

**building systems that are**  
**never done**

**[jalewis@thoughtworks.com](mailto:jalewis@thoughtworks.com)**

**@boicy**

**ThoughtWorks®**

**never done**

# never done

Incomplete

adjective

not having all the necessary or appropriate parts

# never done

Incomplete

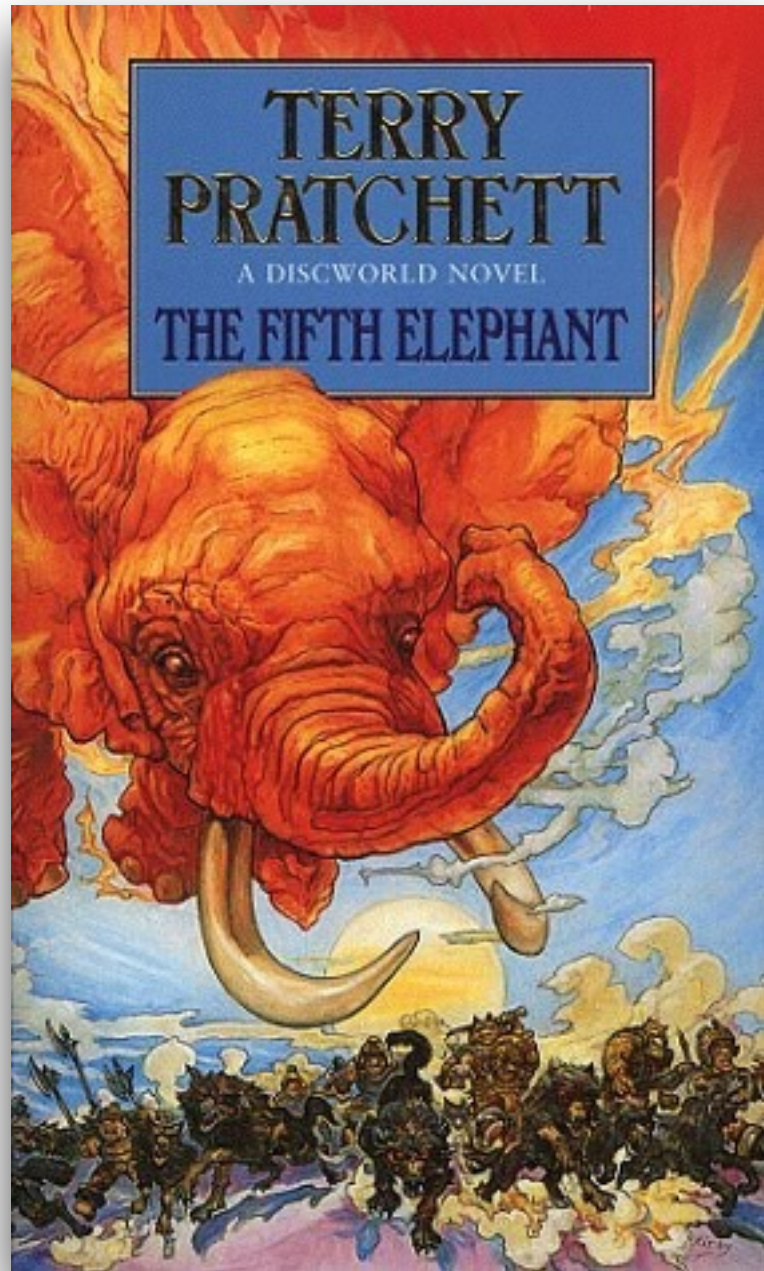
adjective

not having all the appropriate parts

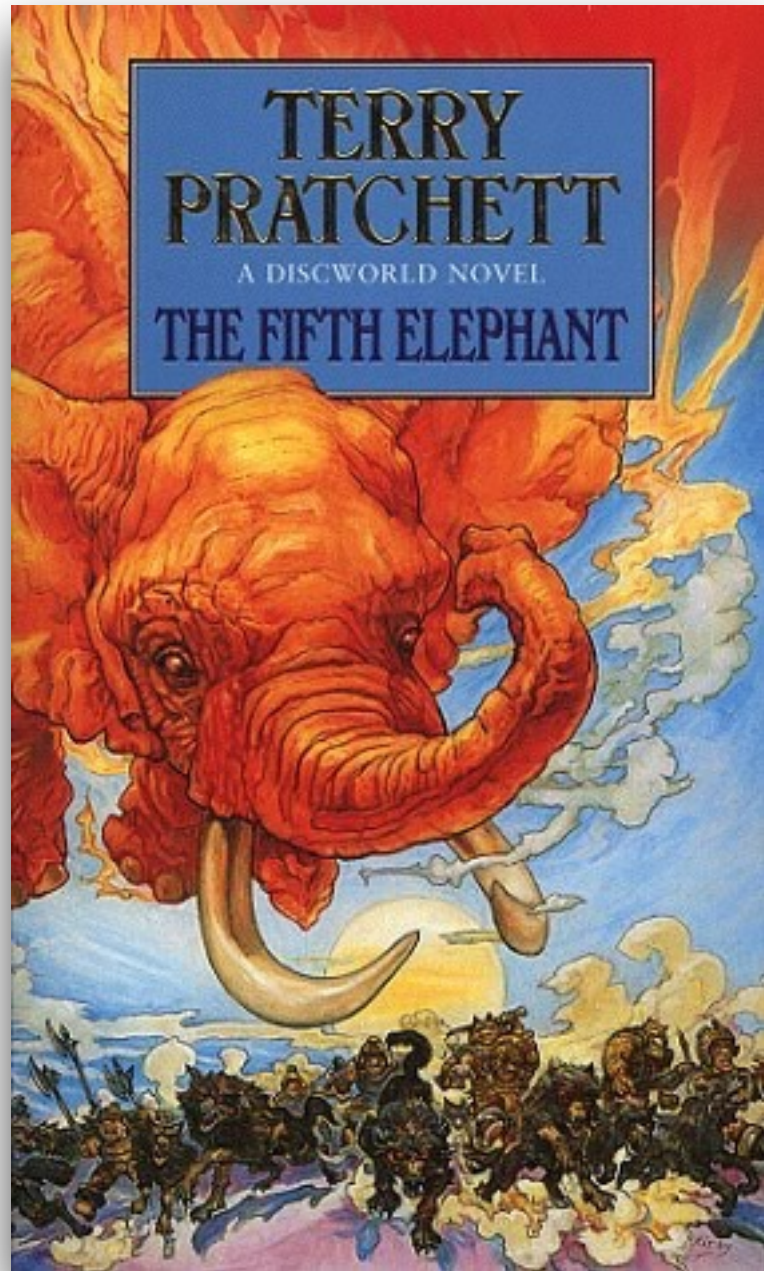




# never done



# 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”*

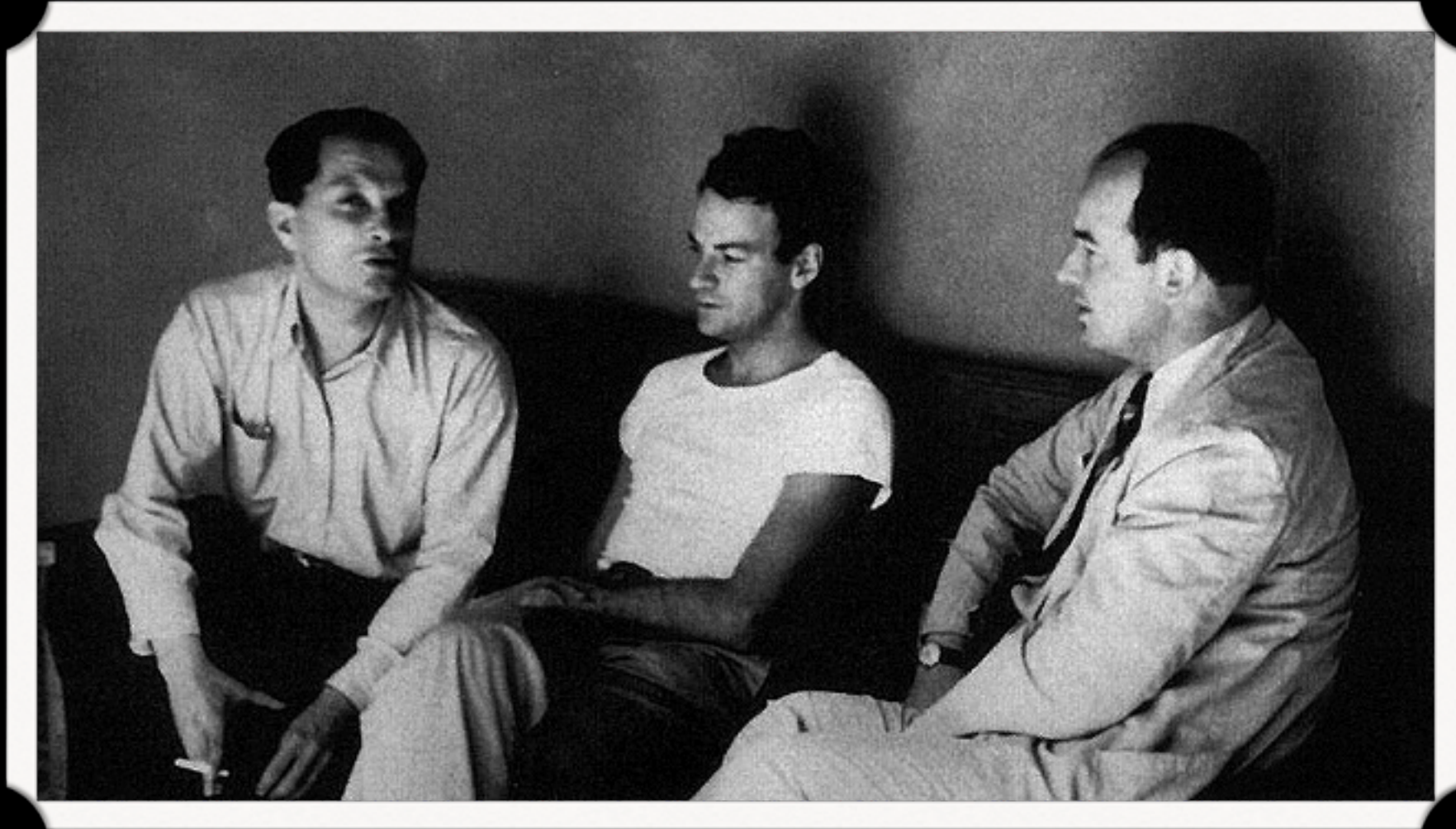
*Neal Ford*

# *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





# Singularity

# JavaScript Singularity

# JavaScript Singularity Container

**Log aggregation**

**JavaScript**

**Singularity**

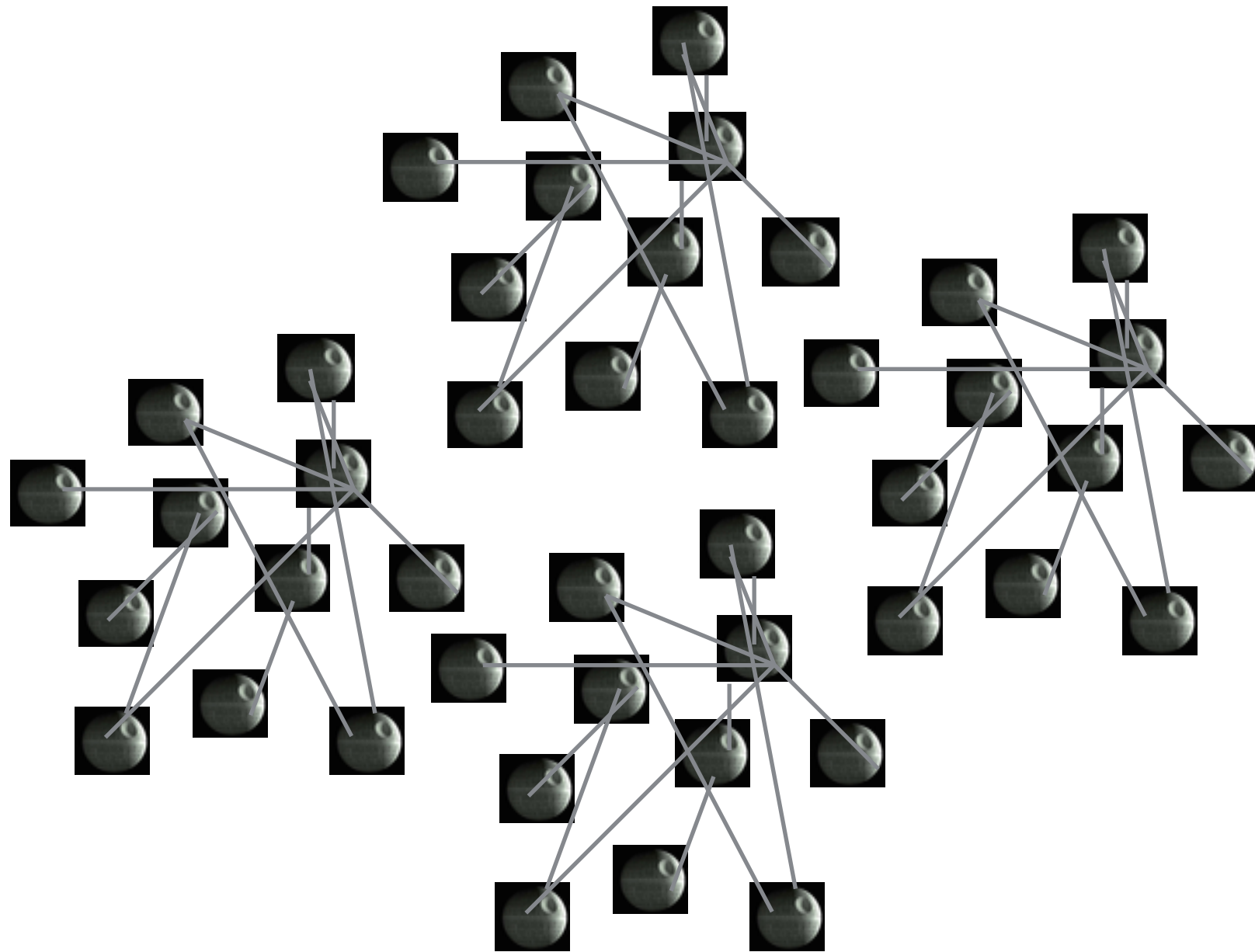
**Container**

**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

### Characteristics of a Microservice Architecture

- Componentization via Services
- Organized around Business Capabilities
- Products not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design

### Are Microservices the Future?

## Sidebars

How big is a microservice?

Microservices and SOA

Many languages, many options

Battle-tested standards and enforced standards

Make it easy to do the right thing

The circuit breaker and production ready code

Synchronous calls considered harmful

*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

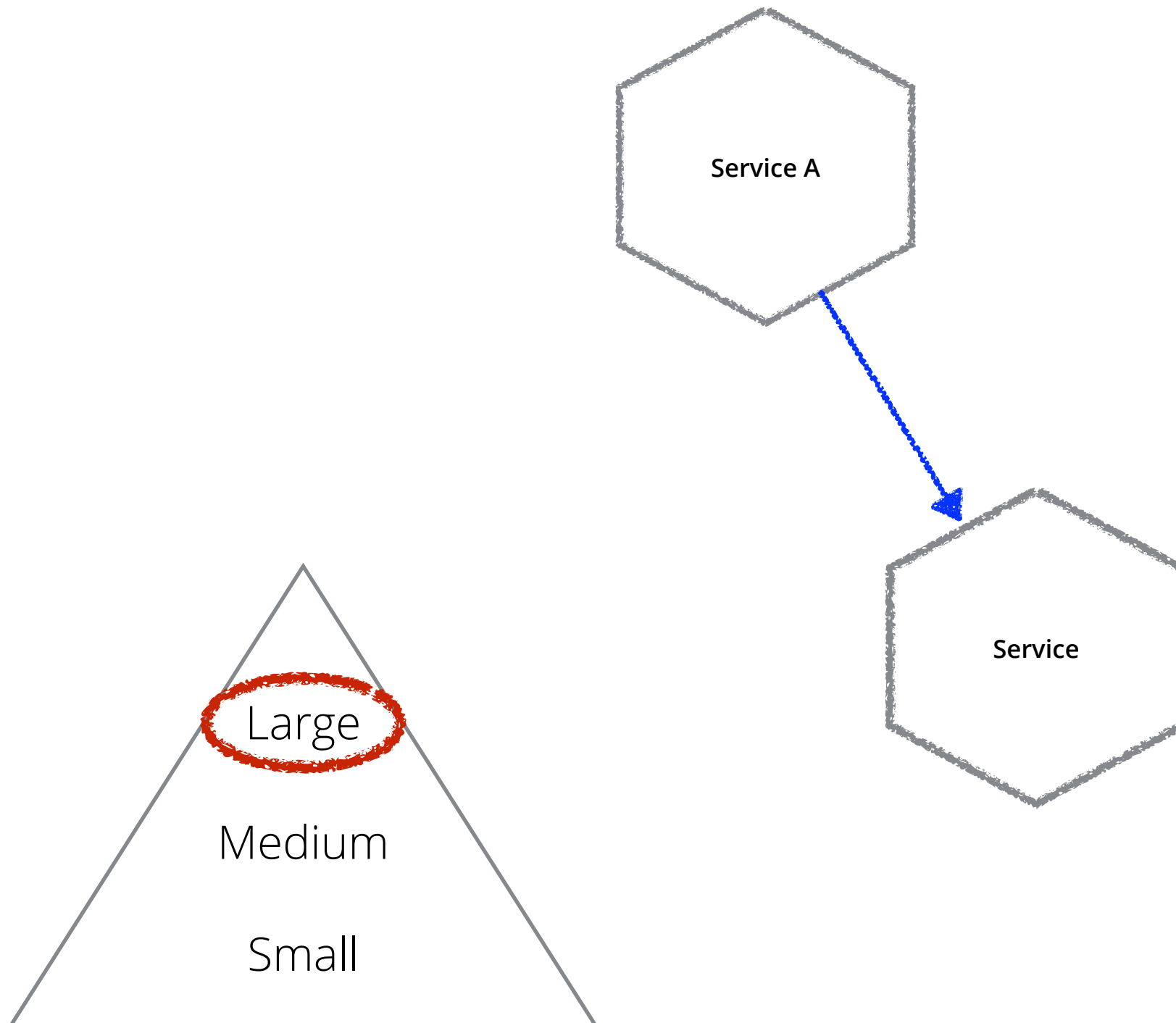
Independent deployment

Service versioning / evolution

---

# TESTING MICROSERVICES IS HARD

---



```

27 def autorizo(): bool = {
28   let url = "http://www.sib.org/WWW/autoriz.asp?lang=es" & lang=es"
29   <body>
30     <div id="styleContent" href="/static/content.css" type="text/css" />
31     <div id="styleForm" href="/static/formstyle.css" type="text/css" />
32     meta http-equiv="refresh" content="30" />
33   </head>
34   <body>
35     { content(htmlid) }
36   </body>
37 </html>
38 }
39
40 private def content(htmlid: String): Elem = {
41   def langType match {
42     case "ingles" => elem { htmlid.map(htmlid => htmlid&langID) } <html>
43     case "espa" => {
44       if (htmlid.length == 1) {
45         elem { htmlid.map(htmlid => htmlid&langID) } <html>
46       } else {
47         elem { htmlid.map(htmlid => htmlid&langID) } <html>
48       }
49     }
50     case _ => elem { htmlid.map(htmlid => htmlid&langID) } <html>
51   }
52 }
53
54 private def htmlid(htmlid: Elem) = {
55   <div class="htmlid" => htmlid.getElem.htmlid.toString() >
56     <div id="htmlid" class="design">content</div>
57     <div { htmlid&langID(htmlid) } <html>
58     </div>
59     </div>
60 }
61

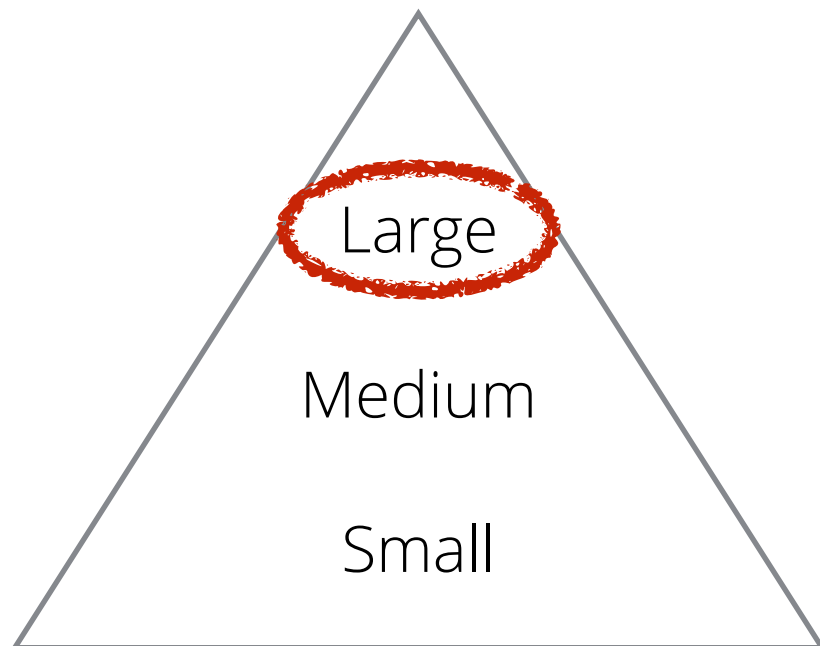
```



```

27 def autorizo(): bool = {
28   let url := http://www.sib.org/WWW/autoriz.asp?lang=es⟨=en/
29   <html>
30     <link rel="stylesheet" href="/static/common.css" type="text/css"/>
31     <link rel="stylesheet" href="/static/autoriz.jsp" type="text/css"/>
32     meta http-equiv="refresh" content="30"/>
33   </head>
34   <body>
35     { content(htmlId) }
36   </body>
37   </html>
38 }
39
40 private def content(htmlId : String)(idId : Elem) = {
41   stringType match {
42     case "single" => elem { htmlId.map(htmlId => htmlId(htmlId)) } <html>
43       case "multi" => {
44         if (htmlId.length == 1) {
45           elem { htmlId.map(htmlId => htmlId(htmlId)) } <html>
46         } else {
47           set elem { htmlId.map(htmlId => htmlId(htmlId)(htmlId)) } <html>
48         }
49       }
50       case _ => set elem { htmlId.map(htmlId => htmlId(htmlId)(htmlId)) } <html>
51     }
52   }
53 }
54 private def autoriza(htmlId : Elem) = {
55   <table class=" border=" border-bottom border-top border-right border-left">
56     <tr>
57       <td> { autoriza(htmlId)(htmlId) } </td>
58     </tr>
59   </table>
60 }

```



---

# INTEGRATING MICROSERVICES IS HARD

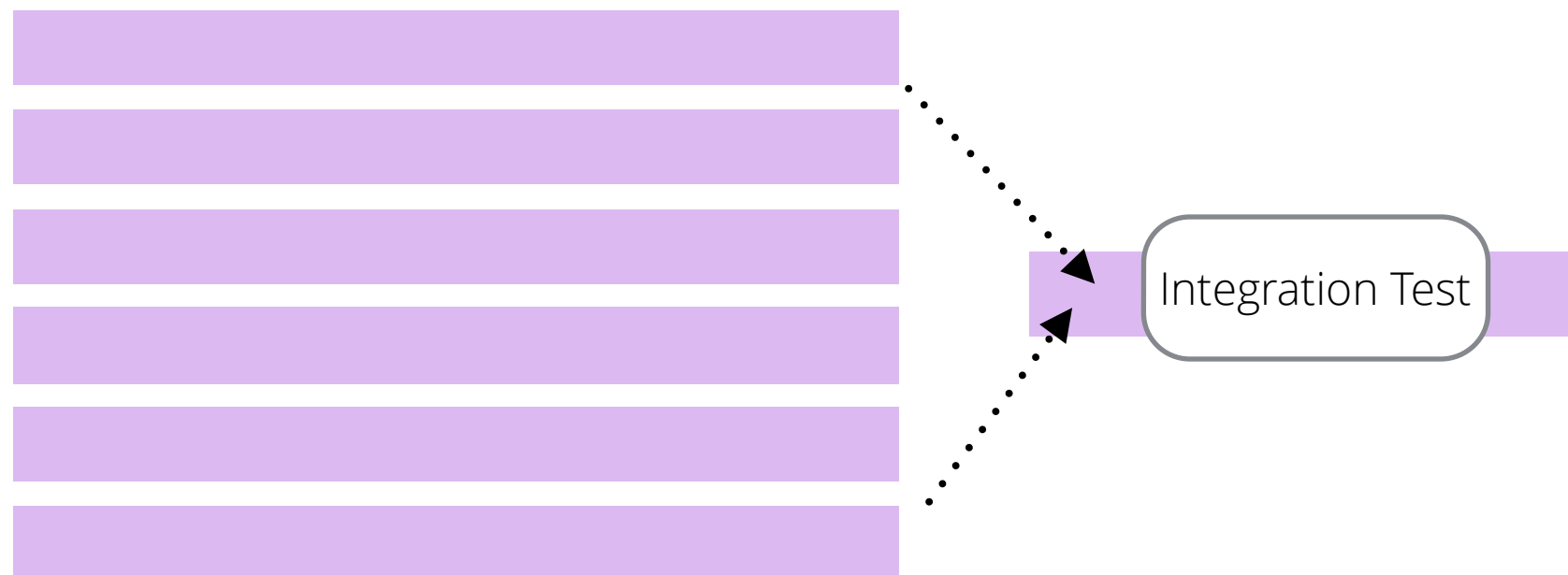
---



---

# INTEGRATING MICROSERVICES IS HARD

---

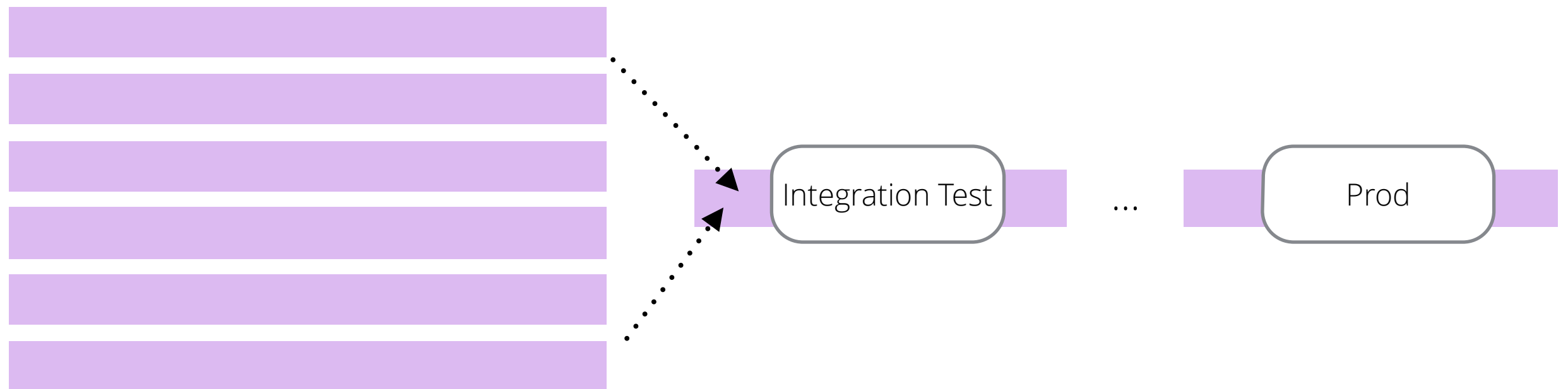




---

# INTEGRATING MICROSERVICES IS HARD

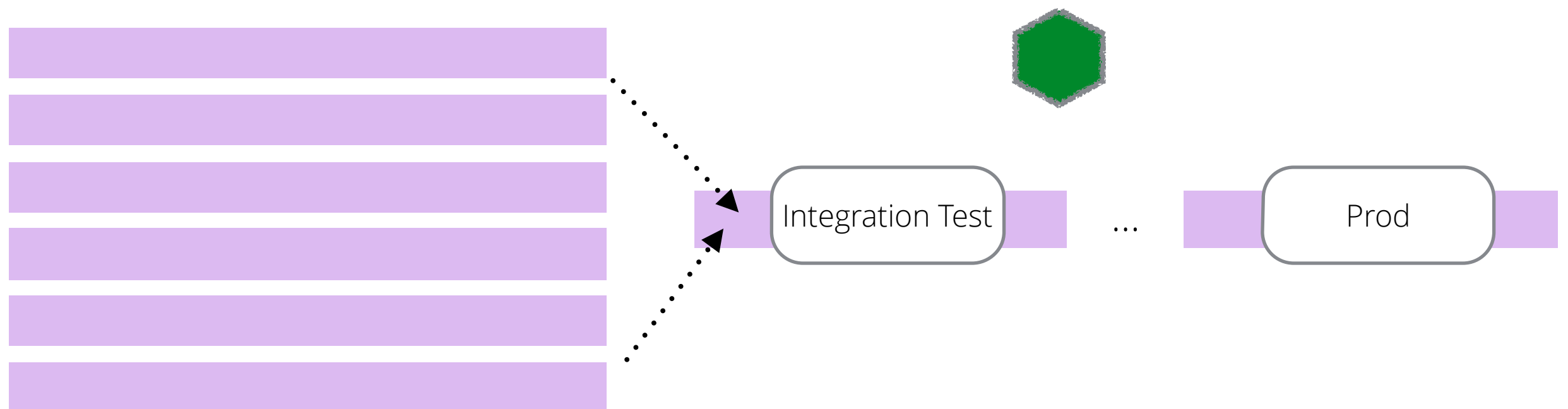
---



---

# INTEGRATING MICROSERVICES IS HARD

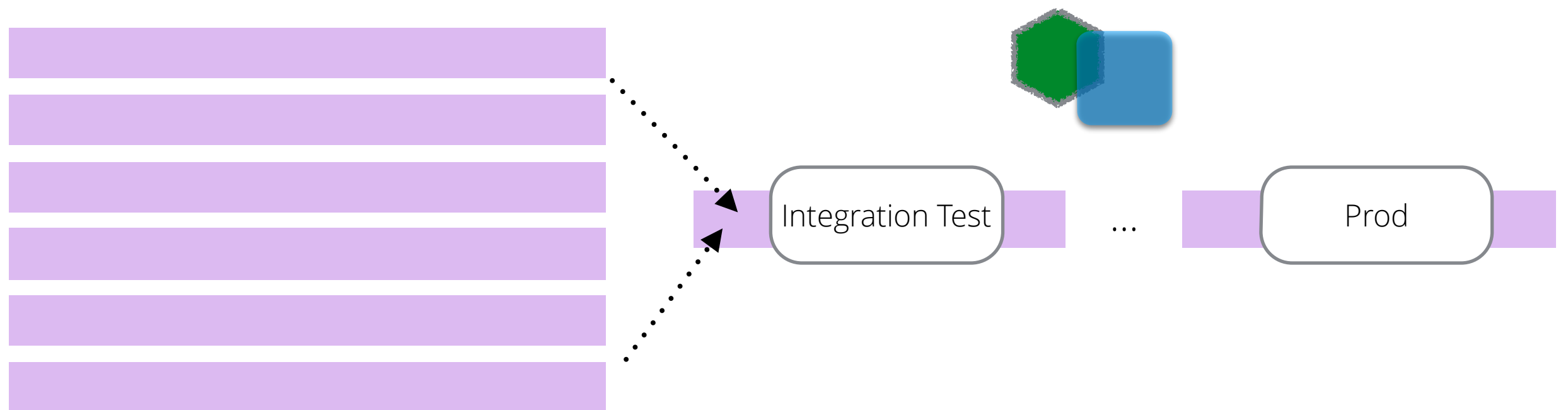
---



---

# INTEGRATING MICROSERVICES IS HARD

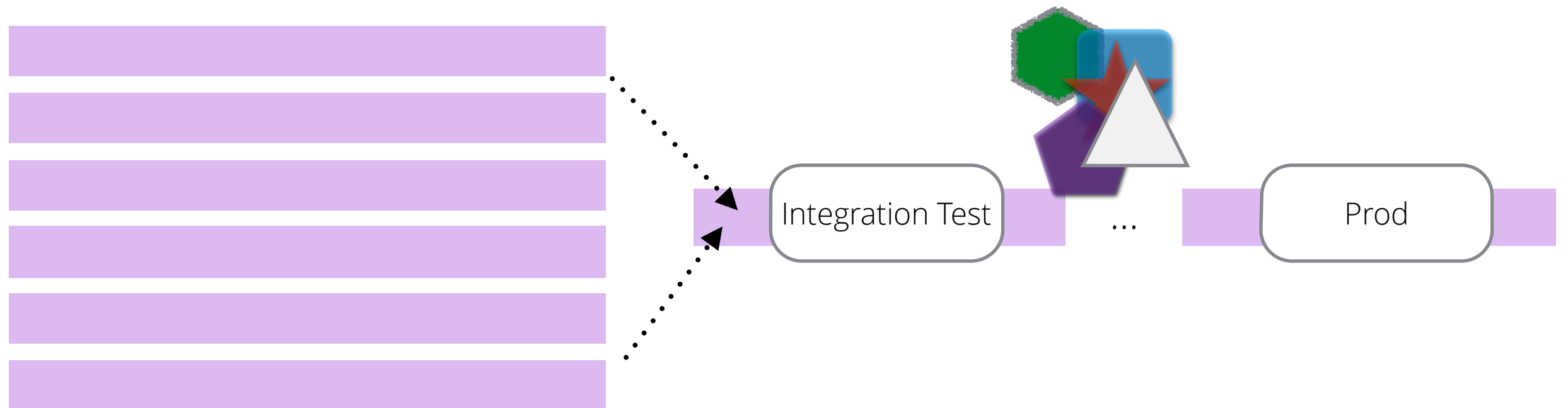
---



---

# INTEGRATING MICROSERVICES IS HARD

---





**<thinks>**

**World of Warcraft**

**YAGNI**

**GRASP**

**SOLID**

**agile**

**DRY**

**BDD**

**emergent design**

**GoF**

**Continuous Delivery**

**TDD**

**XP**

**KISS**

**Refactoring**





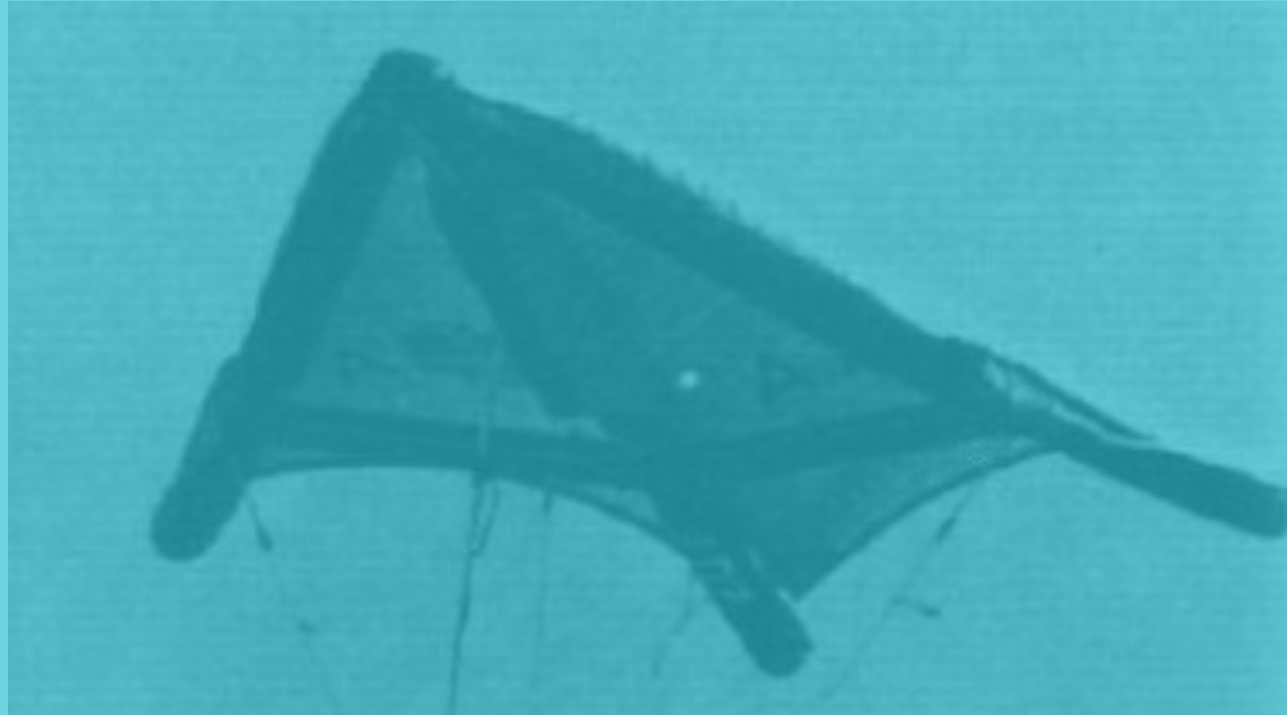
## **Gemini Project, Rogallo wing**

Source: [wikipedia.org](http://wikipedia.org)





**<thinks>**



*it's turtles all the way down*



**Gemini Project, Rogallo wing**

***World of Warcraft***

***YAGNI***

***GRASP***

***SOLID***

***agile***

***DRY***

***BDD***

***emergent design***

***GoF***

***Continuous Delivery***

***TDD***

***XP***

***KISS***

***Refactoring***

*GRASP*

*World of Warcraft*

*SOLID*

*agile*

*DRY*

**YAGNI**

*BDD*

*argued sign*

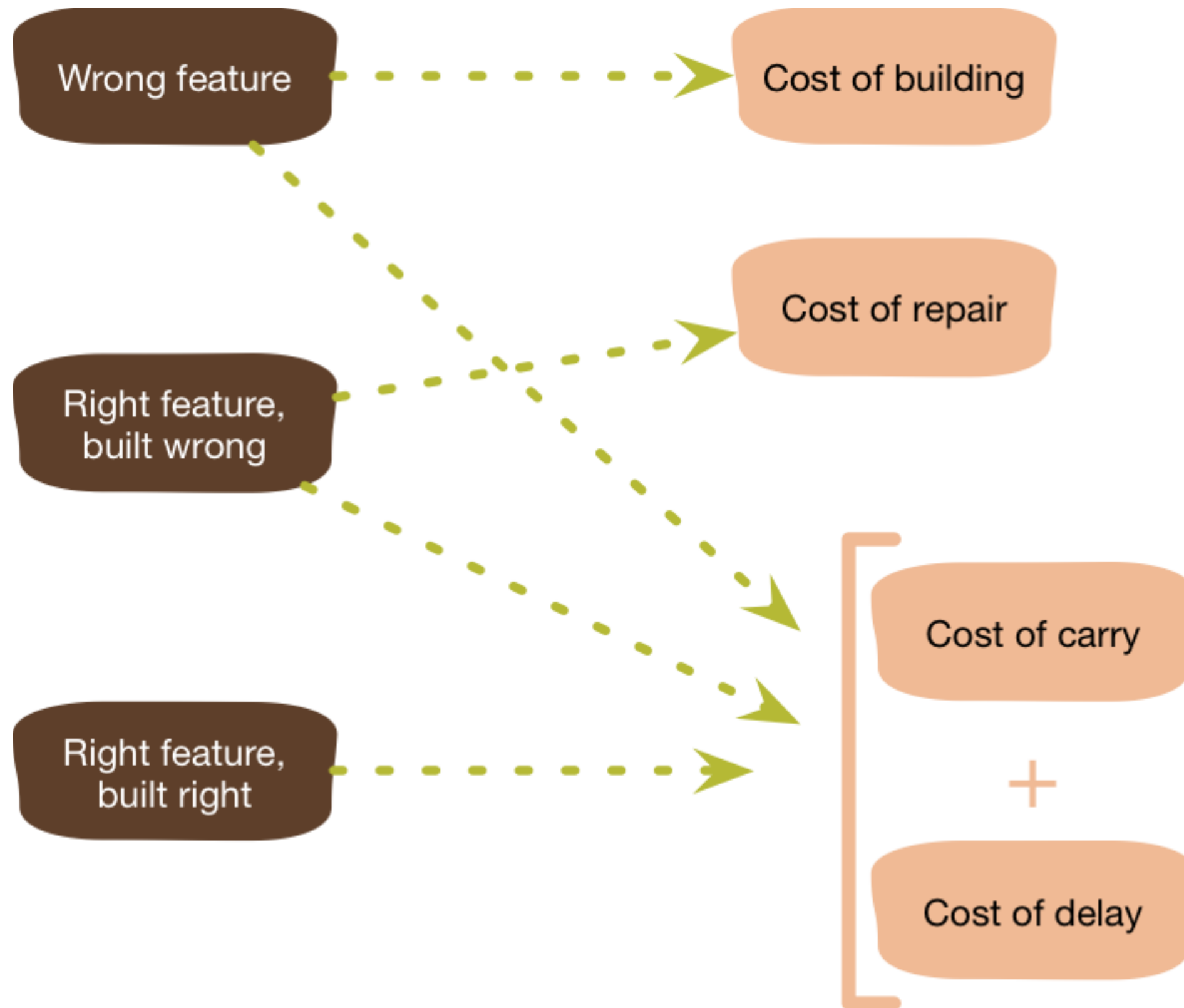
*Continuous Delivery*

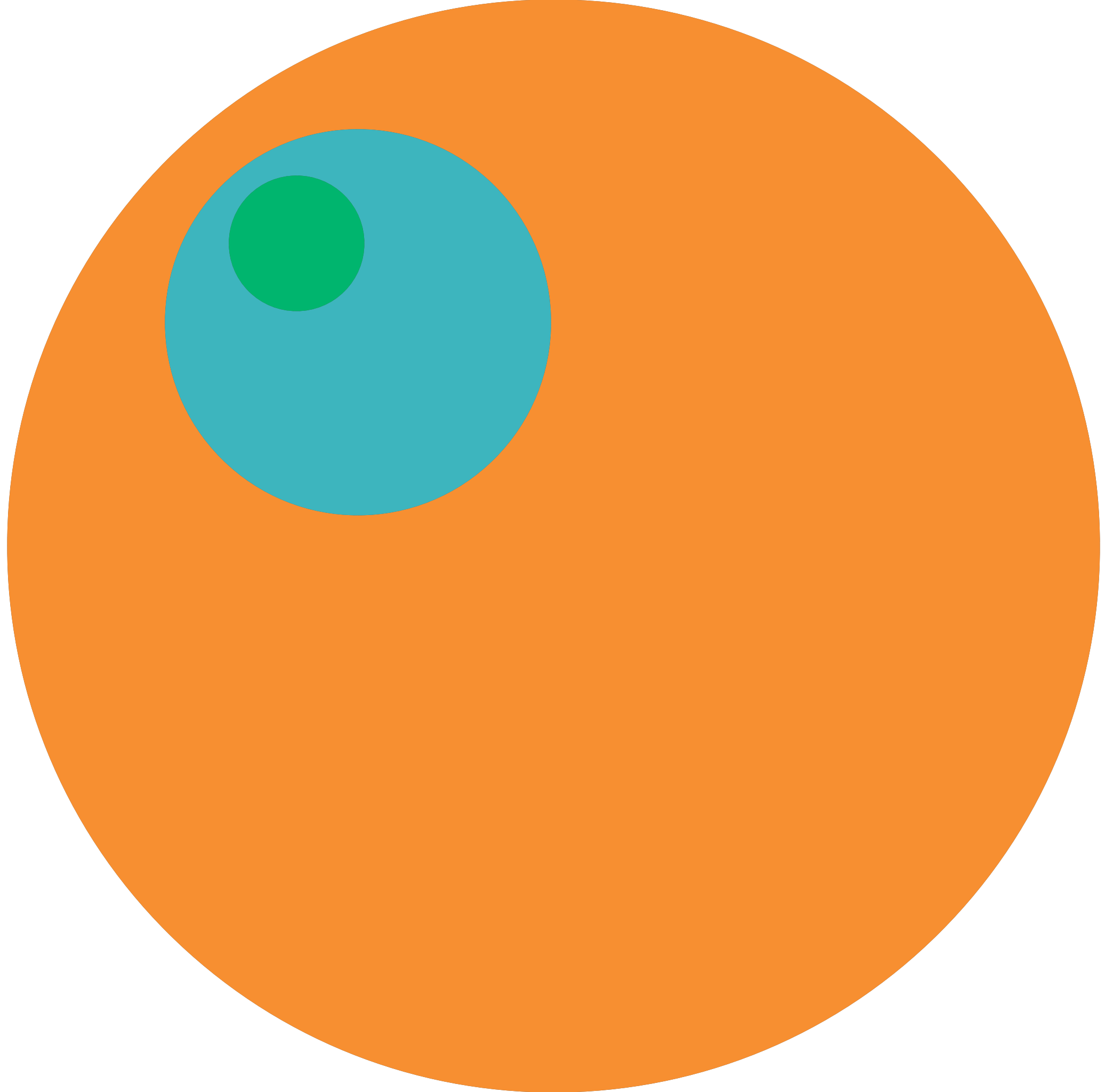
*TDD*

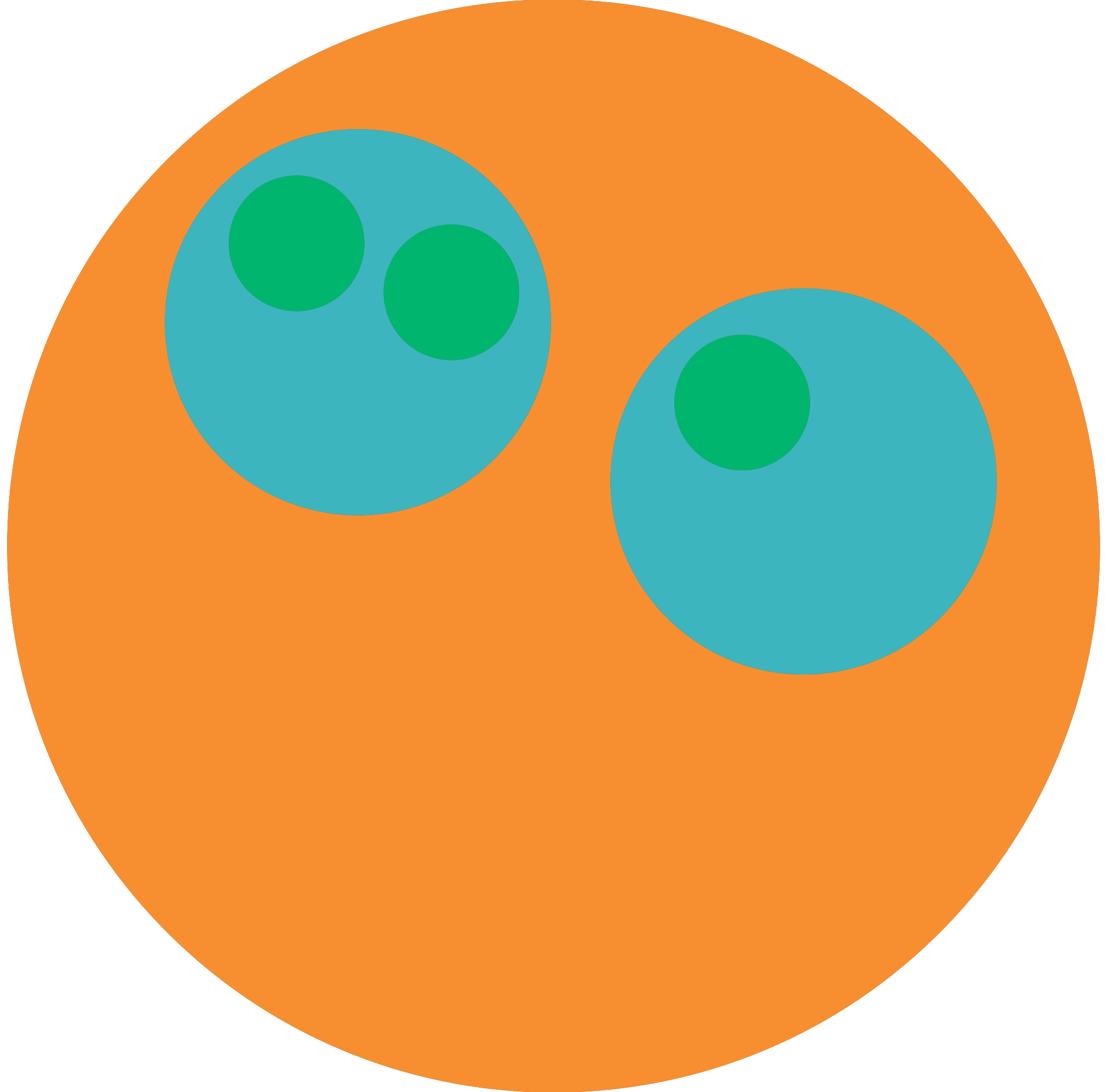
*XP*

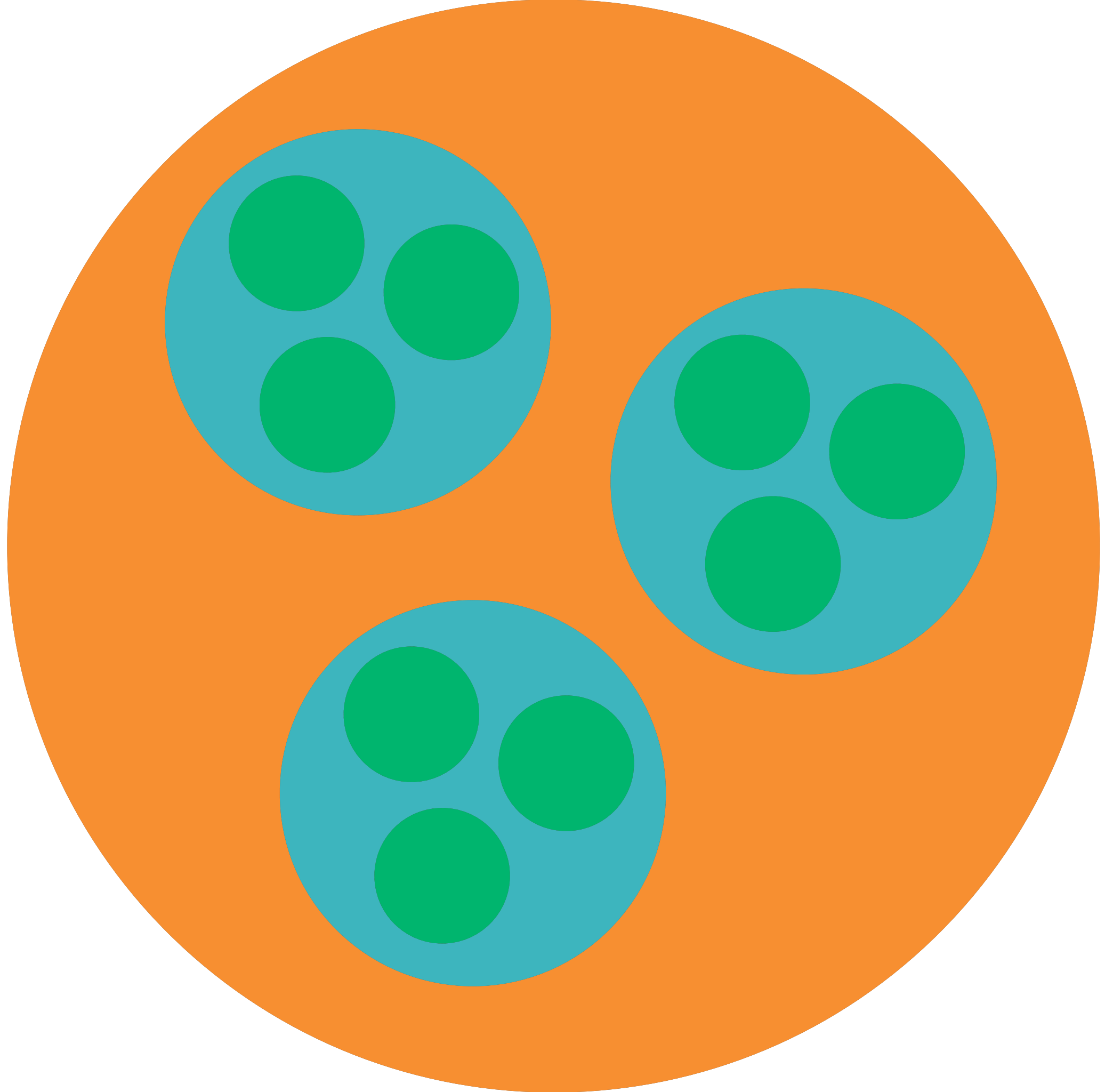
*KISS*

*Refactoring*













**Build out  
services as  
you need  
them**

YAGNI

*World of Warcraft*

SOLID

agile

DRY

**GRASP**

BDD

emergent design

CoE

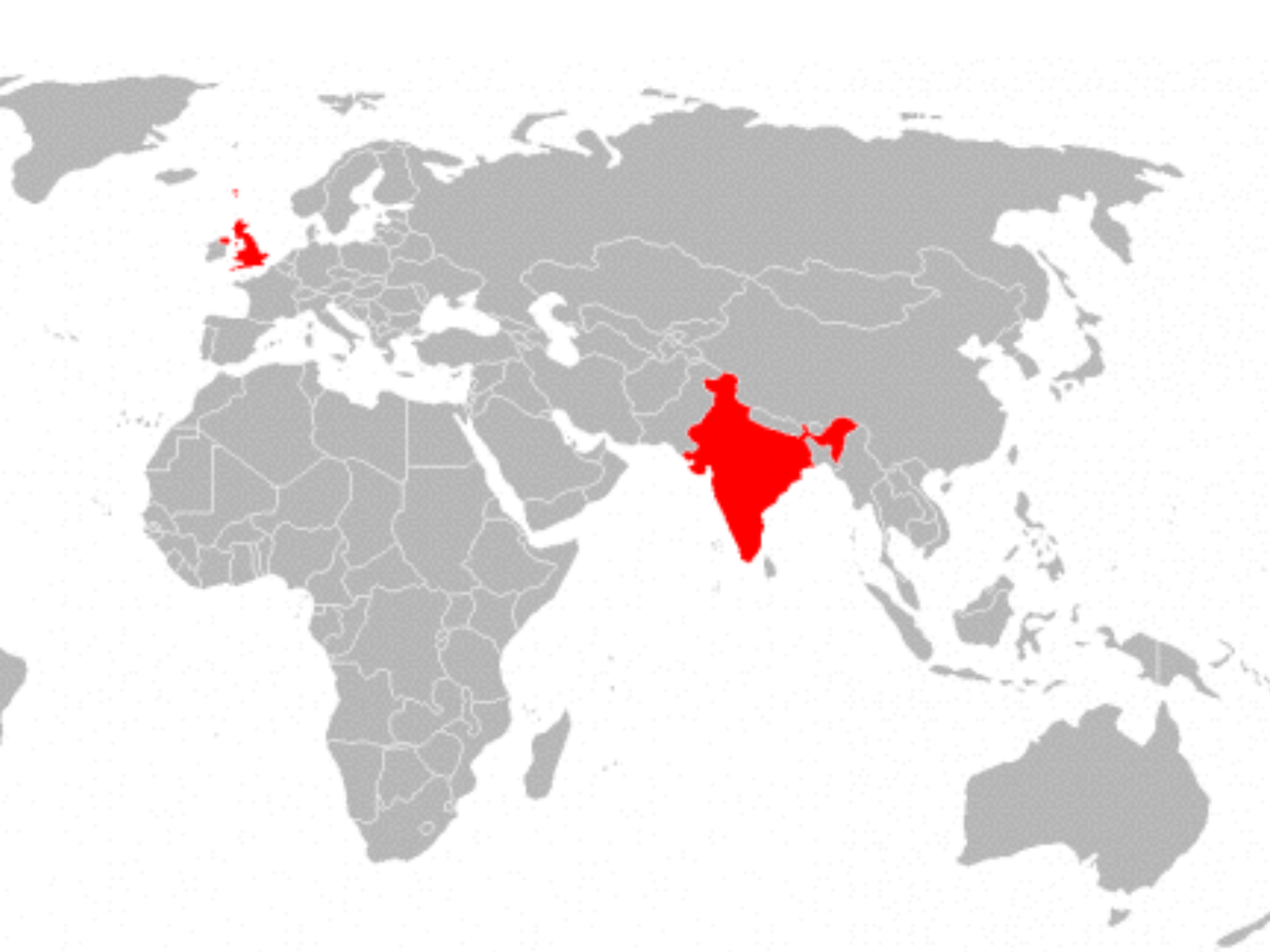
Continuous Delivery

TDD

XP

KISS

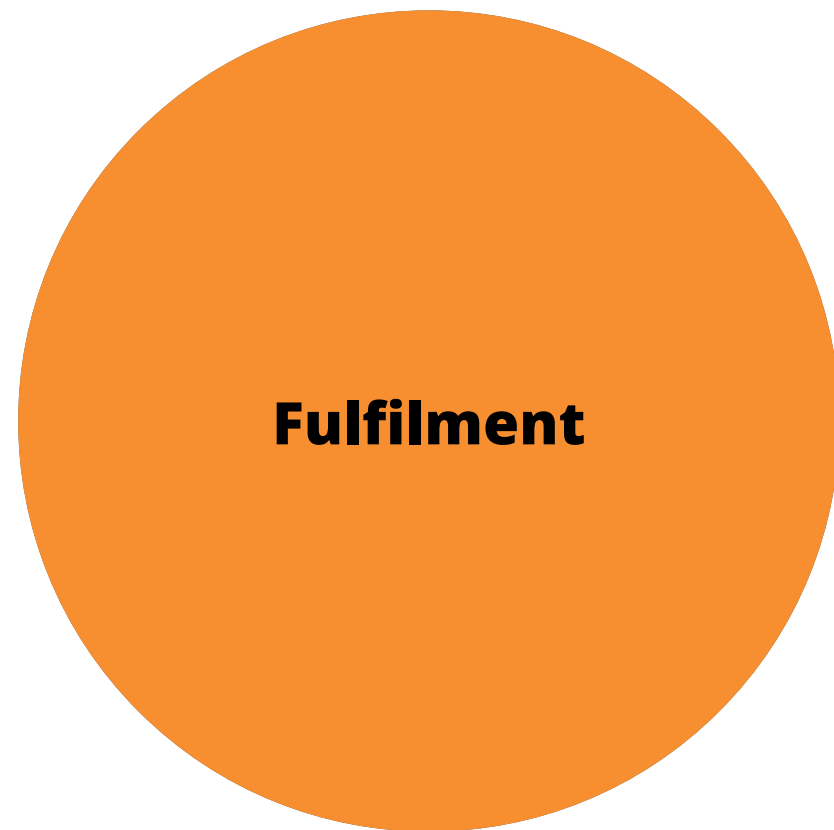
Refactoring

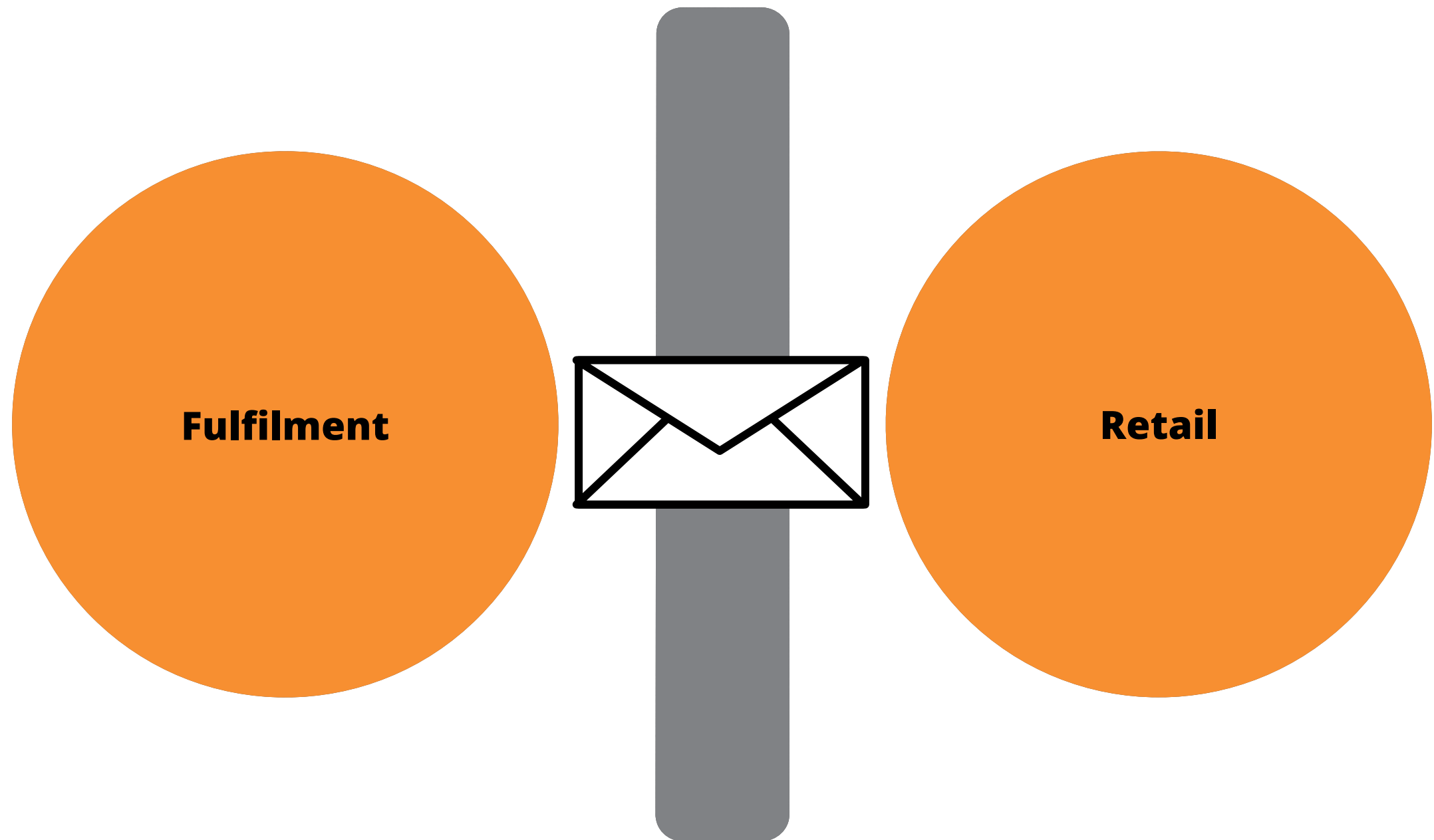


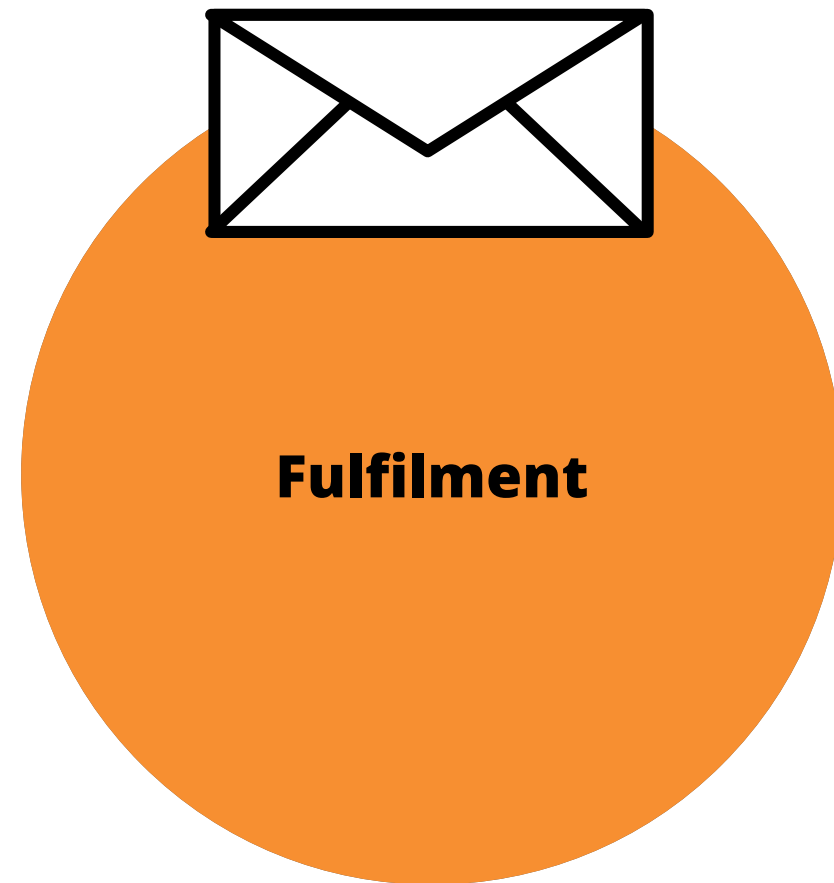
A grayscale world map with orange circles highlighting specific regions. One circle is over Western Europe labeled 'Fulfilment', and a larger circle is over East Asia labeled 'Retail'.

**Fulfilment**

**Retail**







# High cohesion



The diagram consists of two large orange circles positioned side-by-side. A vertical grey bar connects the top and bottom centers of these circles. The word 'Fulfilment' is centered inside the left orange circle, and the word 'Retail' is centered inside the right orange circle. The overall layout is symmetrical and centered on the page.

**Fulfilment**

**Retail**

# Low coupling



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

*World of Warcraft*

*YAGNI*

*GRASP*

*SOLID*

*agile*

**DRY**

*GoF*

*DDD*

*Continuous Delivery*

*TDD*

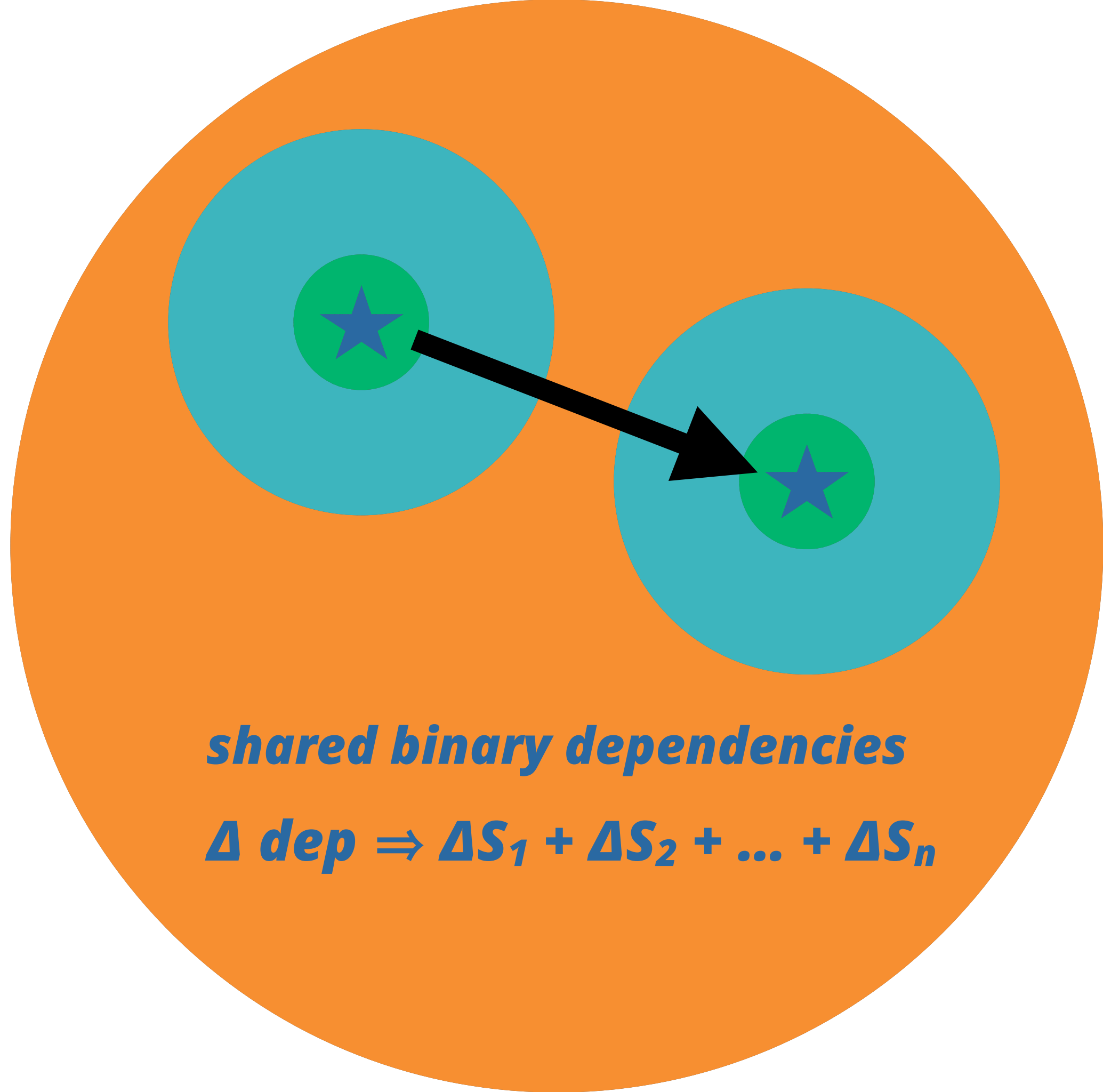
*XP*

*KISS*

*Refactoring*

*design*

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

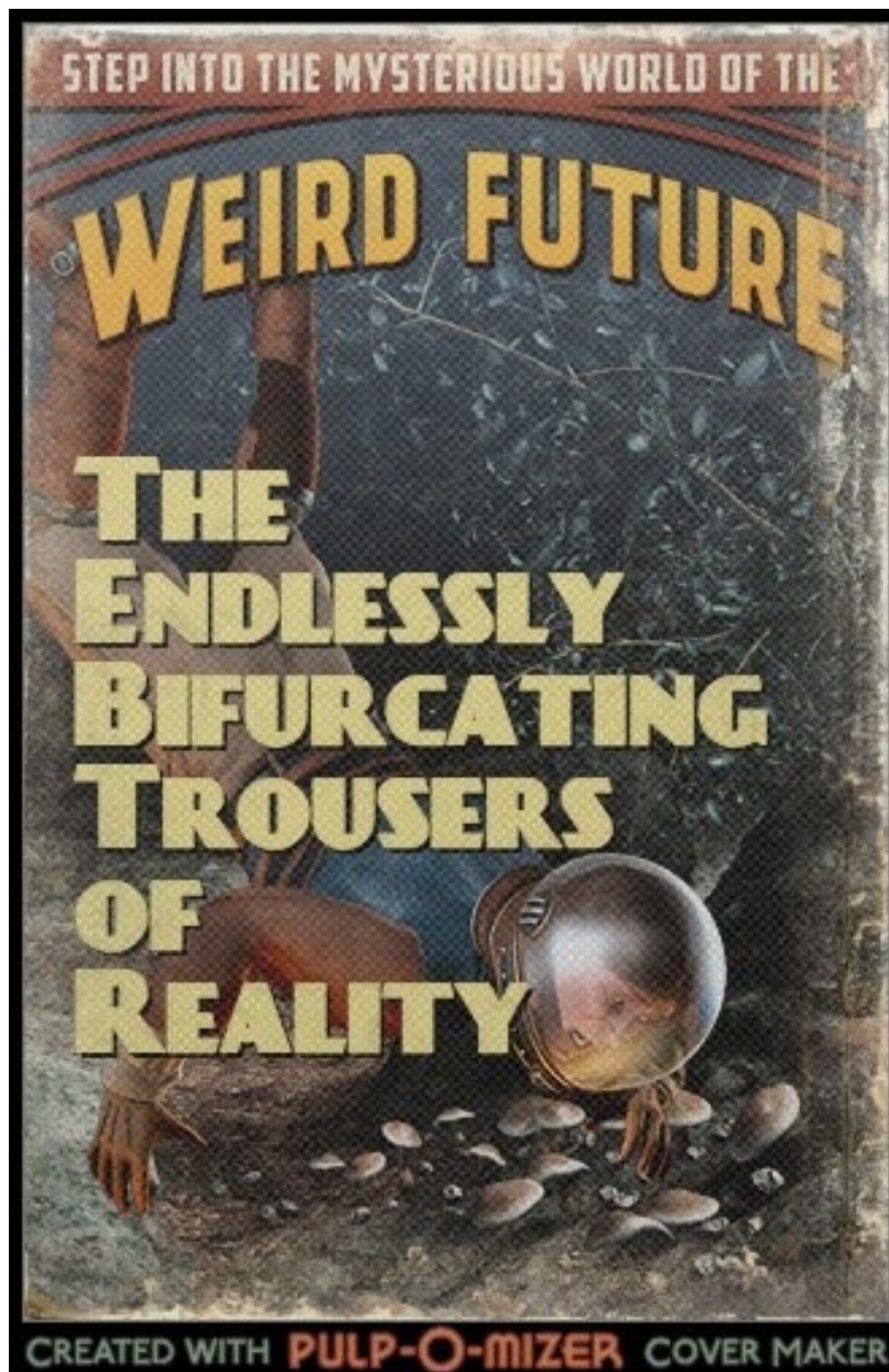


***shared binary dependencies***

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

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

***(note this doesn't exist)***





**DRY within services**  
**duplication between**  
**services**



*World of Warcraft*

*YAGNI*

*GRASP*

*SOLID*

*agile*

*DRY*

*BDD*

*GoF*

**TDD**

*merge*

*Continuous Delivery*

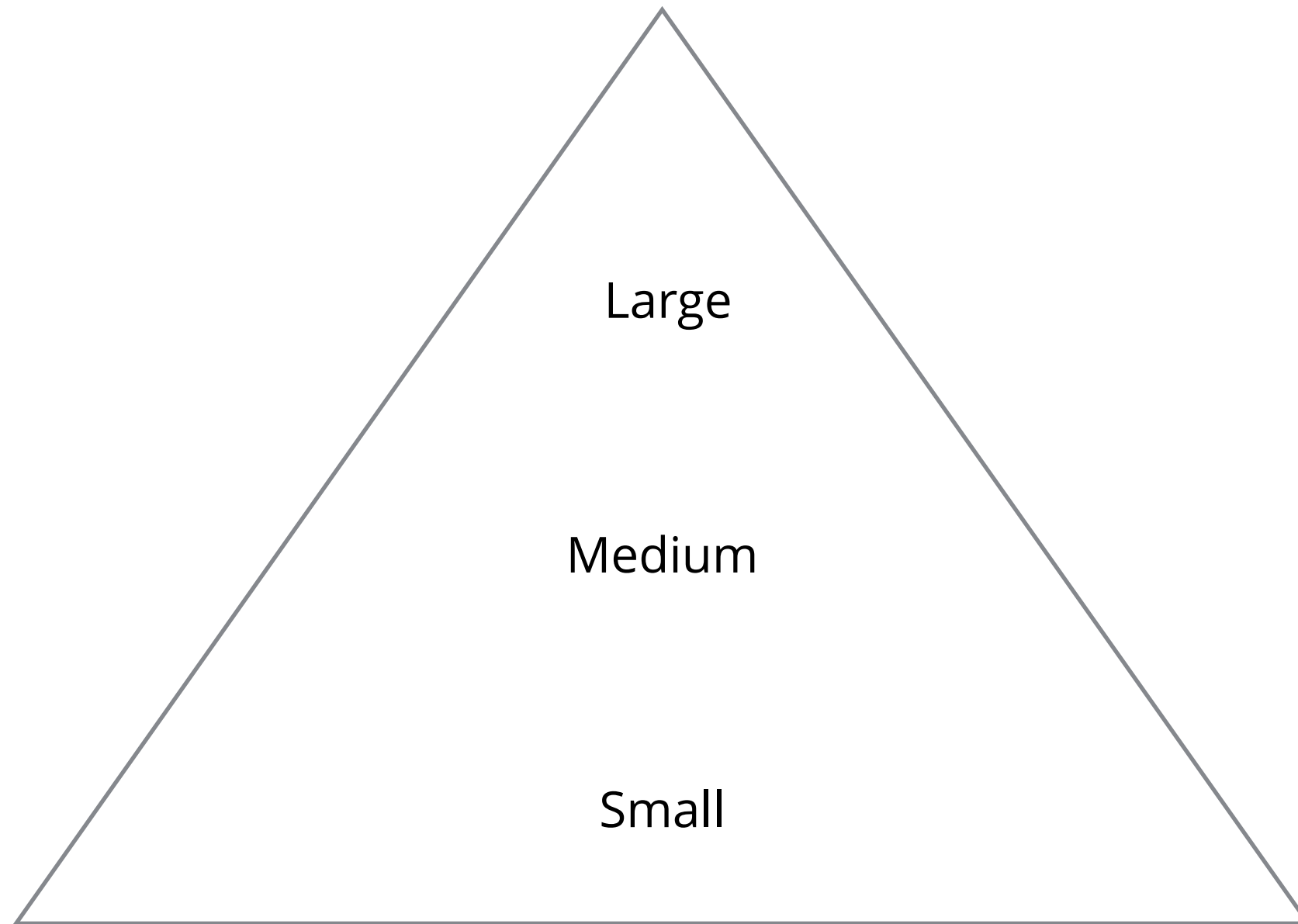
*XP*

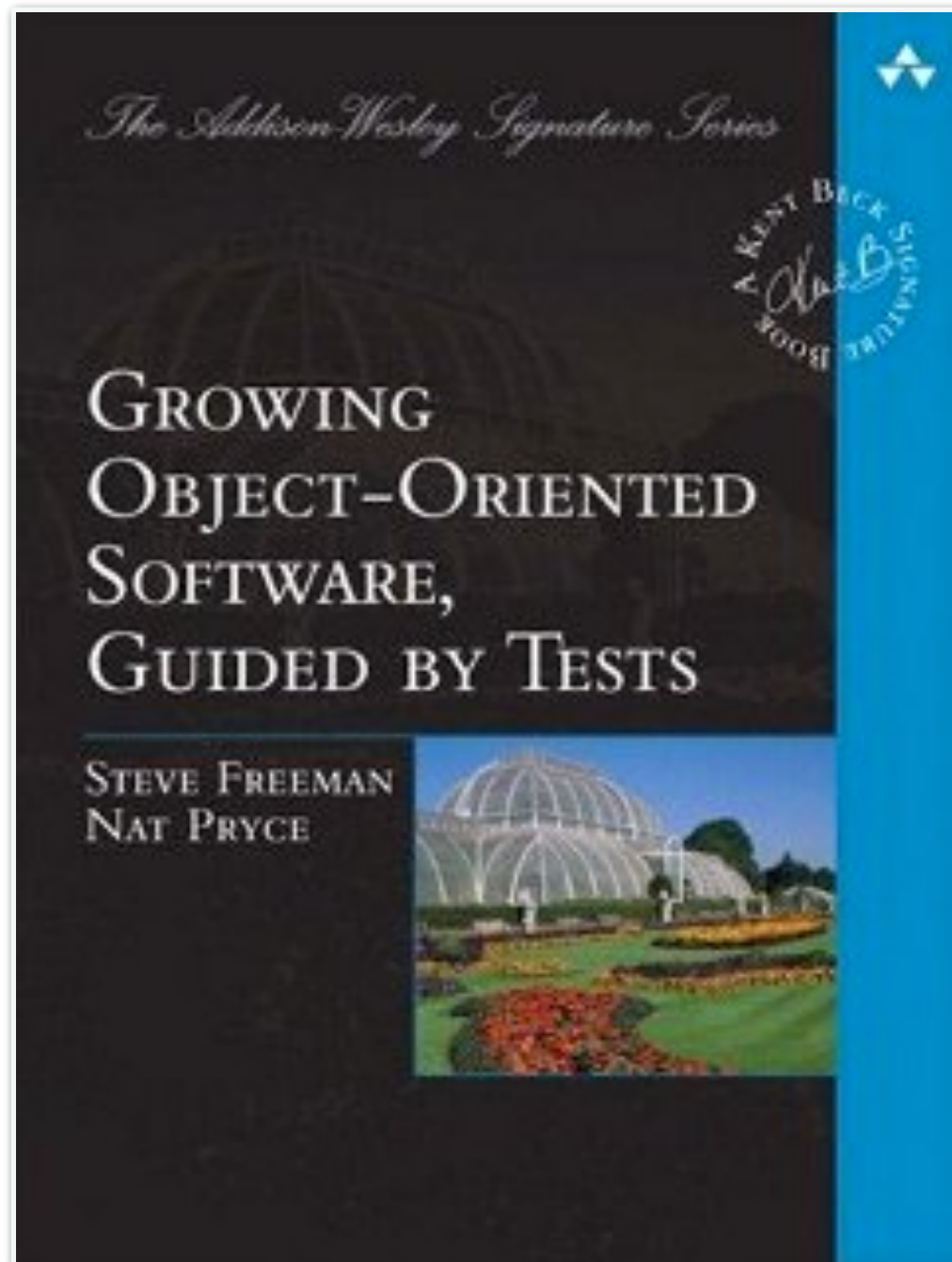
*KISS*

*Refactoring*









# *"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**

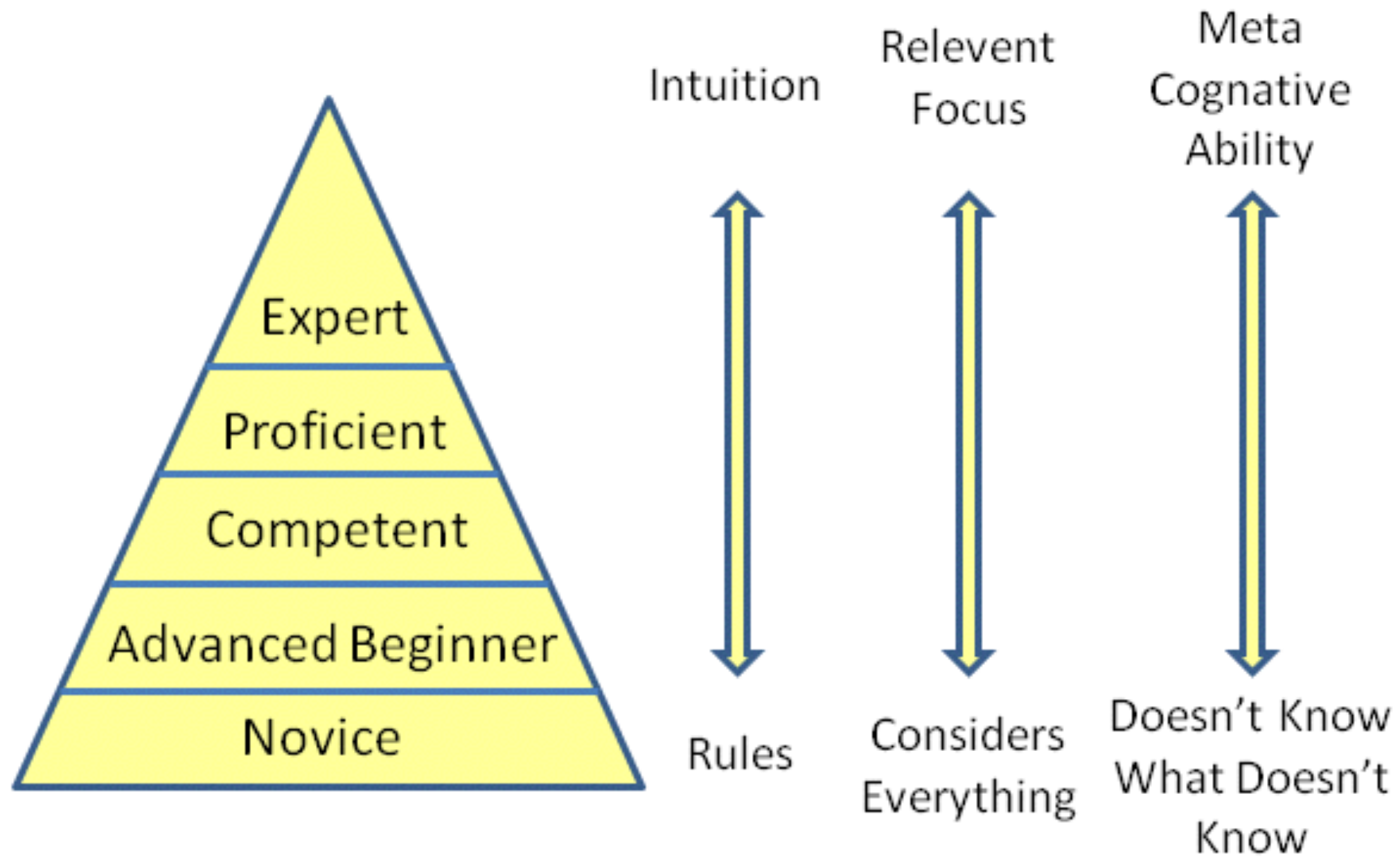
*Jez Humble*

*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\****



*World of Warcraft*

*YAGNI*

*GRASP*

*DRY*

*agile*

*BDD*

*GoF*

**SRP**

*t des*

*Continuous Delivery*

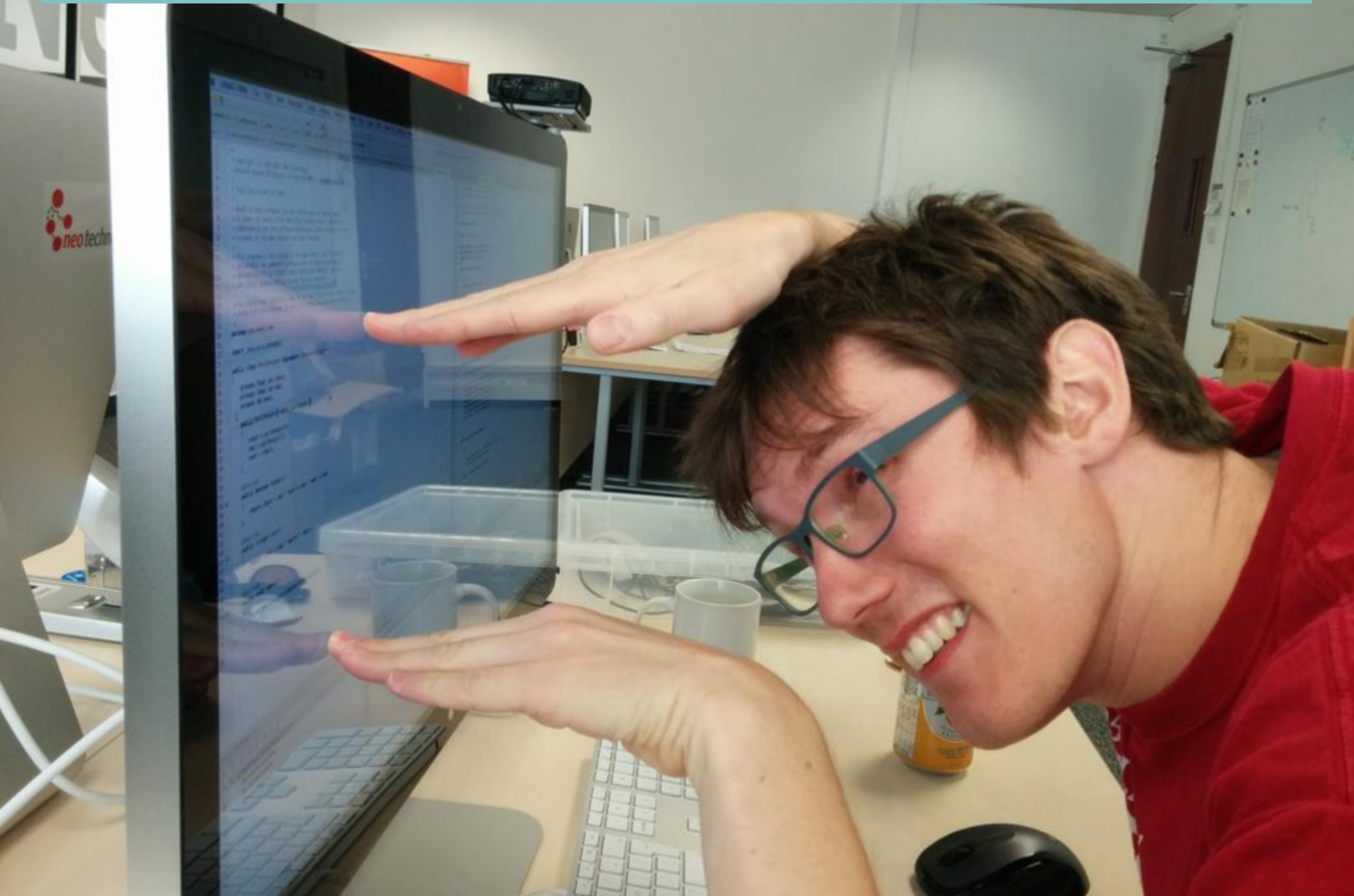
*TDD*

*XP*

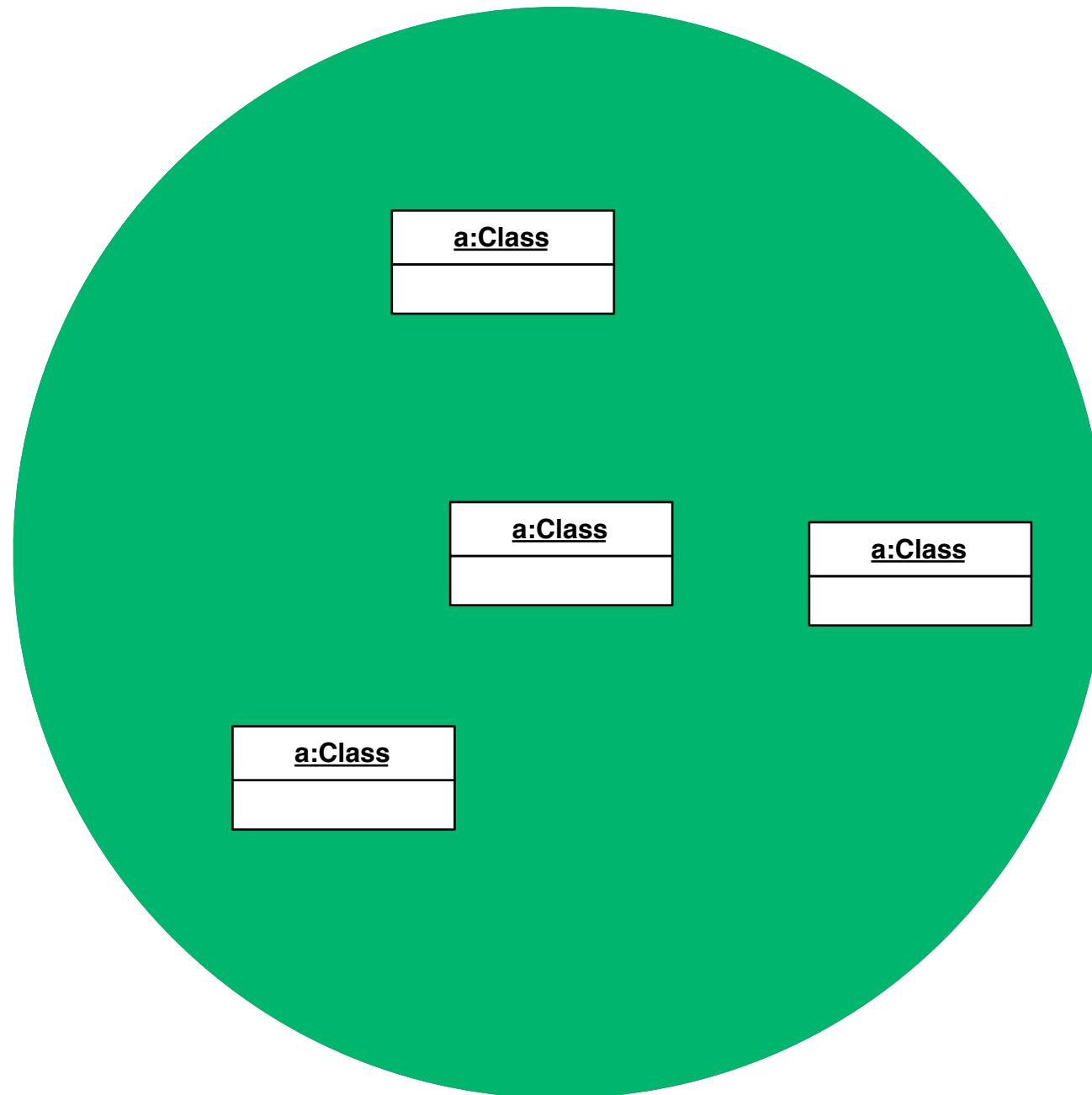
*KISS*

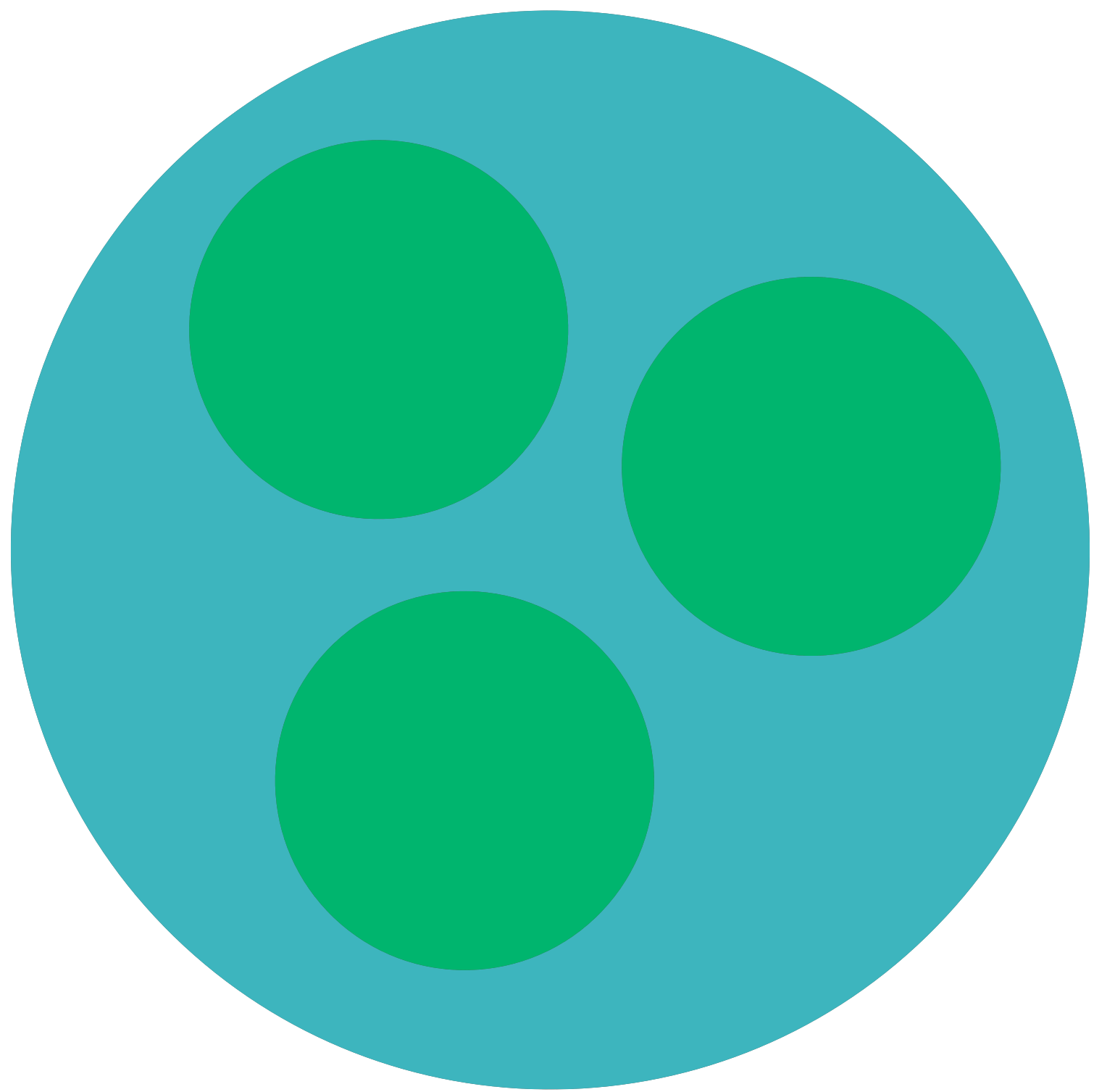
*Refactoring*

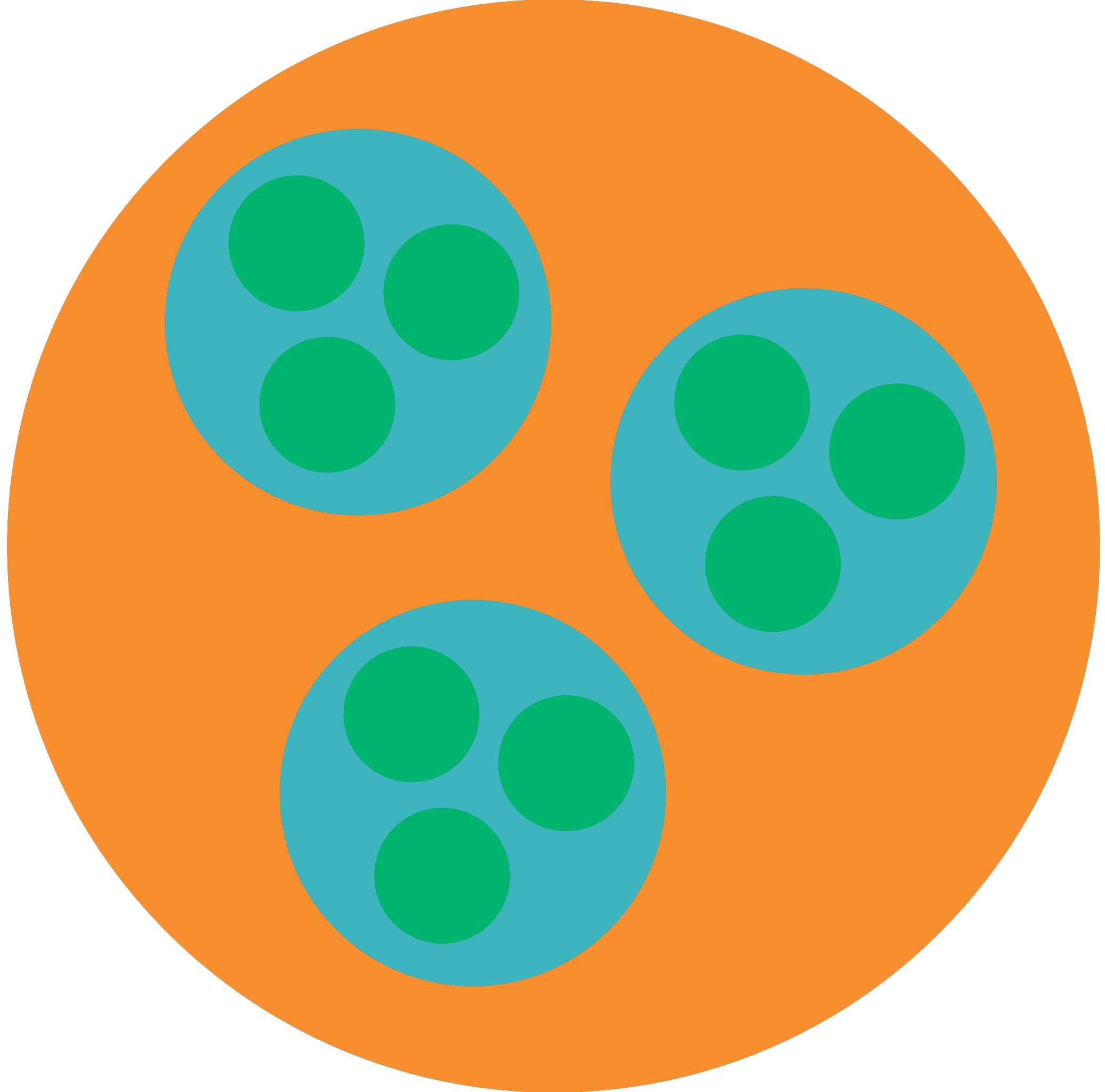
*a class should be no bigger than my head*



<b><u>a:Class</u></b>









A woman with dark hair and glasses is sitting at a desk in an office. She is looking at a large computer monitor that displays some text. Her hands are pressed against her temples, and she has a frustrated or overwhelmed expression on her face. The office environment includes a keyboard, a mouse, and some papers on the desk. The background shows office shelves and a window.

# **SRP**

**a service should  
be no bigger than  
my head**

*World of Warcraft*

*YAGNI*

*GRASP*

*SOLID*

*agile*

*DRY*

*GoF*

**KISS**

*emergent design*

*DD*

*Continuous Delivery*

*TDD*

*XP*

*Refactoring*



## ●HOLD

### 45.Application Servers new

---

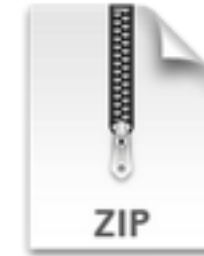
The rise of containers, phoenix servers and continuous delivery has seen a move away from the usual approach to deploying web applications. Traditionally we have built an artifact and then installed that artifact into an application server. The result was long feedback loops for changes, increased build 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

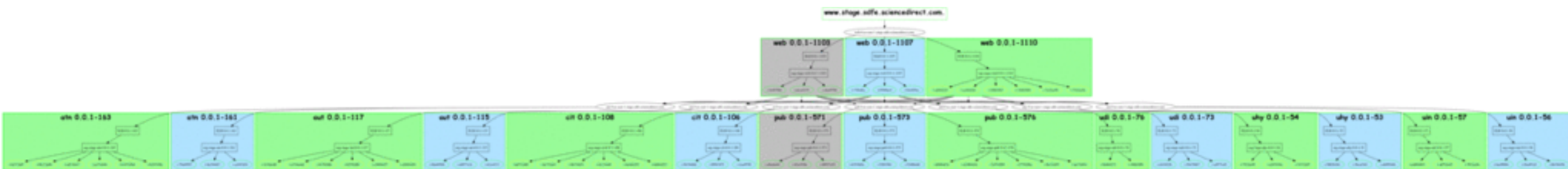
- [Webbit Google Group](#)
- [@webbitserver on Twitter](#)
- [Webbit Wiki](#)

*Get the source code from GitHub: [webbit/webbit](#)*

*cron, python, boto, pydot, graphviz*



*cron, python, boto, pydot, graphviz*

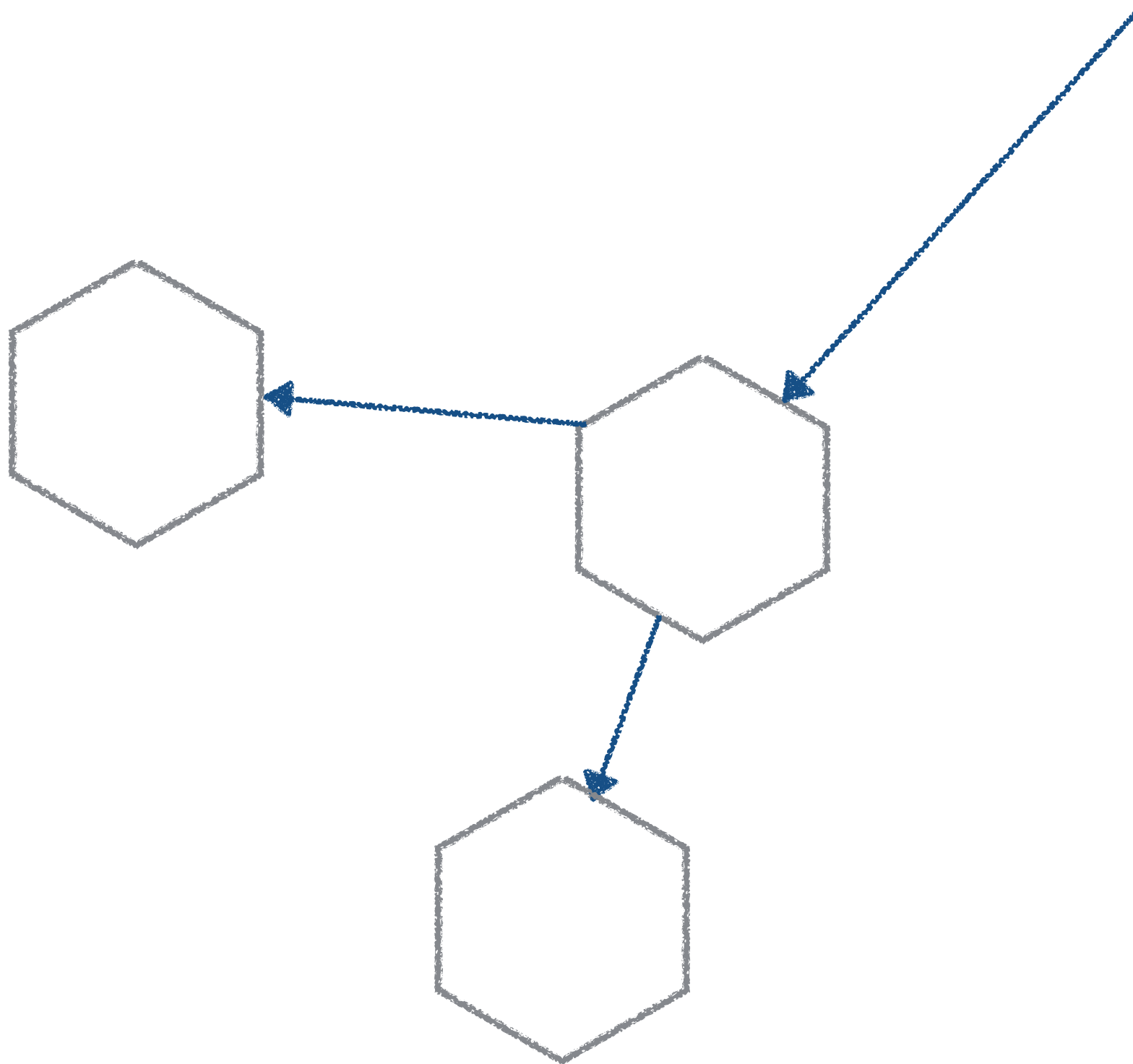


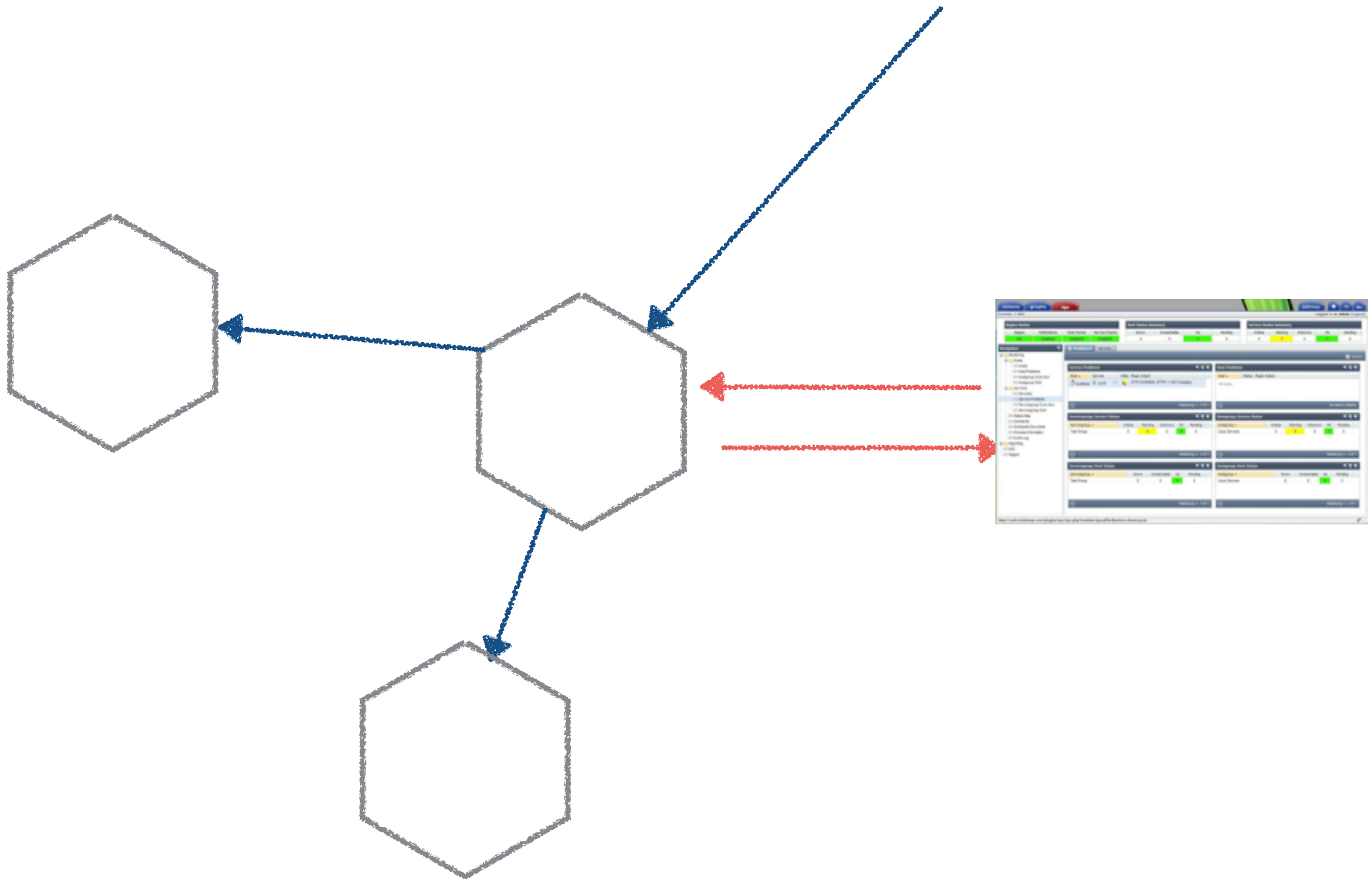
*cron, python, boto, pydot, graphviz*

**Do the simplest  
thing possible**

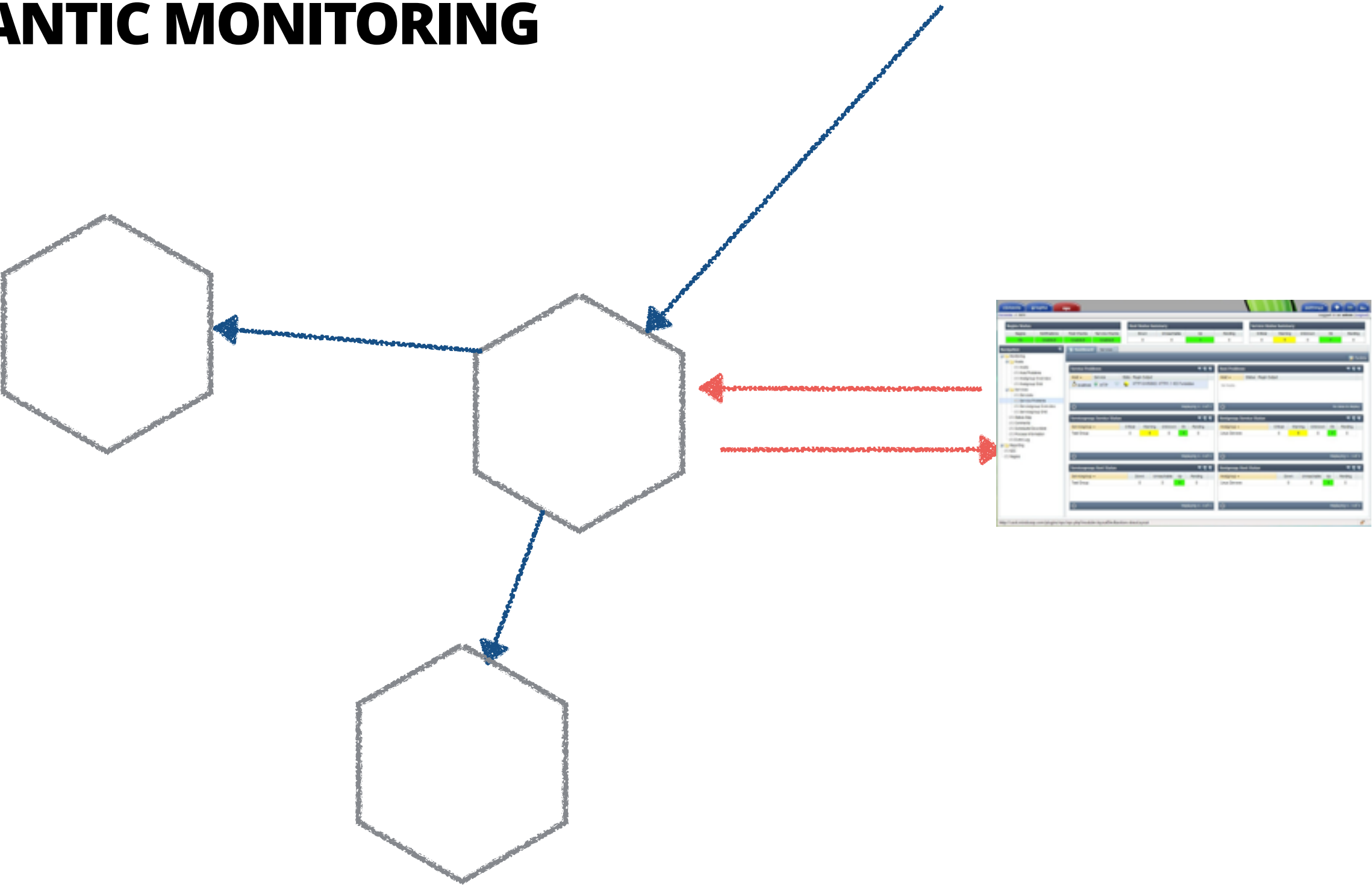
# **integration and deployment**

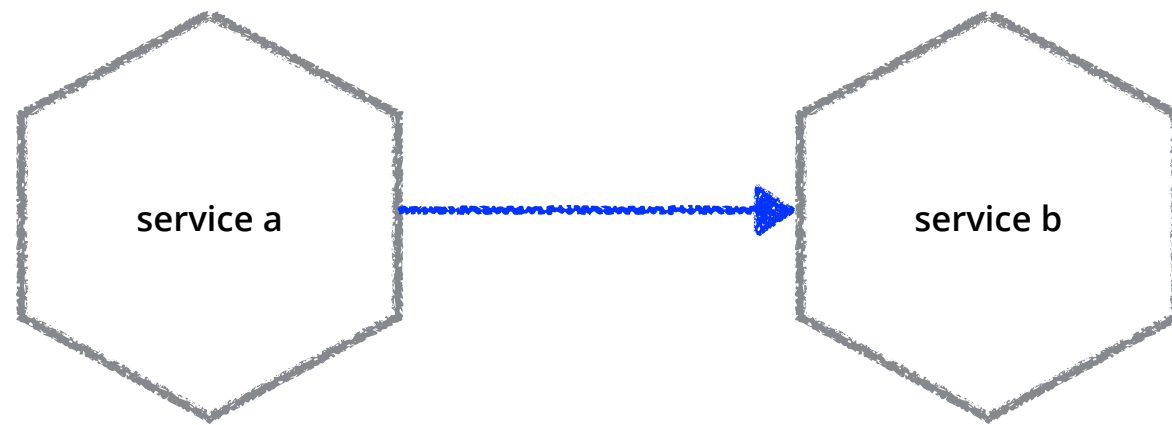


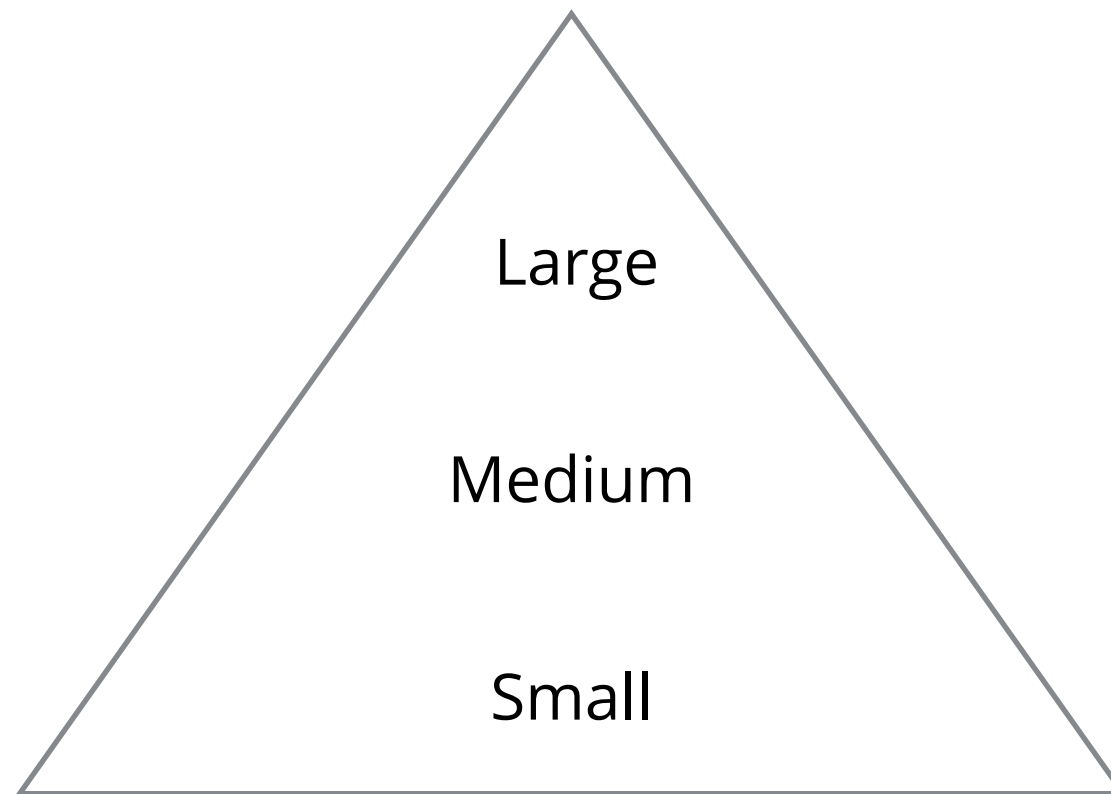
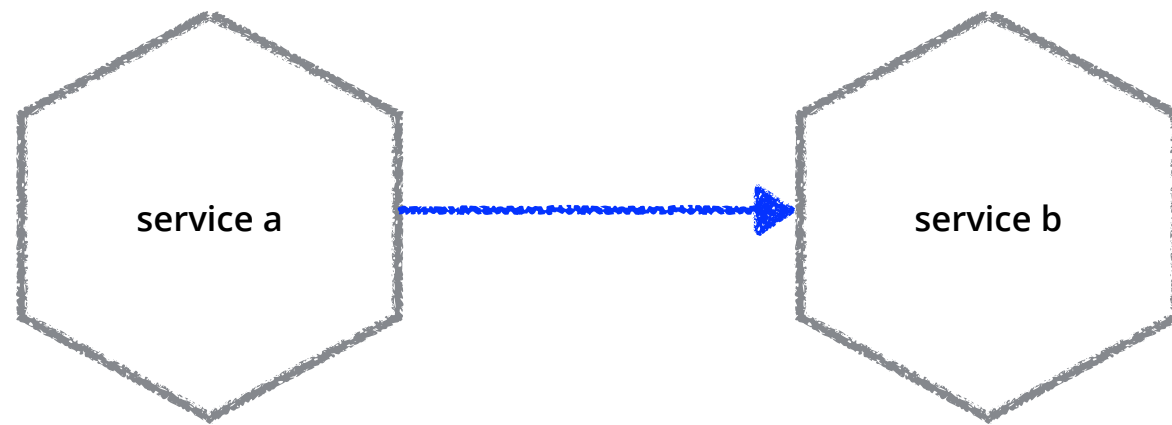


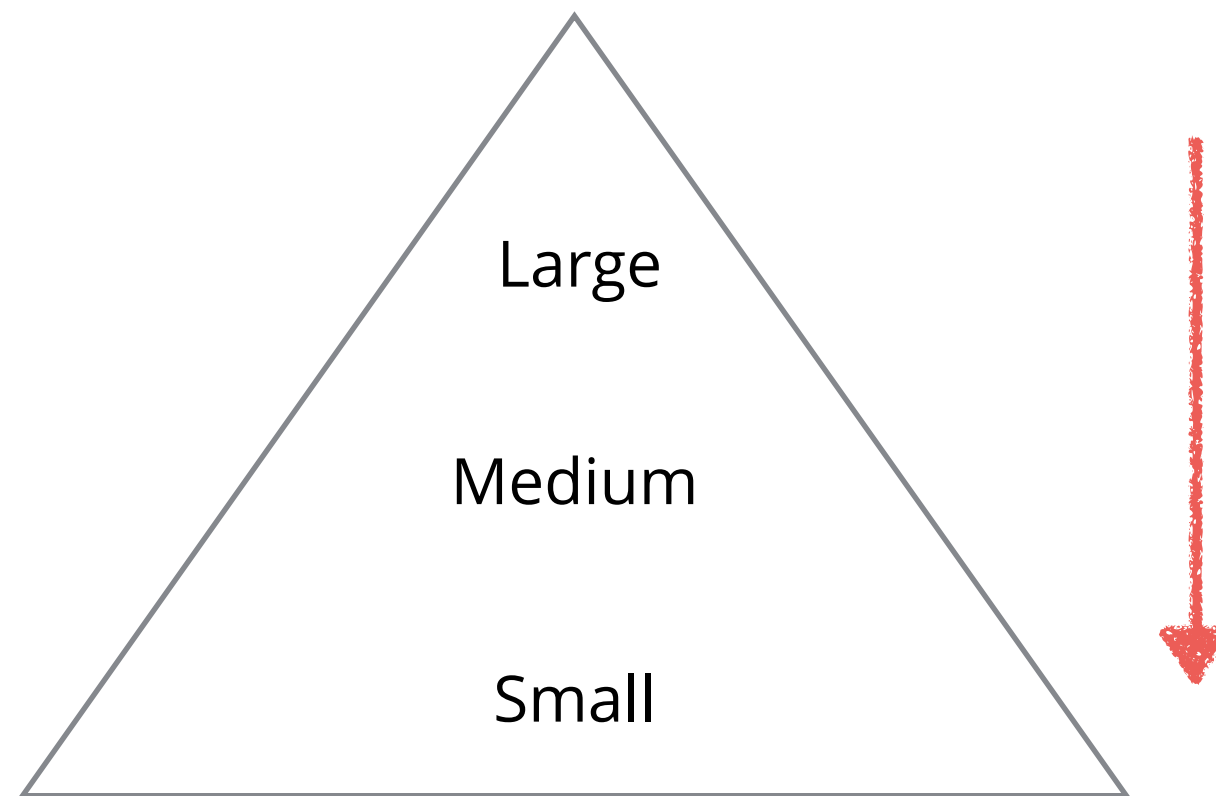
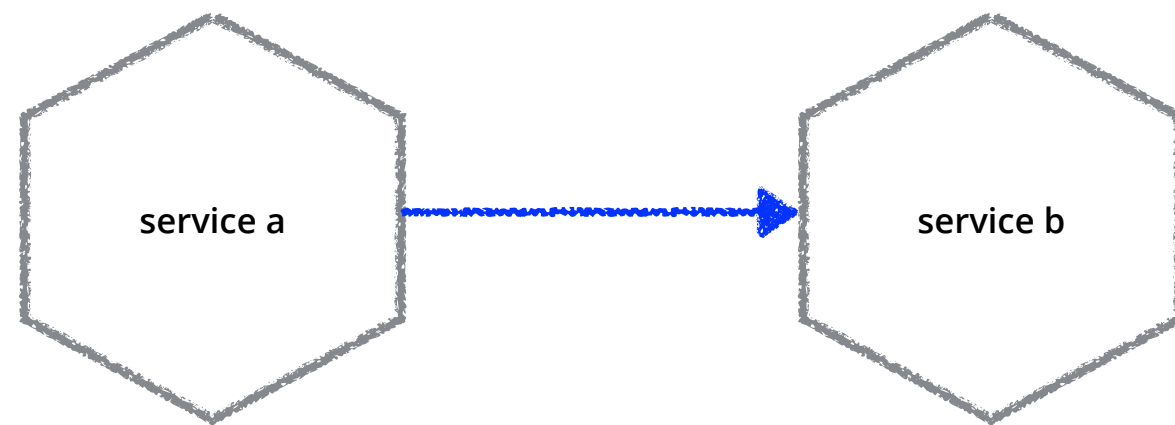


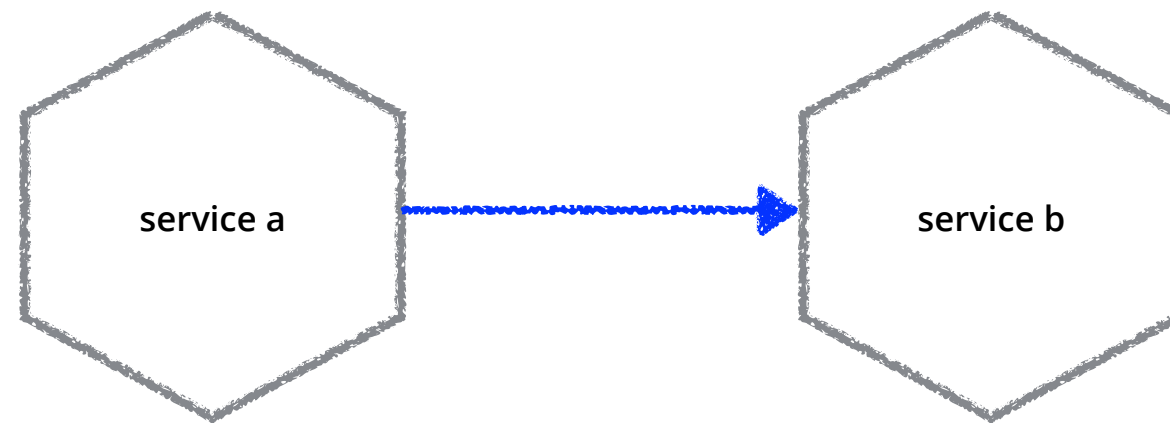
# SEMANTIC MONITORING



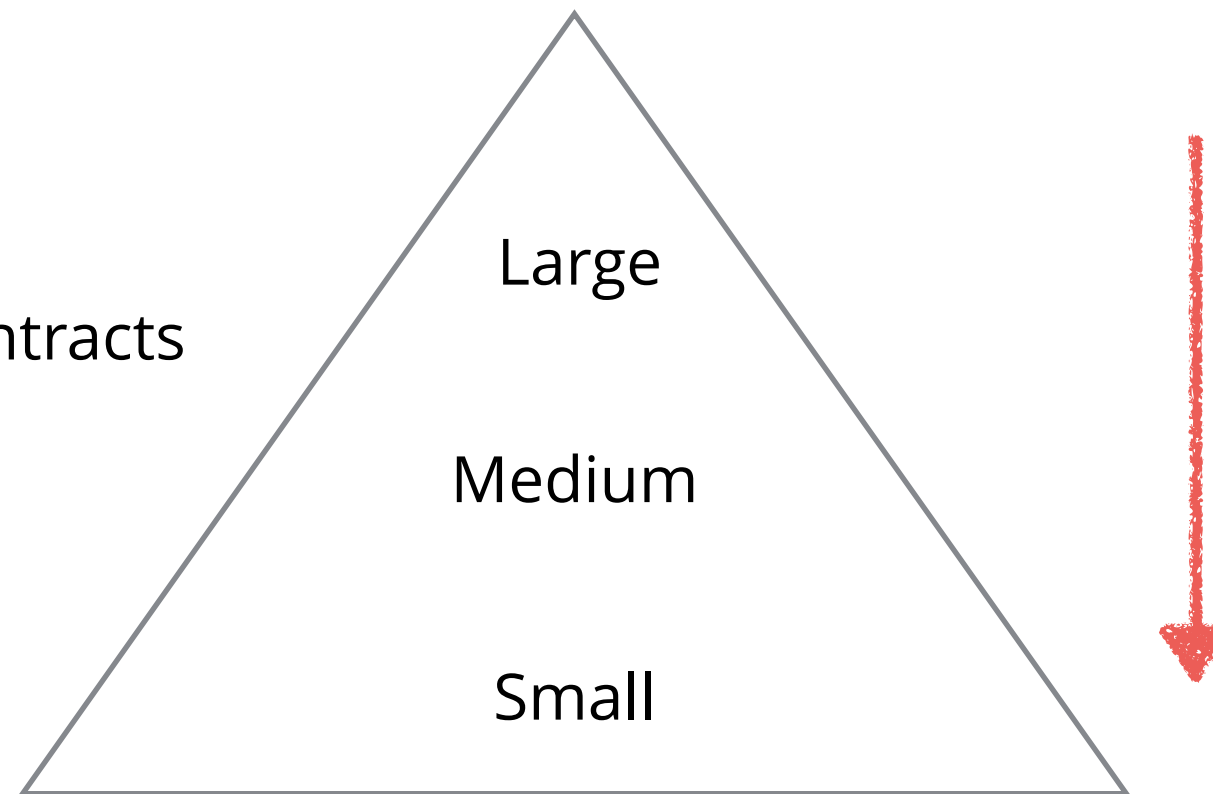


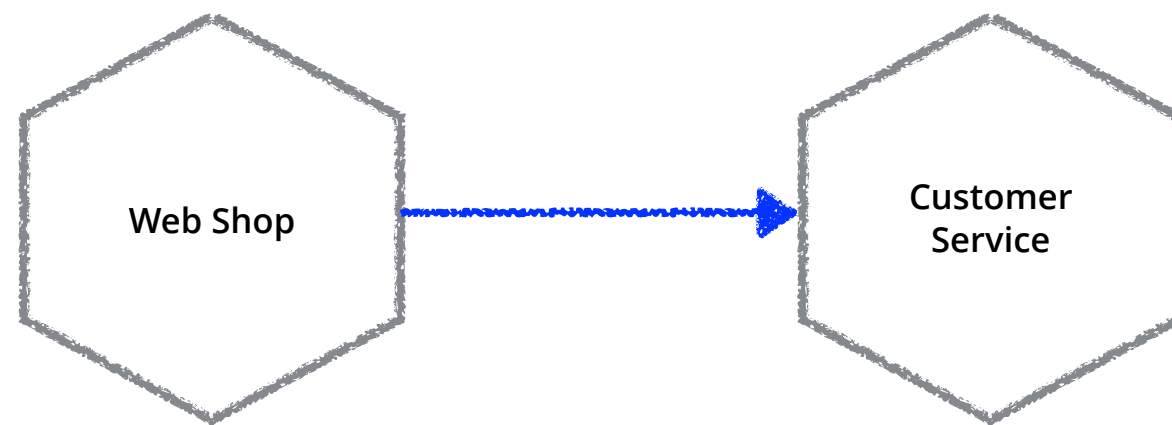




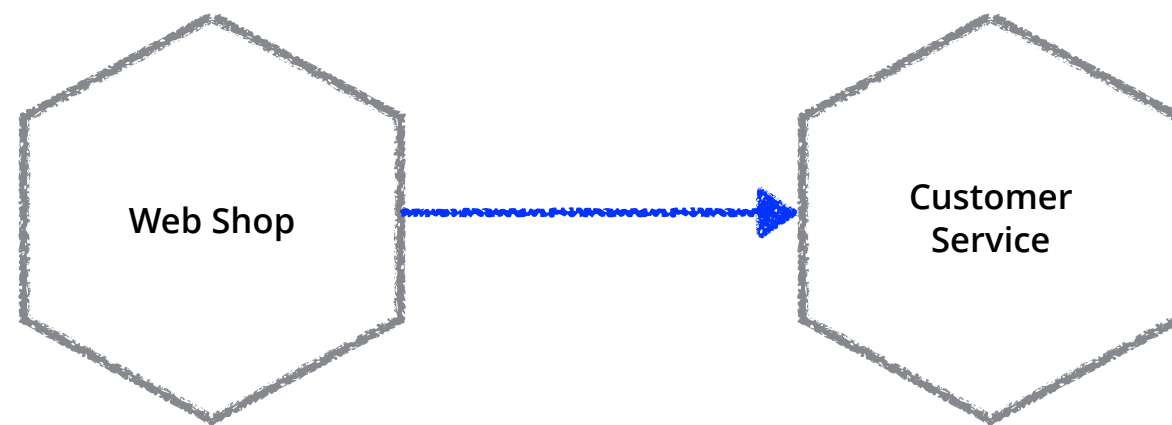


Consumer Driven Contracts

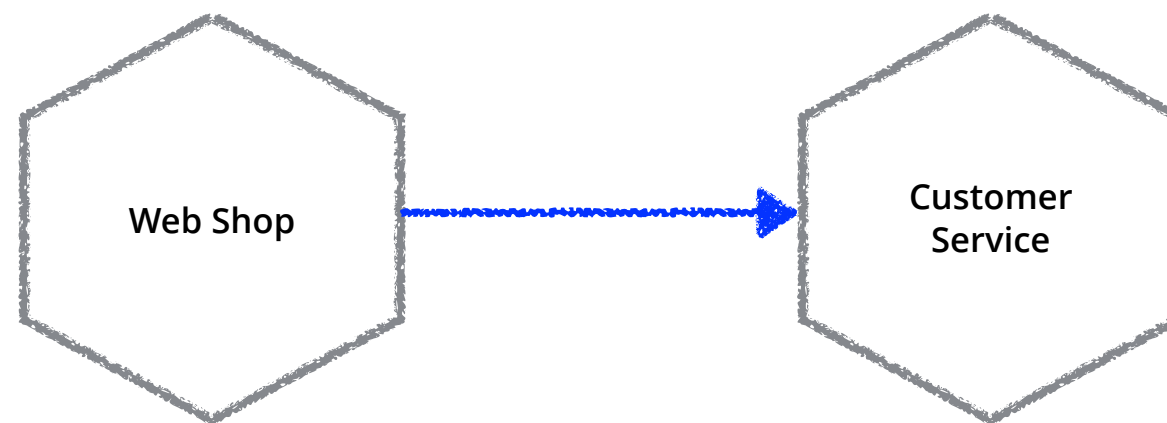








Expectations

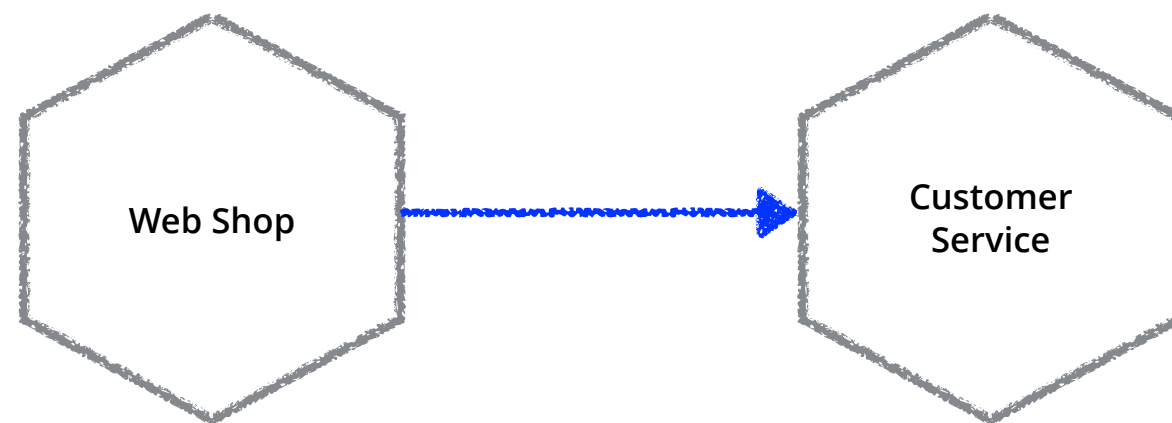


Expectations

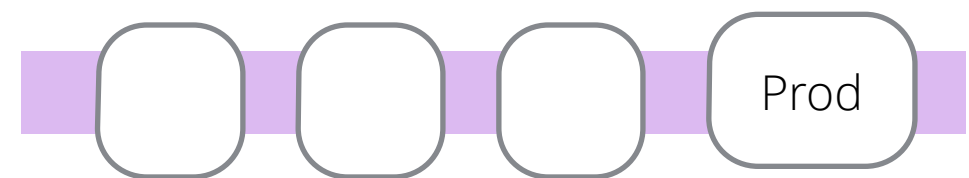


```

26
27 def content(): Node = {
28   <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
29     <head>
30       <link rel="stylesheet" href="/static/common.css" type="text/css" />
31       <link rel="stylesheet" href="/static/display.css" type="text/css" />
32       <meta http-equiv="refresh" content="30" />
33     </head>
34     <body>
35       { content(build) }
36     </body>
37   </html>
38 }
39
40 private def content(build): List[Node] = {
41   displayType match {
42     case "single" => <div { build.map(build => asTable(build)) } </div>
43     case "short" => {
44       if (build.length == 1) {
45         <div { build.map(build => asTable(build)) } </div>
46       } else {
47         <div class="build">{ build.map(build => buildFormat(build)) }</div>
48       }
49     }
50     case _ => <div class="build">{ build.map(build => buildFormat(build)) }</div>
51   }
52 }
53
54 private def asTable(build: Node): List[Node] = {
55   <table class="build"> { build.getFormatName.toString }
56   <tr valign="middle" align="center">
57     <td { linkToBuild(build) }</td>
58   </tr>
59   </table>
60 }
61
  
```

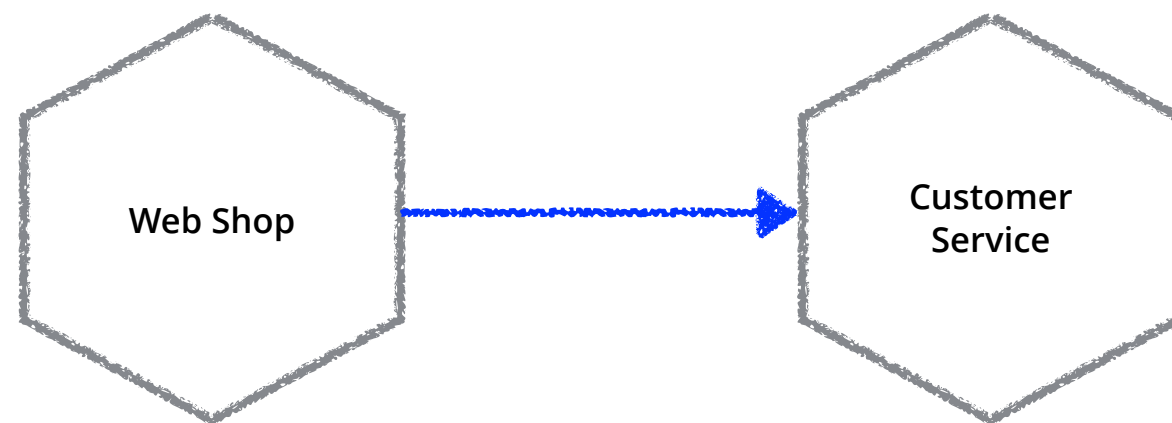


Expectations



```

26
27 def content(): Node = {
28   <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
29     <head>
30       <link rel="stylesheet" href="/static/common.css" type="text/css" />
31       <link rel="stylesheet" href="/static/display.css" type="text/css" />
32       <meta http-equiv="refresh" content="30" />
33     </head>
34     <body>
35       { content(build) }
36     </body>
37   </html>
38 }
39
40 private def content(build: List[Build]): Elem = {
41   displayType match {
42     case "single" => <div { build.map(build => asTable(build)) } </div>
43     case "short" => {
44       if (build.length == 1) {
45         <div { build.map(build => asTable(build)) } </div>
46       } else {
47         <div class="build">{ build.map(build => buildFormat(build)) }</div>
48       }
49     }
50     case _ => <div class="build">{ build.map(build => buildFormat(build)) }</div>
51   }
52 }
53
54 private def asTable(build: Build): Elem = {
55   <table class="build"> { build.getFormatName.shortCase }
56   <tr valign="middle" align="center">
57     <td { linkToBuild(build) }</td>
58   </tr>
59   </table>
60 }
61
  
```



Expectations



```

26 def content(): Node = {
27   <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
28     <head>
29       <link rel="stylesheet" href="/static/common.css" type="text/css" />
30       <link rel="stylesheet" href="/static/display.css" type="text/css" />
31       <meta http-equiv="refresh" content="30" />
32     </head>
33     <body>
34       { content(build) }
35     </body>
36   </html>
37 }
38
39 private def content(build): List[Node] = {
40   displayType match {
41     case "single" => <div { build.map(build => asTable(build)) } </div>
42     case "short" => {
43       if (build.length == 1) {
44         <div { build.map(build => asTable(build)) } </div>
45       } else {
46         <div class="build">{ build.map(build => buildFormat(build)) }</div>
47       }
48     }
49   }
50   case _ => <div class="build">{ build.map(build => buildFormat(build)) }</div>
51 }
52
53 private def asTable(build: Build): List[Node] = {
54   <table class="build" + build.getVendorName.vendorCase >
55     <tr align="middle" align="center">
56       <td { linkToBuild(build) }</td>
57     </tr>
58   </table>
59 }
60 }
  
```

# 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:  build passing

# Pact

---


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

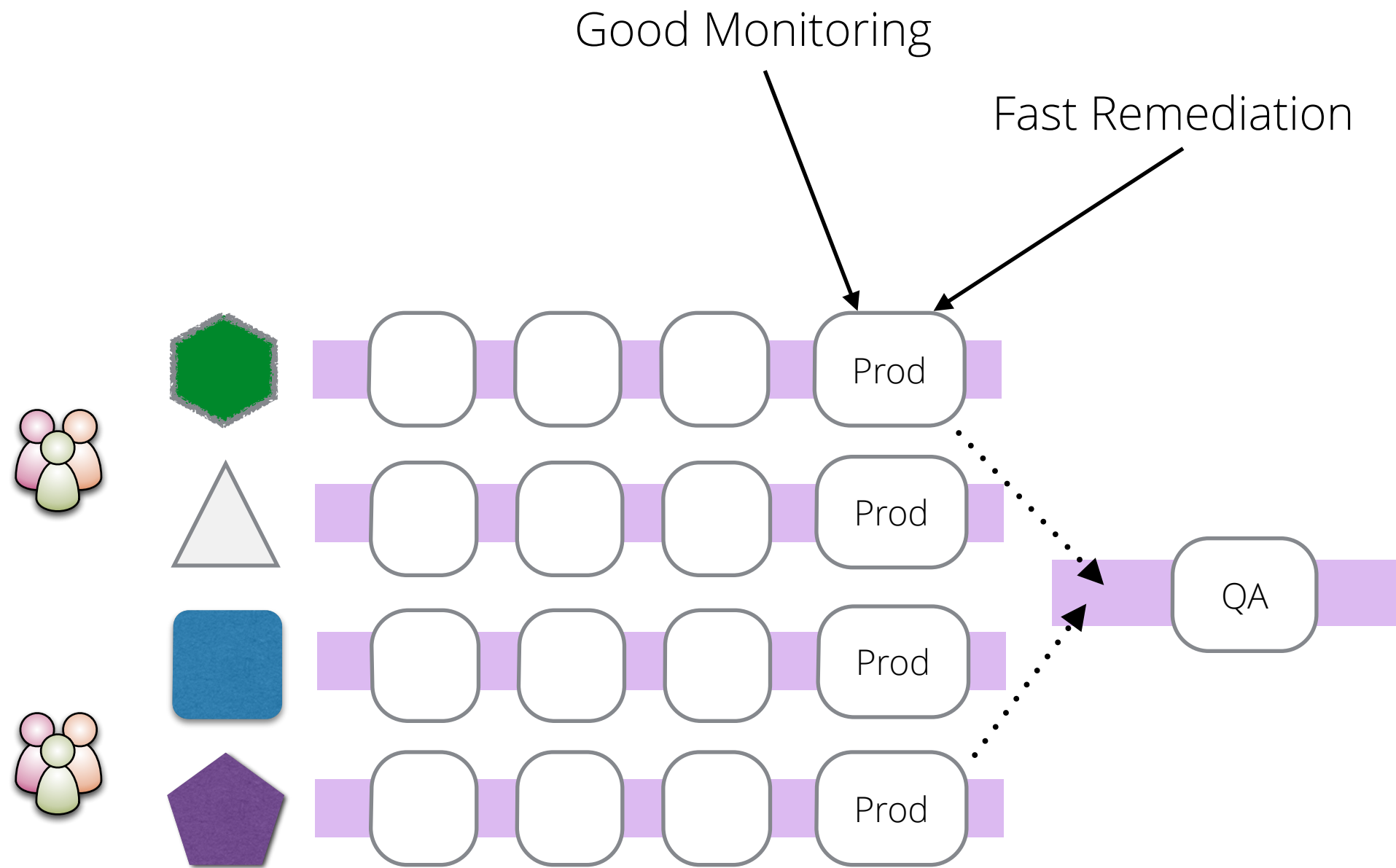
Pact provides an RSpec DSL for service consumers to define the HTTP requests they will make to a service provider and the HTTP responses they expect back. These expectations are used in the consumers specs to provide to ensure that the provider is conforming to the contract.

<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?

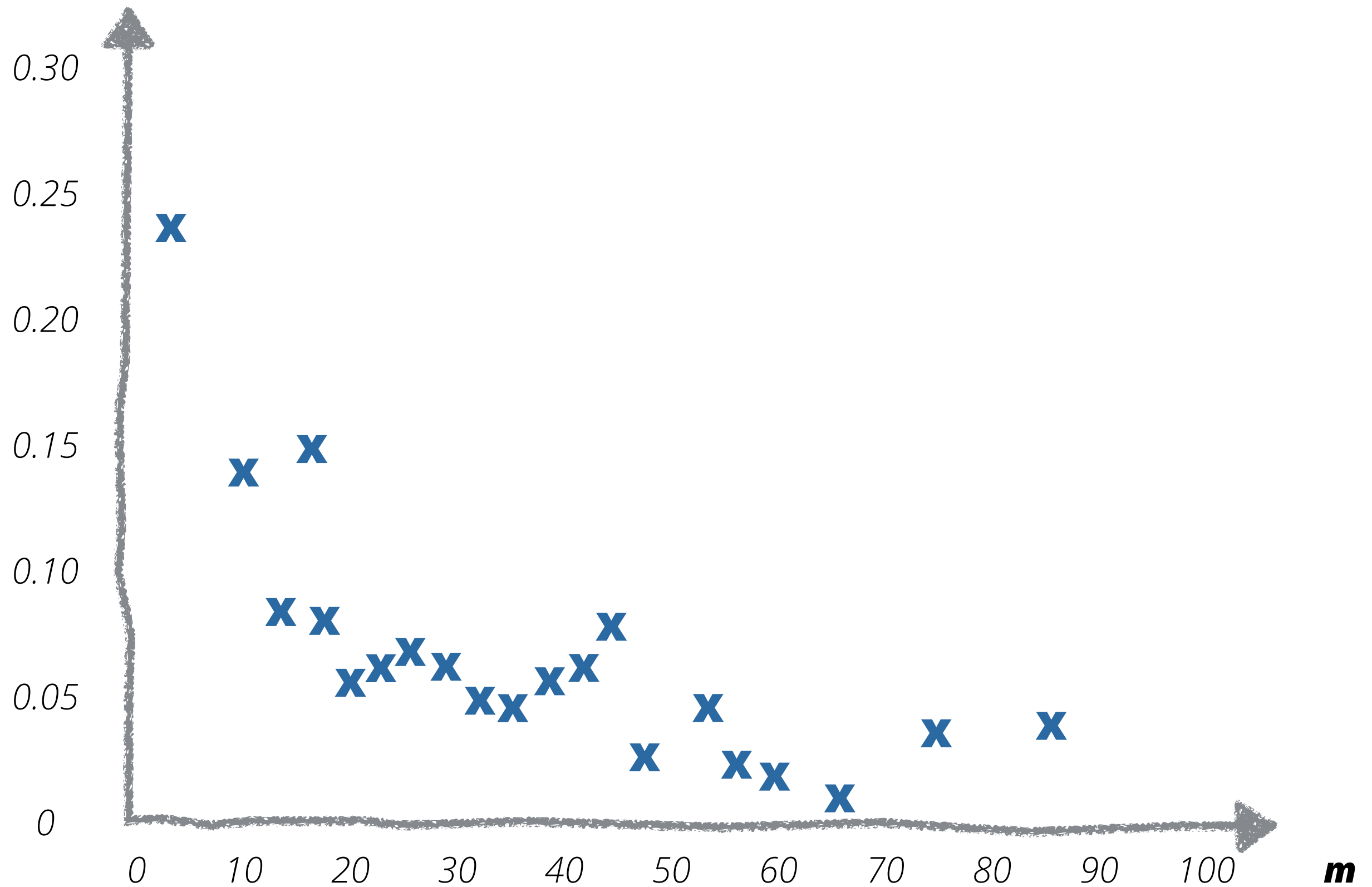
business capability?

organisation?

**Thomas J. Allen, 1977**

**Probability of  
weekly interaction**

*The effect of distance on communication*





# inter-company integration

*Low change rate*

*High stability*

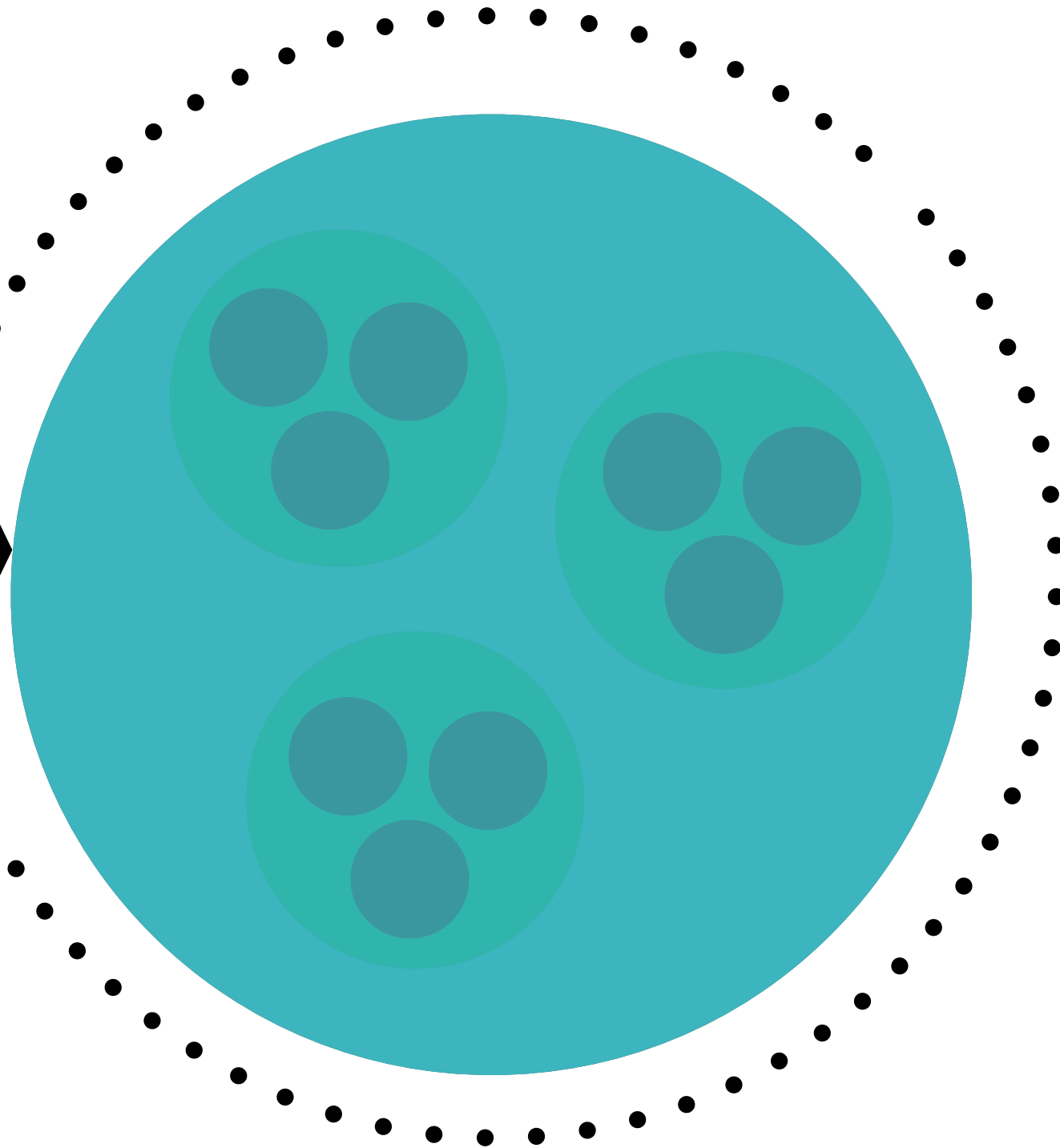


**Semantic Versioning**

*Contract Testing*

*Tolerant Reader*

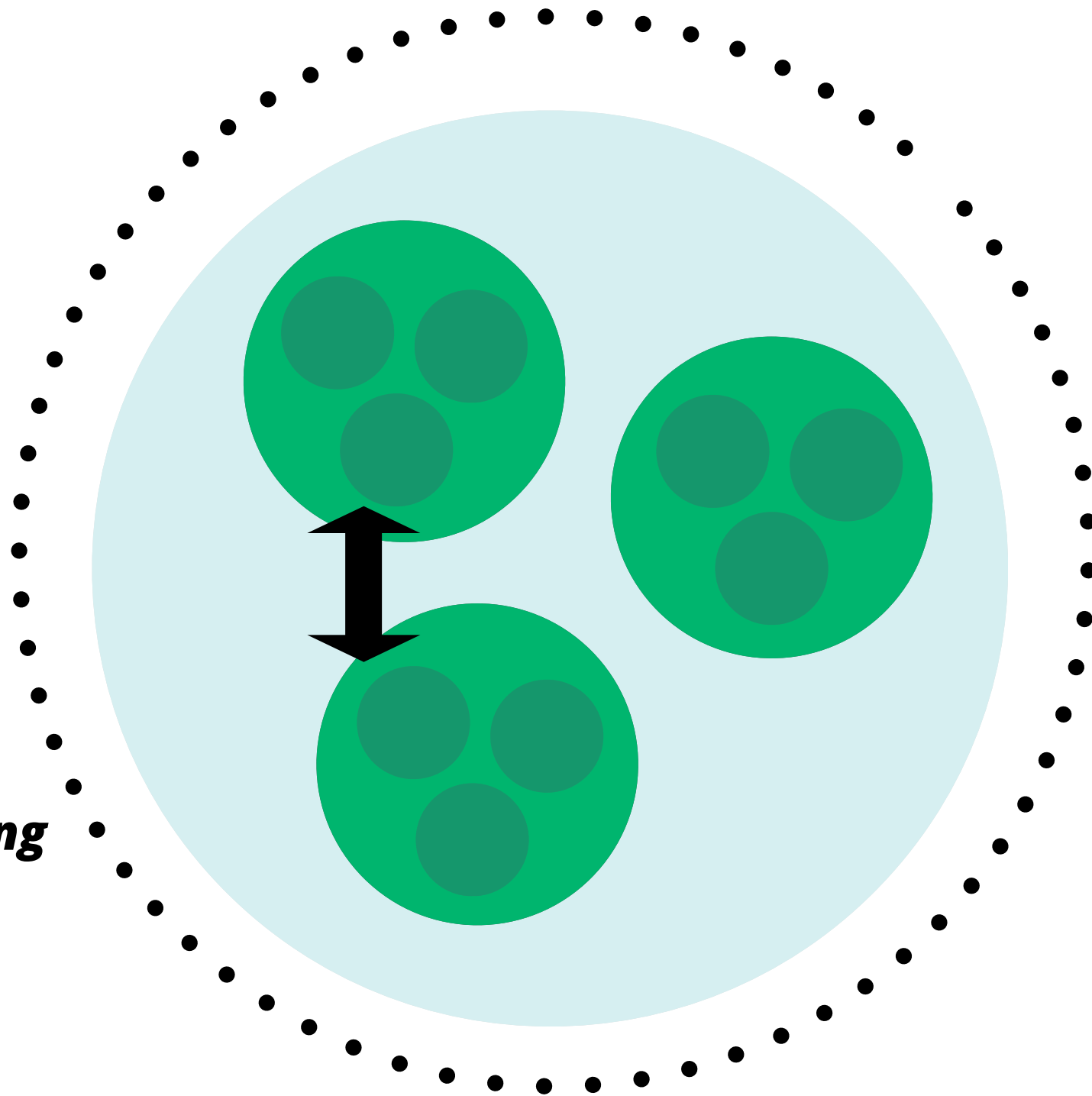
*“Conversational change”*



# inter-team integration

*higher change rate*

*lower stability*



***Semantic Versioning***

***Contract Testing***

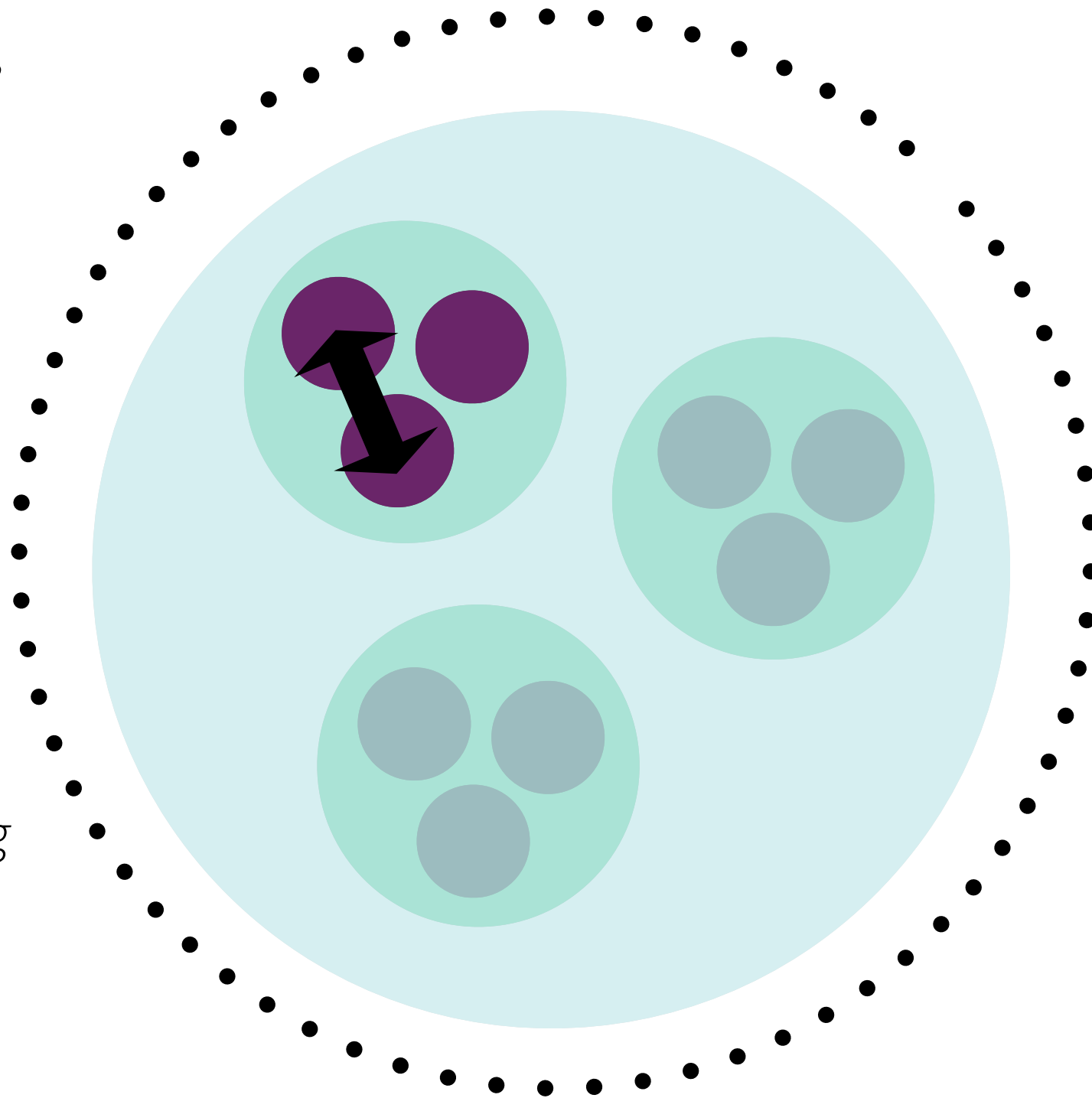
***Tolerant Reader***

*“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**



***we have **old** techniques **that** apply:***

**SRP**

**GRASP**

**YAGNI**

**KISS**

**TDD**

**DRY**

*and new techniques to apply:*

**Consumer Driven Contracts**

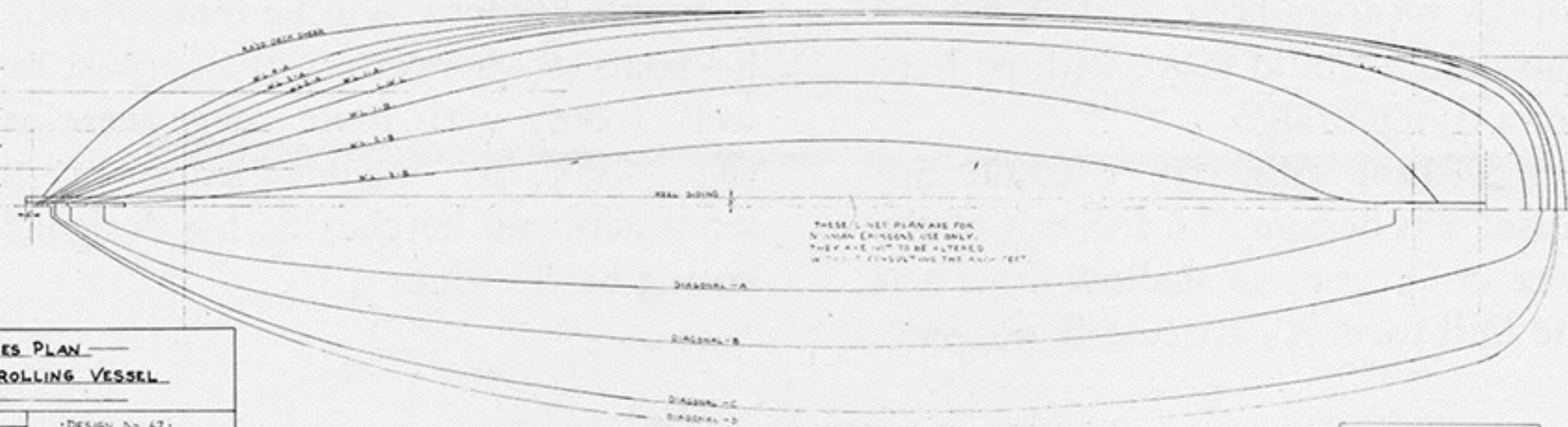
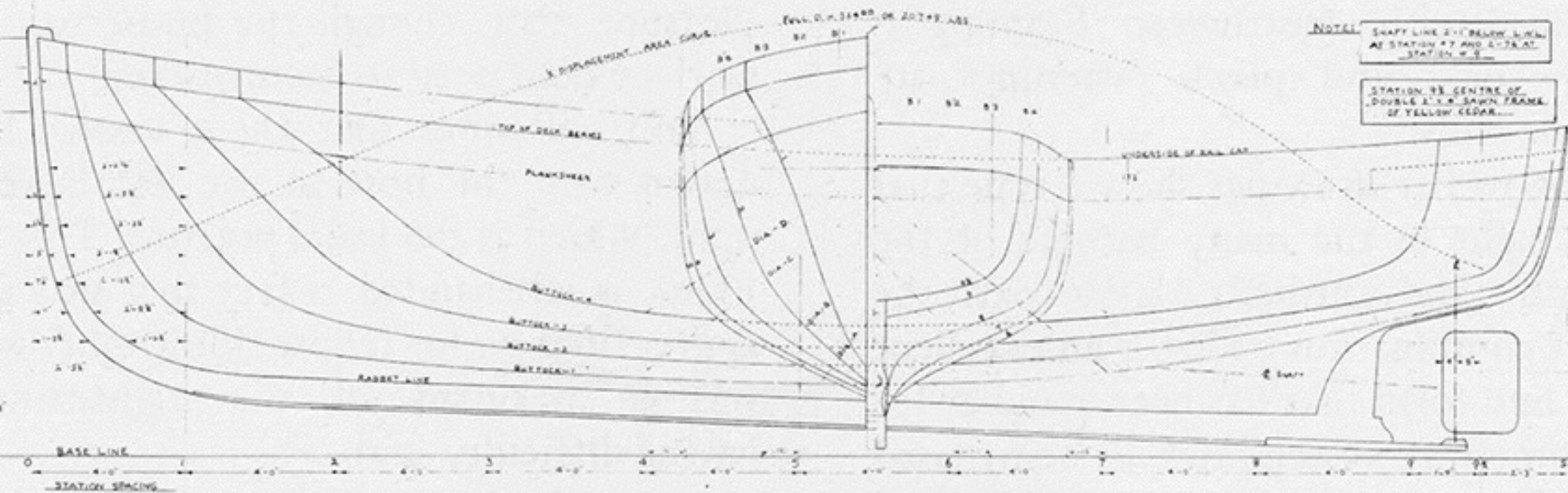
**Semantic Monitoring**

**Semantic Versioning**

**Testing in Production**

**Failure Isolation**





# ES PLAN ROLLING VESSEL

DESIGN No. 67  
F.E. FREDETTE, NAVAL ARCHT.  
31 HITCHMAN ST. VICTORIA B.C.  
DATE - SEPT. 30<sup>TH</sup> 1962

NOTE: ALL LINES DRAWN TO OUTSIDE OF PLANKING. BASE LINE 4'-8" BELOW L.W.L. STATIONS, WATER LINES, BUTTOCKS & DIAGONALS SPACED AS INDICATED BY PLAN.





**jalewis@thoughtworks.com**  
**@boicy**

**ThoughtWorks®**

# Thanks!

**[jalewis@thoughtworks.com](mailto:jalewis@thoughtworks.com)**

**@boicy**

**ThoughtWorks®**