Visualising software architecture with the C4 model

@simonbrown
What is software architecture?
Structure

The definition of software in terms of its building blocks and their interactions
Vision

The process of architecting; making decisions based upon business goals, requirements and constraints, plus being able to communicate this to a team.
Enterprise Architecture
Structure and strategy across people, process and technology

System Architecture
High-level structure of a software system
(software and infrastructure)

Application Architecture
The internal structure of an application
As a noun, **design is the named structure** or behaviour of a system whose presence resolves ... a force on that system. A design thus represents one point in a potential decision space.

Grady Booch
All architecture is design, but not all design is architecture.

Grady Booch
Architecture represents the significant decisions, where significance is measured by cost of change.

Grady Booch
Design a software solution for the "Financial Risk System", and draw one or more architecture diagrams to describe your solution.
Did you find anything about this exercise challenging?
Challenging?

Level of detail needed to stop. Different backgrounds, different implementation.
Easy to get bogged down in detail.
Type of diagrams.
Notation.
Documenting assumptions.

10 Challenging?

- Verifying our assumptions
- Expressing the solution in a clear way
- Use of notation
- Easy to mix levels of abstraction
- How much detail?
Swap and review your diagrams

Focus on the diagrams rather than the solution itself; do you understand the notation, colour coding, symbols, etc?
The perfection game

We rate the diagrams... (1-10)

We liked...

To make the diagrams perfect...
Information is likely still stuck in your heads
This doesn’t make sense, but we’ll explain it.
• What is this shape/symbol?
• What is this line/arrow?
• What do the colours mean?
• What level of abstraction is shown?
• Which diagram do we read first?
Moving fast in the same direction as a team requires good communication
I’ve run diagramming workshops in 30+ countries for 10,000+ people
Software architects struggle to communicate software architecture
Do you use UML?
In my experience, optimistically, 1 out of 10 people use UML.
Who are the stakeholders that you need to communicate software architecture to; what information do they need?
There are many different audiences for diagrams and documentation, all with different interests (software architects, software developers, operations and support staff, testers, Product Owners, project managers, Scrum Masters, users, management, business sponsors, potential customers, potential investors, ...).
The primary use for diagrams and documentation is communication and learning.
Software architecture diagrams, when connected to the code, are also a fantastic tool for architectural improvement.
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.

Figure 1 — The “4+1” view model
“Viewpoints and Perspectives”
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.
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].

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 35250
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

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].

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 35250
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

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].

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 35250
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

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 lack a **common vocabulary**
to describe software architecture
Figure 48. Diagram of a basic circuit.
The construction of low-level components into larger parts of the system are shown on the Component diagram.

Interfaces are modeled on the diagram using the Interface or Exposed Interface elements.

The collection of some components within another are modeled here.
Software System

Web Application

Logging Component

Relational Database

**component**

*noun*  |  com·po·nent  |  

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
Ubiquitous language
Abstractions
Would you code it that way?

(ensure that your diagrams reflect your implementation intent)
When drawing software architecture diagrams, think like a software developer.
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**.
Static structure diagrams
1. System Context
The system plus users and system dependencies.

2. Containers
The overall shape of the architecture and technology choices.

3. Components
Logical components and their interactions within a container.

4. Classes (or Code)
Component implementation details.
Diagrams are maps that help software developers navigate a large and/or complex codebase.
A model of the static structure forms the basis for other views.
Example
(techtribes.je)
techtribes.je

A simple content aggregator for the local tech and digital industry
Anonymous User
[Person]
Anybody on the web.

Aggregated User
[Person]
A user or business with content that is aggregated into the website, signed in using their Twitter ID.

Administration User
[Person]
A system administration user, signed in using a Twitter ID.

View people, tribes (businesses, communities and interest groups), content, events, jobs, etc from the local tech, digital and IT sector.

Manage user profile and tribe membership.

Add people, add tribes and manage tribe membership.

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

Gets profile information and tweets from.

Gets content using RSS and Atom feeds from.

Gets information about public code repositories from.

Twitter
[Software System]

GitHub
[Software System]

Blogs
[Software System]

[System Context] techtribes.je
#### Interface TweetComponent

**Abstract**

- + TweetComponent

**Concrete Implementations**

- # TweetComponentImpl
  - «uses»
  - «throws»
  - # MongodbTweetDao

**Exception**

- + TweetException

**Package**

je.techtribes.component.tweet
Notation
Titles

Short and meaningful, include the **diagram type**, numbered if diagram order is important; for example:

**System Context diagram** for Financial Risk System

[**System Context**] Financial Risk System
Layout

Sticky notes and index cards (e.g. CRC cards) make a great substitute for hand-drawn boxes, especially if you don’t have a whiteboard.
Orientation

Most important thing in the middle; try to be consistent across diagrams
Acronyms

Be wary of using acronyms, especially those related to the business/domain that you work in.
Elements

Start with simple boxes containing the element name, type, technology (if appropriate) and a description/responsibilities
<table>
<thead>
<tr>
<th>Anonymous User</th>
<th>techtribes.je</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Person]</td>
<td>[Software System]</td>
</tr>
<tr>
<td>Anybody on the web.</td>
<td>techtribes.je is the only way to keep up to date with the IT, tech and digital sector in Jersey and Guernsey, Channel Islands.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Web Application</th>
<th>Twitter Connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Container: Java + Spring MVC]</td>
<td>[Component: Spring Bean + Twitter4j]</td>
</tr>
<tr>
<td>Allows users to view people, tribes, content, events, jobs, etc from the local tech, digital and IT sector.</td>
<td>Retrieves profile information and tweets (using the REST and Streaming APIs).</td>
</tr>
</tbody>
</table>
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.
Dependency vs data flow

Financial Risk System
[Software System]

Trade Data System
[Software System]

Gets trade data from

Trade Data System
[Software System]

Financial Risk System
[Software System]

Sends trade data to
Read the relationship out loud

Web Application [Container]

Database [Container]

Reads **from** and writes **to**
Key/legend

Explain shapes, line styles, colours, borders, acronyms, etc
... even if your notation seems obvious!
Use shape, colour and size to complement a diagram that already makes sense
Increase the **readability** of software architecture diagrams, so they can **stand alone**.
Any narrative should complement the diagram rather than explain it.
Frequently asked questions
What's the inspiration behind the C4 model?
Why "container"?
Can we change the terminology?
How do you model microservices and serverless?
How do you diagram large and complex software systems?
Option 1: Introduce additional abstractions
(nested components, sub-components, layers, etc)
Option 2: Partition the diagrams
(one diagram per business concept, feature set, bounded context, aggregate root, vertical slice, etc)
Will the diagrams become outdated quickly?
Why doesn't the C4 model cover business processes, workflows, state machines, domain models, data models, etc?
The C4 model vs UML, ArchiMate and SysML?
Does the C4 model imply a design process or team structure?
Draw System Context and Container diagrams to describe your software system
Abstractions first, notation second

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

simon.brown@codingthearchitecture.com

@simonbrown