Visualising software architecture with the C4 model

@simonbrown
Software Architecture for Developers

Visualise, document and explore your software architecture

Simon Brown

https://leanpub.com/visualising-software-architecture/c/...
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
Draw one or more software architecture diagrams to describe the system/project you are currently working on.
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?
level of detail
- audience
- backgrounds
- easy to get bogged down

Type of diagrams
- notation
- documenting assumptions

7 Challenging
- needed to ask questions
- make assumptions
- temptation to focus on detail
- when do we stop?
- how much detail?
- talked about more than the diagrams
- what notation? - arrows
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?
FUNCTIONAL VIEW

- File Retriever
- Scheduler
- Auditing
- Reference Archiver
- Risk Assessment Processor
- Risk Parameter Configuration
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.
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
{gumurph, 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. For example, this is reported by Wong et al. [WTMS95].

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.

Web Server
- DoRequest()
- ReturnResponse()
- ForwardRequest()
- AcceptRequest()

Firewall
- TransformXML()

XSL Stylesheets

ASP Pages

COM+ Component Environment
- Business Logic

Oracle Database
- ProcessSQLRequest()
Software System

Web Application

Logging Component

Relational Database

**component**

*noun* | com-pö-nent | \kəm-'pö-nənt\, 'kām-, kām-'\n
**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
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.

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.

Twitter
[Software System]

GitHub
[Software System]

Blogs
[Software System]

Gets profile information and tweets from.

Gets content using RSS and Atom feeds from.

Gets information about public code repositories from.

[System Context] techtribes.je
Relational Database
[Container: MySQL 5.5.x]
Reads from and writes data to (SQ/LDBC, port 3306)

File System
[Container]
Writes to

NoSQL Data Store
[Container: MongoDB 2.2.x]
Reads from and writes data to (Mongo DB Wire Protocol, port 27017)

GitHub Component
[Component: Spring Bean and [DB]]
Provides access to GitHub repos.

Search Component
[Component: Spring Bean and Apache Lucene]
Search facilities for news feed entries and tweets.

News Feed Entry Component
[Component: Spring Bean and MongoDB]
Provides access to blog entries and news.

Tweet Component
[Component: Spring Bean and MongoDB]
Provides access to tweets.

Scheduled Content Updater
[Component: Spring Scheduled Task]
Refreshes information from external systems every 15 minutes.

Logging Component
[Component: Spring Bean and log4j]
Provides logging facilities to all other components.

Twitter Connector
[Component: Spring Bean and Twitter4J]
Retrieves profile information and tweets (using the REST and Streaming APIs).

GitHub Connector
[Component: Spring Bean and Eclipse Mylyn]
Retrieves information about public repos.

News Feed Connector
[Component: Spring Bean and ROME]
Retrieves content from RSS and Atom feeds.

Twitter
[Software System]
Gets profile information and tweets from [HTTPS]

GitHub
[Software System]
Gets information about public code repositories from [HTTPS]

Blogs
[Software System]
Gets content using RSS and Atom feeds from [HTTP]

*Used by all components

[Components] techtribes.je - Content Updater
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.
Anonymous User  
[Person]  
Anybody on the web.

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

Web Application  
[Container: Java + Spring MVC]  
Allows users to view people, tribes, content, events, jobs, etc from the local tech, digital and IT sector.

Twitter Connector  
[Component: Spring Bean + Twitter4j]  
Retrieves profile information and tweets (using the REST and Streaming APIs).
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] sends data to Trade Data System [Software System].

Read the relationship out loud:

Web Application [Container] \(\rightarrow\) 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.
Draw a Component diagram to describe a container

(components within the container in scope)
Designing software is where the complexity should be, not communicating it!
Richer diagrams lead to richer design discussions
The perfection game

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

We liked...

To make the diagrams perfect...
Similar levels of abstraction provide a way to easily compare solutions.
C4 and UML
A sample application that illustrates how to build Java web applications using the Spring MVC framework

https://github.com/spring-projects/spring-petclinic/

View components:
- JSP with custom tags || Thymeleaf
- + Bootstrap (CSS) && webjars && Dandelion

Controller components:
- Spring @MVC annotations
- Bean Validation
- @Cacheable
- @Transactional

Service components:
- Spring Data JPA
- default (JPA)
- jdbc

Repository components:
- 3 profiles
- Spring-Data-JPA
- Service
- @Repository
- @Transactional

https://speakerdeck.com/michaelisvy/spring-petclinic-sample-application
System Context diagram for Spring PetClinic
The System Context diagram for the Spring PetClinic system.
Last modified: Monday 26 June 2017 12:59 UTC | Version: 950e1d9f8bf63560915331664b27a4a75e1f1f6
Container diagram for Spring PetClinic

The Container diagram for the Spring PetClinic system.

Last modified: Monday 26 June 2017 12:59 UTC | Version: 95de1d9f8bf63560915331664b27a4a75e1f1f6
Clinic Employee + Uses «HTTP»

«Software System»
Spring PetClinic

«Container»
Web Application
(from Spring PetClinic)

 Allows employees to view and manage information regarding the veterinarians, the clients, and their pets.

«Container»
Database
(from Spring PetClinic)

 Stores information regarding the veterinarians, the clients, and their pets.

«JDBC»
+ Reads from and writes to
System landscape diagrams
Runtime/behavioural diagrams
Static structure diagrams are very useful, but they don’t tell the whole story.
View list of vets

Clinic Employee

VetController

ClinicService

VetRepository

Relational Database

Clinic Employee

VetController

ClinicService

VetRepository

Relational Database

Requests list of vets from /vets

Calls findVets

Calls findAll

select * from vets
Dynamic diagram

Shows how the "view list of vets" feature works.

Last modified: Monday 26 June 2017 12:59 UTC  |  Version: 95de1d9f80f63560915331664b27a4a75ce1f1f6
Customer Service
[Container: Java and Spring Boot]
The point of access for customer information.

Message Bus
[Container: RabbitMQ]
Transport for business events.

Reporting Service
[Container: Ruby]
Creates normalised data for reporting purposes.

Audit Service
[Container: C# .NET]
Provides organisation-wide auditing facilities.

Customer Database

Reporting Database

Audit Store
Use dynamic diagrams to describe patterns or complex interactions
Deployment diagrams
Deployment is about the mapping of containers to infrastructure.
Deployment Node

Physical infrastructure (a physical server or device),
virtualised infrastructure (IaaS, PaaS, a virtual machine),
containerised infrastructure (a Docker container),
database server, Java EE web/application server,
Microsoft IIS, etc
A deployment node can contain other deployment nodes or container instances.
Deployment diagram for Spring PetClinic
An example staging deployment scenario for the Spring PetClinic software system.
Last modified: Monday 26 June 2017 12:59 UTC | Version: 95de1d9f8b63560915331664b27a4a75ce1ff16
**Deployment diagram for Spring PetClinic**

An example development deployment scenario for the Spring PetClinic software system.

Last modified: Monday 26 June 2017 12:59 UTC | Version: 95de1d98b63560915331664b27a4a75ce1f1f6
Deployment diagram for Spring PetClinic

An example live deployment scenario for the Spring PetClinic software system.

Last modified: Monday 26 June 2017 12:59 UTC | Version: 956e1d9f8b63560915331664b27a4a75ce1f1f6
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?
Can we combine C4 and arc42?
Does the C4 model imply a design process or team structure?
Draw a System Context diagram to describe your software system

(people and software systems)
Draw a Container diagram to describe your software system

(deployable/runnable units within the software system in scope)
Draw System Context and Container diagrams to describe your software system

60 minutes
Tooling
What **tools** do you recommend?
Whiteboards and paper for up front design
General purpose diagramming tools allow you to draw anything
Microsoft Visio, OmniGraffle, Gliffy,
Lucidchart, draw.io, Creately, ...
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.

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 to manage

techtribes.je
Static diagrams need to be kept up to date individually.
Diagrams vs models
Text-based tools allow you to create diagrams using text
PlantUML, WebSequenceDiagrams, yUML, nomnoml, ...
Authentication Sequence

Alice -> Bob: Authentication Request

Bob thinks about it

Bob -> Alice: Authentication Response
Welcome to Structurizr Express, a tool to help you quickly create a single System Landscape, System Context, Container or Component diagram from the C4 model by defining it as text. See the help page for more information.

If you would like to create a full software architecture model with multiple diagrams, take a look at the Structurizr client libraries for java and .NET instead.

```
{
  "type": "System Context",
  "scope": "Internet Banking System",
  "description": "The system context diagram for the Internet Banking System."
}

"elements": [
  {
    "type": "Person",
    "name": "Customer",
    "description": "A customer of the bank."
  },
  {
    "type": "Software System",
    "name": "Internet Banking System",
    "description": "Allows customers to view information about their bank accounts and make payments."
  },
  {
    "type": "Software System",
    "name": "Mainframe Banking System",
    "description": "Stores all of the core banking information about customers, accounts, transactions, etc.

System Context diagram for Internet Banking System
This system context diagram for the Internet Banking System.
Last modified: Thursday 18 January 2018 11:20 UTC
```
Text-based diagrams can be **version controlled**, generated from build scripts, etc.
Text-based diagrams can’t be queried.
Architecture description languages provide a way to describe architecture as text.

Darwin, ACME, Koala, Wright, ...
Modelling tools allow you to create a **model** of your software system. Sparx Enterprise Architect, IBM Rational Software Architect, StarUML, GenMyModel, Visual Paradigm, ...
Static analysis and dependency management tools can help understand code structure

Structure101, Lattix, JArchitect, NDepend, JDepend, jQAssistant, Degraph, ...
**Modelling tools**
Sophisticated and often complex tooling to create rich models using UML, ArchiMate, SysML, etc; either by “drag & drop” or reverse-engineering code.

**Static analysis tools**
Often complex to setup; creates a model based upon code only; designed for understanding code rather than communicating software architecture.

**Text-based tools**
Very quick and easy to use; text can be version controlled; designed to create individual diagrams rather than models.

**General purpose tools**
Easy to use; very flexible; unstructured; no underlying model.

**Whiteboard**
Create diagrams easily; usually transient.
What do you see as the future of software architecture documentation?

Eoin: I hope that in the future we’ll need very little software architecture documentation because we’ll be able to see the architecture in the code and the running system! One of the reasons we need much of our architecture documentation today is because there’s no way of representing architectural structures directly using the technologies we have at our disposal. I’d love to see our architectural constructs as first class implementation structures and then architecture documentation can evolve to capture decisions, rationale and analysis, rather than just capturing structures. On the way to this nirvana, I hope that work going on in the areas of DSLs and ADLs (architecture description languages) point the more immediate way forward, as we improve our description languages, on the way to working out how to embed the information right in the running system.

Paulo: The software architecture discipline is fairly new. There is a long path ahead until we get to a point where an architect creates architecture documentation that is readily understood by a developer who has never worked with that architect. The way to get there is to let new architects learn software architecture at school rather than try-and-error in the battlefield. This education includes proper ways to represent the software architecture for other people’s consumption. Important initiatives in the direction of good software architecture education are: the work of Grady Booch on the handbook of software architecture and the publications and curriculum developed at the SEI.

Grady: There is a lot of energy being applied today with regard to architectural frameworks and methods: TOGAF, NEA, DoDAF, MoDAF, FSAM, Zachman, and so on. The good news is that there is a vibrant dialog going on with regard to these frameworks and methods - but I expect there will be a shakeout/simplification over time.

Len: The ideal development environment is one for which the documentation is available for essentially free with the push of a button. This will require an integrated development, requirements management, and project management environment. Although this will be a long time coming, it provides a worthy goal to strive for.
Do structural engineers and building architects use general purpose drawing tools?
Diagrams are not useful for architectural improvement if they are not connected to the code.
Reverse-engineer code to diagrams?
Spring PetClinic
A sample application that illustrates how to build Java web applications using the Spring MVC framework
https://github.com/spring-projects/spring-petclinic/

https://speakerdeck.com/michaelisvy/spring-petclinic-sample-application
Most tools see code, not components
Information about software architecture doesn’t exist in 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.
Examples of architecturally-evident coding styles

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

Naming conventions (*Service)

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

Maven & Gradle modules, OSGi & Java 9 modules

JavaScript module patterns, ECMAScript 6 modules, microservices, etc
Extract as much of the software architecture from the code as possible, and supplement the model where necessary.
Executable architecture description language

Structurizr for Java and .NET
Modelling tools
Sophisticated and often complex tooling to create rich models using UML, ArchiMate, SysML, etc; either by “drag & drop” or reverse-engineering code.

Static analysis tools
Often complex to setup; creates a model based upon code only; designed for understanding code rather than communicating software architecture.

General purpose tools
Easy to use; very flexible; unstructured; no underlying model.

Text-based tools
Very quick and easy to use; text can be version controlled; designed to create individual diagrams rather than models.

Whiteboard
Create diagrams easily; usually transient.

Creates a relatively rich yet simple model, in a developer-friendly way using Java/C# code, supplemented with Markdown/AsciiDoc documentation.
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();
}
SoftwareSystem springPetClinic = model.addSoftwareSystem("Spring PetClinic",
    "Allows employees to view and manage information regarding the veterinarians, the clients, and their pets.");

Person clinicEmployee = model.addPerson("Clinic Employee", "An employee of the clinic");

clinicEmployee.uses(springPetClinic, "Uses");
// containers

Container webApplication = springPetClinic.addContainer(
    "Web Application",
    "Allows employees to view and manage information regarding the
    veterinarians, the clients, and their pets.",
    "Apache Tomcat 7.x");

Container relationalDatabase = springPetClinic.addContainer(
    "Relational Database",
    "Stores information regarding the veterinarians, the clients,
    and their pets.", "HSQLDB");

clinicEmployee.uses(webApplication,
    "Uses", "HTTP");

webApplication.uses(relationalDatabase,
    "Reads from and writes to", "JDBC, port 9001");
// components
ComponentFinder componentFinder = new ComponentFinder(
    webApplication,
    "org.springframework.samples.petclinic",
    new SpringComponentFinderStrategy(
        new ReferencedTypesSupportingTypesStrategy()
    ),
    new SourceCodeComponentFinderStrategy(
        new File(sourceRoot, "/src/main/java/"), 150)
);

componentFinder.findComponents();
// connect components with other model elements

webApplication.getComponents().stream()
    .filter(c -> c.getTechnology().equals(SpringComponentFinderStrategy.SPRING_MVC_CONTROLLER))
    .forEach(c -> clinicEmployee.uses(c, "Uses", "HTTP"));

webApplication.getComponents().stream()
    .filter(c -> c.getTechnology().equals(SpringComponentFinderStrategy.SPRING_REPOSITORY))
    .forEach(c -> c.uses(relationalDatabase, "Reads from and writes to", "JDBC"));
// system context, container and component views
ViewSet viewSet = workspace.getViews();

SystemContextView contextView = viewSet createContextView(
    springPetClinic, "context", "Context view for Spring PetClinic");
contextView.addAllSoftwareSystems();
contextView.addAllPeople();

ContainerView containerView = viewSet.createComponentView(
    springPetClinic, "containers", "Container view for Spring PetClinic");
containerView.addAllPeople();
containerView.addAllSoftwareSystems();
containerView.addAllContainers();

ComponentView componentView = viewSet.createComponentView(
    webApplication, "components", "Component view for the Spring PetClinic webapp.");
componentView.addAllComponents();
componentView.addAllPeople();
componentView.add(relationalDatabase);
You can create many visualisations from a single model
Spring PetClinic - System Context

- Clinic Employee
  - An employee of the clinic

- Software System: Spring PetClinic
  - Allows employees to view and manage information regarding the veterinarians, the clients, and their pets.

The System Context diagram for the Spring PetClinic system.

Spring PetClinic - Containers

- Clinic Employee
  - An employee of the clinic

- Container: Web Application
  - Allows employees to view and manage information regarding the veterinarians, the clients, and their pets.

- Container: Database
  - Stores information regarding the veterinarians, the clients, and their pets.

The Containers diagram for the Spring PetClinic system.

Spring PetClinic - Web Application - Components

- Clinic Employee
  - An employee of the clinic

- Web Application
  - Spring MVC Controller: CrashController
    - Controller used to showcase what happens when an exception is thrown
  - Spring MVC Controller: OwnerController
  - Spring MVC Controller: PetController
  - Spring MVC Controller: VetController
  - Spring MVC Controller: VisitController
  - Spring Service: CRMService
    - Mostly used as a facade so all controllers have a single point of entry
  - Spring Repository: OwnerRepository
    - Repository class for Owner domain objects. All method names are compliant with Spring Data naming conventions so this interface can easily be extended...
  - Spring Repository: VisitRepository
    - Repository class for Visit domain objects. All method names are compliant with Spring Data naming conventions so this interface can easily be extended...
  - Spring Repository: CRMRepository
    - Repository class for CRM domain objects. All method names are compliant with Spring Data naming conventions so this interface can easily be extended...

- Database
  - Reads from and writes to the database

The Components diagram for the Spring PetClinic web application.
Structurizr

Visualise, document and explore your software architecture

Create system context diagrams using Java code.
Extract components from your Java codebase, using static analysis and reflection.
Client-side encrypt your workspace for additional security ('password').
Create animated diagrams to describe dynamic behaviour.
Create enterprise context diagrams using code.
Document your software with Markdown and AsciIDoc, effortlessly embed diagrams.

Create diagrams using text with Structurizr Express.
Create diagrams using C# code.
Extract components from your .NET codebase, using static analysis and reflection.
Explore inbound, outbound and cyclic dependencies between components.
Explore component size.
Embed diagrams into Atlassian Confluence with prebuilt macros.
// upload the software architecture model to structurizr.com

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

{
"id" : 0,
"name" : "Spring PetClinic",
"description" : "This is a C4 representation of the Spring PetClinic sample app (https://github.com/spring-projects/spring-petclinic/)",
"model" : {
  "people" : [ {
    "tags" : "Element,Person",
    "id" : "2",
    "name" : "Clinic Employee",
    "description" : "An employee of the clinic",
    "relationships" : [ {
      "tags" : "Relationship,Synchronous",
      "id" : "3",
      "sourceId" : "2",
      "destinationId" : "1",
      "description" : "Uses",
      "interactionStyle" : "Synchronous"
    }, {
      "tags" : "Relationship,Synchronous",
      "id" : "6",
      "sourceId" : "2",
      "destinationId" : "4",
      "description" : "Uses",
      "technology" : "HTTP",
      "interactionStyle" : "Synchronous"
    }, {
      "tags" : "Relationship,Synchronous",
      "id" : "28",
      "sourceId" : "2",
      "destinationId" : "8",
      "description" : "Uses",
      "technology" : "HTTP",
      "interactionStyle" : "Synchronous"
    }
  ]
}
System Context diagram for Spring PetClinic
The System Context diagram for the Spring PetClinic system.
Last modified: Monday 26 June 2017 12:59 UTC | Version: 95de1d9f80b93360915331664b27a4a75ce1f1f6
**Container diagram for Spring PetClinic**

The Container diagram for the Spring PetClinic system.

Last modified: Monday 26 June 2017 12:59 UTC | Version: 95de1d9f8b033650913331664b27a4a75ce1f1f8
Diagram key

Web Application [Container]

Container, Database

Person

Spring MVC Controller

Spring Repository

Spring Service

Relationship
Diagrams are maps that help software developers navigate a large and/or complex codebase.
Integration with your build process keeps models up to date
Once you have a model of your software system, you can explore it.
Owner
Simple JavaBean domain object representing an owner.
org.springframework.samples.petclinic.model.Owner
(Code: 133 lines, shared)
The 1990’s called and they want their tools back!

It’s 2018 and we shouldn’t be using a general purpose diagramming tool for software architecture
Abstractions first, notation second

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

simon.brown@codingthearchitecture.com
@simonbrown