building systems that are

never done

jalewis@thoughtworks.com

@boicy

ThoughtWorks®

never done

never done

Incomplete

adjective

not having all the necessary or appropriate parts

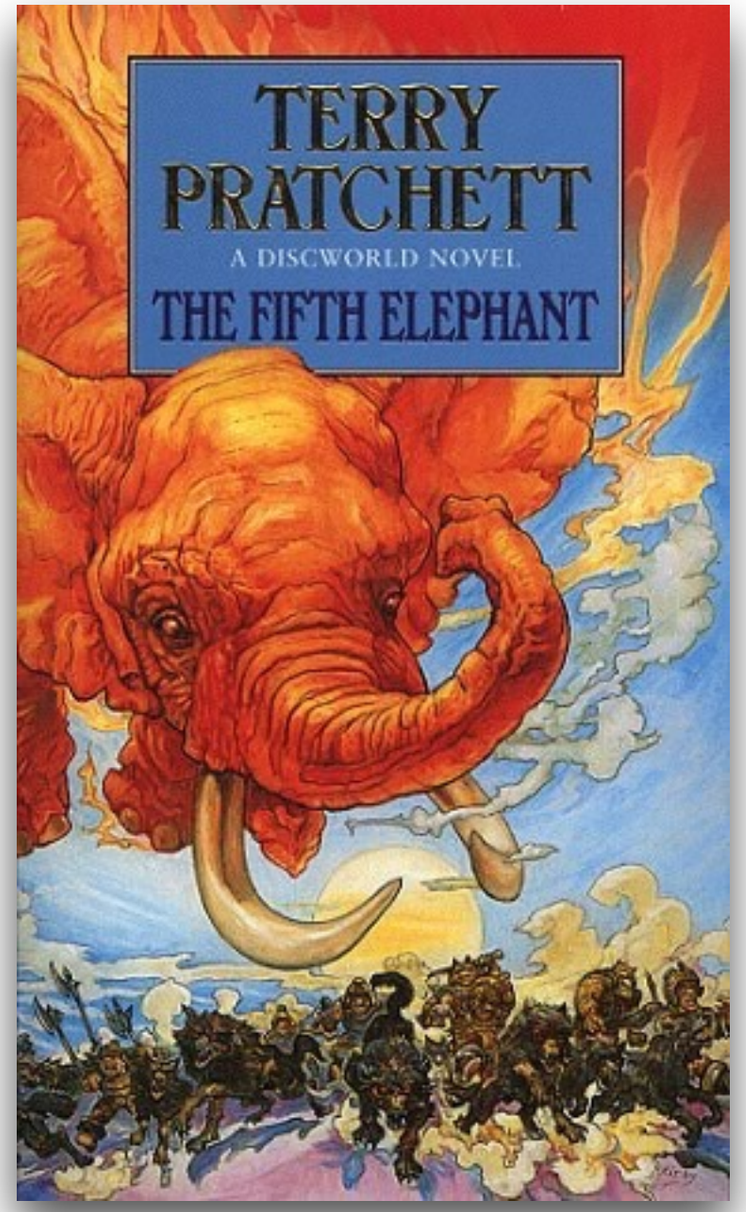
never done

Incomplete

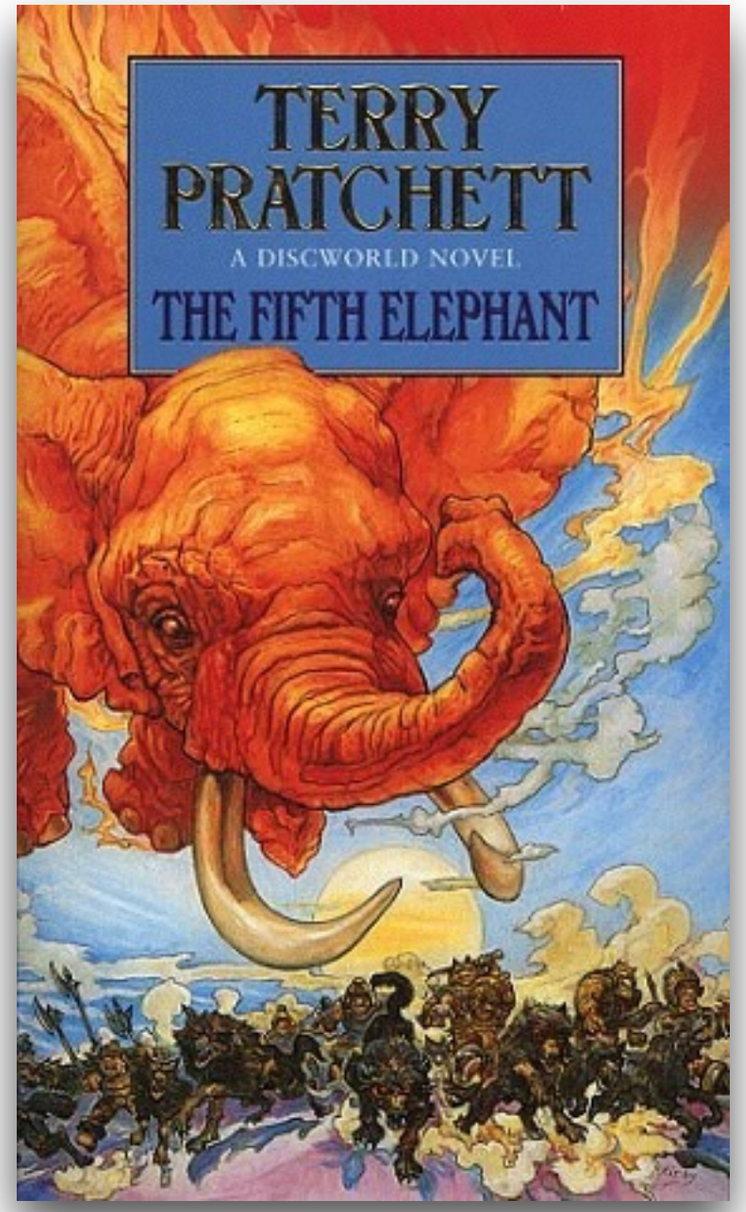
adjective

not having all the ppropriate parts

never done



never done



“This, milord, is my **family's axe**. We have owned it for almost nine hundred years, see. Of course, sometimes it needed a **new blade**. And sometimes it has required a **new handle**, new designs on the metalwork, a little refreshing of the ornamentation . . . but **is this not** the nine hundred-year-old axe of my family? And because it has changed gently over time, it is still a pretty good axe, y'know. Pretty good.”

microservices **should** *be:*

cheap to replace

quick to scale

able to withstand failure

and **should** *allow us to go as*
"fast as possible"?

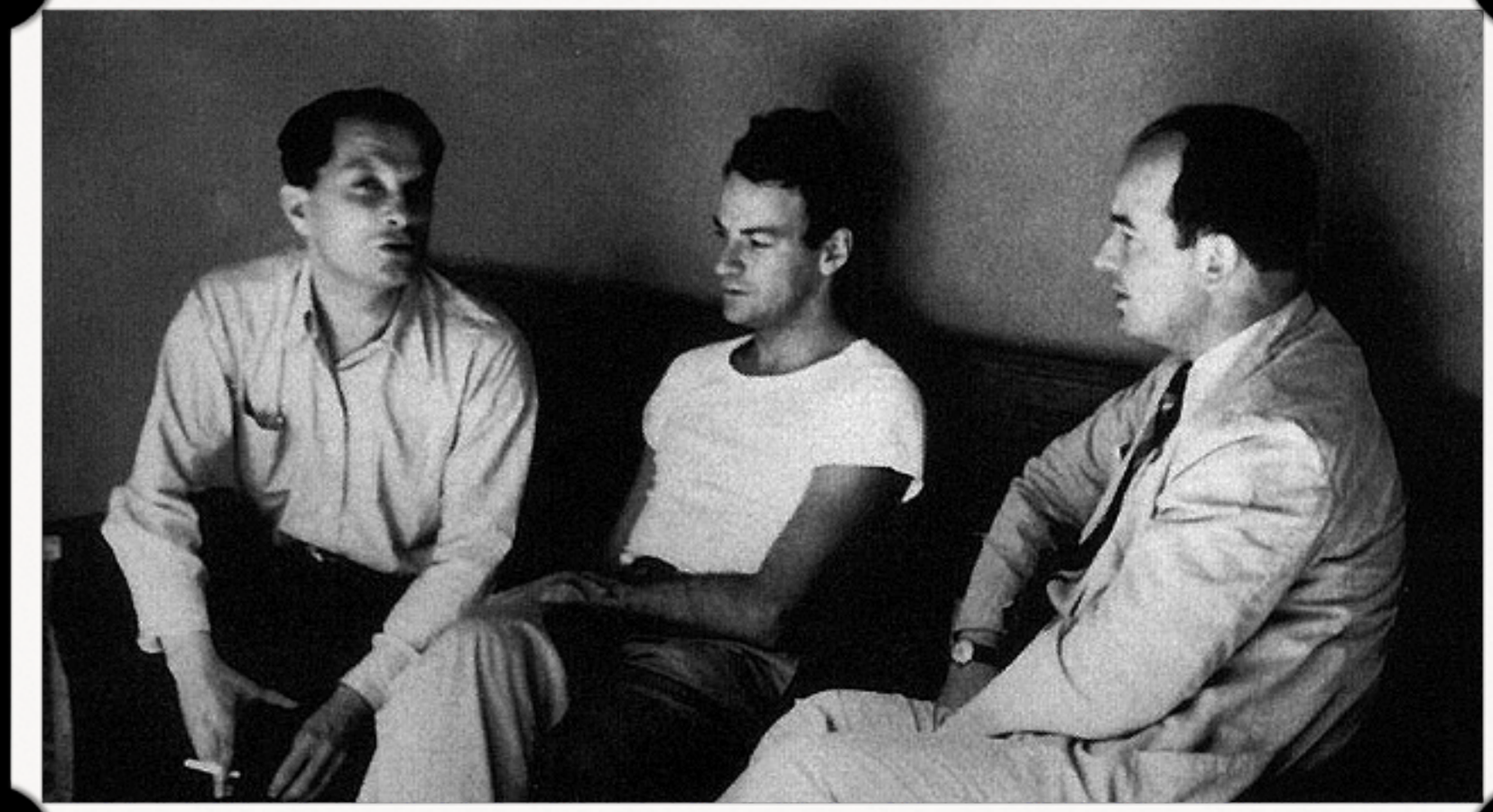
"the first post-devops architectural style"

Neal Ford

replaceable component architectures

Dan North

the future is scary



*"**ever accelerating** progress of technology and changes in the mode of human life, which gives the appearance of approaching some essential **singularity** in the history of the race beyond which human affairs, **as we know them, could not continue**"*

John von Neumann, as recorded by Ulam, 1958

Beyond the Polar Bear, best talk of the day along with the keynote! #gotoaar

Rasmus Christensen, 19 minutes ago

RT @bakjakob:
Attending inspiring
#UX sessions at
#gotoaar

Singularity

JavaScript Singularity

JavaScript

Singularity

Container

Log aggregation

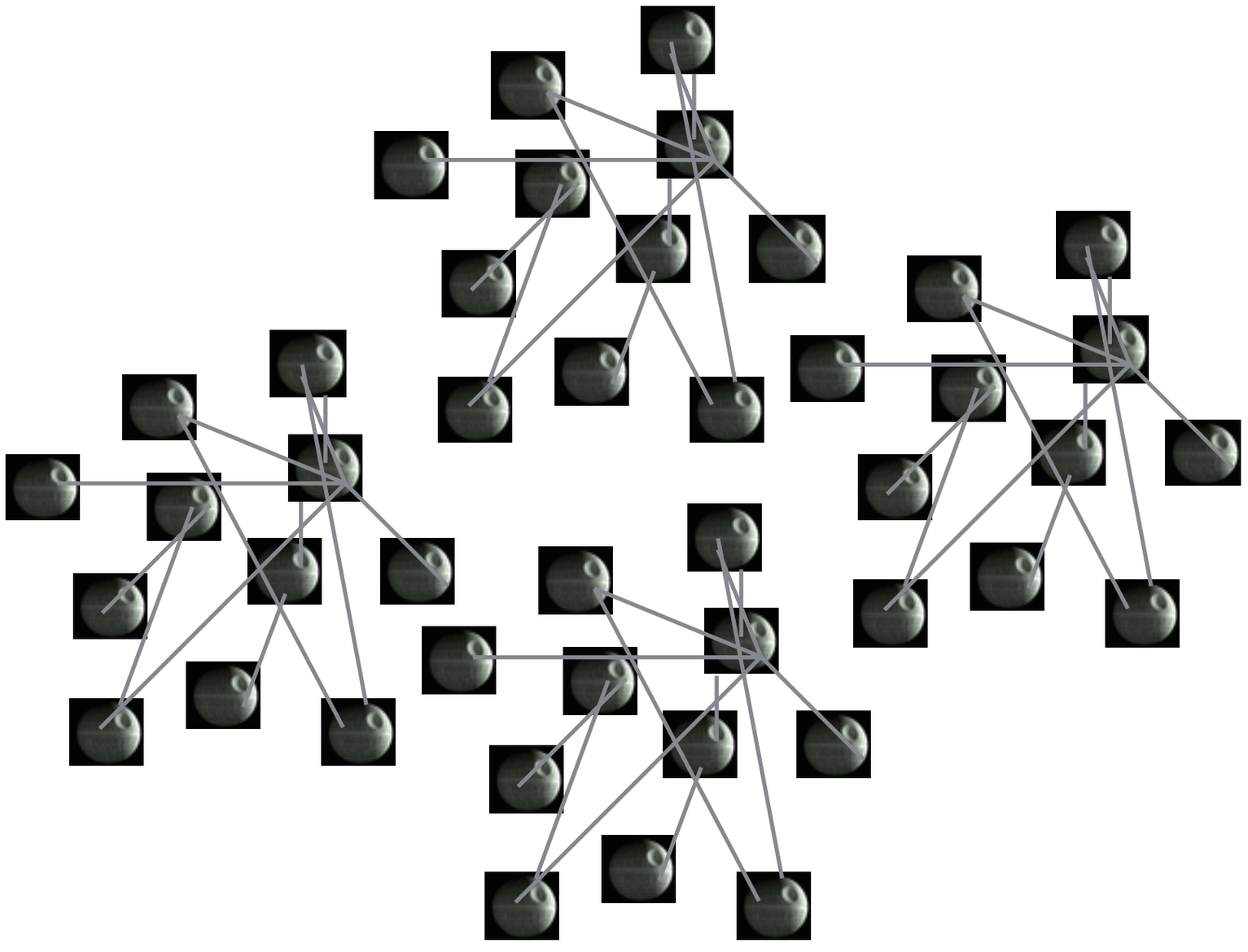
JavaScript

Singularity

Container

even closer to home

HOW WE DESIGN SOFTWARE IS CHANGING



Microservices

The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

---

25 March 2014

James Lewis

James Lewis is a Principal Consultant at ThoughtWorks and member of the Technology Advisory Board. James' interest in building applications out of small collaborating services stems from a background in integrating enterprise systems at scale. He's built a number of systems using microservices and has been an active participant in the growing community for a couple of years.

Martin Fowler

Martin Fowler is an author, speaker, and general loud-mouth on software development. He's long been puzzled by the problem of how to componentize

Contents

Hardest things to do:

Hardest things to do: End-to-end testing

Hardest things to do:

End-to-end testing

Independent deployment

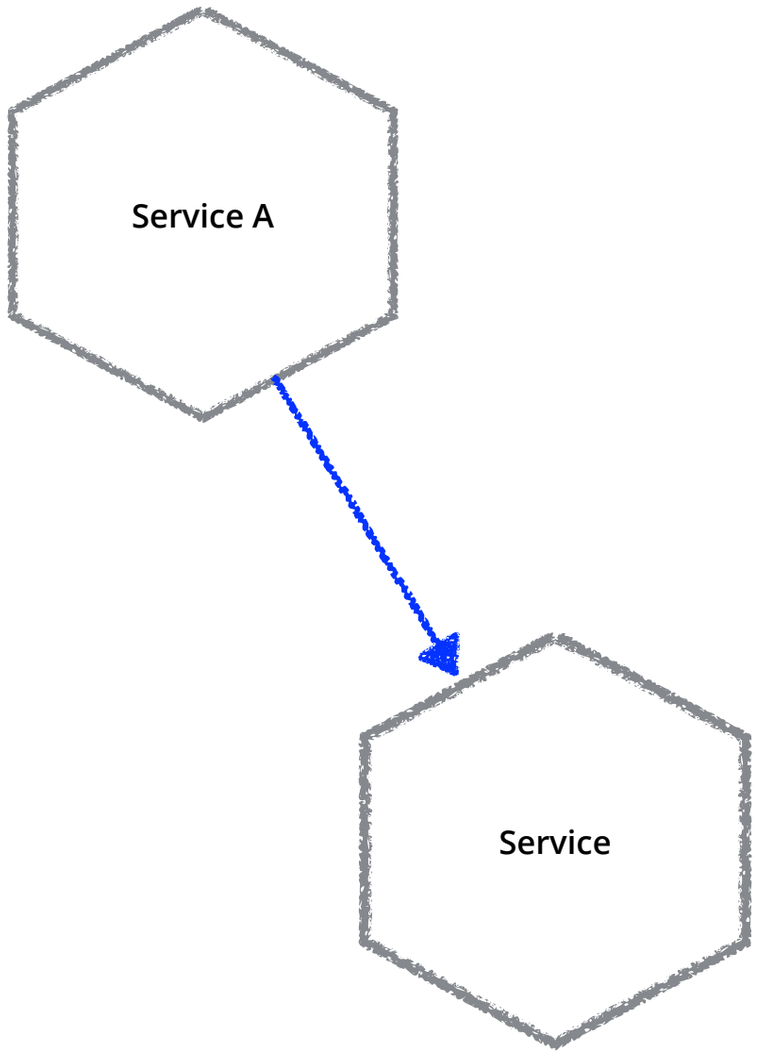
Hardest things to do:

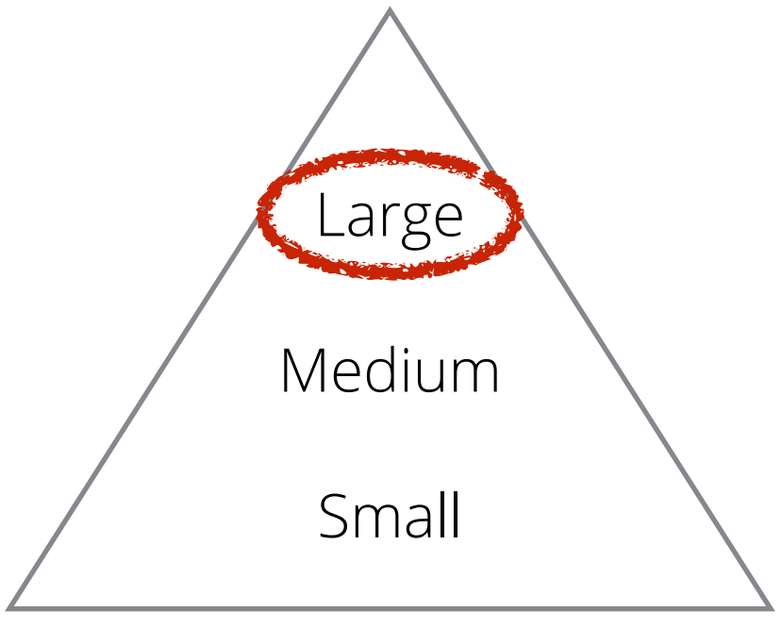
End-to-end testing

Independent deployment

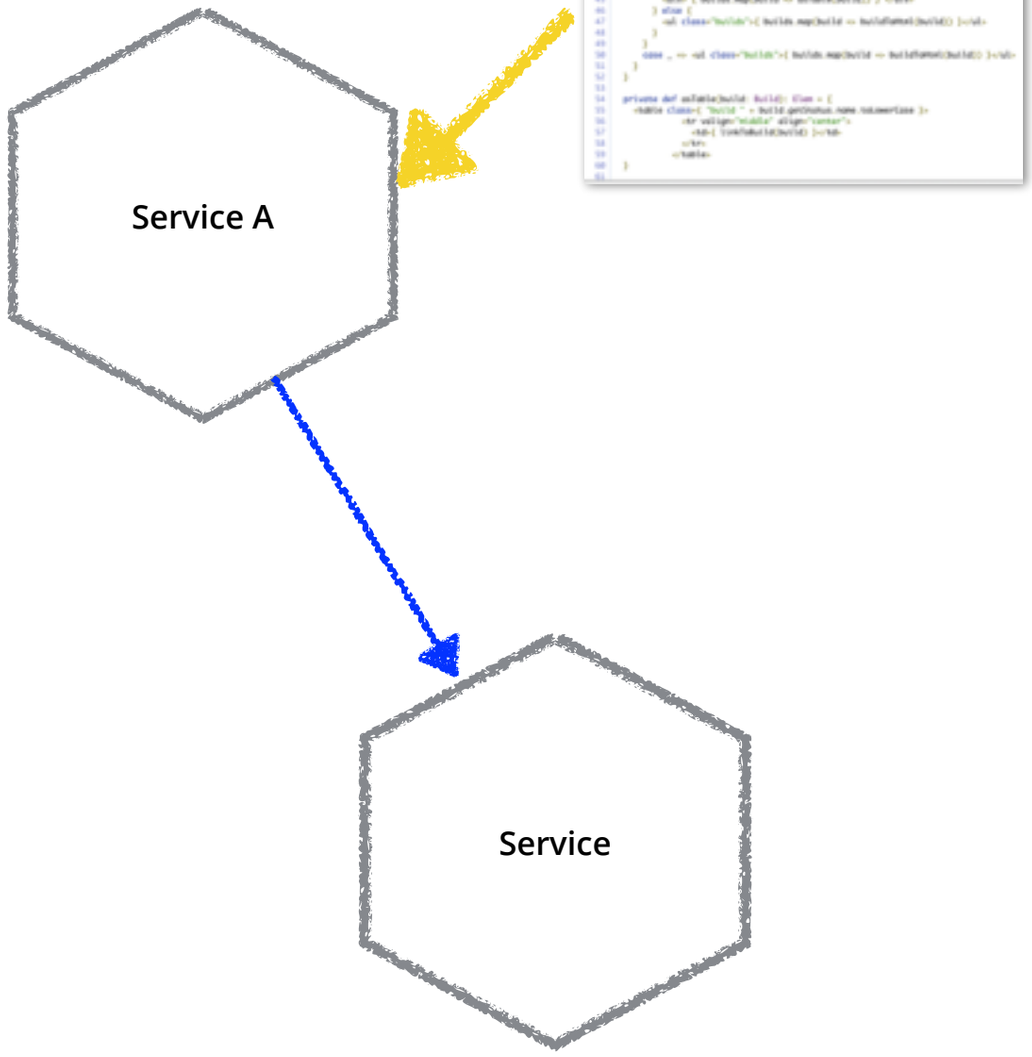
Service versioning / evolution

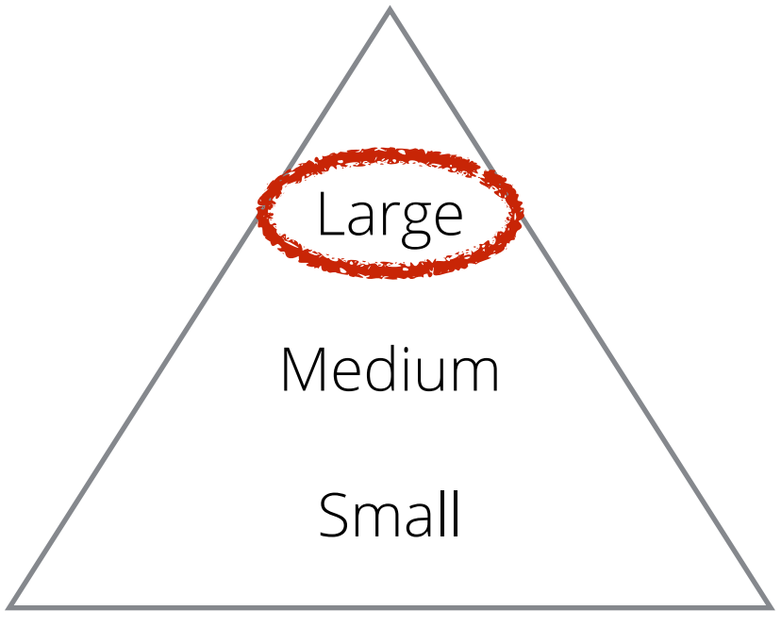
TESTING MICROSERVICES IS HARD



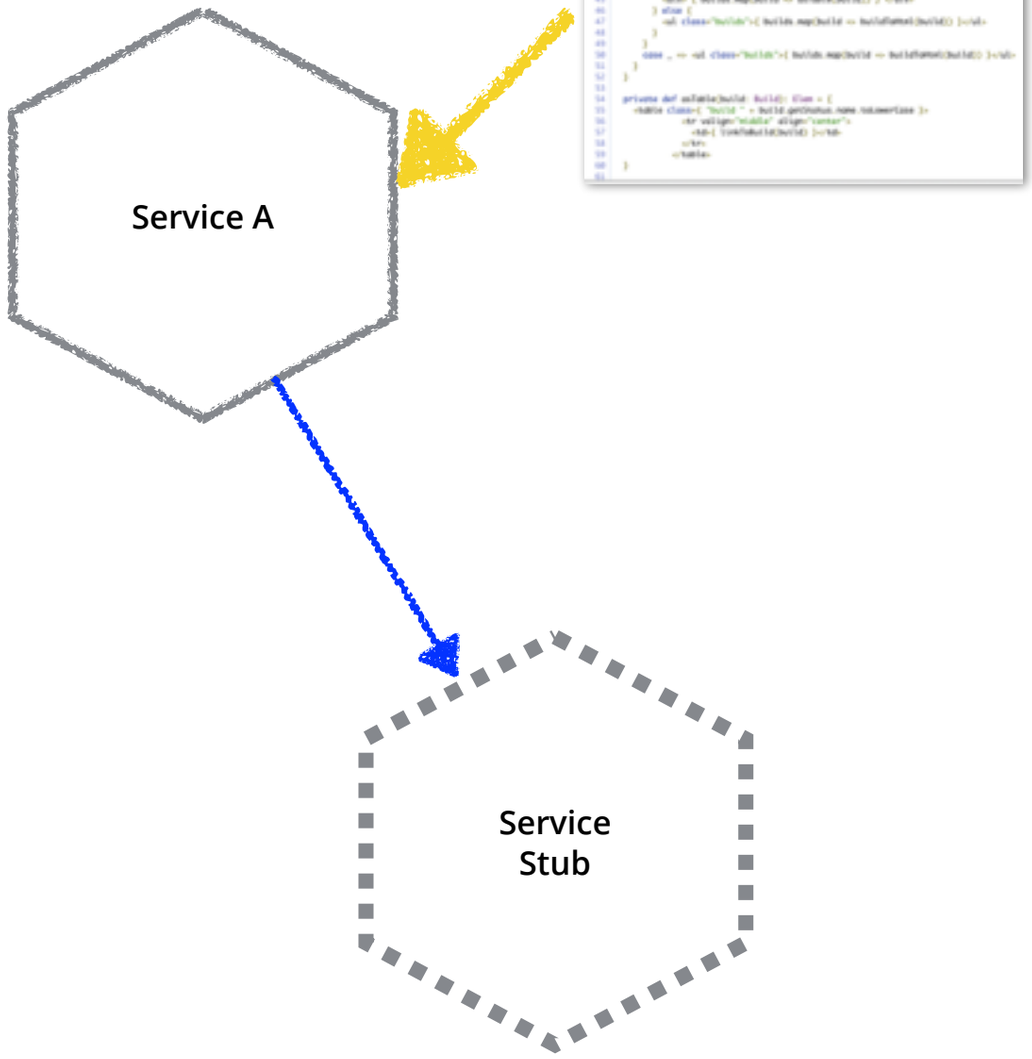


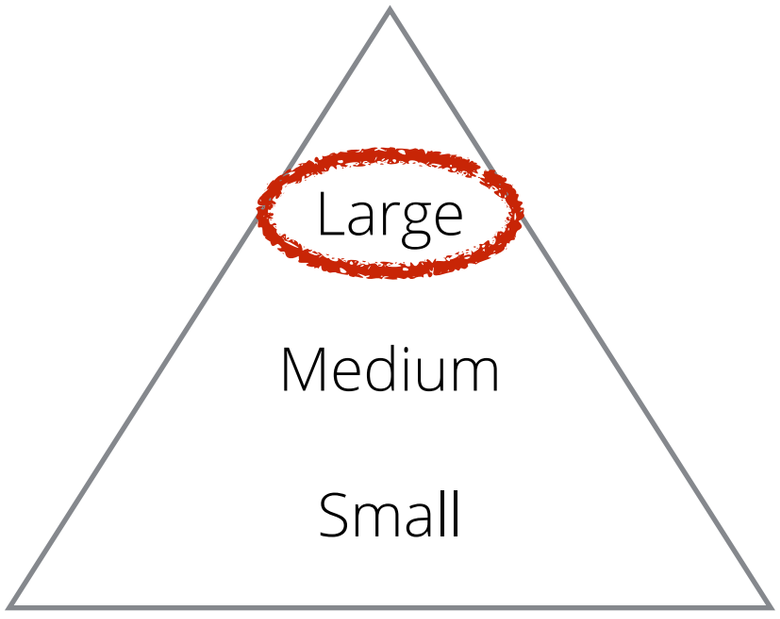
TESTING MICROSERVICES IS HARD





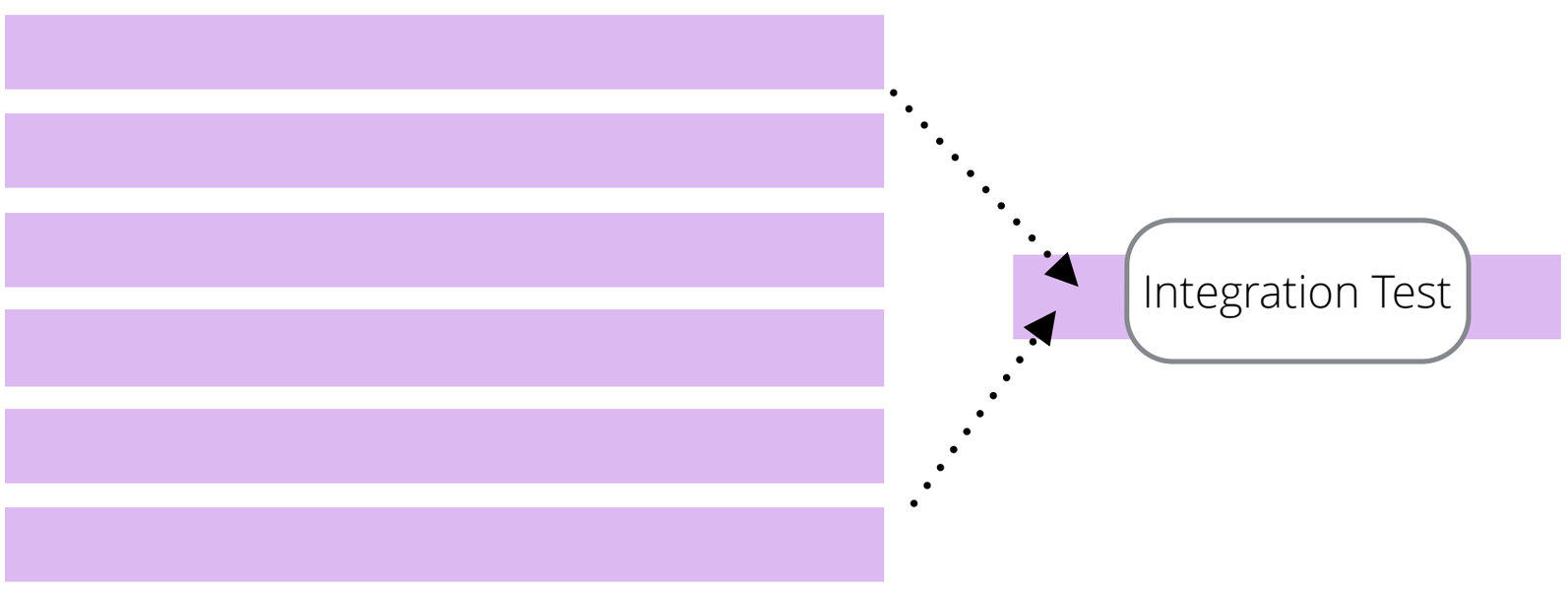
TESTING MICROSERVICES IS HARD



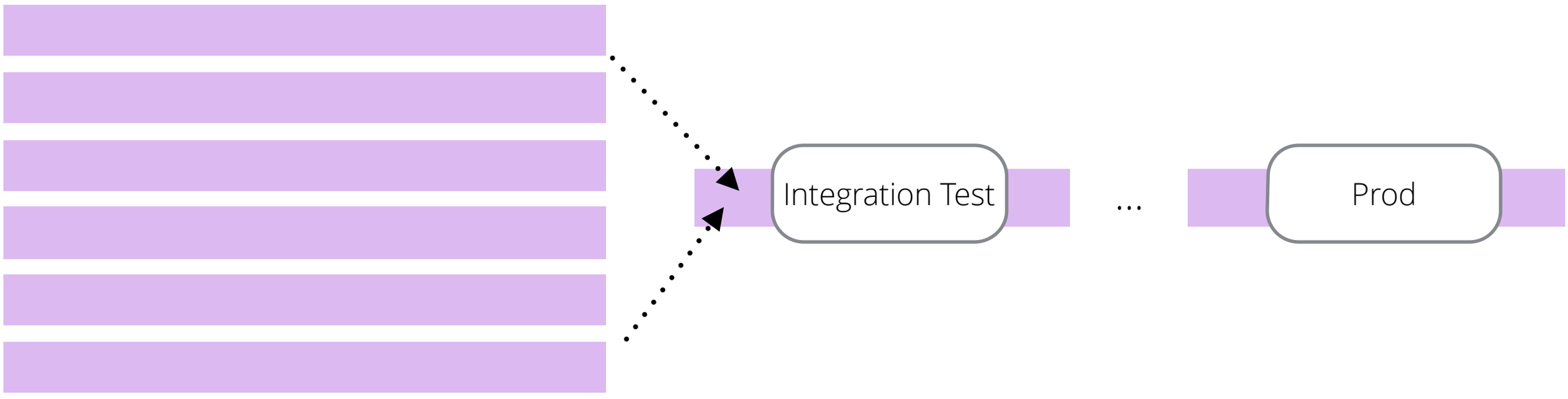


INTEGRATING MICROSERVICES IS HARD

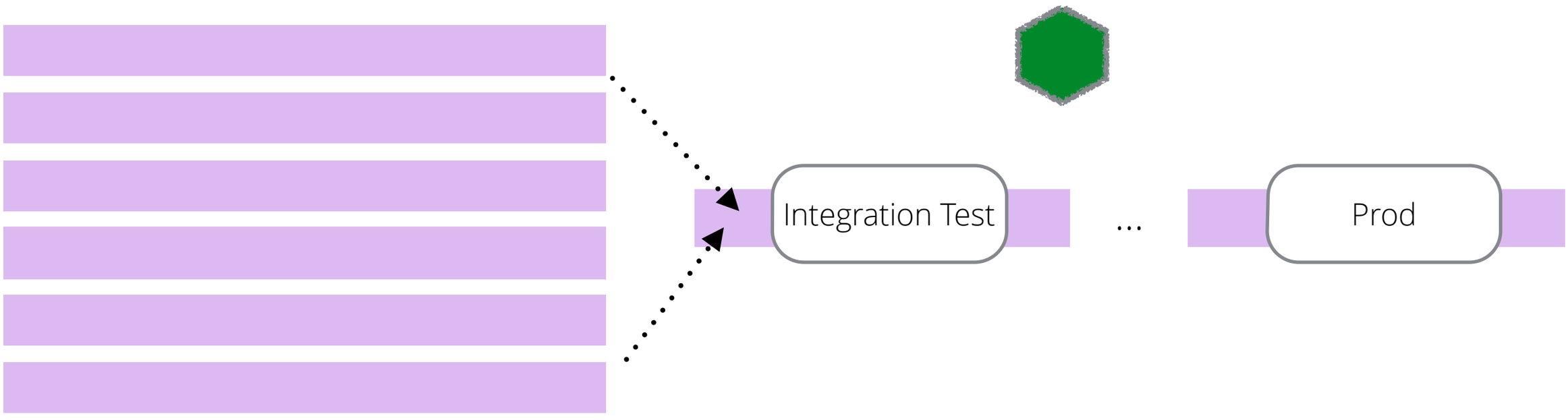
INTEGRATING MICROSERVICES IS HARD



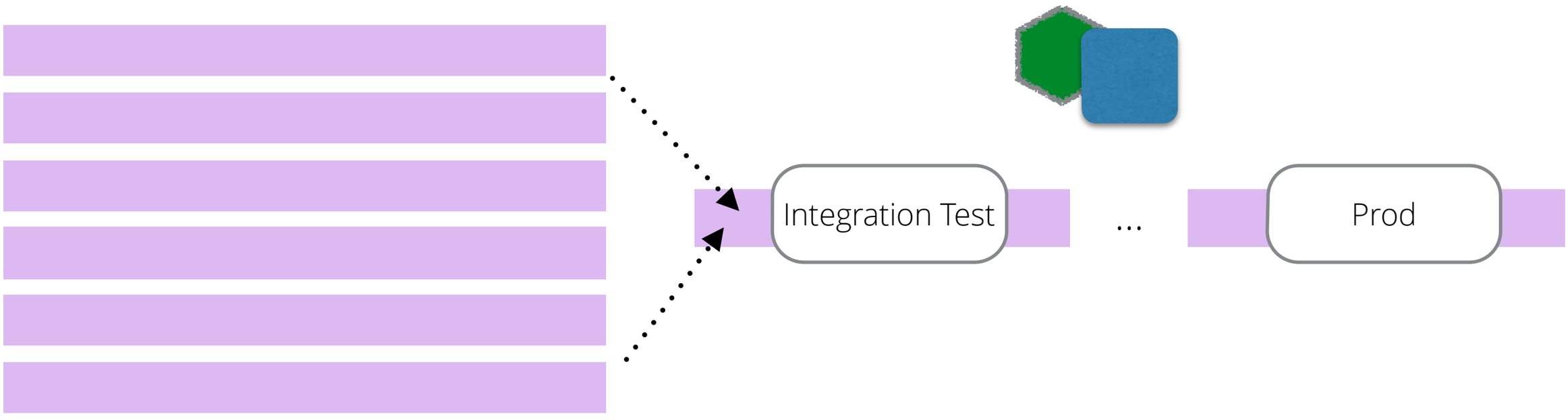
INTEGRATING MICROSERVICES IS HARD



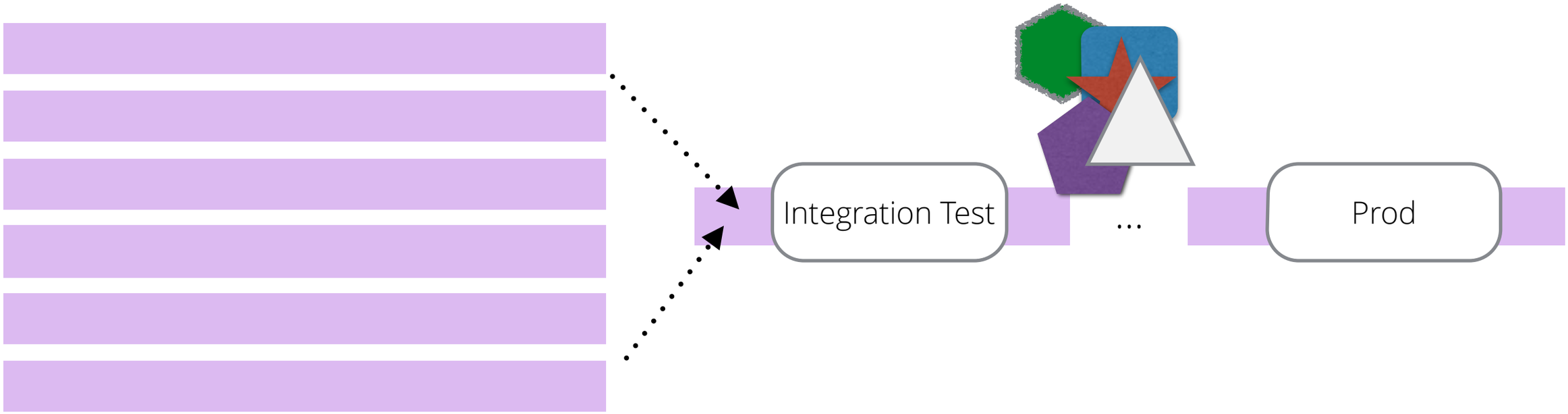
INTEGRATING MICROSERVICES IS HARD



INTEGRATING MICROSERVICES IS HARD



INTEGRATING MICROSERVICES IS HARD




<thinks>

GRASP

YAGNI

World of Warcraft

SOLID

agile

DRY

BDD

emergent design

GoF

Continuous Delivery

TDD

XP

KISS

Refactoring

Gemini Project, Rogallo wing

Source: wikipedia.org


<thinks>

it's turtles all the way down

Gemini Project, Rogallo wing

Source: wikipedia.org

GRASP

YAGNI

World of Warcraft

SOLID

agile

DRY

BDD

emergent design

GoF

Continuous Delivery

TDD

XP

KISS

Refactoring

GRASP

World of Warcraft

SOLID

agile

DRY

BDD

YAGNI

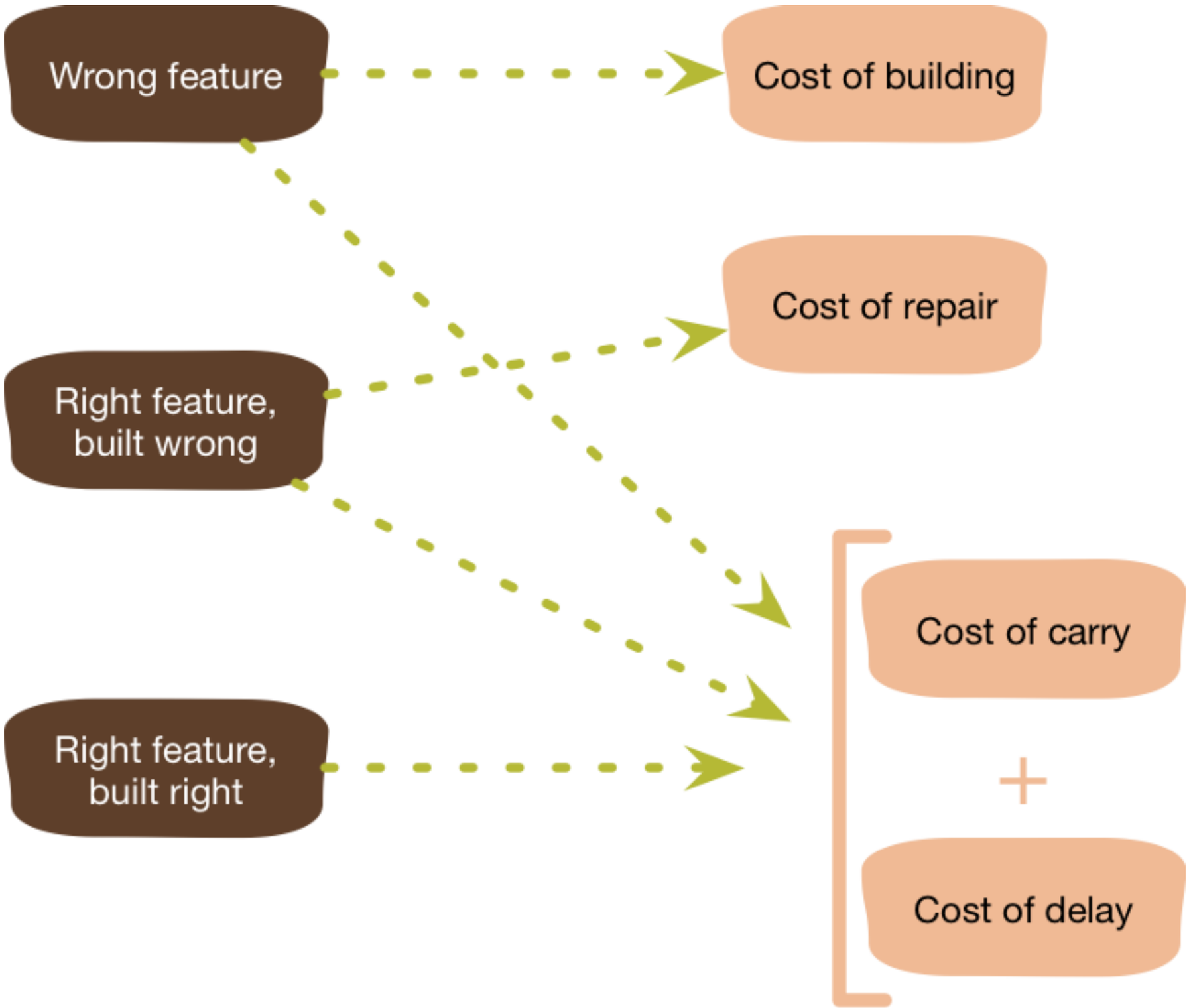
Continuous Delivery

TDD

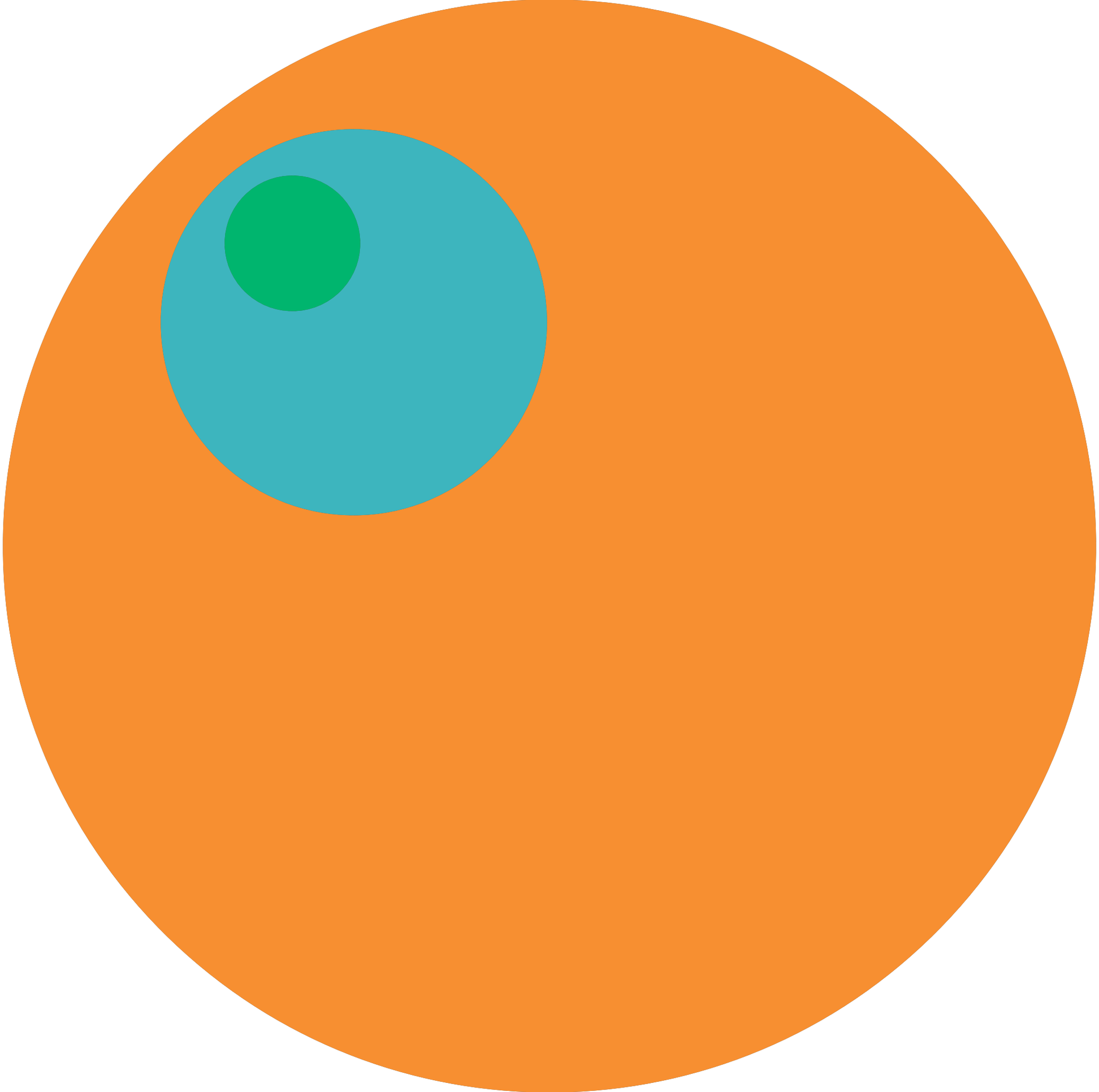
XP

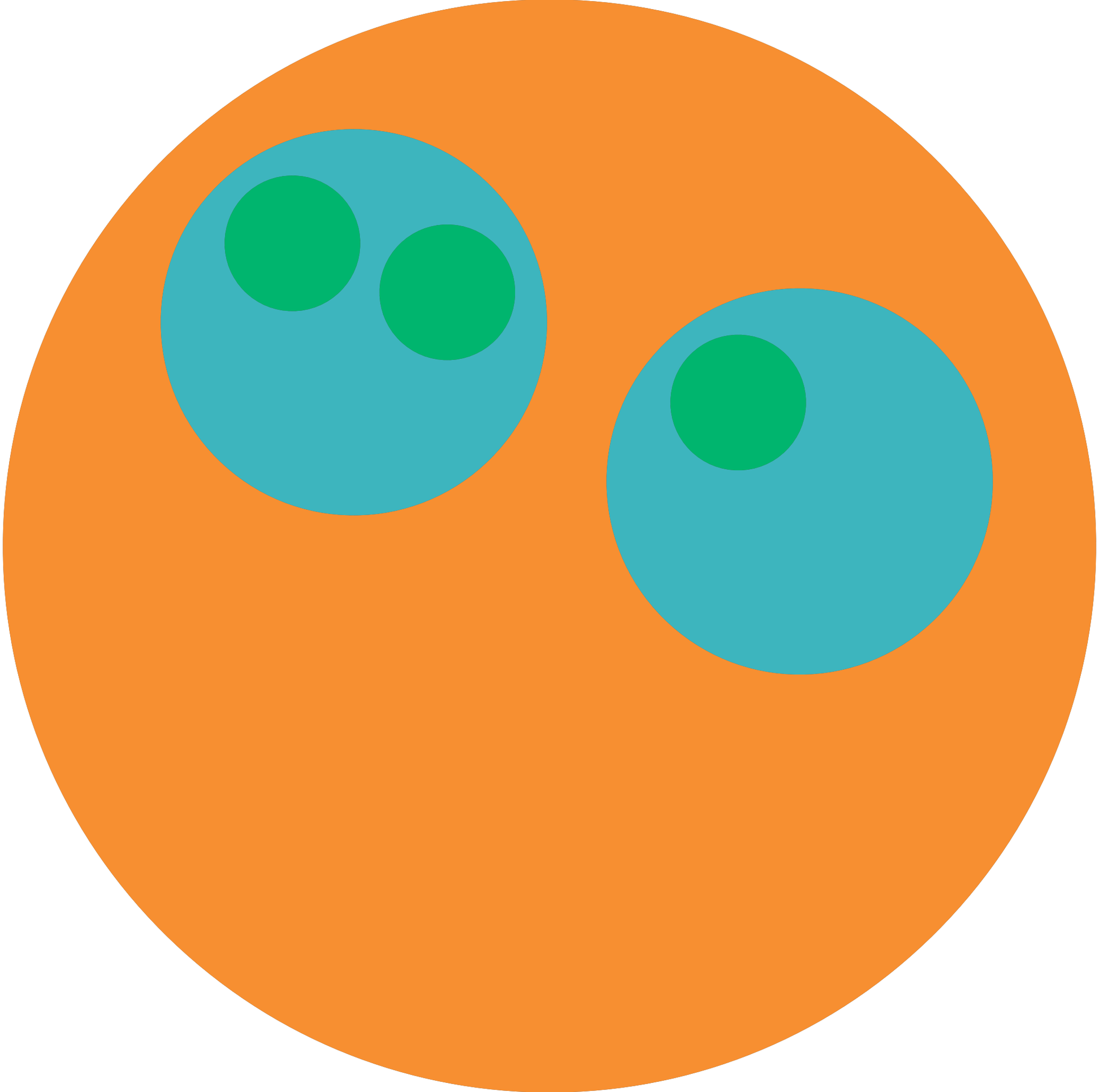
KISS

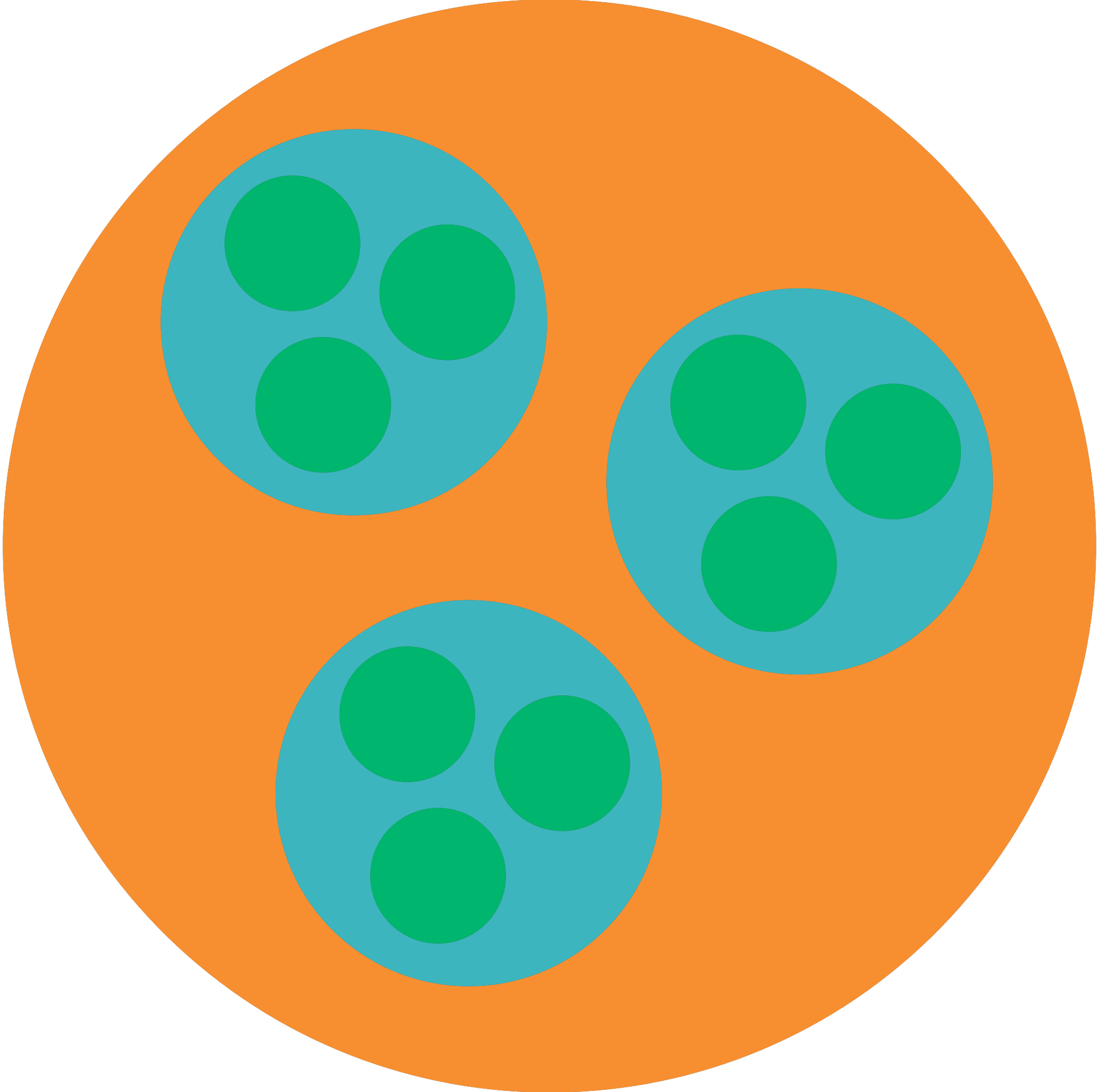
Refactoring



http://martinfowler.com/bliki/Yagni.html







Build out services as you need them

YAGNI

World of Warcraft

SOLID

agile

DRY

GRASP

BDD

emergent design

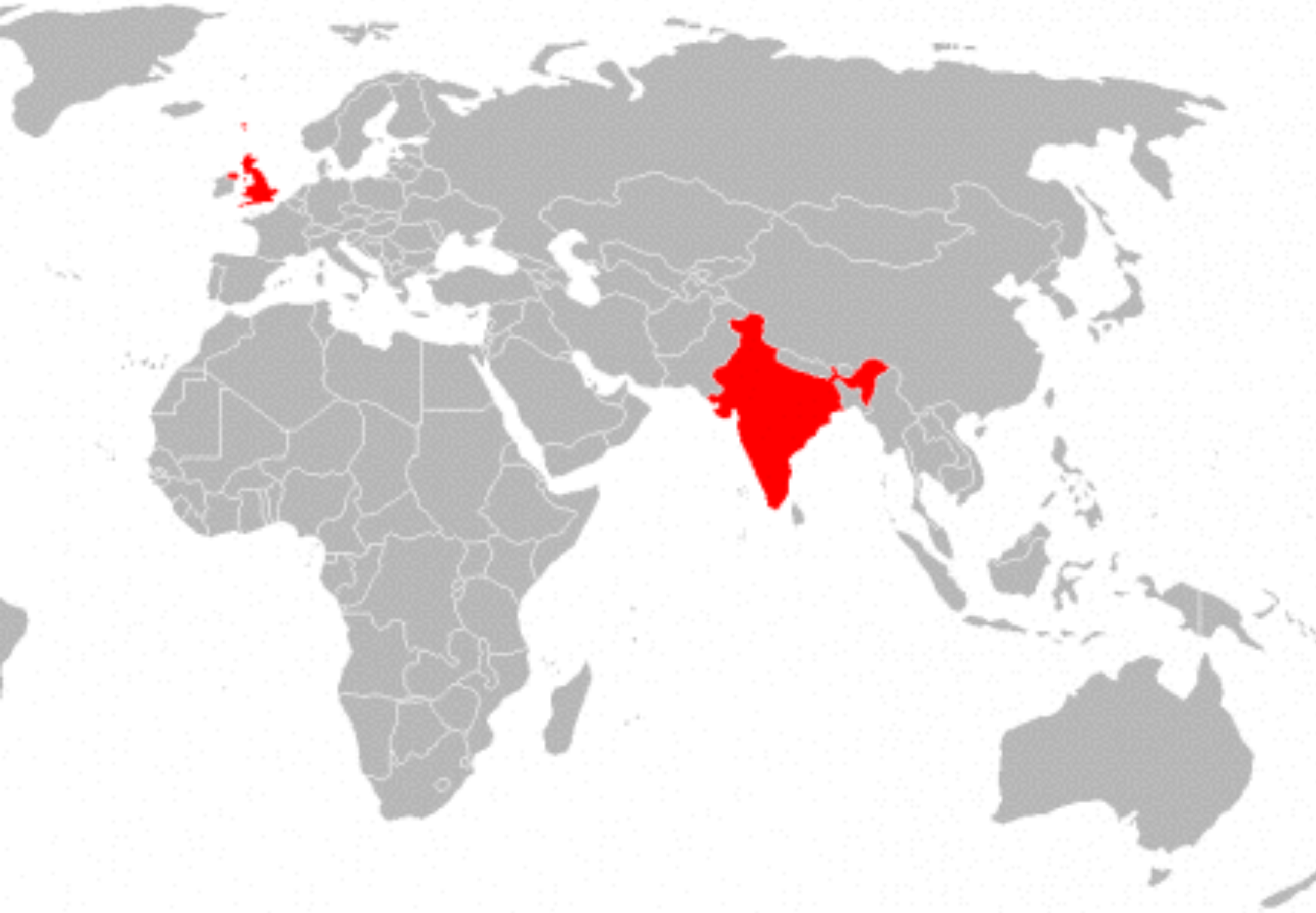
Continuous Delivery

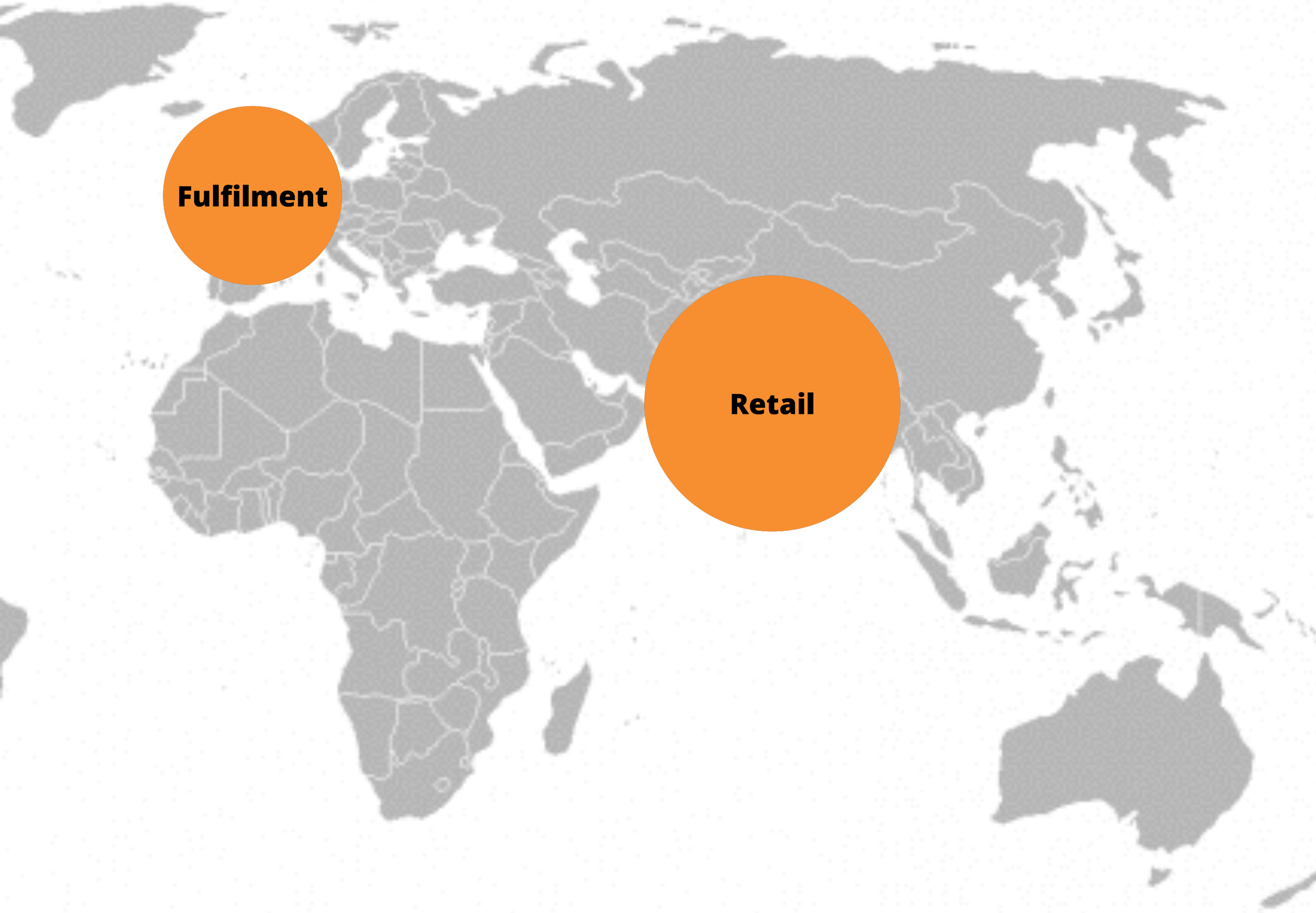
TDD

XP

KISS

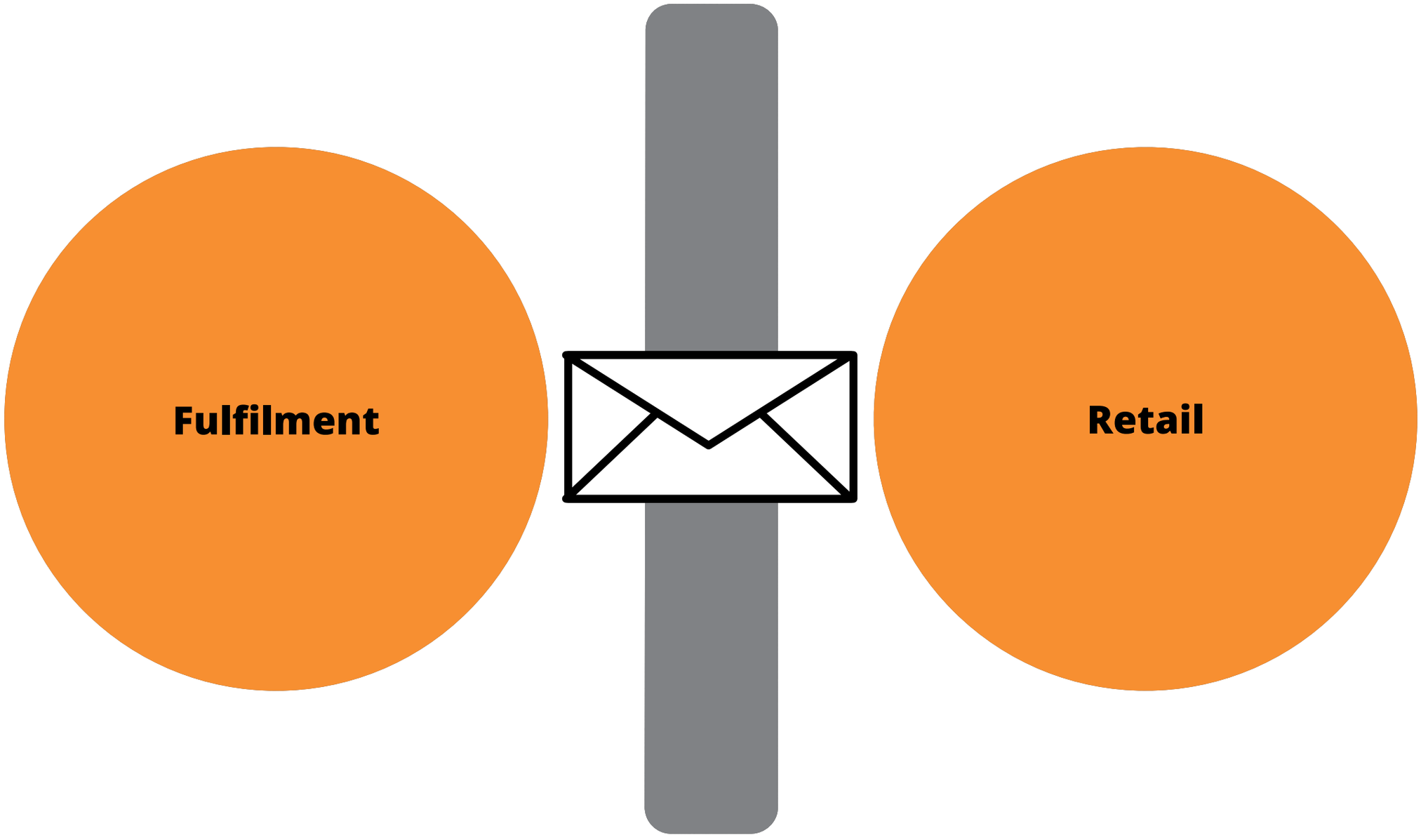
Refactoring

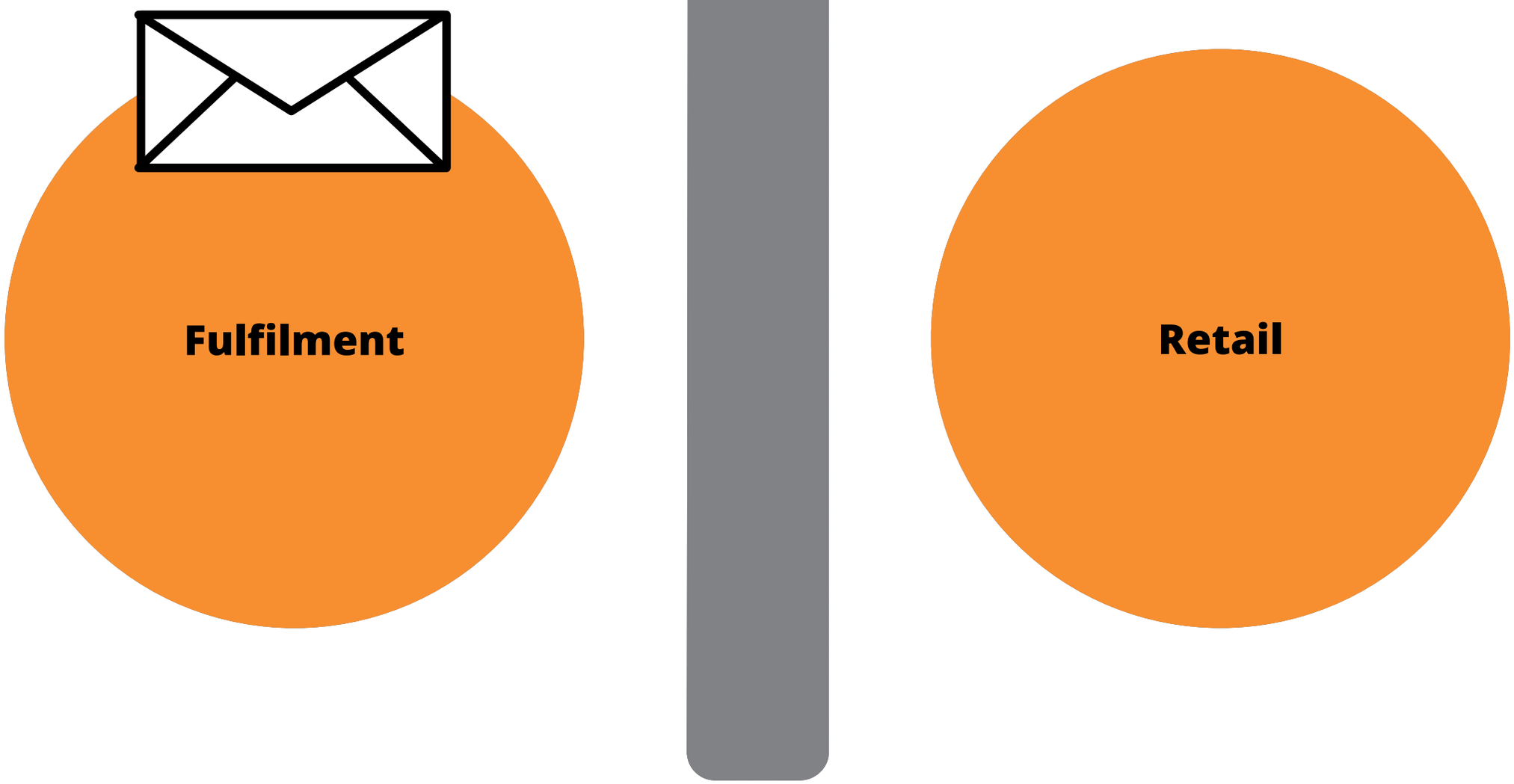



Fulfilment
Retail

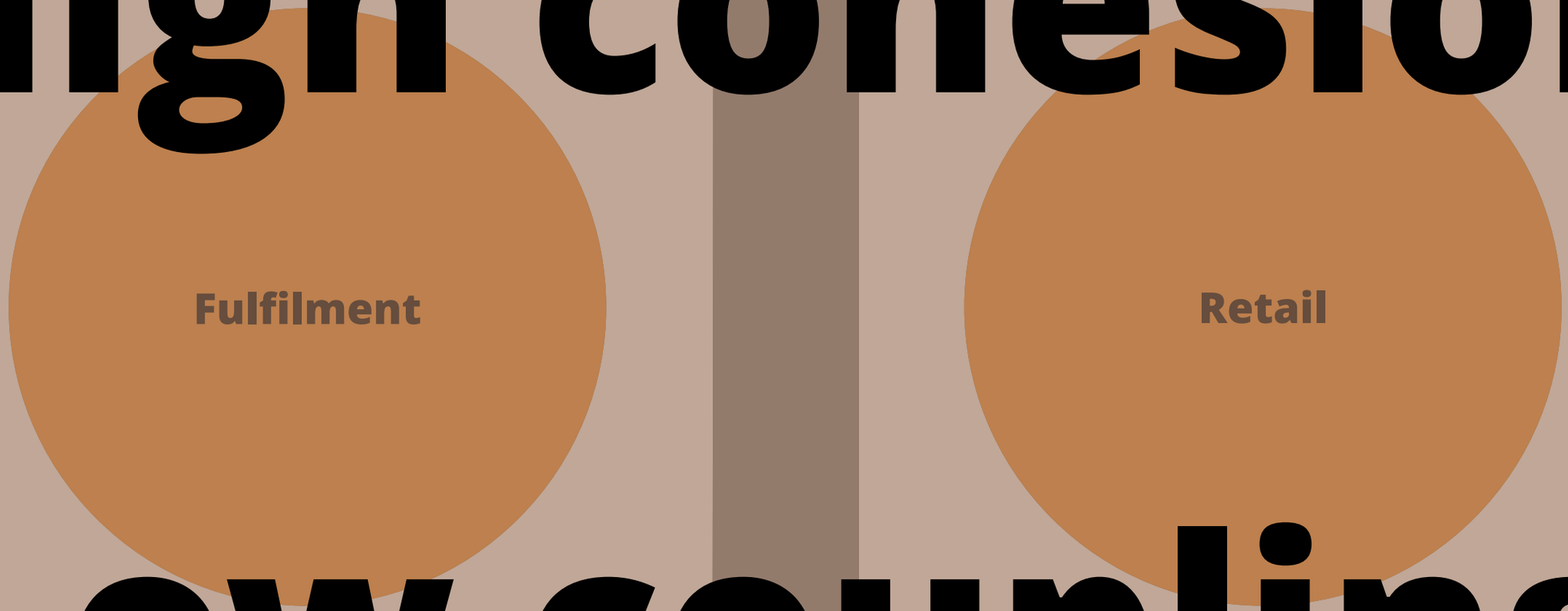
Fulfilment

Retail


Fulfilment
Retail


Fulfilment
Retail

High cohesion



Low coupling

(incidentally, if you were playing the Conway's law lottery, that's when you number came up)

GRASP

YAGNI

World of Warcraft

SOLID

agile

BDD

design

DRY

GoF

Continuous Delivery

TDD

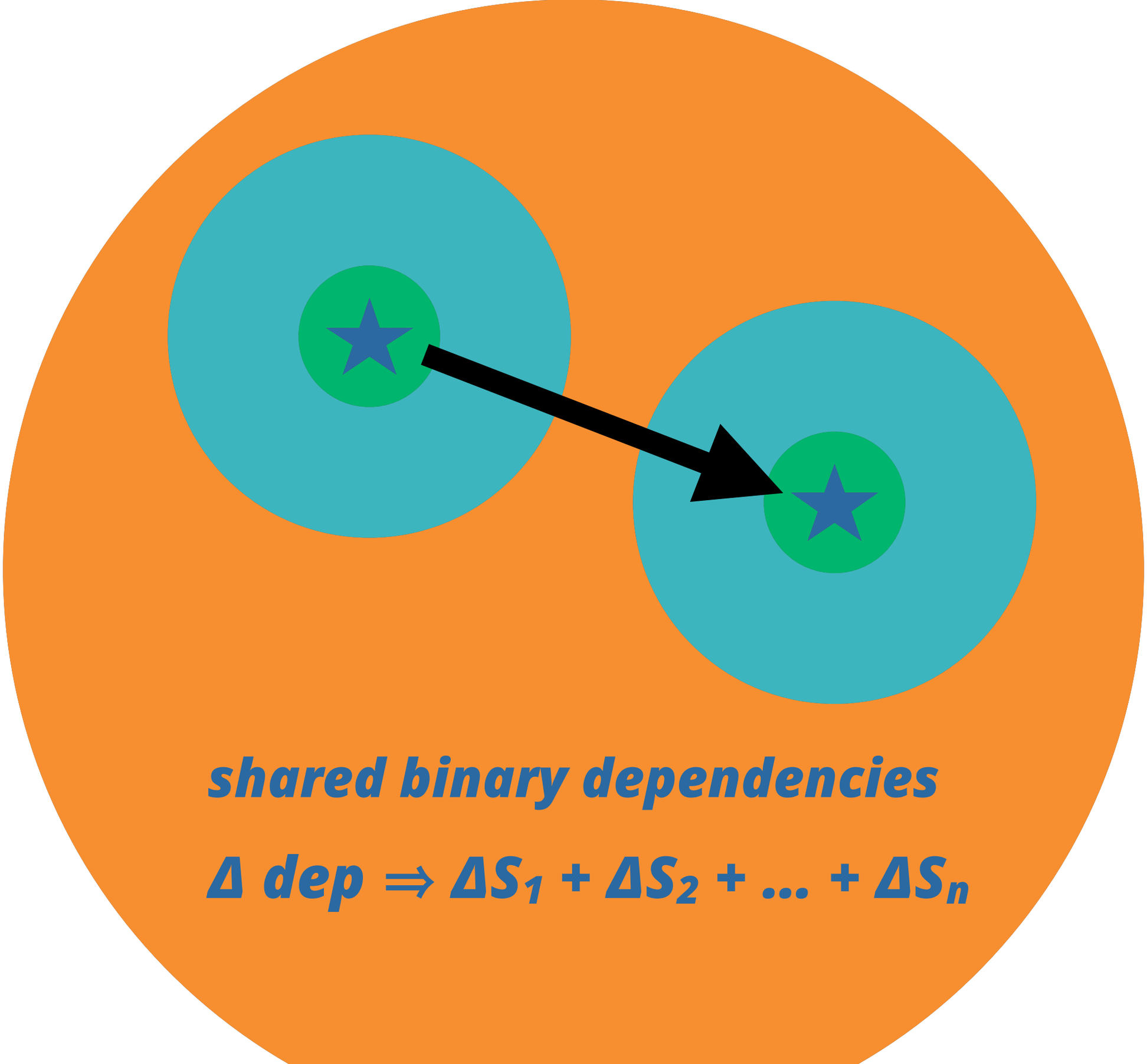
XP

KISS

Refactoring

“Every piece of knowledge must have a single, unambiguous, authoritative representation within a system”

Dave Thomas, interviewed by Bill Venners (2003-10-10). "Orthogonality and the DRY Principle". Retrieved 2006-12-01.

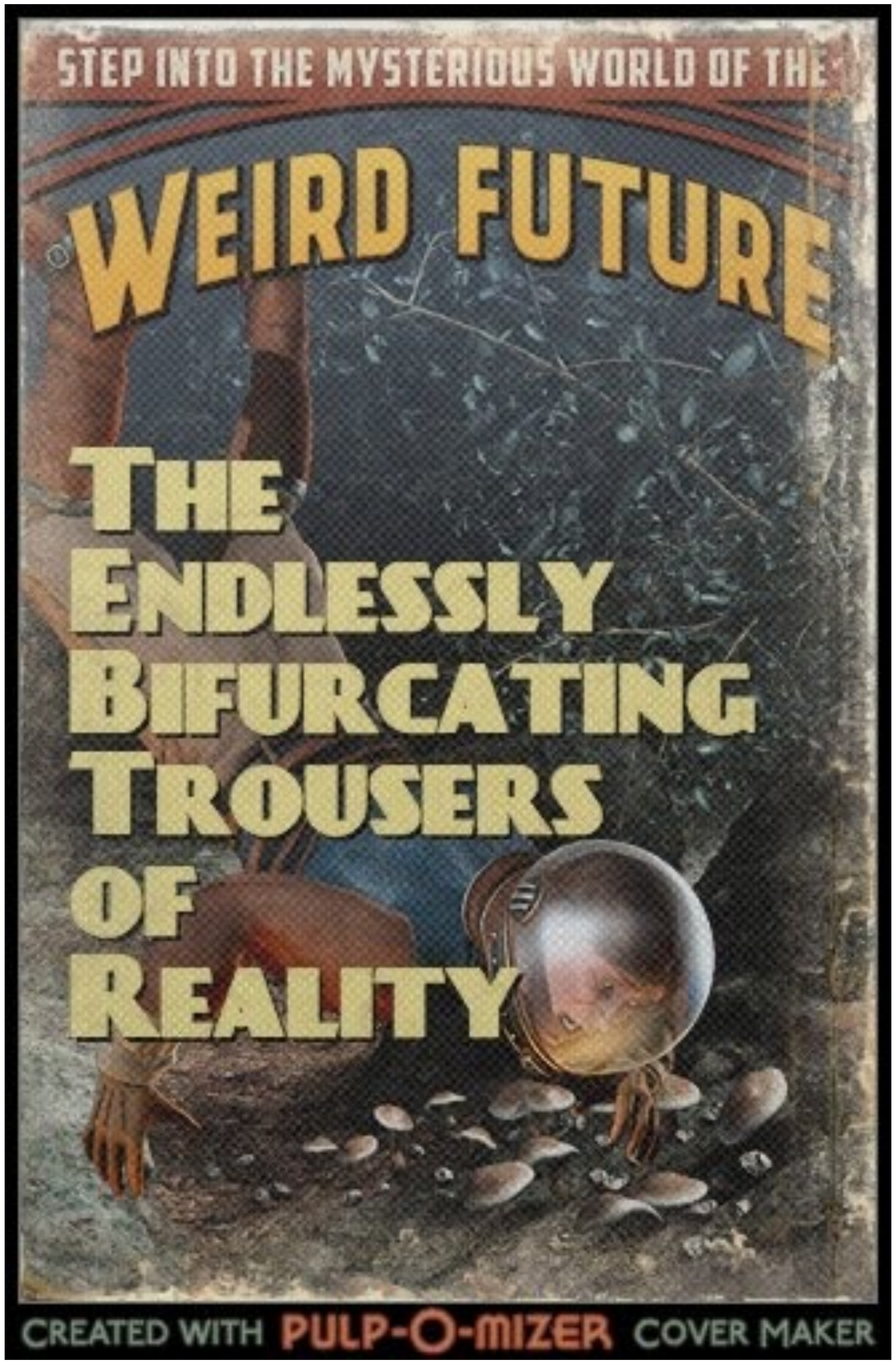


shared binary dependencies

$$\Delta\ dep \Rightarrow \Delta S_1 + \Delta S_2 + \ldots + \Delta S_n$$

```
git clone https://github.com/boicy/service-template
```

(note this doesn't exist)


STEP INTO THE MYSTERIOUS WORLD OF THE
WEIRD FUTURE
THE ENDLESSLY BIFURCATING TROUSERS OF REALITY
CREATED WITH PULP-O-MIZER COVER MAKER

DRY within services

duplication between services

GRASP

YAGNI

World of Warcraft

SOLID

agile

DRY

BDD

TDD

merg de

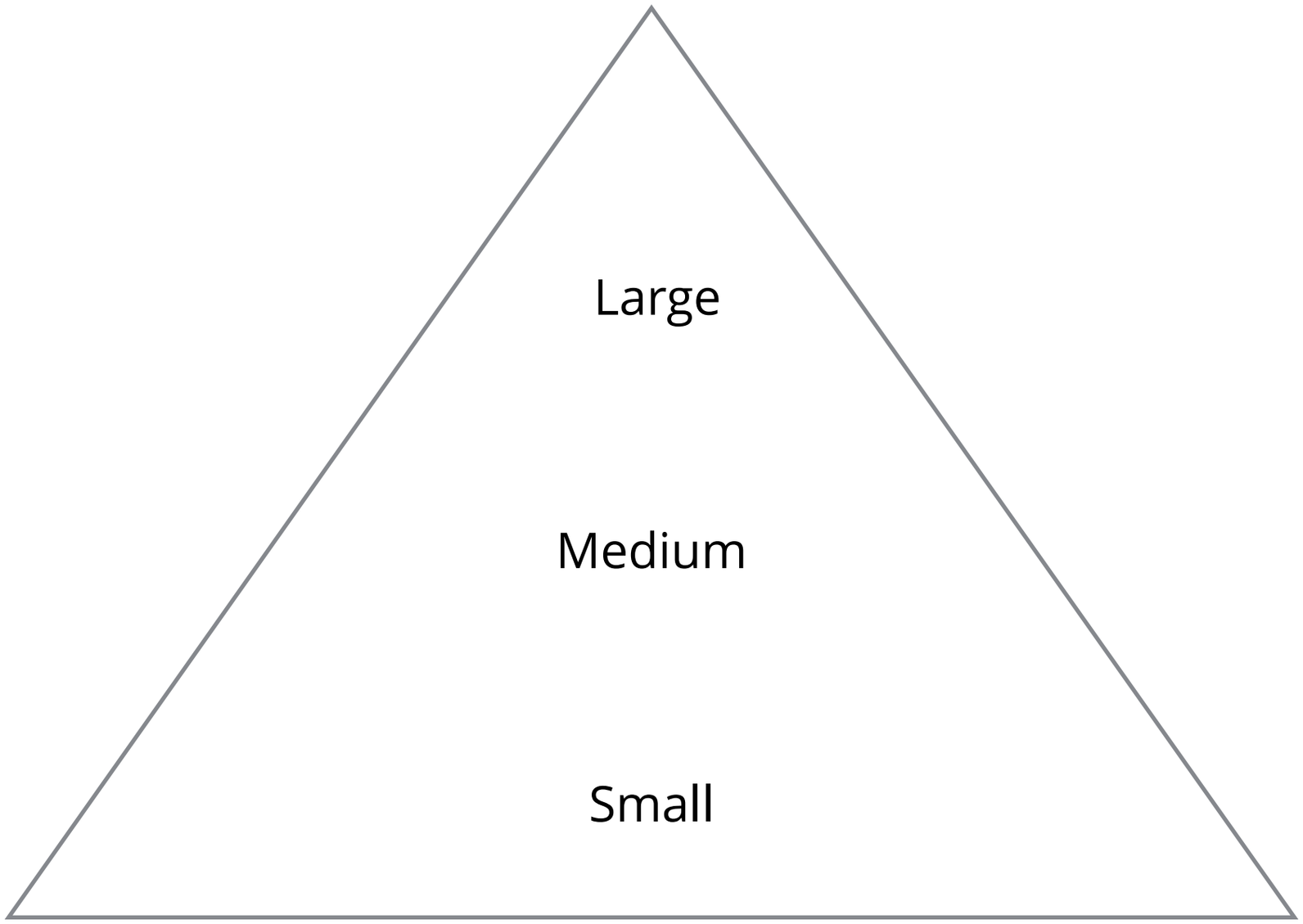
GoF

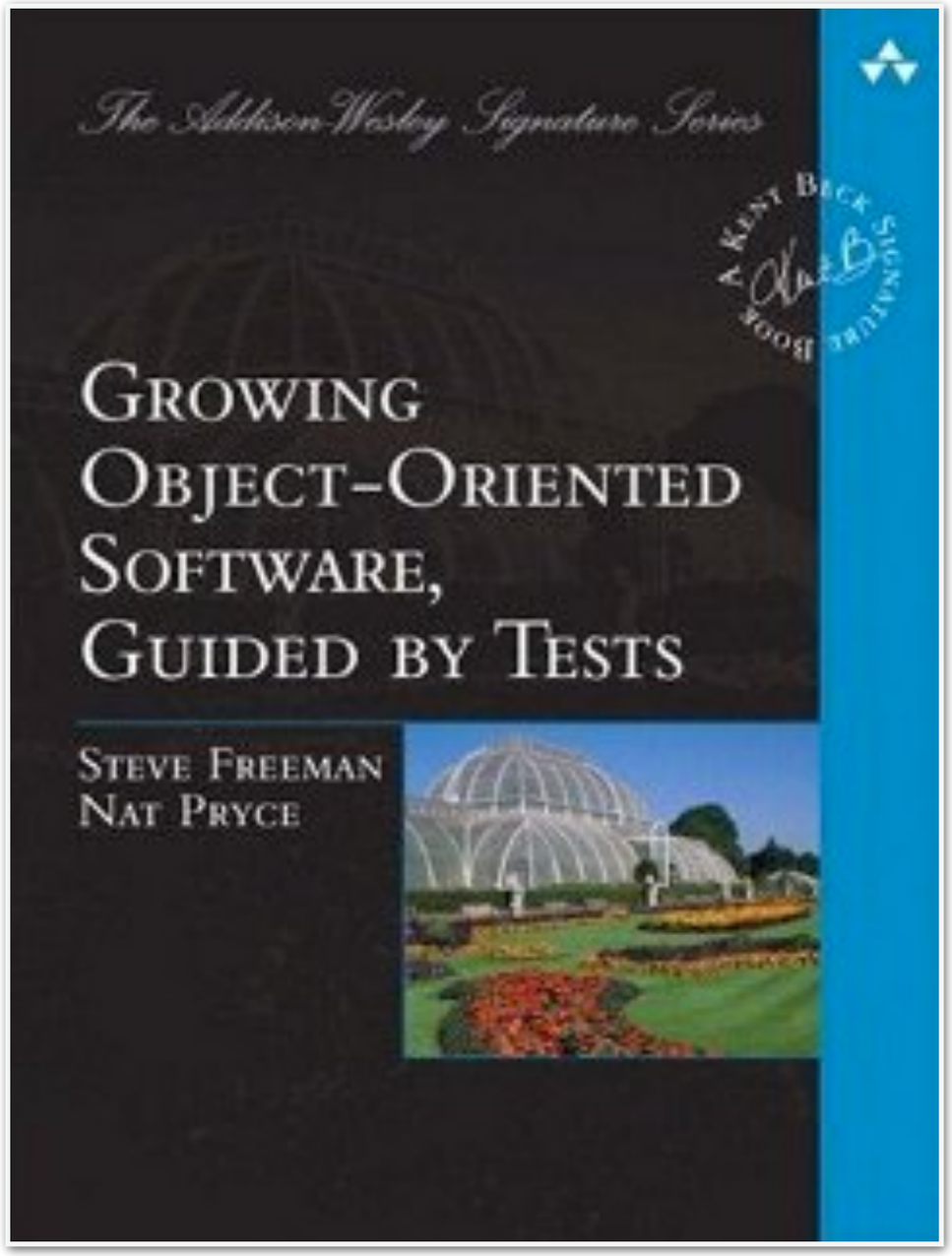
Continuous Delivery

XP

KISS

Refactoring


Large
Medium
Small



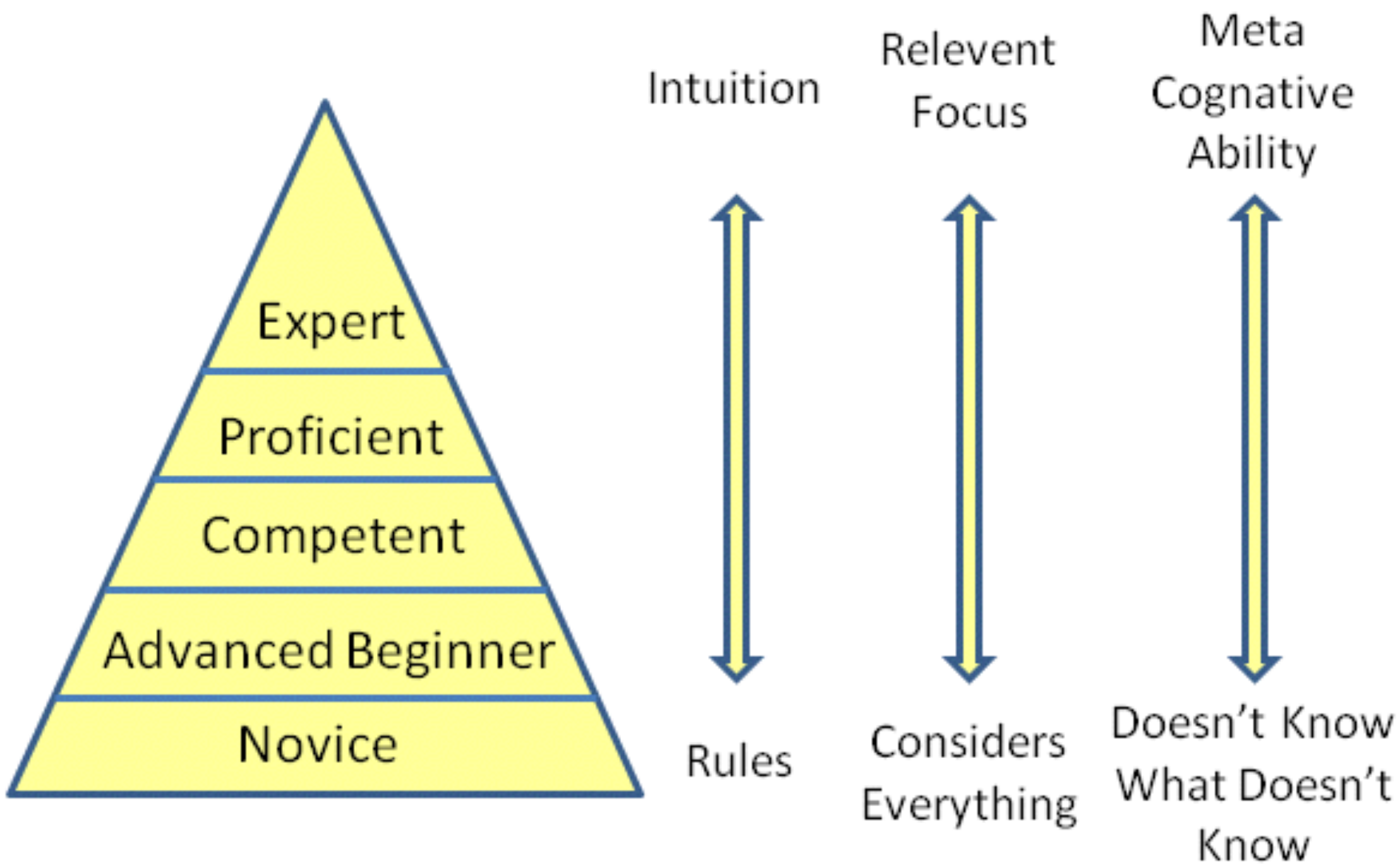
"The London school of Test Driven Development"

Mike Feathers

should we write unit tests?

should bother with test driving our code if we are going to throw it away?

Nat Pryce
Steve Freeman
Dan North
Sydney 'Hoppalong' Redelinghuys
Jim Webber
Ian Robinson
Ivan Moore
Liz Keogh
Simon Stewart
Jez Humble
Dave Farley
Jay Fields
Dan Worthington-Bodart
Joe Walnes



http://moleseyhill.com/blog/2009/08/27/dreyfus-model/

should we write unit tests?

personally I think it’s more important than *ever*

GRASP

YAGNI

World of Warcraft

agile

DRY

BDD

SRP

GoF

Continuous Delivery

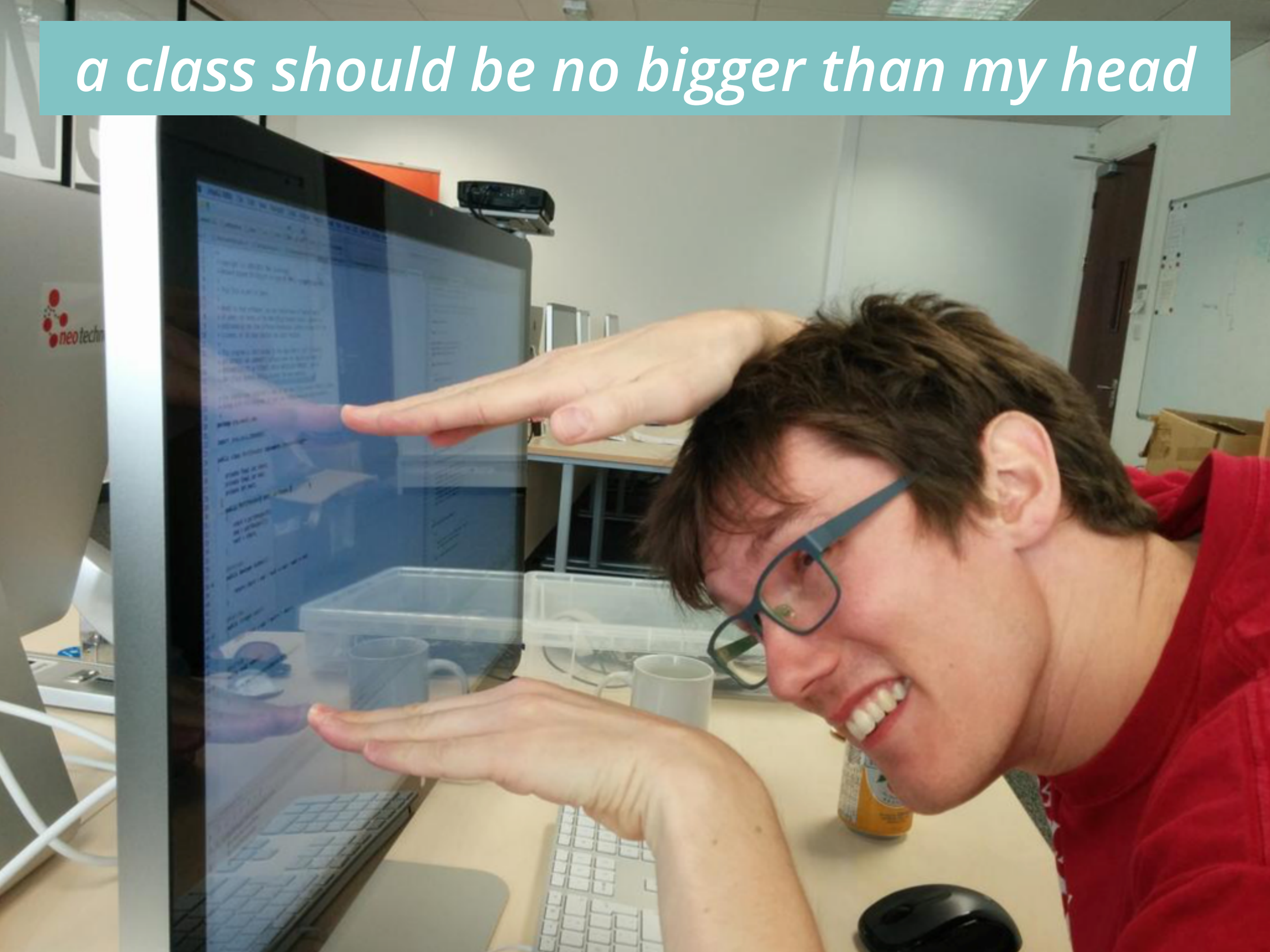
TDD

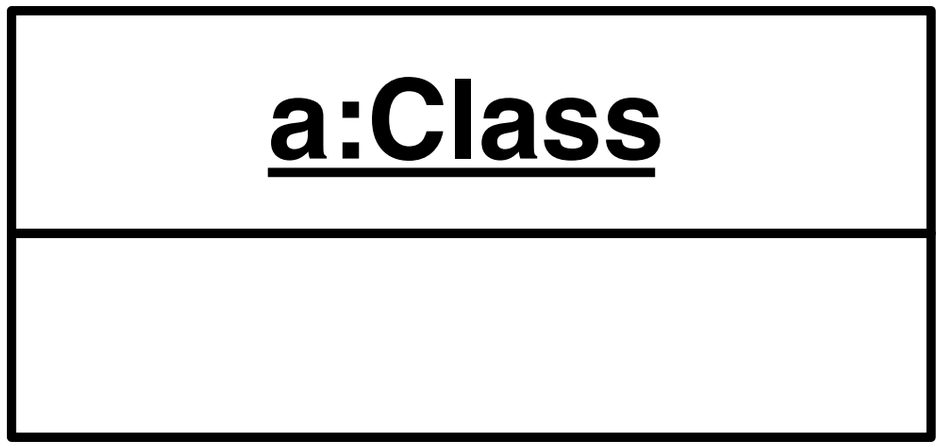
XP

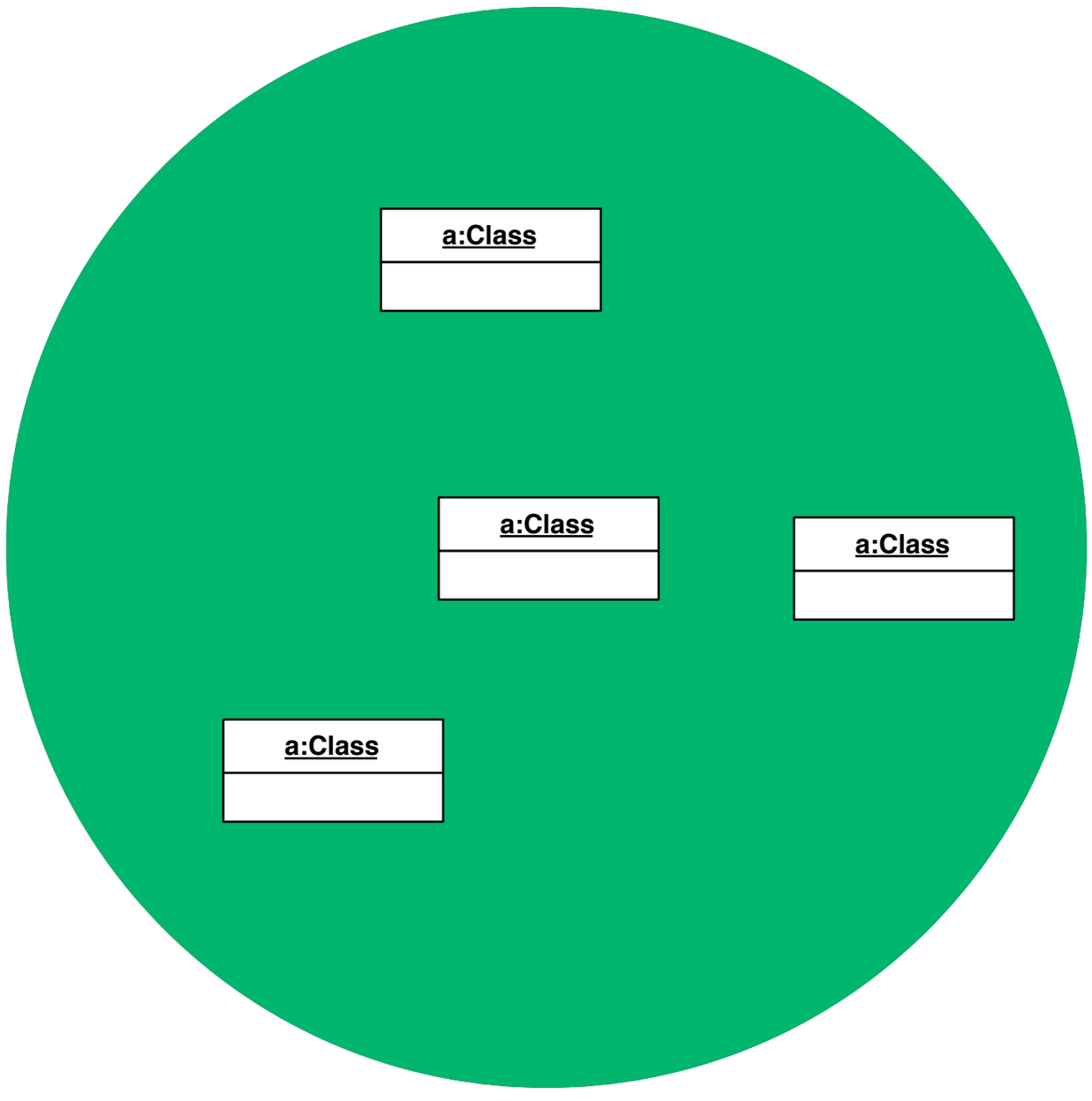
KISS

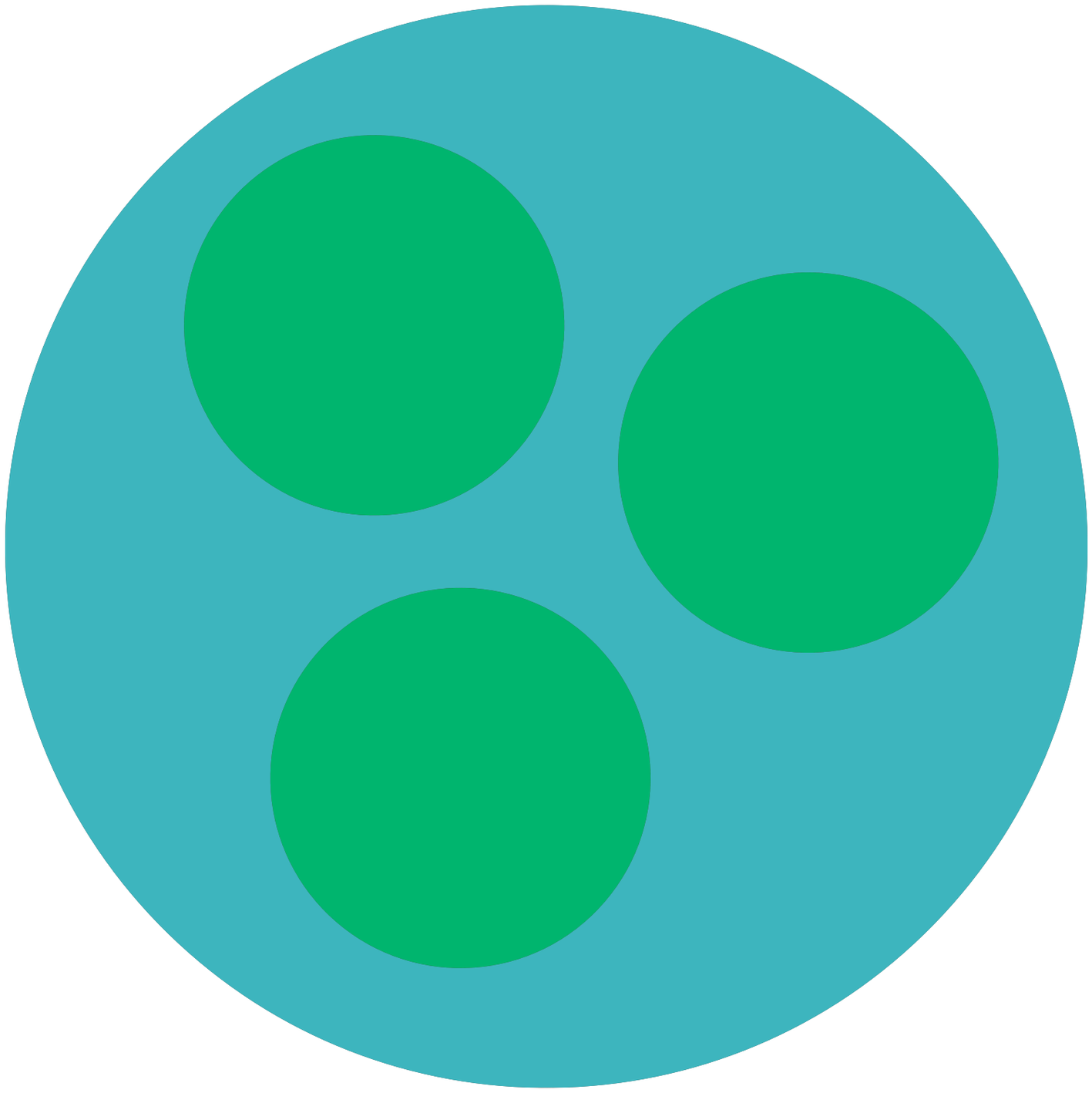
Refactoring

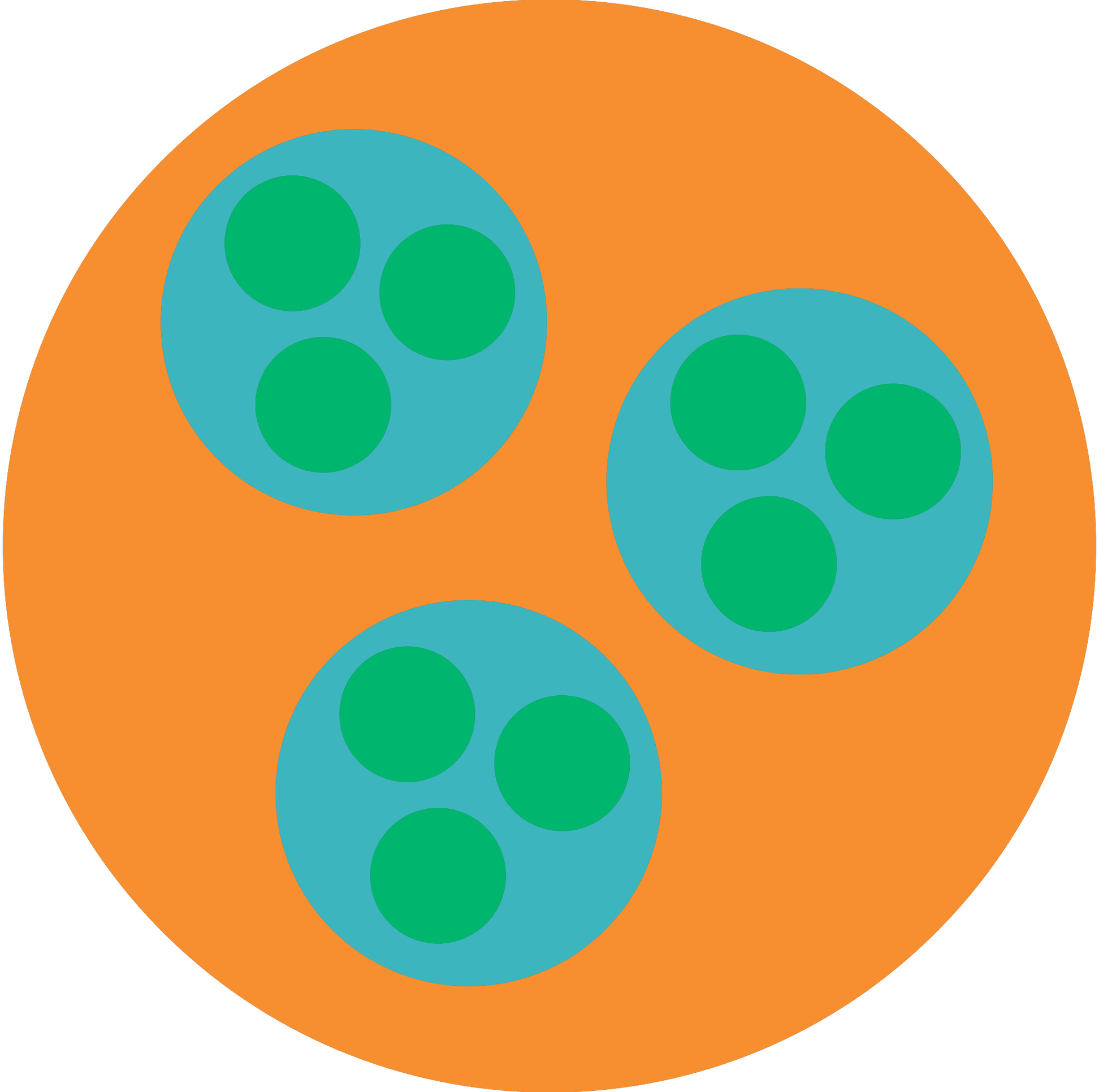
a class should be no bigger than my head

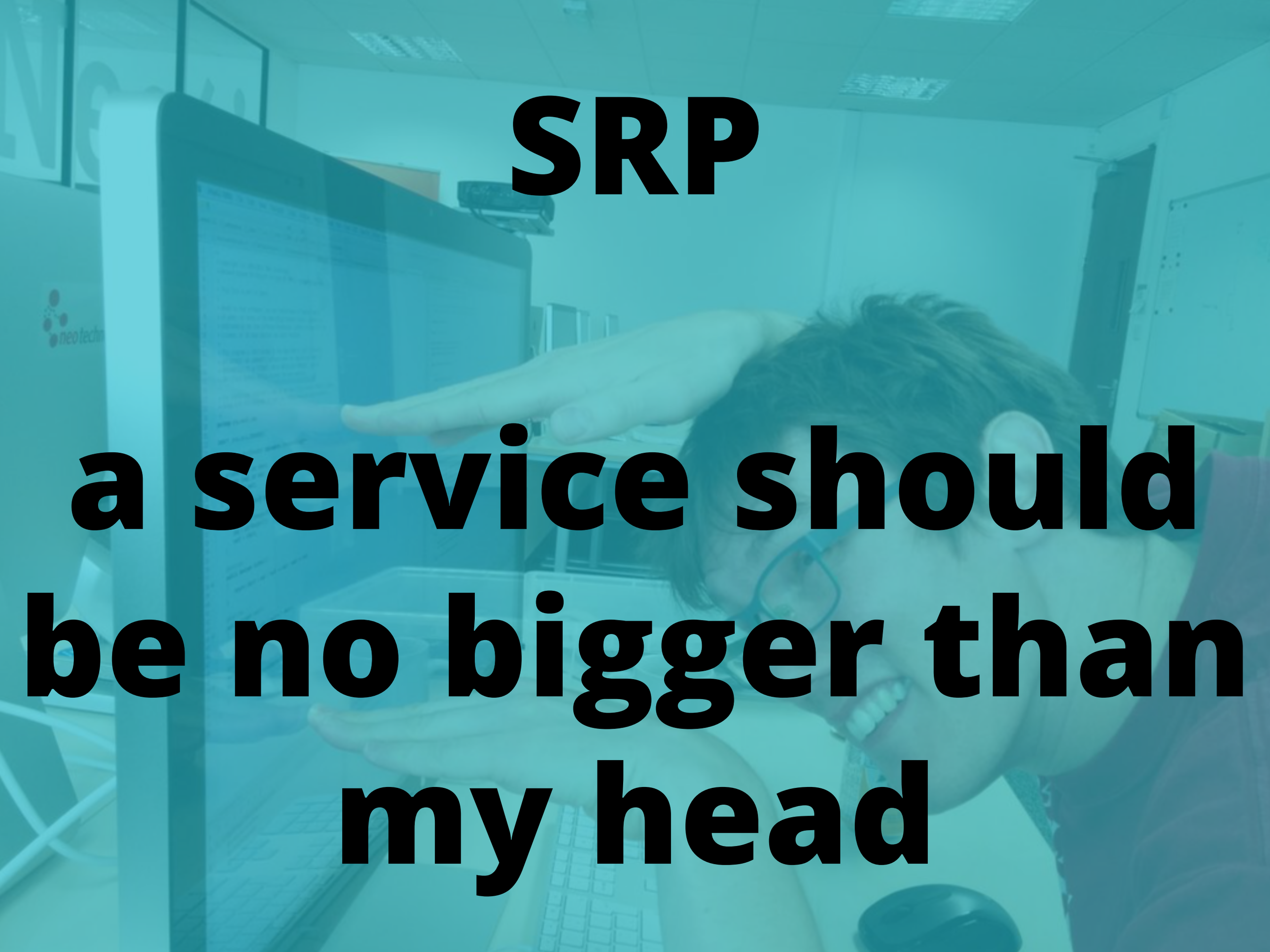



a:Class


a:Class
a:Class
a:Class
a:Class







SRP

a service should be no bigger than my head

GRASP

YAGNI

World of Warcraft

SOLID

agile

DRY

BDD

KISS

Emergent Design

GoF

Continuous Delivery

TDD

XP

Refactoring

HOLD

45.Application Servers new

The rise of containers, phoenix servers and continuous delivery has seen a move away from the usual approach to deploying web applications. Traditionally we have built an artifact and then installed that artifact into an application server. The result was long feedback loops for changes, increased build times and the not insignificant overhead of managing these application servers in production.

WWJD?

WWJD?

(what would Joe do?)

webbit by joewalnes

An event-based WebSocket and HTTP server in Java

Download

You can download this project in either zip or tar formats.

You can also clone the project with Git by running:

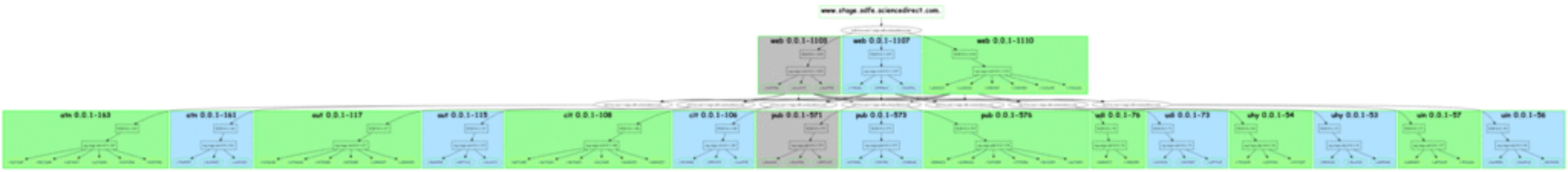
```
$ git clone git://github.com/webbit/webbit
```

Contact

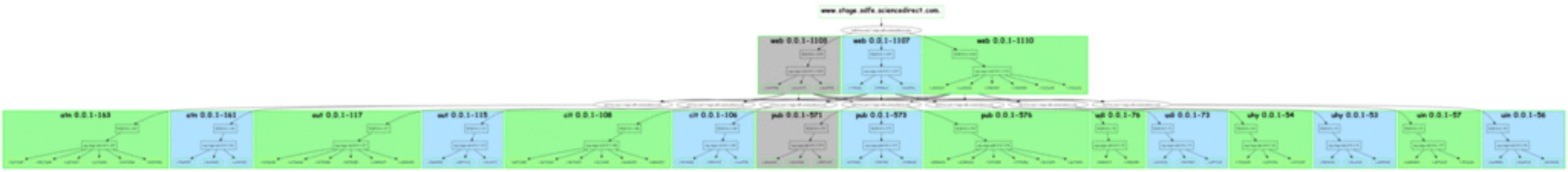
- Webbit Google Group
- @webbitserver on Twitter
- Webbit Wiki

Get the source code from GitHub: webbit/webbit

cron, python, boto, pydot, graphviz



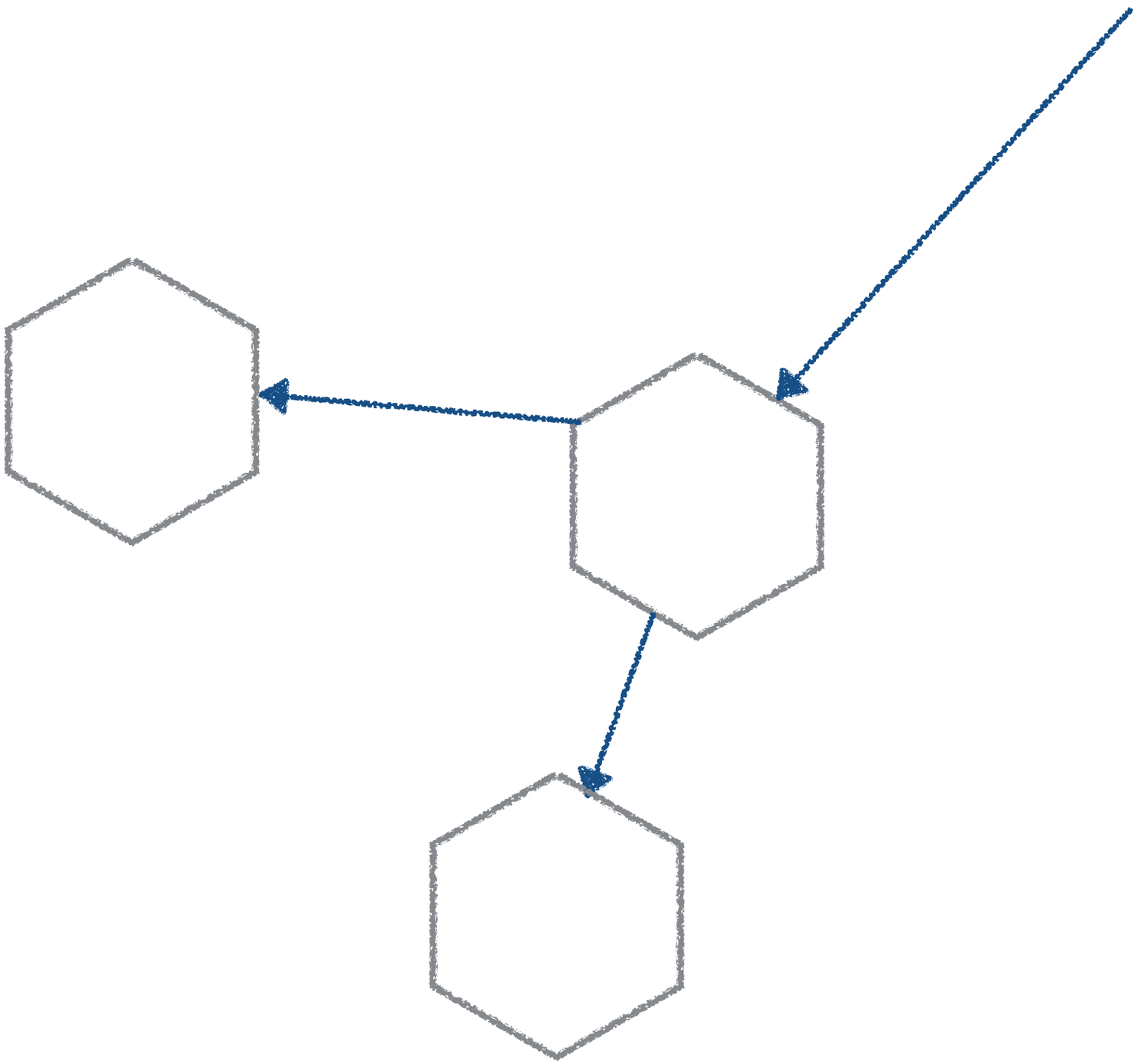
cron, python, boto, pydot, graphviz

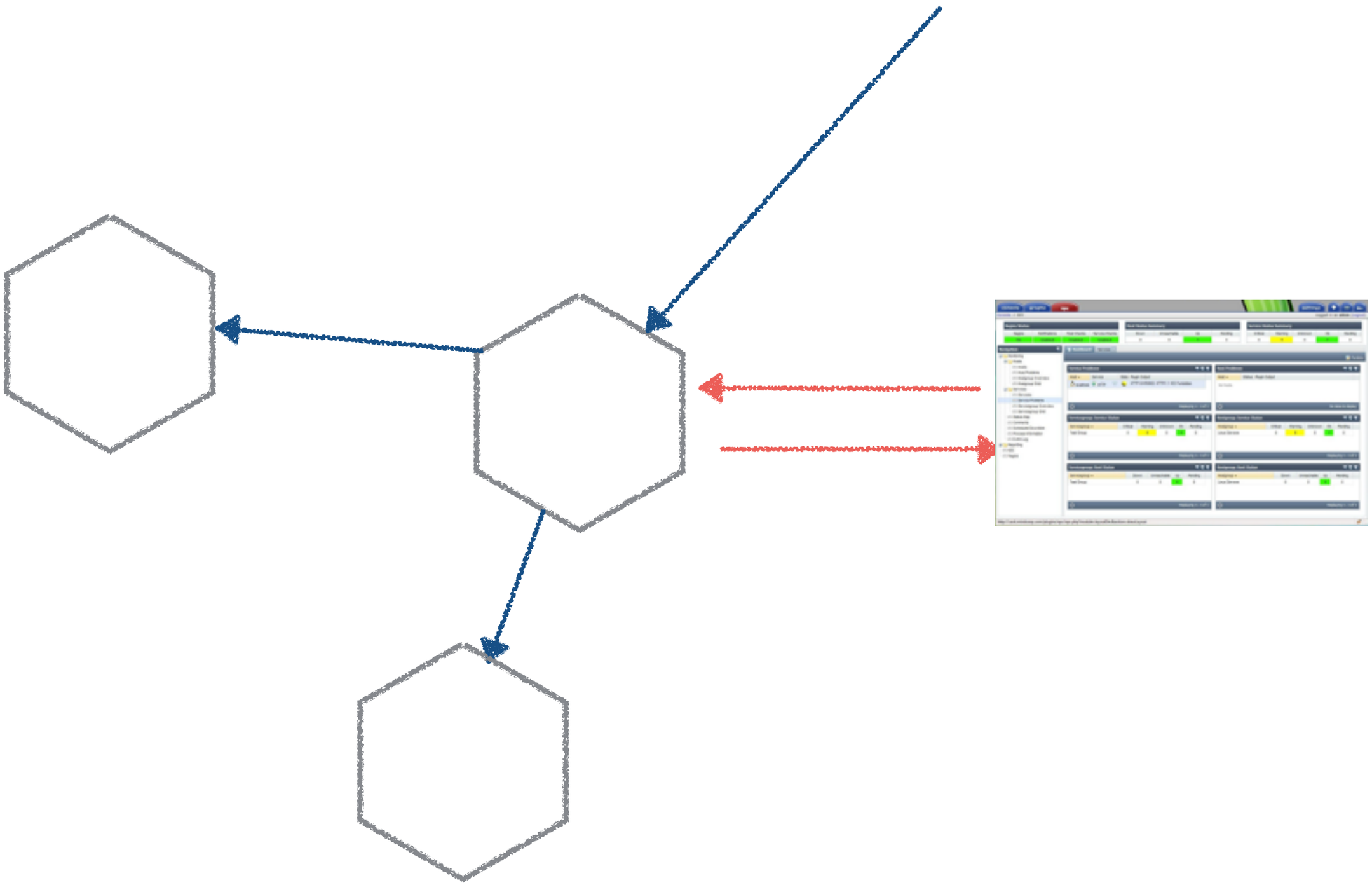


cron, python, boto, pydot, graphviz

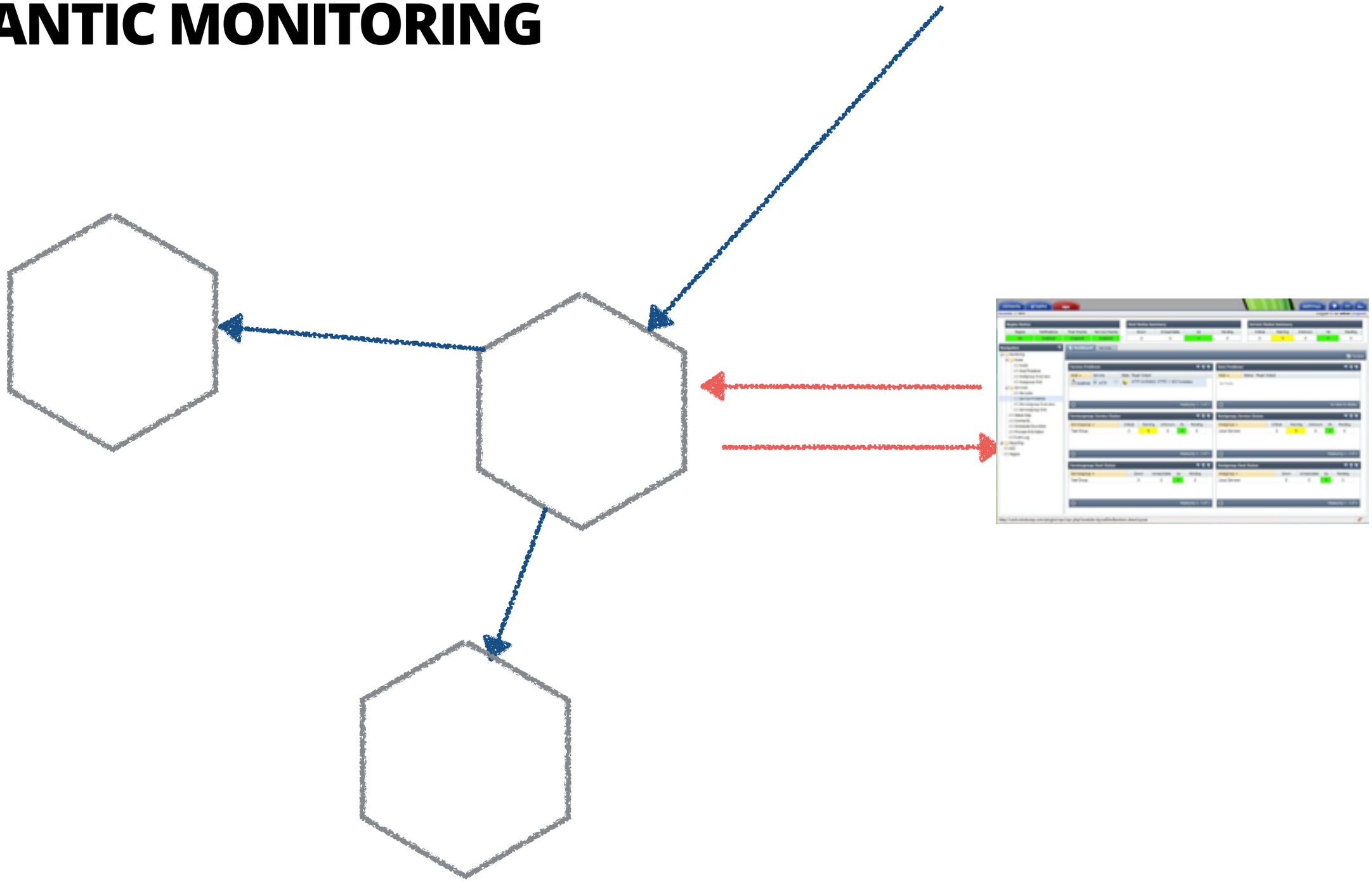
Do the simplest thing possible

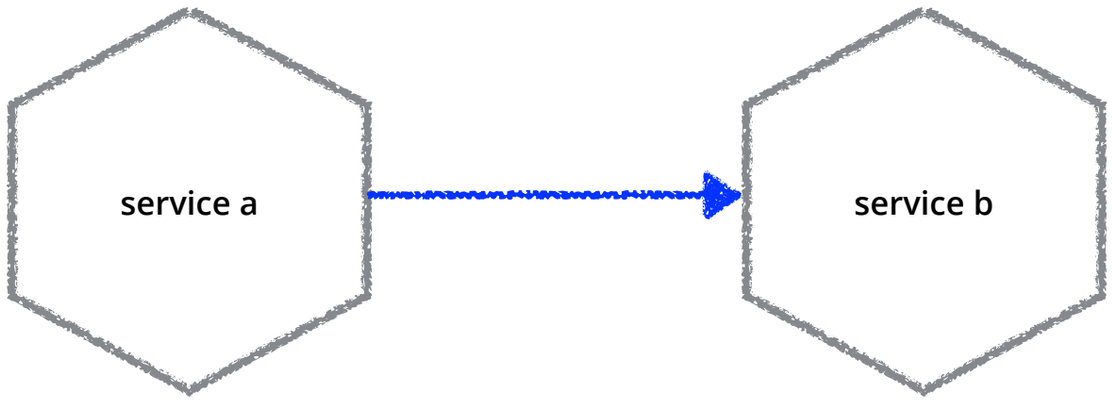
integration and deployment

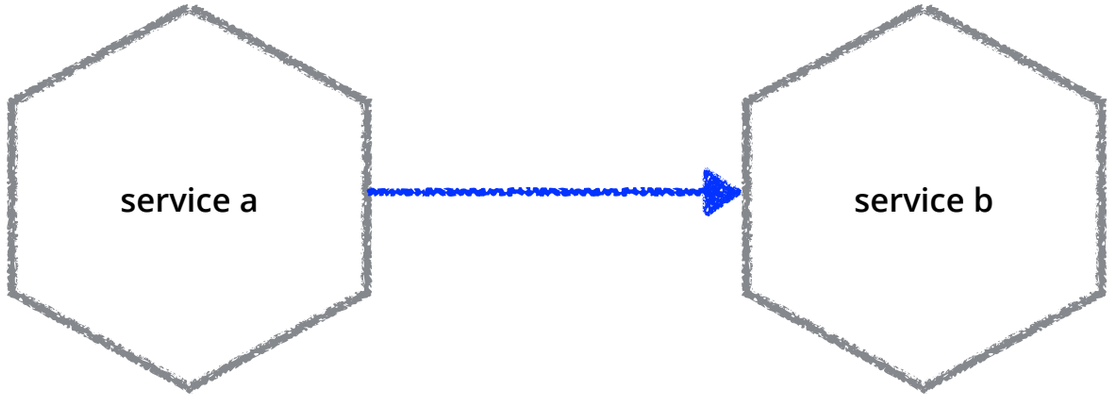


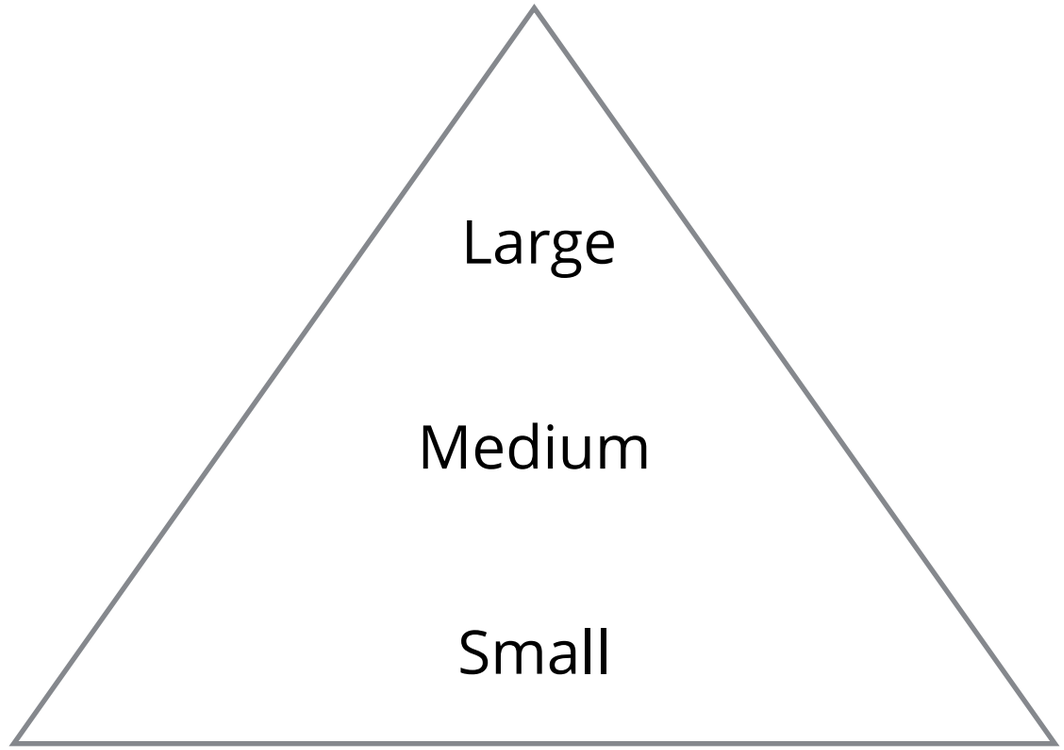


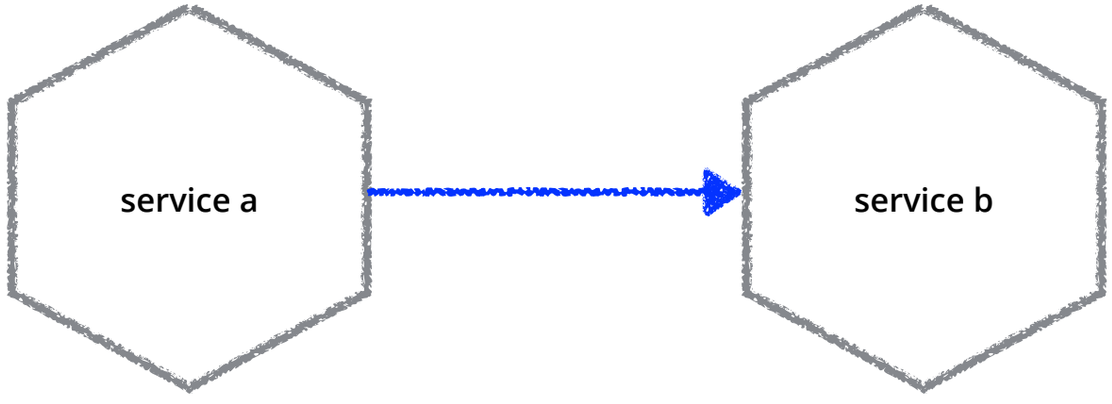
SEMANTIC MONITORING

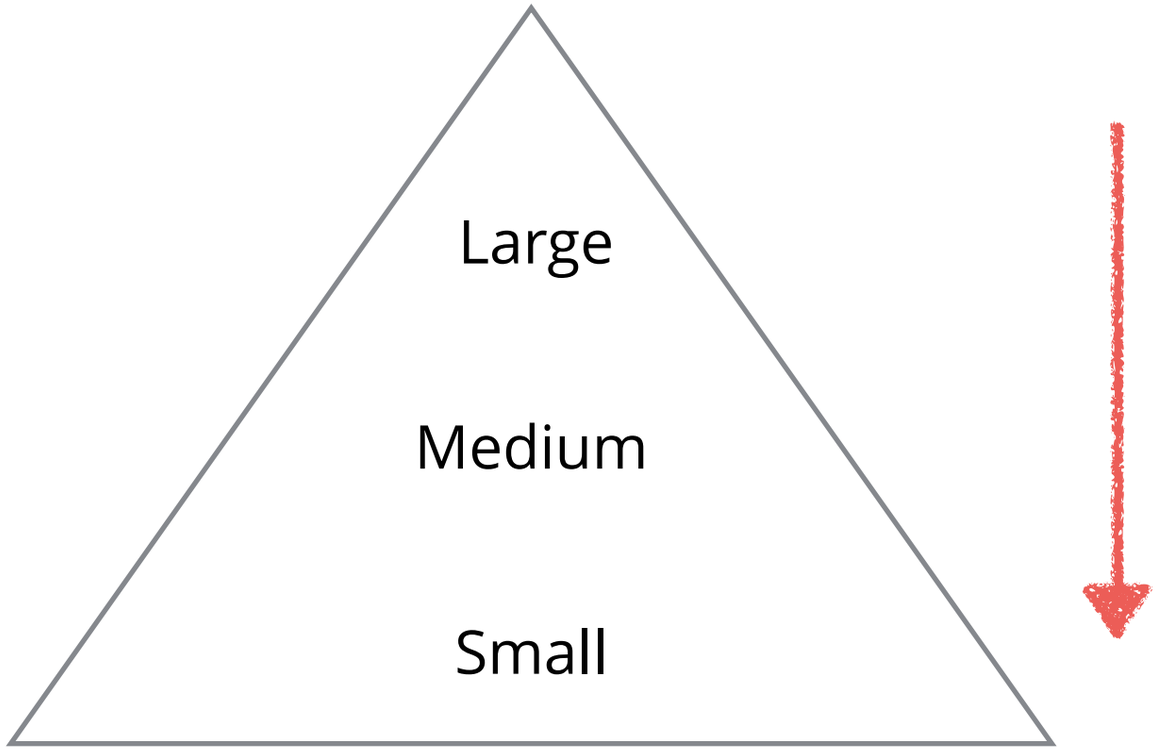


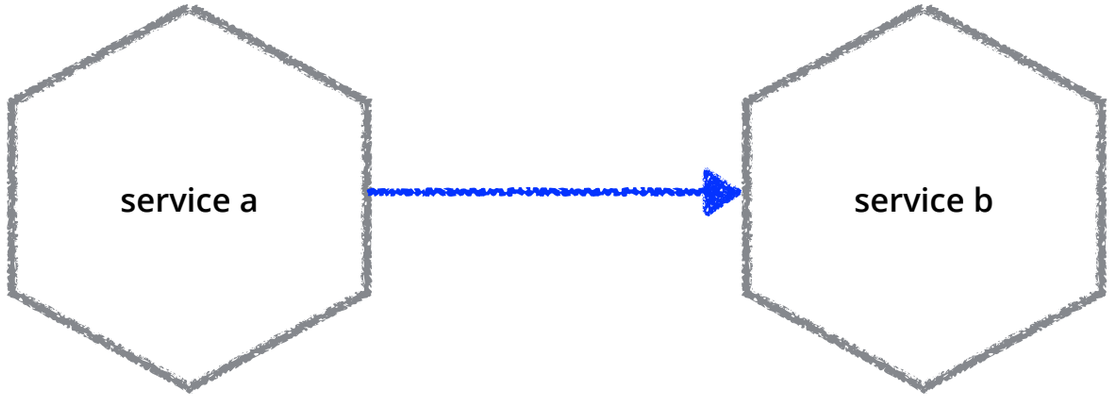

service a
service b


service a
service b

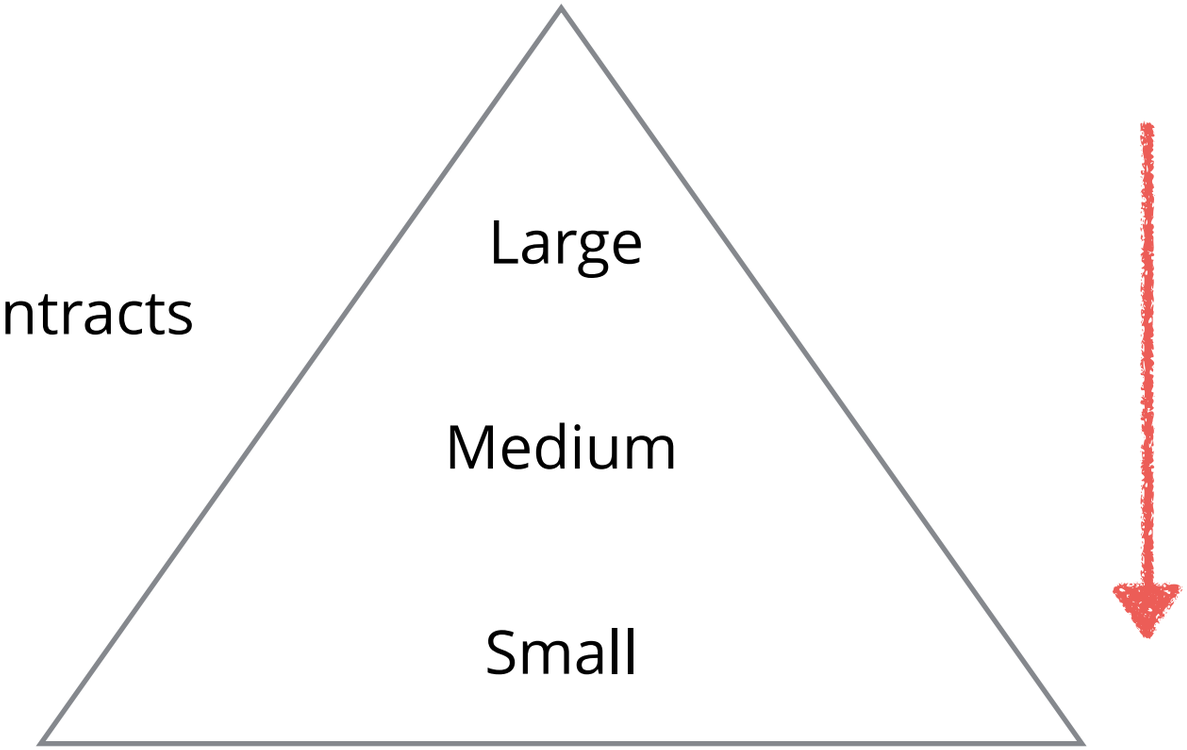

Large
Medium
Small

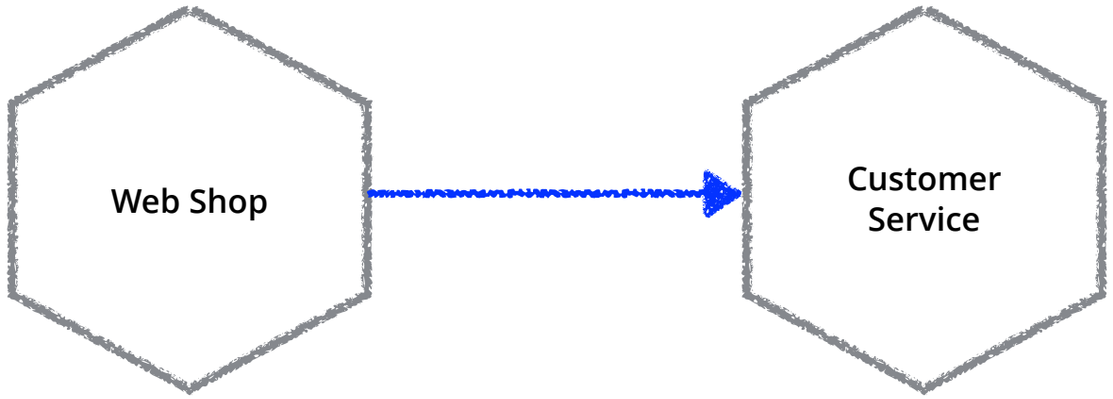

service a
service b

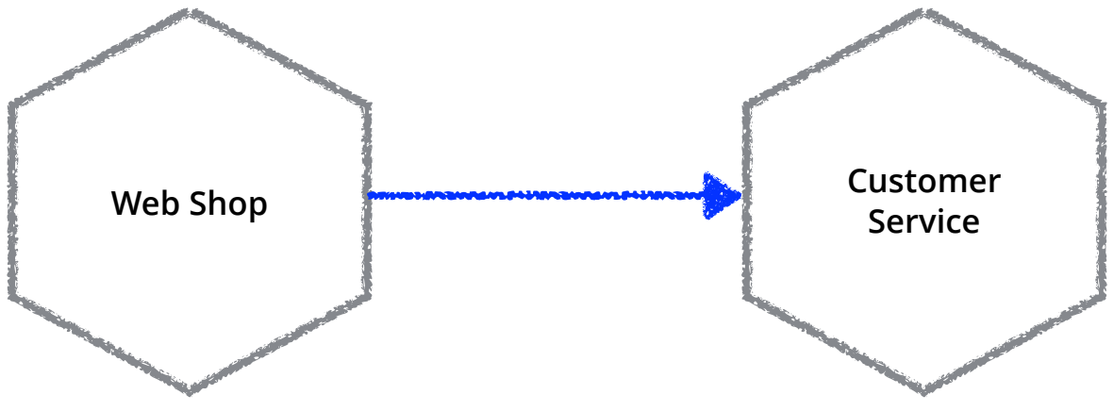

Large
Medium
Small



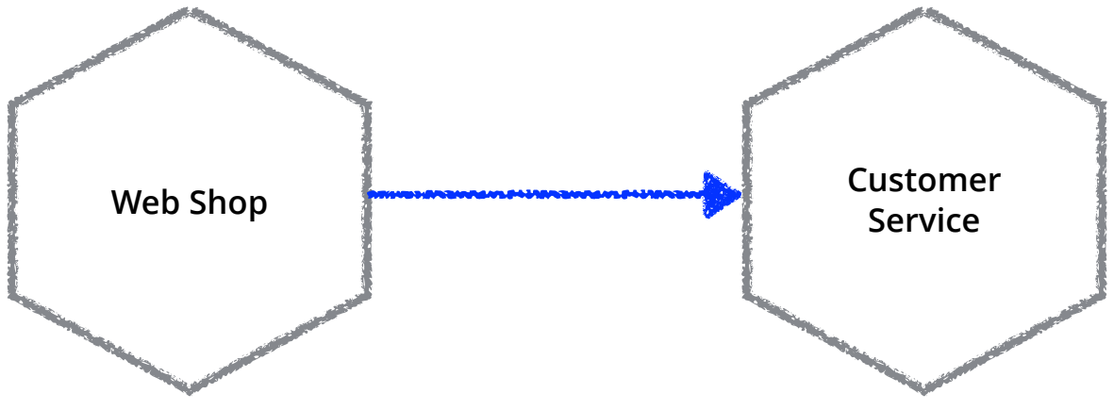
Consumer Driven Contracts




Web Shop
Customer Service

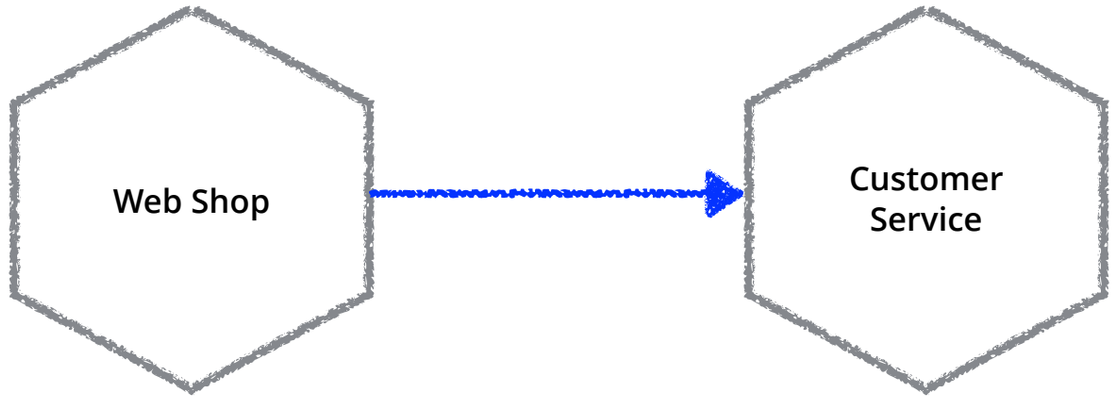


Expectations

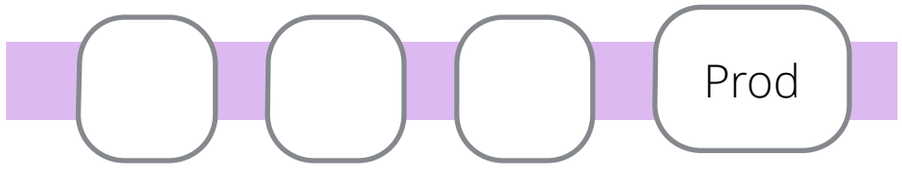


Expectations

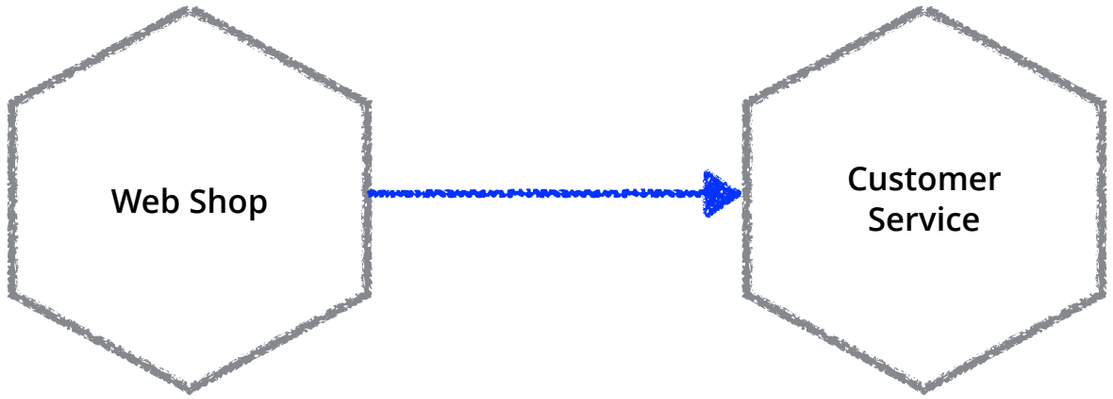
```
def asHtml(): Node = {
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <link rel="stylesheet" href="/static/common.css" type="text/css" />
        <link rel="stylesheet" href={ cssForDisplayType } type="text/css" />
        <meta http-equiv="refresh" content="30" />
    </head>
    <body>
      { content(builds) }
    </body>
  </html>
}

private def content(builds: List[Build]): Elem = {
  displayType match {
    case "single" => <div> { builds.map(build => asTable(build)) } </div>
    case "short" => {
      if (builds.length == 1) {
        <div> { builds.map(build => asTable(build)) } </div>
      } else {
        <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
      }
    }
    case _ => <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
  }
}

private def asTable(build: Build): Elem = {
  <table class={ "build " + build.getStatus.name.toLowerCase }>
            <tr valign="middle" align="center">
              <td>{ linkToBuild(build) }</td>
            </tr>
          </table>
}
```



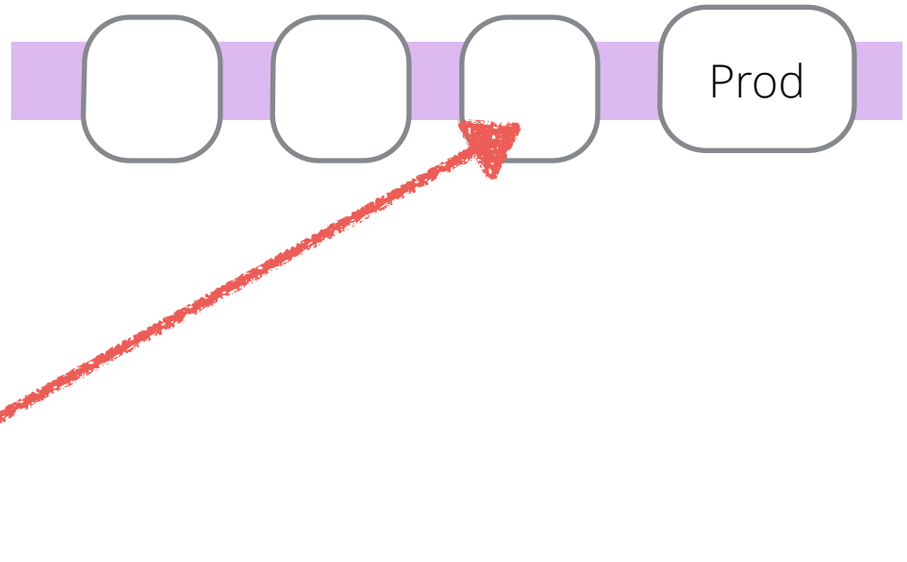
Expectations

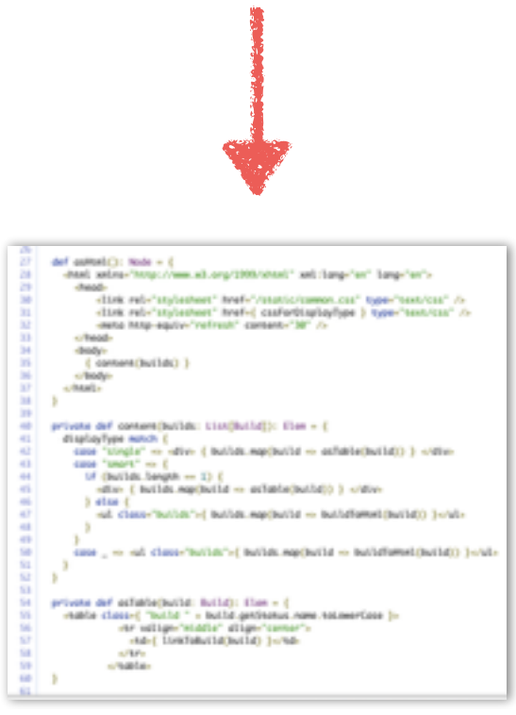


```
def xhtml(): Node = {
  <xml:group>
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <link rel="stylesheet" href="/static/common.css" type="text/css" />
        <link rel="stylesheet" href={ cssForDisplayType } type="text/css" />
        <meta http-equiv="refresh" content="30" />
    </head>
    <body>
      { content(builds) }
    </body>
  </html>
  </xml:group>
}

private def content(builds: List[Build]): Elem = {
  displayType match {
    case "single" => <div> { builds.map(build => asTable(build)) } </div>
    case "smart" => {
      if (builds.length == 1) {
        <div> { builds.map(build => asTable(build)) } </div>
      } else {
        <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
      }
    }
    case _ => <ul class="builds">{ builds.map(build => buildToHtml(build)) }</ul>
  }
}

private def asTable(build: Build): Elem = {
  <table class={ "build " + build.getStatus.name.toLowerCase }>
          <tr valign="middle" align="center">
            <td>{ linkToBuild(build) }</td>
          </tr>
        </table>
}
```



Expectations





```
def xhtml(): Node = {
  <xml:group>
    <head>
        <link rel="stylesheet" href="/static/common.css" type="text/css" />
        <link rel="stylesheet" href={ cssForDisplayType } type="text/css" />
        <meta http-equiv="refresh" content="30" />
    </head>
    <body>
      { content(builds) }
    </body>
  </xml:group>
}

private def content(builds: List[Build]): Elem = {
  displayType match {
    case "single" => <div> { builds.map(build => oneTable(build)) } </div>
    case "smart" => {
      if (builds.length == 1) {
        <div> { builds.map(build => oneTable(build)) } </div>
      } else {
        <ul class="build">{ builds.map(build => buildToHtml(build)) }</ul>
      }
    }
    case _ => <ul class="build">{ builds.map(build => buildToHtml(build)) }</ul>
  }
}

private def oneTable(build: Build): Elem = {
  <table class={ "build " + build.getStatus.name.toLowerCase }>
          <tr valign="middle" align="center">
            <td>{ linkToBuild(build) }</td>
          </tr>
        </table>
}
```

README.md

Pact

Define a pact between service consumers and providers, enabling "consumer driven contract" testing.

Pact provides an RSpec DSL for service consumers to define the HTTP requests they will make to a service provider and the HTTP responses they expect back. These expectations are used in the consumers specs to provide a mock service provider. The interactions are recorded, and played back in the service provider specs to ensure the service provider actually does provide the response the consumer expects.

This allows testing of both sides of an integration point using fast unit tests.

This gem is inspired by the concept of "Consumer driven contracts". See http://martinfowler.com/articles/consumerDrivenContracts.html for more information.

Travis CI Status: 

README.md

Pact

Define a pact between service consumers and providers, enabling "consumer driven contract" testing.

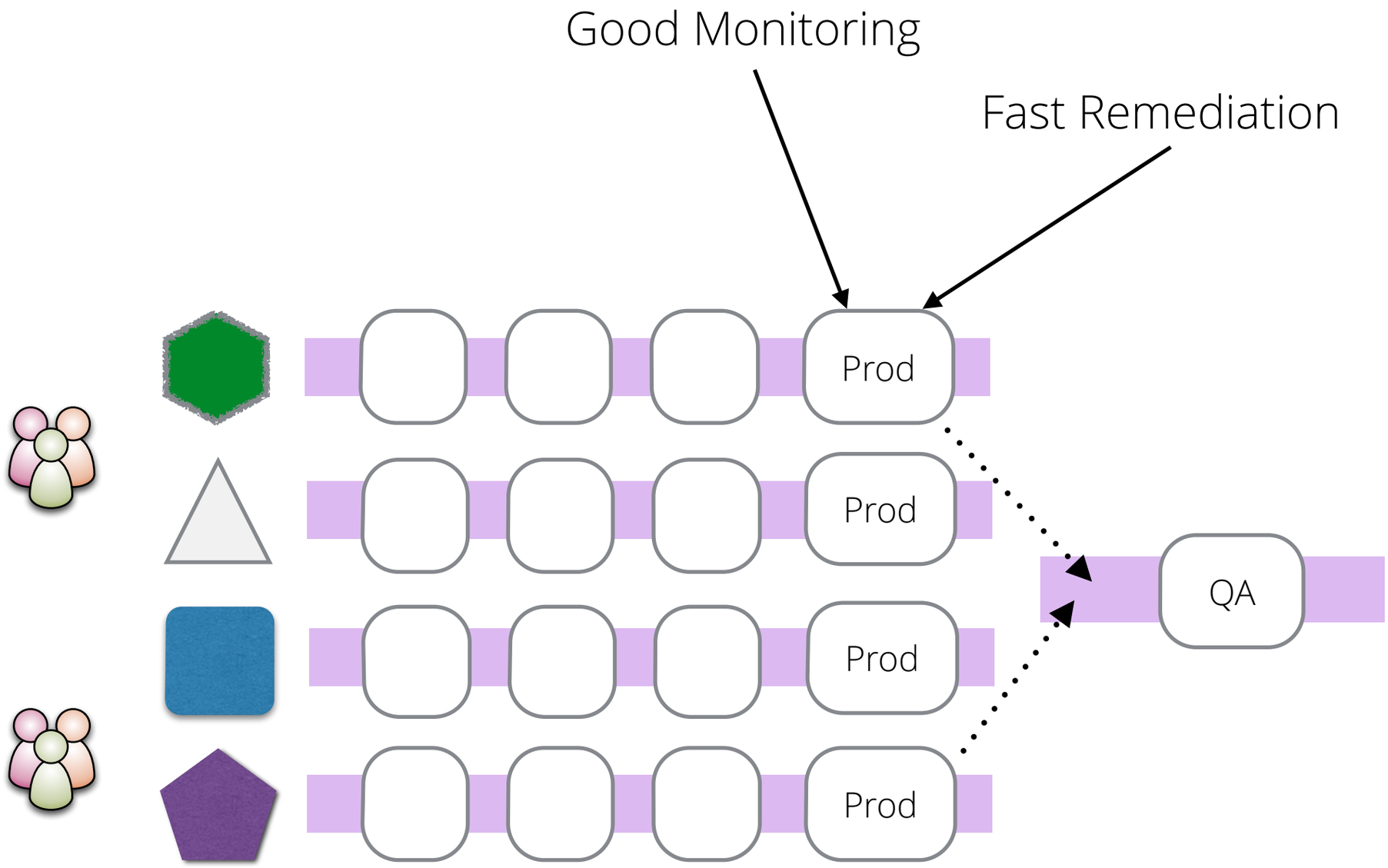
Pact provides an RSpec DSL for service consumers to define the HTTP requests they will make to a service provider and the HTTP responses they expect back. These expectations are used in the consumers specs to

to ens https://github.com/realestate-com-au/pact

This allows testing of both sides of an integration point using fast unit tests.

This gem is inspired by the concept of "Consumer driven contracts". See http://martinfowler.com/articles/consumerDrivenContracts.html for more information.

Travis CI Status: build passing



TESTING IN PRODUCTION

the death of the integration environment

production != live

CHANGING SERVICES INDEPENDENTLY

CHANGING SERVICES INDEPENDENTLY

What is the blast radius of the change?

CHANGING SERVICES INDEPENDENTLY

What is the blast radius of the change?

Limited to your team?

CHANGING SERVICES INDEPENDENTLY

What is the blast radius of the change?

Limited to your team?

business capability?

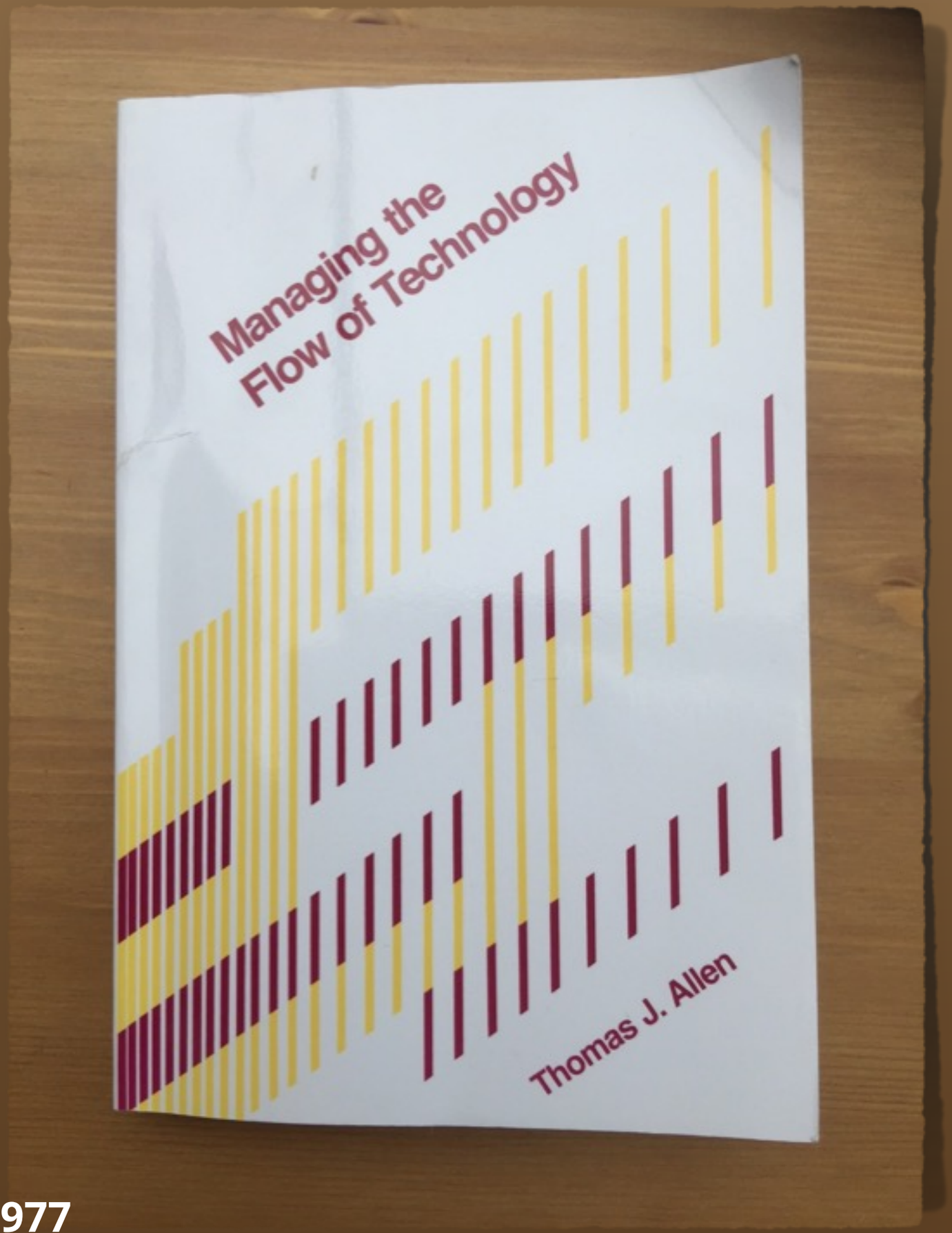
CHANGING SERVICES INDEPENDENTLY

What is the blast radius of the change?

Limited to your team?

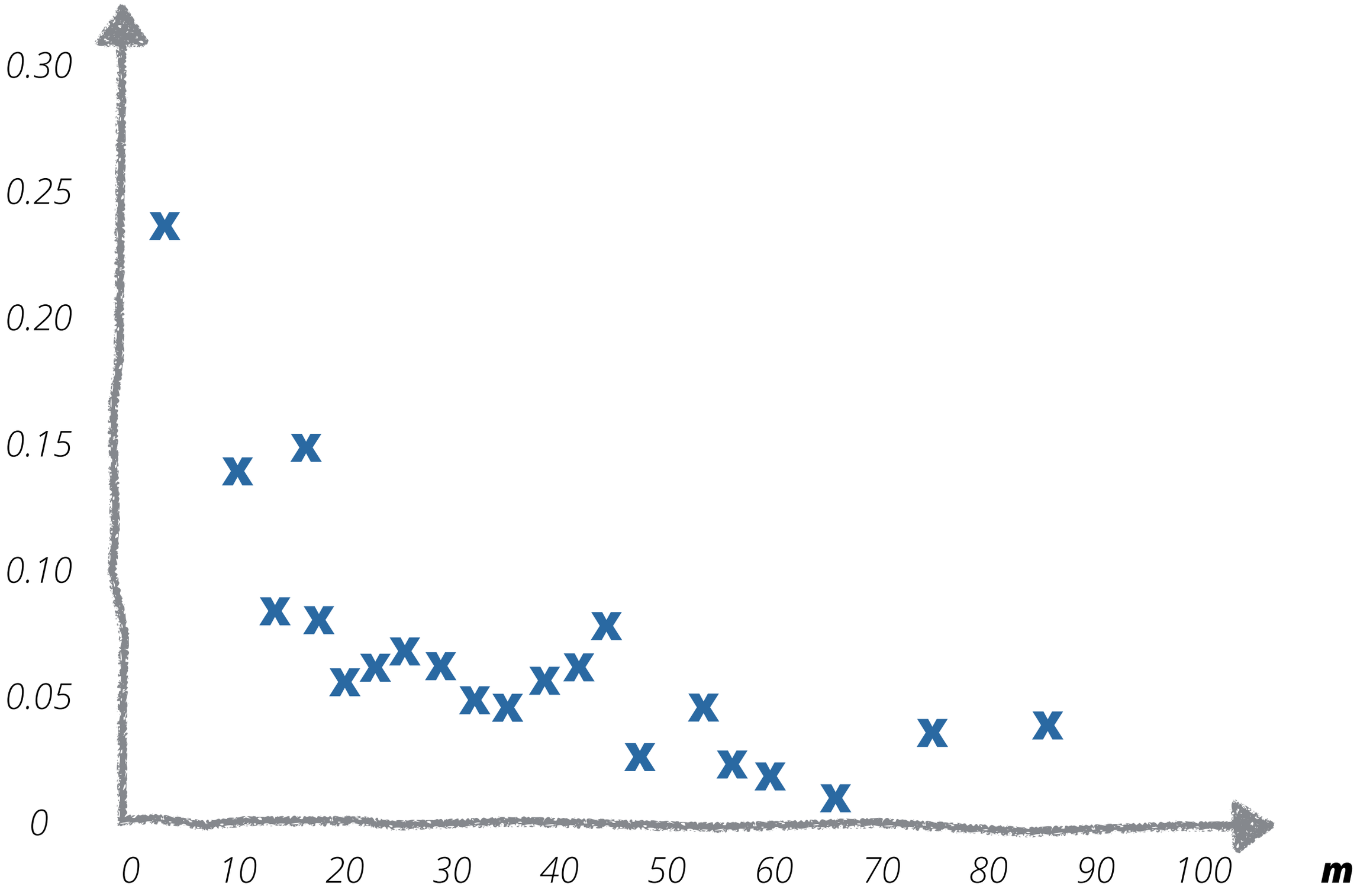
business capability?

organisation?



Thomas J. Allen, 1977

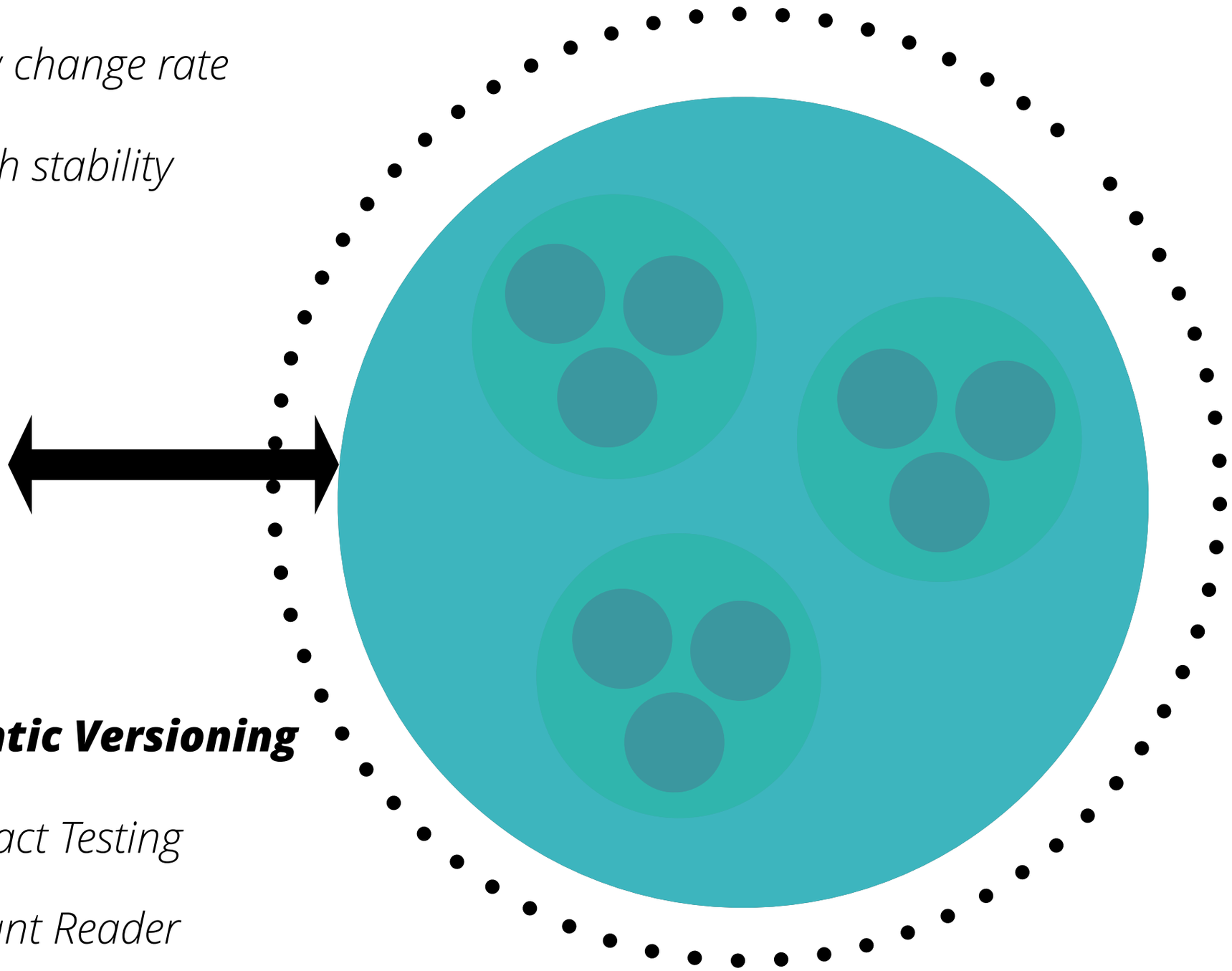
The effect of distance on communication



inter-company integration

Low change rate

High stability



Semantic Versioning

Contract Testing

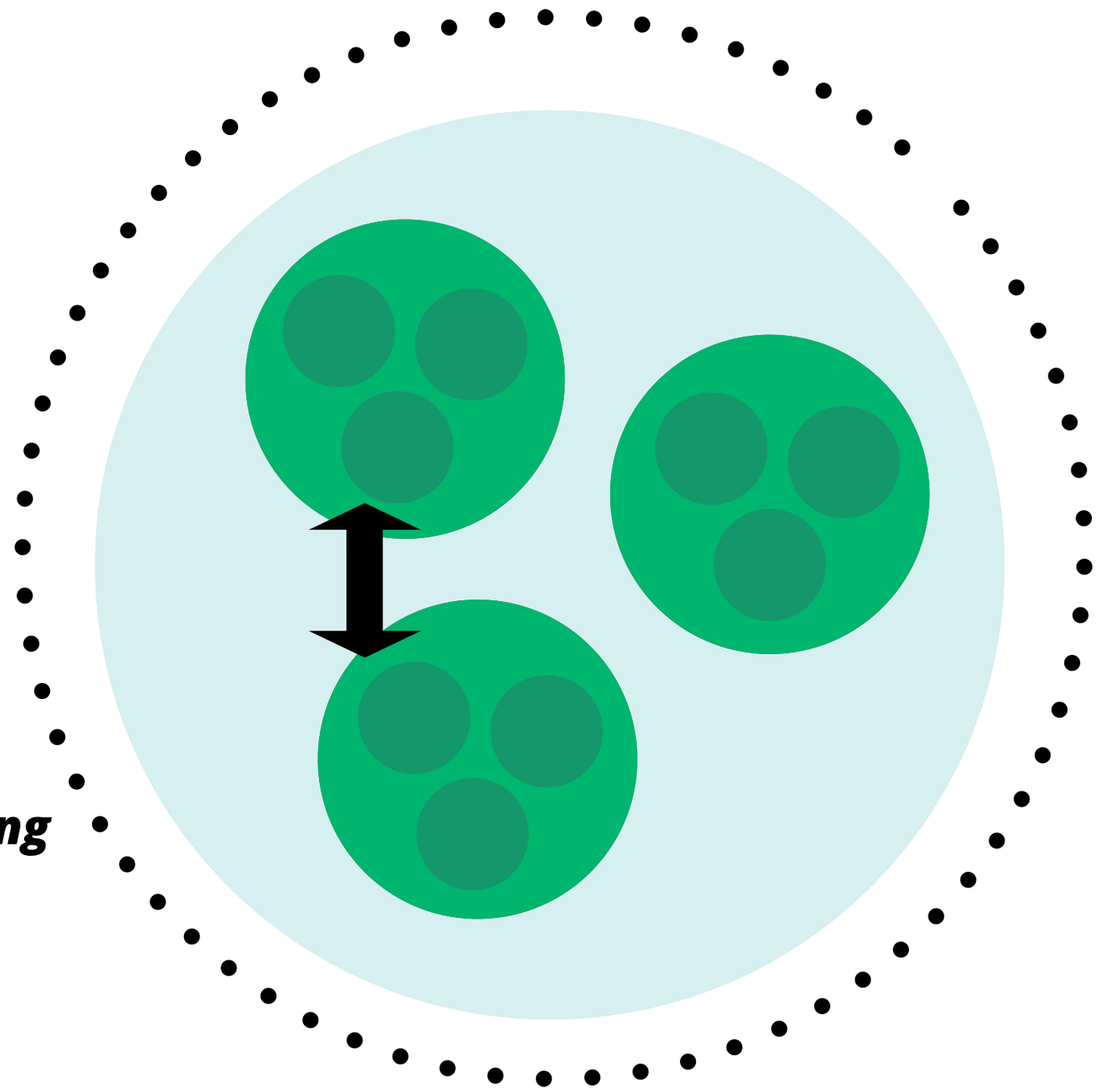
Tolerant Reader

“Conversational change”

inter-team integration

higher change rate

lower stability



Semantic Versioning

Contract Testing

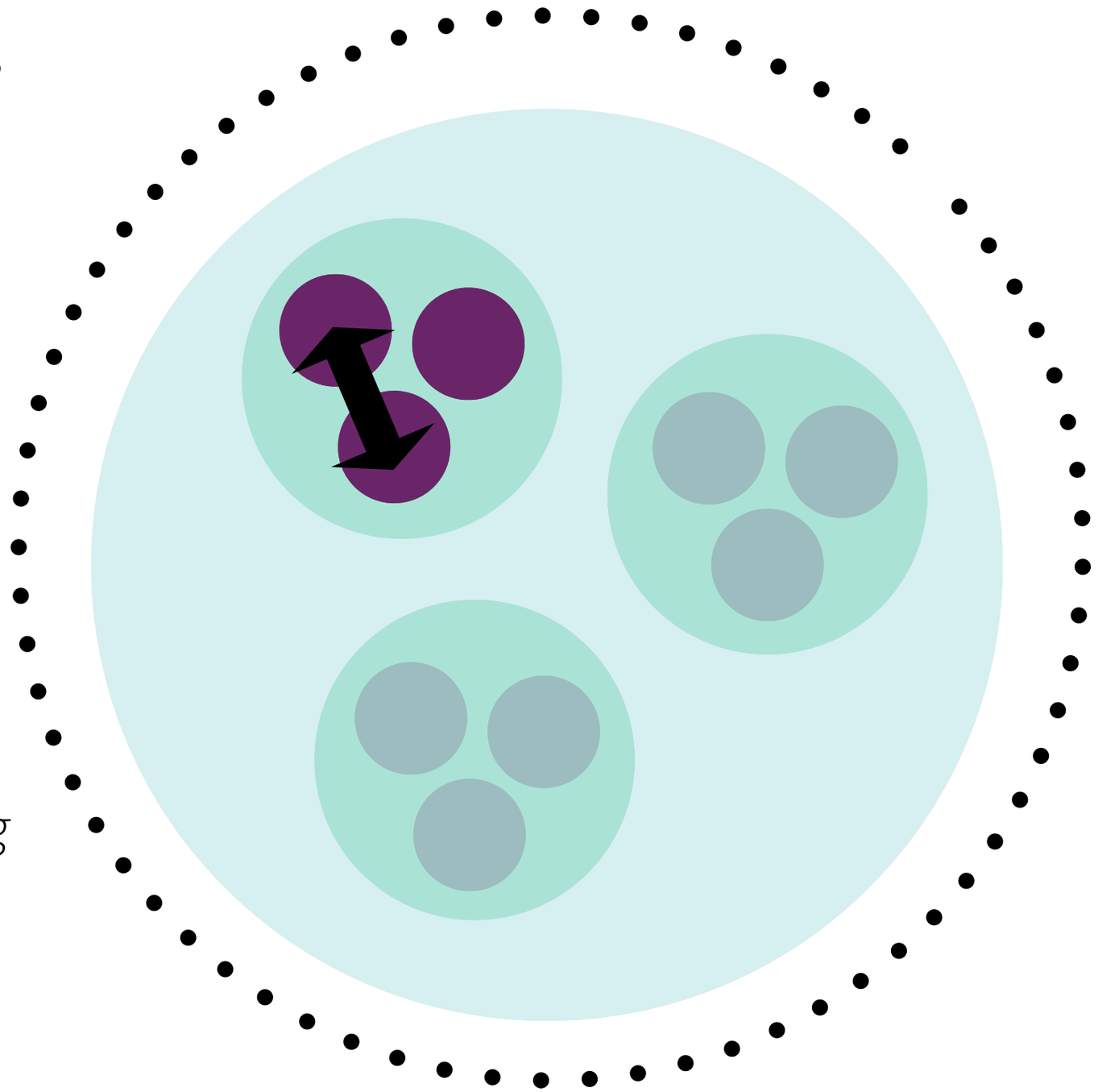
Tolerant Reader

"Conversational change"

intra-team

highest change rate

lowest stability



Semantic Versioning

Contract Testing

Tolerant Reader

“Conversational change”

the future is scary

we are learning how to:

Craft my families axe

Deploy small services independently

Test microservices in isolation
in production

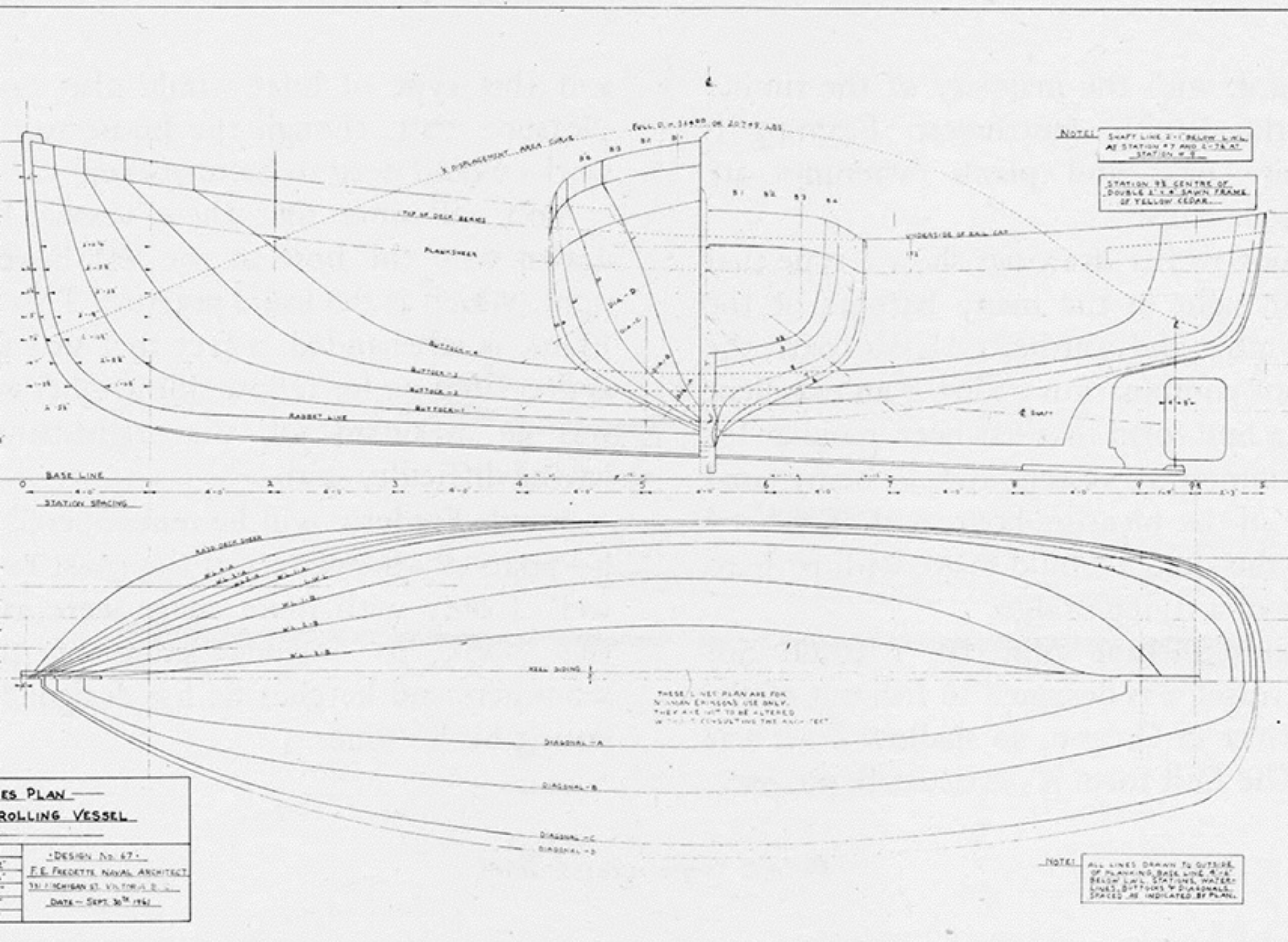

the future is bright
Sun-Earth Day 2008: Space Weather Around the World
sunearthday.nasa.gov

we have old techniques that apply:

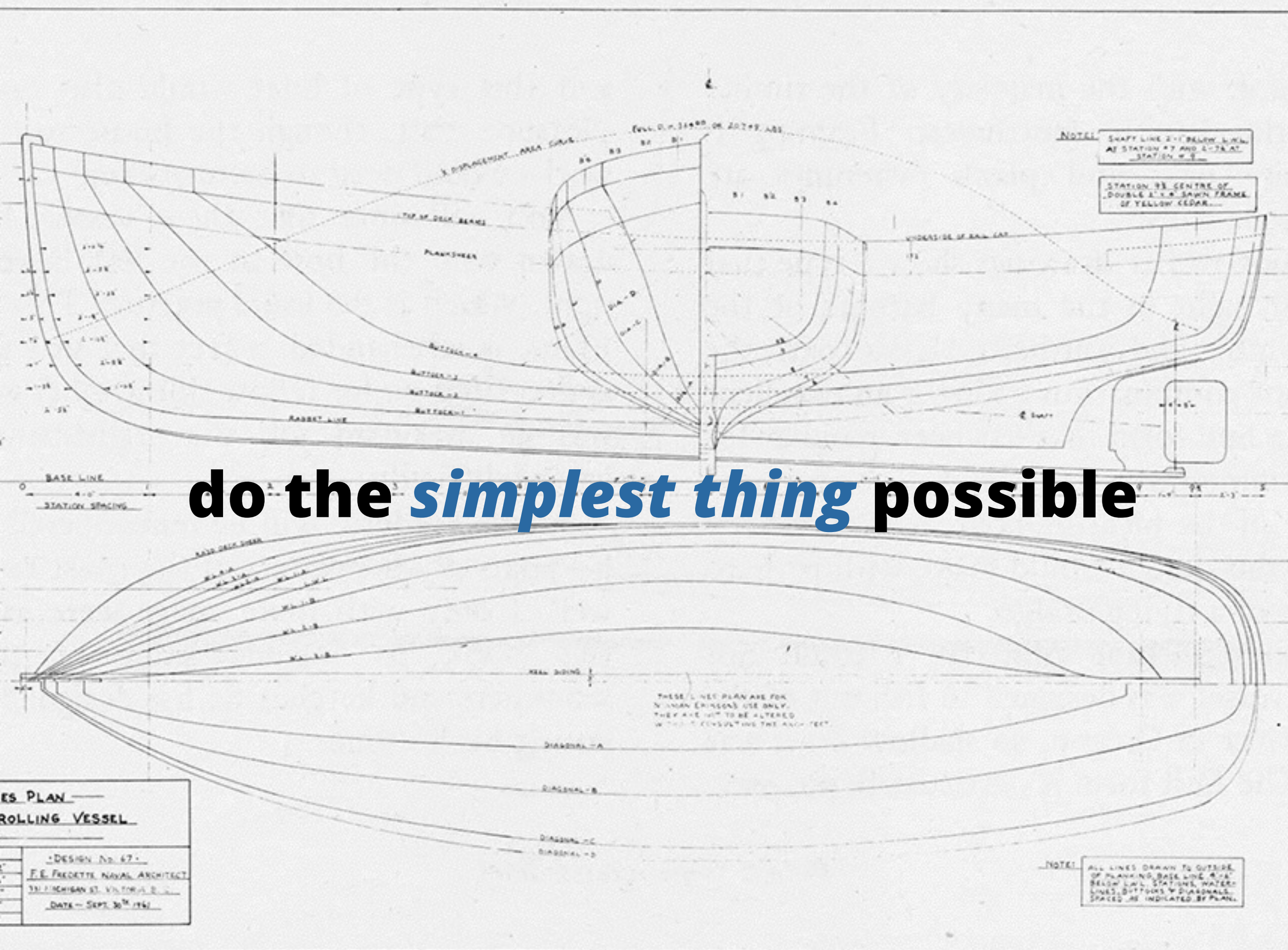
SRP
GRASP
YAGNI
KISS
TDD
DRY

and new techniques to apply:

Consumer Driven Contracts
Semantic Monitoring
Semantic Versioning
Testing in Production
Failure Isolation


ES PLAN
ROLLING VESSEL
·DESIGN No. 67·
F. E. FREDETTE NAVAL ARCHITECT
DATE — SEPT. 30TH 1961
NOTE:
SHAFT LINE 2'-1" BELOW L.W.L. AT STATION #7 AND 2'-7½" AT STATION #9
STATION 9½ CENTRE OF DOUBLE 2" x 4" SAWN FRAME OF YELLOW CEDAR
NOTE:
ALL LINES DRAWN TO OUTSIDE OF PLANKING. BASE LINE 4'-6" BELOW L.W.L. STATIONS, WATER-LINES, BUTTOCKS & DIAGONALS SPACED AS INDICATED BY PLAN.
TOP OF DECK BEAMS
PLANKSHEER
UNDERSIDE OF RAIL CAP
BUTTOCK - A
BUTTOCK - 3
BUTTOCK - 2
BUTTOCK - 1
RABBET LINE
BASE LINE
STATION SPACING
DIAGONAL - A
DIAGONAL - B
DIAGONAL - C
DIAGONAL - D
KEEL SIDING

do the *simplest thing* possible



jalewis@thoughtworks.com

@boicy

ThoughtWorks®

Thanks!

jalewis@thoughtworks.com

@boicy

ThoughtWorks®