Visualising software architecture with the C4 model

Simon Brown
@simonbrown
Simon Brown
Independent consultant specialising in software architecture, plus the creator of the C4 model and Structurizr

@simonbrown

Software Architecture for Developers

Volume 1
Technical leadership and the balance with agility
Simon Brown

Volume 2
Visualise, document and explore your software architecture
Simon Brown

The Software Architect Elevator
Gregor Hohpe
Foreword by Simon Brown and David Evans

Clean Architecture
A Craftsman’s Guide to Software Structure and Design
Robert C. Martin

Design It!
From Programmer to Software Architect
Michael Keating

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
As architects, we define the significant decisions.
Design a software solution for the "Financial Risk System".

1. Draw one or more **software architecture diagrams** to describe your solution, on flip chart paper.

2. Keep a record of the **significant decisions** that you make.
Did you find anything about this exercise challenging?
Challenging?
Level of detail
Where to stop different implementation
easy to get bogged down in detail
Type of diagrams
Notation
Documenting assumptions

10) Challenging?
Verifying our assumptions
Expressing the solution
- communicating it in a clear way
- use of notation
- easy to mix levels of abstraction
- how much detail?

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?

1. Things you like
2. Things that could improve the diagram(s)
3. A score between 1-10

15 minutes
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
The producer-consumer conflict of software architecture diagrams

I don’t want to put technology choices on the diagrams...

Software design should be technology independent...

I wish these diagrams included technology choices...

Producer

Consumer
Significant decisions:
- F/E <-> B/E
- Make use of OS’ watchdog mechanism
- Data storage ORM:
  - framework: Entity
- ASP.NET B/E
- Angular F/I/E
DECISIONS:
- Calculations are a job triggered on schedule.
- Execute calculations in parallel for each contingency.
- Web UI for viewing reports and modifying risk parameters.
- Authenticate and authorize users based on SSO.
- Single point of entry.

Financial Risk System

Monitoring System

Audit

SSO Broker

FINANCIAL RISK SYSTEM

Backup

Data Encryptor

Calculation Parameter Service

Report Generator

Web App
If software developers created building architecture diagrams...
Moving fast in the same direction as a team requires good communication
Do you use UML?
In my experience, optimistically,

1 out of 10 people use 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.”
If you’re using UML, ArchiMate, SysML, BPML, DFDs, etc and it’s working ... keep doing that!
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.
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 Reflection 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

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].

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.
https://en.wikipedia.org/wiki/Component_diagram
**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
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.
Abstractions
A **software system** is made up of one or more **containers** (web applications, mobile apps, desktop applications, databases, file systems, etc), each of which contains one or more **components**, which in turn are implemented by one or more **code** elements (e.g. classes, interfaces, objects, functions, etc).
Static structure diagrams
The C4 model for visualising software architecture

c4model.com
Diagrams are maps that help software developers navigate a large and/or complex codebase.
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. Code (e.g. classes)
   Component implementation details.
Example
(Internet Banking System)
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)
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)
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)
Level 2

Container diagram
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)
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)
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)
Level 3

Component diagram
The component diagram shows the components that reside inside an individual container.
Component diagram for Internet Banking System - API Application

The component diagram for the API Application.
Workspace last modified: Wed Feb 05 2020 09:33:00 GMT+0100 (Central European Standard Time)
Component diagram for Internet Banking System - API Application

The component diagram for the API Application.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Component diagram for Internet Banking System - API Application

- **Single-Page Application** (Container: JavaScript and Angular)
  - Provides all of the Internet banking functionality to customers via their web browser.
  - Makes API calls to [JSON/HTTPS]

- **Mobile App** (Container: Xamarin)
  - Provides a limited subset of the Internet banking functionality to customers via their mobile device.
  - Makes API calls to [JSON/HTTPS]

- **Sign In Controller** (Component: Spring MVC Rest Controller)
  - Allows users to sign in to the Internet Banking System.

- **Security Component** (Component: Spring Bean)
  - Provides functionality related to signing in, changing passwords, etc.

- **API Application** (Container)
  - Reads from and writes to [ODBC]

- **Database** (Container: Oracle Database Schema)
  - Stores user registration information, hashed authentication credentials, access logs, etc.

- **Accounts Summary Controller** (Component: Spring MVC Rest Controller)
  - Provides customers with a summary of their bank accounts.

- **Mainframe Banking System Facade** (Component: Spring Bean)
  - A facade onto the mainframe banking system.

- **Mainframe Banking System** (Software System)
  - Stores all of the core banking information about customers, accounts, transactions, etc.

- **E-mail System** (Software System)
  - The internal Microsoft Exchange e-mail system.

The component diagram for the API Application.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Component diagram for Internet Banking System - API Application

The component diagram for the API Application.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
Level 4

Class diagram
Notation
The C4 model is notation independent
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.
Visual consistency

Try to be consistent with notation and element positioning 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
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

No

Yes
Single Page Application [Container]

Makes an API request to

API Application [Container]

Sends an API response to

Single Page Application [Container]

Makes API calls using

API Application [Container]

Summarise the intent of the relationship
Summary:

- Single Page Application [Container]
- API Application [Container]

Diagram:

Single Page Application [Container] \(\rightarrow\) Uses \(\rightarrow\) API Application [Container]

Single Page Application [Container] \(\rightarrow\) Makes API calls using \(\rightarrow\) API Application [Container]

Summarise, yet be specific
Show both directions when the intents are different

Microservice A
[Container]

Requests a list of customers from
[JSON/HTTPS]

Sends new customers to
[Kafka topic]

Microservice B
[Container]
Beware of hiding the true story
Beware of hiding the true story
Beware of hiding the true story
Beware of hiding the true story


Microservice B [Container] Sends order creation messages to [via Kafka topic Y] Microservice D [Container]
Add more words to make the intent explicit
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.
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 plug-in files, config, themes, plugins, etc.

AWS Reference Architectures

© 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Container diagram for Internet Banking System

The container diagram for the Internet Banking System.

- **Database**
  - Container: Relational Database Schema
  - Stores user registration information, hashed authentication credentials, access logs, etc.

- **Web Application**
  - Container: Java and Spring MVC
  - Delivers the static content and the Internet banking single page application.

- **Single-Page Application**
  - Container: JavaScript and Angular
  - Provides all of the Internet banking functionality to customers via their web browser.

- **Mobile App**
  - Container: Xamarin
  - Provides a limited subset of the Internet banking functionality to customers via their mobile device.

- **API Application**
  - Container: Java and Spring MVC
  - Provides Internet banking functionality via a JSON/HTTPS API.

- **E-mail System**
  - Container: Software System
  - The internal Microsoft Exchange e-mail system.

- **Mainframe Banking System**
  - Container: Software System
  - Stores all of the core banking information about customers, accounts, transactions, etc.

- **Personal Banking Customer**
  - Person
  - A customer of the bank, with personal bank accounts.

- **Internet Banking System**
  - Container: Software System

Workflow:
- The Personal Banking Customer uses the Web Application, Single-Page Application, or Mobile App to access their bank accounts.
- The Web Application delivers the static content to the customer's web browser.
- The Single-Page Application and Mobile App make API calls to the API Application using JSON/HTTPS.
- The API Application reads from and writes to the Database using JDBC.
- The E-mail System sends e-mails to the Personal Banking Customer.
- The Mainframe Banking System stores all of the core banking information.

Last modified: Sat Jan 11 2020 14:47:20 GMT+0000 (Greenwich Mean Time)
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.
## General

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

Ensure that your team has a ubiquitous language to describe software architecture
1. Draw **System Context** and **Container** diagrams to describe your software system, on flip chart paper.
Designing software is where the complexity should be, not communicating it!
The diagrams should spark meaningful questions
No

“What does that arrow mean?”

“Why are some boxes red?”

“Is that a Java application?”

“Is that a monolithic application, or a collection of microservices?”

“How do the users get their reports?”
Yes

“What protocol are your two Java applications using to communicate with each other?”

“Why do you have two separate C# applications instead of one?”

“Why are you using MongoDB?”

“Why are you using MySQL when our standard is Oracle?”

“Should we really build new applications with .NET Framework rather than .NET Core?”
Richer diagrams lead to richer design discussions
Richer diagrams lead to better communication, making it easier to scale teams.
Similar levels of abstraction provide a way to easily compare solutions.
System landscape diagrams
**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)
Runtime/behavioural diagrams
Static structure diagrams are very useful, but they don't tell the whole story.
**Dynamic diagram for API Application**

Summarises how the sign in feature works in the single-page application.

Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)
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.

3: Stores data in [DDB]

4: Sends customer update events to

5: Sends customer update events to

6: Stores data in

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 software system/container instances.
Deployment diagram for Internet Banking System - Development

An example development deployment scenario for the Internet Banking System.

Tuesday, 4 August 2020, 09:32 GMT+01:00

Web Browser
[Deployment Node: Chrome, Firefox, Safari, or Edge]

Single-Page Application
[Container: JavaScript and Angular]
Provides all of the Internet banking functionality to customers via their web browser.

Delivers to the customer's web browser

Makes API calls to
[JSON/HTTPS]

Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.

API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.

Apache Tomcat
[Deployment Node: Apache Tomcat 8.x]

Docker Container - Web Server
[Deployment Node: Docker]

Reads from and writes to
[DB2]

Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.

Docker Container - Database Server
[Deployment Node: Docker]

Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.

bigbank-dev001
[Deployment Node: Big Bank plc data center]
Infrastructure Node
Routers, firewalls, load balancers, DNS providers, edge caches, etc
Frequently asked questions
What's the inspiration behind the C4 model?
Are these diagrams for design or documentation purposes?
Why "container"?
Can we change the terminology?
Why did you reinvent UML?
The C4 model is a tool for structured thinking about software architecture, at different levels of (named and explicit) abstractions. There is no “C4 model” notation.

UML + C4 model
A standard notation, with diagrams at explicit levels of abstraction.

UML
1. A standard set of notations (boxes and lines) for describing things.
2. A standard set of diagram types (with rules) for combining those things to tell a story.

C4 model
1. A common set of abstractions, at different levels of detail (software systems, containers, components, code).
2. A common set of diagram types (with rules) for combining these things to tell a story.
What tooling do you recommend?
Tooling

For design sessions, you might find a whiteboard or flip chart paper better for collaboration, and iterating quickly. For long-lived documentation, the following modelling and diagramming tools can help create software architecture diagrams based upon the C4 model.

**Modelling tools (recommended: why?)**

**Structurizr**
Structurizr is a collection of tools to create software architecture diagrams and documentation based upon the C4 model. Structurizr was started in 2014 by Simon Brown (creator of the C4 model), and has grown into a community of tools, much of which is open source.

Structurizr is unique in that it supports diagrams as code (Java, Clojure, .NET, TypeScript, PHP, Python), on a canvas or text (DSL, or YAML) via a number of different authoring methods, with it being possible to render diagrams using a number of different tools (Structurizr Cloud, service-oriented, premade installation, PlantUML, Mermaid, WebSequenceDiagrams, etc).

**Archi**
Archi provides a way for you to create C4 model diagrams with ArchiMate. See C4 Model, Architecture Viewpoint and Archi 4.1 for more details.

**IcePanel**
IcePanel is a structured diagramming tool that supports the C4 model.

**Sparx Enterprise Architect**
Sparx Enterprise Architect is an extension for the C4 model, based upon the MDG Technology built into Sparx Enterprise Architect.

**MooD**
MooD has support for the C4 model via a set of blueprints.

**Astah**
Astah supports the C4 model via a C4 model plugin.

**Diagramming tools**

**PlantUML**
There are a number of extensions for PlantUML to assist in the creation of C4 model diagrams:
- C4-PlantUML by Ricardo Nepel
- C4-PlantUMLBi by Savvas Klenkous
- C4Builder by Victor Lupu
- plantuml4hs by Thibault Morin

You can also create C4-PlantUML diagrams using C4 code via the C4Sharp library.

**diagrams.net**
diagrams.net includes support for the C4 model, and there are also a number of plugins that allow you to create diagrams using pre-built shapes:
- c4-draw.js by Chris Kamiński
- c4-draw.js by Tobias Hochgertler
- EasyC by Maciek Szwierski

**OmniGraffle**
Dennis Launen has created a C4 model stencil for OmniGraffle, that allows you to create diagrams using pre-built shapes.

**Microsoft Visio**
"pikhafer" has created a C4 model template for Microsoft Visio, that allows you to create diagrams using pre-built shapes.

**yEd**
Berhat Kaficici has created some C4 model shapes for yEd.
Diagramming tools
You create and maintain multiple diagrams, remembering to keep them all in sync whenever you change a diagram
title Software System - System Context

caption An example of a System Context diagram.

Person(1, "User", "A user of my software system." )
System(2, "Software System", "My software system.")
Rel(1, 2, "Uses", "")

#enduml
Modelling tools
You create and maintain a single model, and the tool generates multiple diagrams, automatically keeping them all in sync whenever you change the model.
Simon Brown Updated to reflect release.

- **.github/workflows**
  - GitHub Actions fixes.
  - 5 months ago

- **docs**
  - Updated to reflect release.
  - 3 days ago

- **examples**
  - Fixes an issue where `this` didn't work when defining relationships i...
  - 5 days ago

- **gradle**
  - Add gradle JAR.
  - 15 months ago

- **src**
  - Adds support for formatting the branding logo and font as DSL.
  - 4 days ago

- **.gitignore**
  - Add gradle JAR.
  - 15 months ago

- **LICENSE**
  - Initial commit
  - 16 months ago

- **README.md**
  - Added a link to the changelog.
  - 3 months ago

- **build.gradle**
  - Updated to reflect release.
  - 3 days ago

- **gradle.properties**
  - Adds a dummy Gradle properties file for GitHub Actions.
  - 5 months ago

- **gradlew**
  - Initial commit of source code for the DSL parser.
  - 16 months ago

- **gradlew.bat**
  - Initial commit of source code for the DSL parser.
  - 16 months ago

---

https://github.com/structurizr/dsl
workspace {

model {
    user = person "User"
    softwareSystem = softwareSystem "Software System"

    user -> softwareSystem "Uses"
}

views {
    systemContext softwareSystem {
        include *
        autoLayout
    }
}
}
workspace {

    model {
        user = person "User"
        softwareSystem = softwareSystem "Software System" {
            webapp = container "Web Application"
            database = container "Database"
        }

        user -> webapp "Uses"
        webapp -> database "Reads from and writes to"
    }

    views {
        systemContext softwareSystem {
            include *
            autoLayout
        }

        container softwareSystem {
            include *
            autoLayout
        }
    }
}
workspace {

model {
    user = person "User"
    softwareSystem = softwareSystem "Software System" {
        webapp = container "Web Application"
        database = container "Database"
    }
    user -> webapp "Uses"
    webapp -> database "Reads from and writes to"
}

views {
    systemContext softwareSystem {
        include *
        autoLayout
    }
    container softwareSystem {
        include *
        autoLayout
    }
    theme default
}
Structurizr Lite

Overview

Packaged as a Docker container, and designed for developers, this version of Structurizr provides a way to quickly work with a single workspace. It’s free to use, and allows you to view/edit diagrams, view documentation, and view architecture decision records defined in a DSL or JSON workspace.

Structurizr Lite will look for a `workspace.dsl` and `workspace.json` file in a given directory, in that order, and use the file it finds first. If you change this file (e.g. via your text editor or one of the Structurizr client libraries), you can refresh your web browser to immediately see the changes.

https://structurizr.com/help/lite
A command line utility for Structurizr.

- structurizr.com
- markdown
- plantuml
- ascdoc
- software-architecture
- architecture-doc
- architecture-decision-records
- structurizr
c4model
- adrs
- websequencediagrams
- architecture-diagrams
- click

Readme

Apache-2.0 License

Releases 30

- v1.15.0 (Latest)
  3 days ago

+ 29 releases

Contributors 6

https://github.com/structurizr/cli
<table>
<thead>
<tr>
<th>Feature</th>
<th>Structurizr (diagram)</th>
<th>Structurizr (graph)</th>
<th>PlantUML</th>
<th>Mermaid</th>
<th>DOT</th>
<th>WebSequenceDiagrams</th>
<th>Ilograph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td></td>
<td></td>
<td><img src="image1" alt="Example Diagram" /></td>
<td><img src="image2" alt="Example Diagram" /></td>
<td><img src="image3" alt="Example Diagram" /></td>
<td><img src="image4" alt="Example Diagram" /></td>
<td><img src="image5" alt="Example Diagram" /></td>
</tr>
<tr>
<td>Shapes</td>
<td></td>
<td></td>
<td><img src="image6" alt="Shapes Diagram" /></td>
<td><img src="image7" alt="Shapes Diagram" /></td>
<td><img src="image8" alt="Shapes Diagram" /></td>
<td><img src="image9" alt="Shapes Diagram" /></td>
<td><img src="image10" alt="Shapes Diagram" /></td>
</tr>
<tr>
<td>Automatic diagram key/legend</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(colours only with C4-PlantUML)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Click to zoom</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive/animations</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rendering</td>
<td>Web browser</td>
<td>Web browser</td>
<td>Server</td>
<td>Server</td>
<td>Server</td>
<td>Server</td>
<td>Web browser or desktop app</td>
</tr>
<tr>
<td>PNG export</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SVG export</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PDF export</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
How do you model microservices and serverless?
How do you diagram large and complex software systems?
container softwareSystem {
  include user -> service1 -> autolayout
}

Software System - Containers

Web Application [Container]

Service 1 API [Container]

Service 1 Database [Container]

Service 2 API [Container]

Service 1 [Group]

Service 2 [Group]
Will the diagrams become outdated quickly?
Why doesn't the C4 model cover business processes, workflows, state machines, domain models, data models, etc?
Does the C4 model imply a design process or team structure?
Thank you!