coding {the} architecture

Modular Monoliths

Simon Brown
@simonbrown

Big design up front

vs

no design up front

Working software

over

comprehensive documentation

#NoEstimates

#NoProjects

Monoliths

vs

microservices

Monolithic architecture

Service-based architecture

(SOA, microservices, etc)

The hidden assumption is that all

monoliths are big balls of mud

Simon Brown
@simonbrown

I'll keep saying this ... if people can't build monoliths properly, microservices won't help. #qconlondon #DesignThinking #Modularity

RETWEETS 244 LIKES 100

10:49 AM - 4 Mar 2015

Architect Clippy
@architectclippy

I see you have a poorly structured monolith. Would you like me to convert it into a poorly structured set of microservices?

RETWEETS 4,031 LIKES 2,345

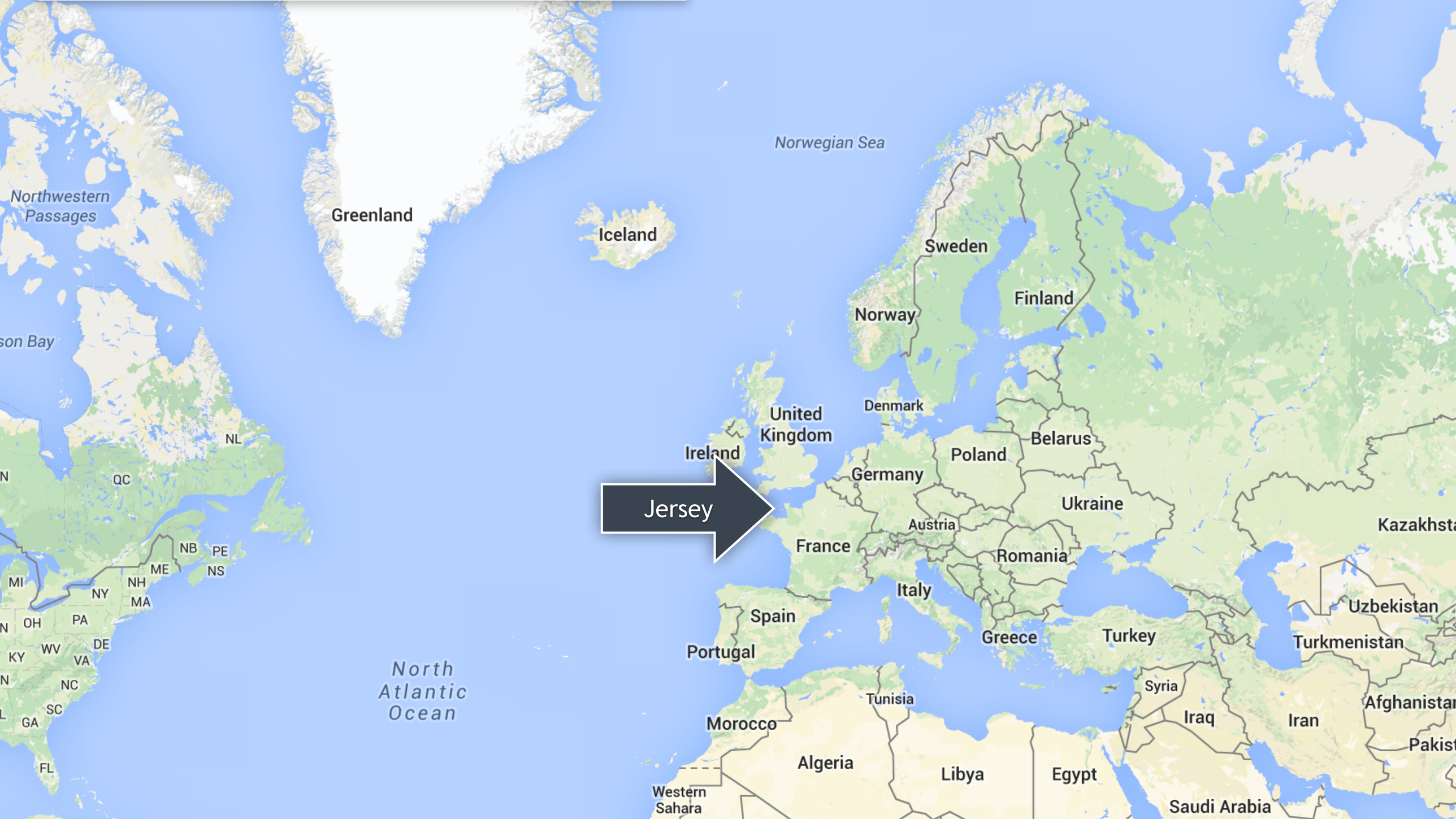
12:59 AM - 24 Feb 2015



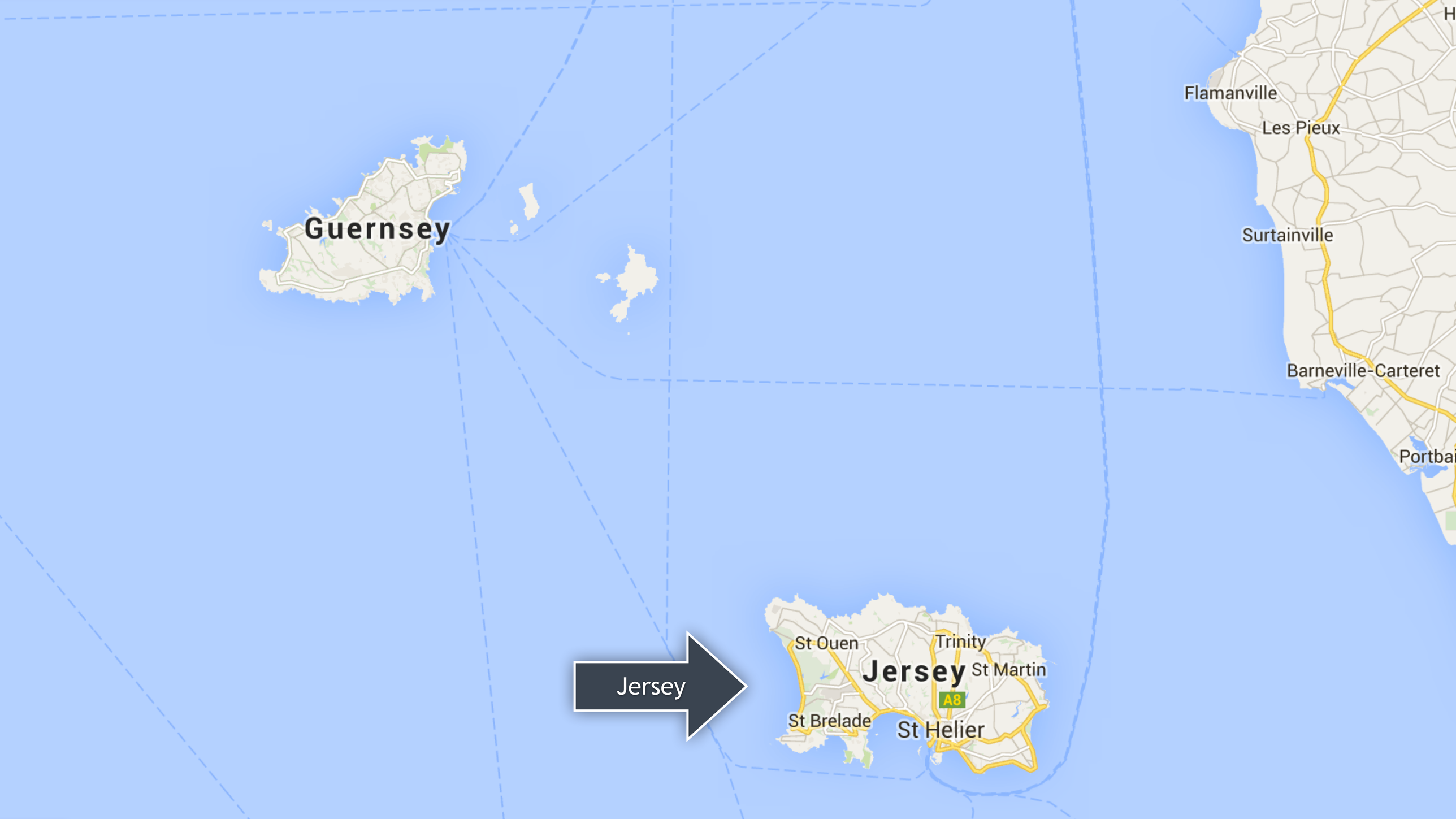

Monolithic vs Microservices

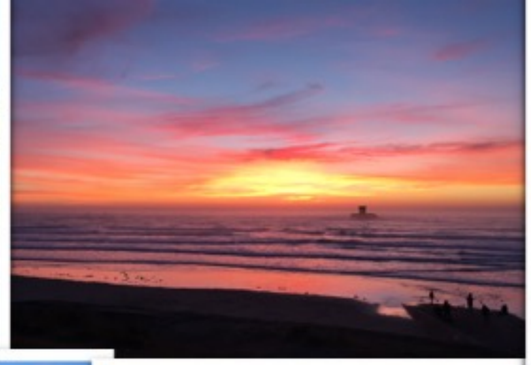
Monolithic

Microservices


Jersey
Norwegian Sea
Northwestern Passages
Greenland
Iceland
Sweden
Finland
Norway
Denmark
United Kingdom
Ireland
Belarus
Poland
Germany
Ukraine
Austria
France
Romania
Italy
Spain
Portugal
Greece
Turkey
Uzbekistan
Turkmenistan
Kazakhsta
Syria
Iraq
Iran
Afghanistan
Pakist
Tunisia
Morocco
Algeria
Libya
Egypt
Western Sahara
Saudi Arabia
North Atlantic Ocean
NL
QC
NB
PE
NS
ME
NH
MA
NY
MI
OH
PA
WV
VA
DE
KY
NC
SC
GA
FL


Jersey
Guernsey
Jersey
English Channel
England
Wales
Netherlands
Belgium
Wicklow
Carlow
Kilkenny
Gorey
Enniscorthy
Wexford
Waterford
Derby
Boston
Cromer
King's Lynn
North Walsham
Den Helder
Shrewsbury
Leicester
Peterborough
Dereham
Norwich
Hoorn
Alkmaar
Lelystad
Zwolle
Newtown
Aberystwyth
Kidderminster
Corby
Attleborough
Lowestoft
Beccles
Amsterdam
Worcester
Cambridge
Leiden
Hereford
Bedford
Ipswich
Rotterdam
Cheltenham
Luton
Colchester
Felixstowe
Arnhem
Haverfordwest
Carmarthen
Gloucester
Chelmsford
Dordrecht
Llanelli
St Albans
Clacton-on-Sea
Nijmegen
Milford Haven
Neath
Newport
Swindon
Oxford
Basildon
Tilburg
Swansea
Pontypridd
Bristol
Reading
London
Southend-on-Sea
Vlissingen
Breda
Barry
Bath
Croydon
Maidstone
Margate
Eindhoven
Bruges
Antwerp
Barnstaple
Taunton
Crawley
Deal
Dunkirk
Ostend
Ghent
Lier
Geel
Mönchengladbach
Bideford
Yeovil
Southampton
Ashford
Dover
Calais
Genk
Brighton
Hastings
Kortrijk
Brussels
Maastricht
Aachen
Exeter
Portsmouth
Boulogne-sur-Mer
Weymouth
Eastbourne
Liège
Tavistock
Swanage
Shanklin
Lille
Tournai
Verviers
Torquay
Béthune
Newquay
Plymouth
Arras
Douai
Mons
Truro
St. Austell
Dartmouth
St. Ives
Cambrai
Penzance
Falmouth
Abbeville
Dieppe
Amiens
Saint-Quentin
Fécamp
Charleville-Mézières
Cherbourg
Arlon
Sedan
Laon
Le Havre
Bricquebec
Rouen
Beauvais
Virton
Luxemburg
Deauville
Thionville
Elbeuf
Soissons
Reims
Caen
Lisieux
Louviers
Verdun
Saint-Lô
Metz
Évreux
Cergy
Épernay
Coutances
Paris
Pont-à-Mousson
Granville
Vire
Châlons-en-Champagne
Lannion
Flers
Dreux
Boulogne-Billancourt
Bar-le-Duc
Saint-Pol-de-Léon
St-Malo
Morlaix
Saint-Brieuc
Évry
Melun
Saint-Dizier
Dinan
Brest
Landerneau
Lamballe
Fougères
Alençon
Chartres
Mayenne
Troyes
Plougastel-Daoulas
Parc Naturel
Marin d'Iroise
Sens
Rennes
Vitré
Chaumont
M1
M54
M6 Toll
M42
M40
M11
M4
M25
M3
M20
E22
E231
E35
E312
E40
E313
E42
E19
E15
E411
E25
E17
E402
E44
E46
E401
E3
E50
E5
E54


Guernsey
Flamanville
Les Pieux
Surtainville
Barneville-Carteret
Portbail
Jersey
St Ouen
Trinity
Jersey
St Martin
A8
St Brelade
St Helier











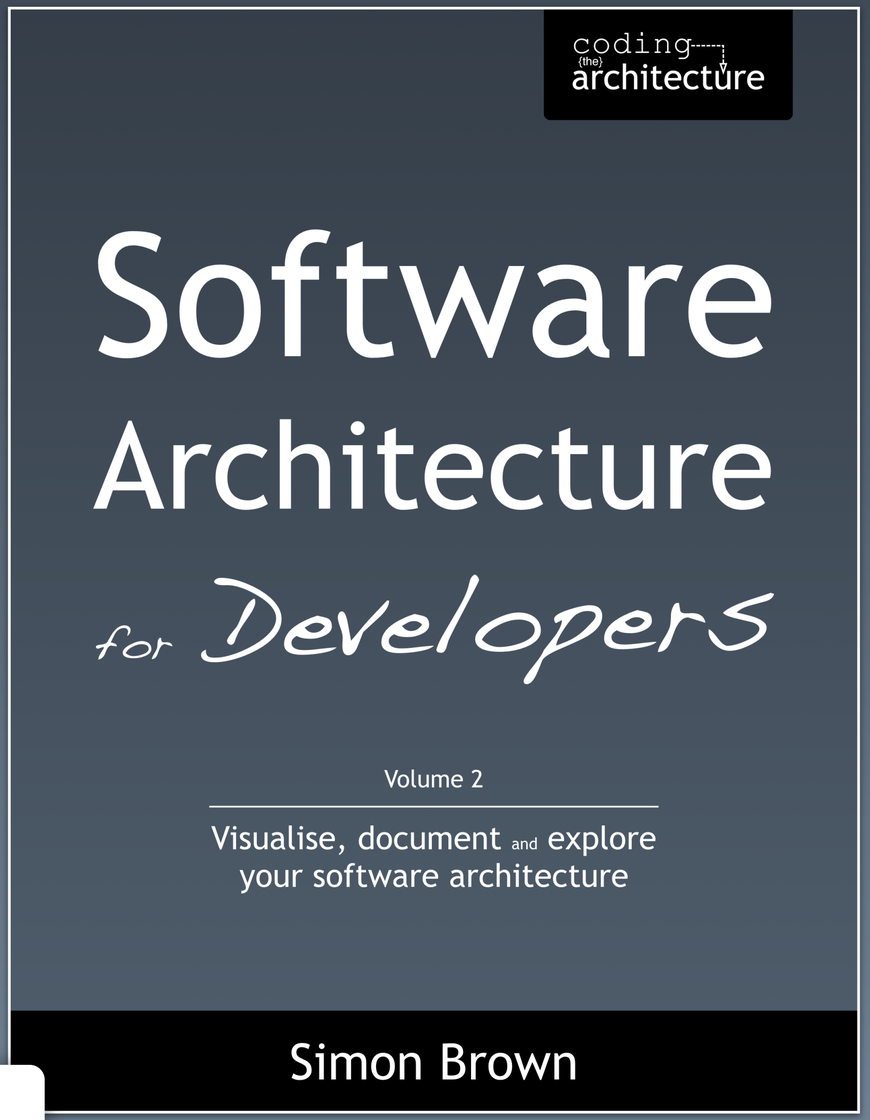

I help software teams understand

software architecture, technical leadership and the balance with agility

Software architecture
needs to be more

accessible

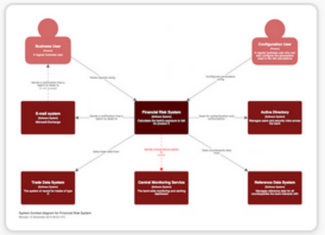

coding
{the}
architecture
Software
Architecture
for Developers
Volume 1
Technical leadership by coding, coaching,
collaboration and just enough up front design
Simon Brown

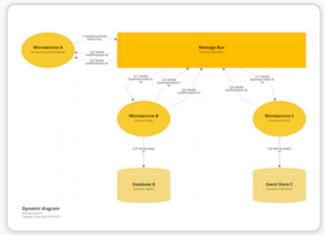

coding
{the}
architecture
Software
Architecture
for Developers
Volume 2
Visualise, document and explore
your software architecture
Simon Brown

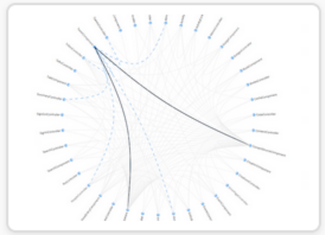
Leanpub

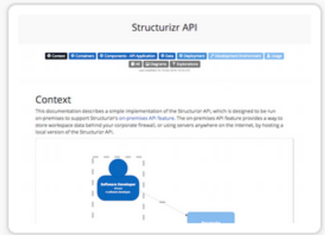
Structurizr

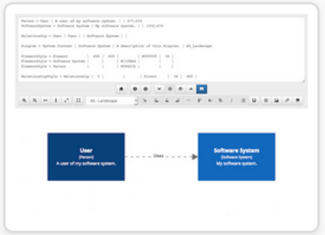
Visualise, document and explore your software architecture

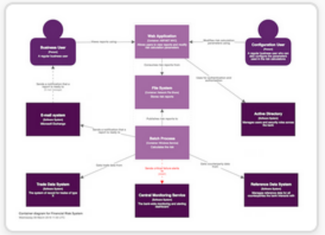


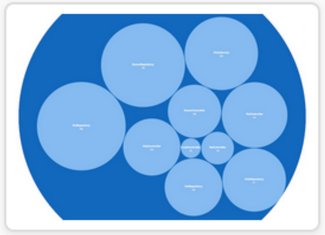


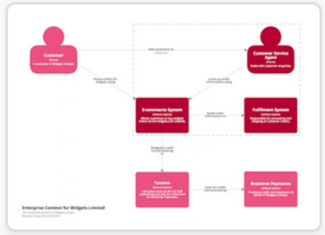


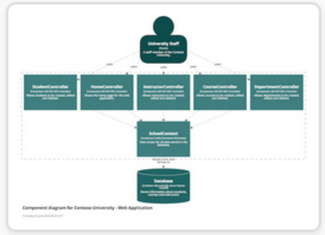


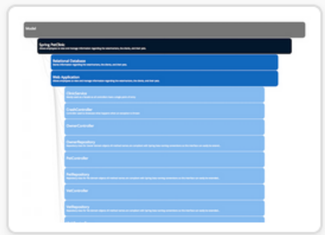


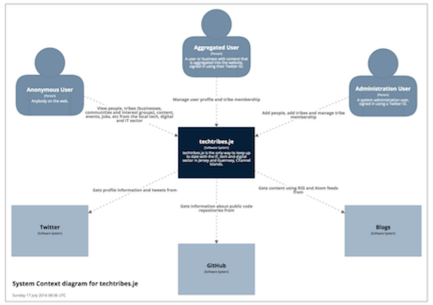


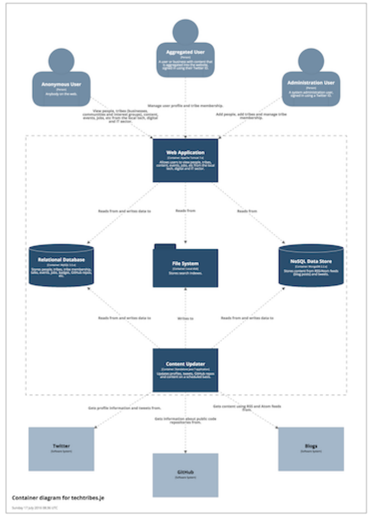


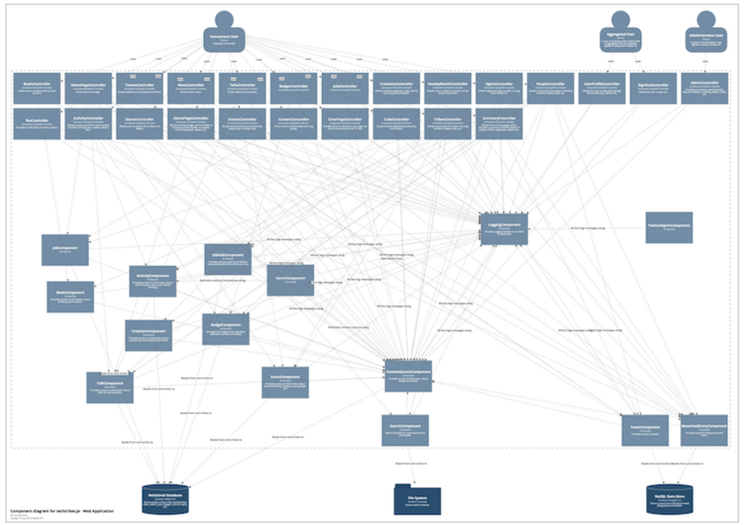


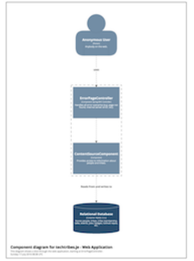


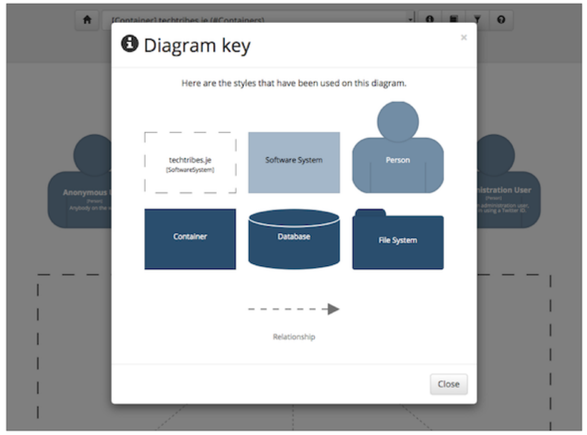


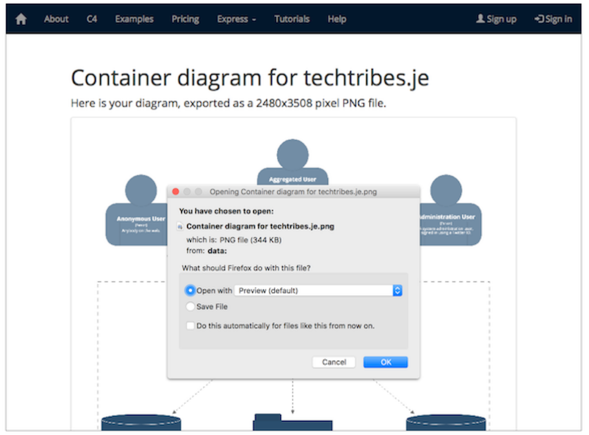




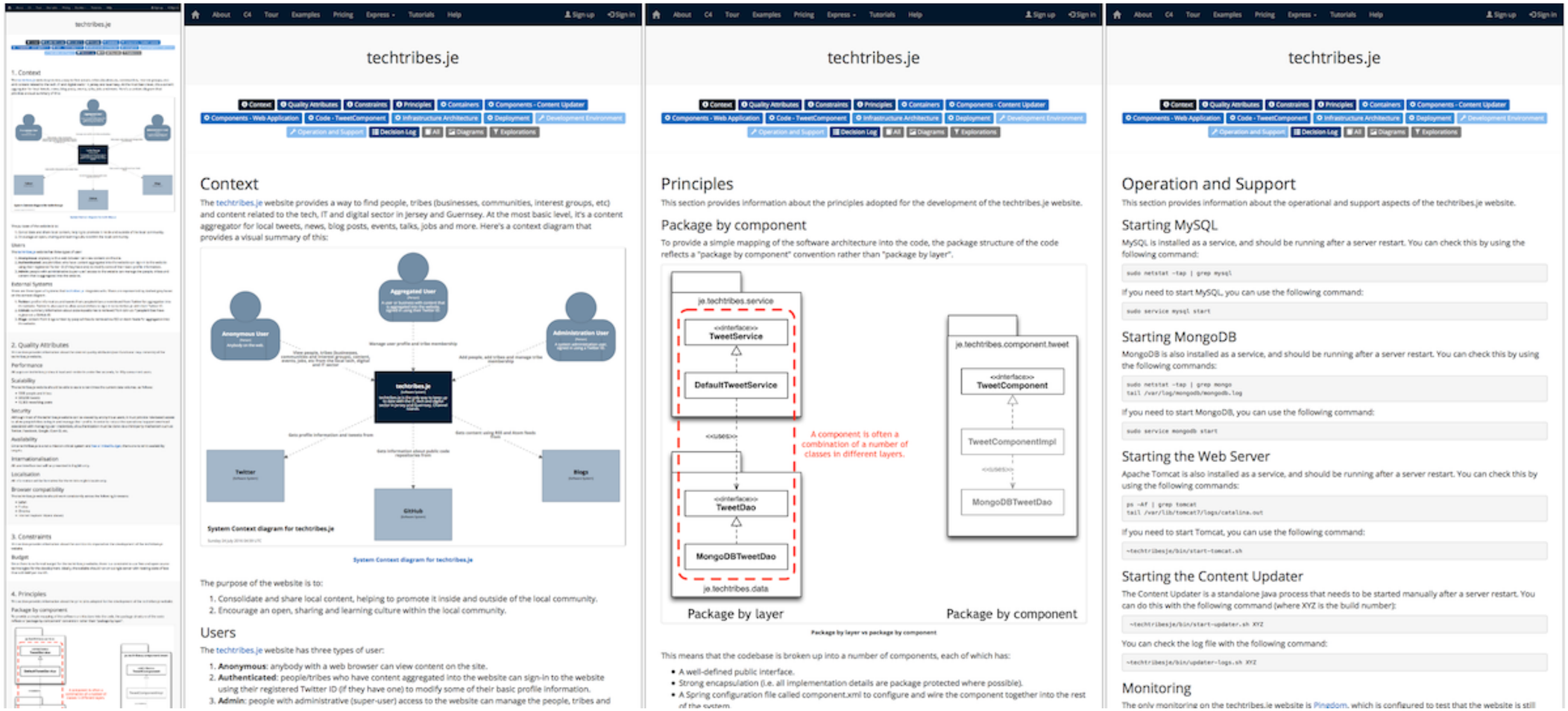




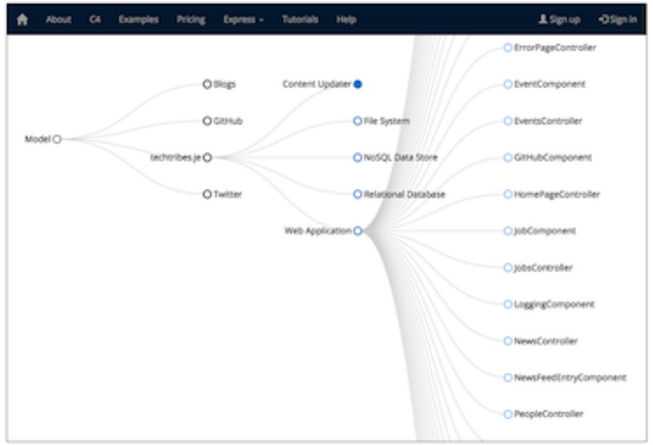


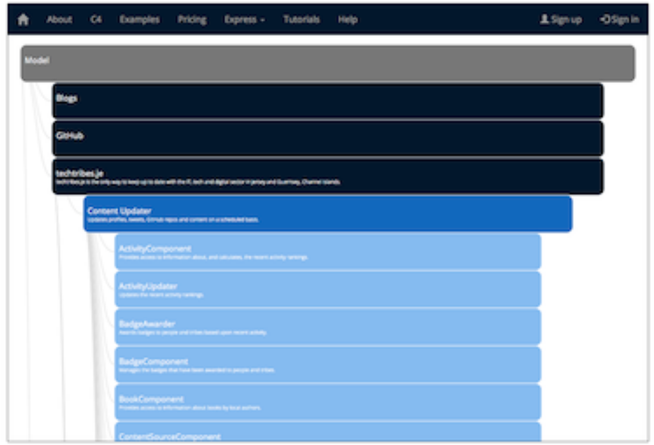


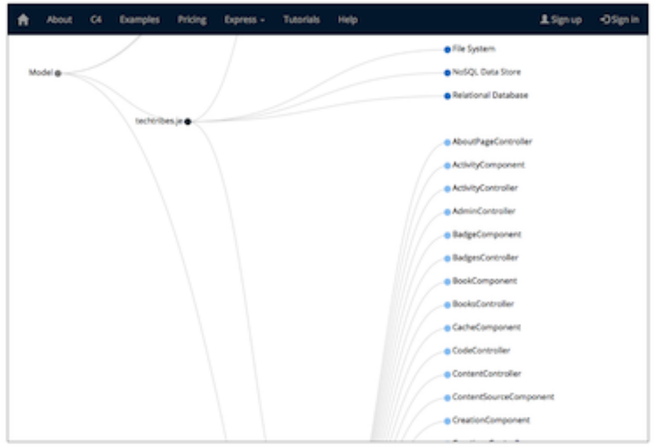
Visualise, document and explore your software architecture



Visualise, **document** and explore your software architecture

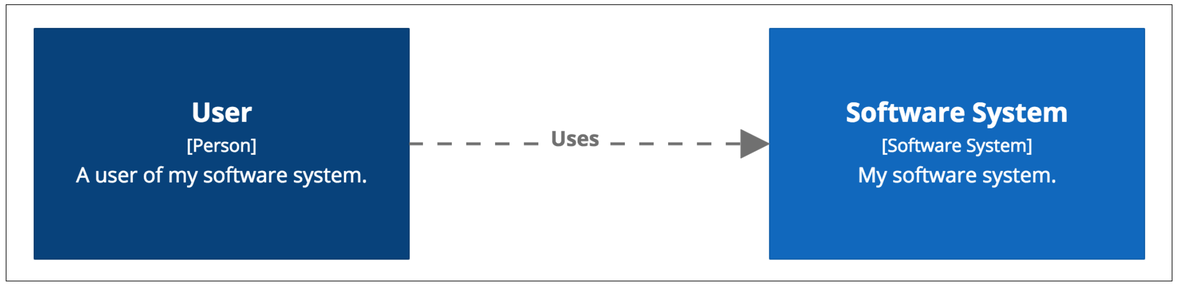


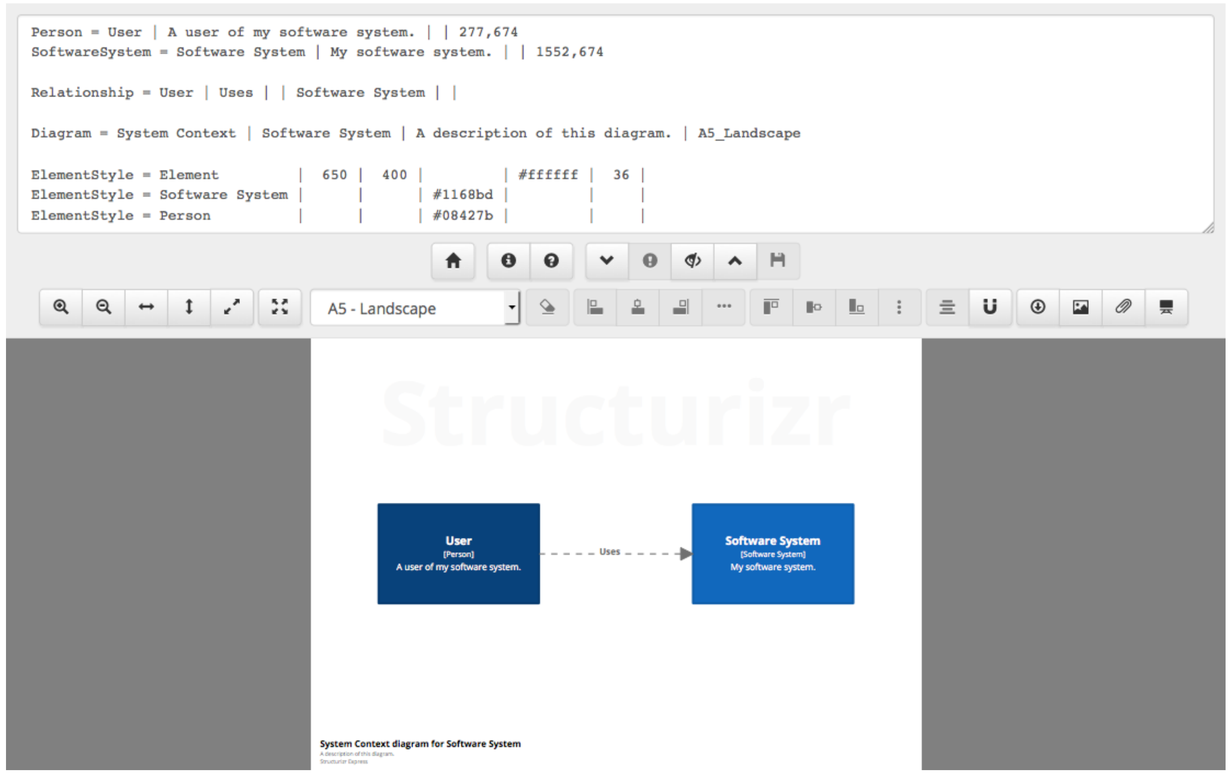




Visualise, document and **explore** your software architecture

```
Workspace workspace = new Workspace("My model", "This is a model of my software system.");
Model model = workspace.getModel();

Person user = model.addPerson("User", "A user of my software system.");
SoftwareSystem softwareSystem = model.addSoftwareSystem("Software System", "My software system.");
user.uses(softwareSystem, "Uses");

ViewSet viewSet = workspace.getViews();
SystemContextView contextView = viewSet.createSystemContextView(softwareSystem, "context", "A simple example...");
contextView.addAllSoftwareSystems();
contextView.addAllPeople();

Styles styles = viewSet.getConfiguration().getStyles();
styles.addElementStyle(Tags.SOFTWARE_SYSTEM).background("#1168bd").color("#ffffff");
styles.addElementStyle(Tags.PERSON).background("#08427b").color("#ffffff");

StructurizrClient structurizrClient = new StructurizrClient("key", "secret");
structurizrClient.putWorkspace(1234, workspace);
```





Software architecture diagrams as **code** or **text**

A well structured codebase is easy to visualise

Abstraction

is about reducing detail

rather than creating a different representation

Abstractions help us

reason about

a big and/or complex

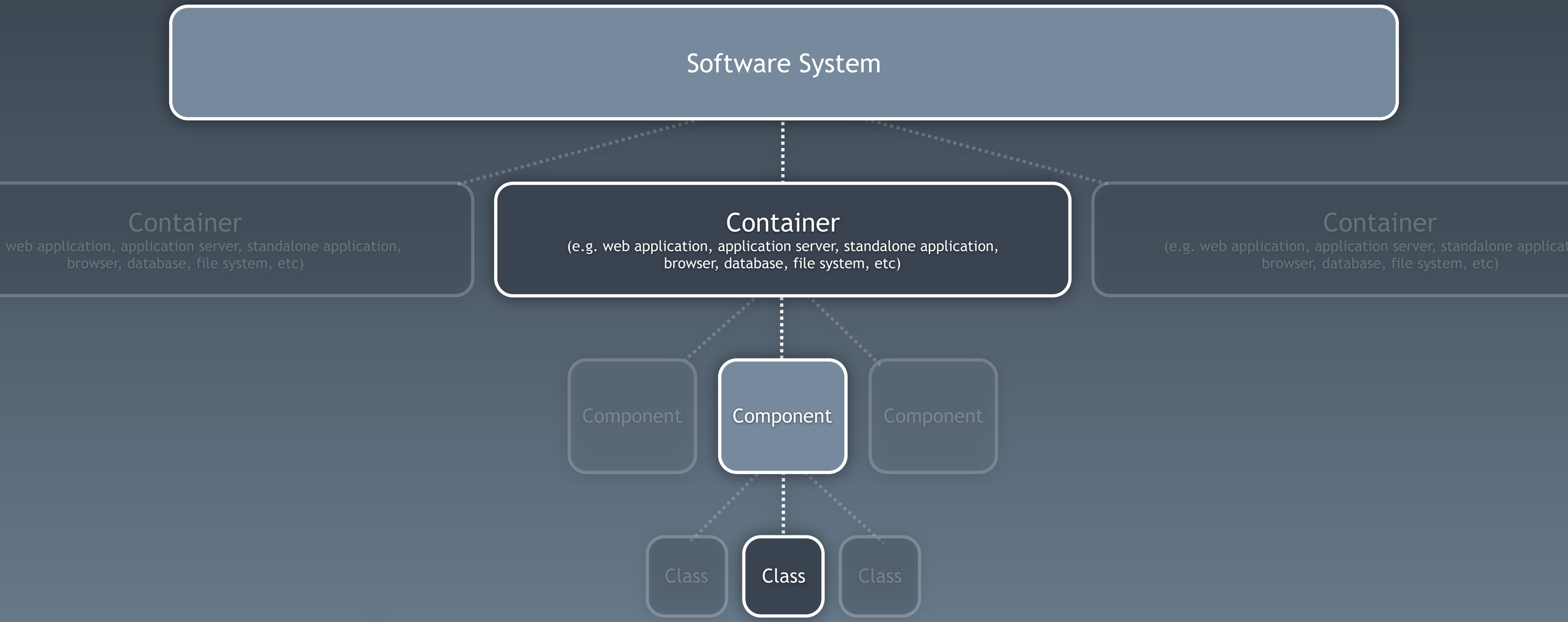
software system

Abstractions

should reflect the

code

As an industry, we lack a common vocabulary with which to think about, describe and communicate software architecture



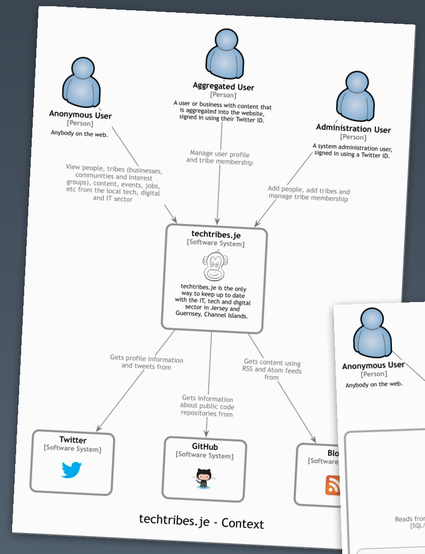
A **software system** is made up of one or more **containers**,

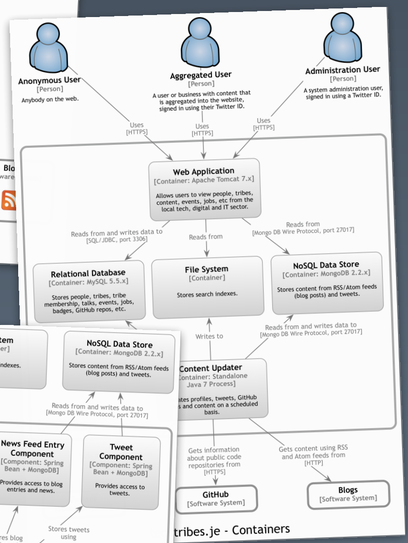
each of which contains one or more **components**,

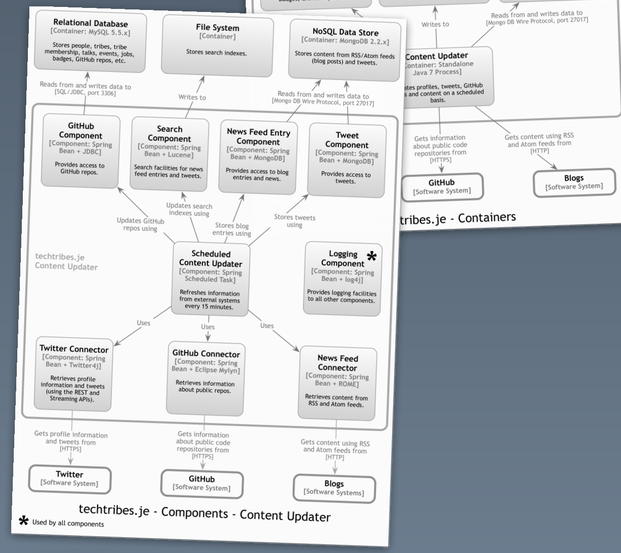
which in turn are implemented by one or more **classes**.

C4

Context, Containers, Components and Classes (or Code)

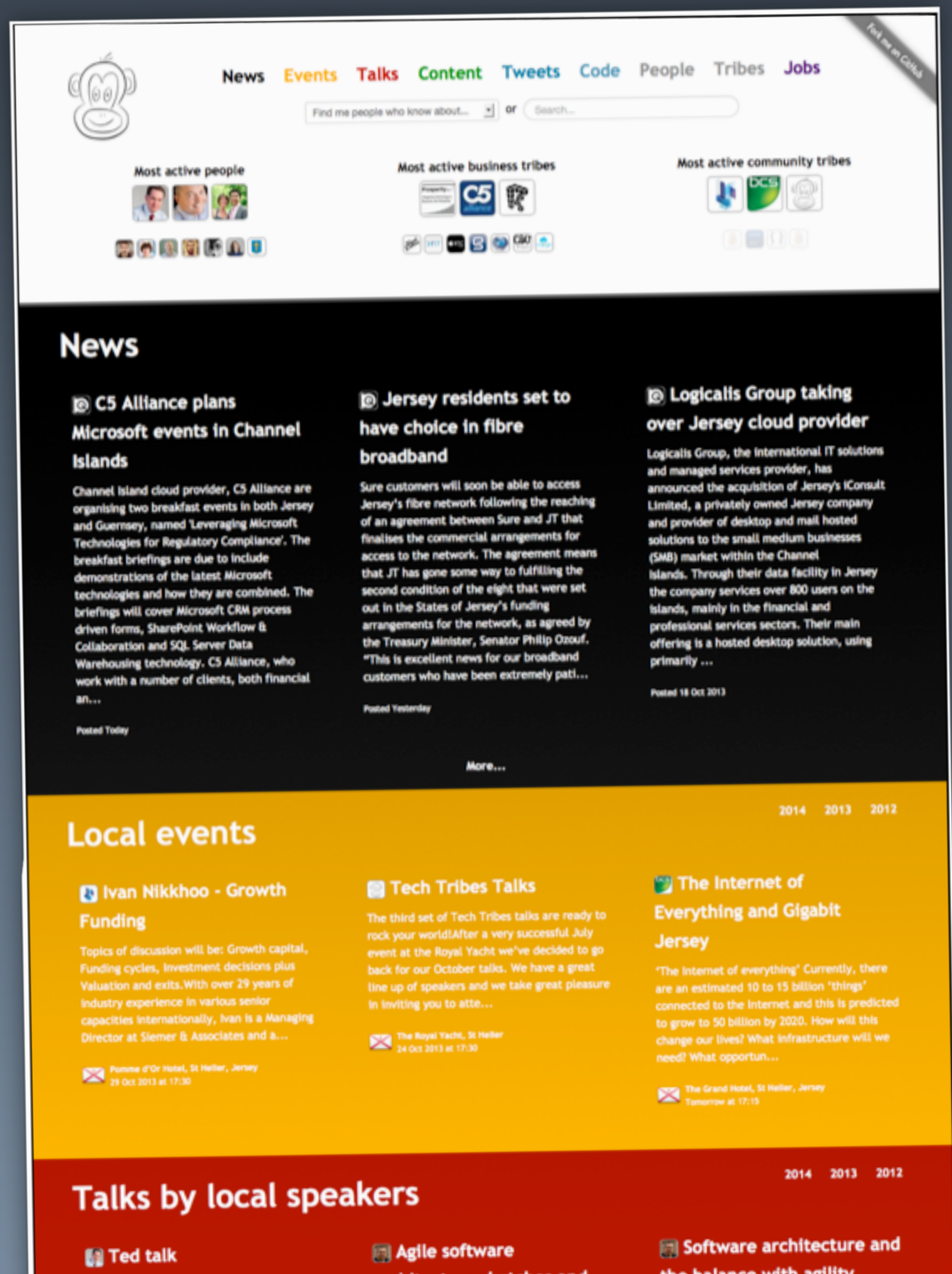




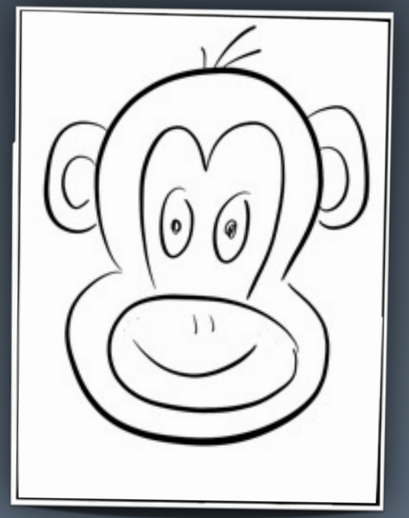


Diagrams are maps

that help a team navigate a complex codebase



techtribes.je

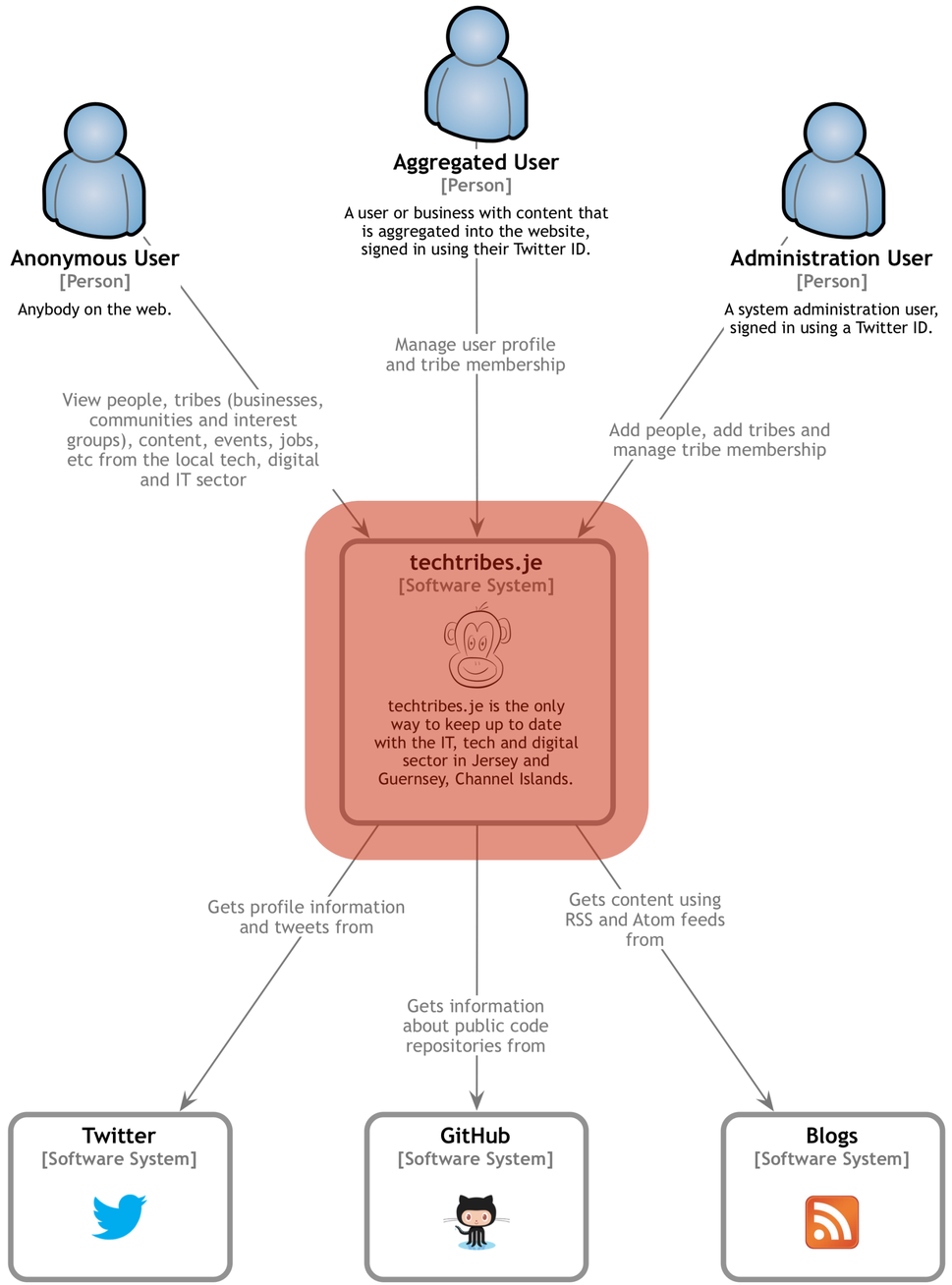


Context diagram
(level 1)

Container diagram
(level 2)

Component diagram
(level 3)

Class diagram
(level 4)



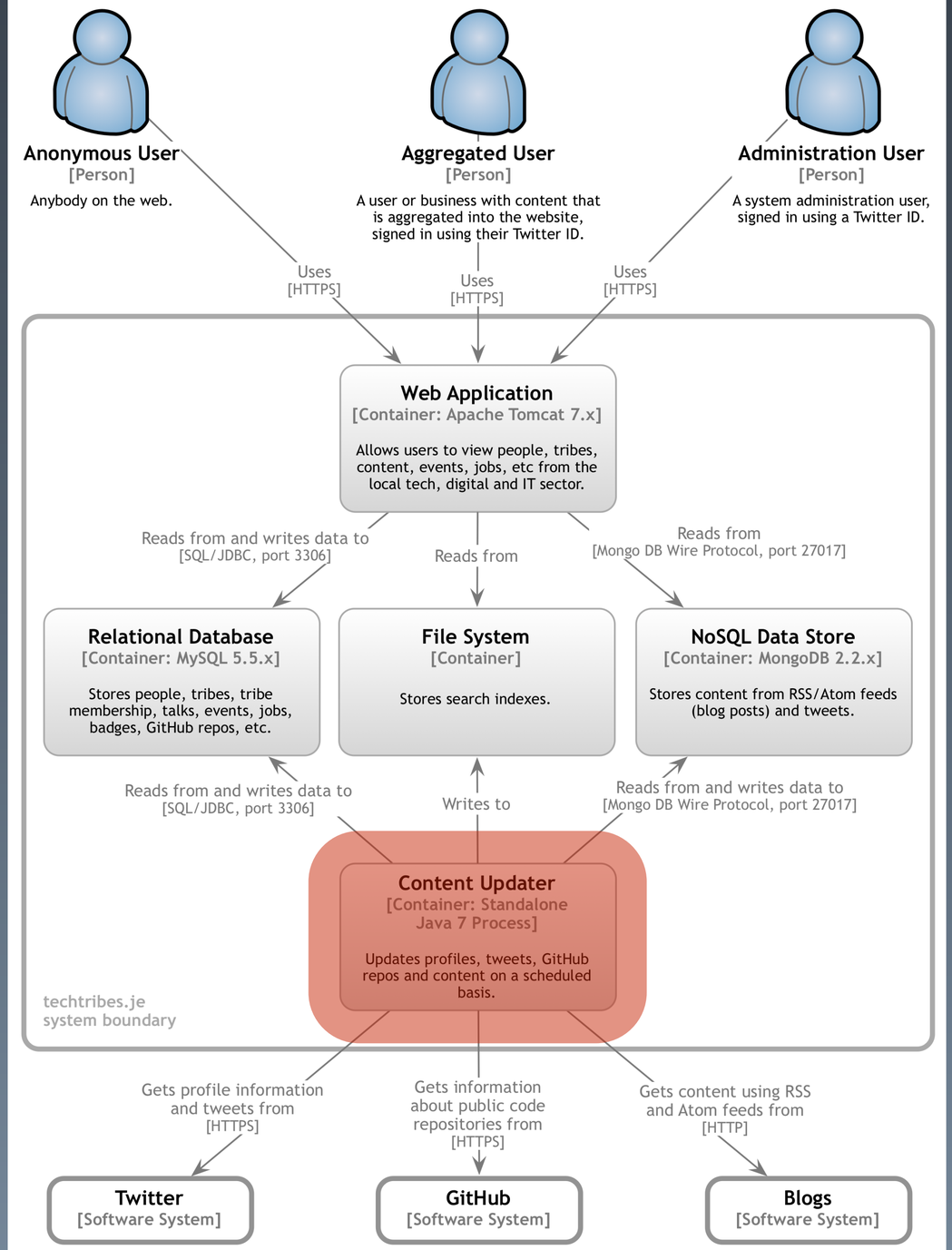
techtribes.je - Context

Context diagram
(level 1)

Container diagram
(level 2)

Component diagram
(level 3)

Class diagram
(level 4)



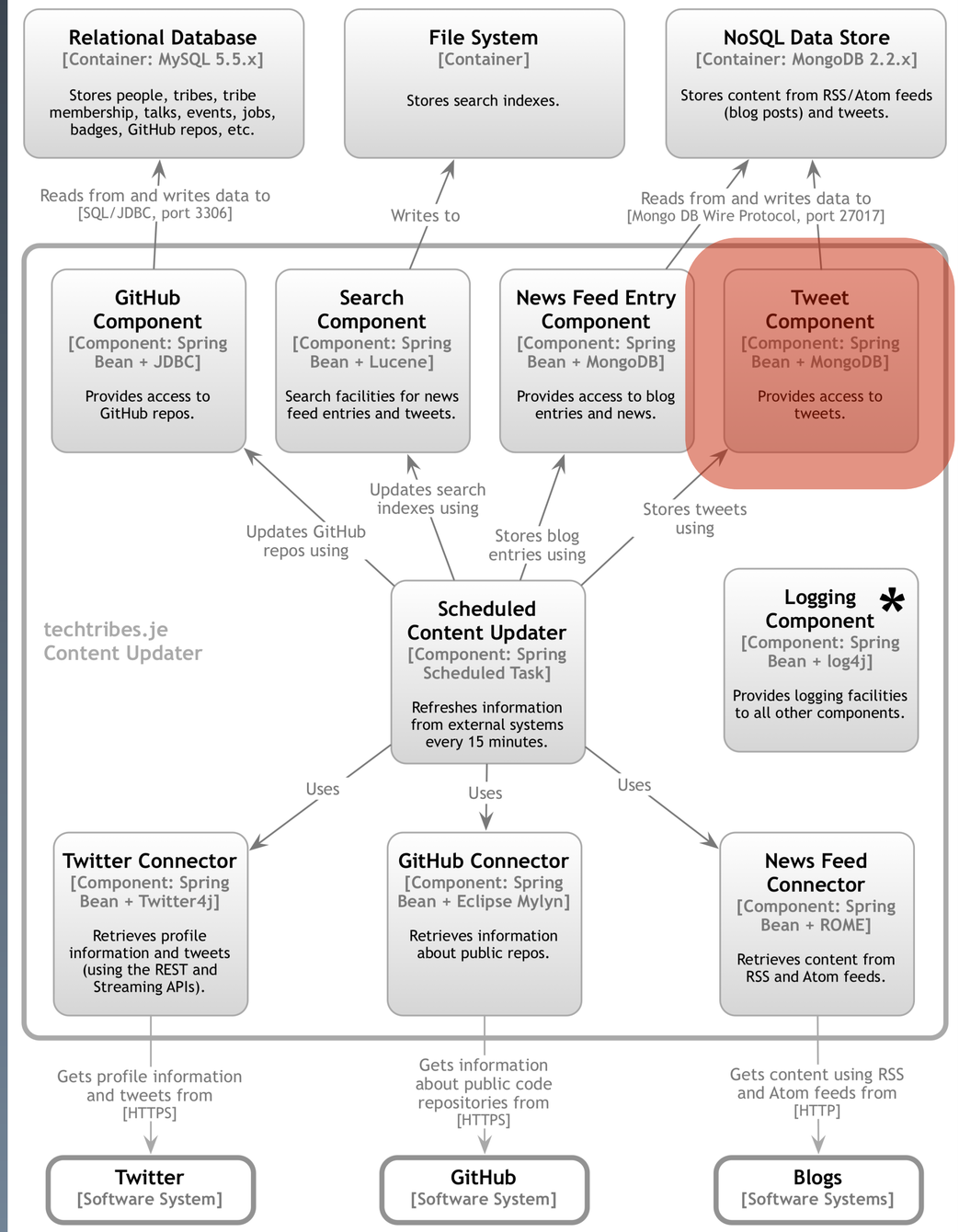
techtribes.je - Containers

Context diagram
(level 1)

Container diagram
(level 2)

Component diagram
(level 3)

Class diagram
(level 4)



techtribes.je - Components - Content Updater

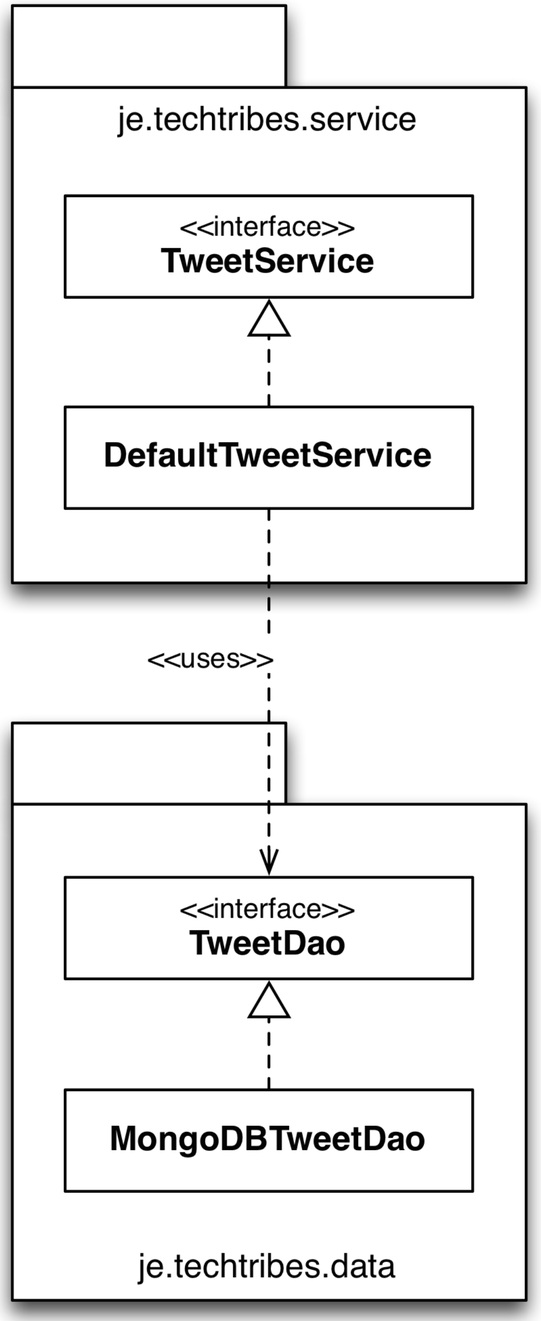
* Used by all components

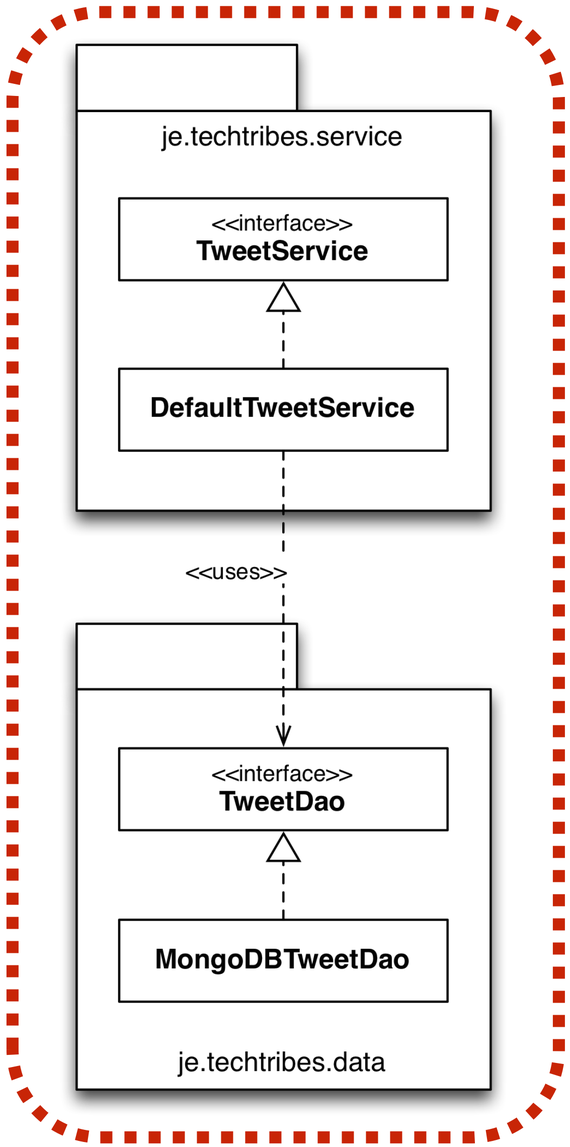
Context diagram
(level 1)

Container diagram
(level 2)

Component diagram
(level 3)

Class diagram
(level 4)





The
“Tweet Component”
doesn’t exist as a
single thing …
it’s a combination of
interfaces and classes
across a layered
architecture

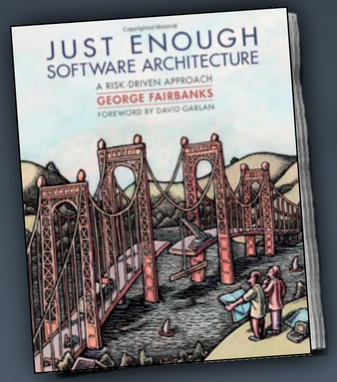
Where’s my component?

“But the component exists conceptually”

Does your code reflect the

abstractions

that you think about?



“the model-code gap”

Model-code gap. Your architecture models and your source code will not show the same things. The difference between them is the *model-code gap*. Your architecture models include some abstract concepts, like components, that your programming language does not, but could. Beyond that, architecture models include intensional elements, like design decisions and constraints, that cannot be expressed in procedural source code at all.

Consequently, the relationship between the architecture model and source code is complicated. It is mostly a refinement relationship, where the extensional elements in the architecture model are refined into extensional elements in source code. This is shown in Figure 10.3. However, intensional elements are not refined into corresponding elements in source code.

Upon learning about the model-code gap, your first instinct may be to avoid it. But reflecting on the origins of the gap gives little hope of a general solution in the short term: architecture models help you reason about complexity and scale because they are abstract and intensional; source code executes on machines because it is concrete and extensional.

Software Reflexion Models:
Bridging the Gap between Source and High-Level Models*

Gail C. Murphy and David Notkin

Dept. of Computer Science & Engineering
University of Washington
Box 352350
Seattle WA, USA 98195-2350
{gmurphy, notkin}@cs.washington.edu

Kevin Sullivan

Dept. of Computer Science
University of Virginia
Charlottesville VA, USA 22903
sullivan@cs.virginia.edu

Abstract

Software engineers often use high-level models (for instance, box and arrow sketches) to reason and communicate about an existing software system. One problem with high-level models is that they are almost always inaccurate with respect to the system's source code. We have developed an approach that helps an engineer use a high-level model of the structure of an existing software system as a lens through which to see a model of that system's source code. In particular, an engineer defines a high-level model and specifies how the model maps to the source. A tool then computes a software reflexion model that shows where the engineer's high-level model agrees with and where it differs from a model of the source.

The paper provides a formal characterization of reflexion models, discusses practical aspects of the approach, and relates experiences of applying the approach and tools to a number of different systems. The illustrative example used in the paper describes the application of reflexion models to NetBSD, an implementation of Unix comprised of 250,000 lines of C code. In only a few hours, an engineer computed several reflexion models that provided him with a useful, global overview of the structure of the NetBSD virtual memory subsystem. The approach has also been applied to aid in the understanding and experimental reengineering of the Microsoft Excel spreadsheet product.

*This research was funded in part by the NSF grant CCR-8858804 and a Canadian NSERC post-graduate scholarship.

SIGSOFT '95 Washington, D.C., USA
©1995 ACM 0-89791-716-2/95/0010...$3.50

1 Introduction

Software engineers often think about an existing software system in terms of high-level models. Box and arrow sketches of a system, for instance, are often found on engineers' whiteboards. Although these models are commonly used, reasoning about the system in terms of such models can be dangerous because the models are almost always inaccurate with respect to the system's source.

Current reverse engineering systems derive high-level models from the source code. These derived models are useful because they are, by their very nature, accurate representations of the source. Although accurate, the models created by these reverse engineering systems may differ from the models sketched by engineers; an example of this is reported by Wong et al. [WTMS95].

We have developed an approach, illustrated in Figure 1, that enables an engineer to produce sufficiently accurate high-level models in a different way. The engineer defines a high-level model of interest, extracts a source model (such as a call graph or an inheritance hierarchy) from the source code, and defines a declarative mapping between the two models. A *software reflexion model* is then computed to determine where the engineer's high-level model does and does not agree with the source model.[1] An engineer interprets the reflexion model and, as necessary, modifies the input to iteratively compute additional reflexion models.

[1]The old English spelling differentiates our use of "reflexion" from the field of reflective computing [Smi84].

1 Introduction

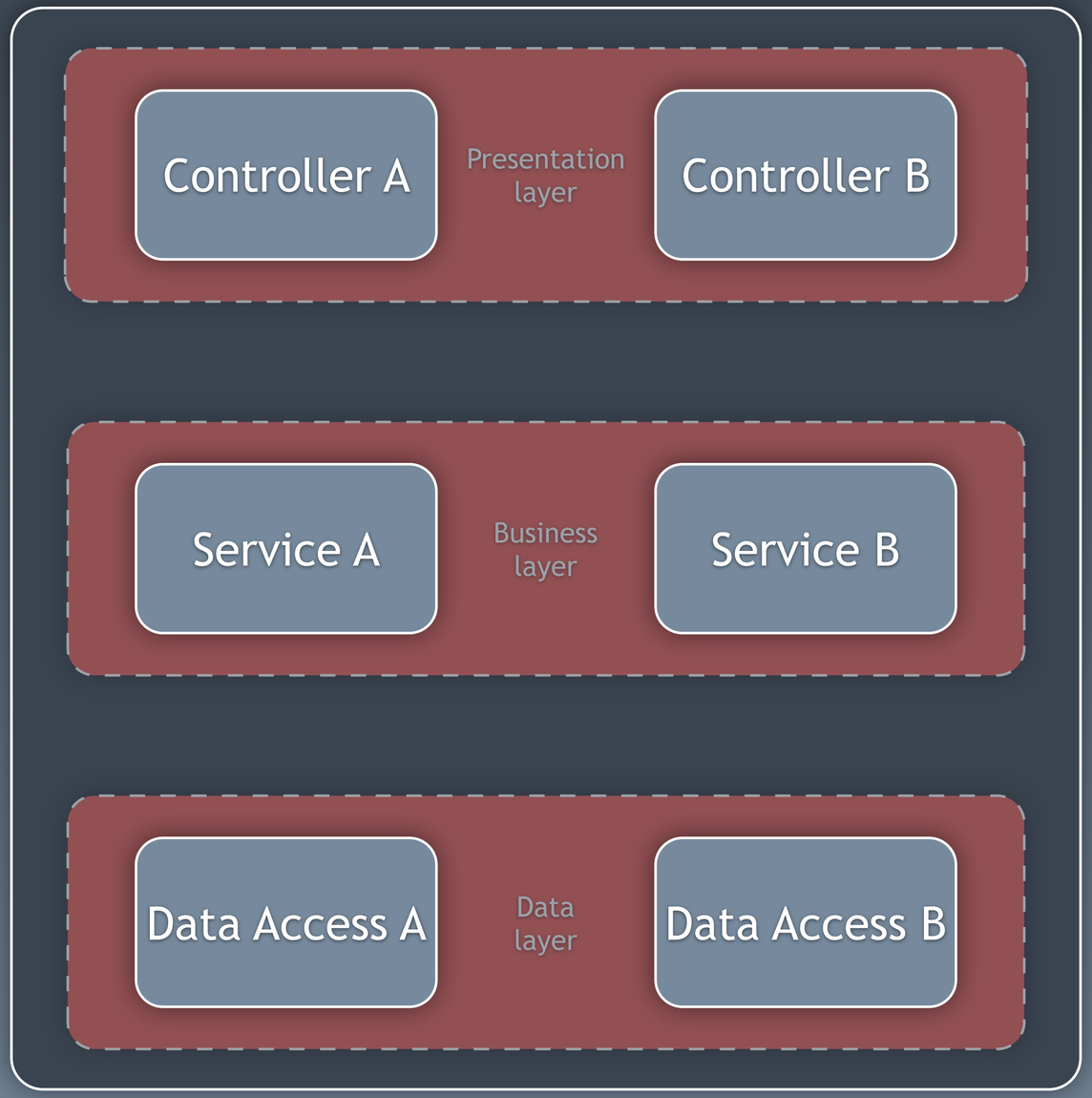
Software engineers often think about an existing software system in terms of high-level models. Box and arrow sketches of a system, for instance, are often found on engineers' whiteboards. Although these models are commonly used, reasoning about the system in terms of such models can be dangerous because the models are almost always inaccurate with respect to the system's source.

Current reverse engineering systems derive high-level models from the source code. These derived models are useful because they are, by their very nature, accurate representations of the source. Although accurate, the models created by these reverse engineering systems may differ from the models sketched by engineers; an exam-

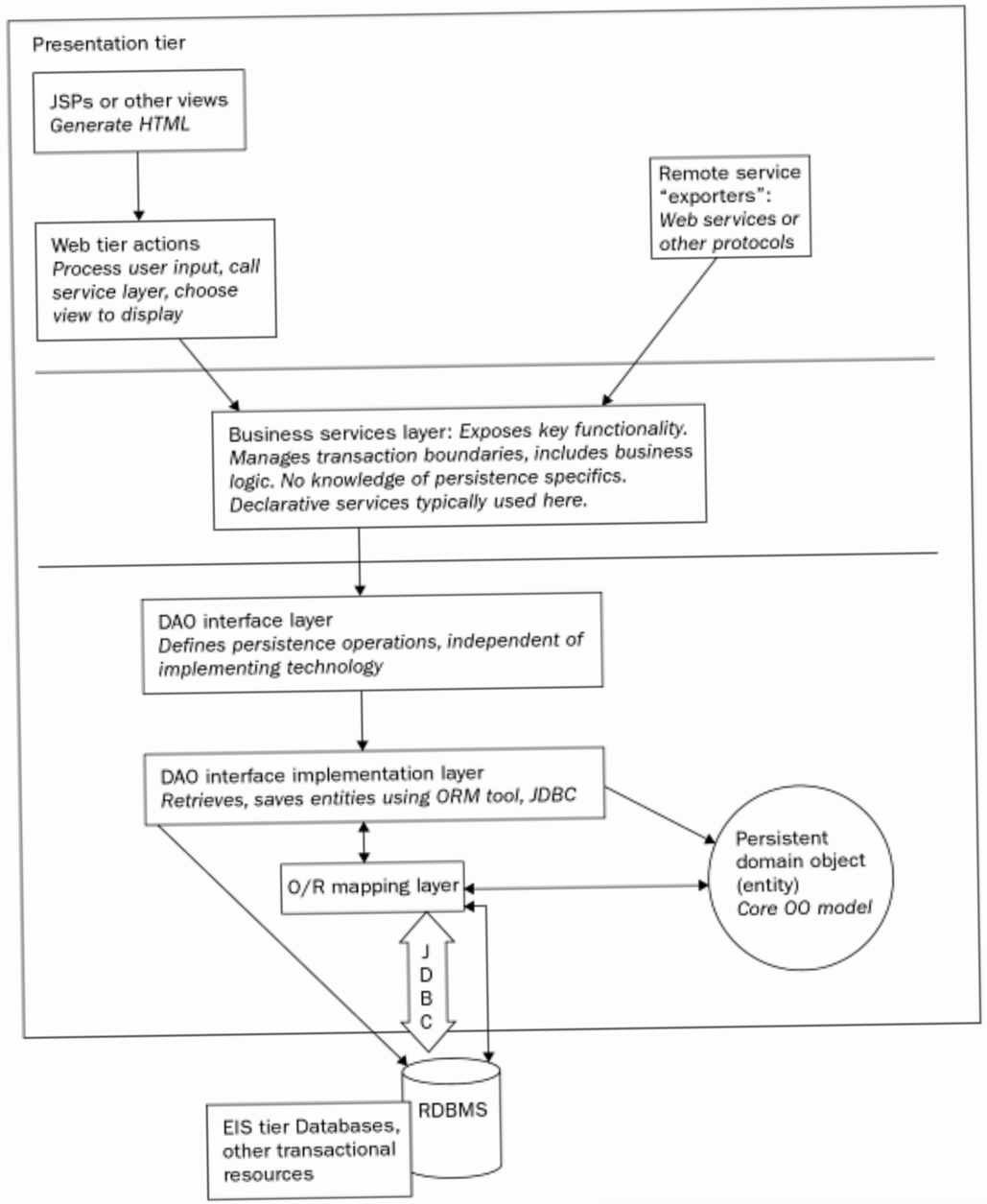
We often think

in components

but write classes



Package by layer (horizontal slicing)


Presentation tier
JSPs or other views
Generate HTML
Web tier actions
Process user input, call service layer, choose view to display
Remote service "exporters":
Web services or other protocols
Business services layer: Exposes key functionality. Manages transaction boundaries, includes business logic. No knowledge of persistence specifics. Declarative services typically used here.
DAO interface layer
Defines persistence operations, independent of implementing technology
DAO interface implementation layer
Retrieves, saves entities using ORM tool, JDBC
O/R mapping layer
Persistent domain object (entity)
Core OO model
J
D
B
C
RDBMS
EIS tier Databases, other transactional resources

Let's summarize each layer and its responsibilities, beginning closest to the database or other enterprise resources:

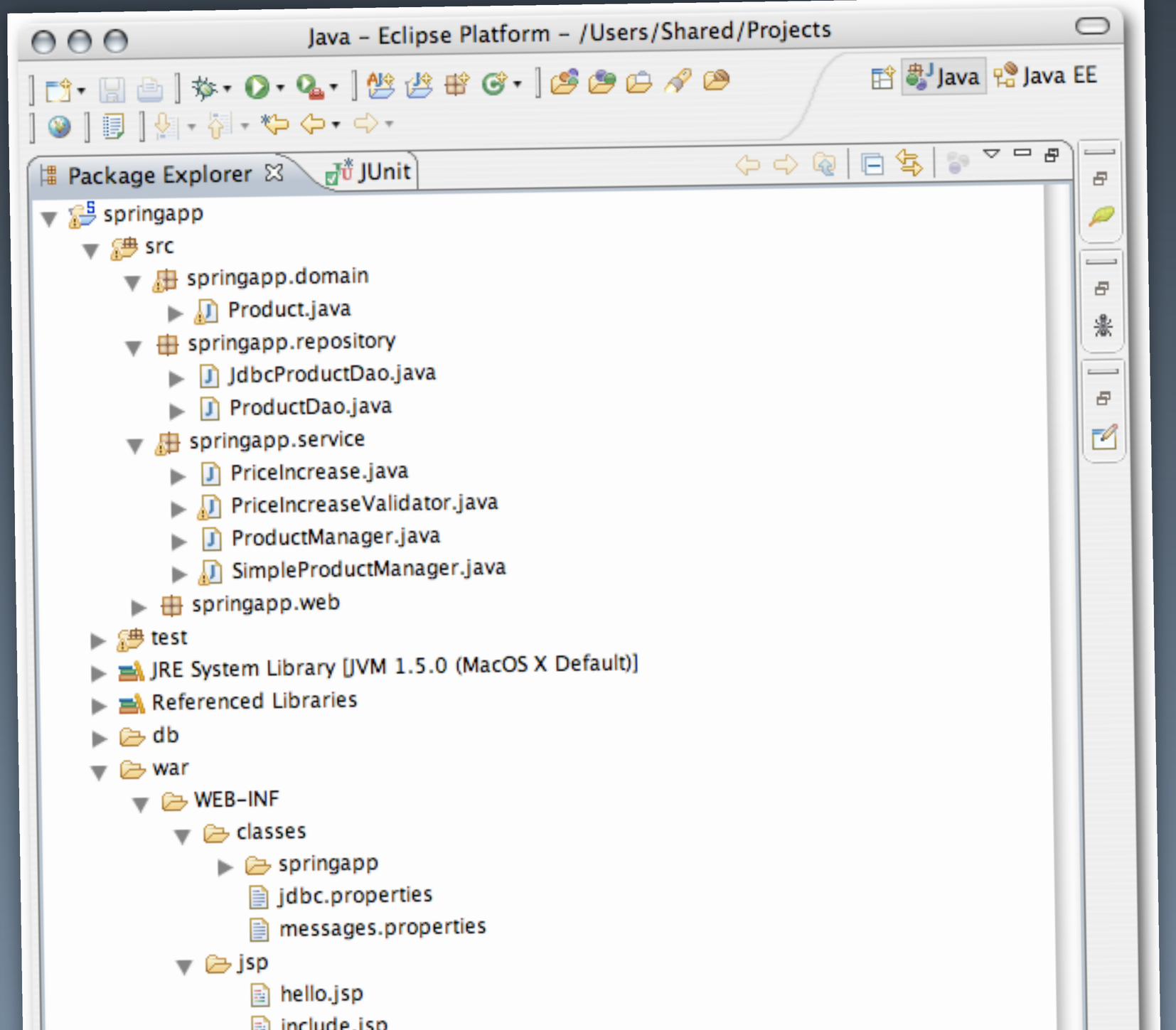
- **Presentation layer:** This is most likely to be a web tier. This layer should be as thin as possible. It should be possible to have alternative presentation layers — such as a web tier or remote web services facade — on a single, well-designed middle tier.
- **Business services layer:** This is responsible for transactional boundaries and providing an entry point for operations on the system as a whole. This layer should have no knowledge of presentation concerns, and should be reusable.
- **DAO interface layer:** This is a layer of interfaces *independent of any data access technology* that is used to find and persist persistent objects. This layer effectively consists of *Strategy* interfaces for the Business services layer. This layer should not contain business logic. Implementations of these interfaces will normally use an O/R mapping technology or Spring's JDBC abstraction.
- **Persistent domain objects:** These model real objects or concepts such as a bank account.
- **Databases and legacy systems:** By far the most common case is a single RDBMS. However, there may be multiple databases, or a mix of databases and other transactional or non-transactional legacy systems or other enterprise resources. The same fundamental architecture is applicable in either case. This is often referred to as the *EIS* (*Enterprise Information System*) tier.

In a J2EE application, all layers except the EIS tier will run in the application server or web container. Domain objects will typically be passed up to the presentation layer, which will display data they contain, *but not modify them*, which will occur only within the transactional boundaries defined by the business services layer. Thus there is no need for distinct Transfer Objects, as used in traditional J2EE architecture.

In the following sections we'll discuss each of these layers in turn, beginning closest to the database.

Spring aims to decouple architectural layers, so that each layer can be modified as

Spring aims to decouple architectural layers, so that each layer can be modified as far as possible without impacting other layers. No layer is aware of the concerns of the layer above; as far as possible, dependency is purely on the layer immediately below. Dependency between layers is normally in the form of interfaces, ensuring that coupling is as loose as possible.


Java - Eclipse Platform - /Users/Shared/Projects
Java
Java EE
Package Explorer
JUnit
springapp
src
springapp.domain
Product.java
springapp.repository
JdbcProductDao.java
ProductDao.java
springapp.service
PriceIncrease.java
PriceIncreaseValidator.java
ProductManager.java
SimpleProductManager.java
springapp.web
test
JRE System Library [JVM 1.5.0 (MacOS X Default)]
Referenced Libraries
db
war
WEB-INF
classes
springapp
jdbc.properties
messages.properties
jsp
hello.jsp

java - Standard project/package structure of a j2ee web application - Stack Overflow

stackoverflow.com/questions/5878774/standard-project-package-structure-of-a-j2ee-web-application

Reader

1 Answer

active oldest votes

8

If you are using maven, it's best to follow the standard maven project layout. You can get maven to generate this structure for you by doing,

```
mvn archetype:generate
```

and select spring-mvc-jpa-archetype from the list of choices

This will give you a package structure like,

```
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   └── mygroup
    │   │       ├── controller
    │   │       │   ├── HomeController.java
    │   │       │   └── PersonController.java
    │   │       ├── dao
    │   │       │   └── PersonDao.java
    │   │       └── model
    │   │           └── Person.java
    │   ├── resources
    │   │   ├── db.properties
    │   │   ├── log4j.xml
    │   │   └── META-INF
    │   │       └── persistence.xml
    │   └── webapp
    │       ├── index.html
    │       ├── META-INF
    │       │   ├── context.xml
    │       │   └── MANIFEST.MF
    │       ├── resources
    │       │   └── css
    │       │       └── screen.css
    │       └── WEB-INF
    │           ├── spring
    │           │   ├── app
    │           │   │   ├── controllers.xml
    │           │   │   └── servlet-context.xml
    │           │   ├── db.xml
    │           │   └── root-context.xml
    │           └── views
```

Technical Operations Engineer, Devops, Cloud Systems and AWS
Voyanta
London, United Kingdom

Linux Service Engineers
Shazam
London, United Kingdom

3 x Senior Engineer (Ruby) 3 month+ contract
We are Friday Limited
London, United Kingdom

More jobs near London...

Related

2 How do I use a custom authentication mechanism for a Java web application with Spring Security?

0 Is there a sample web project of integrating Spring-MVC and Spring-Data-JPA?

1 Reusing DAOs in another web application

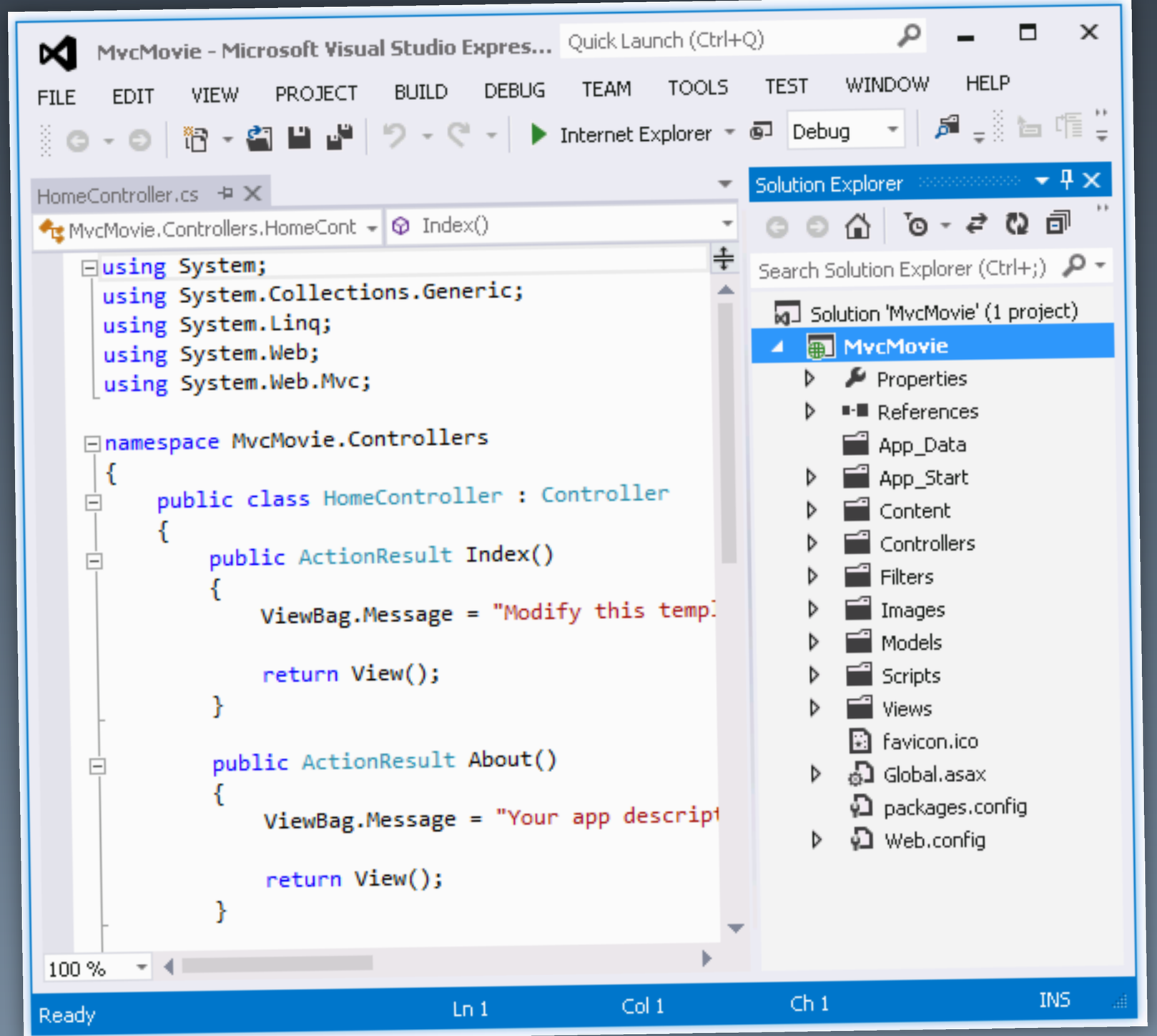
0 Java Web Application Warming

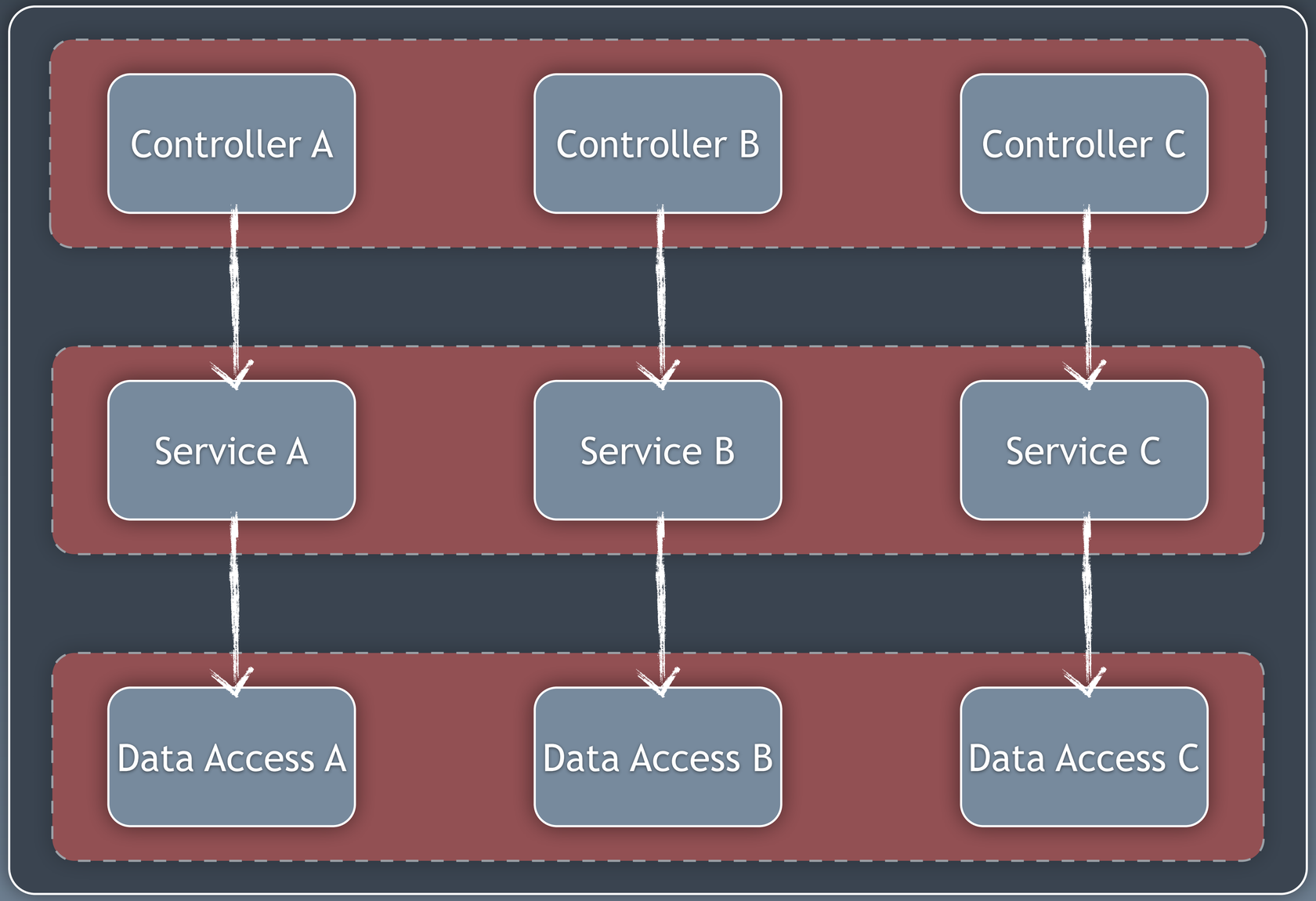
3 Is a parallel Spring-MVC application possible with a non-spring web app?

2 Adding a web interface (Spring MVC) to existing Java application

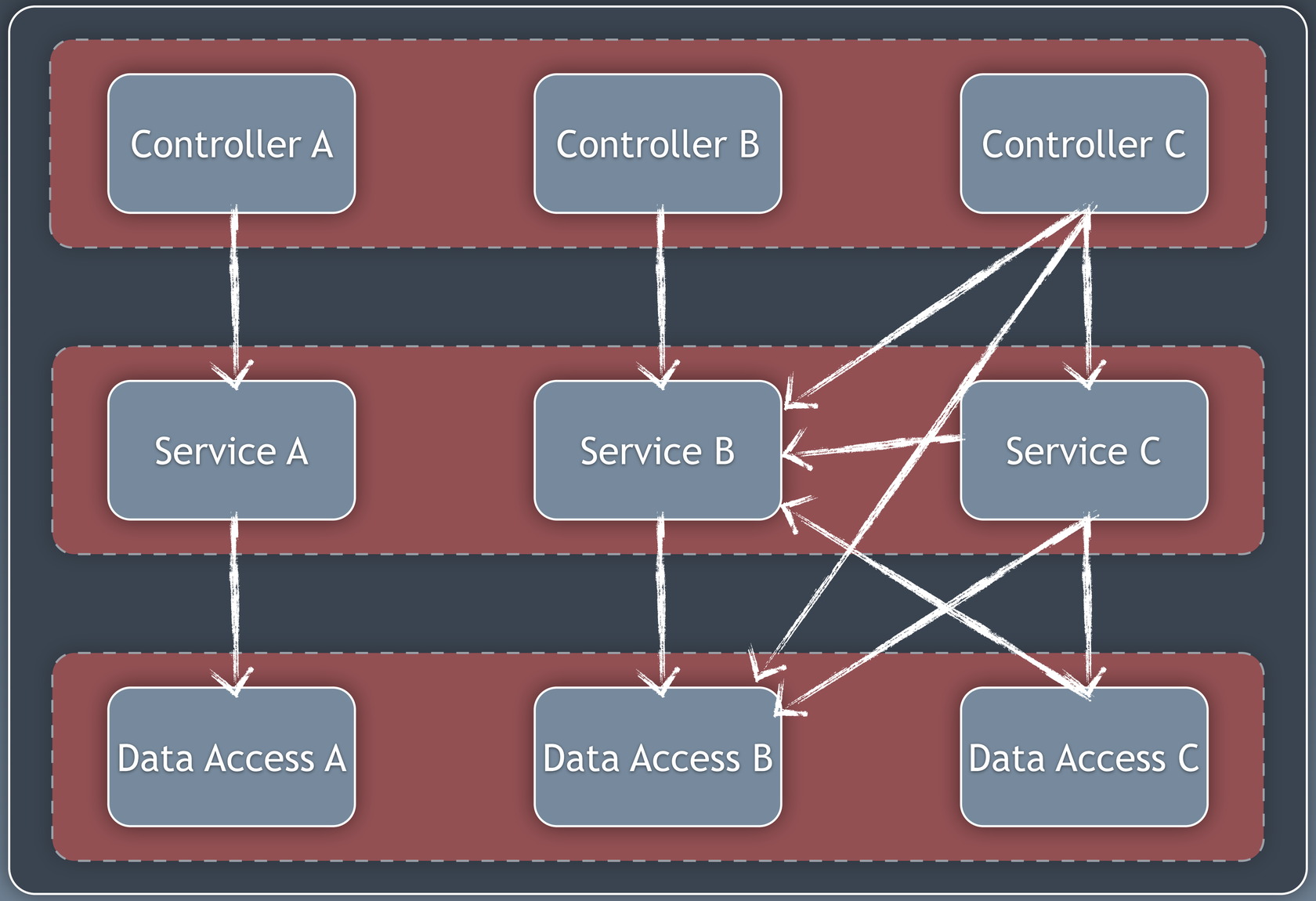
0 package strucutre for Spring MVC project with multiple sub projects using maven

0 How to add a setup

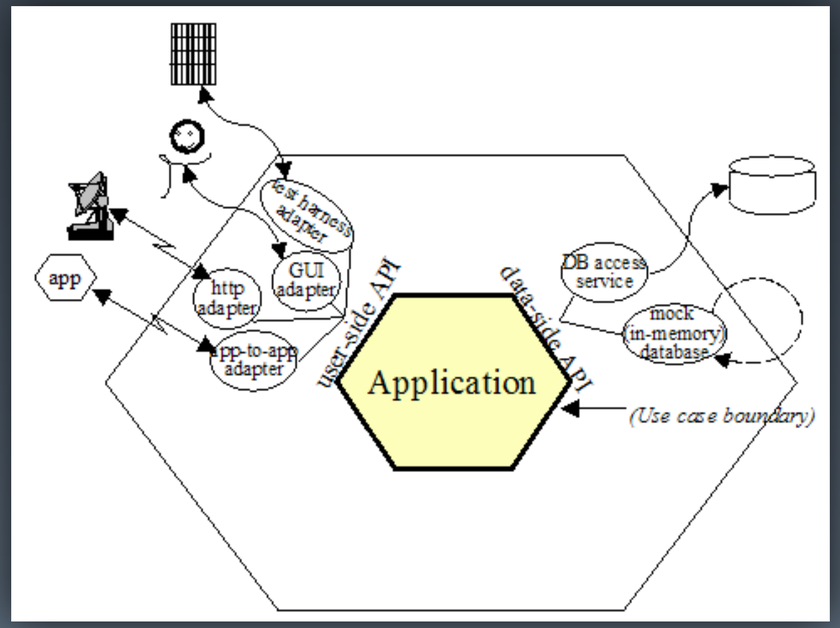

MvcMovie - Microsoft Visual Studio Expres...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Internet Explorer
Debug
HomeController.cs
MvcMovie.Controllers.HomeCont
Index()
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace MvcMovie.Controllers
{
public class HomeController : Controller
{
public ActionResult Index()
{
ViewBag.Message = "Modify this templ
return View();
}
public ActionResult About()
{
ViewBag.Message = "Your app descript
return View();
}
100 %
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'MvcMovie' (1 project)
MvcMovie
Properties
References
App_Data
App_Start
Content
Controllers
Filters
Images
Models
Scripts
Views
favicon.ico
Global.asax
packages.config
Web.config
Ready
Ln 1
Col 1
Ch 1
INS

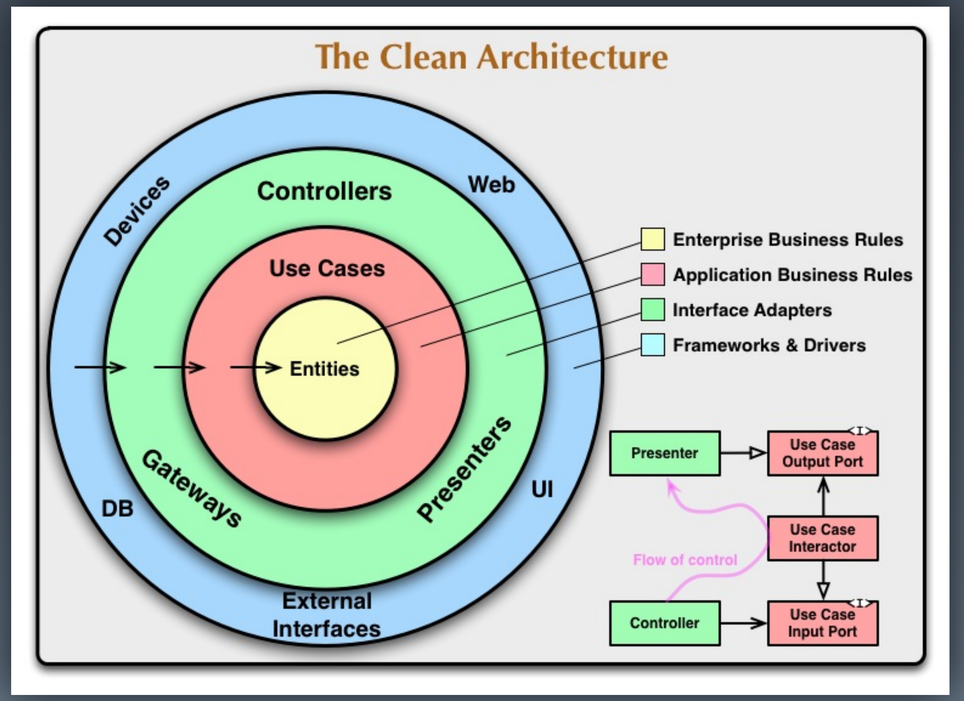


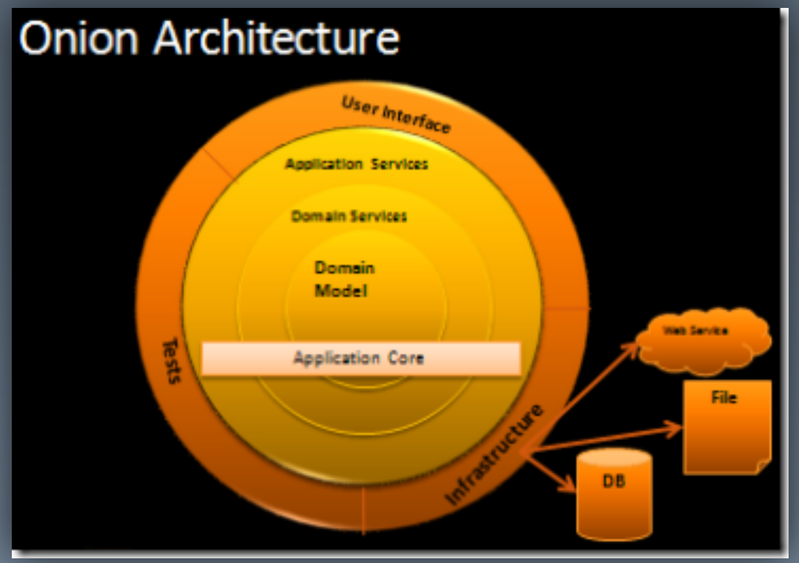
Package by layer (horizontal slicing)



Package by layer (horizontal slicing)



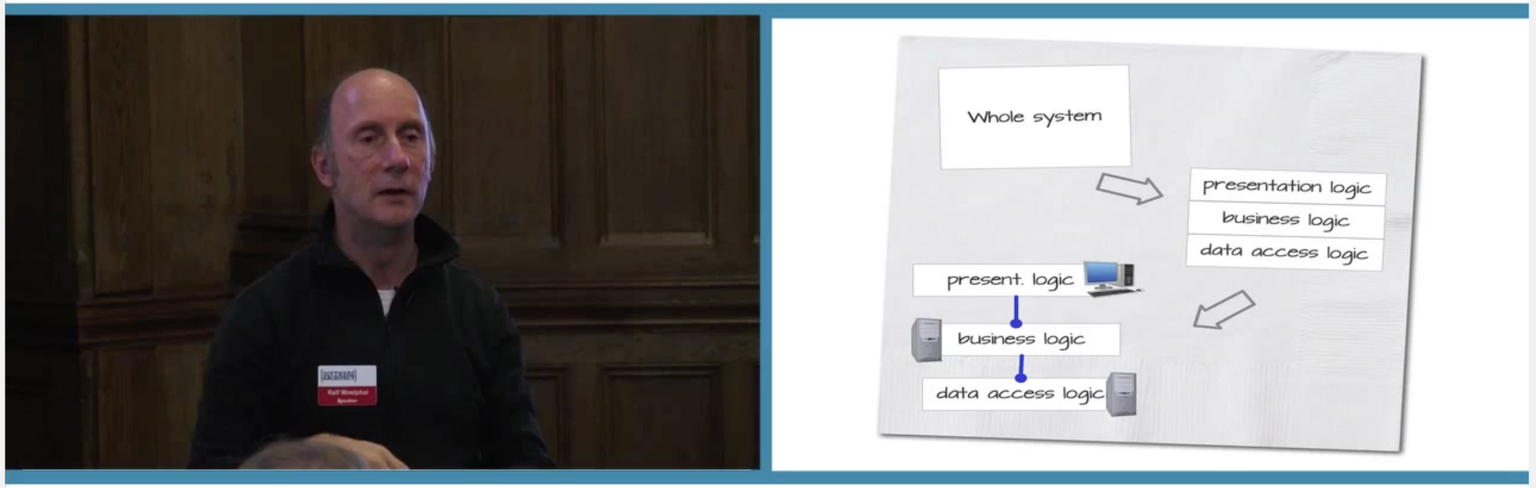




Hexagons, onions, ports & adapters, etc

Should layers be

considered harmful?



Let software design come to life using software cells - Ralf Westphal

DevWeek Events

Subscribe 403

230 views

Add to Share More 4 0

OSI layers vs software layers

(~13 minutes)

Cargo cult programming can also refer to the results of applying a design pattern or coding style blindly without understanding the reasons behind that design principle.

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

Is architectural layering in software another cargo cult?

Are layers significant

structural elements

or just an

implementation detail?

LINKS

Blogroll
Subscribe via RSS

AUTHORS

Uncle Bob
Paul Pagel
Micah Martin
Eric Smith
Doug Bradbury
Colin Jones
Mike Jansen
Jim Suchy
Craig Demyanovich
Dariusz Pasciak
Kevin Buchanan
Eric Meyer
Myles Megyesi
Dave Moore
Chris Peak
Patrick Gombert
Margaret Pagel
Steve Kim
Sandro Padin
Kevin Liddle
Malcolm Newsome

Screaming Architecture

Uncle Bob → 30 Sep 2011 + Architecture

Imagine that you are looking at the blueprints of a building. This document, prepared by an architect, tells you the plans for the building. What do these plans tell you?

If the plans you are looking at are for a single family residence, then you'll likely see a front entrance, a foyer leading to a living room and perhaps a dining room. There'll likely be a kitchen a short distance away, close to the dining room. Perhaps a dinette area next to the kitchen, and probably a family room close to that. As you looked at those plans, there'd be no question that you were looking at a *house*. The architecture would *scream*: **house**.

Or if you were looking at the architecture of a library, you'd likely see a grand entrance, an area for check-in-out clerks, reading areas, small conference rooms, and gallery after gallery capable of holding bookshelves for all the books in the library. That architecture would *scream*: **Library**.

So what does the architecture of your application scream? When you look at the top level directory structure, and the source files in the highest level package; do they scream: **Health Care System**, or **Accounting System**, or **Inventory Management System**? Or do they scream: **Rails**, or **Spring/Hibernate**, or **ASP**?

The Theme of an Architecture

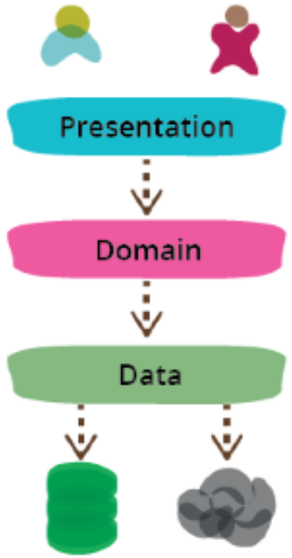
MARTIN FOWLER

Intro Videos Design Agile Refactoring NoSQL DSL Continuous Del

PresentationDomainDataLayering

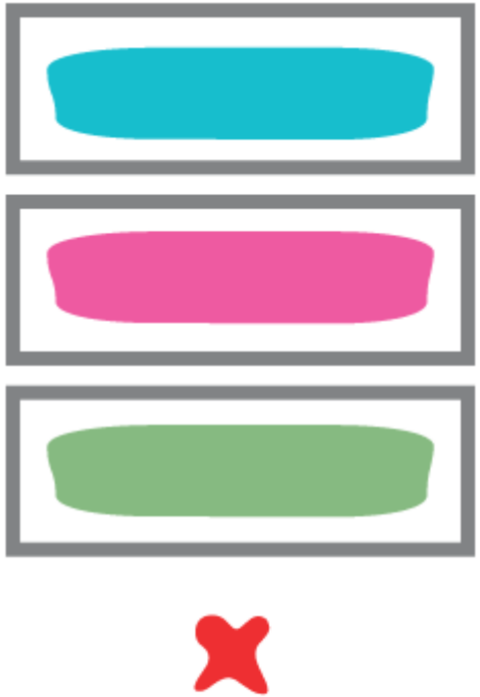
Martin Fowler
26 August 2015

One of the most common ways to modularize an information-rich
to separate it into three broad layers: presentation (UI), domain l
business logic), and data access. So you often see web applicati
into a web layer that knows about handling http requests and ren
HTML, a business logic layer that contains validations and calcul
a data access layer that sorts out how to manage persistant data
database or remote services.

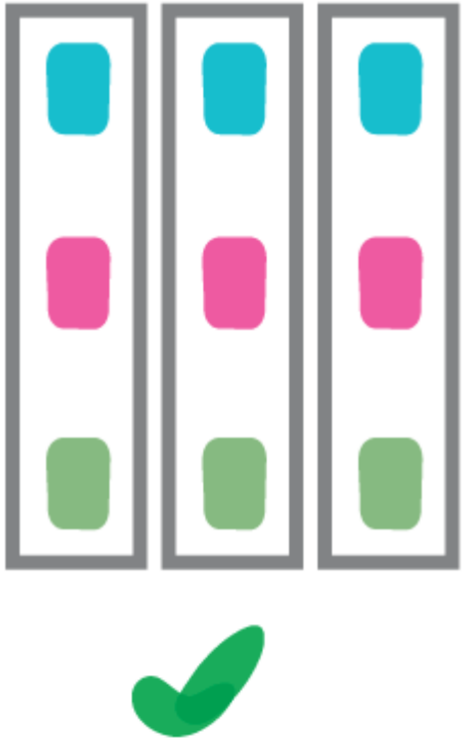


On the whole I've found this to be an effective form of modulariz
many applications and one that I regularly use and encourage. It
advantage (for me) is that it allows me to **reduce the scope of** n

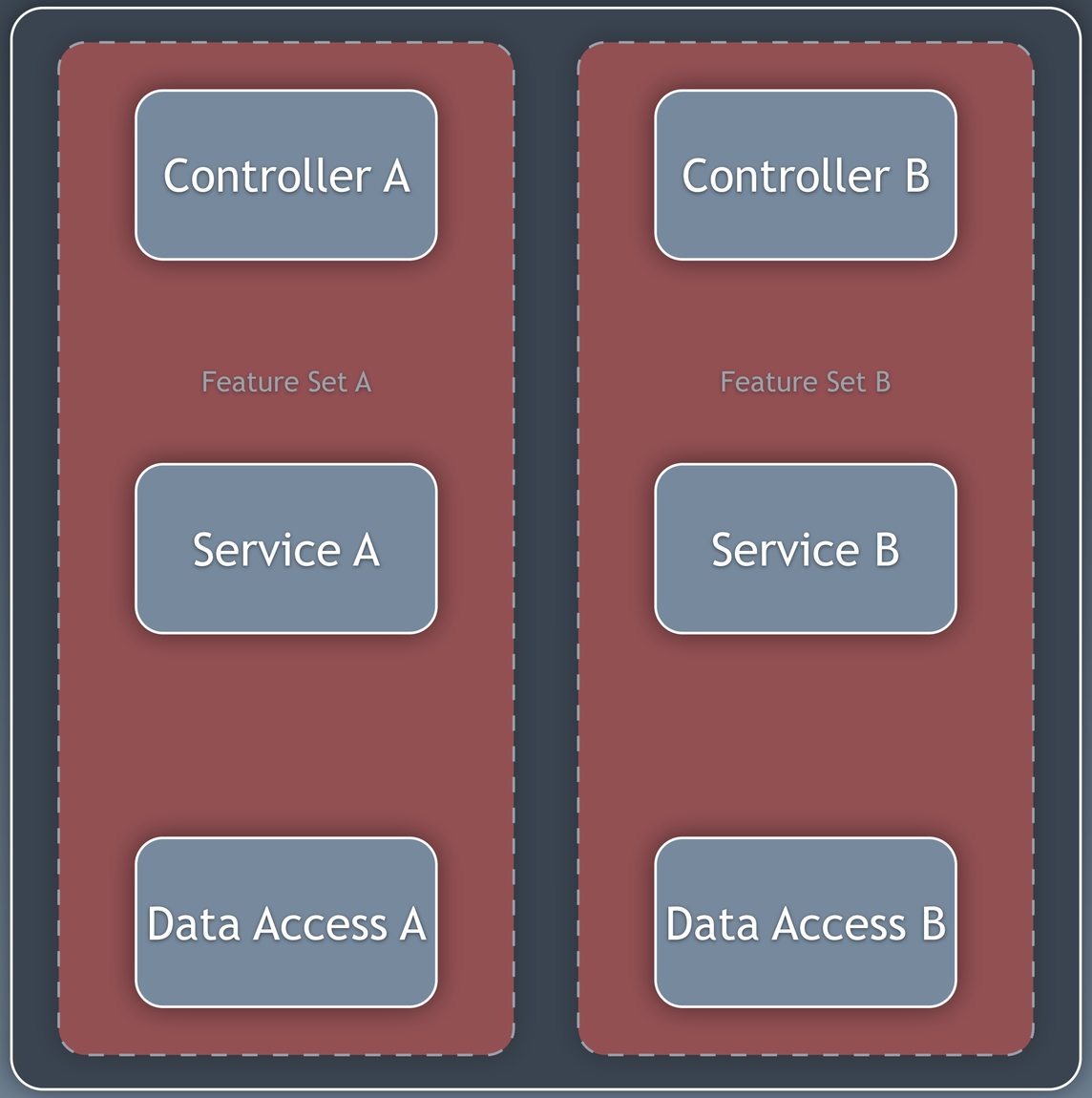
Although presentation-domain-data separation is a common approach, it should only be applied at a relatively small granularity. As an application grows, each layer can get sufficiently complex on its own that you need to modularize further. When this happens it's usually not best to use presentation-domain-data as the higher level of modules. Often frameworks encourage you to have something like view-model-data as the top level namespaces; that's ok for smaller systems, but once any of these layers gets too big you should split your top level into domain oriented modules which are internally layered.



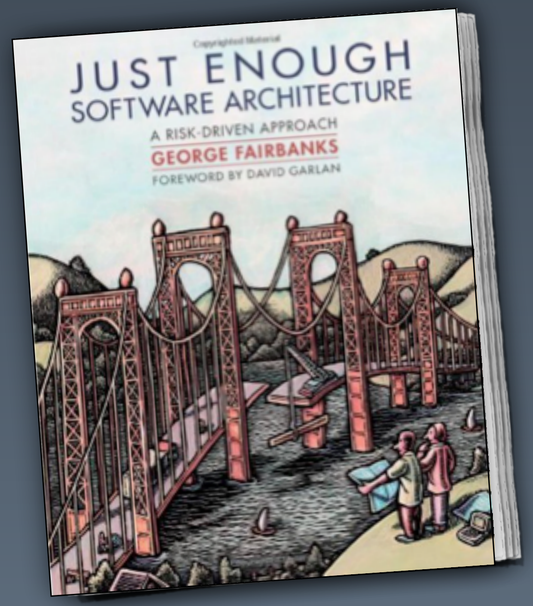
Don't use layers as the top level modules in a complex application...



... instead make your top level modules be full-stack



Package by feature (vertical slicing)



“architecturally-evident coding style”

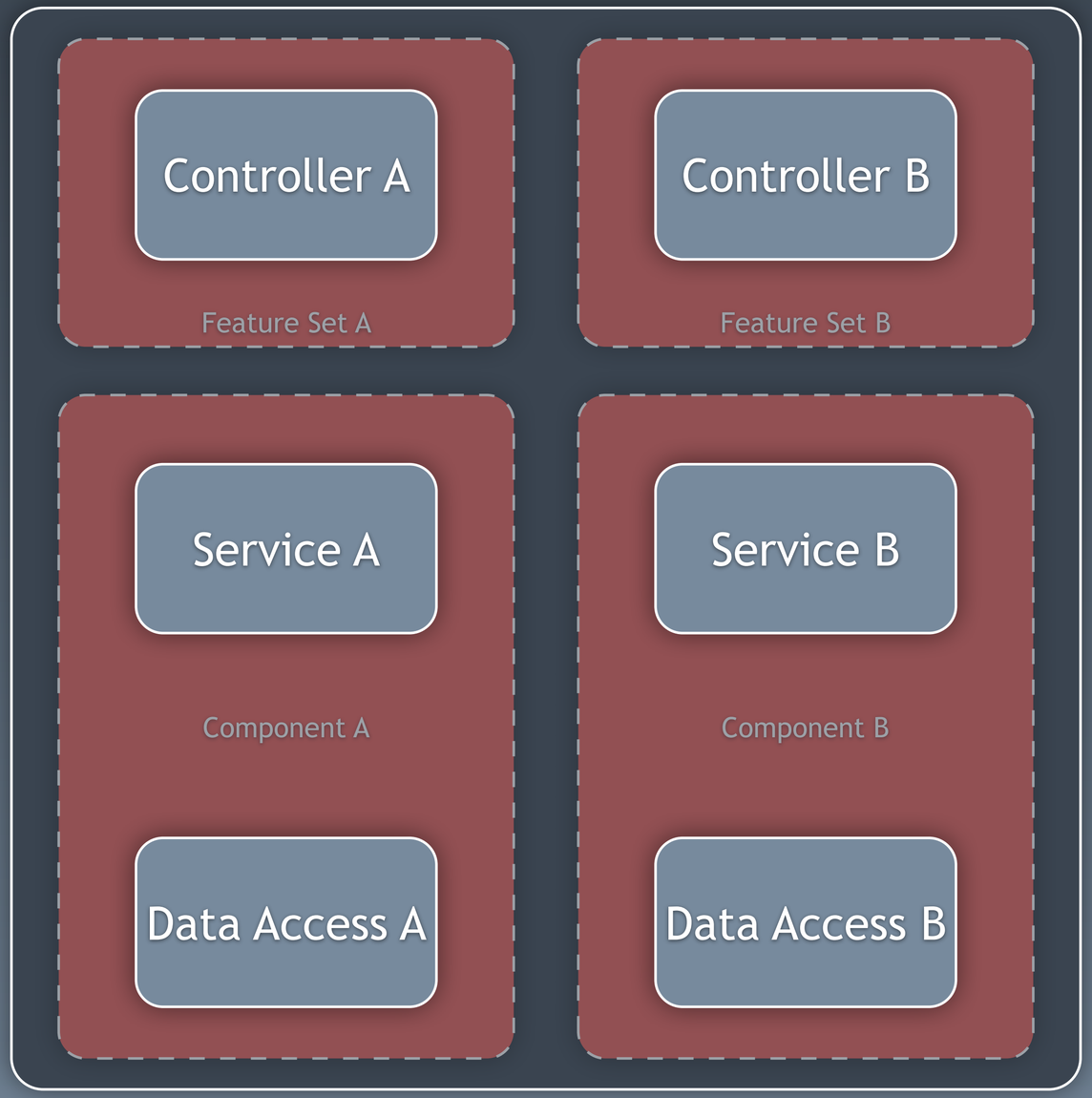
Architecturally-evident coding styles include:

Annotations/attributes (`@Component`, `[Component]`, etc)

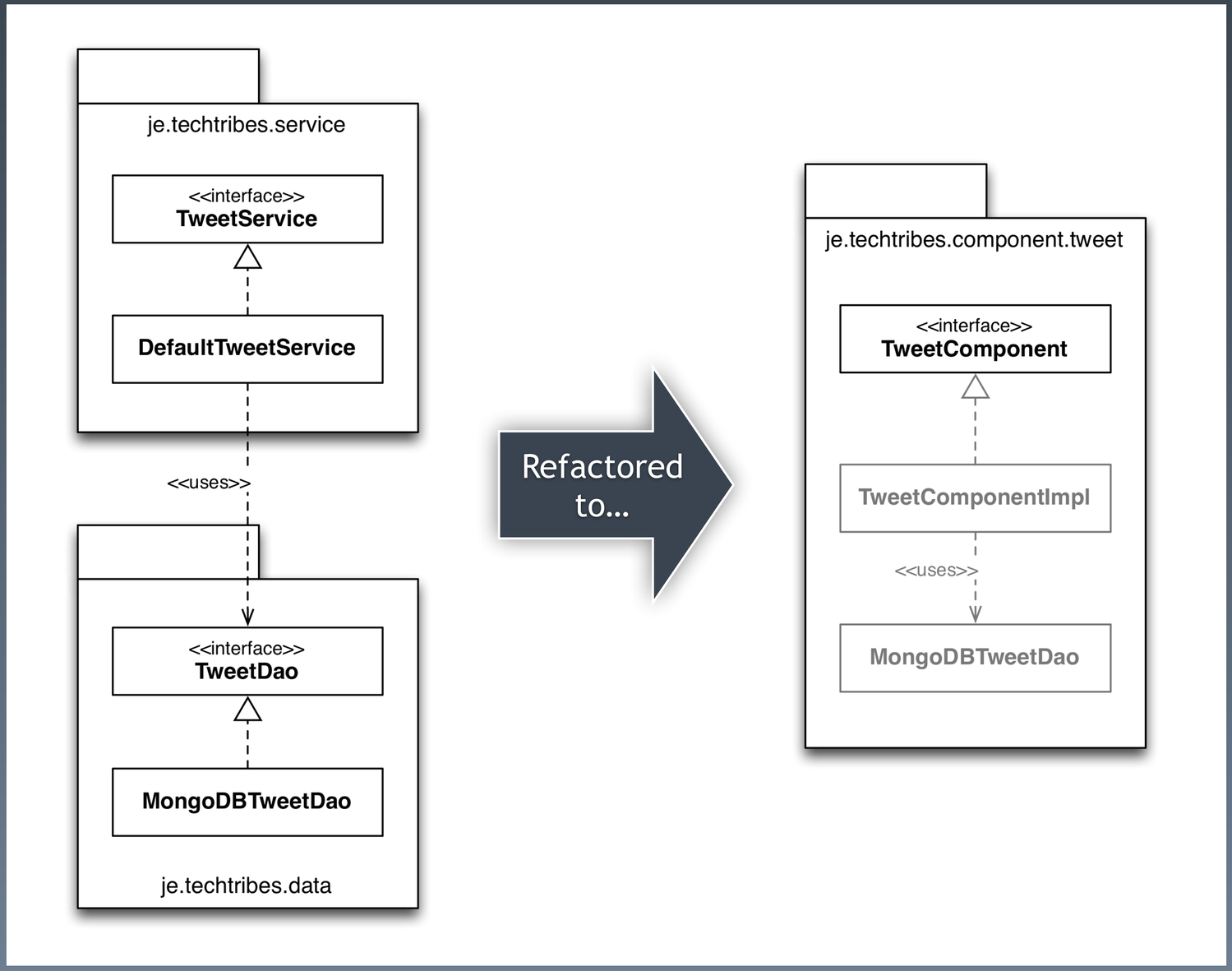
Naming conventions (`*Service`)

Namespacing/packaging (`com.mycompany.system.components.*`)

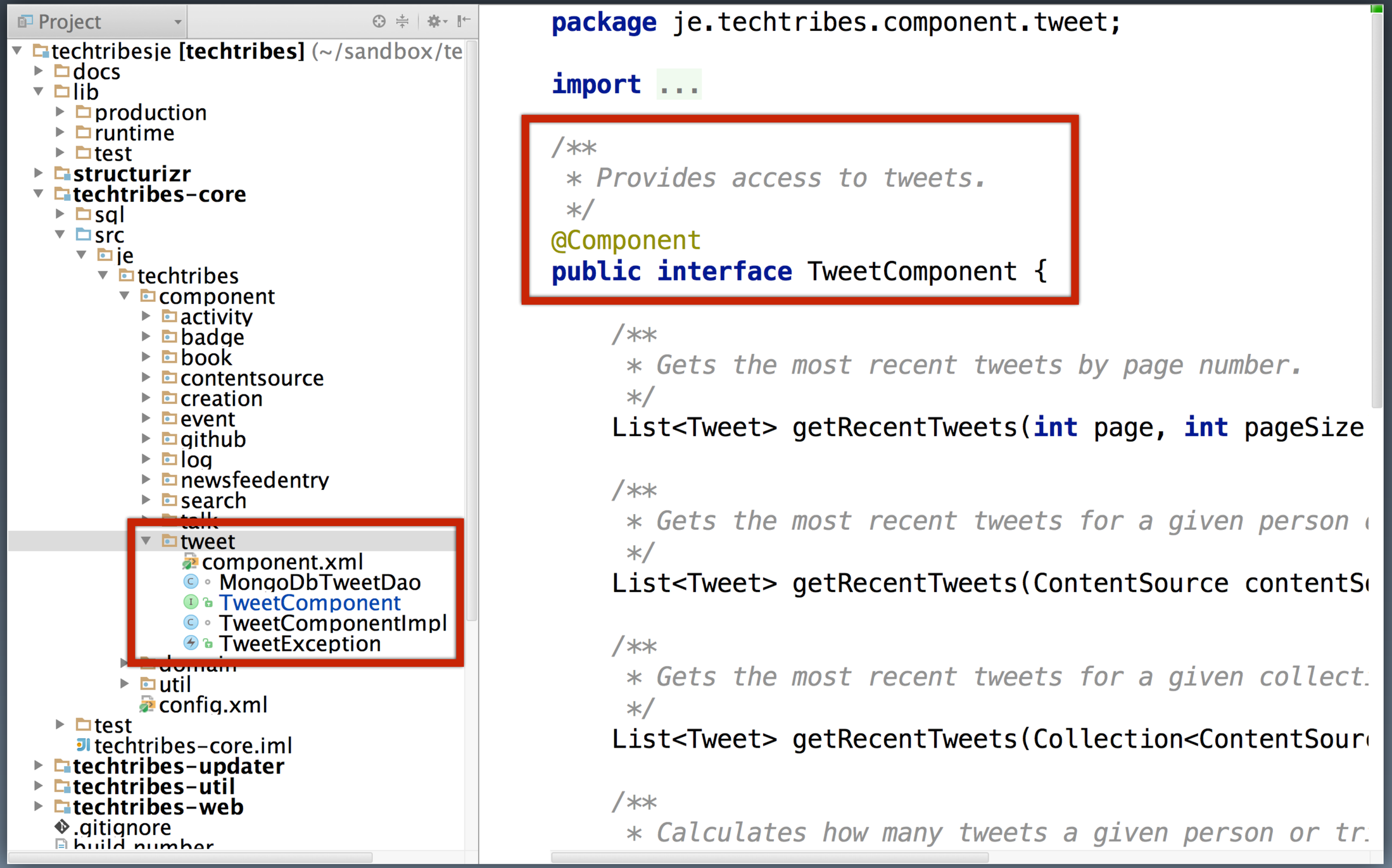
Maven modules, OSGi modules, Java 9 and Jigsaw, JavaScript module patterns, ECMAScript 6 modules, microservices, etc



Package by component



An example of an architecturally-evident coding style

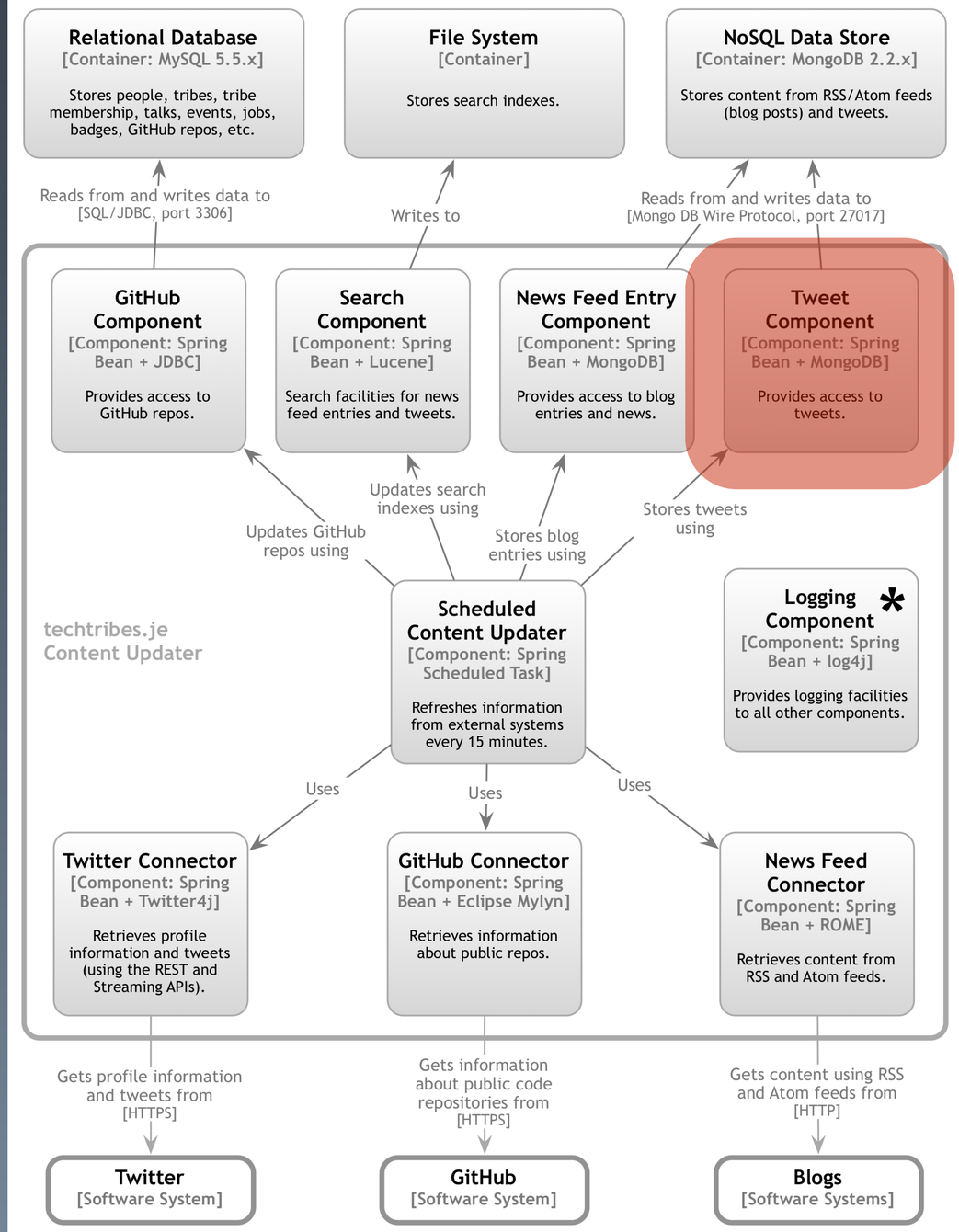

Project
techtribesje [techtribes] (~/sandbox/te
docs
lib
production
runtime
test
structurizr
techtribes-core
sql
src
je
techtribes
component
activity
badge
book
contentsource
creation
event
github
log
newsfeedentry
search
tweet
component.xml
MongoDbTweetDao
TweetComponent
TweetComponentImpl
TweetException
util
config.xml
test
techtribes-core.iml
techtribes-updater
techtribes-util
techtribes-web
.gitignore
package je.techtribes.component.tweet;
import ...
/**
* Provides access to tweets.
*/
@Component
public interface TweetComponent {
/**
* Gets the most recent tweets by page number.
*/
List<Tweet> getRecentTweets(int page, int pageSize
/**
* Gets the most recent tweets for a given person
*/
List<Tweet> getRecentTweets(ContentSource contentS
/**
* Gets the most recent tweets for a given collect
*/
List<Tweet> getRecentTweets(Collection<ContentSour
/**
* Calculates how many tweets a given person or tr

Context diagram
(level 1)

Container diagram
(level 2)

Component diagram
(level 3)

Class diagram
(level 4)



techtribes.je - Components - Content Updater

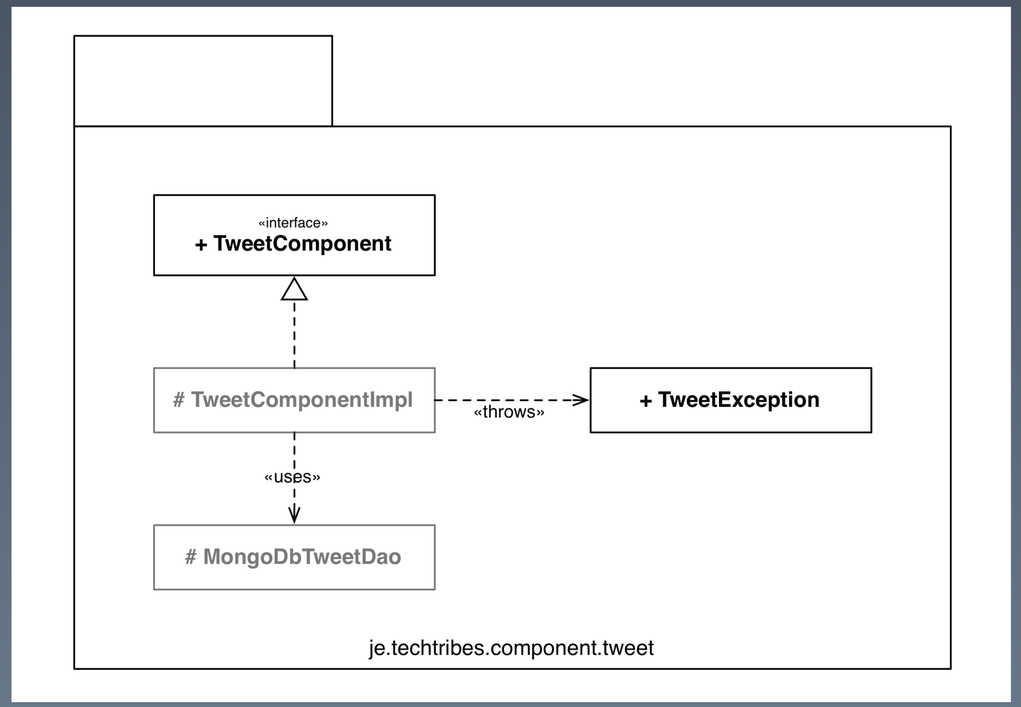
* Used by all components

Context diagram
(level 1)

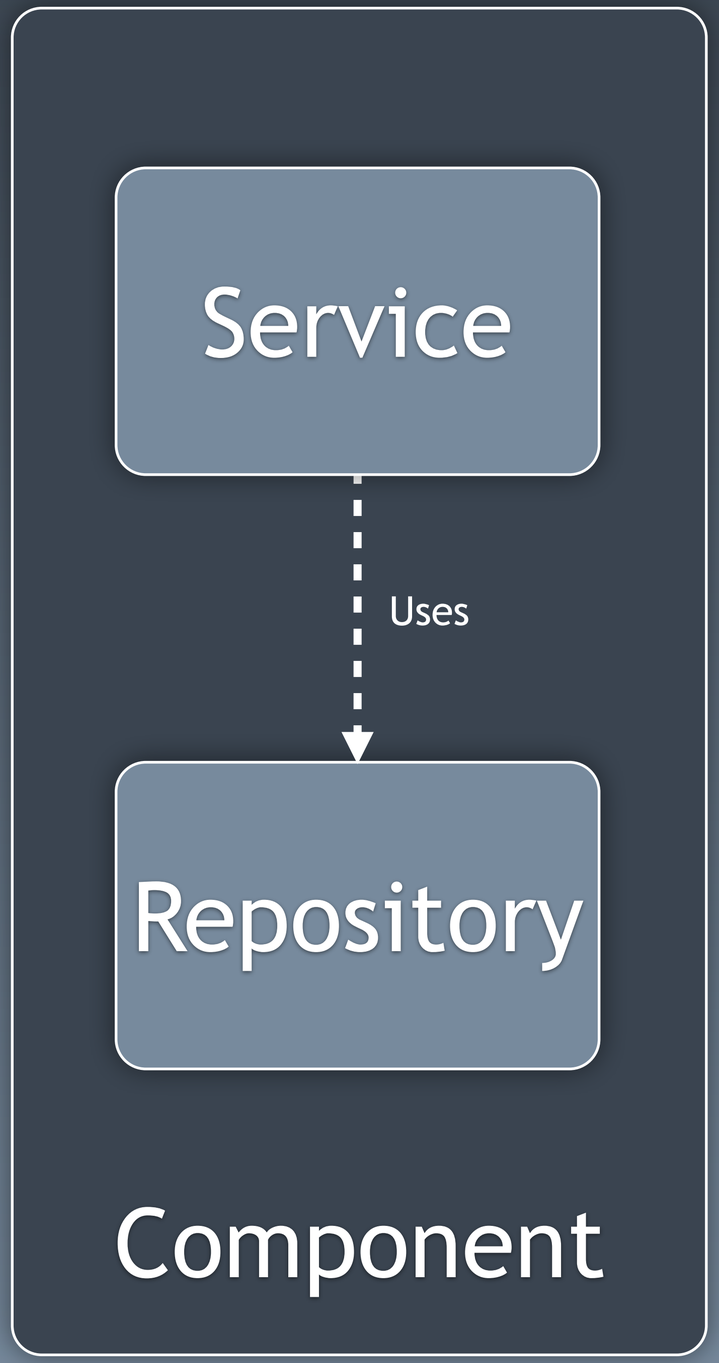
Container diagram
(level 2)

Component diagram
(level 3)

Class diagram
(level 4)

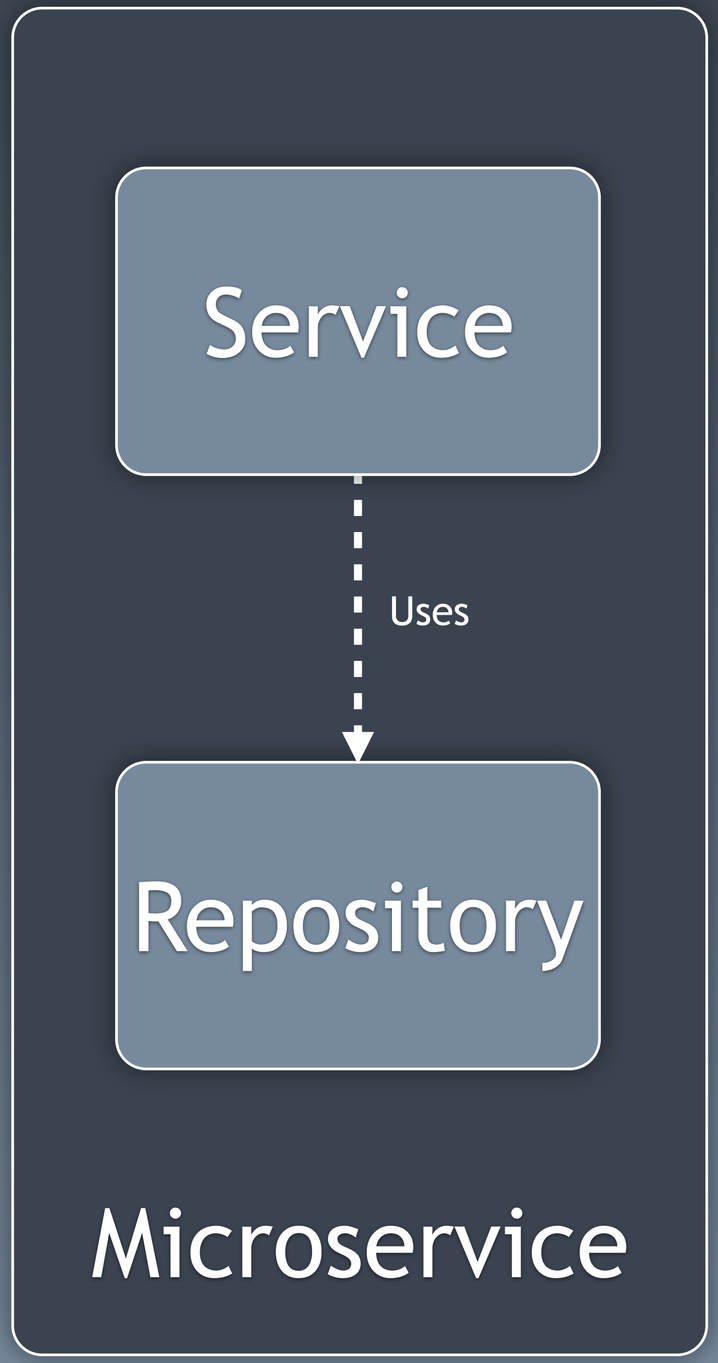


Modularity as a principle



Impermeable boundaries

Apply component-based design and service-orientation to a monolithic application



Design decisions

How do *you* **design** software?

What is your decomposition strategy?

Decomposition (computer science)

From Wikipedia, the free encyclopedia

Decomposition in computer science, also known as **factoring**, is breaking a complex problem or system into parts that are easier to conceive, understand, program, and maintain.

Contents [hide]

Decomposition paradigm [edit]

A decomposition paradigm in computer programming is a strategy for organizing a program as a number of parts, and it usually implies a specific way to organize a program text. Usually the aim of using a decomposition paradigm is to optimize some metric related to program complexity, for example the modularity of the program or its maintainability.

Most decomposition paradigms suggest breaking down a program into parts so as to minimize the static dependencies among those parts, and to maximize the cohesiveness of each part. Some popular decomposition paradigms are the procedural, modules, abstract data type and object oriented ones.

More generally, functional decomposition in computer science is a technique for mastering the complexity of the function of a model. A functional model of a system is thereby replaced by a series of functional models of subsystems.[3]

Programming Techniques
R. Morris Editor

On the Criteria To Be Used in Decomposing Systems into Modules

D.L. Parnas
Carnegie-Mellon University

Expected Benefits of Modular Programming

The benefits expected of modular programming are: (1) managerial—development time should be shortened because separate groups would work on each module with little need for communication: (2) product flexibility—it should be possible to make drastic changes to one module without a need to change others; (3) comprehensibility—it should be possible to study the system one module at a time. The whole system can therefore be better designed because it is better understood.

Interaction style

(synchronous, asynchronous,
commands, messages, events, etc)

Where do you put

domain classes?

Shared code, and utilities?

Shared database tables,
database schemas
and foreign keys

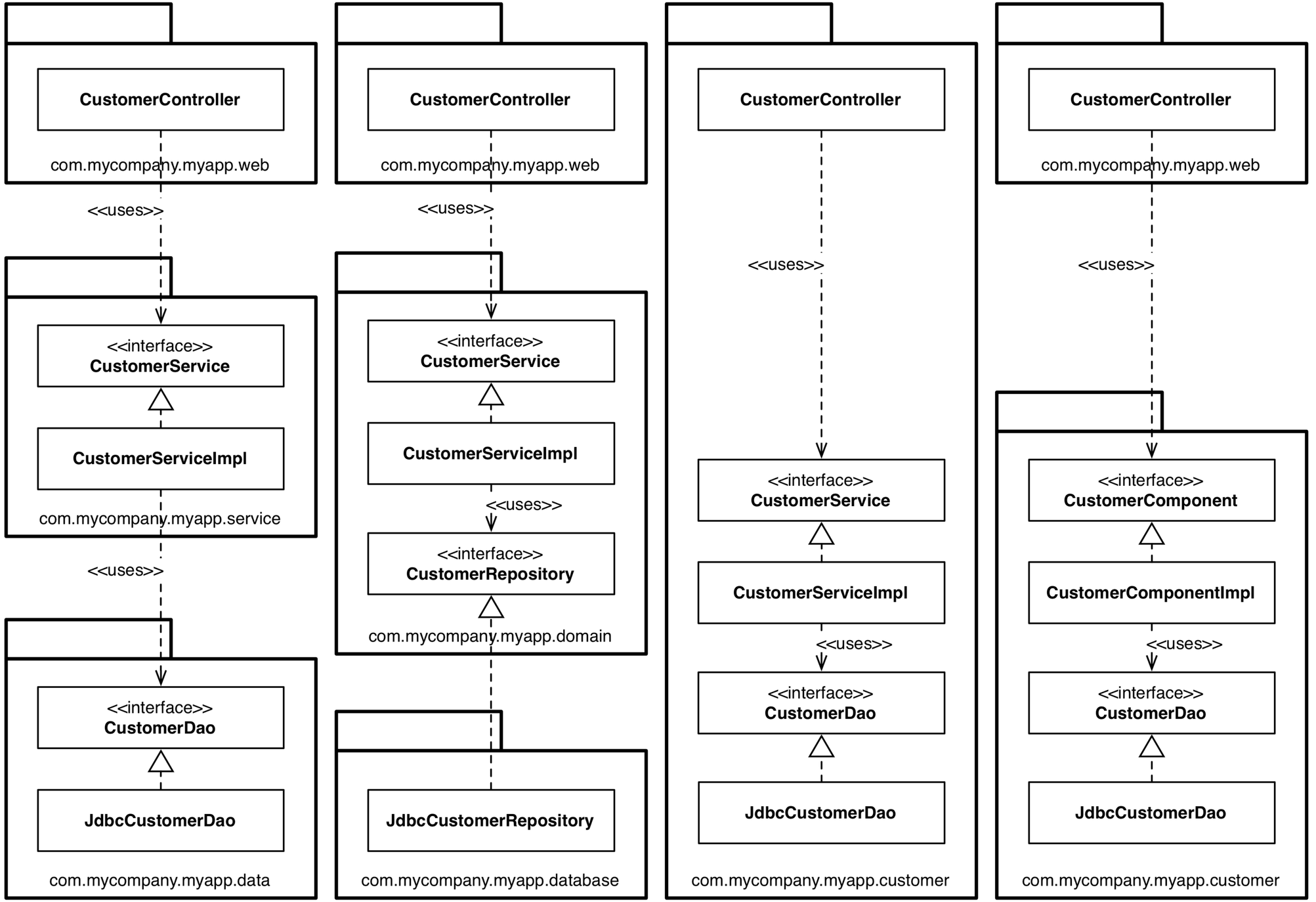
or
polyglot persistence?

Conscious decisions

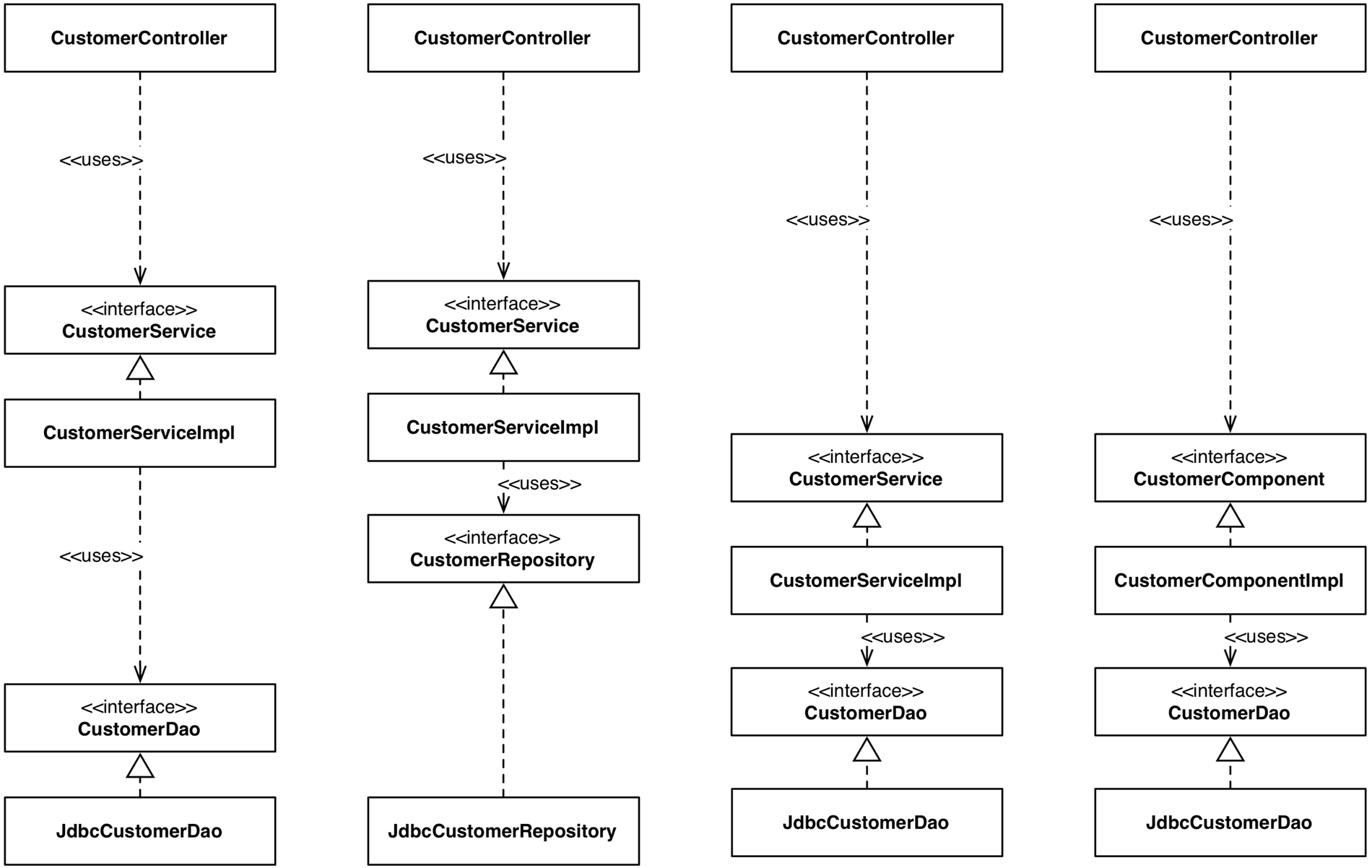
rather than cargo cult

Implementation details

But the devil is in the implementation details…

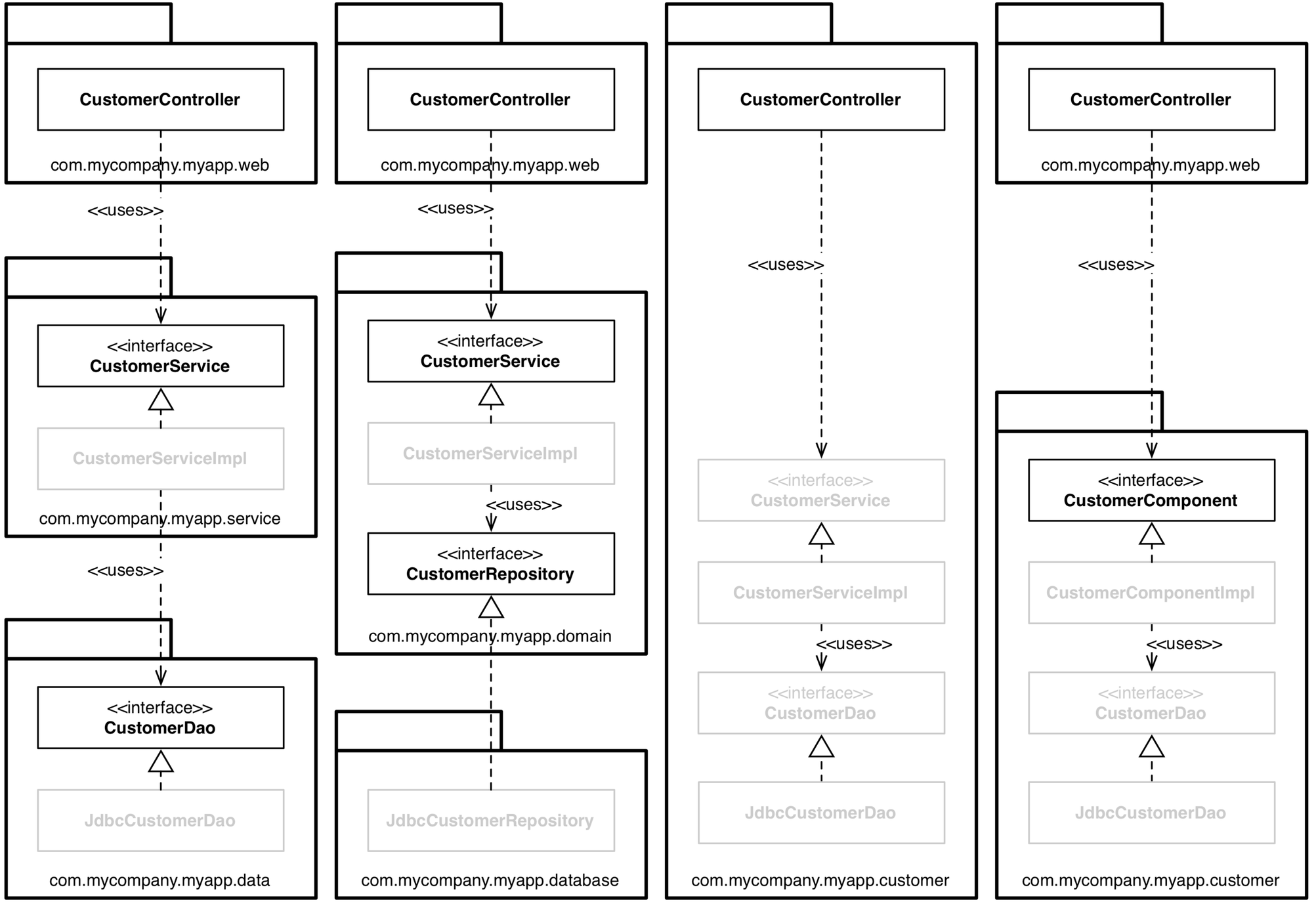

CustomerController
com.mycompany.myapp.web
<<uses>>
<<interface>>
CustomerService
CustomerServiceImpl
com.mycompany.myapp.service
<<uses>>
<<interface>>
CustomerDao
JdbcCustomerDao
com.mycompany.myapp.data
CustomerController
com.mycompany.myapp.web
<<uses>>
<<interface>>
CustomerService
CustomerServiceImpl
<<uses>>
<<interface>>
CustomerRepository
com.mycompany.myapp.domain
JdbcCustomerRepository
com.mycompany.myapp.database
CustomerController
<<uses>>
<<interface>>
CustomerService
CustomerServiceImpl
<<uses>>
<<interface>>
CustomerDao
JdbcCustomerDao
com.mycompany.myapp.customer
CustomerController
com.mycompany.myapp.web
<<uses>>
<<interface>>
CustomerComponent
CustomerComponentImpl
<<uses>>
<<interface>>
CustomerDao
JdbcCustomerDao
com.mycompany.myapp.customer

If all types are `public`,
Java packages are about
organisation of code
rather than encapsulation


CustomerController
<<uses>>
<<interface>>
CustomerService
CustomerServiceImpl
<<uses>>
<<interface>>
CustomerDao
JdbcCustomerDao
CustomerController
<<uses>>
<<interface>>
CustomerService
CustomerServiceImpl
<<uses>>
<<interface>>
CustomerRepository
JdbcCustomerRepository
CustomerController
<<uses>>
<<interface>>
CustomerService
CustomerServiceImpl
<<uses>>
<<interface>>
CustomerDao
JdbcCustomerDao
CustomerController
<<uses>>
<<interface>>
CustomerComponent
CustomerComponentImpl
<<uses>>
<<interface>>
CustomerDao
JdbcCustomerDao

The four implementation strategies are conceptually different, but syntactically the same

Let’s bring back the packages and mark (fade) those types that can be made more restrictive


CustomerController
com.mycompany.myapp.web
<<uses>>
<<interface>>
CustomerService
CustomerServiceImpl
com.mycompany.myapp.service
<<uses>>
<<interface>>
CustomerDao
JdbcCustomerDao
com.mycompany.myapp.data
CustomerController
com.mycompany.myapp.web
<<uses>>
<<interface>>
CustomerService
CustomerServiceImpl
<<uses>>
<<interface>>
CustomerRepository
com.mycompany.myapp.domain
JdbcCustomerRepository
com.mycompany.myapp.database
CustomerController
<<uses>>
<<interface>>
CustomerService
CustomerServiceImpl
<<uses>>
<<interface>>
CustomerDao
JdbcCustomerDao
com.mycompany.myapp.customer
CustomerController
com.mycompany.myapp.web
<<uses>>
<<interface>>
CustomerComponent
CustomerComponentImpl
<<uses>>
<<interface>>
CustomerDao
JdbcCustomerDao
com.mycompany.myapp.customer

Testing

“In the early days of computing when computers were slow, unit tests gave the developer more immediate feedback about whether a change broke the code instead of waiting for system tests to run. Today, with cheaper and more powerful computers, that argument is less persuasive.”

RBCS

TIME TESTED.
TESTING IMPROVED.
www.RBCS-US.com

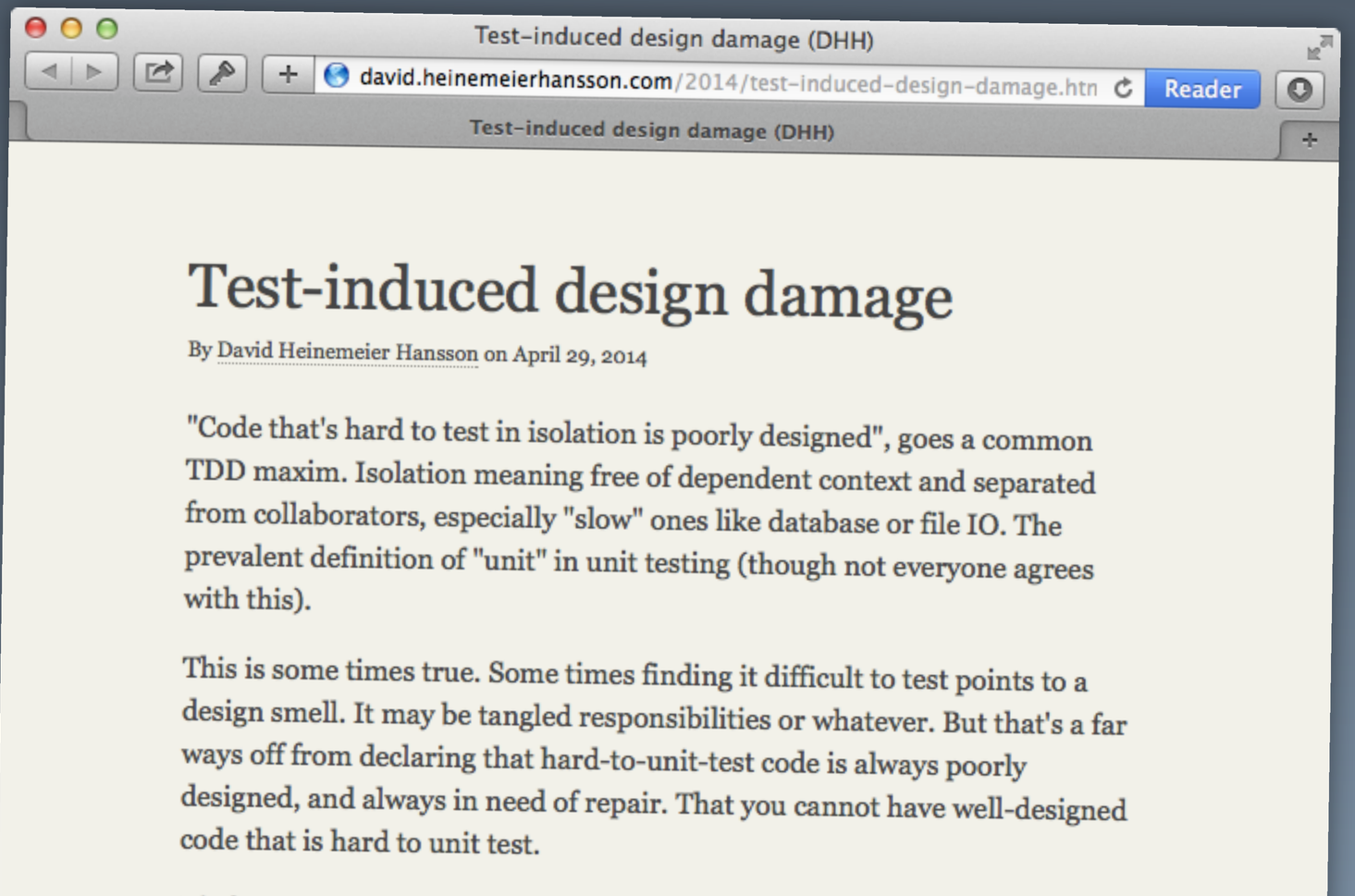
Why Most Unit Testing is Waste

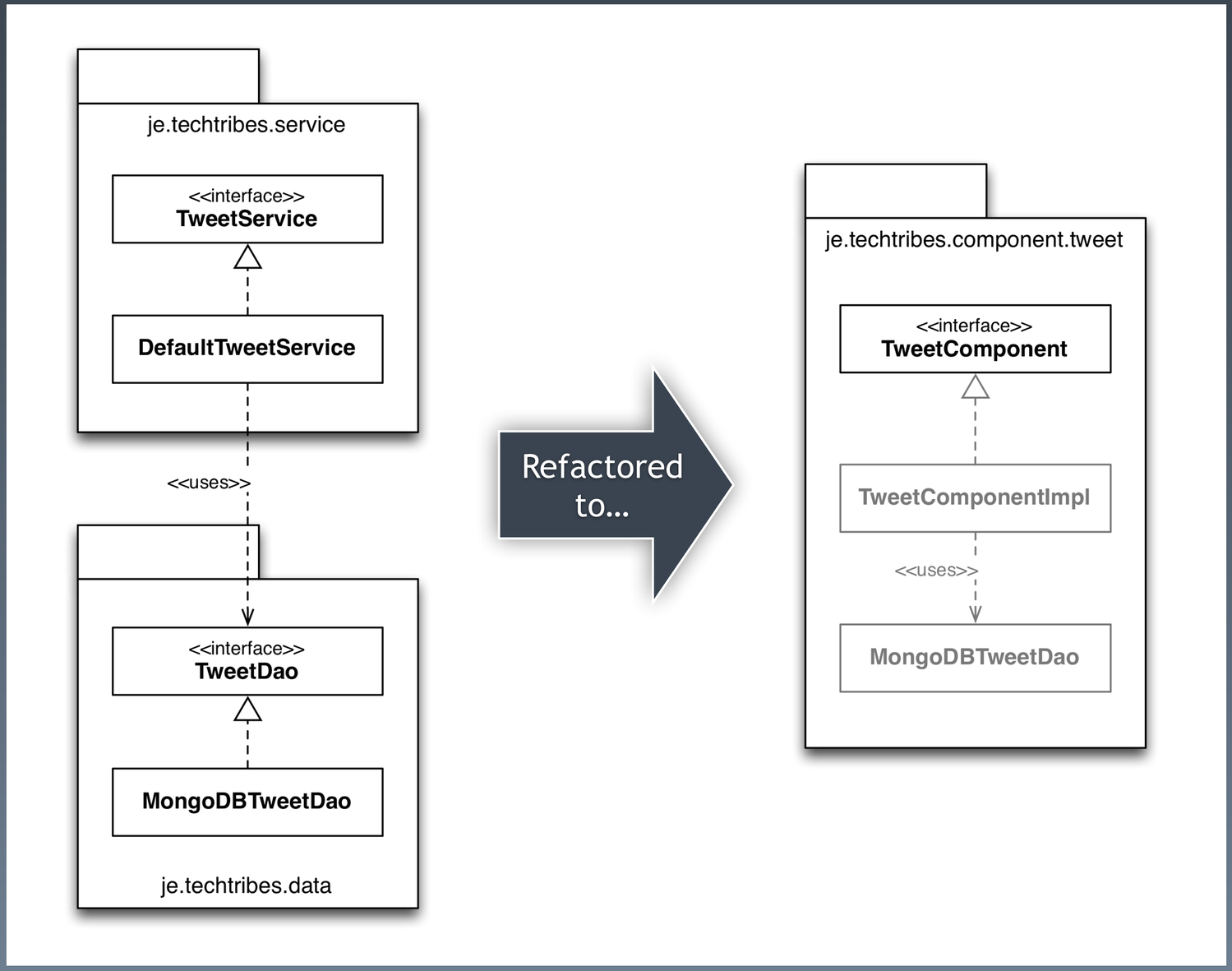
By James O Coplien

1.1 Into Modern Times

Unit testing was a staple of the FORTRAN days, when a function was a function and was sometimes worthy of functional testing. Computers computed, and functions and procedures

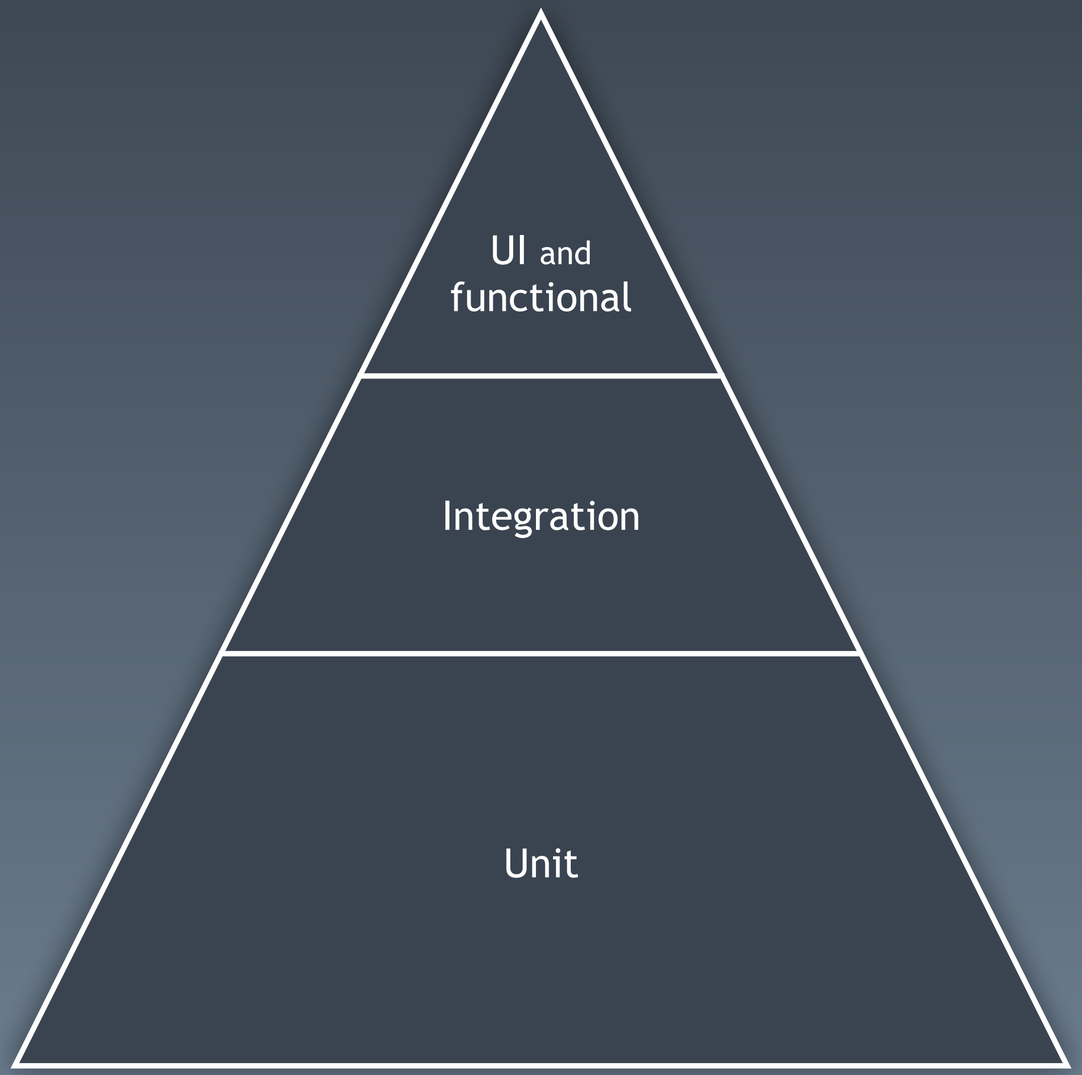
“do not let your tests drive your design”



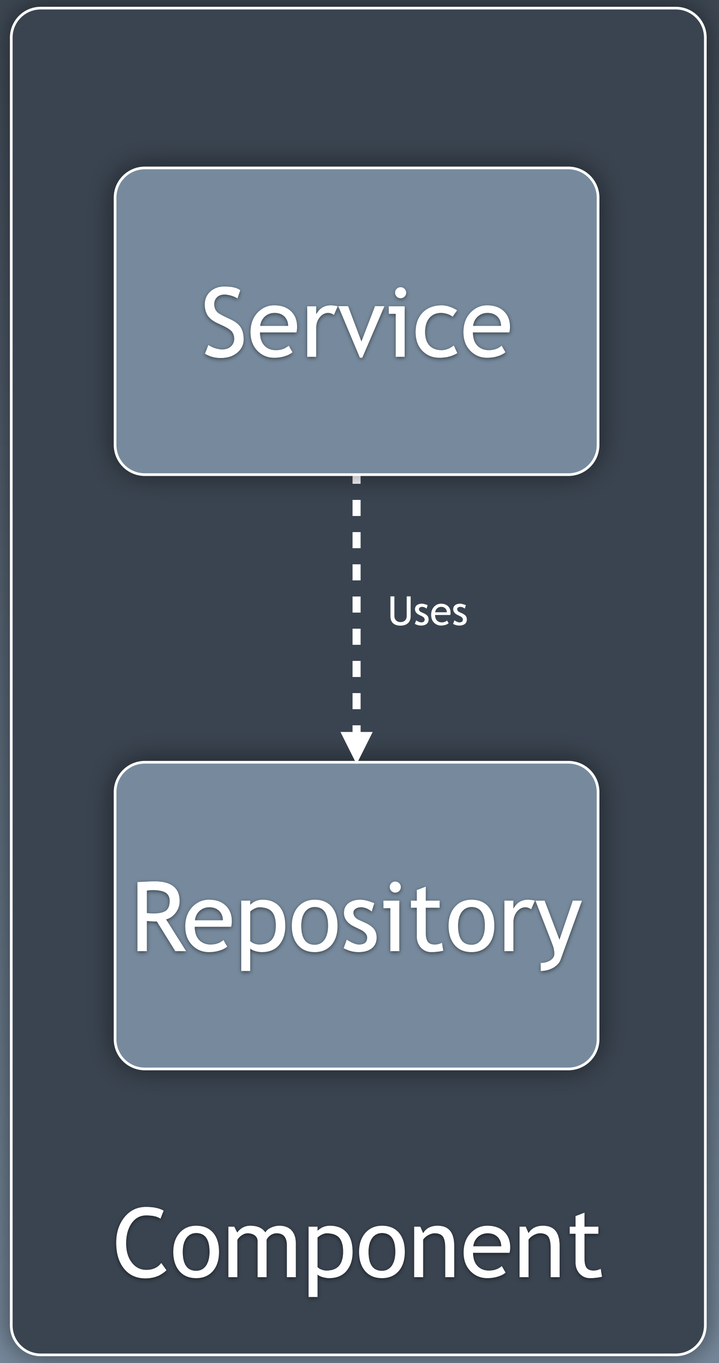


An example of an architecturally-evident coding style

Do we also
cargo cult
“unit testing”?



Do we need to rethink the testing pyramid?



Test the service in isolation?

(“unit tests”, with a mock repository)

Test the repository in isolation?

(“integration tests”, against the database)

Or test the component through the `public interface`?

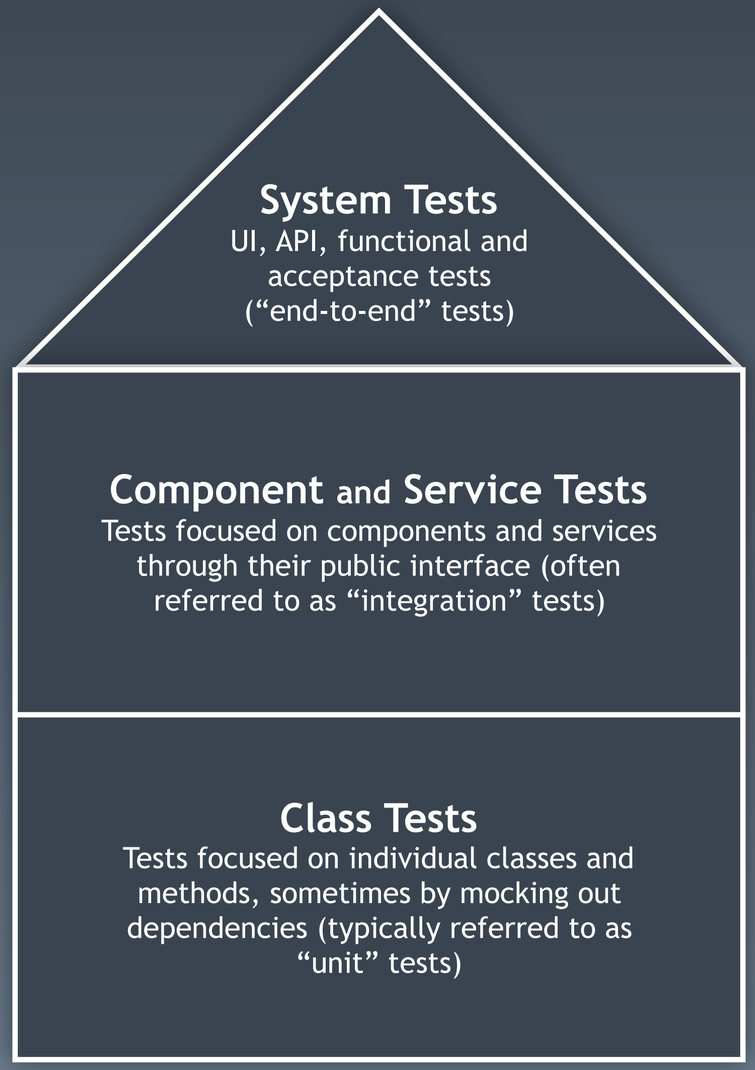
(“component tests”)

Instead of blindly unit testing everything, what about

testing your significant structural elements as black boxes?

Caveats apply!

Component testing doesn’t necessarily suit asynchronous behaviour or integration with third-party systems outside of your control

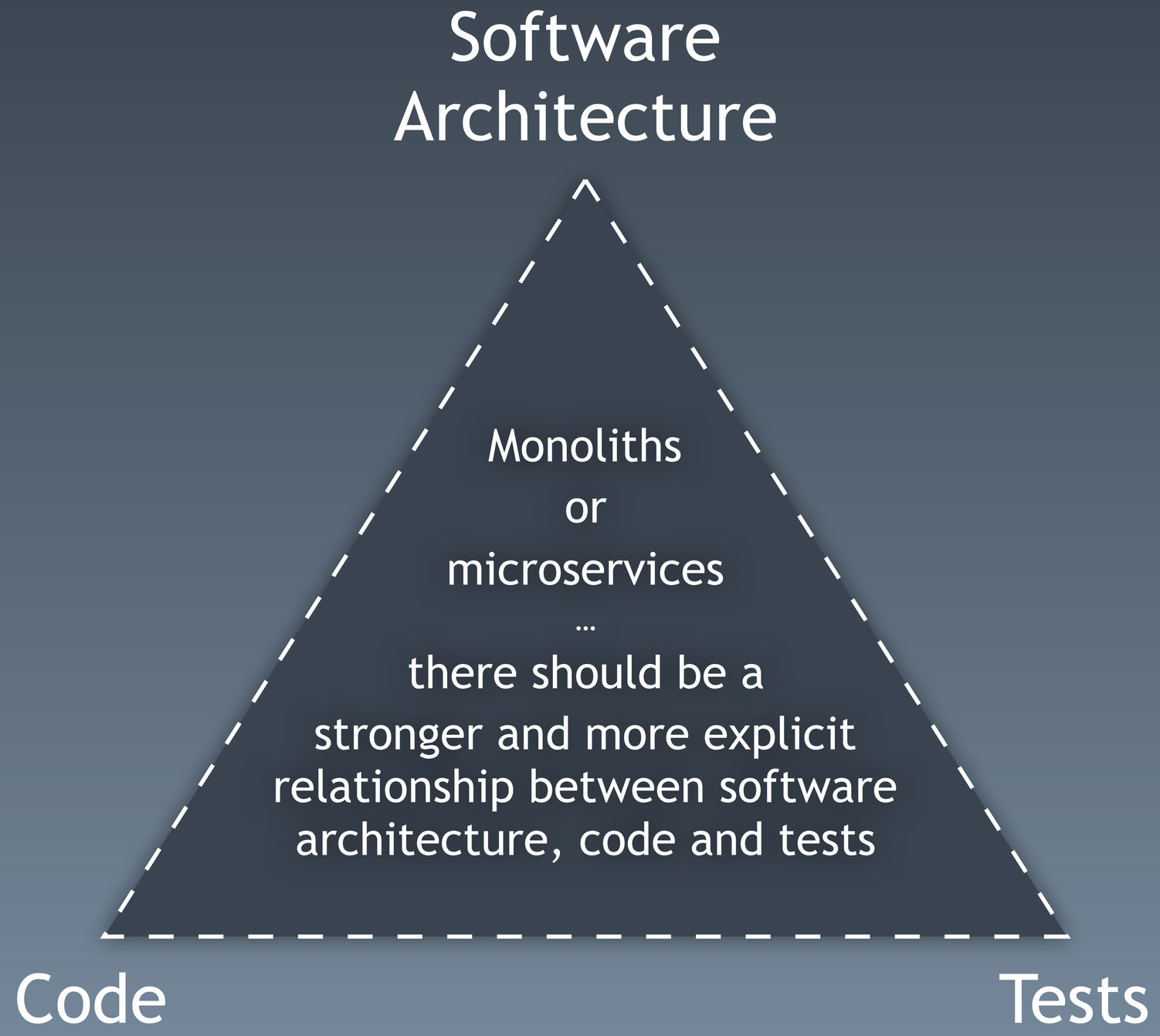


Architecturally-aligned testing

(and a reshaped testing pyramid)

I want to achieve maximum code coverage from a minimum amount of test code

(unit testing everything is one extreme,
writing one gigantic acceptance test is the other
… there is a sweet spot somewhere in-between)


Software
Architecture
Monoliths
or
microservices
…
there should be a
stronger and more explicit
relationship between software
architecture, code and tests
Code
Tests

Why?

Maintainability is
inversely proportional
to the number of

[`public` classes
dependencies
microservices]

A good architecture enables *agility*

Agility

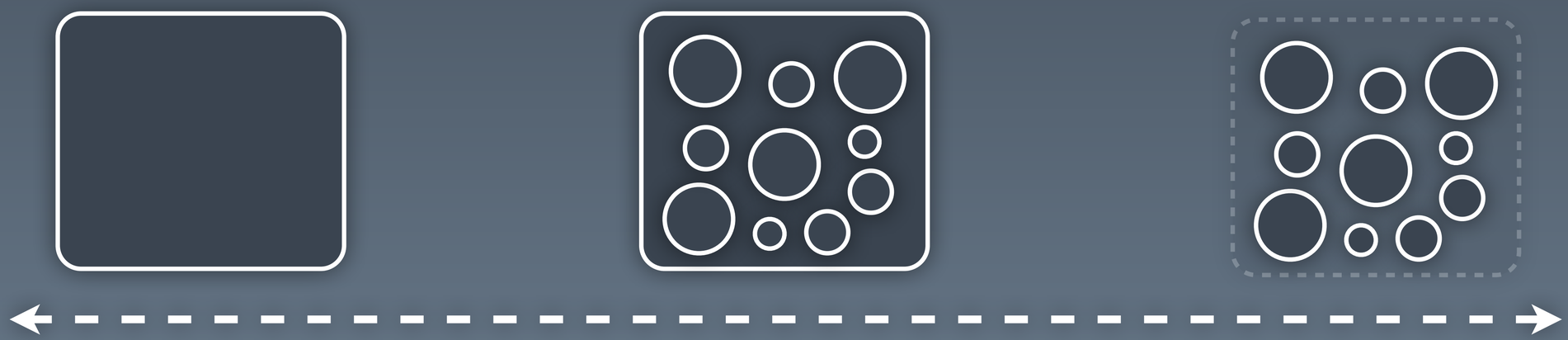
is a

quality attribute

Monolithic architecture

Service-based architecture

(SOA, microservices, etc)

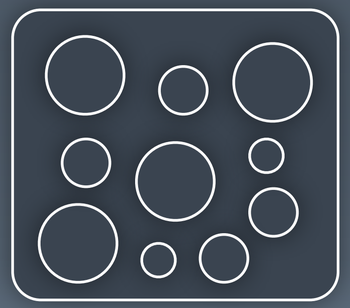


Something in between

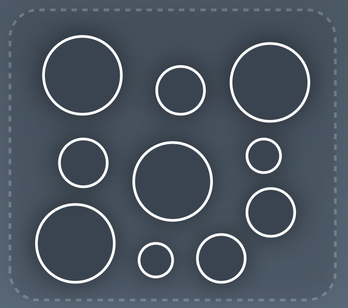
(components)

Well-defined, in-process components is a stepping stone to out-of-process components

(i.e. microservices)



From components to microservices



High cohesion
Low coupling
Focussed on a business capability
Bounded context or aggregate
Encapsulated data
Substitutable
Composable

<- All of that plus

Individually deployable
Individually upgradeable
Individually replaceable
Individually scalable
Heterogeneous technology stacks

Choose microservices for the benefits,

not because your monolithic codebase is a mess

Microservices … also known as a

distributed big ball of mud

If your software system
is hard to work with,

change it!

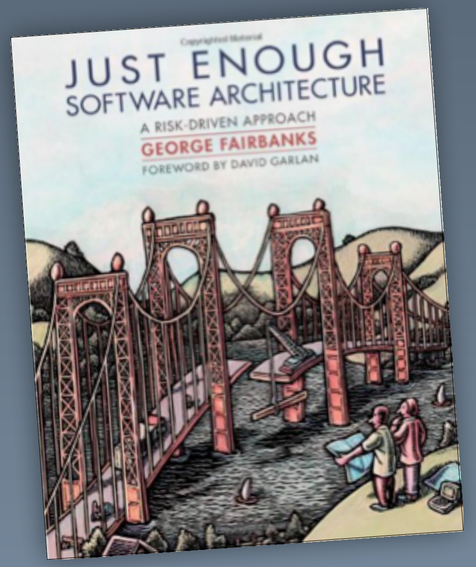
Think about how to align the

software architecture

and the

code

Be conscious of the software architecture model … adopt an architecturally-evident coding style



Stop making every class

`public`

£1 charity donation
every time you type

```
public class
```

without thinking :-)

If you can't build a

modular monolith,

what makes you think
microservices is the answer?

simon.brown@codingthearchitecture.com
@simonbrown on Twitter