Software Architecture

as Code

Simon Brown
@simonbrown
Any recommendations for software for drawing software architecture but not MS Visio?
I help software teams understand software architecture, technical leadership and the balance with agility.
Software architecture needs to be more accessible
A developer-friendly guide to software architecture, technical leadership and the balance with agility
I code too
The intersection between software architecture and code
How do we communicate software architecture?
The tension between software architecture and code
It’s usually difficult to show the entire design on a single diagram.

Different views of the design can be used to manage complexity and highlight different aspects of the solution.
Software architecture deals with abstraction, with decomposition and composition, with style and esthetics.

To describe a software architecture, we use a model composed of multiple views or perspectives.
The description of an architecture—the decisions made—can be organized around these four views, and then illustrated by a few selected use cases, or scenarios which become a fifth view. The architecture is in fact partially evolved from these scenarios as we will see later.

![Diagram](image)

**Figure 1 — The “4+1” view model**

We apply Perry & Wolf's equation independently on each view, i.e., for each view we define the set of elements to use (components, containers, and connectors), we capture the forms and patterns that work, and we capture the rationale and constraints, connecting the architecture to some of the requirements.
Do the names of those views make sense?

- Conceptual vs Logical
- Process vs Functional
- Development vs Physical
- Development vs Implementation
- Physical vs Implementation
- Physical vs Deployment
Logical and development views are often separated.
Brain freeze!
9 out of 10 people don’t use UML

(in my experience)
In my experience, software teams aren’t able to effectively communicate the software architecture of their systems.
Choose your own adventure
Would we code it that way?
Did we code it that way?
Abstraction is about reducing detail rather than creating a different representation.
Abstractions help us reason about a big and/or complex software system.
Does your code reflect the abstractions that you think about?
We often think in components but write classes (usually in layers)
Package by layer (horizontal slicing)
Chapter 1

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
- Core 00 model

Persistent domain object (entity)

EIS tier
- Databases, other transactional resources

RDBMS
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 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.**
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,

```
<table>
<thead>
<tr>
<th>package</th>
<th>contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>main/</td>
<td></td>
</tr>
<tr>
<td>main/java</td>
<td></td>
</tr>
<tr>
<td>main/java/mygroup</td>
<td></td>
</tr>
<tr>
<td>main/java/mygroup/controller</td>
<td>mojo, controllers, views</td>
</tr>
<tr>
<td>main/java/mygroup/dao</td>
<td></td>
</tr>
<tr>
<td>main/java/mygroup/model</td>
<td></td>
</tr>
<tr>
<td>main/resources</td>
<td></td>
</tr>
<tr>
<td>webapp/index.html</td>
<td></td>
</tr>
<tr>
<td>webapp/META-INF</td>
<td></td>
</tr>
<tr>
<td>webapp/resources</td>
<td></td>
</tr>
<tr>
<td>webapp/webapp</td>
<td></td>
</tr>
<tr>
<td>webapp/webapp/app</td>
<td></td>
</tr>
<tr>
<td>webapp/webapp/app/app</td>
<td></td>
</tr>
</tbody>
</table>
```

This structure fits the pattern of a Spring MVC application.
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 temp:"
            return View();
        }

        public ActionResult About()
        {
            ViewBag.Message = "Your app description here."
            return View();
        }
    }
}
Package by layer (horizontal slicing)
Hexagons and onions
Should layers be considered harmful?
Are layers significant structural elements or just an implementation detail?
Package by feature (vertical slicing)
Organisation of code vs the architectural views
“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.
Sketches get out of date, so why not auto-generate the diagrams?
Diagramming tools see packages and classes rather than components.
Describing software architecture
“Modules, components and connectors”

Is this vocabulary in common use?
A common set of abstractions is more important than a common notation.
Agree on a simple set of abstractions that the whole team can use to communicate.
The C4 model

System Context
The system plus users and system dependencies

Containers
The overall shape of the architecture and technology choices

Components
Logical components and their interactions within a container

Classes
Component or pattern implementation details
Context

• What are we building?

• Who is using it?
  (users, actors, roles, personas, etc)

• How does it fit into the existing IT environment?
  (systems, services, etc)
Containers

- What are the high-level technology decisions? (including responsibilities)
- How do containers communicate with one another?
- As a developer, where do I need to write code?
Components

- What components/services is the container made up of?
- Are the technology choices and responsibilities clear?
A notationless notation

(whiteboard and sticky note friendly, supplemented with colour coding)

My Web Application

[Container: Apache Tomcat 7.x]

Here is a list of the key responsibilities for my web application.
System Context
The system plus users and system dependencies

Containers
The overall shape of the architecture and technology choices

Components
Logical components and their interactions within a container

Classes
Component or pattern implementation details

Shneiderman’s mantra
Overview first
Zoom and filter
Details on demand
Diagrams are maps that help a team navigate a complex codebase.
Think about the target audience

Non-technical → Semi-technical → Very technical
C4++

Enterprise context
User interface mockups and wireframes
Domain model
Sequence and collaboration diagrams
Business process and workflow models
Infrastructure model
Deployment model
...

4+1 architectural view model

Philippe Kruchten

Software Systems Architecture
Working with Stakeholders Using Viewpoints and Perspectives (2nd Edition)

Nick Rozanski and Eoin Woods
Static Model
(at different levels of abstraction)

- Runtime/Behavioural
- Data
- Operation & Support
- Deployment
- Infrastructure
C4 is about the static structure of software, which is ultimately about code.
Software developers are the most important stakeholders of software architecture.
Components?
What is a “component”?
Spring PetClinic
https://github.com/spring-projects/spring-petclinic/

https://speakerdeck.com/michaelisvy/spring-petclinic-sample-application
An auto-generated UML class diagram
What are the architecturally significant elements?
A UML class diagram showing architecturally significant elements
A component diagram, based upon the code
The intersection of software architecture and code
Merge the code and the model?
Abstractions on diagrams should reflect the code
“architecturally-evident coding style”
(subclassing, naming conventions, module dependencies, package structure, ...)
What's a “component”?

A component is often a combination of a number of classes in different layers.
Package by component

- Controller A
  - Feature Set A
  - Service A
    - Component A
    - Data Access A
- Controller B
  - Feature Set B
  - Service B
    - Component B
    - Data Access B
What’s a “component”? A component is often a combination of a number of classes in different layers.
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...

    /**
     * Gets the most recent tweets for a given content source.
     */
    List<Tweet> getRecentTweets(ContentSource...

    /**
     * Gets the most recent tweets for a given collection.
     */
    List<Tweet> getRecentTweets(Collection<

    /**
     * Calculates how many tweets a given person has.
     */
    int getTweetsByPerson...
}
Architecturally-evident coding styles include:

- **Annotations/attributes** (@Component, [Component], etc)
- **Naming conventions** (*Service*)
- **Namespacing/packaging** (com.mycompany.system.components.*)
- **Maven modules, OSGi modules, microservices, etc**
Modularity as a principle
Software architecture vs testing
(a quick aside)
“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.”
“do not let your tests drive your design”
What’s a “component”?

A component is often a combination of a number of classes in different layers.
Instead of blindly unit testing everything, what about testing your significant structural elements as black boxes?
Do we need to rethink the testing pyramid?
Architecturally-aligned testing
(and a reshaped testing pyramid)

System Tests
UI, API, functional and acceptance tests
("end-to-end" tests)

Component and Service Tests
Tests focused on components and services through their public interface (often referred to as "integration" tests)

Class Tests
Tests focused on individual classes and methods, sometimes by mocking out dependencies (typically referred to as "unit" tests)
Software architecture as code
The code is the embodiment of the architecture.
Is the architecture in the code?
In practice, architecture is embodied and recoverable from code, and many languages provide architecture-level views of the system.
Context

People
Security groups/roles in configuration files, etc.

Software Systems
Integration points, APIs, known libraries, credentials for inbound consumers, etc.

Containers
IDE projects/modules, build output (code and infrastructure), etc.

Components
Extractable from the code if an architecturally-evident coding style has been adopted.
Containers

Security groups/roles in configuration files, etc.

People

Software Systems

Integration points, APIs, known libraries, credentials for inbound consumers, etc.

Components

Extractable from the code if an architecturally-evident coding style has been adopted.

Containers

IDE projects/modules, build output (code and infrastructure), etc.
Components

Software Systems
Integration points, APIs, known libraries, credentials for inbound consumers, etc.

People
Security groups/roles in configuration files, etc.

Containers
IDE projects/modules, build output (code and infrastructure), etc.

Components
Extractable from the code if an architecturally-evident coding style has been adopted.
Extract as much of the software architecture from the code as possible, and supplement where necessary.
Create an architecture description language using code
public class SpringPetClinic {

    public static void main(String[] args) throws Exception {
        Workspace workspace = new Workspace("Spring PetClinic",
            "This is a C4 representation of the Spring PetClinic sample app (https://github.com/spring-projects/spring-petclinic/)",
            Model model = workspace.getModel();

        // create the basic model (the stuff we can't get from the code)
        SoftwareSystem springPetClinic = model.addSoftwareSystem("Spring PetClinic", "");
        Person user = model.addPerson("User", "");
        user.uses(springPetClinic, "Uses");

        Container webApplication = springPetClinic.addContainer(
            "Web Application", "", "Apache Tomcat 7.x");
        Container relationalDatabase = springPetClinic.addContainer(
            "Relational Database", "", "HSQLDB");
        user.uses(webApplication, "Uses");
        webApplication.uses(relationalDatabase, "Reads from and writes to");

        // and now automatically find all Spring @Controller, @Component, @Service and @Reposi
        ComponentFinder componentFinder = new ComponentFinder(
            webApplication, "org.springframework.samples.petclinic"
ComponentFinder componentFinder = new ComponentFinder(  
    webApplication, "org.springframework.samples.petclinic",  
    new SpringComponentFinderStrategy(),  
    new JavadocComponentFinderStrategy(new File("/Users/simon/Documents/sandbox/sp")
);  
componentFinder.findComponents();

// connect the user to all of the Spring MVC controllers
webApplication.getComponents().stream()  
    .filter(c -> c.getTechnology().equals("Spring Controller"))  
    .forEach(c -> user.uses(c, "Uses"));

// connect all of the repository components to the relational database
webApplication.getComponents().stream()  
    .filter(c -> c.getTechnology().equals("Spring Repository"))  
    .forEach(c -> c.uses(relationalDatabase, "Reads from and writes to"));

for (Component component : webApplication.getComponents()) {
    if (component.getSourcePath() != null) {
        component.setSourcePath(component.getSourcePath().replace(
            "/Users/simon/Documents/sandbox/spring/spring-petclinic/",
            "https://github.com/spring-projects/spring-petclinic/tree/master/"));
    }
}

// finally create some views
ViewSet viewSet = workspace.getViews();
SystemContextView contextView = viewSet createContextView(springPetClinic);
// finally create some views
ViewSet viewSet = workspace.getViews();
SystemContextView contextView = viewSet.createContextView(springPetClinic);
contextView.addAllSoftwareSystems();
contextView.addAllPeople();

ContainerView containerView = viewSet.createContainerView(springPetClinic);
containerView.addAllPeople();
containerView.addAllSoftwareSystems();
containerView.addAllContainers();

ComponentView componentView = viewSet.createComponentView(webApplication);
componentView.addAllComponents();
componentView.addAllPeople();
componentView.add(relationalDatabase);

// tag and style some elements
springPetClinic.addTags("Spring PetClinic");
webApplication.getComponents().stream().filter(c -> c.getTechnology().equals("Spring"))
webApplication.getComponents().stream().filter(c -> c.getTechnology().equals("Spring "));

viewSet.getStyles().add(new ElementStyle("Spring PetClinic", null, null, "#6CB33E", "w"
Structurizr for Java
(open source on GitHub)
Structurizr

No more messing with drawing tools. Create software architecture models and diagrams as code based upon the C4 software architecture model.

From code

You create a software architecture model by extracting them automatically from your code.

Anonymous User

Anybody on the web.

Aggregated User

A user or business with content that is aggregated into the web.

Administration User

A system administration user.

techtribes.je

(Views)

techtribes.je is the only way to keep up to date with IT, tech and digital sector in Jersey and Guernsey, Channel Islands.

Gets profile information and tweet from.

Gets information about public code repositories from.

Gets content using API and Atom feeds from.
Spring PetClinic - System Context

User
(Person)

Uses

Spring PetClinic
(Software System)
Spring PetClinic - Containers

User
[Person]

Web Application
[Container: Apache Tomcat 7.x]

Uses

Reads from and writes to

Relational Database
[Container: HSQLDB]
Spring PetClinic
- Web Application
- Components
A model as code provides opportunities...
...and build pipeline integration keeps software architecture models up-to-date
Software architecture really is for developers :-)
The point of this?
Maintainability is inversely proportional to the number of public classes, dependencies, and microservices.
A good architecture enables agility
Agility is a quality attribute
Monolithic architecture

Service-based architecture (SOA, micro-services, etc)

Something in between (components)
If you can’t build a structured monolith, what makes you think microservices is the answer!?
Simon Brown
@simonbrown

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

#qconlondon #DesignThinking #Modularity
Well-defined, in-process components is a stepping stone to out-of-process components (i.e. microservices)

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

From components to microservices

<- 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 :-}
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 the software architecture model is *in* the code, it can be extracted *from* the code.
$10
until May 7th
with this code

https://leanpub.com/software-architecture-for-developers

/c/saturn15