Visualising software architecture with the C4 model
Over the past decade, many teams have thrown away big design up front
Unfortunately, architectural thinking, documentation, diagramming, and modelling were also often discarded.
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Retriever</strong></td>
<td><strong>Scheduler</strong></td>
<td><strong>Auditing</strong></td>
</tr>
<tr>
<td><strong>Reference Archiver</strong></td>
<td><strong>Risk Assessment Processor</strong></td>
<td><strong>Risk Parameter Configuration</strong></td>
</tr>
</tbody>
</table>
UML?
“Not everybody else on the team knows it.”
“I’m the only person on the team who knows it.”
“You’ll be seen as old.”
“You’ll be seen as old-fashioned.”
“The tooling sucks.”
“It’s too detailed.”
“It’s a very elaborate waste of time.”
“It’s not expected in agile.”
“The value is in the conversation.”
Just use a whiteboard!
“Just use a whiteboard!”
If you’re going to use “boxes & lines”, at least do so in a **structured way**, using a **self-describing notation**
Moving fast in the same direction as a team requires good communication
To describe a software architecture, we use a model composed of multiple views or perspectives.

Architectural Blueprints - The “4+1” View Model of Software Architecture
Philippe Kruchten
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 of the "4+1" view model]

*Figure 1 — The "4+1" view model*
Why is there a separation between the logical and development views?
Our architecture diagrams don’t match the code.
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.
We lack a **common vocabulary** to describe software architecture
Figure 48. Diagram of a basic circuit.
https://en.wikipedia.org/wiki/Component_diagram
Software System

Web Application

Logging Component

Relational Database

**component**

*noun* | com·po·nent | \kəmˈpoʊ·nənt, ˈkām-, kām-

**Simple Definition of** component

: one of the parts of something (such as a system or mixture) : an important piece of something

Source: Merriam-Webster's Learner's Dictionary
When drawing software architecture diagrams, think like a software developer.
If software developers created building architecture diagrams...
A common set of abstractions is more important than a common notation.
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 **code elements**.
The C4 model for visualising software architecture

c4model.com
Diagrams are maps that help software developers navigate a large and/or complex codebase.
System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.

Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
The container diagram shows the containers that reside inside the software system boundary.
Container diagram for Internet Banking System

The container diagram for the Internet Banking System.

Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
The component diagram shows the components that reside inside an individual container.
The code level diagram shows the code elements that make up a component.
Plus some supplementary diagrams...
The C4 model is notation independent
**Personal Banking Customer**

*Person*

A customer of the bank, with personal bank accounts.

**Internet Banking System**

*Software System*

Allows customers to view information about their bank accounts, and make payments.

**API Application**

*Container: Java and Spring MVC*

Provides Internet banking functionality via a JSON/HTTPS API.

**Mainframe Banking System Facade**

*Component: Spring Bean*

A facade onto the mainframe banking system.
Lines

Favour uni-directional lines showing the most important dependencies or data flow, with an annotation to be explicit about the purpose of the line and direction.
Summarise the intent of the relationship
Show both directions when the intents are different
Beware of hiding the true story
Beware of hiding the true story
Beware of hiding the true story
Beware of hiding the true story
Key/legend

Explain shapes, line styles, colours, borders, acronyms, etc ... even if your notation seems obvious!
Arrowheads

Be careful, using different arrowheads is very subtle; readers may miss them.
Use shape, colour and size to complement a diagram that already makes sense
Be careful with icons
WordPress Hosting
How to run WordPress on AWS

WordPress is one of the world’s most popular web publishing platforms, being used to publish 27% of all websites, from personal blogs to some of the biggest news sites. This reference architecture simplifies the complexity of deploying a scalable and highly available WordPress site on AWS.

1. Static and dynamic content is delivered by Amazon CloudFront.
2. An Internet gateway allows communication between instances in your VPC and the Internet.
3. NAT gateways in each public subnet enable Amazon EC2 instances in private subnets (App & Data) to access the Internet.
4. Use an Application Load Balancer to distribute web traffic across an Auto Scaling Group of Amazon EC2 instances in multiple AZs.
5. Run your WordPress site using an Auto Scaling group of Amazon EC2 instances. Install the latest versions of WordPress, Apache web server, PHP 7, and Opcache and build an Amazon Machine Image that will be used by the Auto Scaling group launch configuration to launch new instances in the Auto Scaling group.
6. If database access patterns are read-heavy, consider using a WordPress plugin that takes advantage of a caching layer like Amazon ElastiCache (Memcached) in front of the database layer to cache frequently accessed data.
7. Simplify your database administration by running your database layer in Amazon RDS using either Aurora or MySQL.
8. Amazon EC2 instances access shared WordPress data in an Amazon EFS file system using Mount Targets in each AZ in your VPC.
9. Use Amazon EFS, a simple, highly available, and scalable network file system so WordPress instances have access to your shared, unstructured WordPress data, like pdf files, config, themes, plugins, etc.
Use icons to supplement text, not replace it.
Increase the **readability** of software architecture diagrams, so they can **stand alone**
Any narrative should complement the diagram rather than explain it.
# Notation, notation, notation

A software architecture diagram review checklist

<table>
<thead>
<tr>
<th>General</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does the diagram have a title?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Do you understand what the diagram type is?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Do you understand what the diagram scope is?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Does the diagram have a key/legend?</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Abstractions first, notation second

Ensure that your team has a ubiquitous language to describe software architecture
Thank you!