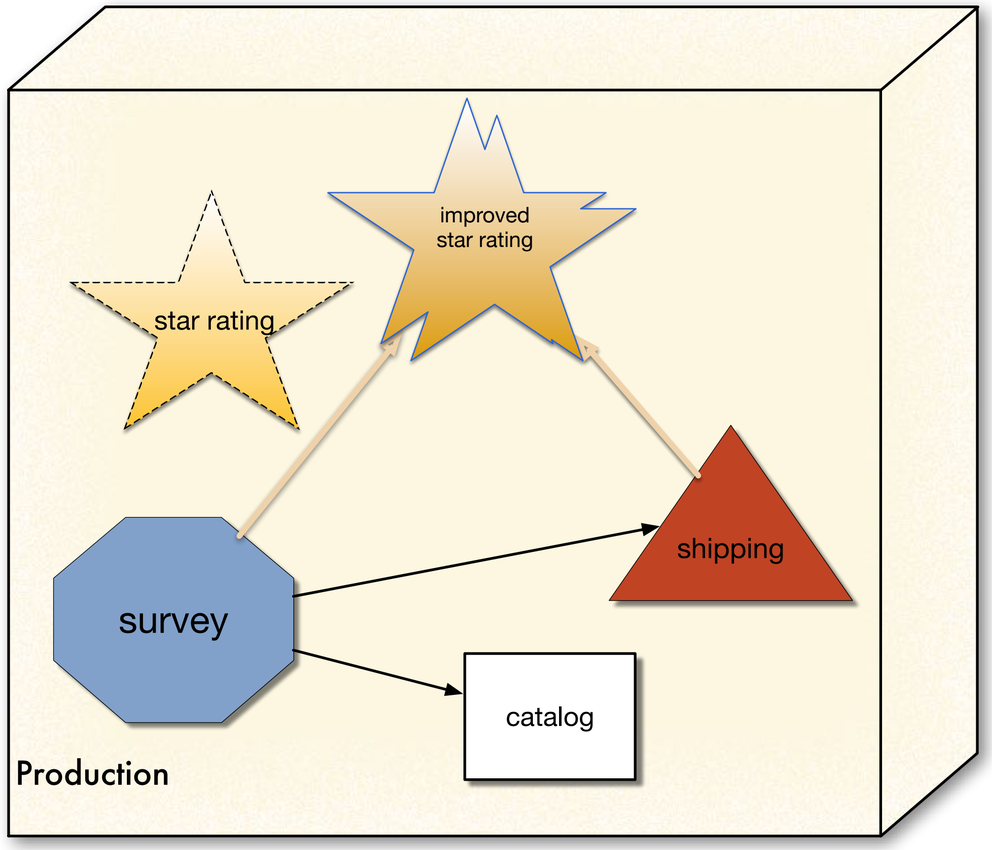
An evolutionary architecture supports

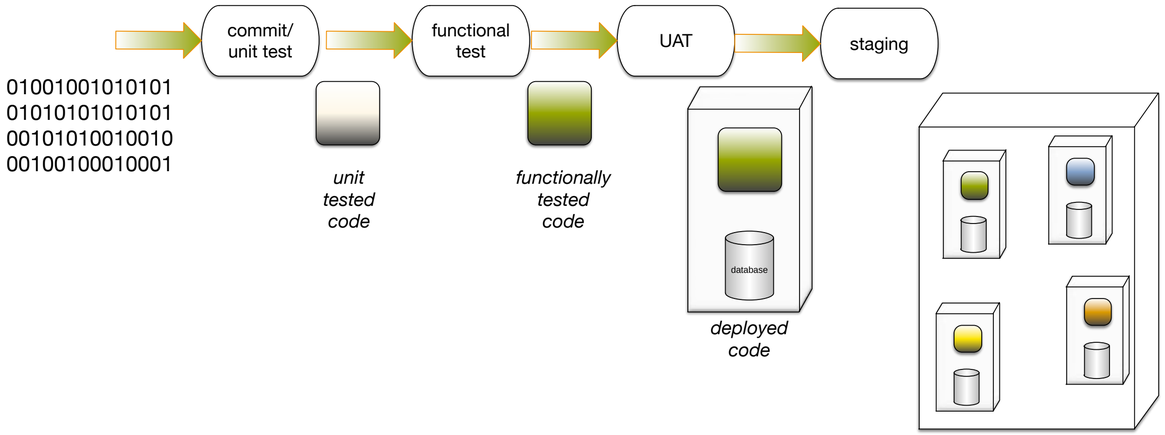
guided

incremental change

across multiple dimensions.

incremental

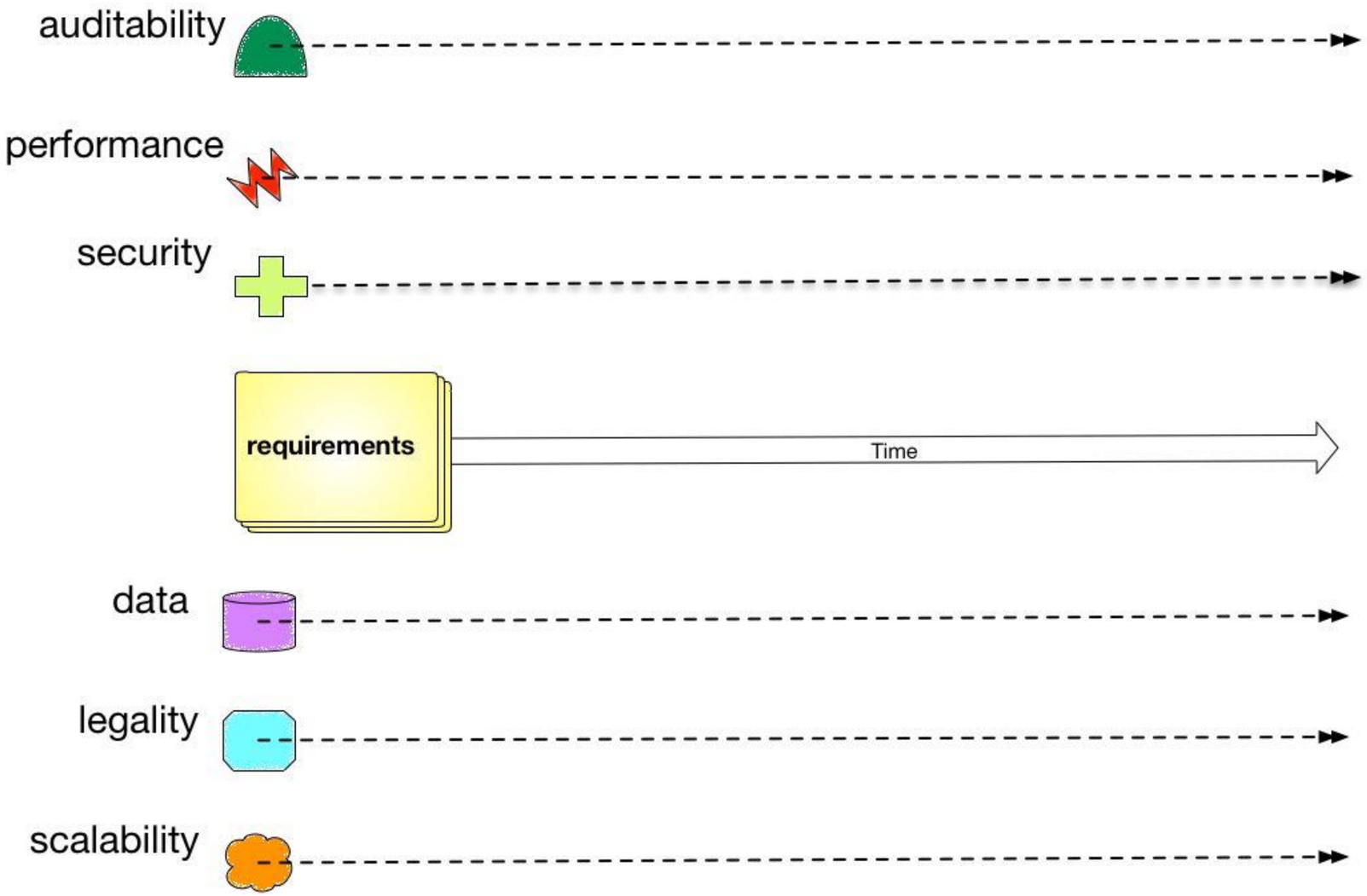




Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions

multiple dimensions



Evolutionary Architecture

An evolutionary architecture supports
guided,
incremental change
across multiple dimensions

agile

continual

emergent

Why *evolutionary?*

reactive

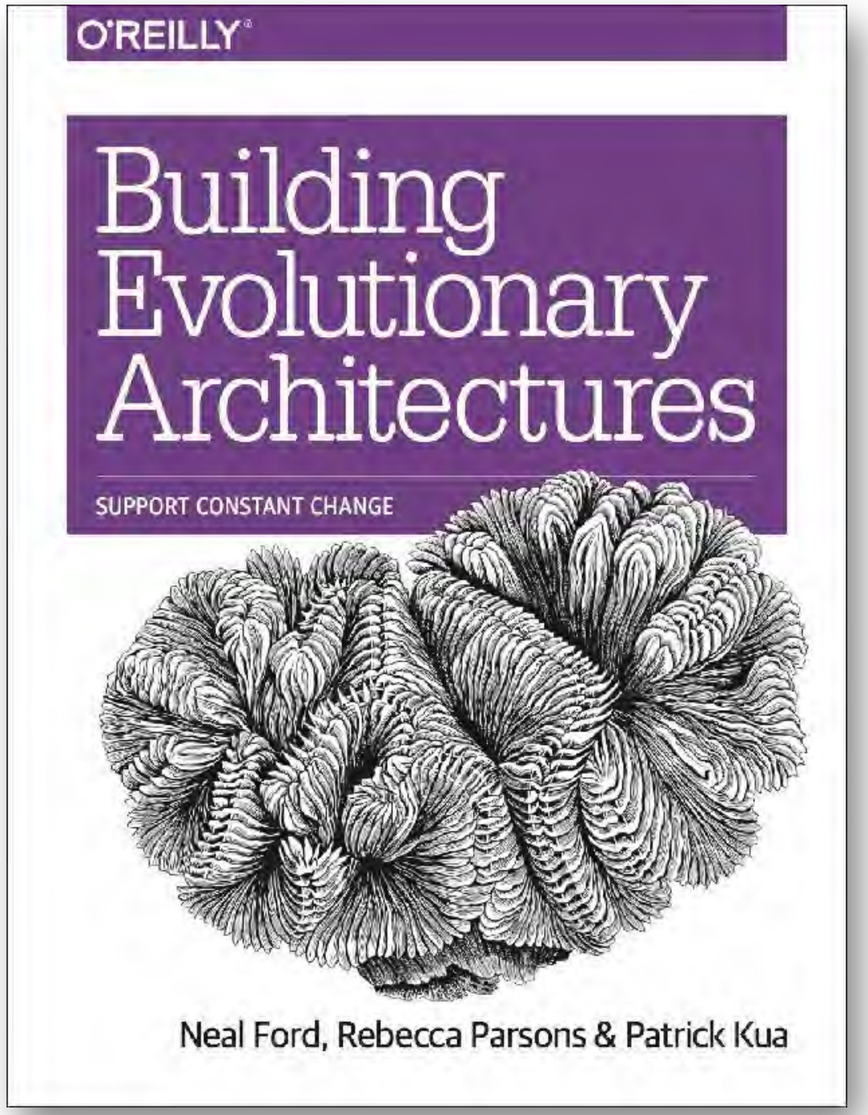
incremental

Why *evolutionary*?

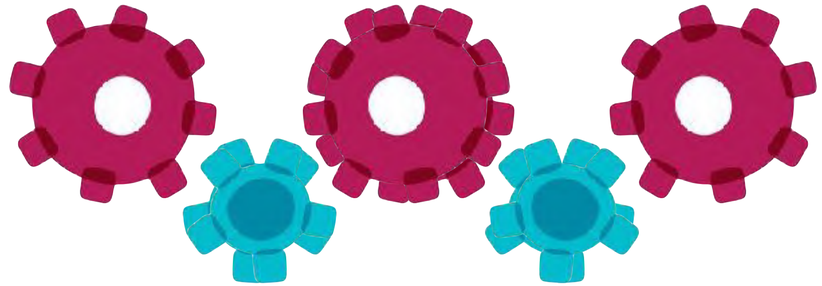
adaptable?

#OSCON

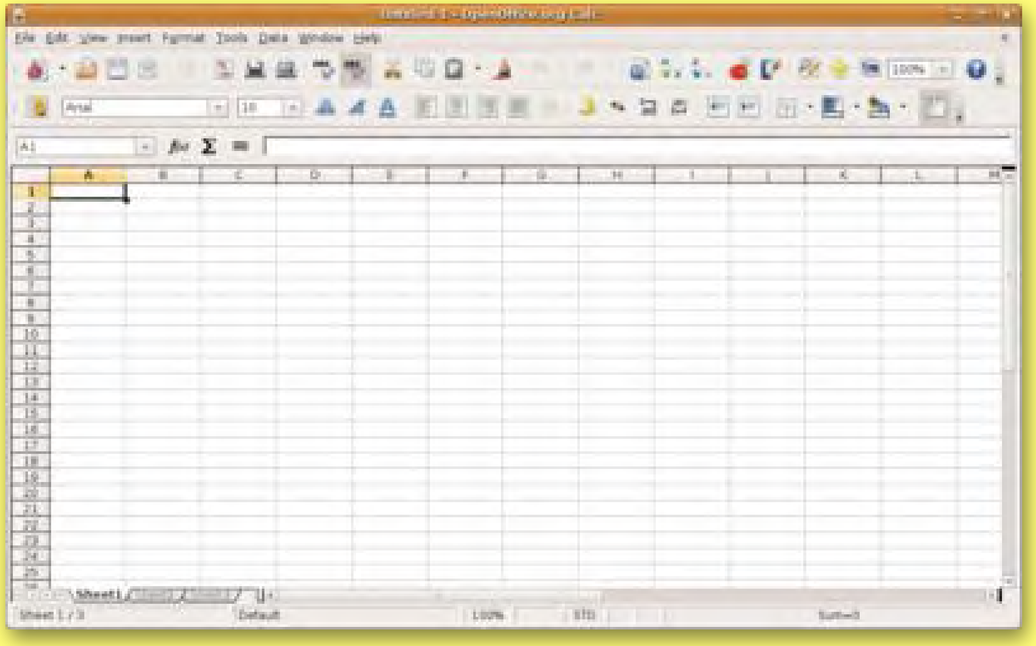
Building Evolutionary Architectures



Penultimate



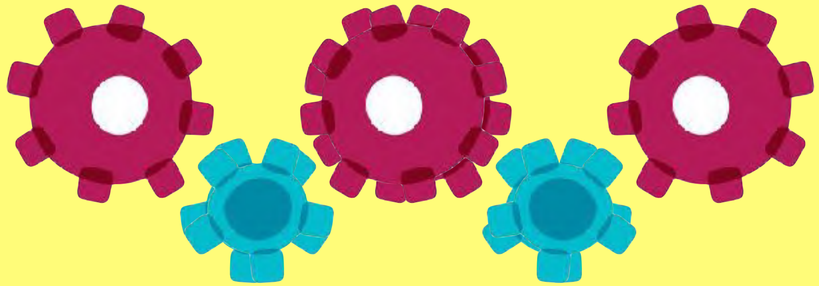
EA Spreadsheet



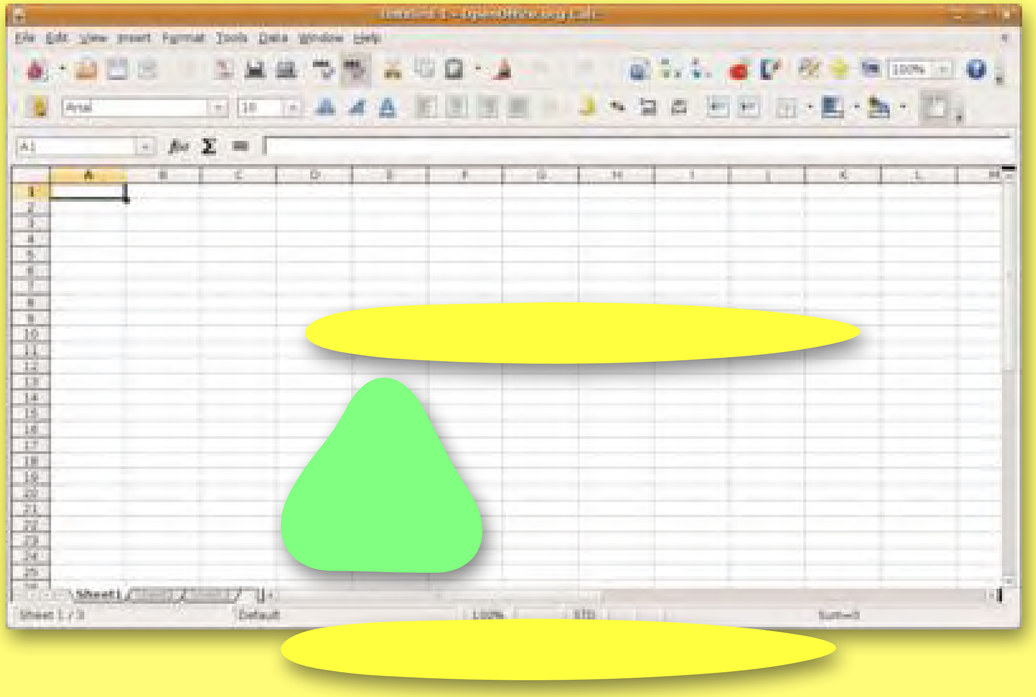
✓ definition

! verification

Penultima⬆e



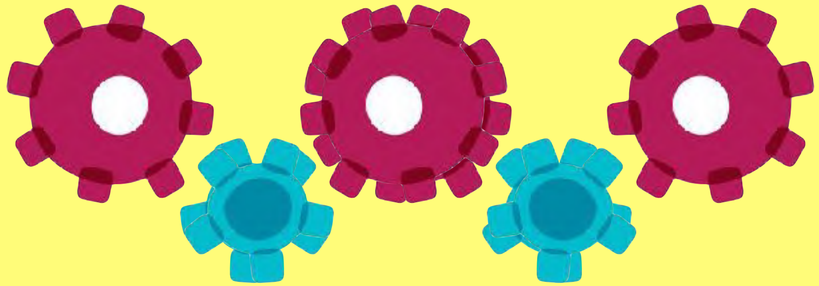
EA Spreadsheet



✓ definition

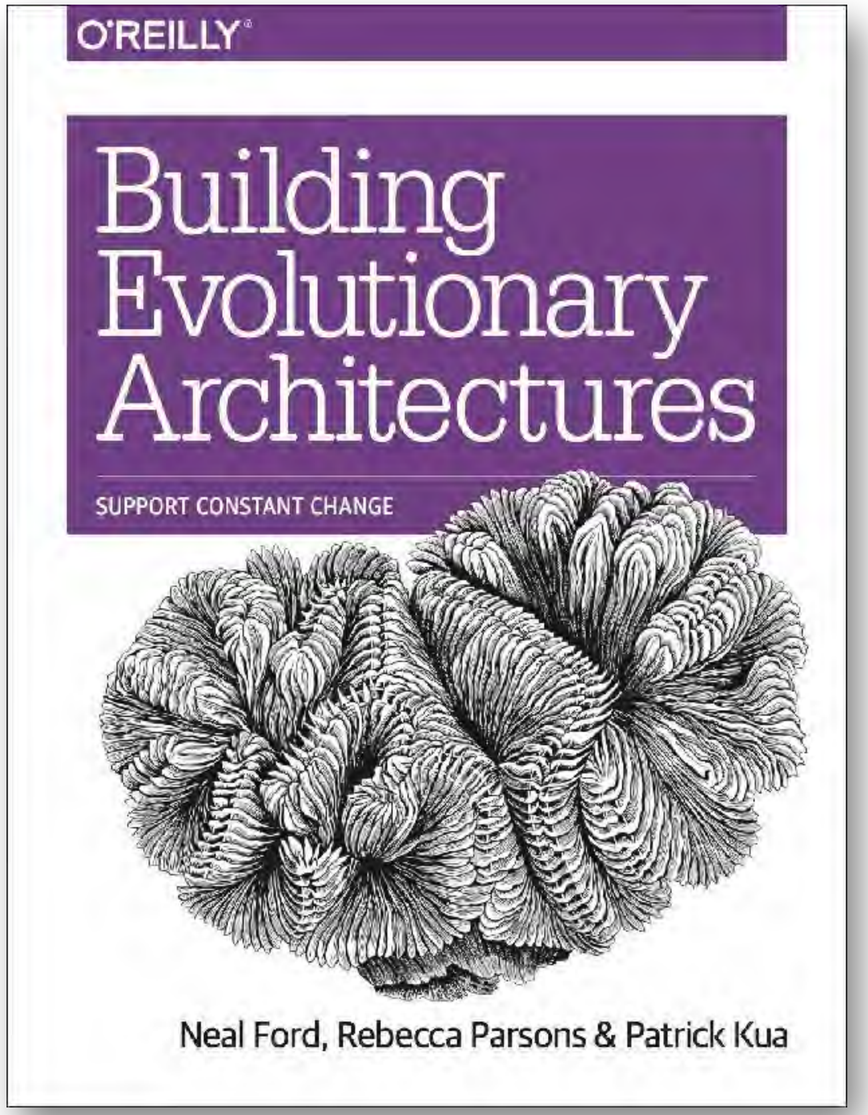
✓! verification

Penultima⬆e



#OSCON

Building Evolutionary Architectures



FITNESS FUNCTIONS:
CATEGORIES AND EXAMPLES

Categories of Fitness Functions

run against a singular context and exercise one particular aspect of the architecture.

run against a shared context and exercise a combination of architectural aspects such as security and scalability

Categories of Fitness Functions

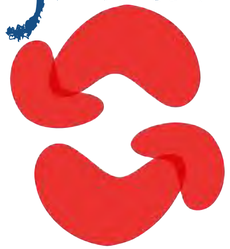
run based on a particular event, such as a developer executing a unit test, a deployment pipeline running unit tests, or a QA person performing exploratory testing.

don’t run on a schedule, but instead execute constant verification of architectural aspect(s) such as transaction speed.

Categories of Fitness Functions

static 

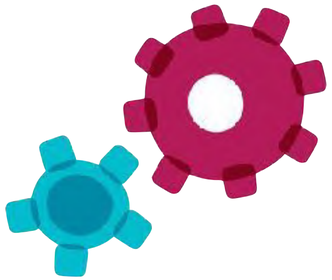
have a fixed result, such as the binary pass/fail of a unit test.

dynamic 

rely on a shifting definition based on extra context. Some values may be contingent on circumstances, and most architects will accept lower performance metrics when operating at high scale.

Categories of Fitness Functions

automated



tests and other verification mechanism that run without human interaction.

manual

must involve at least one human.

Categories of Fitness Functions

architects may want to build a time component into assessing fitness

break on upgrade

overdue library update

Categories of Fitness Functions

Some architectures have specific concerns, such as special security or regulatory requirements

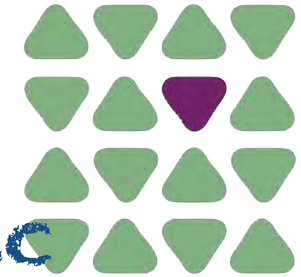
Categories of Fitness Functions

architects will define most fitness functions at project inception as they elucidate the characteristics of the architecture...

...some fitness functions will emerge during development of the system

Fitness Function

atomic

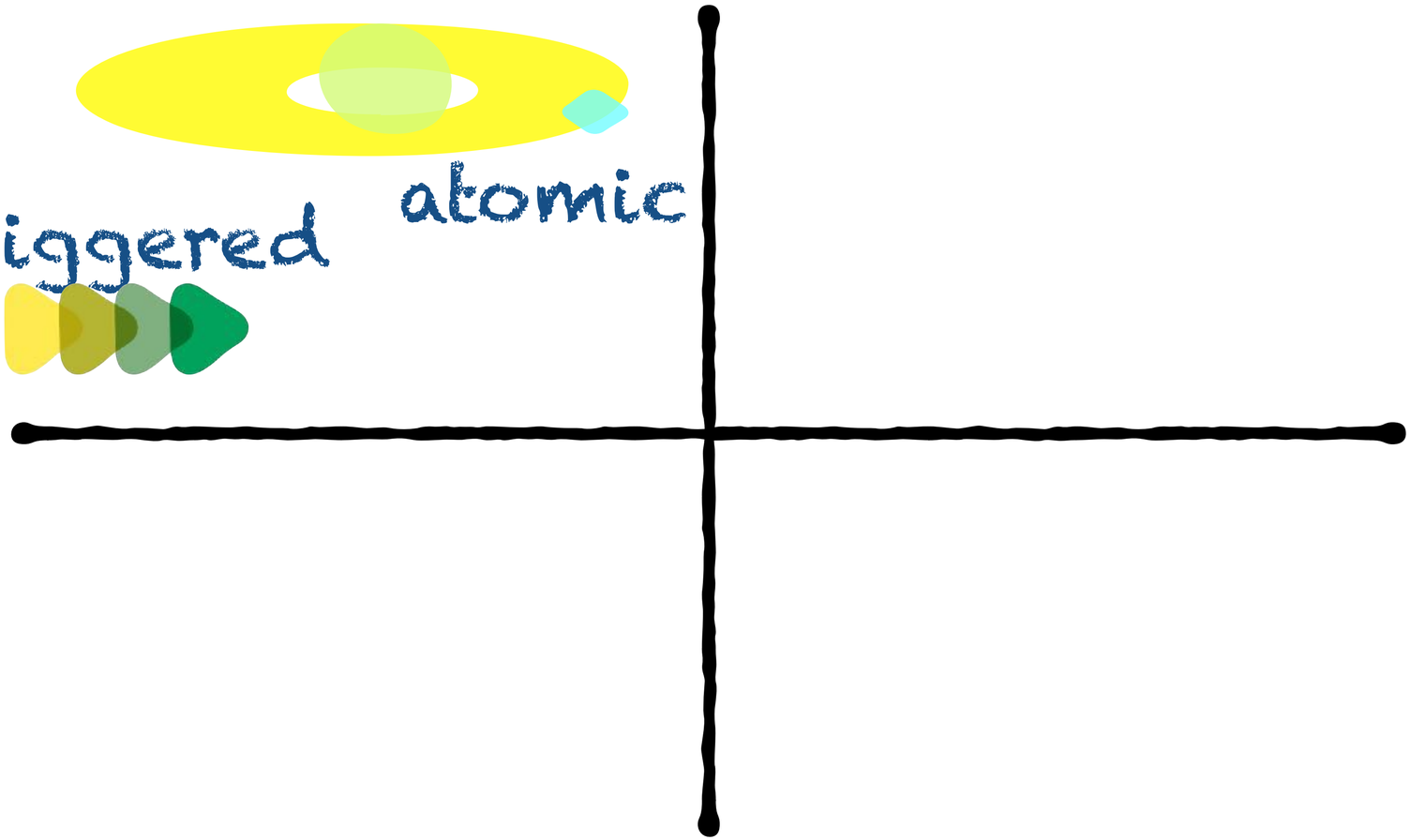


holistic

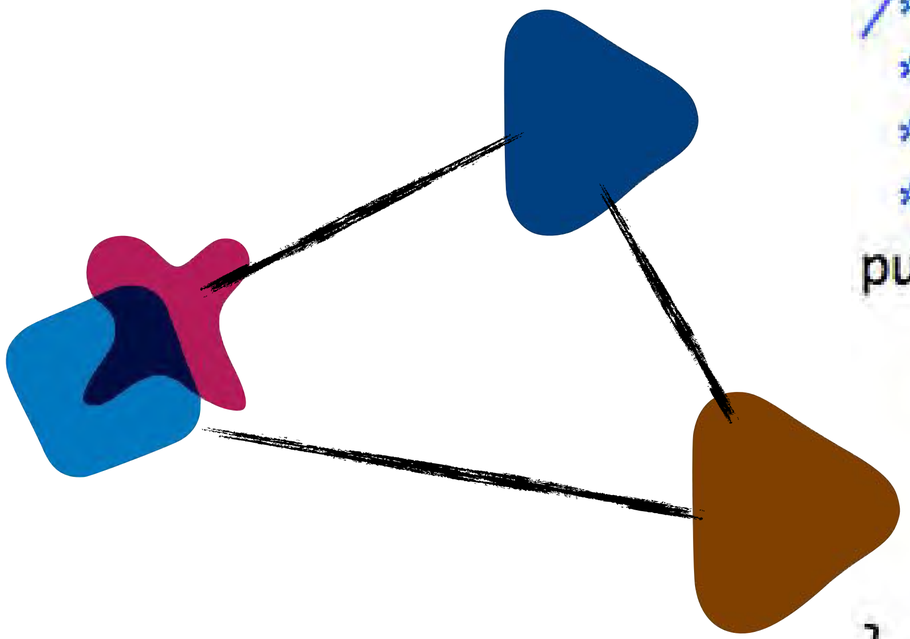
triggered

continuous

Fitness Function



Cyclic Dependency Function

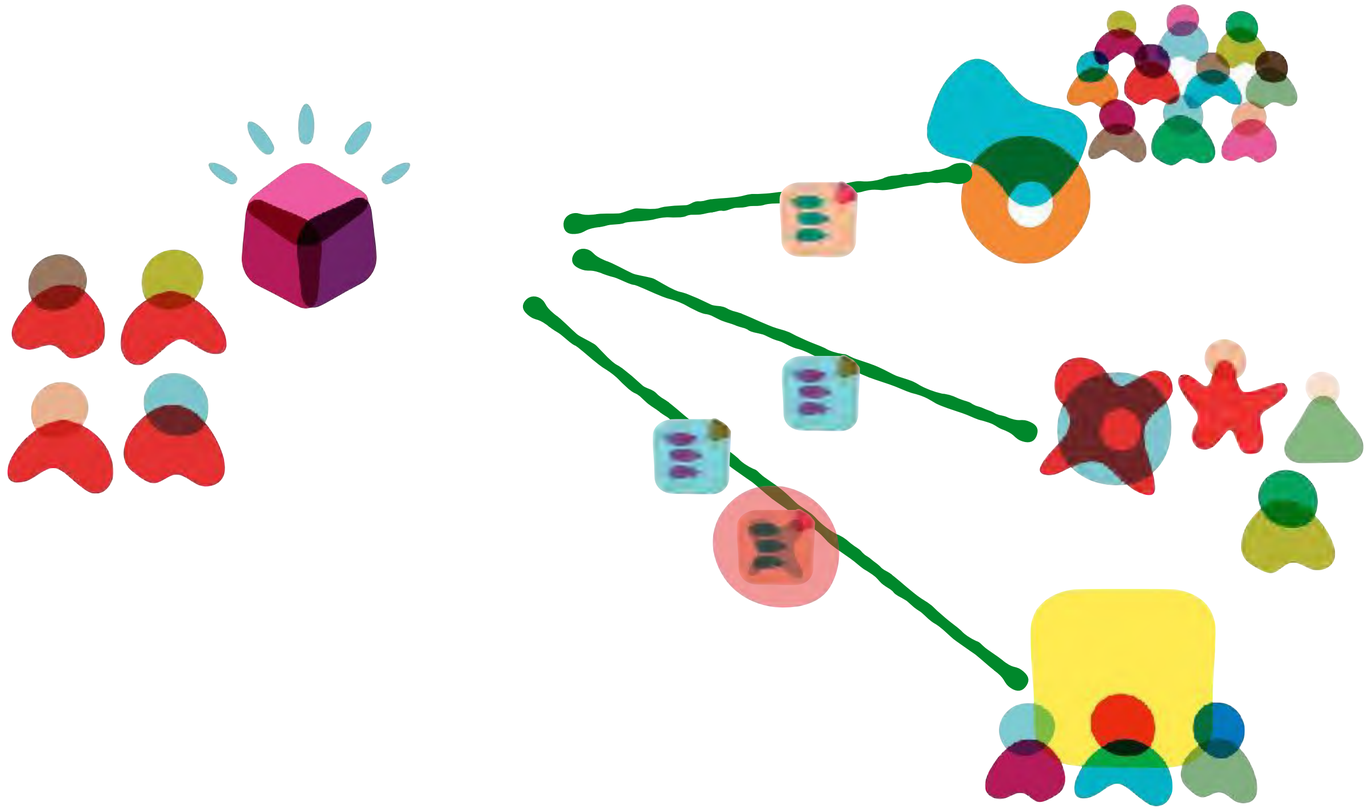


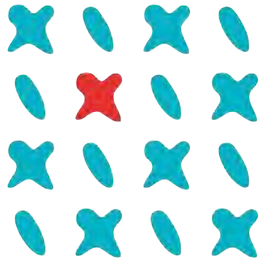
```
/**
 * Tests that a package dependency cycle does not
 * exist for any of the analyzed packages.
 */
public void testAllPackages() {

    Collection packages = jdepend.analyze();

    assertEquals("Cycles exist",
                 false, jdepend.containsCycles());
}
```

clarkware.com/software/JDepend.html

application

Consumer Driven Contracts

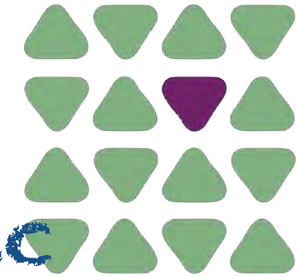




martinfowler.com/articles/consumerDrivenContracts.html

Fitness Function

atomic



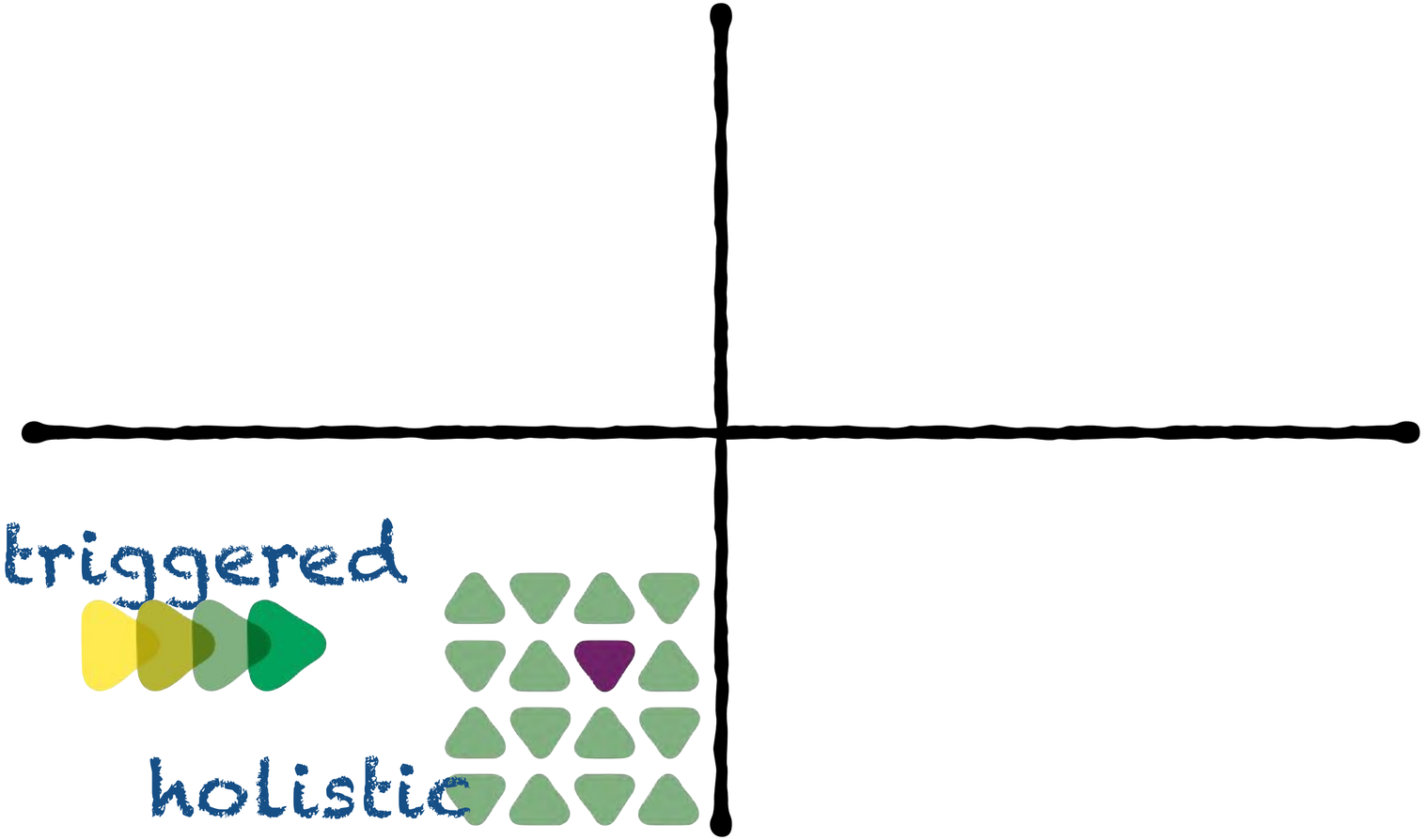
holistic

triggered

continuous

Fitness Function

atomic

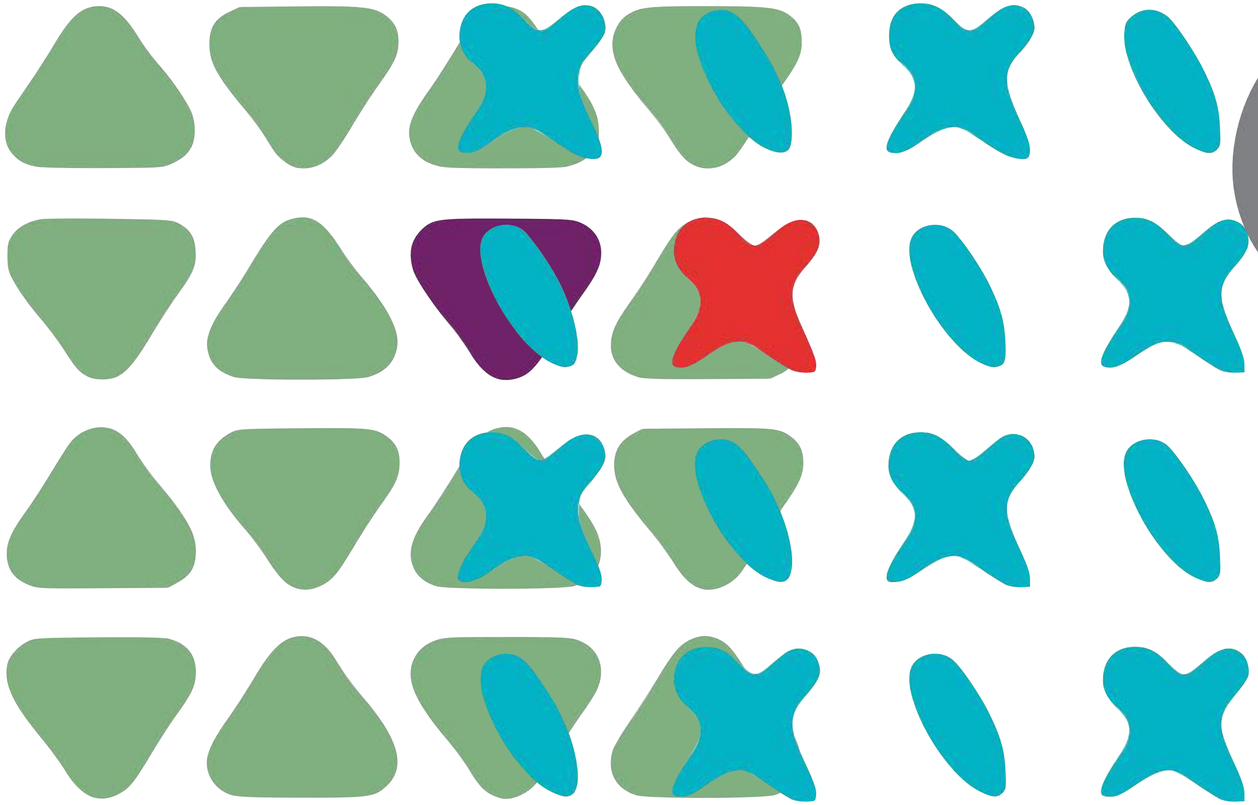


triggered

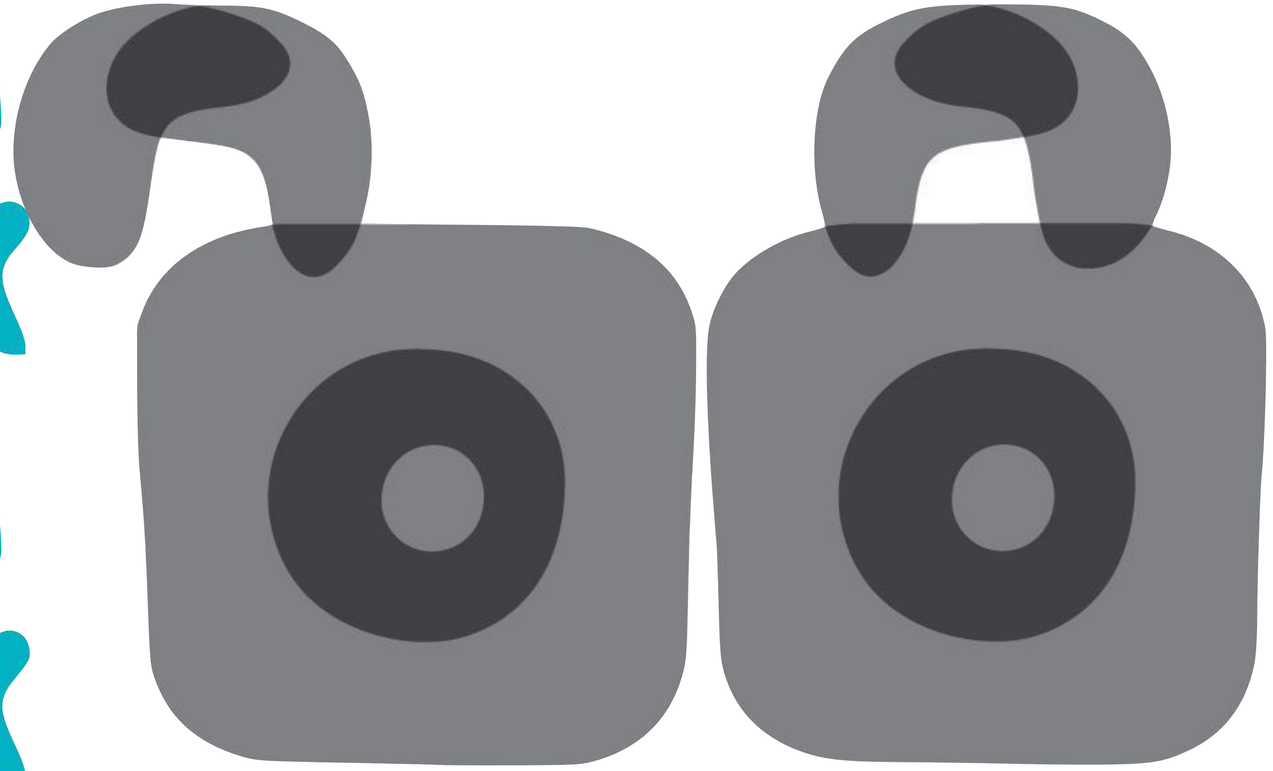
holistic

continuous

triggered



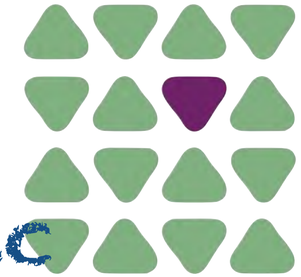
holistic



Holistic fitness functions must run in a specific (shared) context.

Fitness Function

atomic

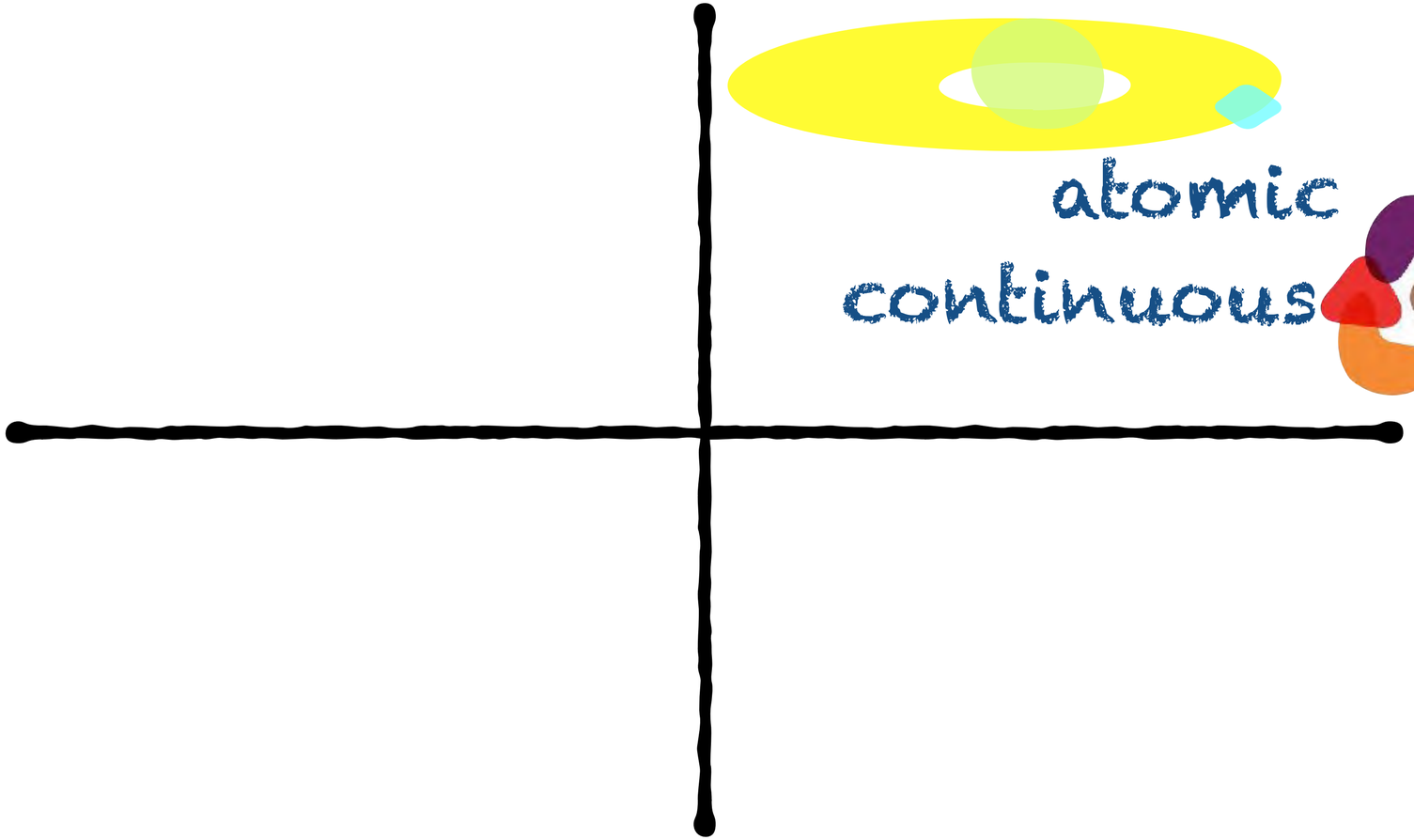


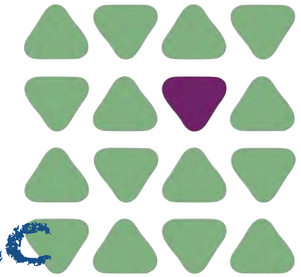
holistic

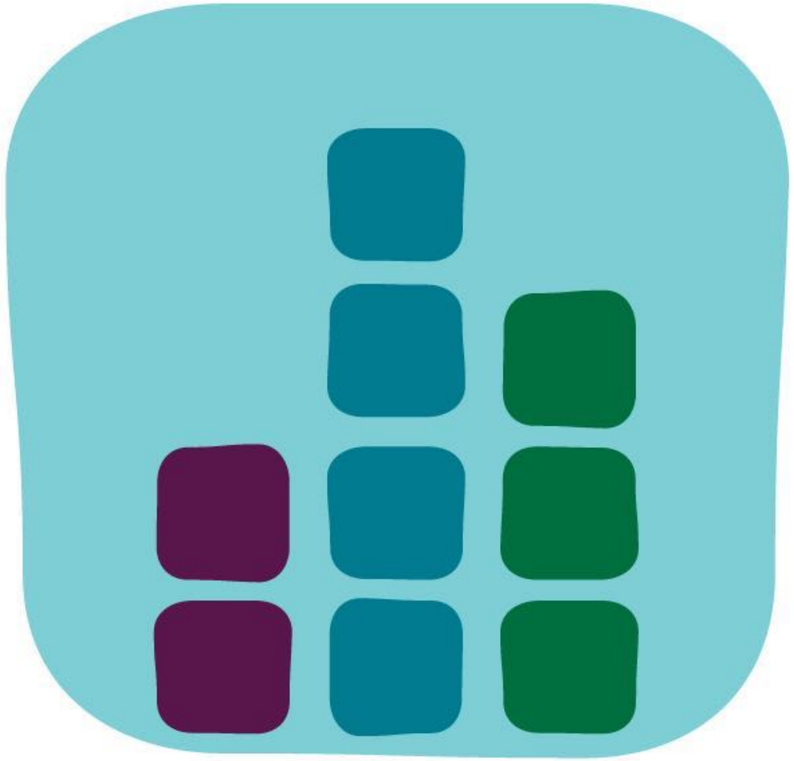
triggered

continuous

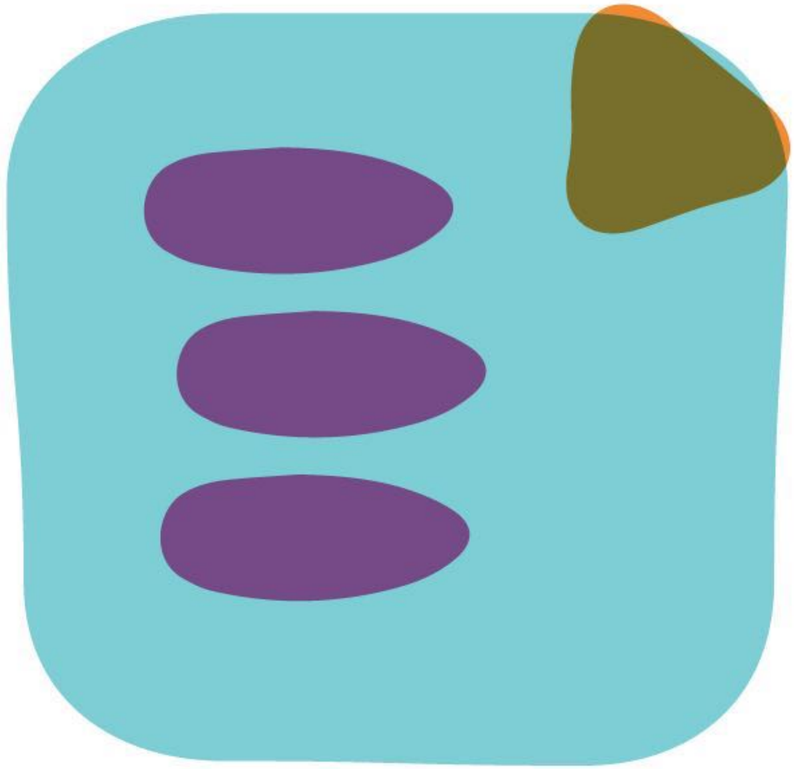
Fitness Function





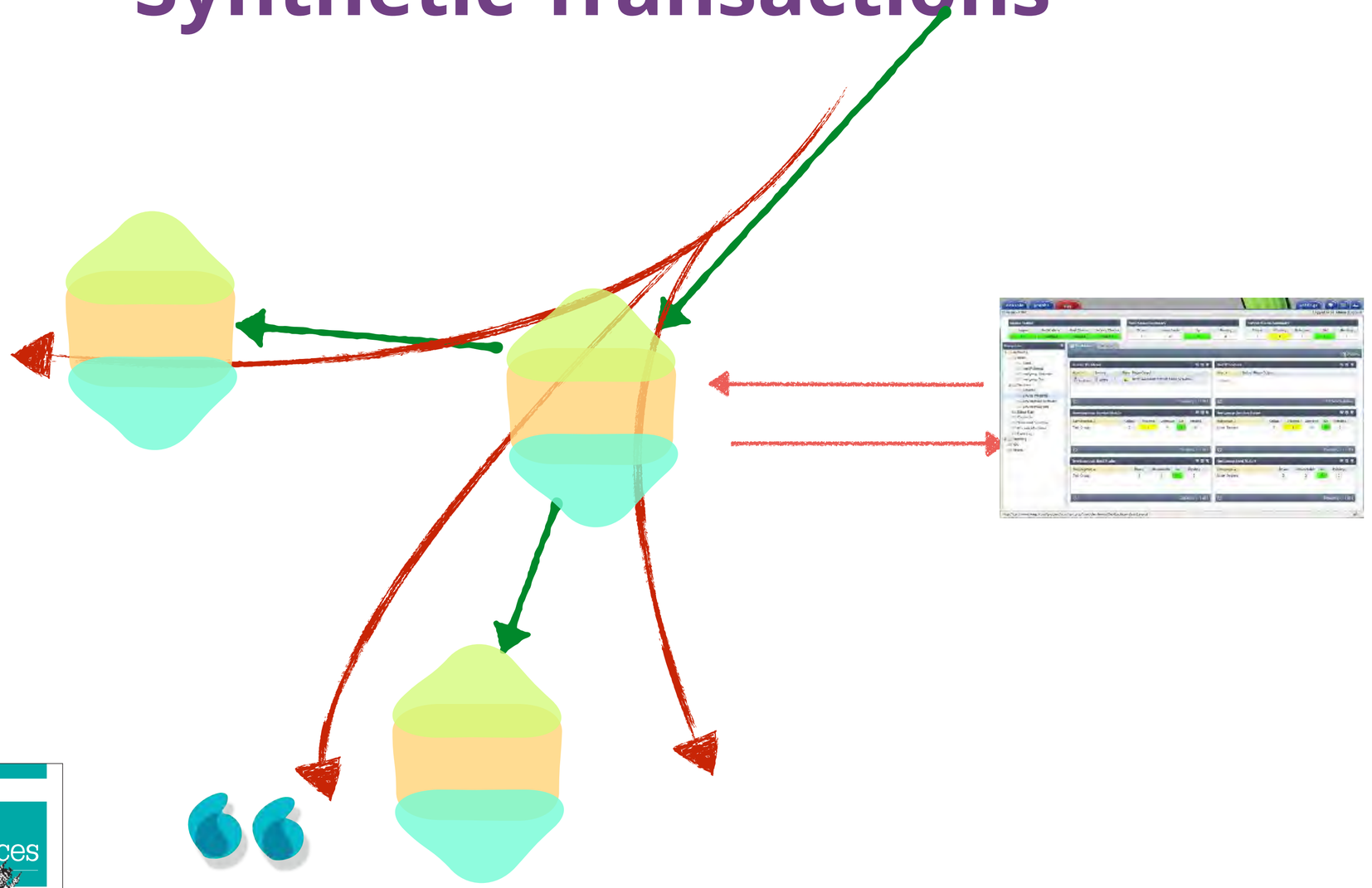


monitoring



logging

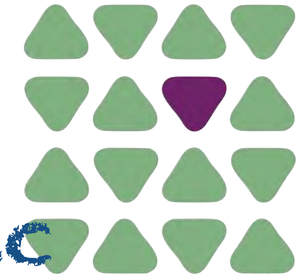
Synthetic Transactions



Use synthetic transactions to test production systems.

Fitness Function

atomic



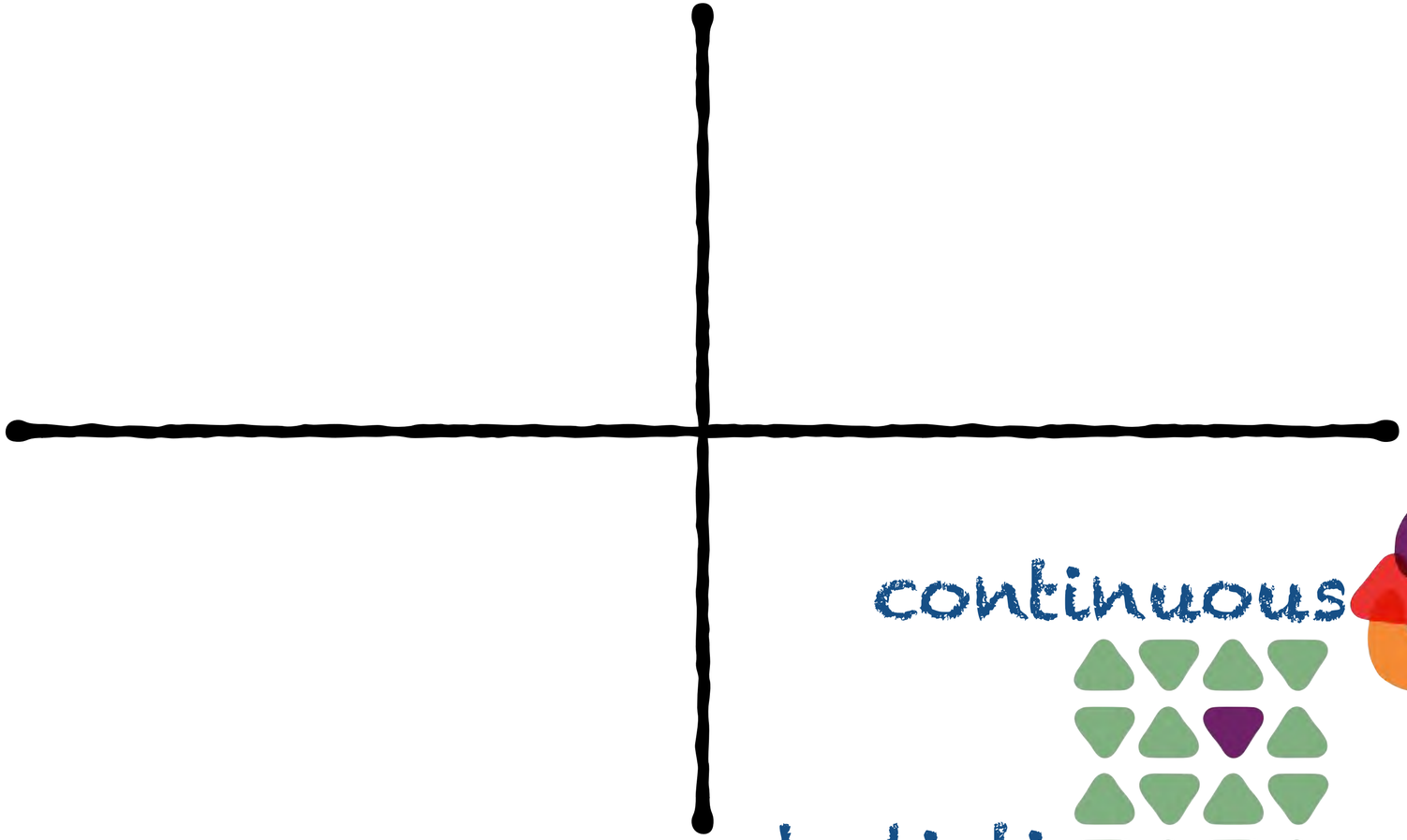
holistic

triggered

continuous

Fitness Function

atomic



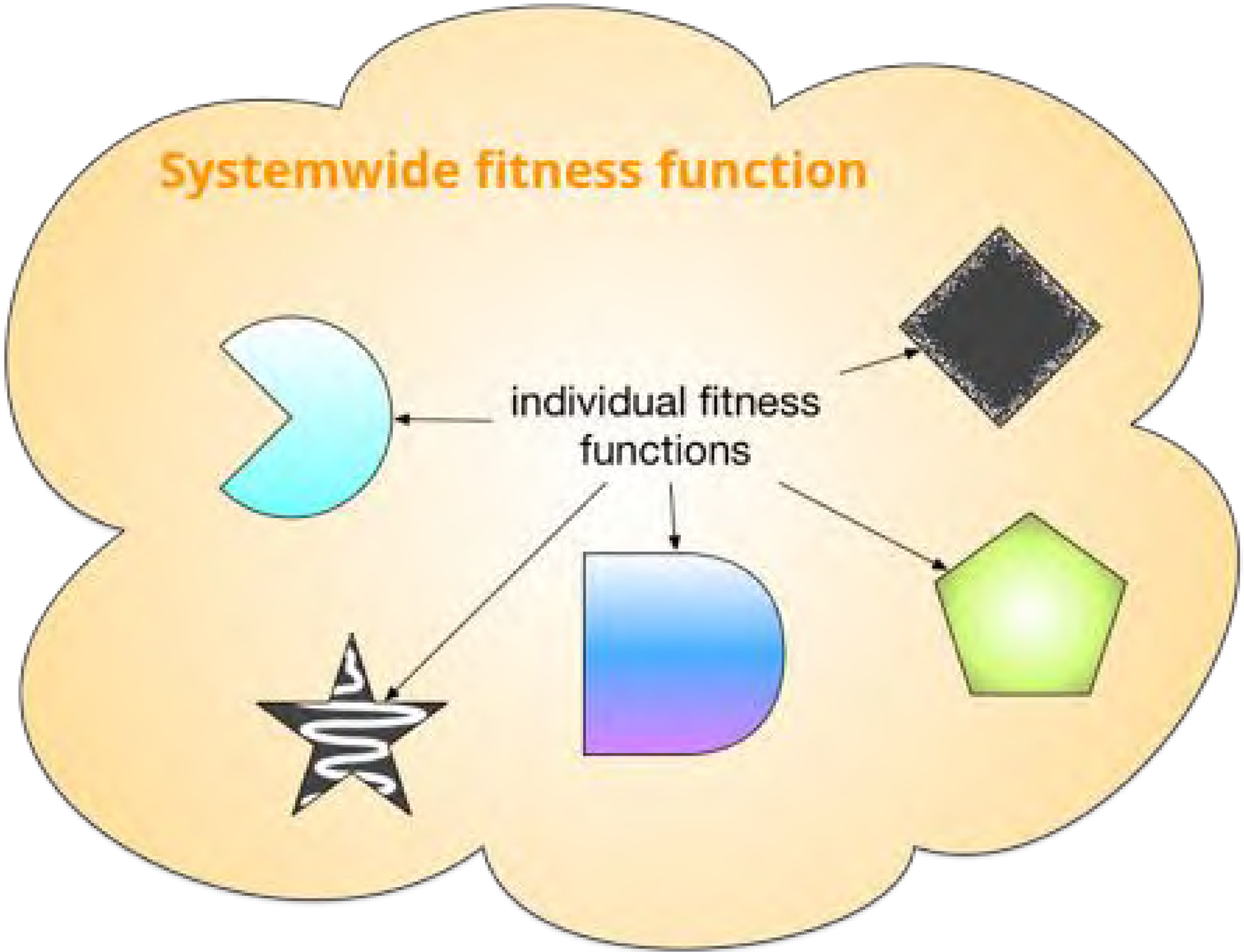
continuous

holistic

triggered

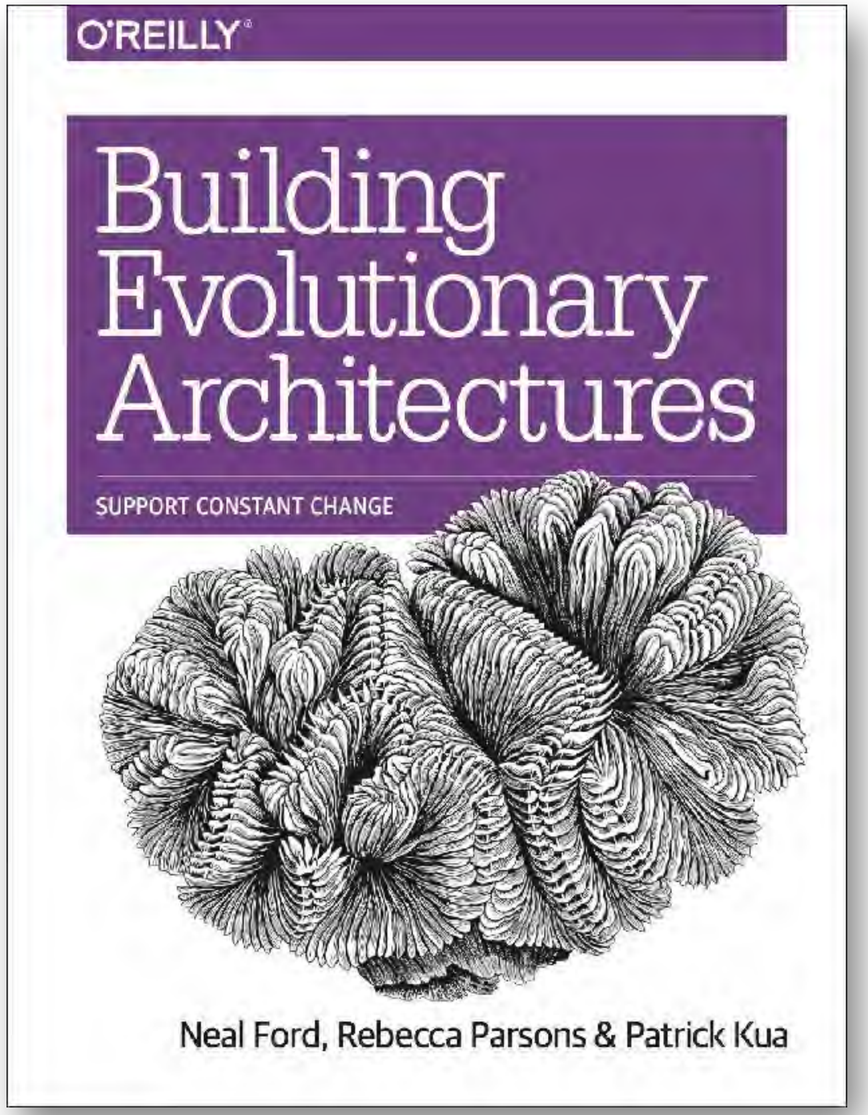

amazon
web services
CHAOS
MONKEY

System-wide Fitness Function



#OSCON

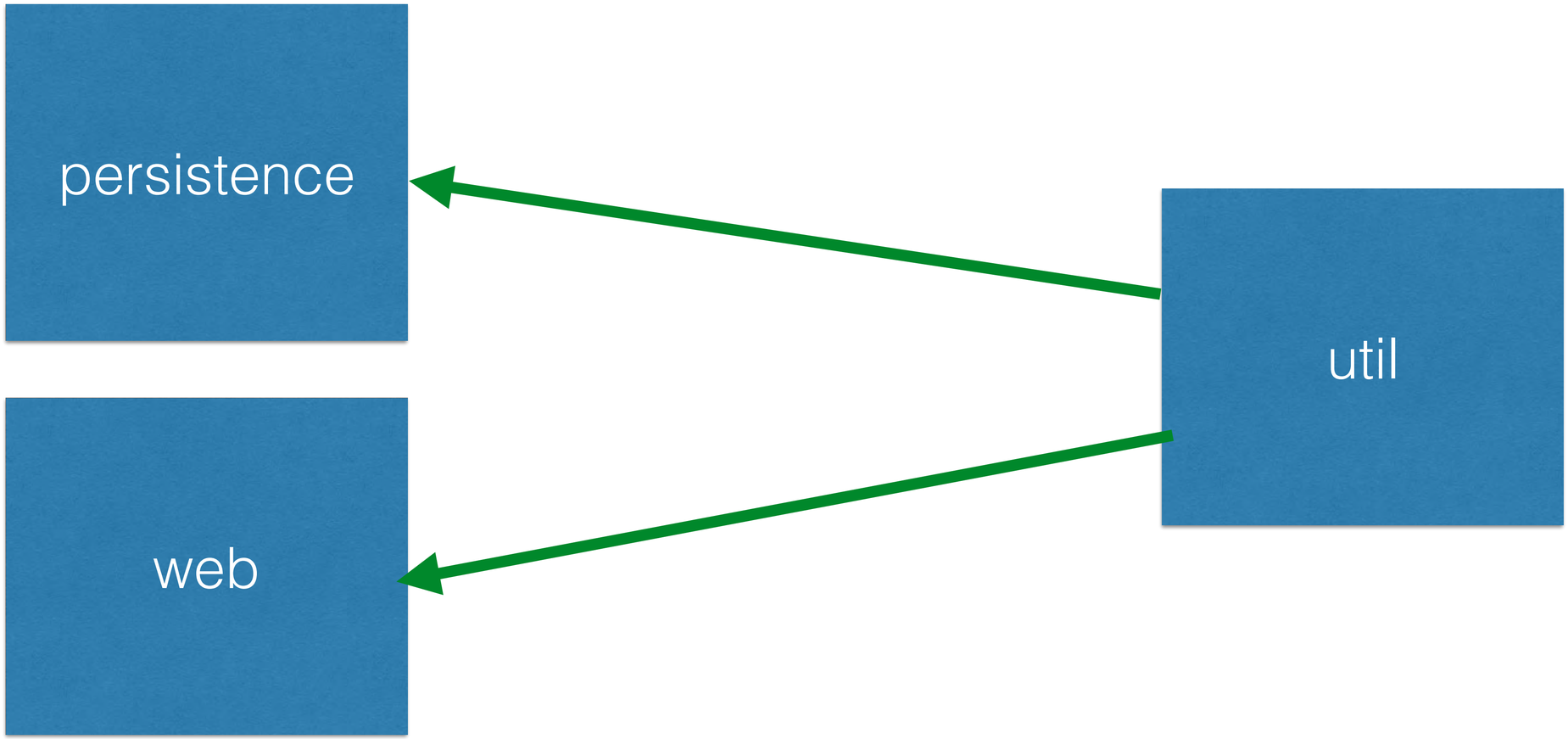
Building Evolutionary Architectures



Fitness Functions:

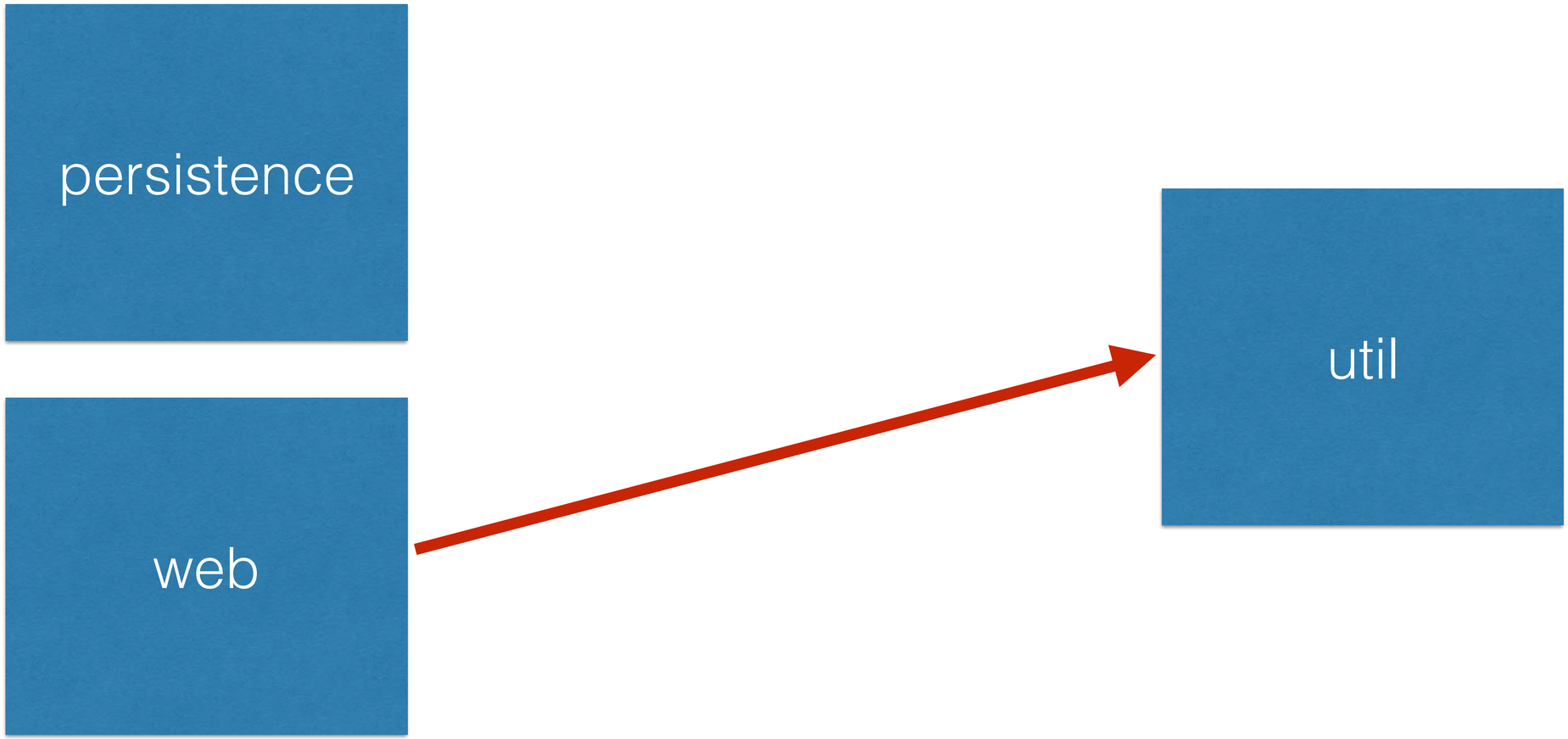
testing and automation

Testable



packages/namespaces

Testable

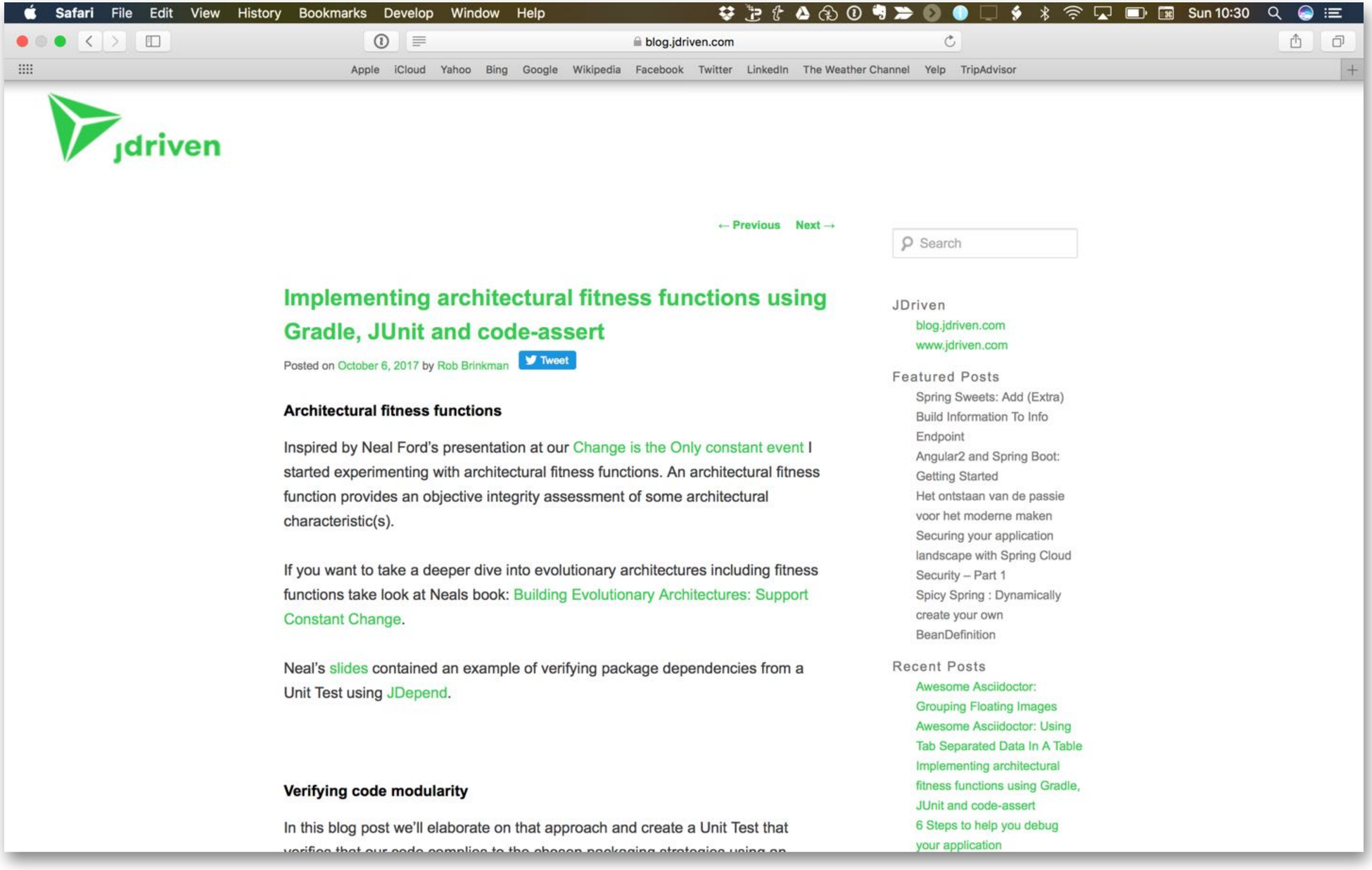


packages/namespaces

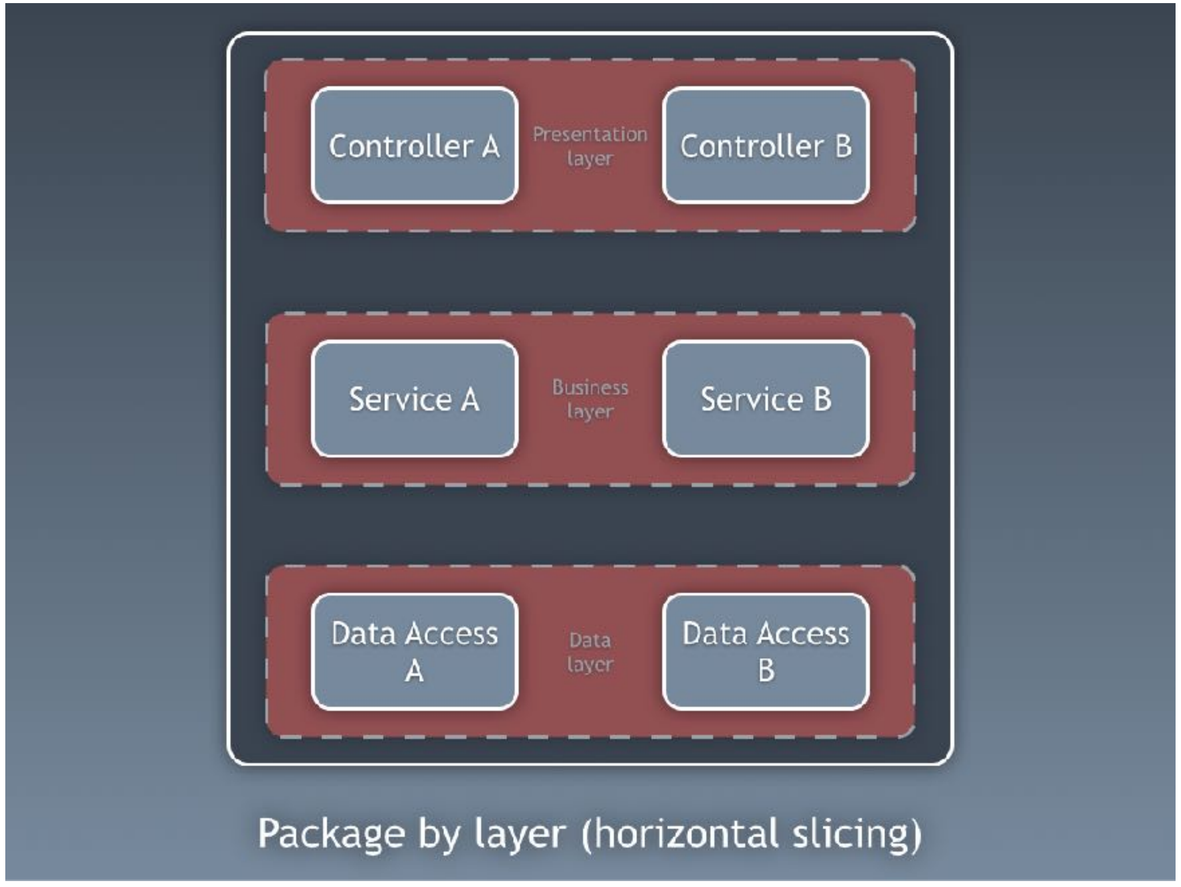
Testable

```
public void testMatch() {
    DependencyConstraint constraint = new DependencyConstraint();

    JavaPackage persistence = constraint.addPackage("com.xyz.persistence");
    JavaPackage web = constraint.addPackage("com.xyz.web");
    JavaPackage util = constraint.addPackage("com.xyz.util");

    persistence.dependsUpon(util);
    web.dependsUpon(util);

    jdepend.analyze();

    assertEquals("Dependency mismatch",
            true, jdepend.dependencyMatch(constraint));
    }
```

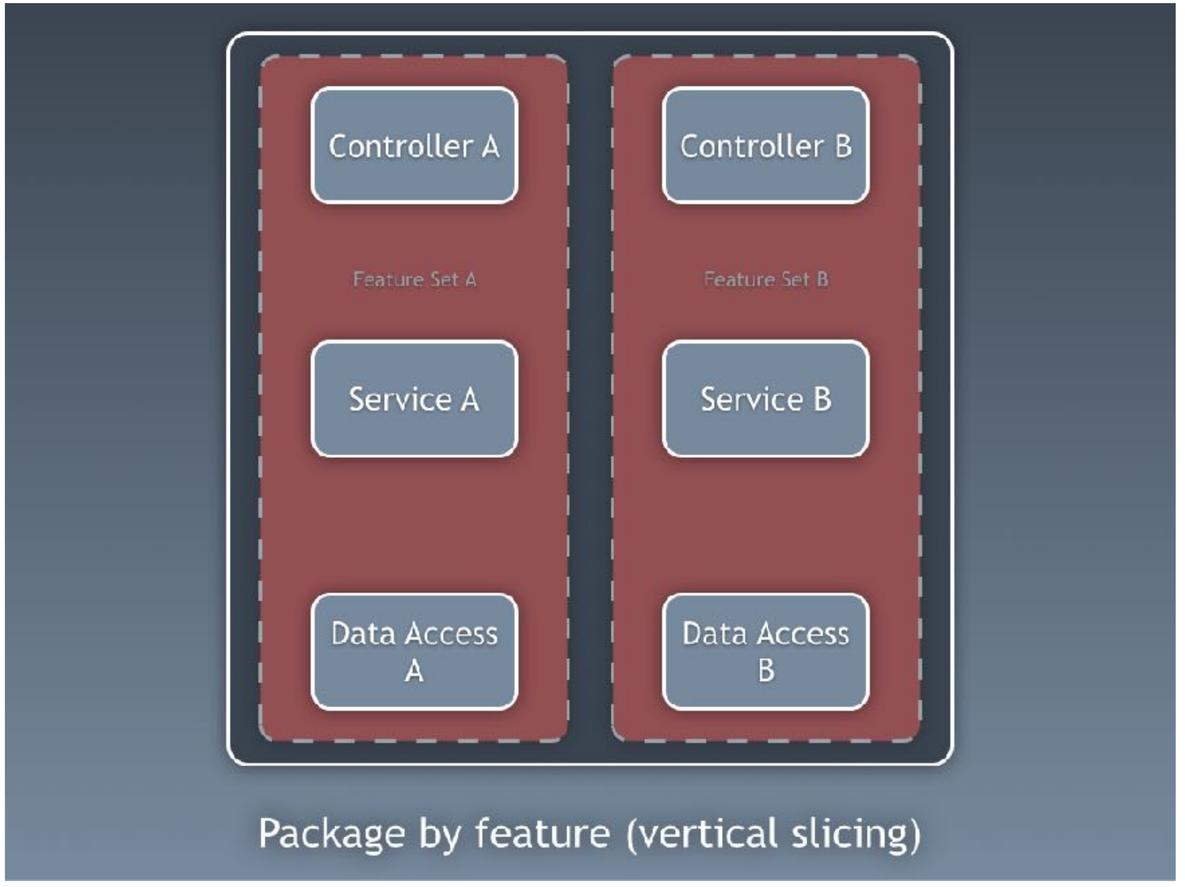
https://github.com/clarkware/jdepend/



https://blog.jdriven.com/2017/10/implementing-architectural-fitness-functions-using-gradle-junit-code-assert/



Package by layer (horizontal slicing)



Package by feature (vertical slicing)

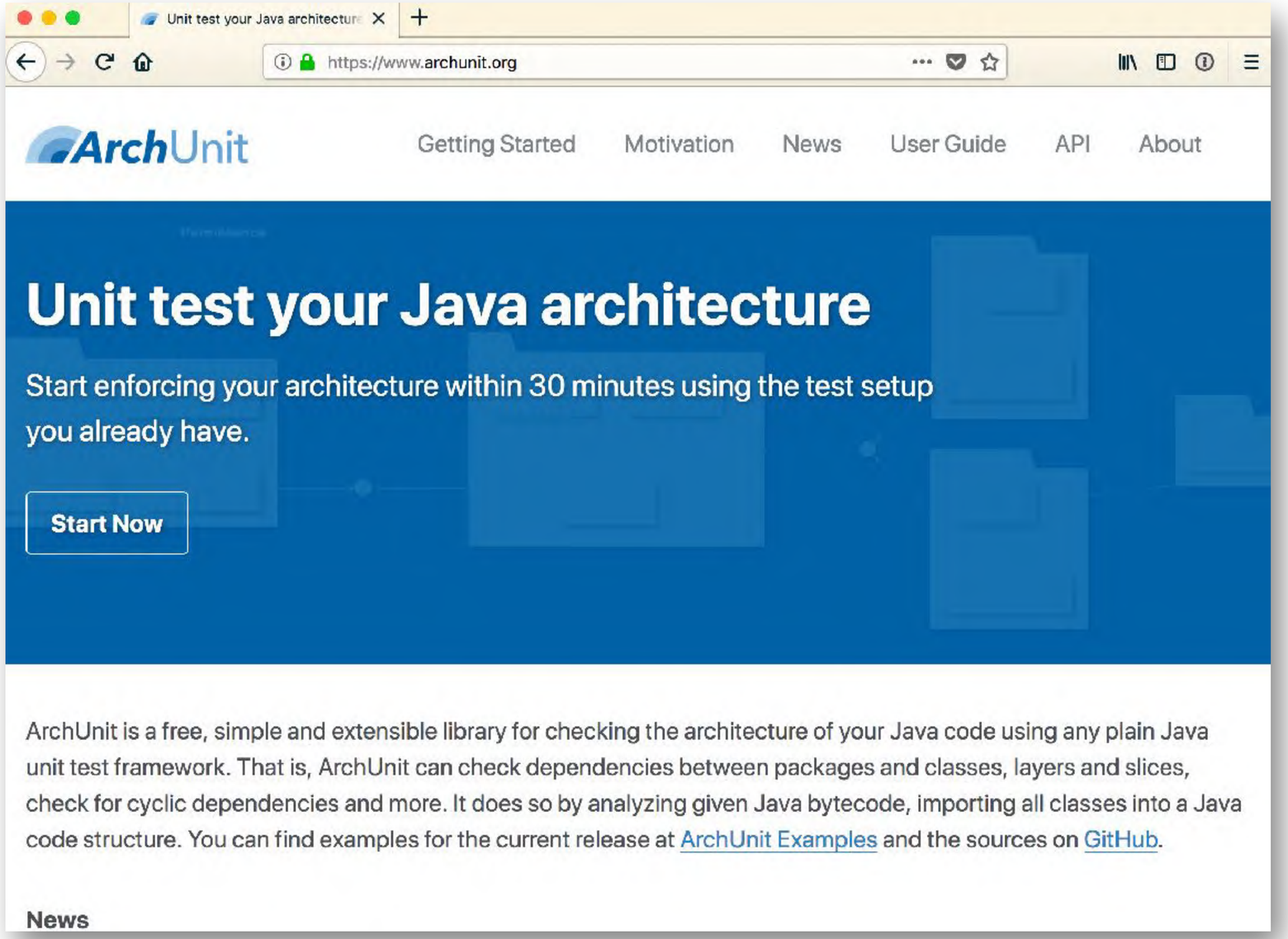
```java
public class VerifyPackageByLayerTest {

    @Test
    public void verifyPackageByLayer() {

        /// Create an analyzer config for the package we'd like to verify
        AnalyzerConfig analyzerConfig = GradleAnalyzerConfig.gradle().main("com.jdriven.fitness.packaging.by.layer");

        // Dependency rules for Packaging by Layer
        // NOTE: the classname should match the packagename
        class ComJdrivenFitnessPackagingByLayer extends DependencyRuler {

            // Rules for layer child packages
            // NOTE: they should match the name of the sub packages
            DependencyRule controller, service, repository;

            @Override
            public void defineRules() {
                // Our App classes depends on all subpackages because it constructs all of them
                base().mayUse(base().allSub());
                // Controllers may use Services
                controller.mayUse(service);
                // Services may use Repositories
                service.mayUse(repository);
            }
        }

        // All dependencies are forbidden, except the ones defined in ComJdrivenFitnessPackagingByLayer
        // java, org, net packages may be used freely
        DependencyRules rules = DependencyRules.denyAll()
                .withRelativeRules(new ComJdrivenFitnessPackagingByLayer())
                .withExternals("java.*", "org.*", "net.*");

        DependencyResult result = new DependencyAnalyzer(analyzerConfig).rules(rules).analyze();
        assertThat(result, matchesRulesExactly());
    }
}
```

```java
public class ControllerA {

    private final ServiceA serviceA;
    private final RepositoryA repositoryA;

    public ControllerA(ServiceA serviceA, RepositoryA repositoryA) {
        this.serviceA = serviceA;
        this.repositoryA = repositoryA;
    }
}
```

```
java.lang.AssertionError:
 Expected: Comply with rules
 but: DENIED com.jdriven.fitness.packaging.by.layer.controller -&gt;
 com.jdriven.fitness.packaging.by.layer.repository (by com.jdriven.fitness.packaging.by.layer.controller.ControllerA)
```

ArchUnit

https://www.archunit.org/



ArchUnit

https://www.archunit.org/

coding rules

```java
import static com.tngtech.archunit.lang.syntax.ArchRuleDefinition.noClasses;
import static com.tngtech.archunit.library.GeneralCodingRules.ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS;
import static com.tngtech.archunit.library.GeneralCodingRules.NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING;

public class CodingRulesTest {
    private JavaClasses classes;

    @Before
    public void setUp() throws Exception {
        classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);
    }

    @Test
    public void classes_should_not_access_standard_streams_defined_by_hand() {
        noClasses().should(ACCESS_STANDARD_STREAMS).check(classes);
    }

    @Test
    public void classes_should_not_access_standard_streams_from_library() {
        NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS.check(classes);
    }

    @Test
    public void classes_should_not_throw_generic_exceptions() {
        NO_CLASSES_SHOULD_THROW_GENERIC_EXCEPTIONS.check(classes);
    }

    @Test
    public void classes_should_not_use_java_util_logging() {
        NO_CLASSES_SHOULD_USE_JAVA_UTIL_LOGGING.check(classes);
    }
}
```

ArchUnit

https://www.archunit.org/

```java
public class InterfaceRules {

    @Test
    public void interfaces_should_not_have_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
                SomeBusinessInterface.class,
                SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveNameMatching(".*Interface").check(classes);
    }

    @Test
    public void interfaces_should_not_have_simple_class_names_ending_with_the_word_interface() {
        JavaClasses classes = new ClassFileImporter().importClasses(
                SomeBusinessInterface.class,
                SomeDao.class
        );

        noClasses().that().areInterfaces().should().haveSimpleNameContaining("Interface").check(classes);
    }

    @Test
    public void interfaces_must_not_be_placed_in_implementation_packages() {
        JavaClasses classes = new ClassFileImporter().importPackagesOf(SomeInterfacePlacedInTheWrongPackage.class);

        noClasses().that().resideInAPackage("..impl..").should().beInterfaces().check(classes);
    }
}
```

interface rules

ArchUnit

https://www.archunit.org/

```java
public class LayerDependencyRulesTest {
    private JavaClasses classes;

    @Before
    public void setUp() throws Exception {
        classes = new ClassFileImporter().importPackagesOf(ClassViolatingCodingRules.class);
    }

    @Test
    public void services_should_not_access_controllers() {
        noClasses().that().resideInAPackage("..service..")
                .should().accessClassesThat().resideInAPackage("..controller..").check(classes);
    }

    @Test
    public void persistence_should_not_access_services() {
        noClasses().that().resideInAPackage("..persistence..")
                .should().accessClassesThat().resideInAPackage("..service..").check(classes);
    }

    @Test
    public void services_should_only_be_accessed_by_controllers_or_other_services() {
        classes().that().resideInAPackage("..service..")
                .should().onlyBeAccessed().byAnyPackage("..controller..", "..service..").check(classes);
    }
}
```

layer dependency

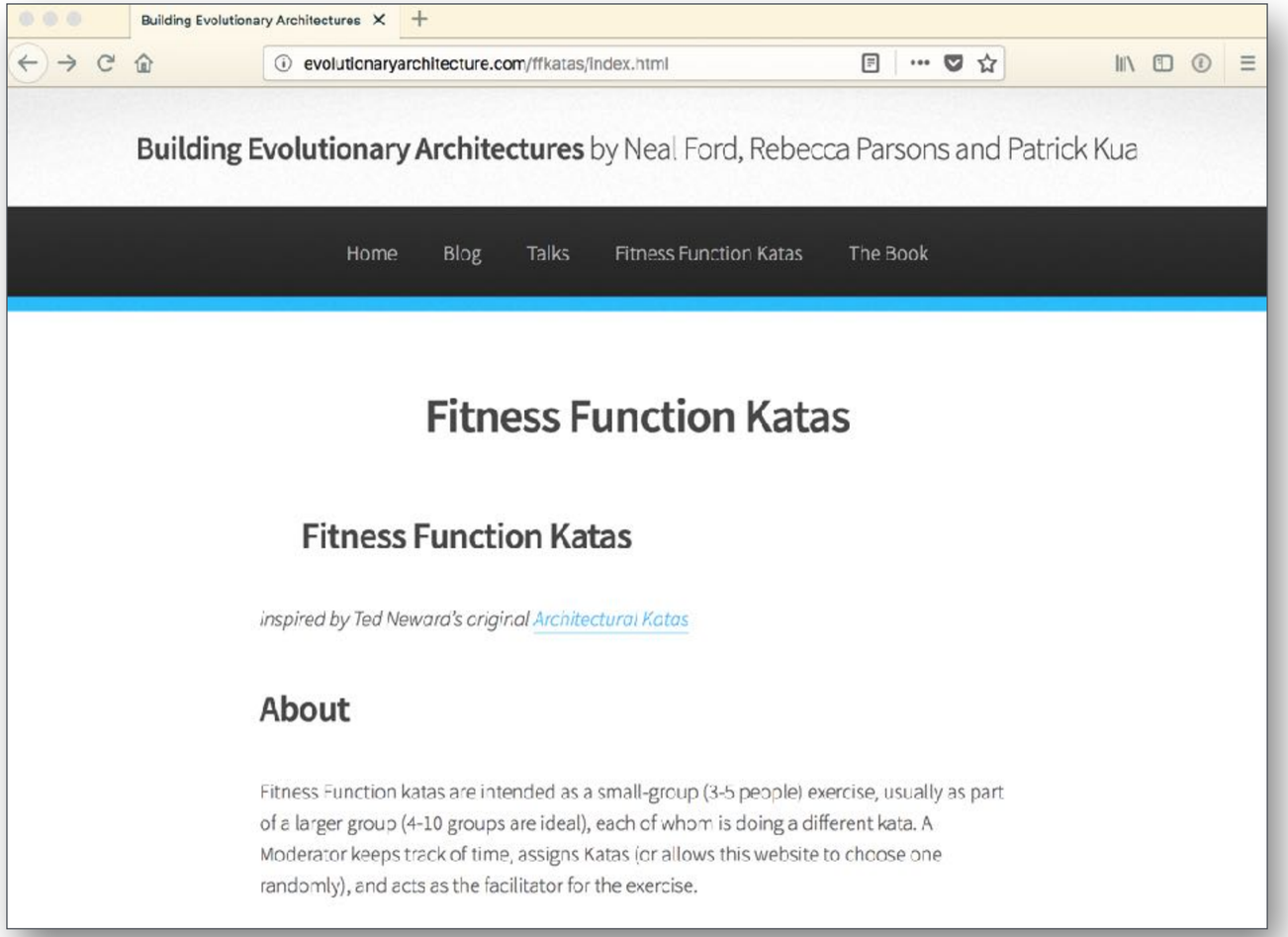
ArchUnit

https://www.archunit.org/

```
@Test
public void third_party_class_should_only_be_instantiated_via_workaround() {
    classes().should(notCreateProblematicClassesOutsideOfWorkaroundFactory()
            .as(THIRD_PARTY_CLASS_RULE_TEXT))
            .check(classes);
}

private ArchCondition<JavaClass> notCreateProblematicClassesOutsideOfWorkaroundFactory() {
    DescribedPredicate<JavaCall<?>> constructorCallOfThirdPartyClass =
            target(is(constructor())).and(targetOwner(is(assignableTo(ThirdPartyClassWithProblem.class))));

    DescribedPredicate<JavaCall<?>> notFromWithinThirdPartyClass =
            originOwner(is(not(assignableTo(ThirdPartyClassWithProblem.class)))).forSubType();

    DescribedPredicate<JavaCall<?>> notFromWorkaroundFactory =
            originOwner(is(not(equivalentTo(ThirdPartyClassWorkaroundFactory.class)))).forSubType();

    DescribedPredicate<JavaCall<?>> targetIsIllegalConstructorOfThirdPartyClass =
            constructorCallOfThirdPartyClass.
                    and(notFromWithinThirdPartyClass).
                    and(notFromWorkaroundFactory);

    return never(callCodeUnitWhere(targetIsIllegalConstructorOfThirdPartyClass));
}
```

governance

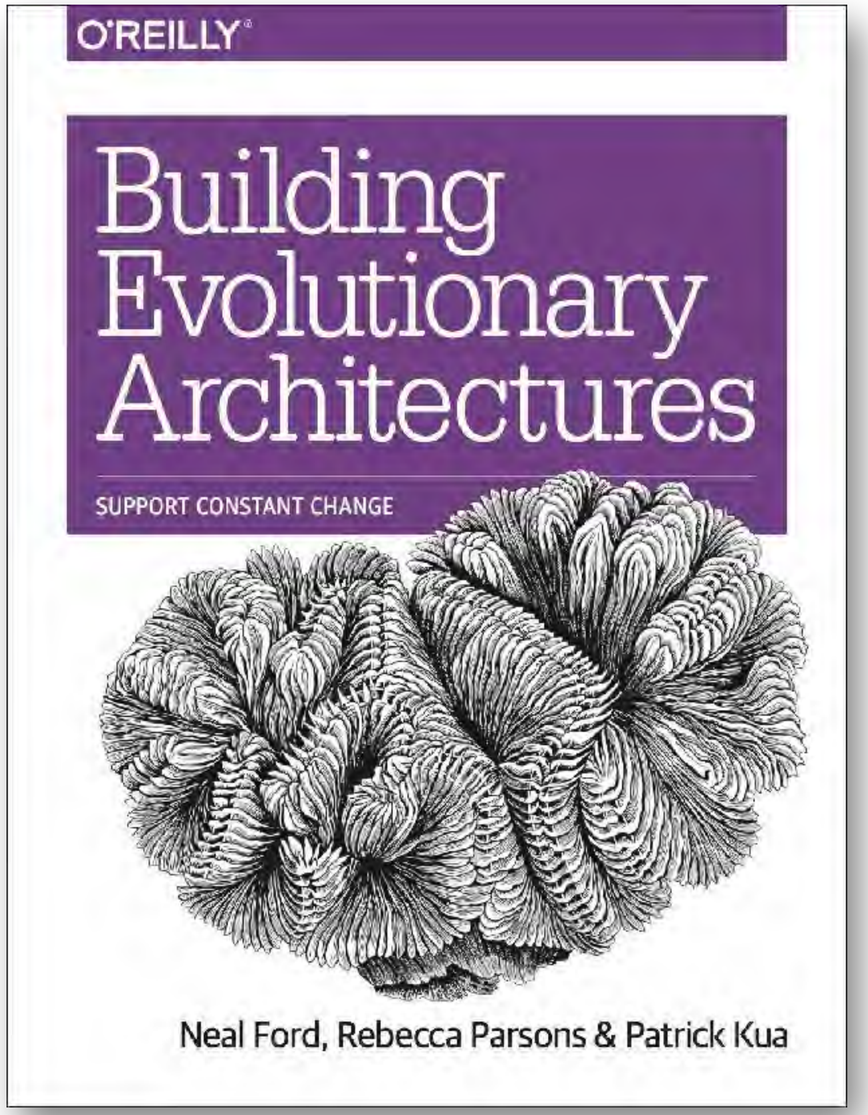
Fitness Function Katas



http://evolutionaryarchitecture.com/ffkatas/

#OSCON

Building Evolutionary Architectures



Afternoon Tea

Mike Mason

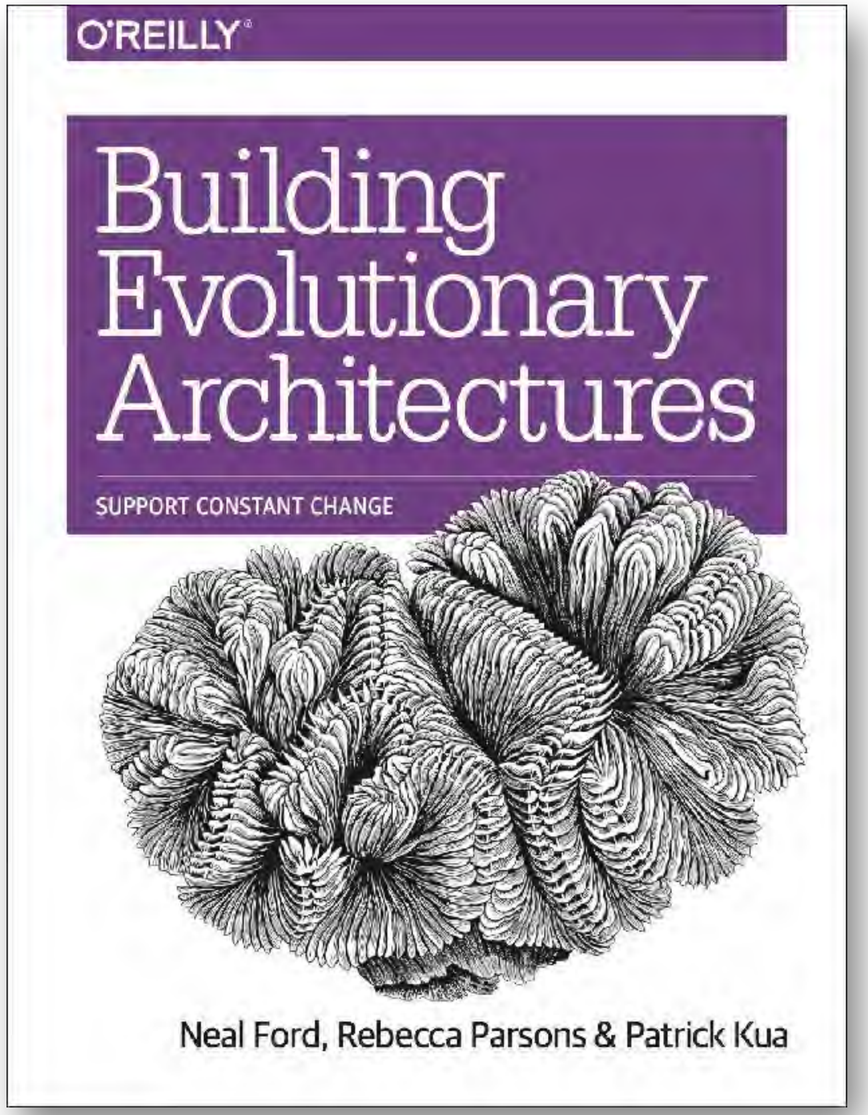
@mikemasonca

Zhamak Dehghani

@zhamakd

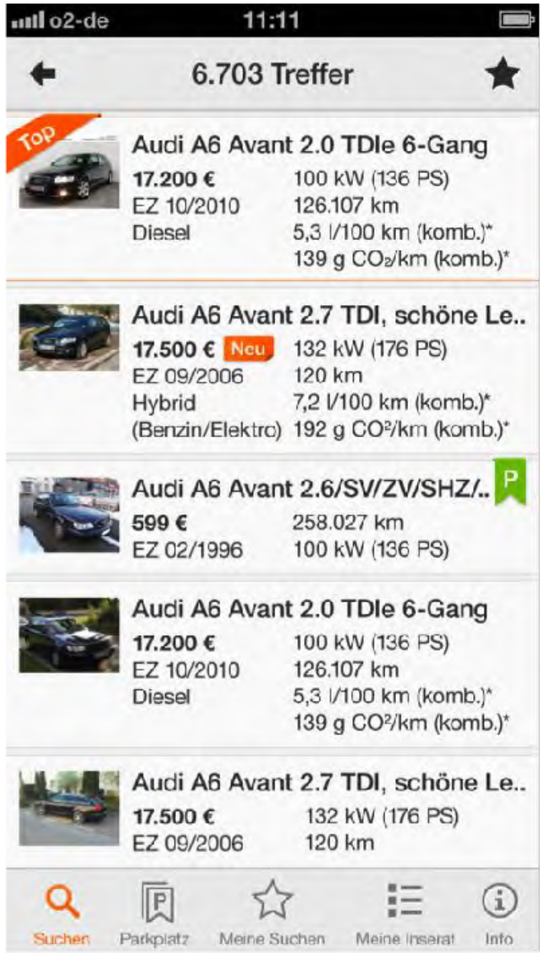
#OSCON

Building Evolutionary Architectures

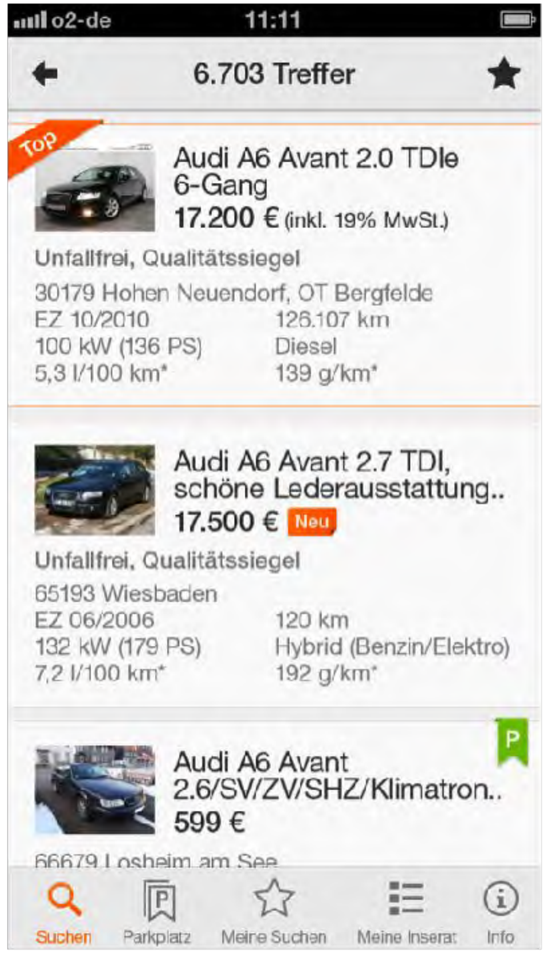


Hypothesis and Data-Driven Development

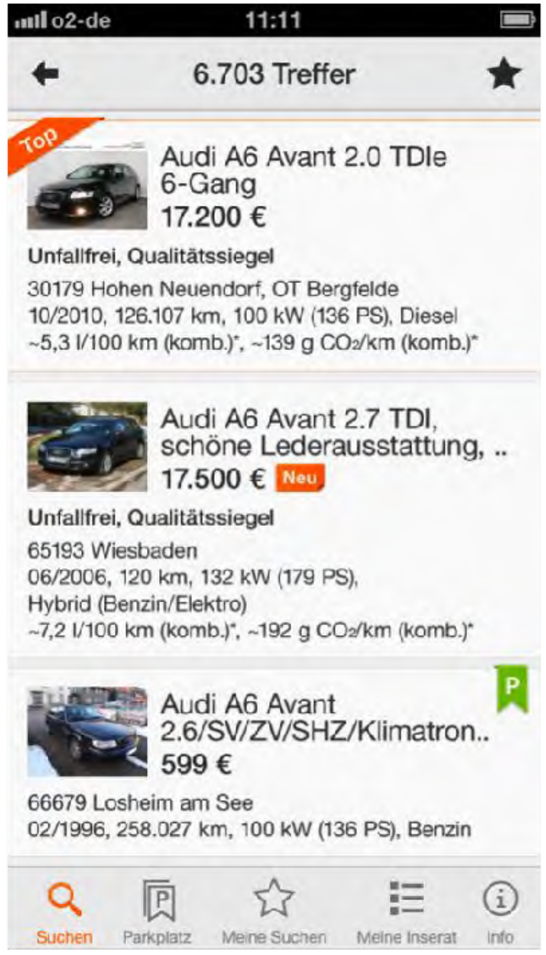
Experiments to Perform



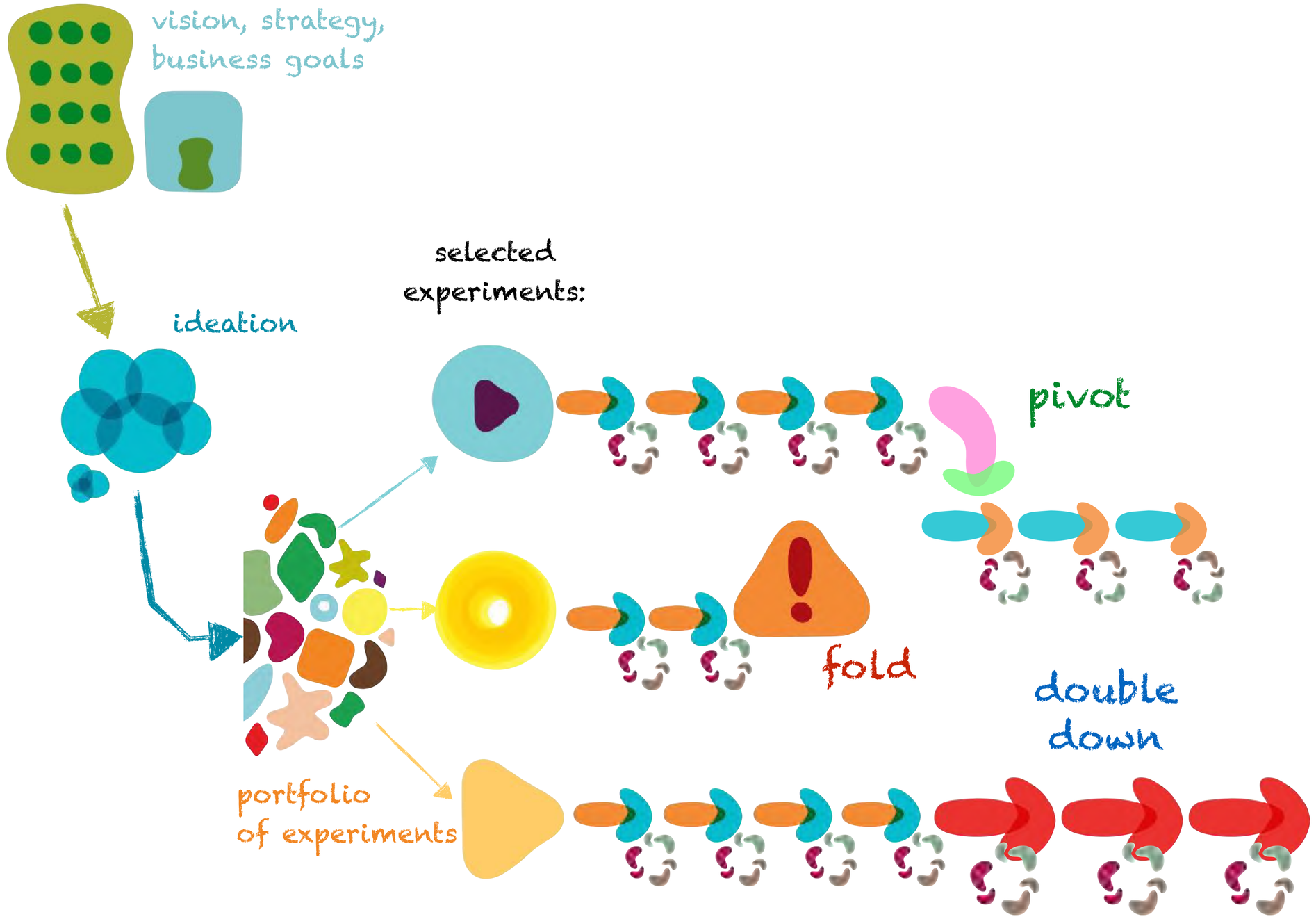
More Listings



Better Structure

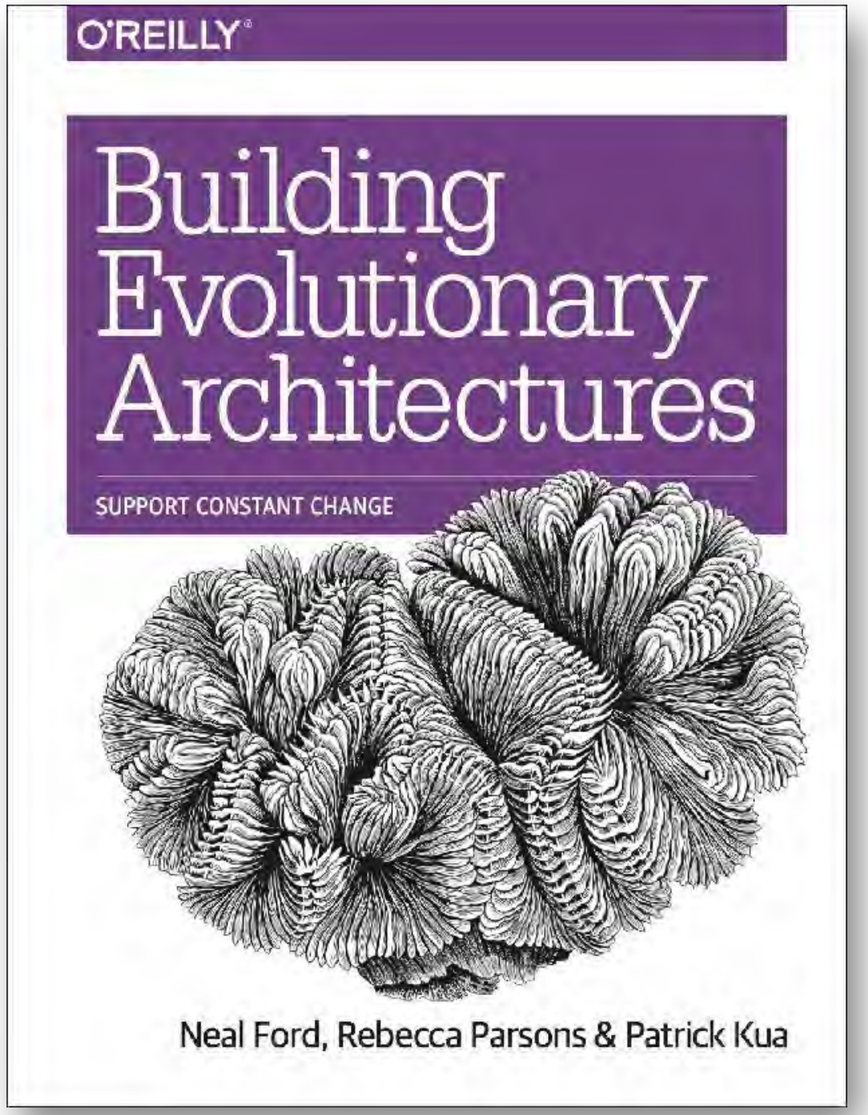


Better Prioritization

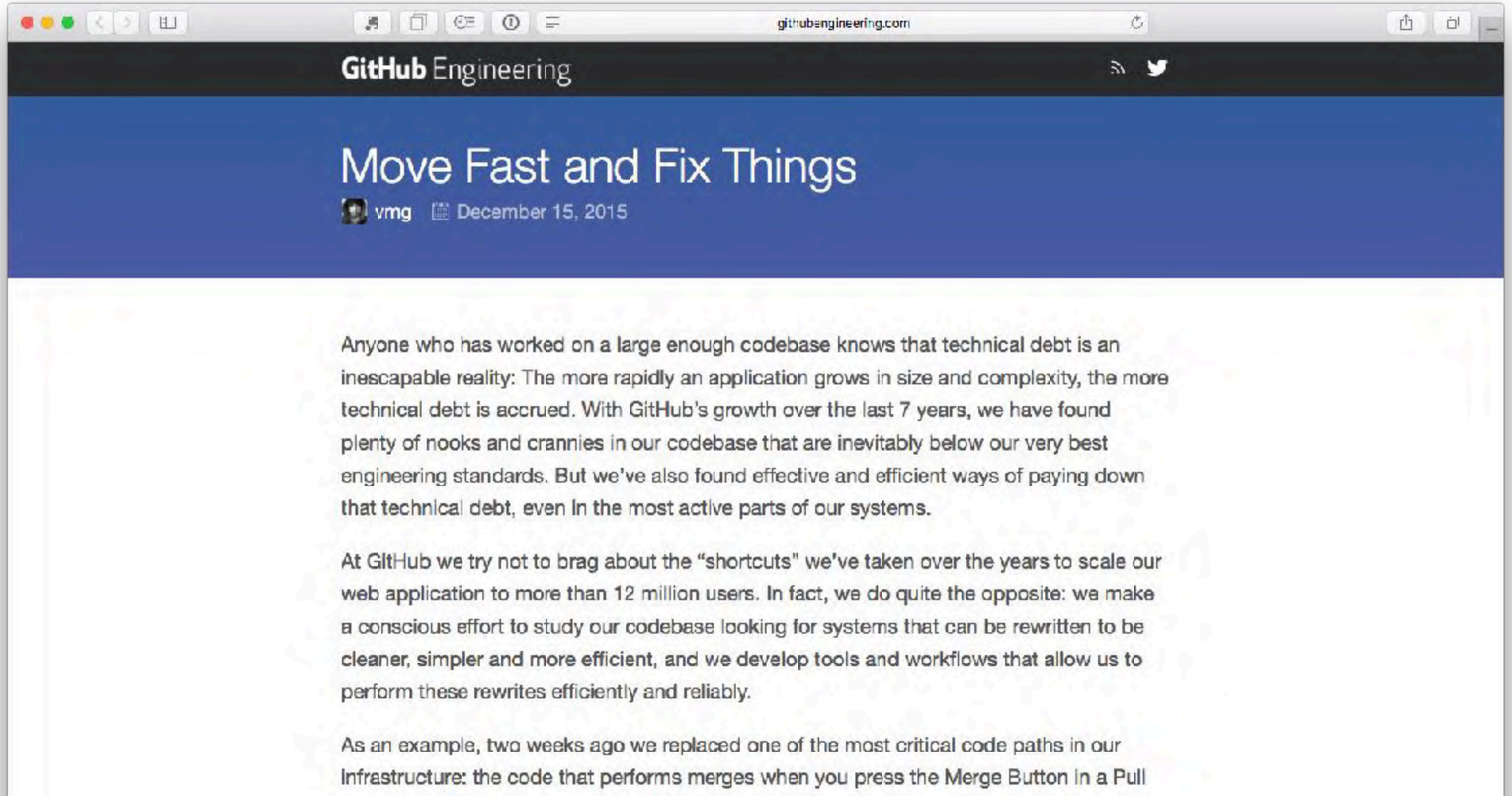

vision, strategy,
business goals
ideation
selected
experiments:
pivot
fold
double
down
portfolio
of experiments

#OSCON

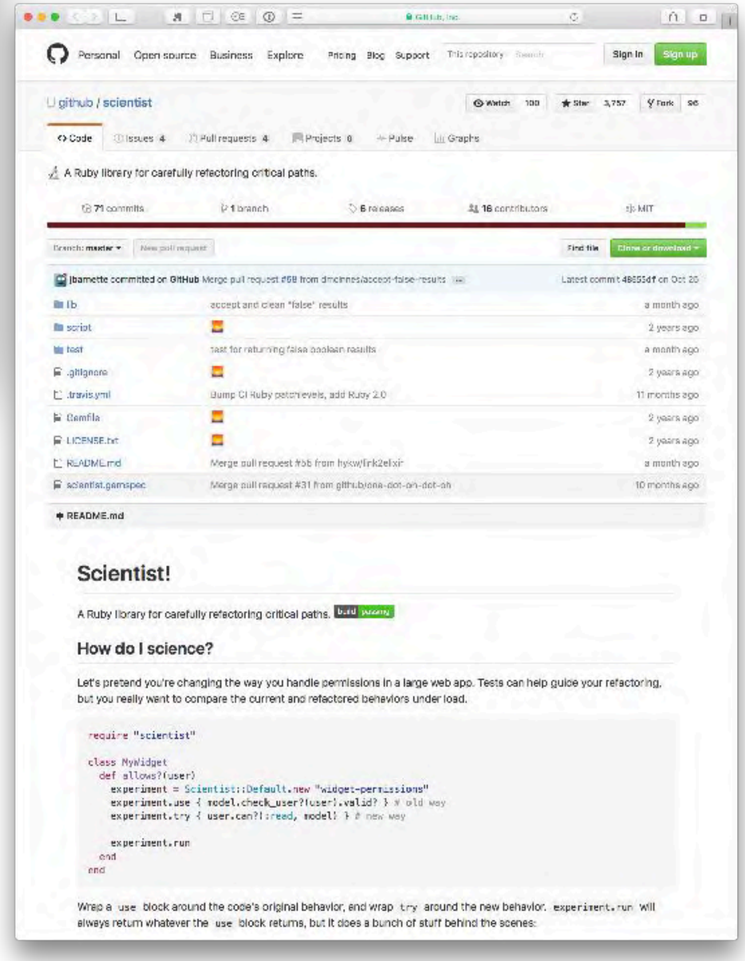
Building Evolutionary Architectures



Move Fast and Fix Things




```ruby
def create_merge_commit(base, head, author, commit_message)
  base = resolve_commit(base)
  head = resolve_commit(head)
  commit_message = Rugged.prettify_message(commit_message)

  merge_base = rugged.merge_base(base, head)
  return [nil, "already_merged"] if merge_base == head.oid

  ancestor_tree = merge_base && Rugged::Commit.lookup(rugged, merge_base).tree
  merge_options = {
    :fail_on_conflict => true,
    :skip_reuc => true,
    :no_recursive => true,
  }
  index = base.tree.merge(head.tree, ancestor_tree, merge_options)
  return [nil, "merge_conflict"] if (index.nil? || index.conflicts?)

  options = {
    :message    => commit_message,
    :committer  => author,
    :author     => author,
    :parents    => [base, head],
    :tree       => index.write_tree(rugged)
  }

  [Rugged::Commit.create(rugged, options), nil]
end
```




```ruby
def create_merge_commit(author, base, head, options = {})
  commit_message = options[:commit_message] || "Merge #{head} into #{base}"
  now = Time.current

  science "create_merge_commit" do |e|
    e.context :base => base.to_s, :head => head.to_s, :repo => repository.nwo
    e.use { create_merge_commit_git(author, now, base, head, commit_message) }
    e.try { create_merge_commit_rugged(author, now, base, head, commit_message) }
  end
end
```

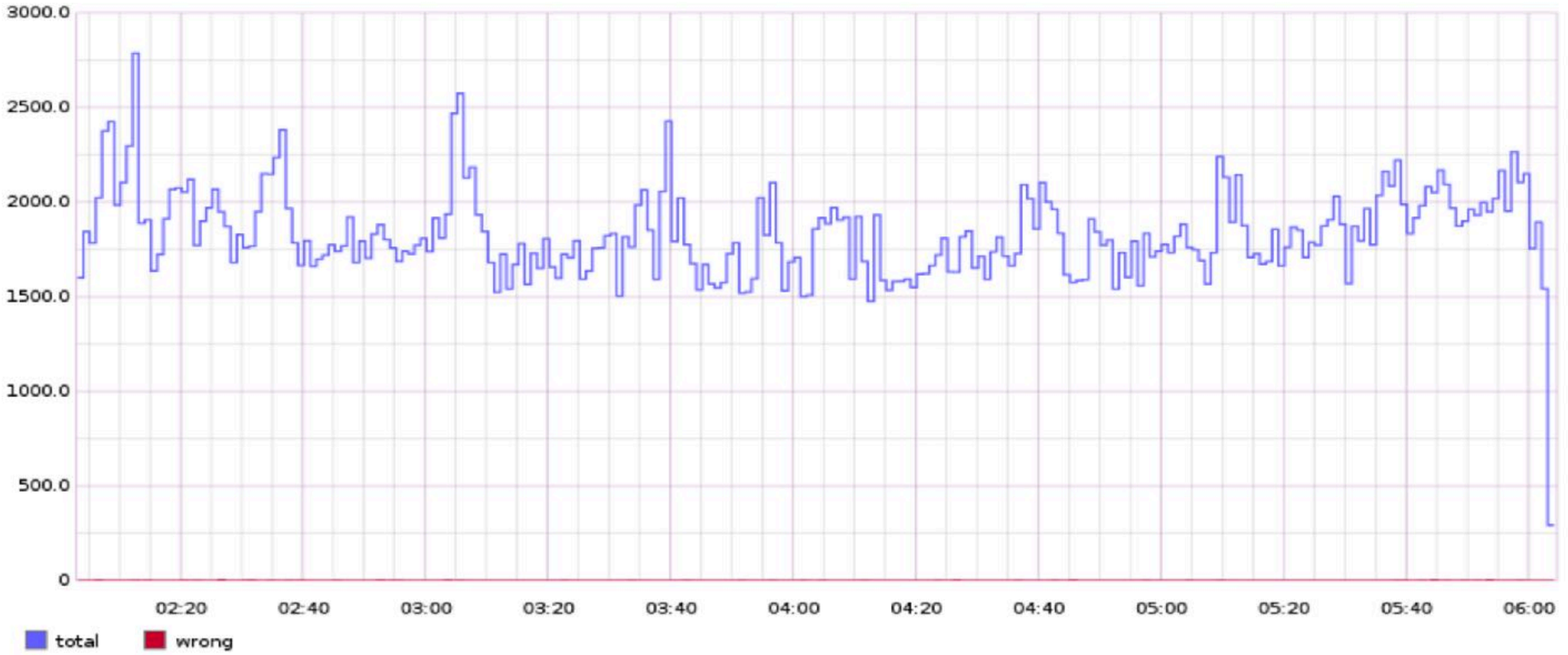
https://github.com/github/scientist

```ruby
require "scientist"

class MyWidget
  def allows?(user)
    experiment = Scientist::Default.new "widget-permissions"
    experiment.use { model.check_user?(user).valid? } # old way
    experiment.try { user.can?(:read, model) } # new way

    experiment.run
  end
end
```

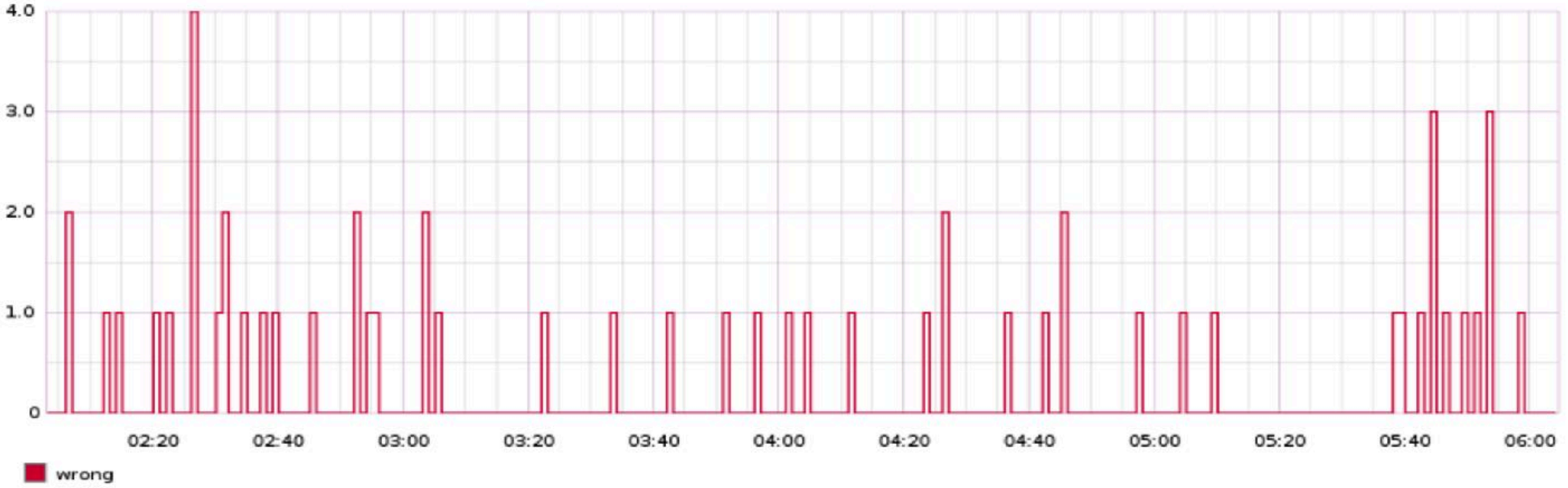
- It decides whether or not to run the try block,
- Randomizes the order in which use and try blocks are run,
- Measures the durations of all behaviors,
- Compares the result of try to the result of use,
- Swallows (but records) any exceptions raised in the try block
- Publishes all this information.

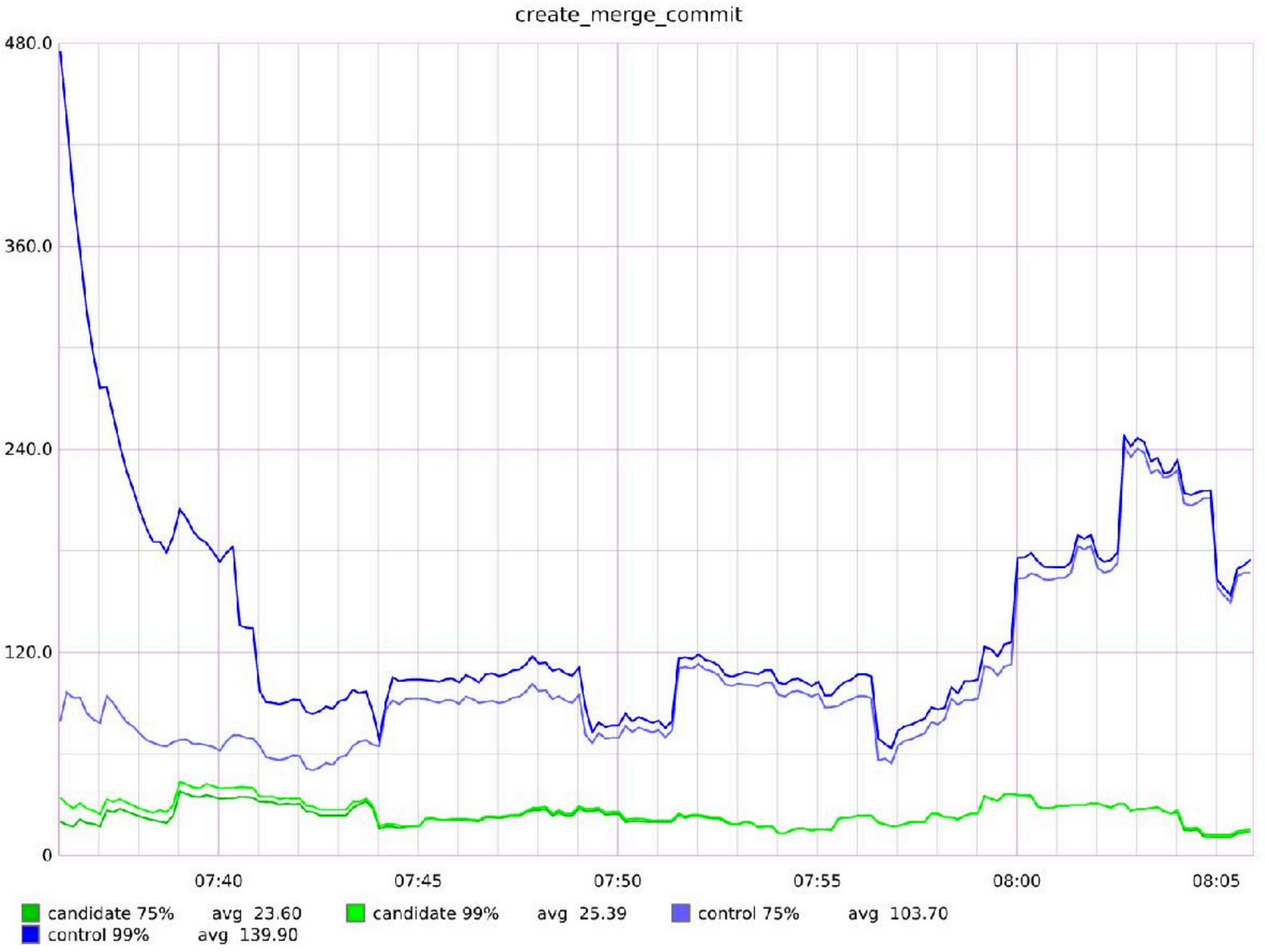
Accuracy

The number of times that the candidate and the control agree or disagree. View mismatches



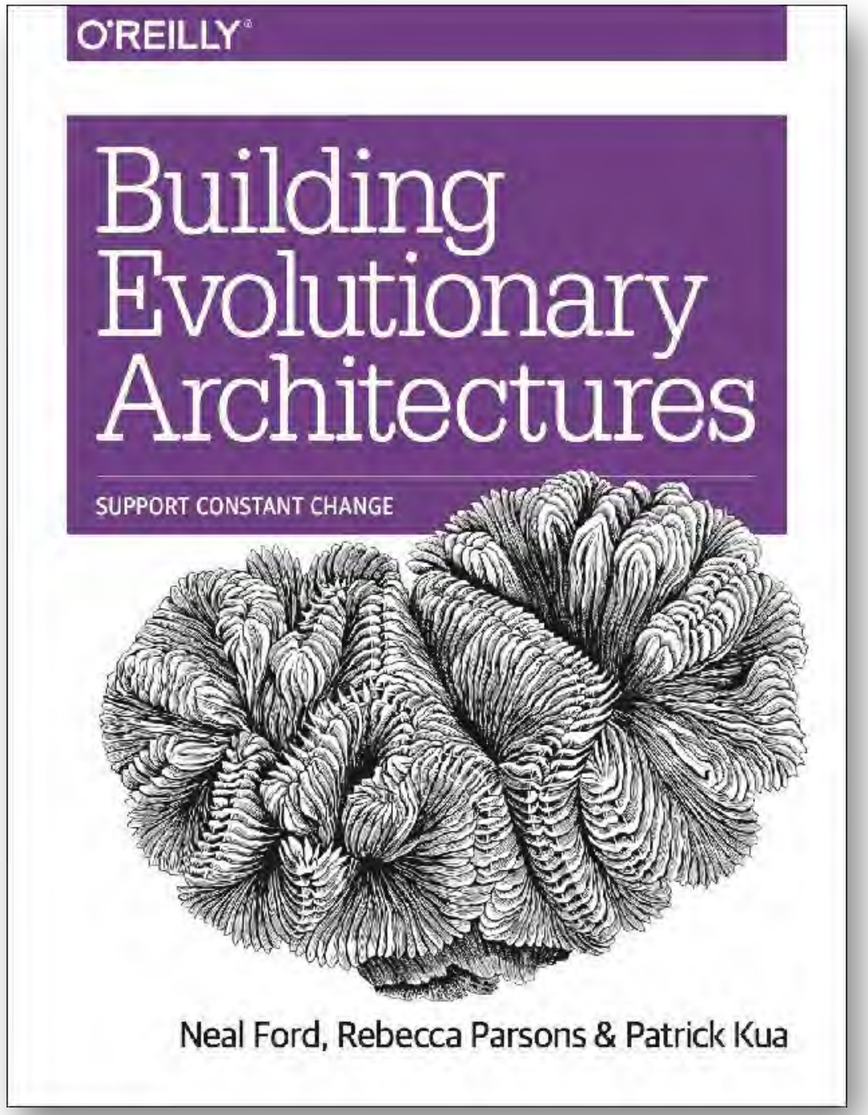
The number of incorrect/ignored only.



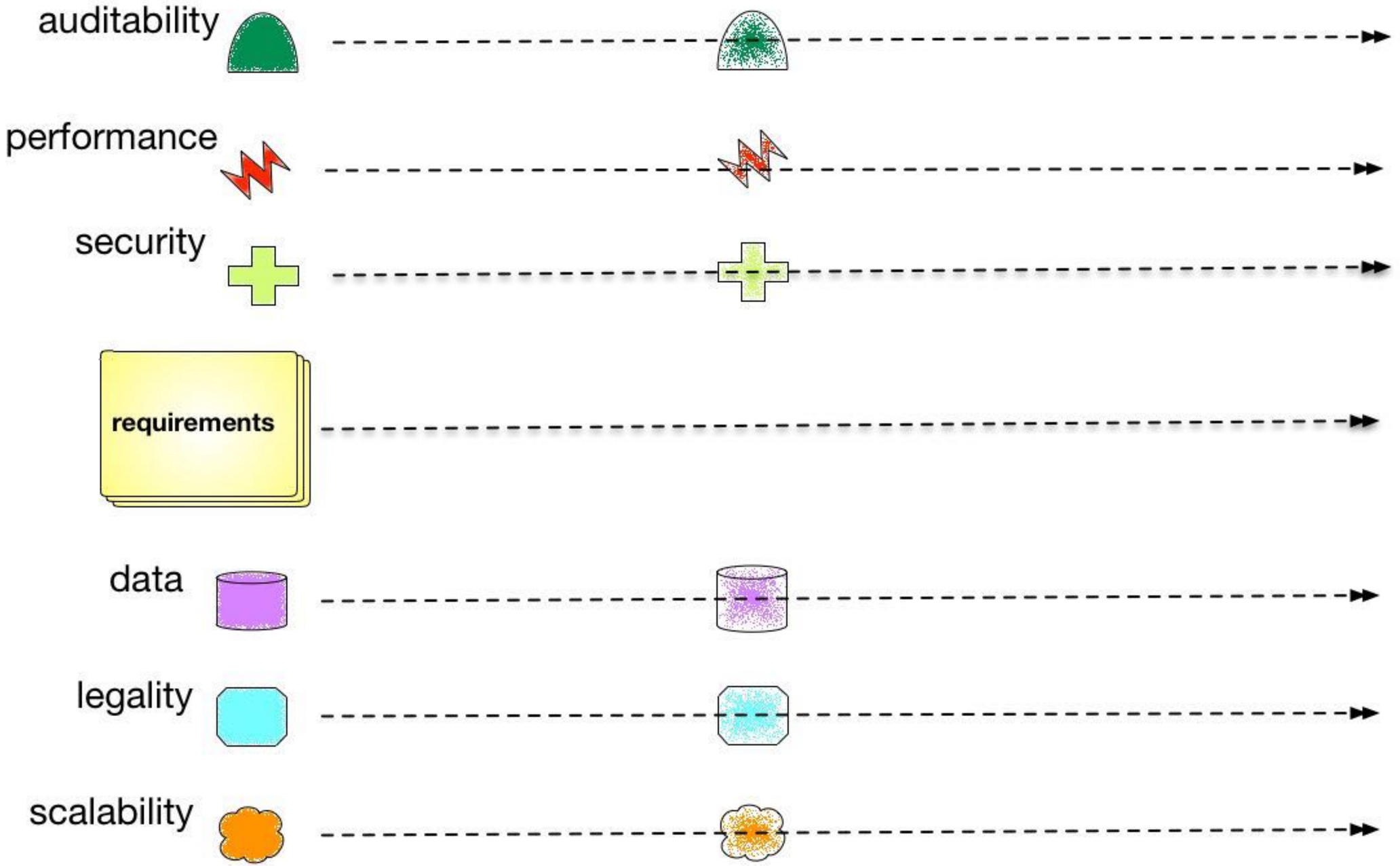

create_merge_commit
480.0
360.0
240.0
120.0
0
07:40
07:45
07:50
07:55
08:00
08:05
candidate 75% avg 23.60
candidate 99% avg 25.39
control 75% avg 103.70
control 99% avg 139.90

#OSCON

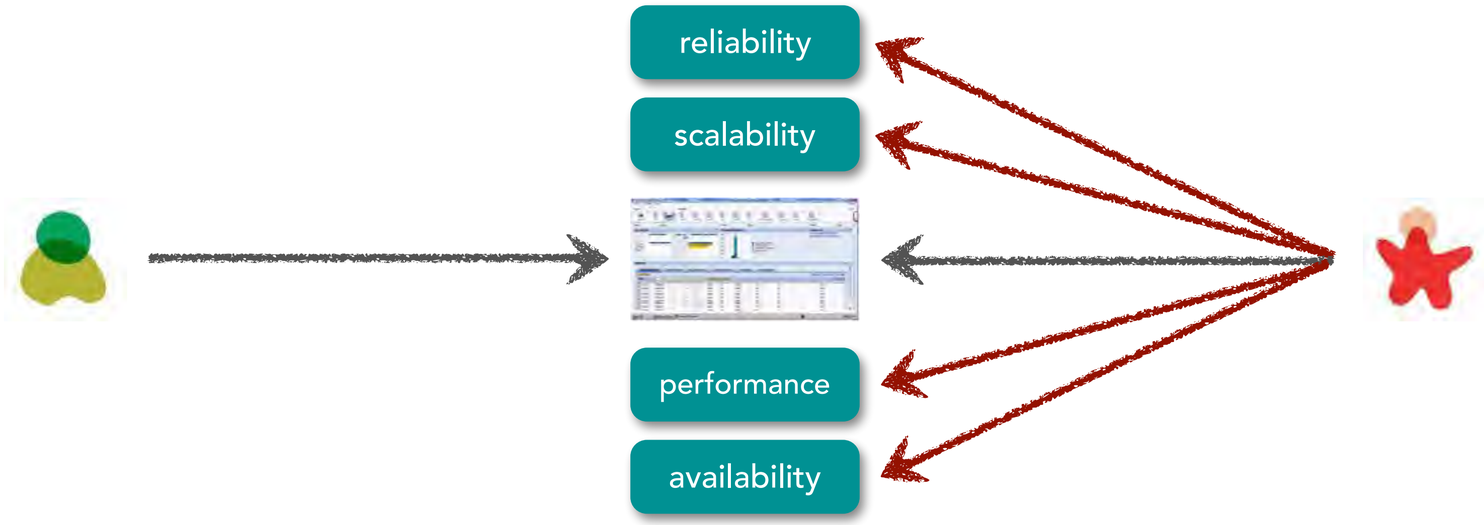
Building Evolutionary Architectures



Architectural Characteristics


auditability
performance
security
requirements
data
legality
scalability

Architecture Characteristics



Architecture Characteristics

accessibility
accountability
accuracy
adaptability
administrability
affordability
agility
auditability
autonomy
availability
compatibility
composability
configurability
correctness
credibility
customizability
debugability
degradability
determinability
demonstrability
dependability
deployability
discoverability
distributability
durability
effectiveness
efficiency
evolvability
extensibility
failure transparency
fault-tolerance
fidelity
flexibility
inspectability
installability
integrity
interchangeability
interoperability
learnability
maintainability
manageability
mobility
modifiability
modularity
operability
orthogonality
portability
precision
predictability
process capabilities
producibility
provability
recoverability
relevance
reliability
repeatability
reproducibility
resilience
responsiveness
reusability
robustness
safety
scalability
seamlessness
self-sustainability
serviceability
supportability
securability
simplicity
stability
standards compliance
survivability
sustainability
tailorability
testability
timeliness
traceability
transparency
ubiquity
understandability
upgradability
usability

https://en.wikipedia.org/wiki/List_of_system_quality_attributes

Architecture Characteristics

For each of the following business challenges, decide on the appropriate “ilities” that would be necessary in the system you learned about in the icebreaker. For each characteristic, how would you measure it in the example system?

Architecture Characteristics

"our business is constantly changing to meet new demands of the marketplace"

extensibility, maintainability, agility, modularity

Architecture Characteristics

"due to new regulatory requirements, it is imperative that we complete end-of-day processing in time"

performance, scalability, availability, reliability

Architecture Characteristics

"we need faster time to market to remain competitive"

maintainability, agility, modularity, deployability, testability

Architecture Characteristics

“our plan is to engage heavily in mergers and acquisitions in the next three years”

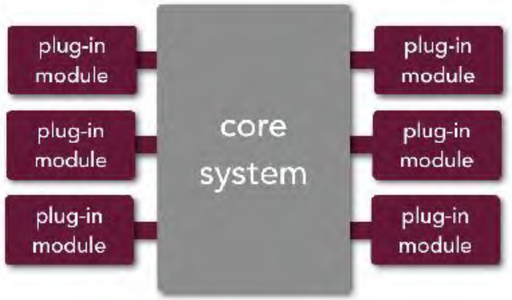
scalability, extensibility, openness, standards-based, agility, modularity

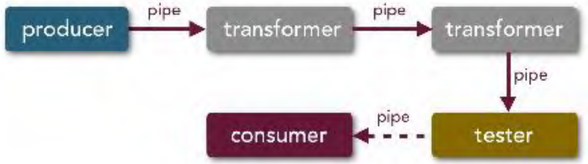
Architecture Characteristics

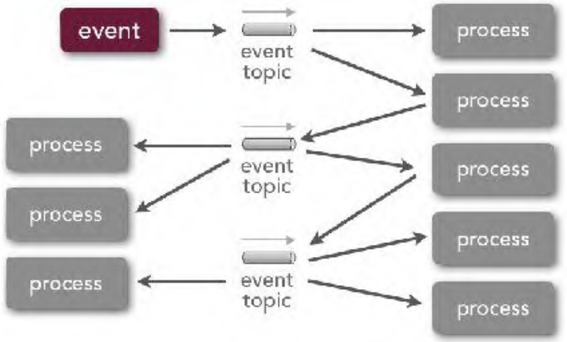
“we have a very tight timeframe and budget for this project”

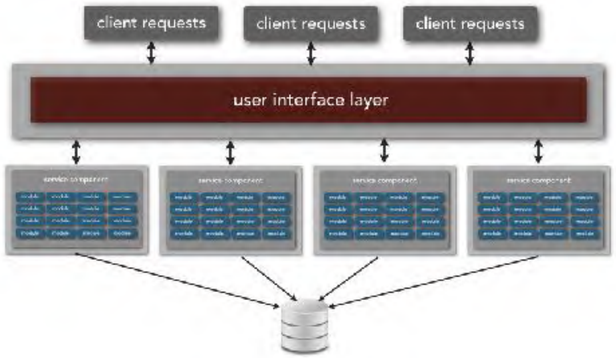
feasibility

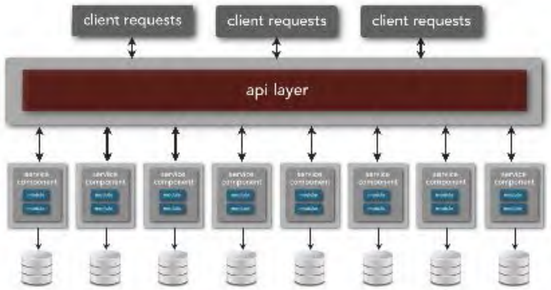
architecture patterns help define the basic characteristics and behavior of the application

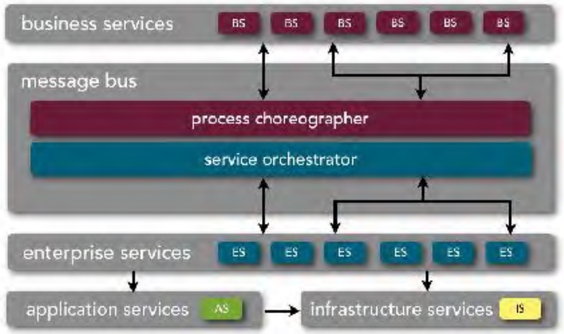


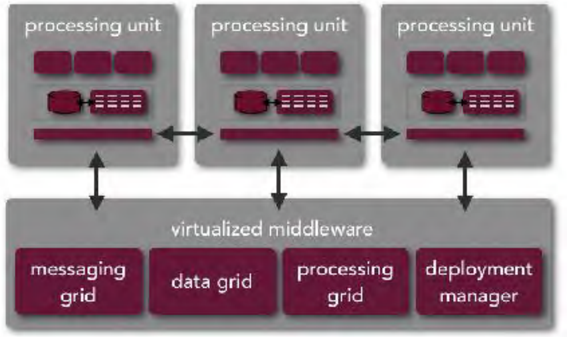




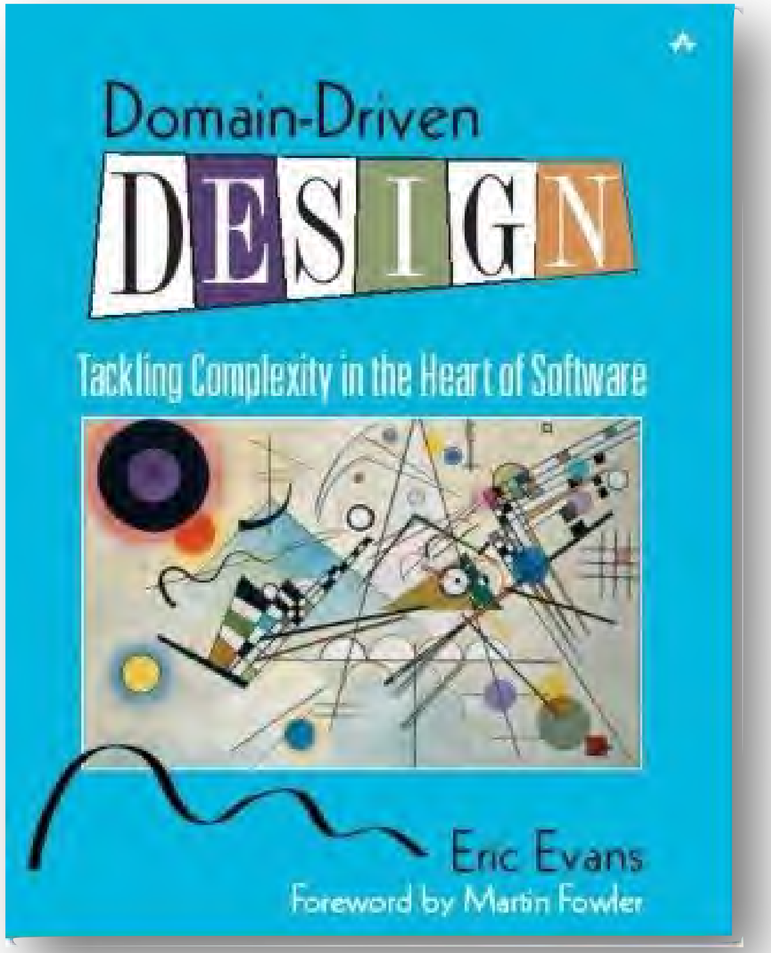


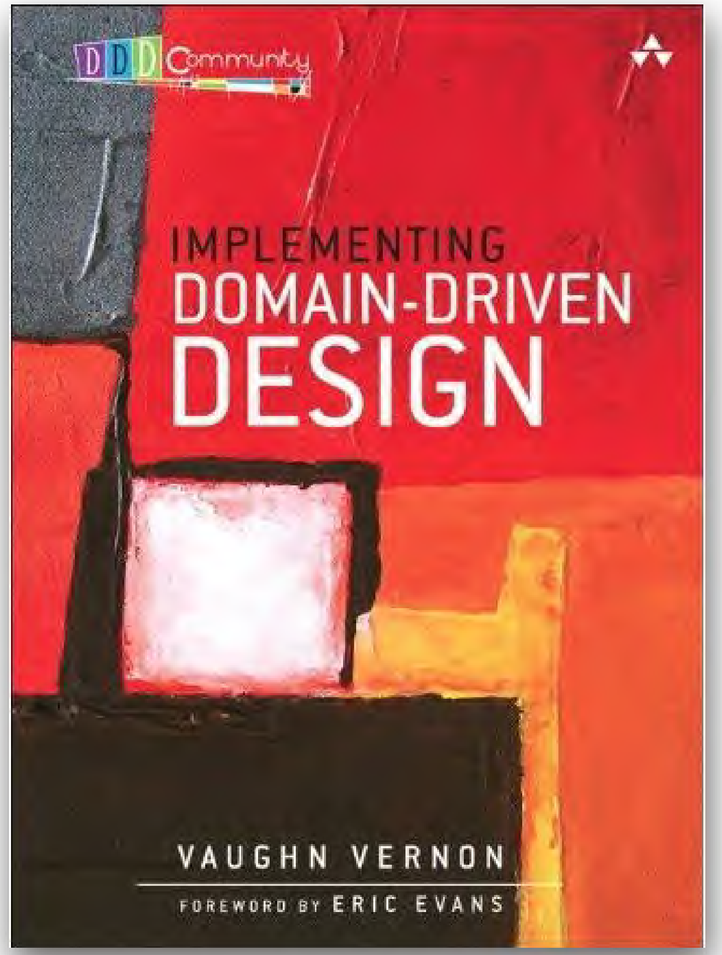






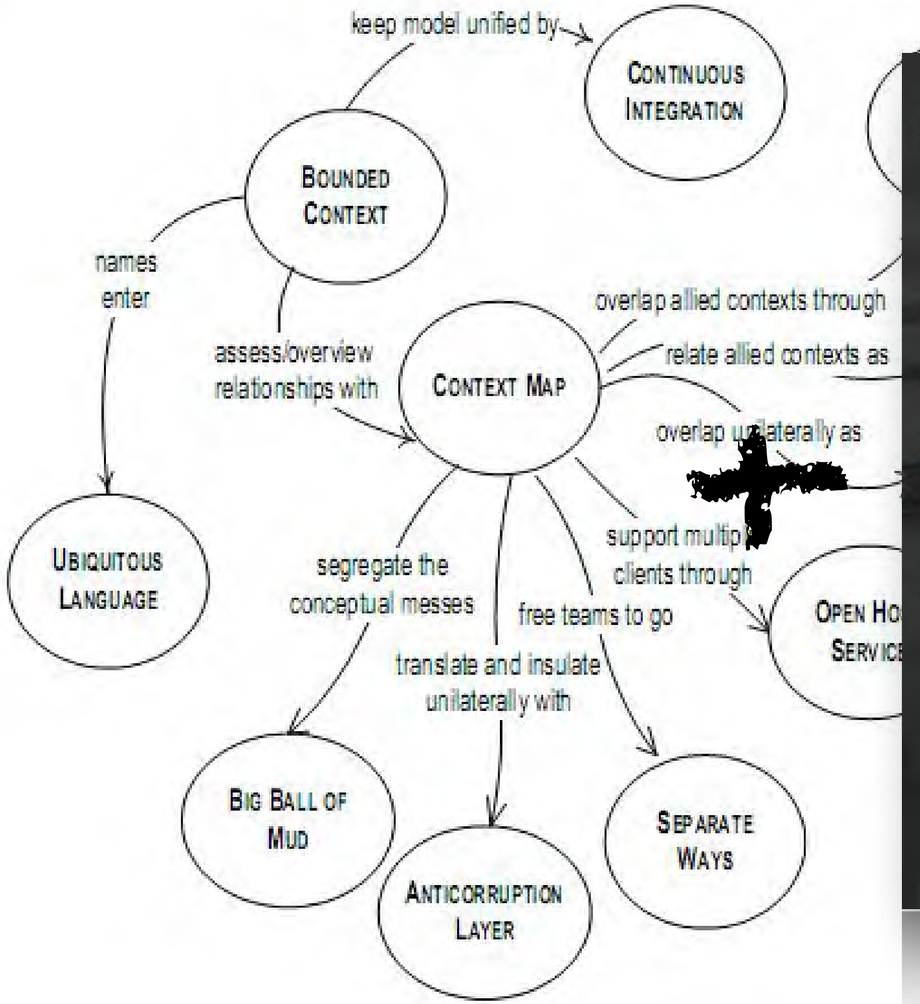
Domain Driven Design

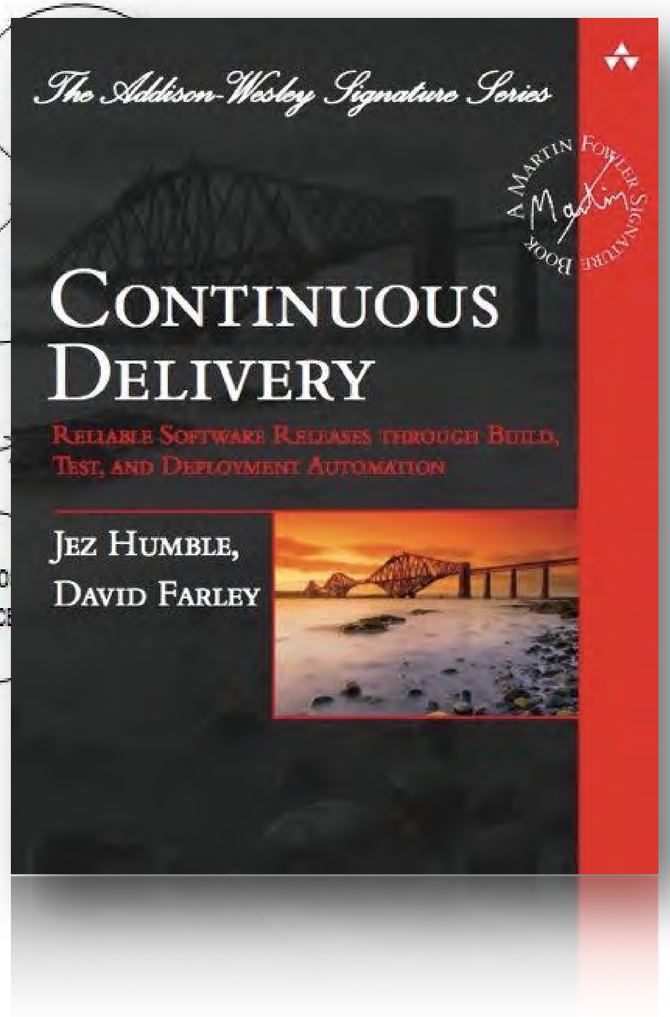




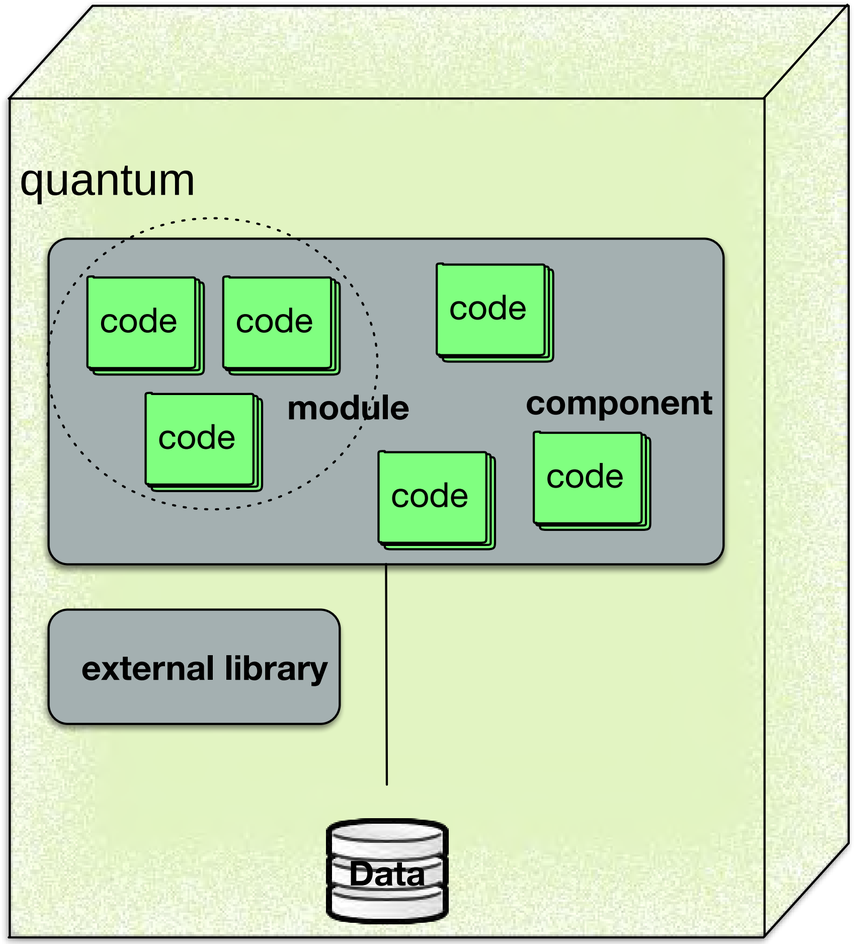
Bounded Context

Maintaining Model Integrity

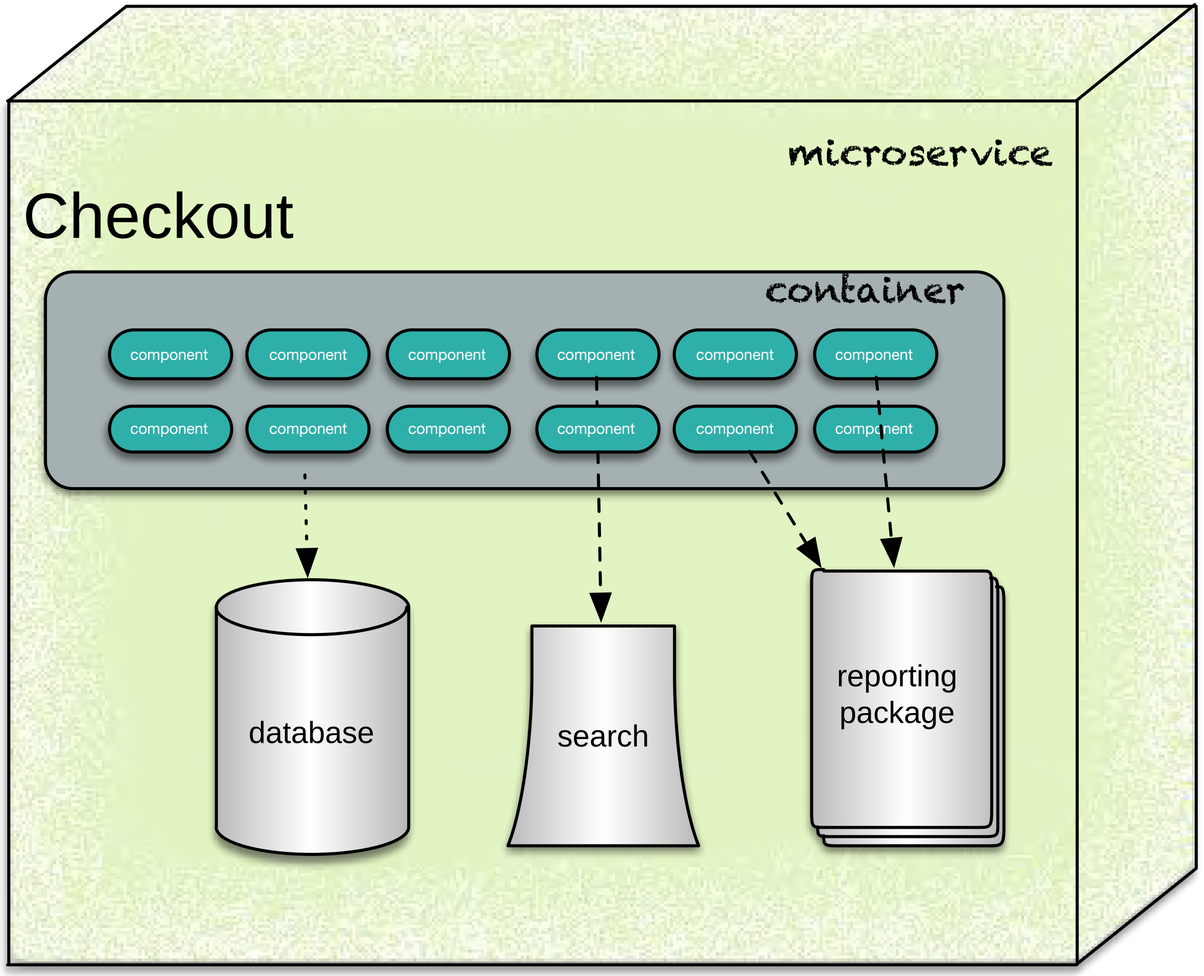




Architectural Quantum



Microservices Quantum

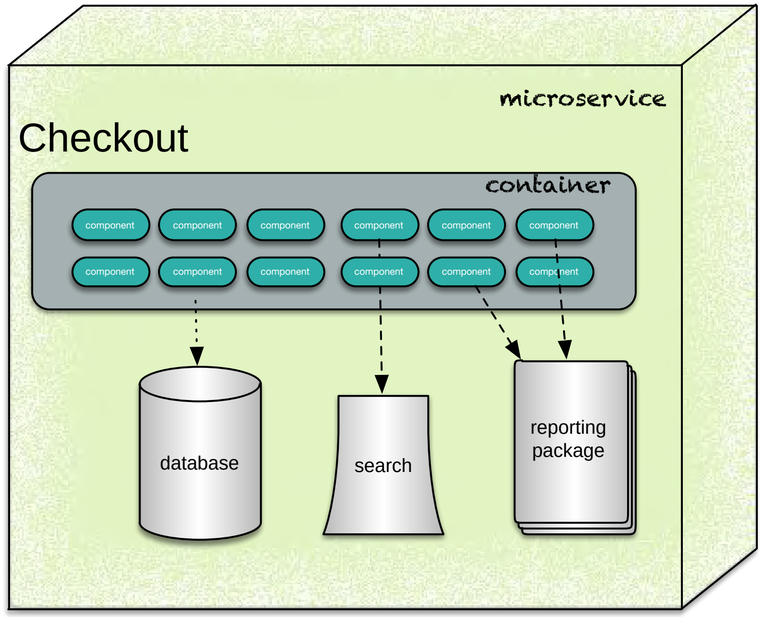


Architectural Quantum

An architectural quantum is an independently deployable component with high functional cohesion, which includes all the structural elements
required for the system to function properly.

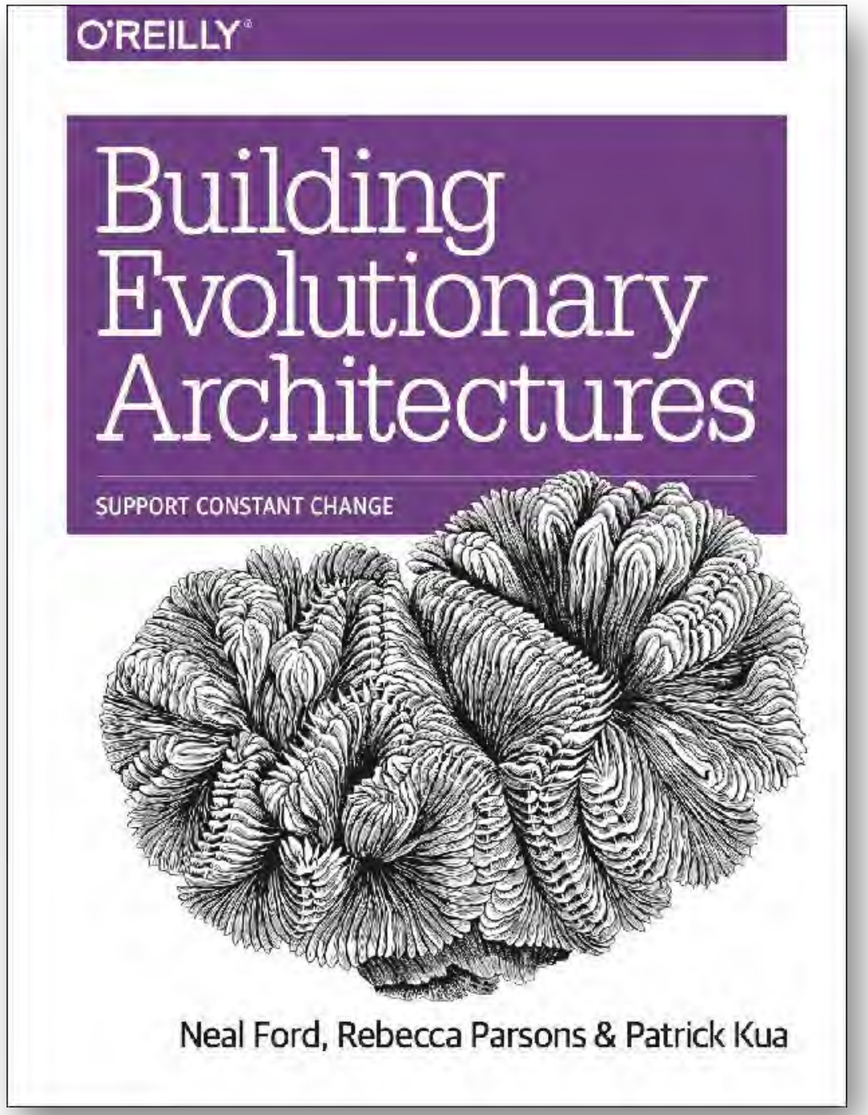
Architectural Quantum

An architectural quantum is an independently deployable component with high functional cohesion, which includes all the structural elements required for the system to function properly.



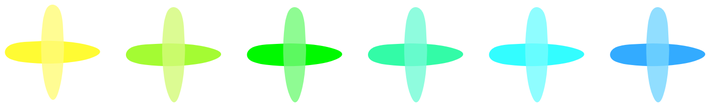
#OSCON

Building Evolutionary Architectures



Evolvability of Architectural Styles

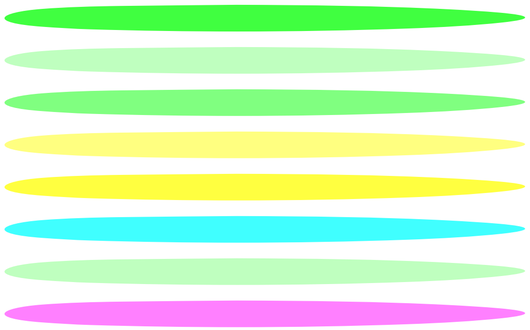
For Each Pattern:



incremental change

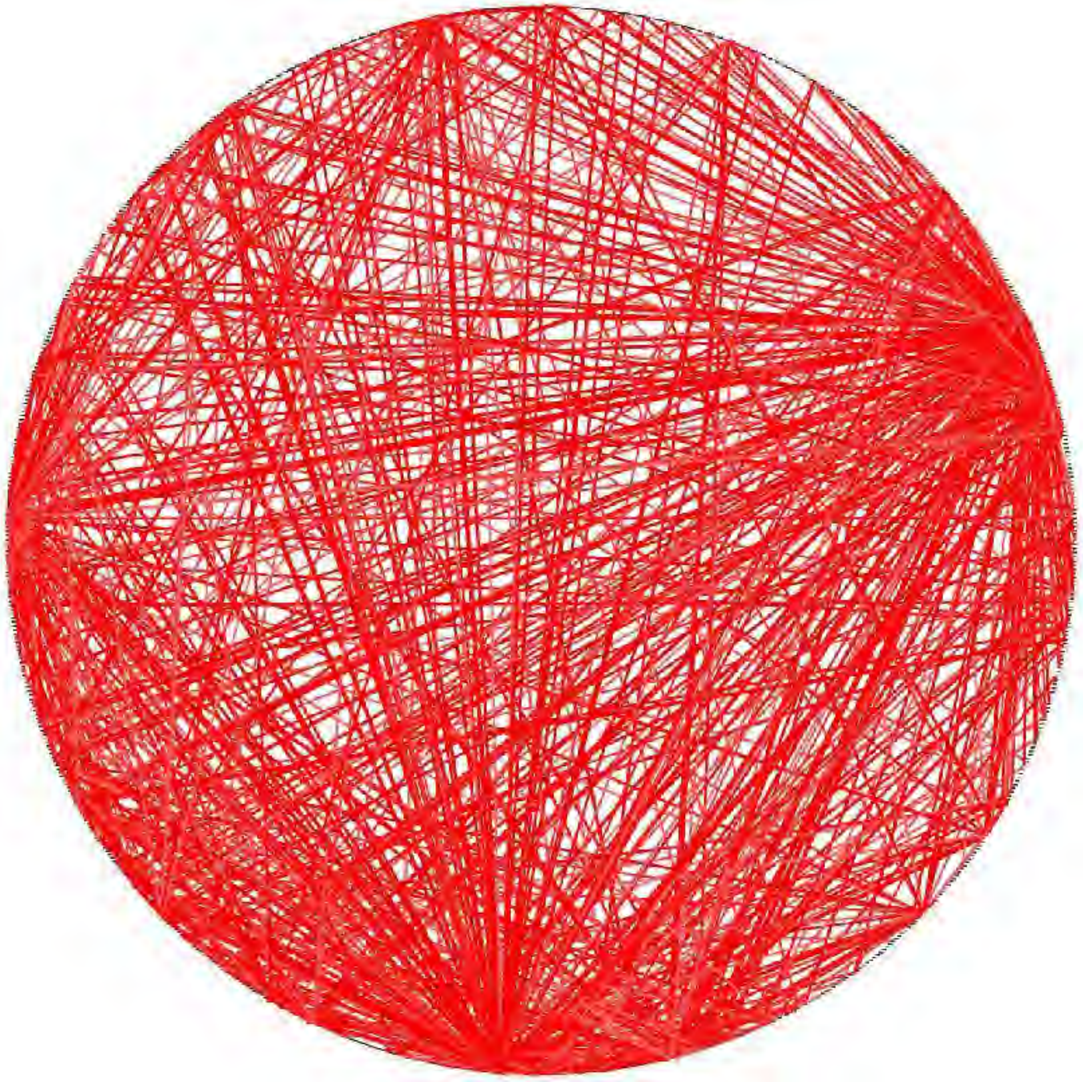
guided change via fitness functions

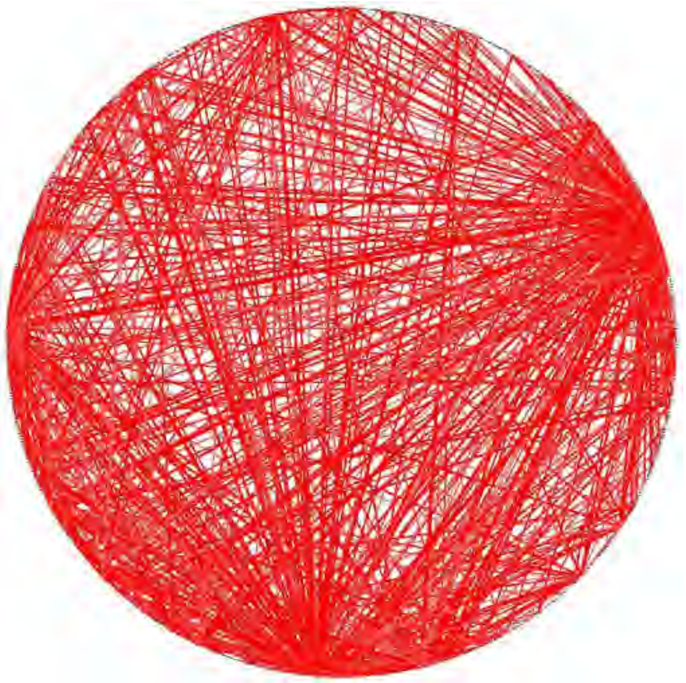
architectural quantum



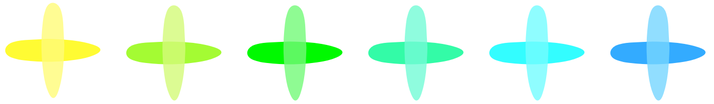
appropriate coupling

Big Ball of Mud



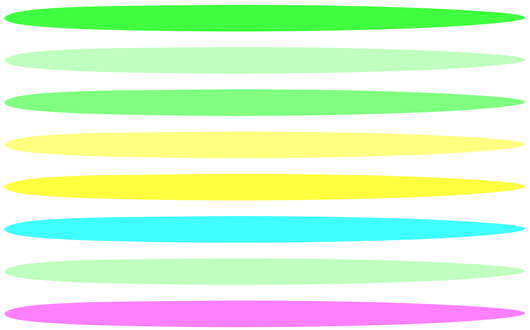


Big Ball of Mud



Rippling side effects for any change

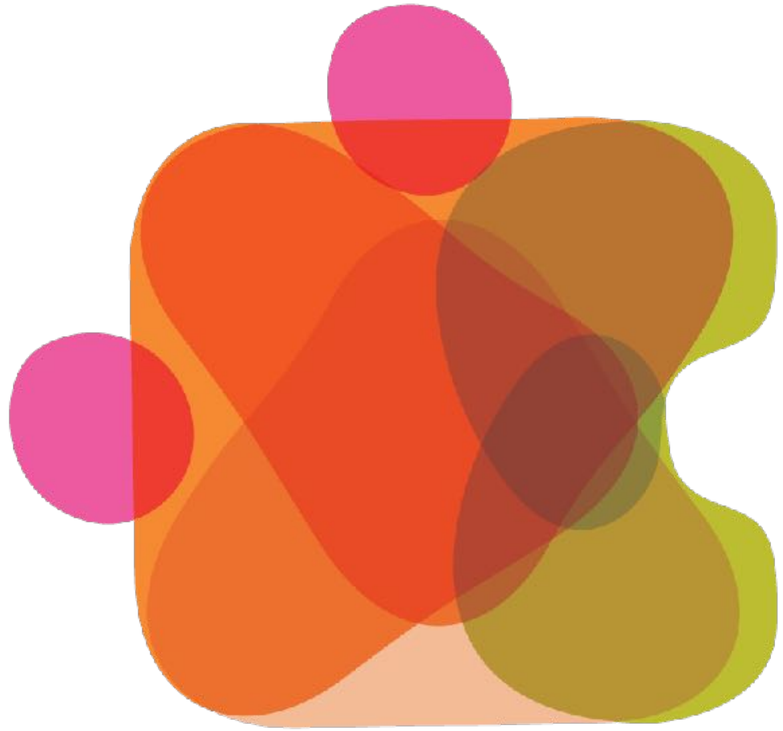
difficult because no clearly defined partitioning exists



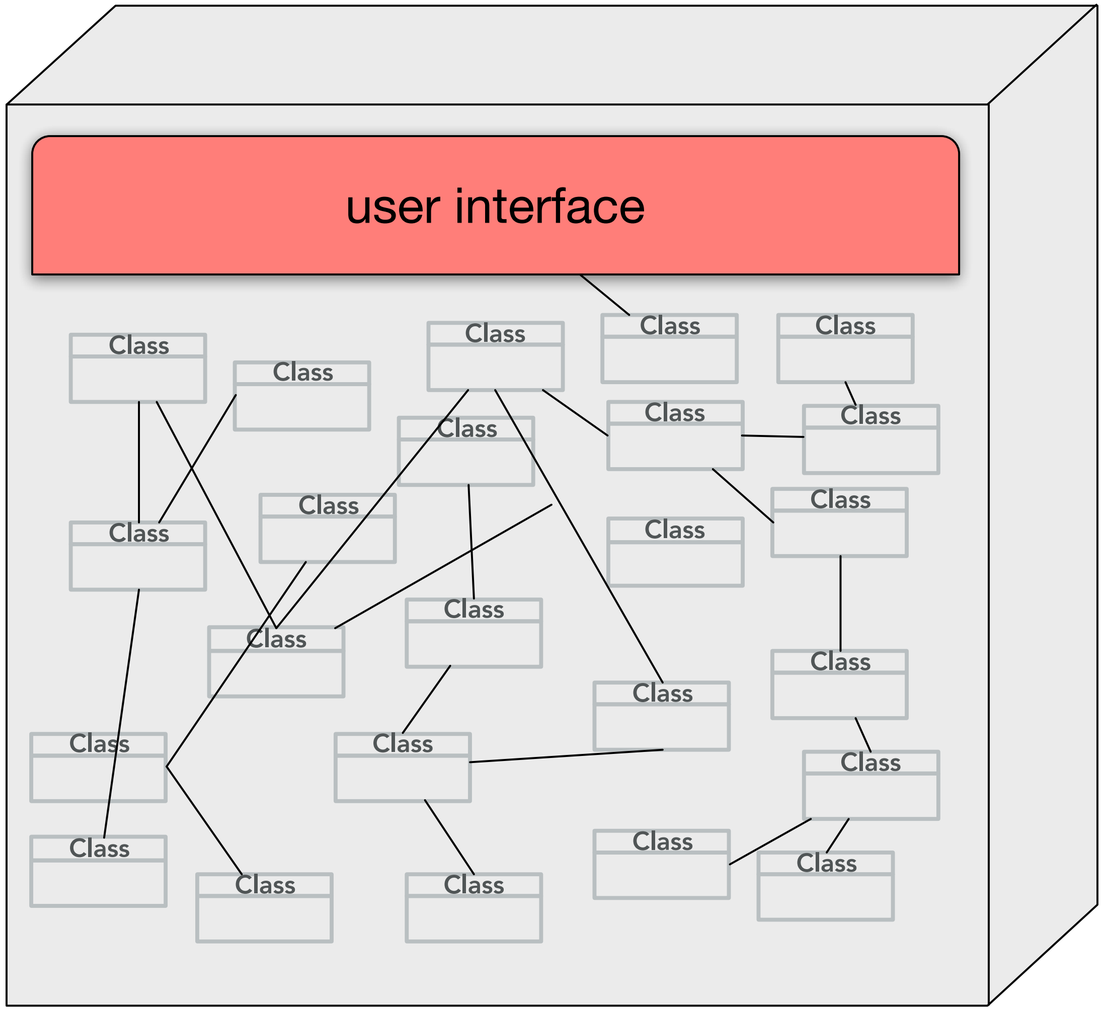
Epitome of **in**appropriate coupling

the entire system

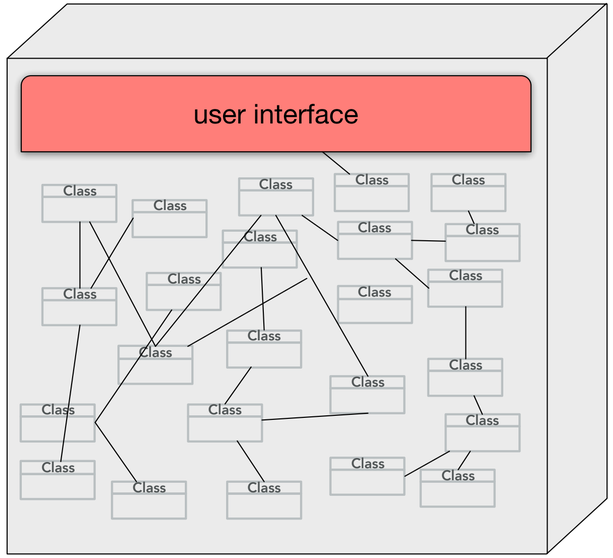
Monoliths



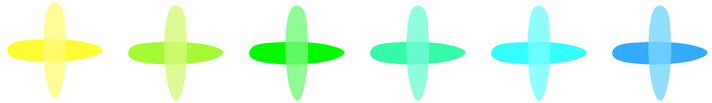
Unstructured Monoliths



Unstructured Monoliths

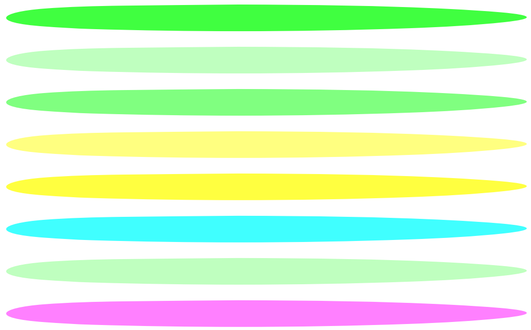


the entire system



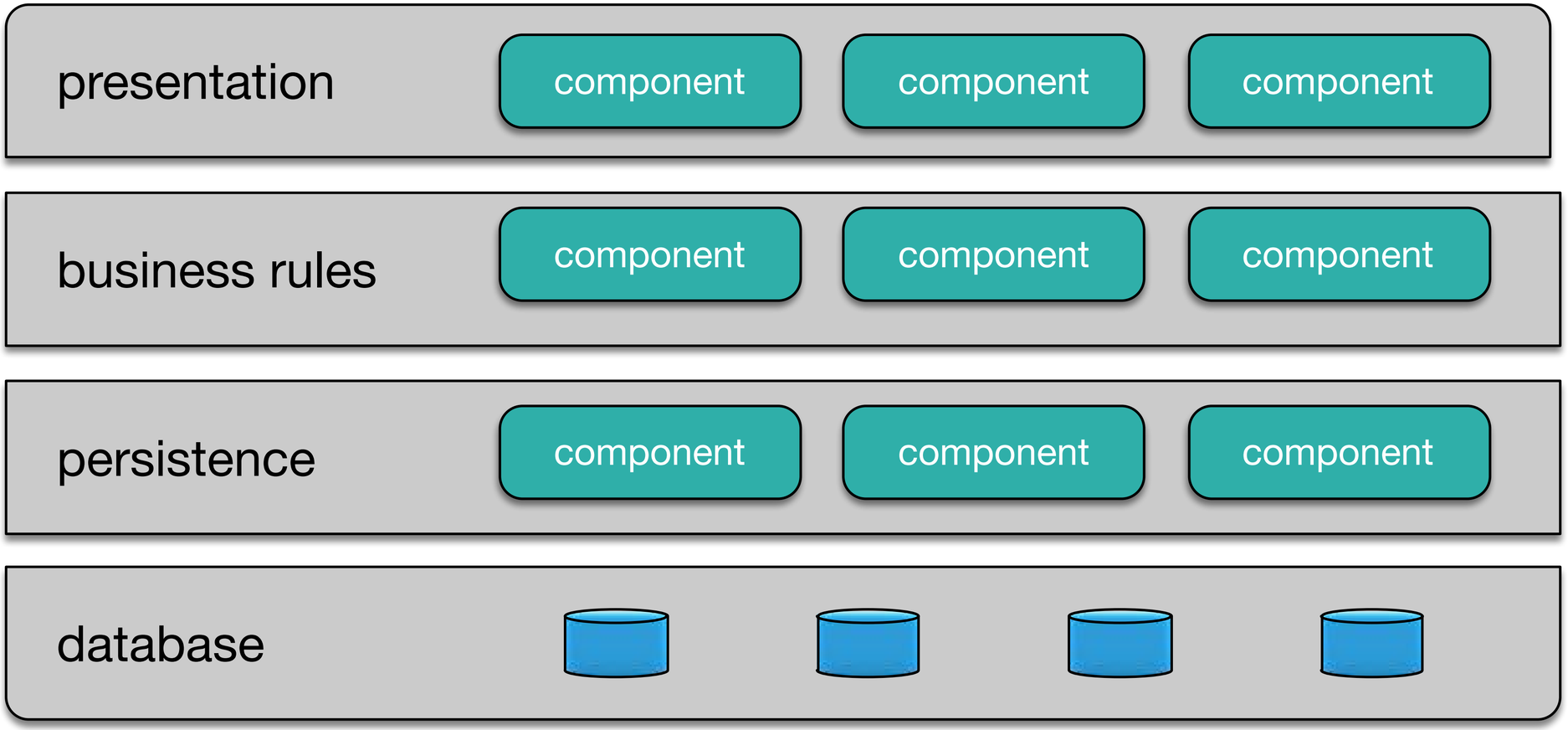
Large quantum size hinders incremental change because high coupling requires deploying large chunks of the application.

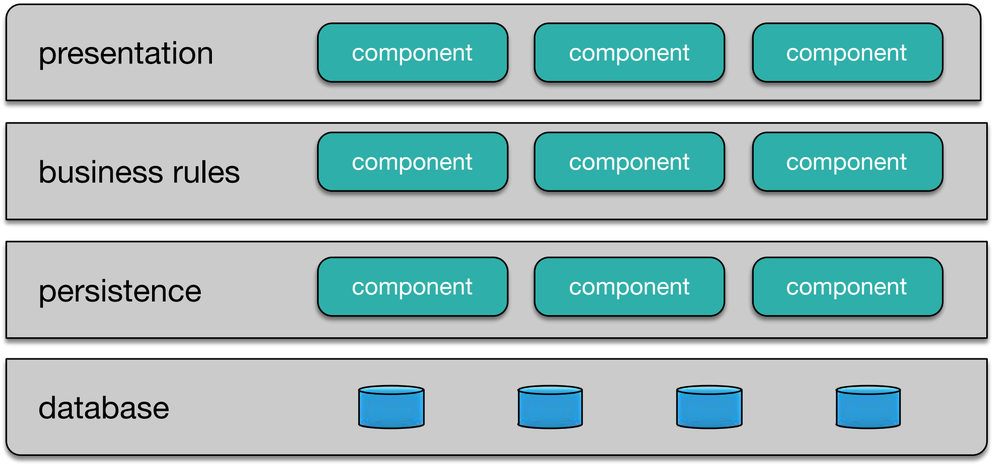
difficult but not impossible



functionally almost as bad as the Big Ball of Mud

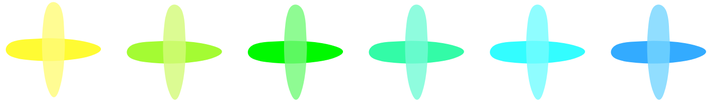
Layered Monolith





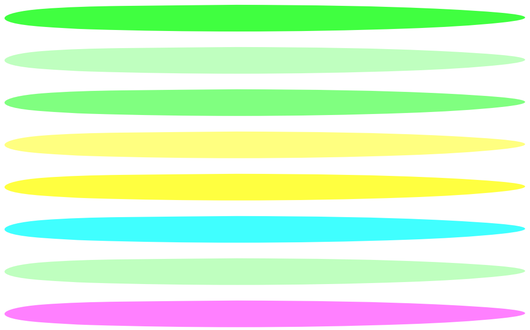
Layered Monolith

the entire system



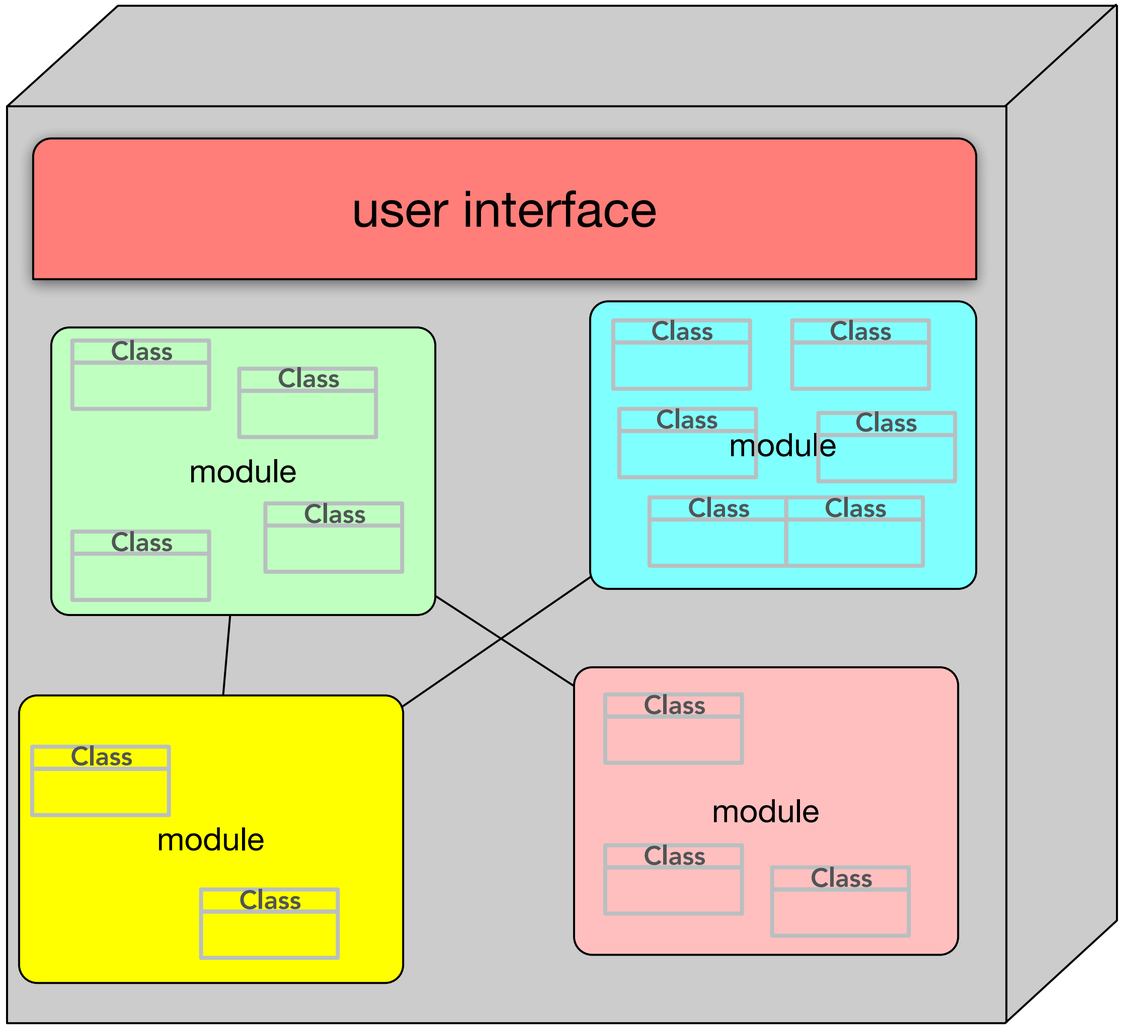
Selectively easy based on partitioning

easier because structure is more apparent

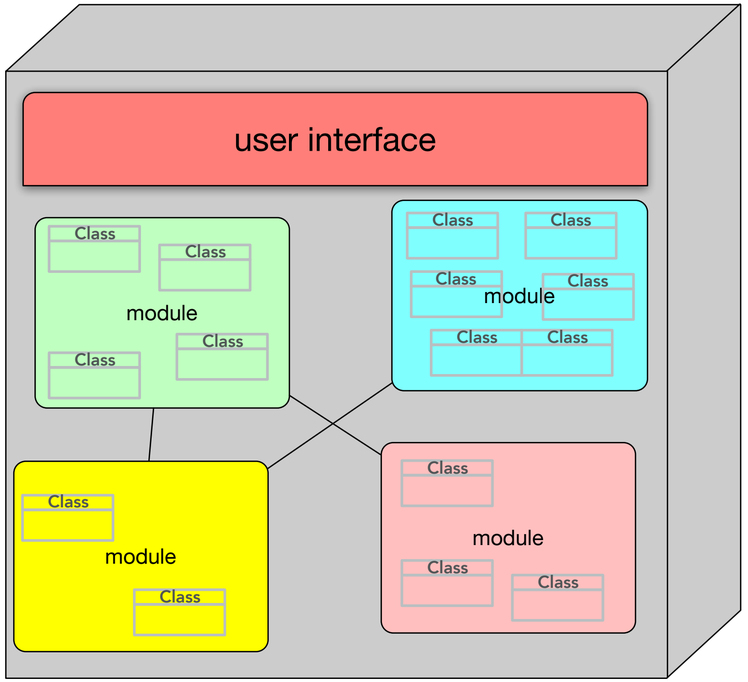


— good technical architecture partitioning

Modular Monoliths



Modular Monoliths



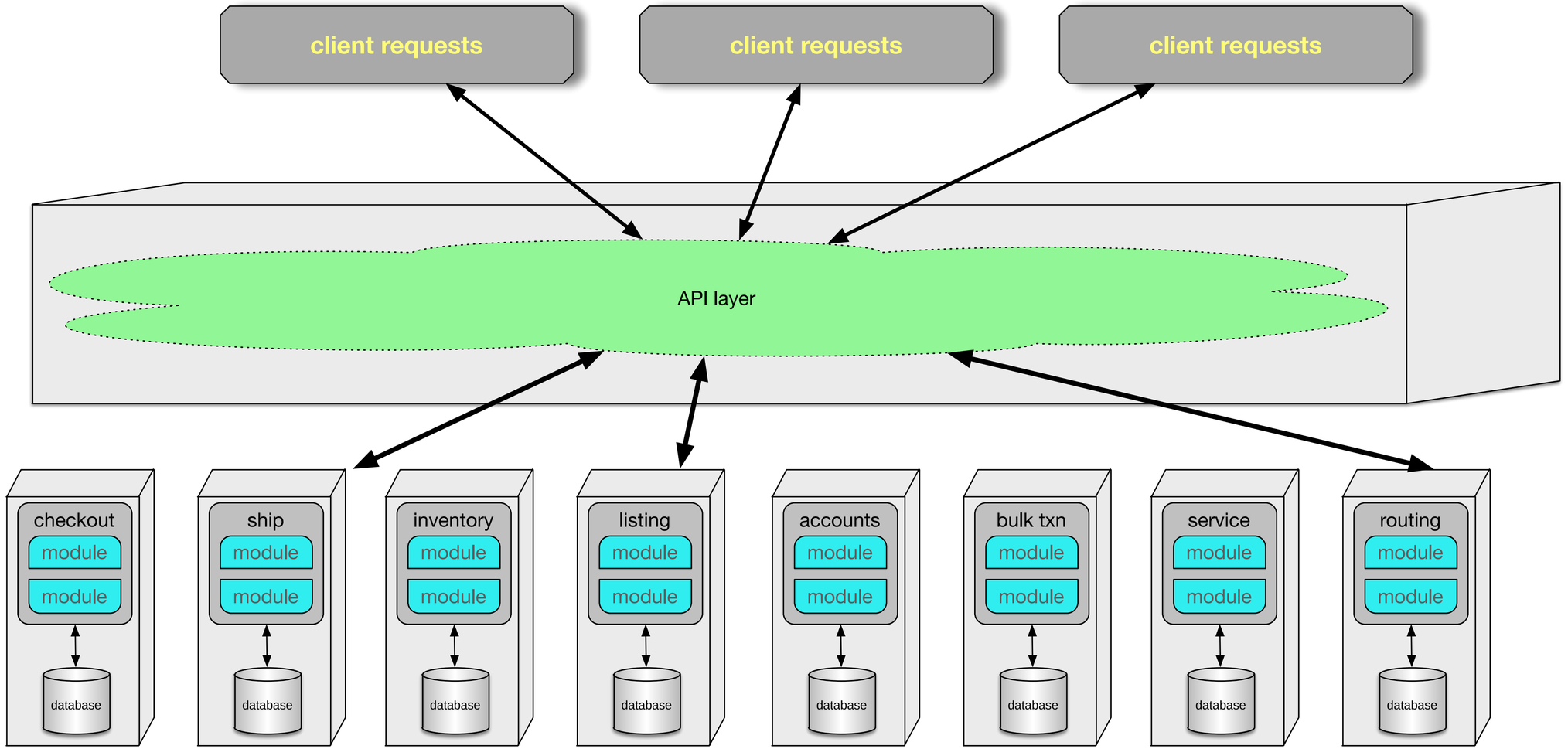
the entire system, with selective better granularity

the degree of deployability of the components determines the rate of incremental change.

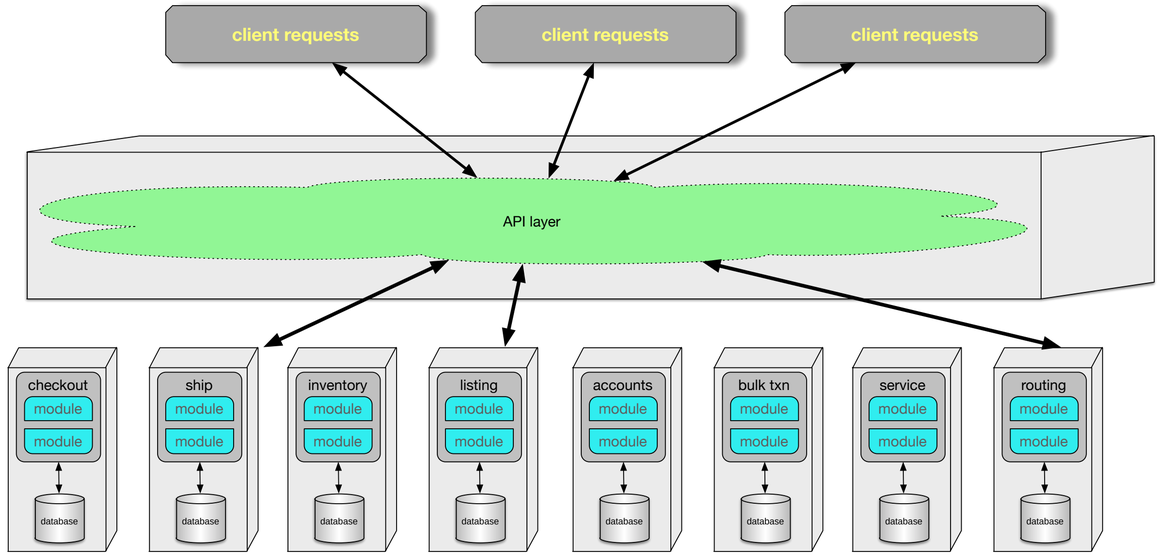
easier to design and implement in this architecture because of good separation of components

Each component is functionally cohesive, with good interfaces between them and low coupling.

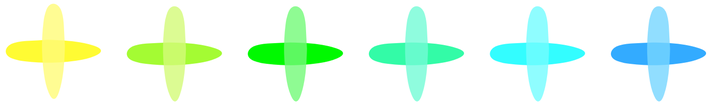
Microservices



Microservices

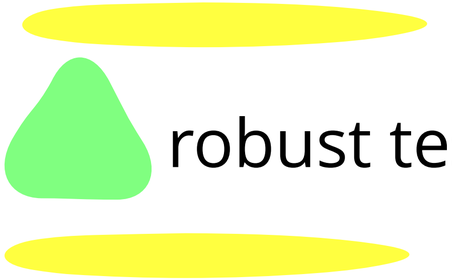


extremely fine grained

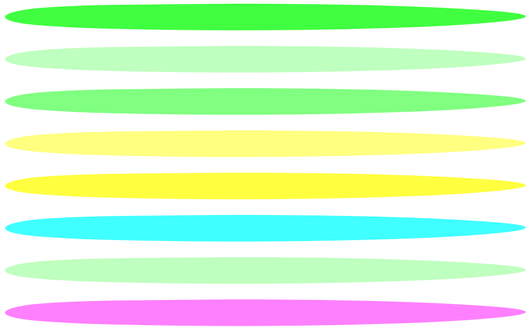


deployment pipeline considered standard

21st century DevOps practices



robust testing & fitness function culture



extremely decoupled fine-grained
quanta with well defined boundaries

“Serverless” Architecture

FaaS

Function as a Service

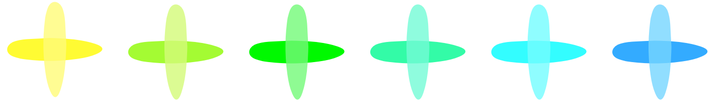
API Gateway

Amazon Lambda

DynamoDB

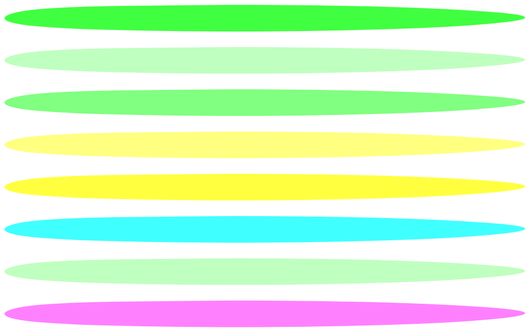
“Serverless” Architecture

N/A



redeploy code

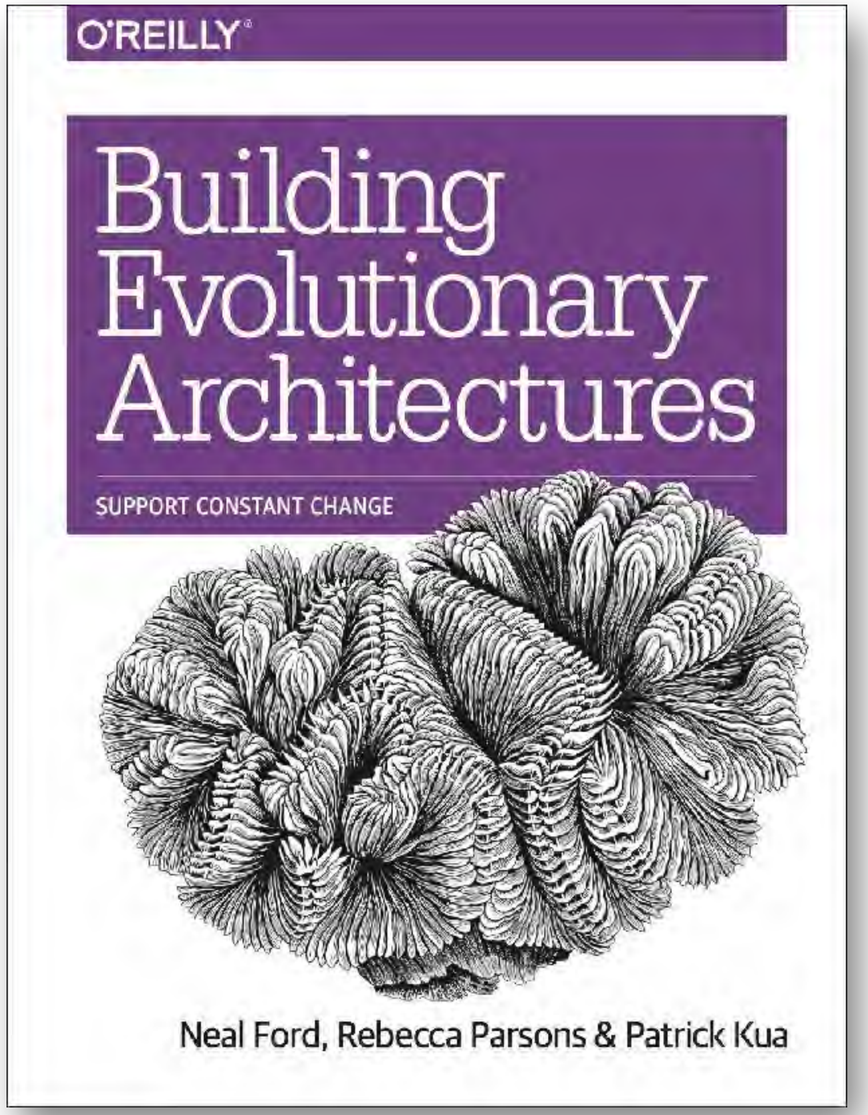
critical



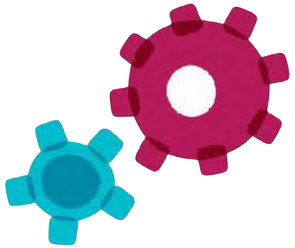
deployment pipeline integration challenging

#OSCON

Building Evolutionary Architectures



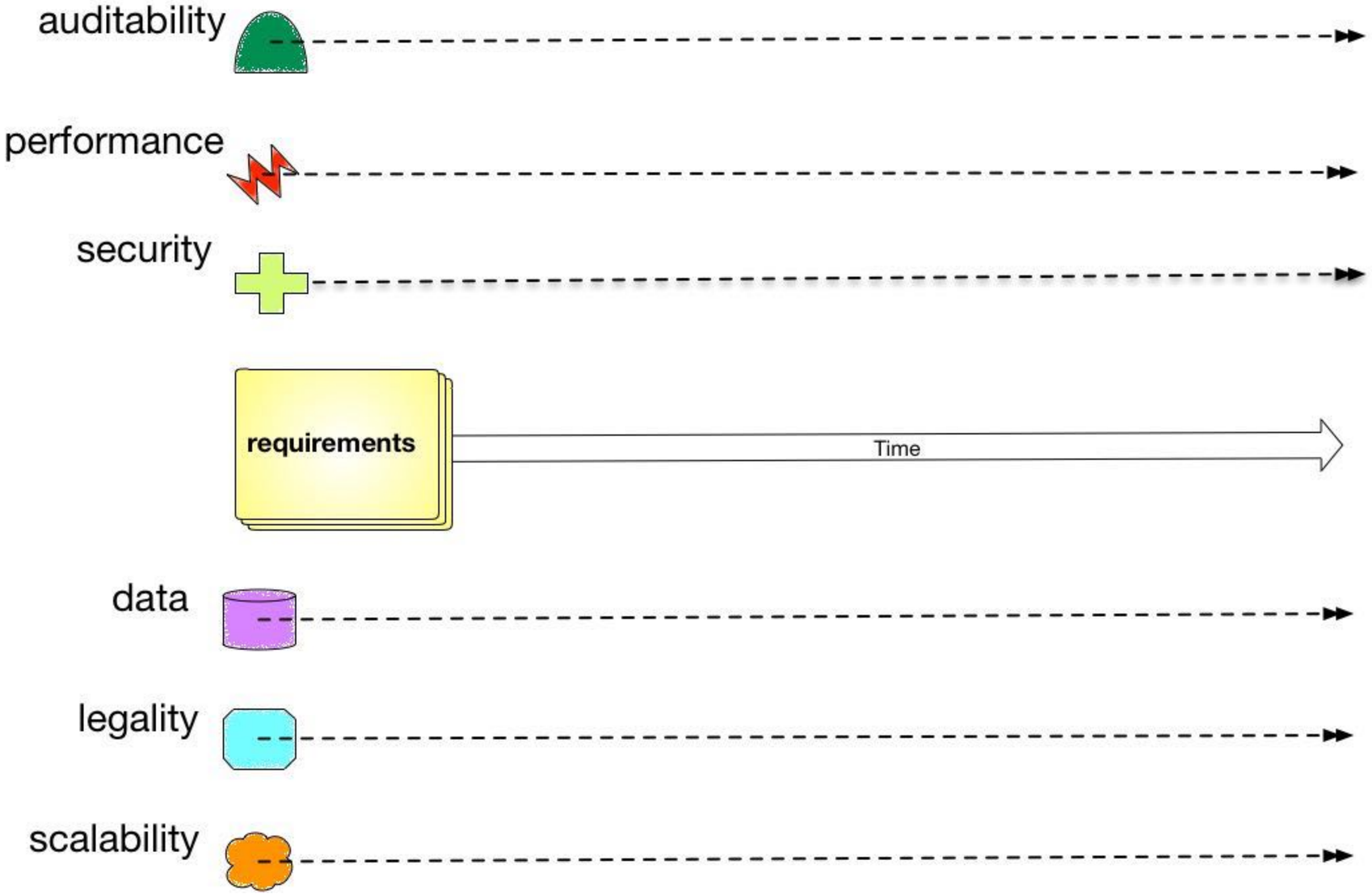
Building Evolvable Architectures: mechanics

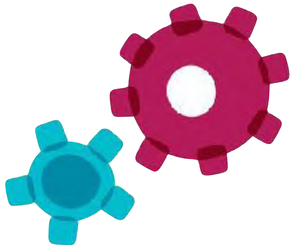


Mechanics

1. Identify dimensions affect by evolution

1. Identify dimensions affect by evolution

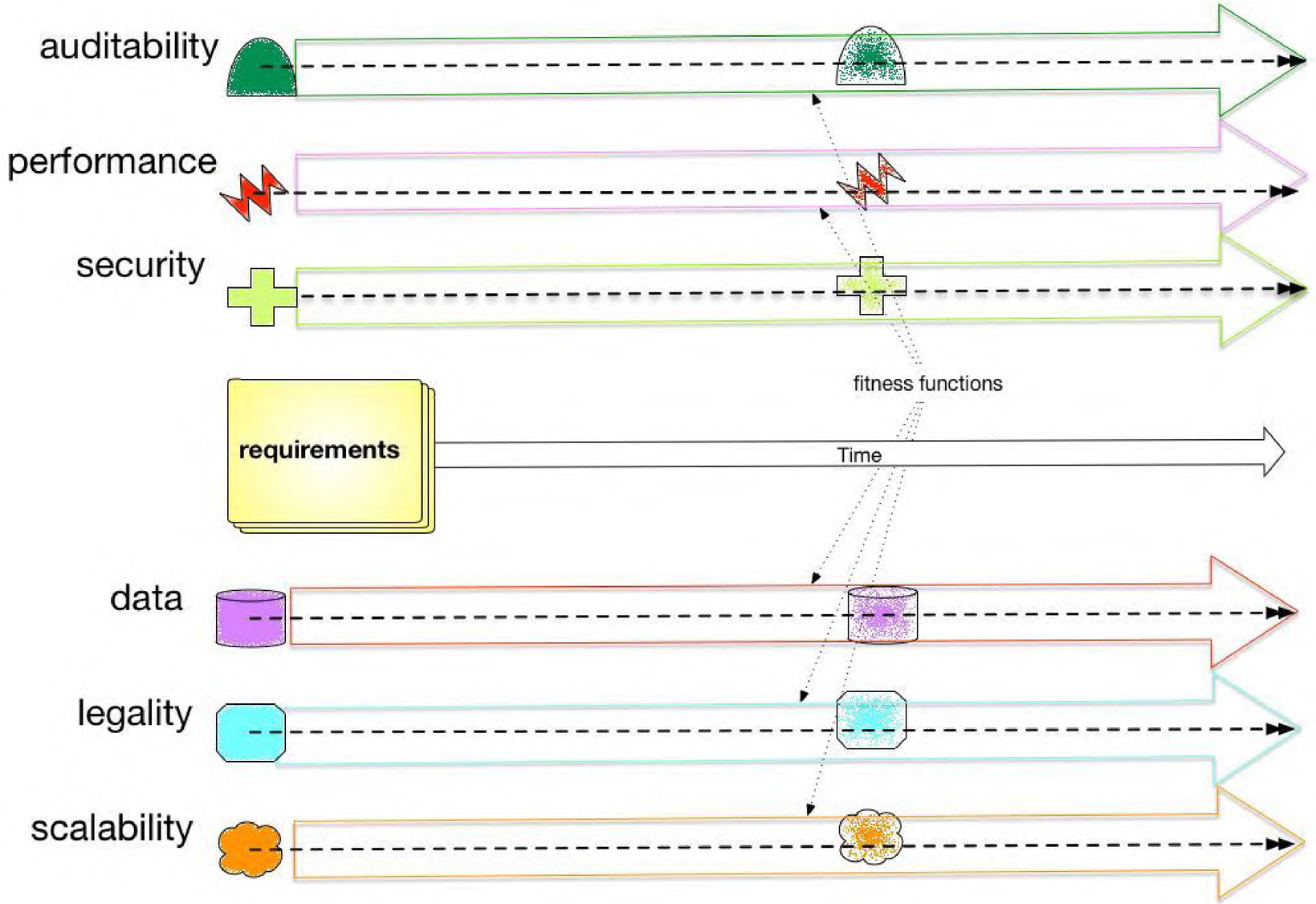


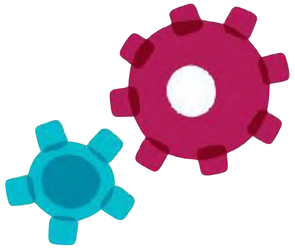


Mechanics

1. Identify dimensions affect by evolution
2. Define Fitness Function(s) for Each Dimension

2. Define Fitness Function(s) for Each Dimension

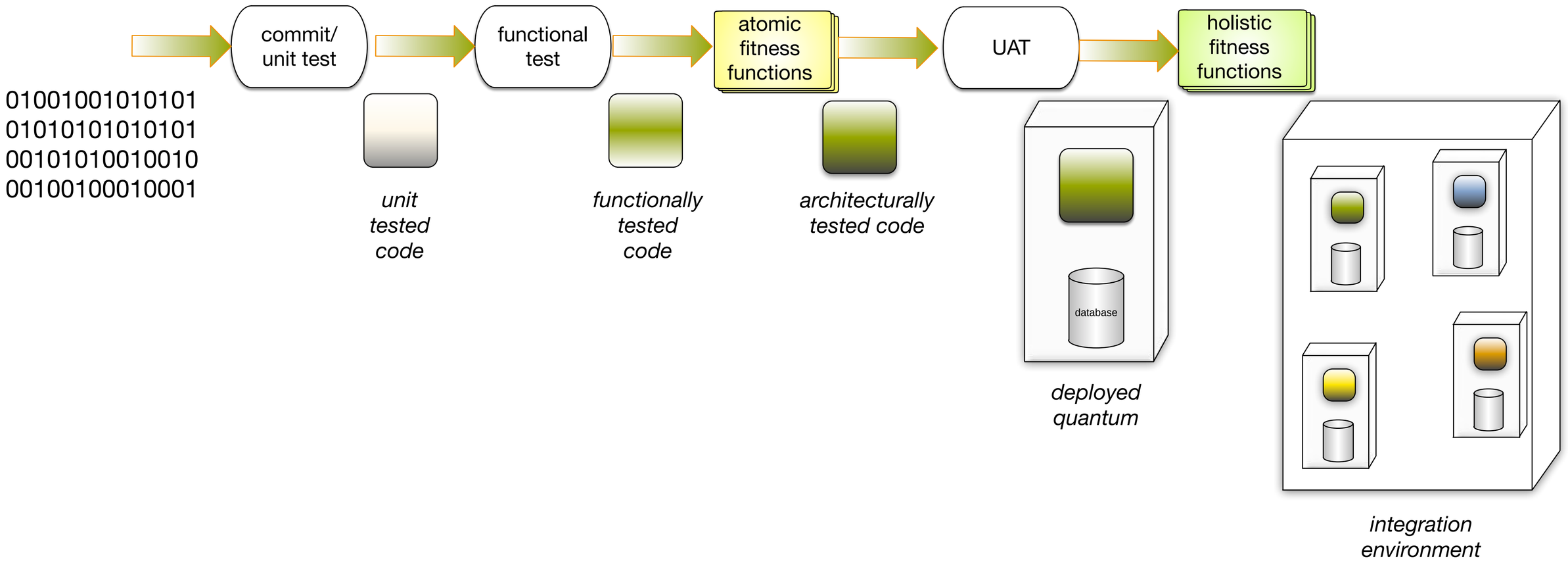


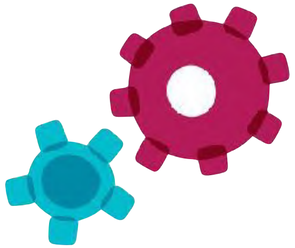


Mechanics

1. Identify dimensions affect by evolution
2. Define Fitness Function(s) for Each Dimension
3. Use Deployment Pipelines to Automate Fitness Functions

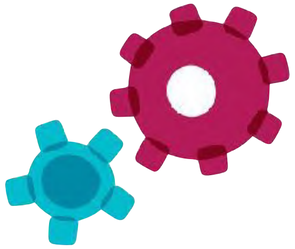
3. Use Deployment Pipelines to Automate Fitness Functions





Mechanics

1. Identify dimensions affect by evolution
2. Define Fitness Function(s) for Each Dimension
3. Use Deployment Pipelines to Automate Fitness Functions

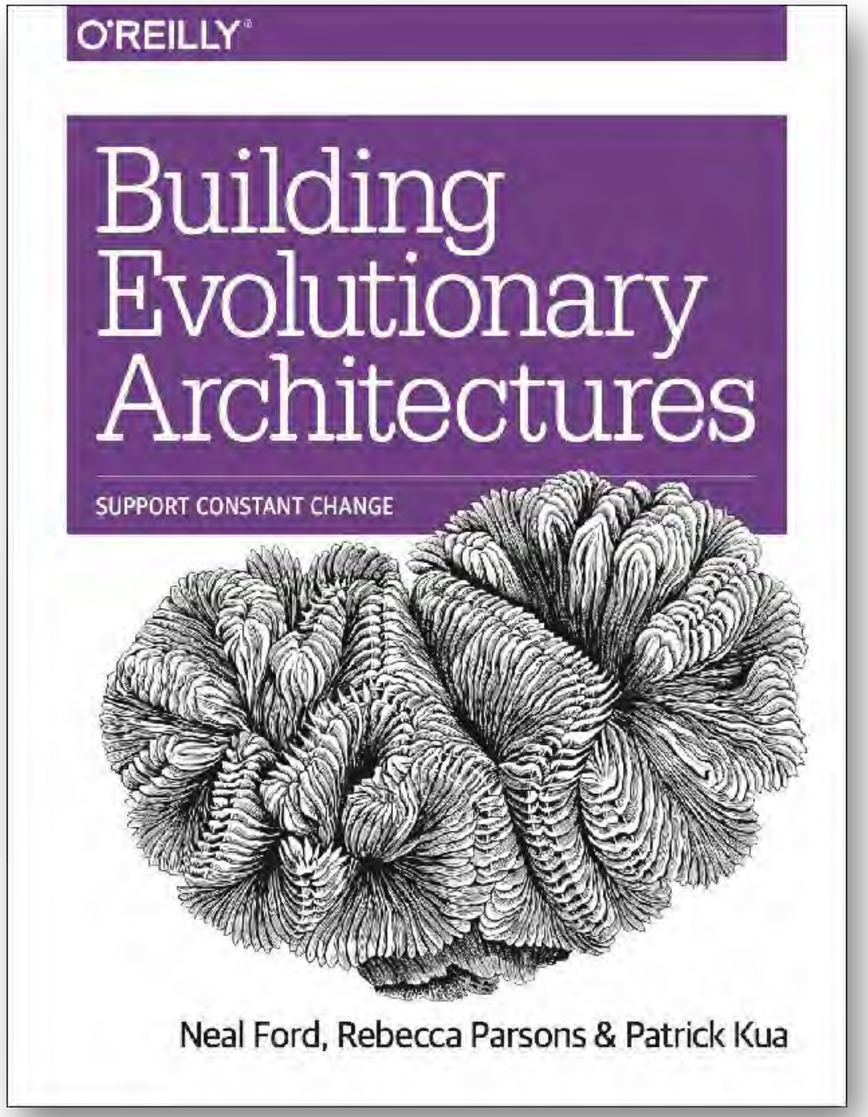


Mechanics

1. Identify dimensions affect by evolution
2. Define Fitness Function(s) for Each Dimension
3. Use Deployment Pipelines to Automate Fitness Functions

#OSCON

Building Evolutionary Architectures



Building Evolvable Architectures: Greenfield & Brownfield Architectures

Greenfield Projects

Greenfield Projects

— implement incremental change at project inception

— fitness functions definition easier before implementation

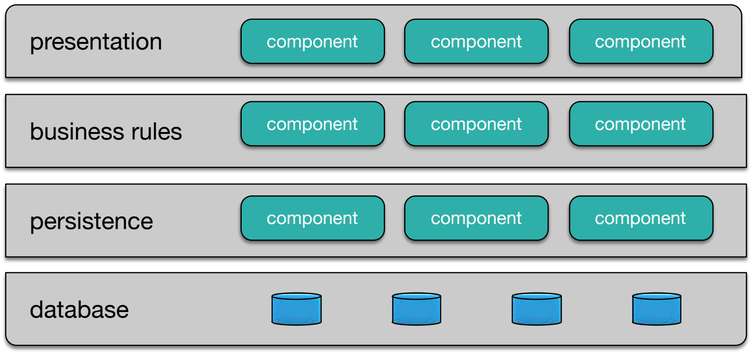
— architects don’t have to untangle legacy coupling points

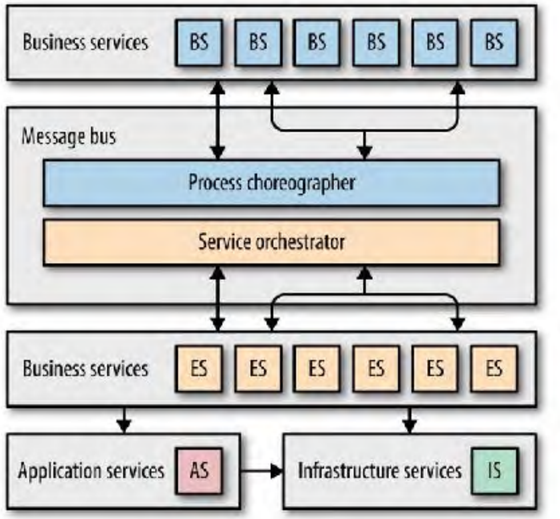
— protective fitness functions from project outset

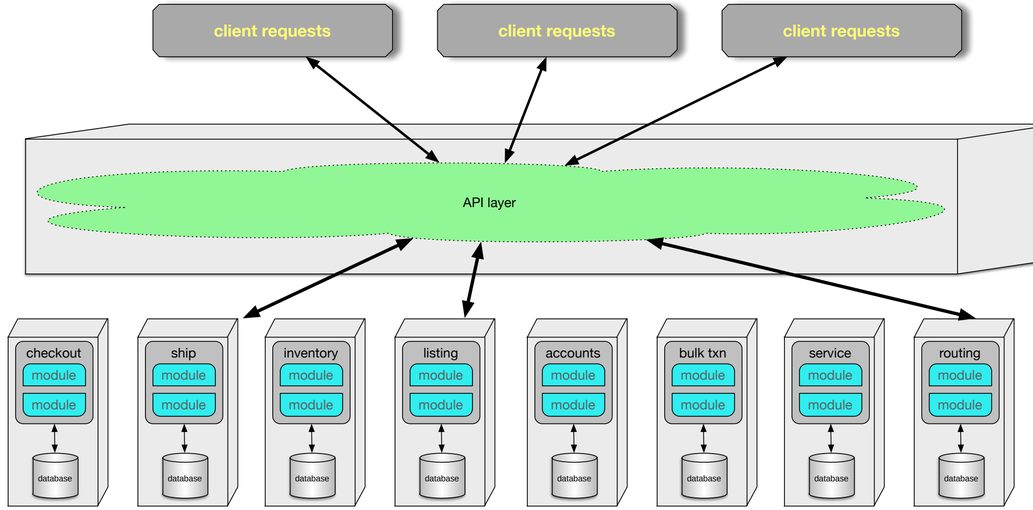
— choose architecture patterns that support evolution

Brownfield Projects

Appropriate Coupling & Cohesion

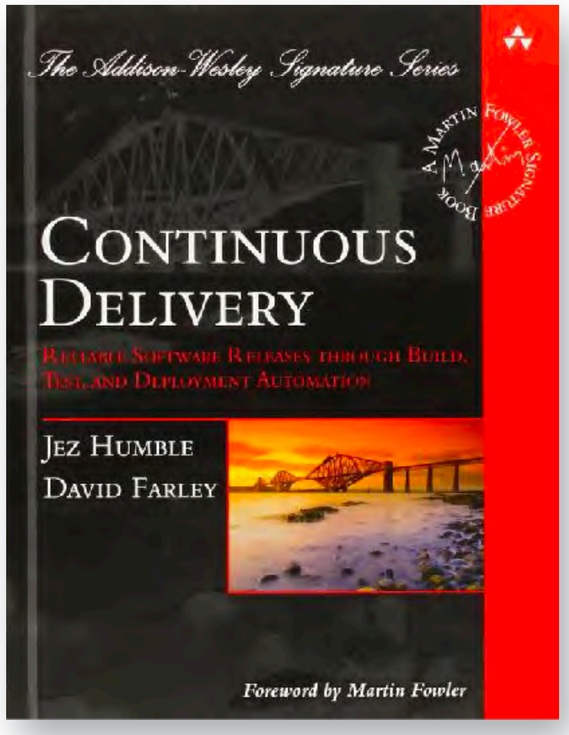


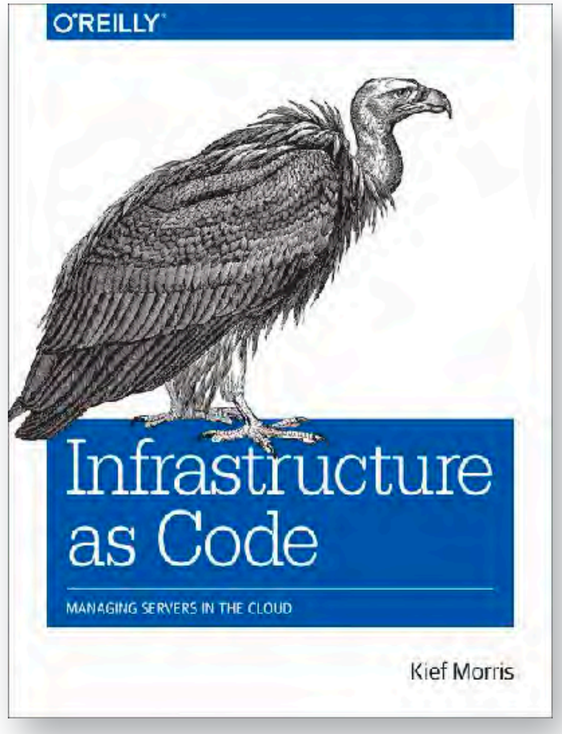




Understand the business problem before choosing an architecture.

Improve Engineering Practices





What About COTS?*

— **quantum size**: the package

— **incremental change**: generally scores poorly

— **appropriate coupling**: generally scores poorly

— **fitness functions**: generally scores very^ poorly

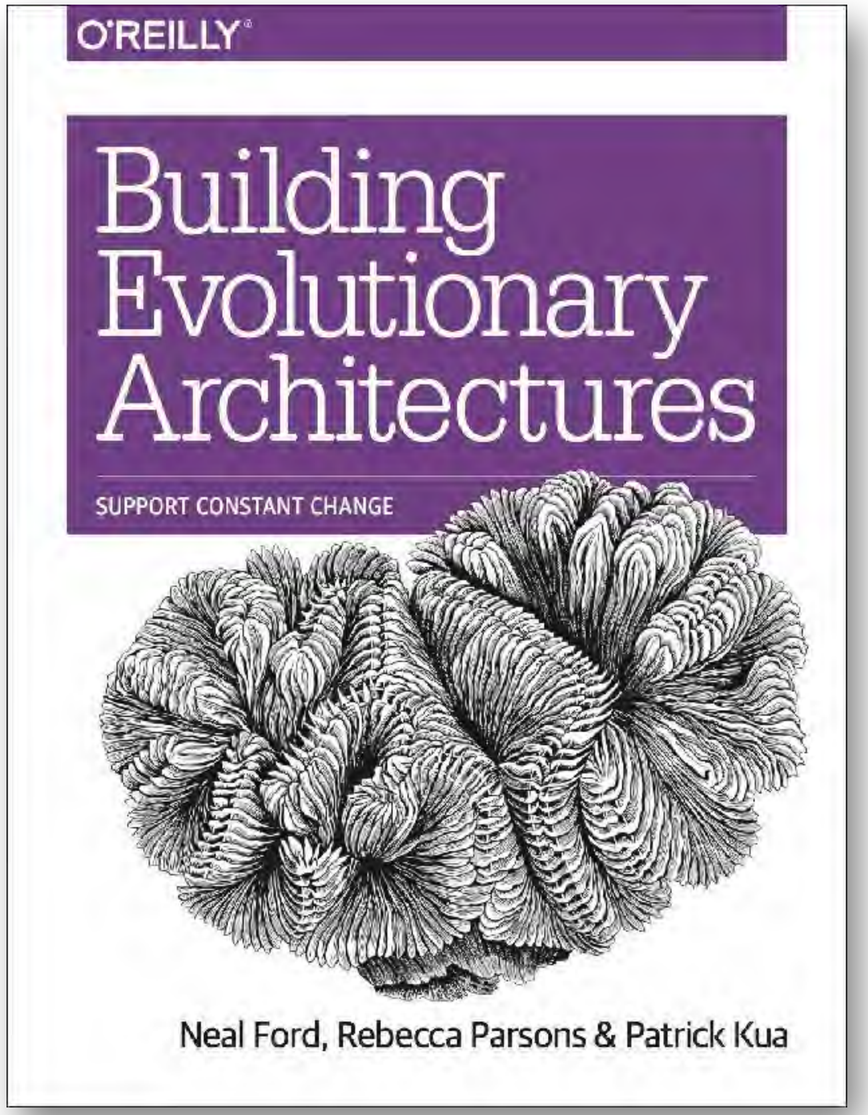
*Commercial Off-the-shelf Software

Work diligently to hold integration points to your level of maturity…

…if that isn’t possible, realize that some parts of the system will be easier for developers to evolve than others.

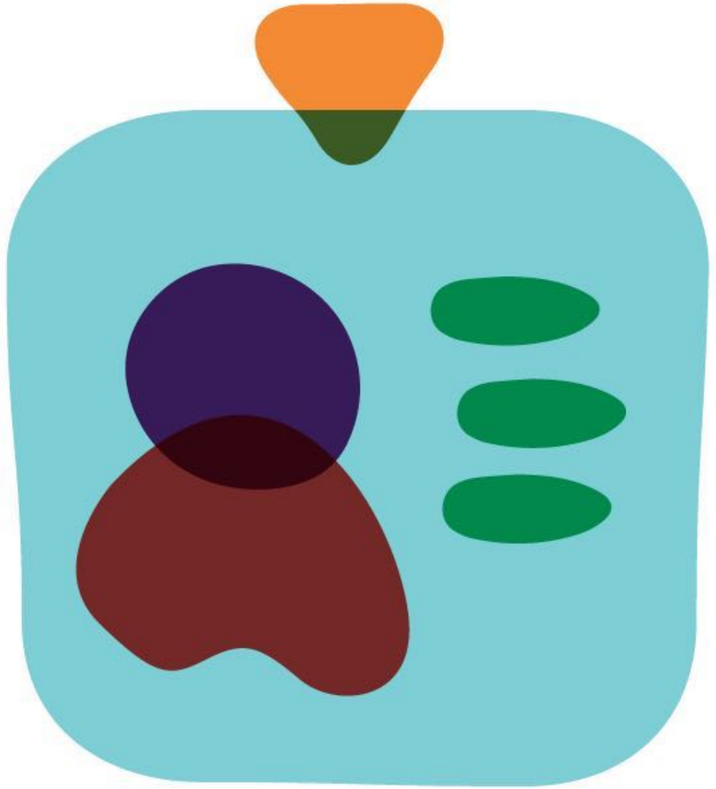
#OSCON

Building Evolutionary Architectures

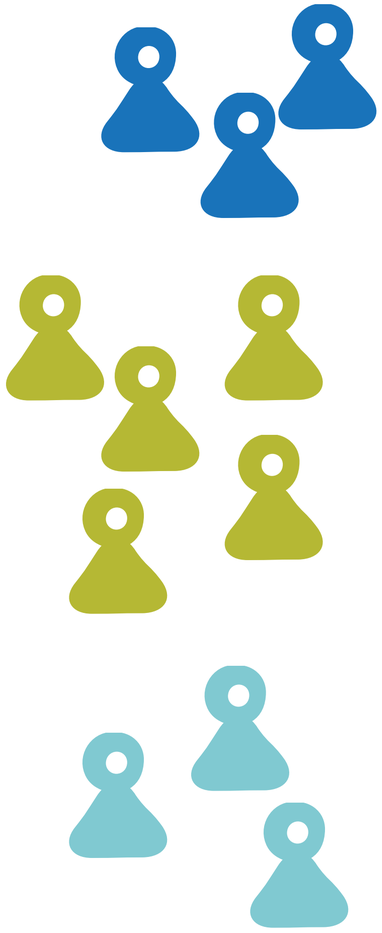


Putting Evolutionary Architecture into Practice

Organizational Factors



Incidentally Coupled Teams



user interface

server-side

DBA

Autonomous Teams

Orders

Shipping

Inverse Conway Maneuver

Catalog

Autonomous Teams...

Orders

Shipping

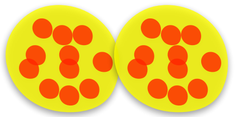
...Organized around Business Capability

Catalog

Product over Project

Autonomous Teams...

Orders



Shipping

Catalog

Product over Project

Product over Project

— Projects are ephemeral

— Projects isolate developers from operational aspects

— Products live forever

— Products have owners

— Products consist of cross-functional teams

Product over Project

Long-term company buy-in

Team Coupling Characteristics

Culture

— Does everyone on the team know what fitness functions are and consider the impact of new tool or product choices on the ability to evolve new fitness functions?

— Are teams measuring how well their system meets their defined fitness functions?

— Do engineers understand cohesion and coupling?

— Are there conversations about what domain and technical concepts belong together?

— Do teams choose solutions not based on what technology they want to learn, but based on its ability to make changes?

— How are teams responding to business changes?

Culture of Experimentation

— Bring ideas from outside

— Encouraging explicit improvement

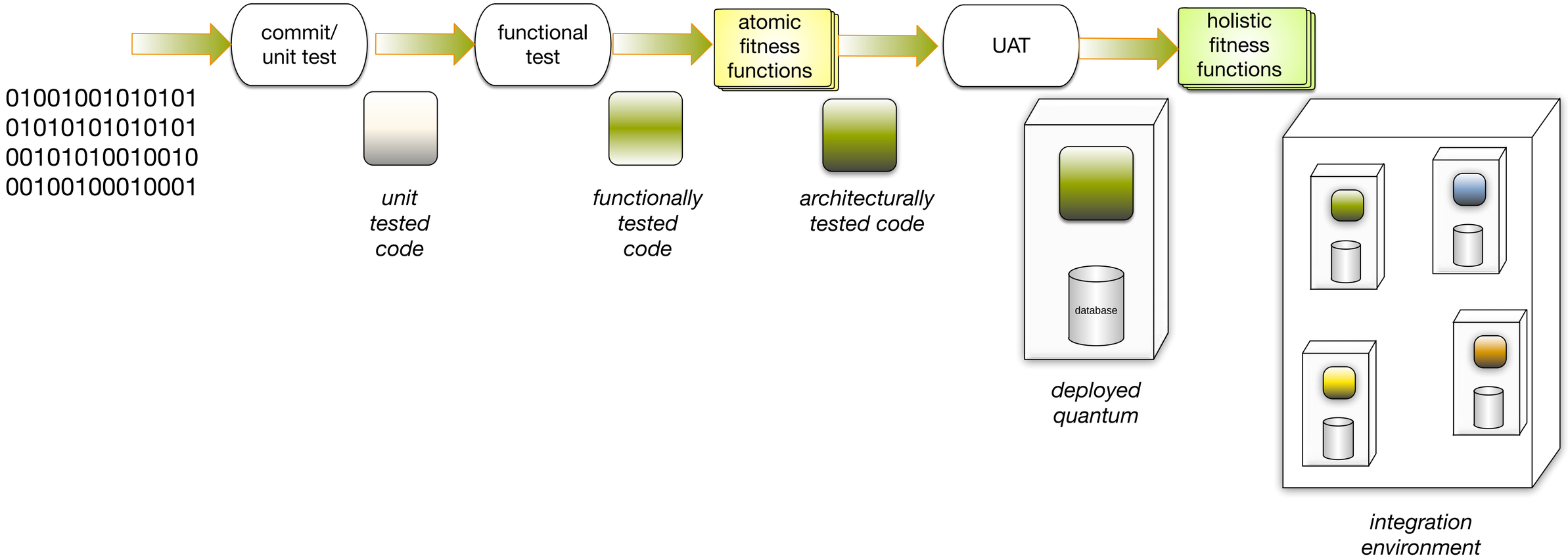
— Spike and stabilize

— Creating innovation time

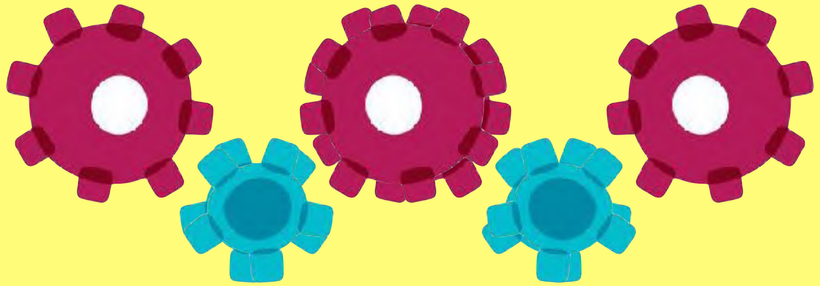
— Following set-based development

— Connecting engineers with end-users

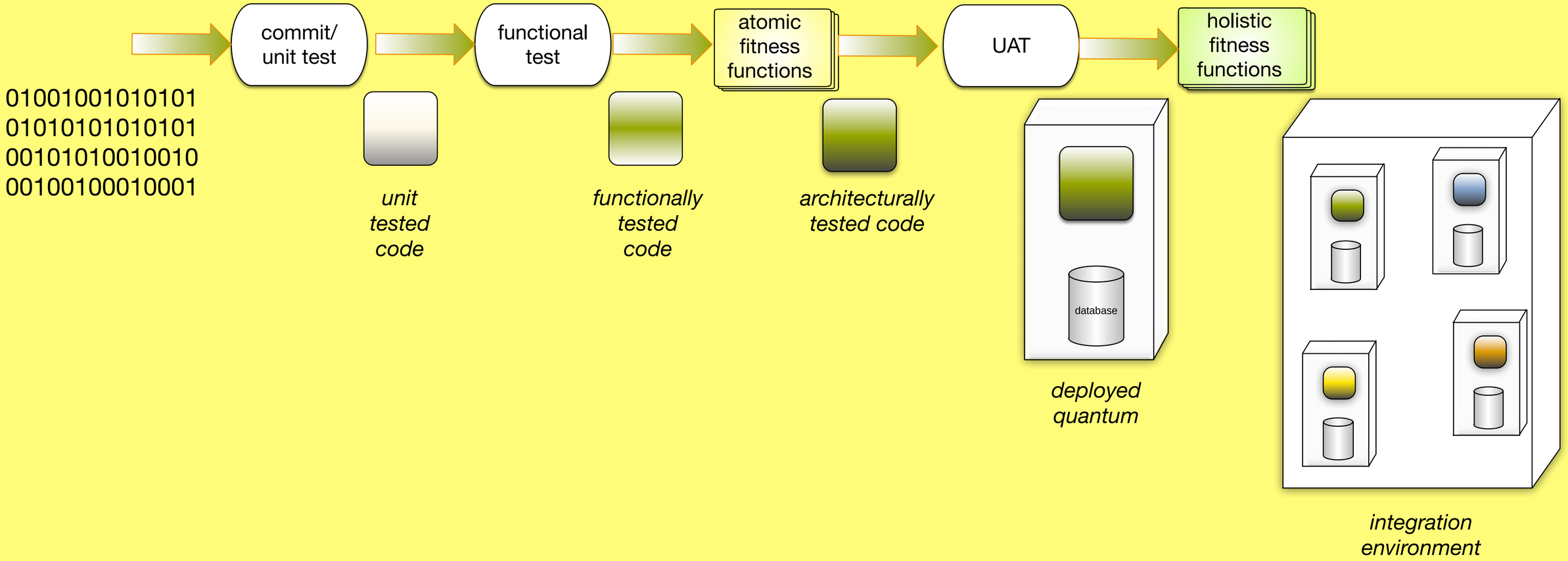
Building Enterprise Fitness Functions



Penultimate

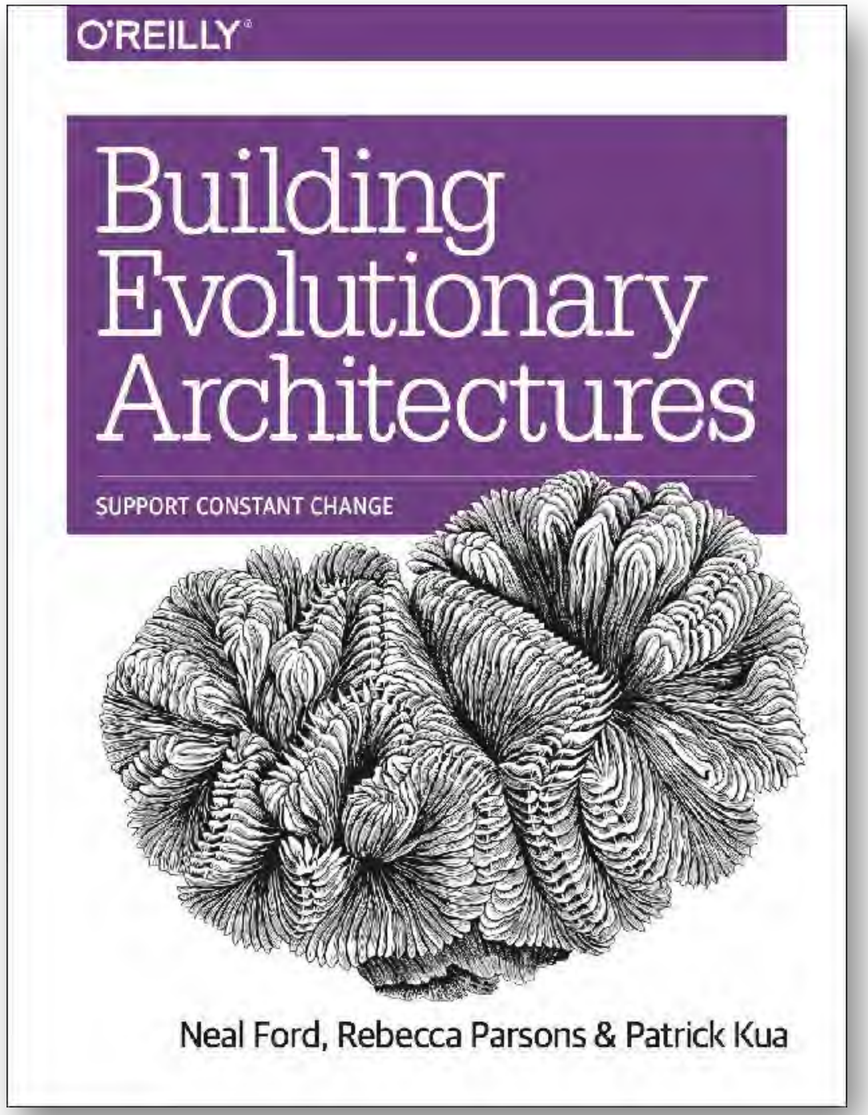


Legality of Open Source Libraries

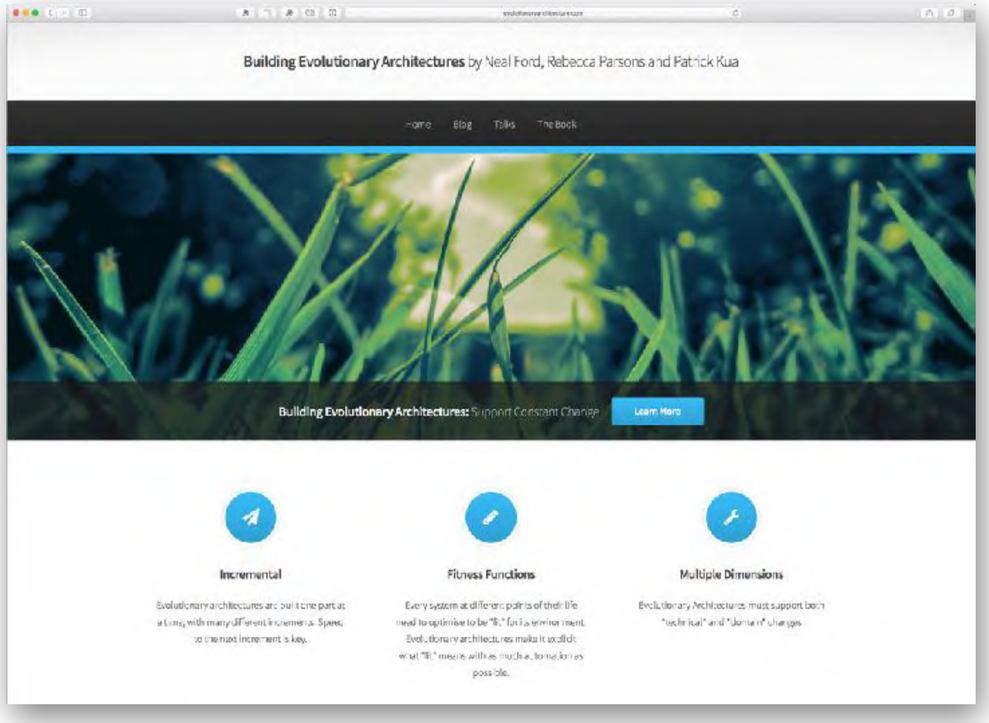

01001001010101
01010101010101
00101010010010
00100100010001
commit/
unit test
functional
test
atomic
fitness
functions
UAT
holistic
fitness
functions
unit
tested
code
functionally
tested
code
architecturally
tested code
database
deployed
quantum
integration
environment

#OSCON

Building Evolutionary Architectures



For more information:



http://evolutionaryarchitecture.com