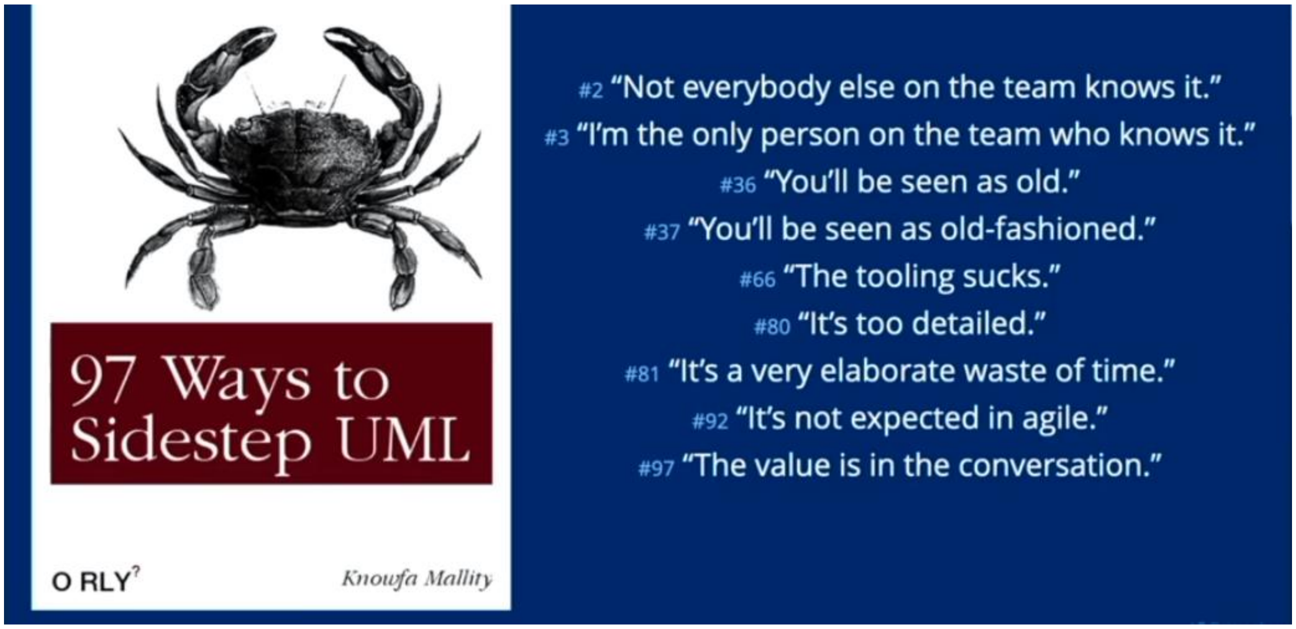
Diagrams as code 2.0

Diagrams as code is becoming a popular way to diagram software architecture, particularly for long-lived high-level documentation. You write the diagram source in a text-based domain specific language (e.g. PlantUML or Mermaid) or a programming language, and render diagrams using web-based or command line tooling. The benefits are well understood – writing the diagram source as text allows for easy integration into software development practices and toolchains, plus the automatic layout facilities allow authors to focus on content. The problem with this approach is that it's easy for diagrams to get out of sync. Enter "diagrams as code 2.0" — a way to define a model of our software architecture and the views that we'd like to see, ultimately resulting in a consistent set of diagrams that are generated for us.

+ some free and open source tooling for creating software architecture diagrams

Teams need a **ubiquitous language** to communicate effectively

Fewer people are using UML

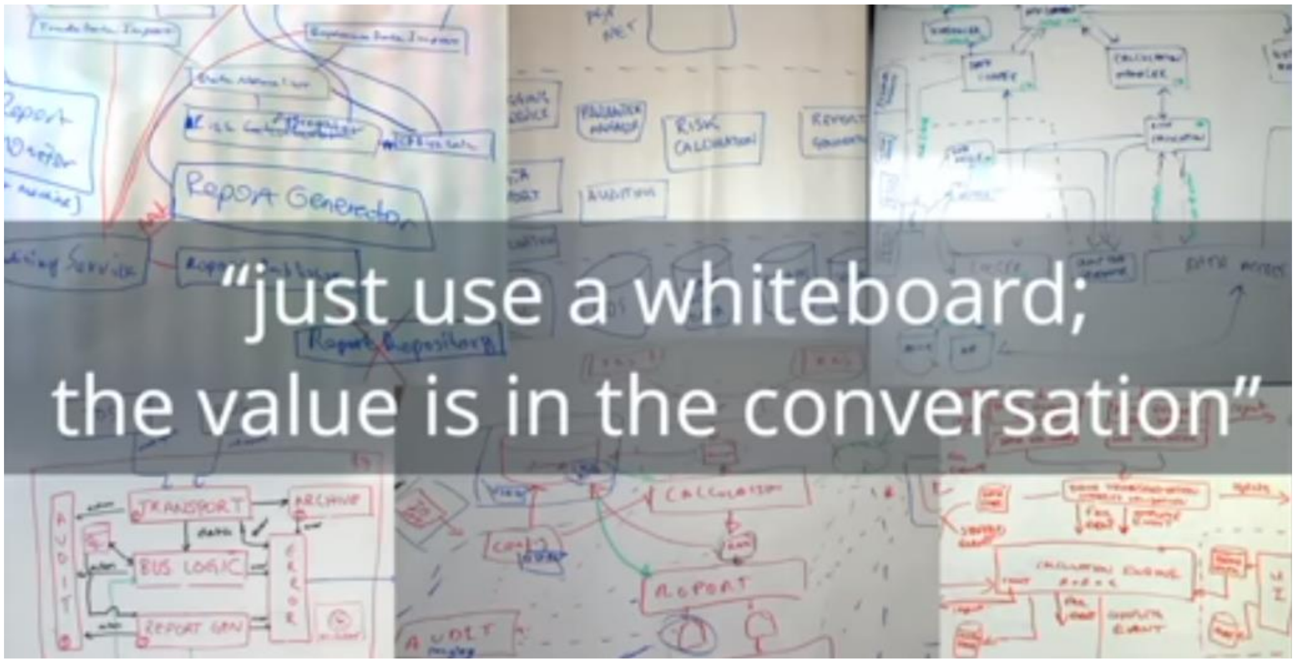


97 Ways to Sidestep UML

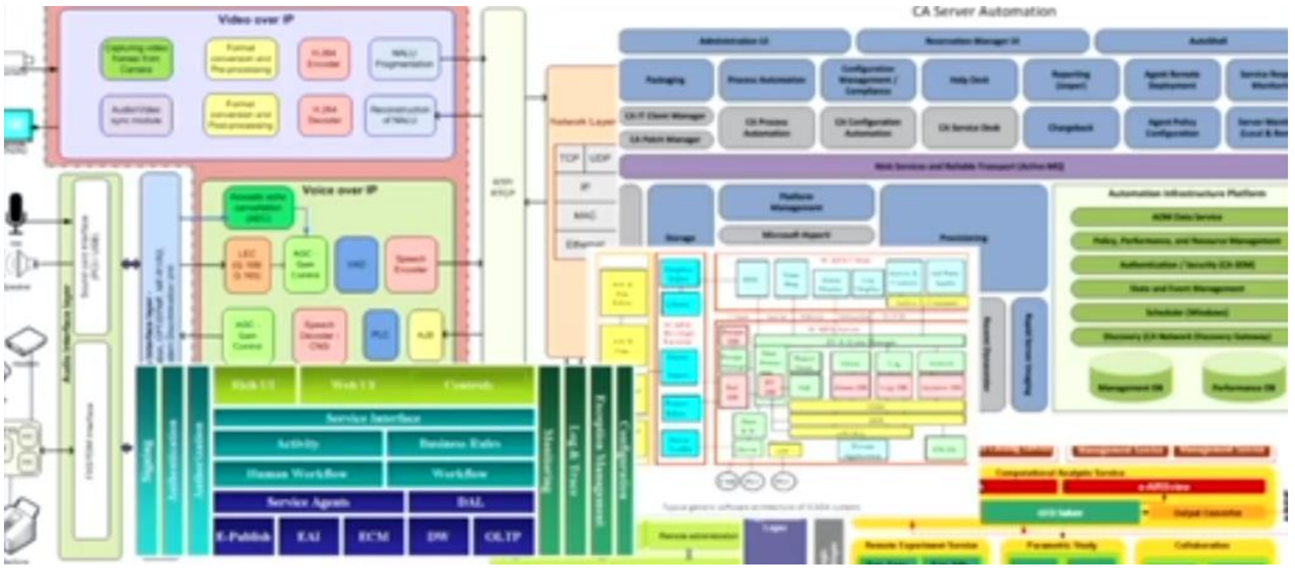
O RLY?

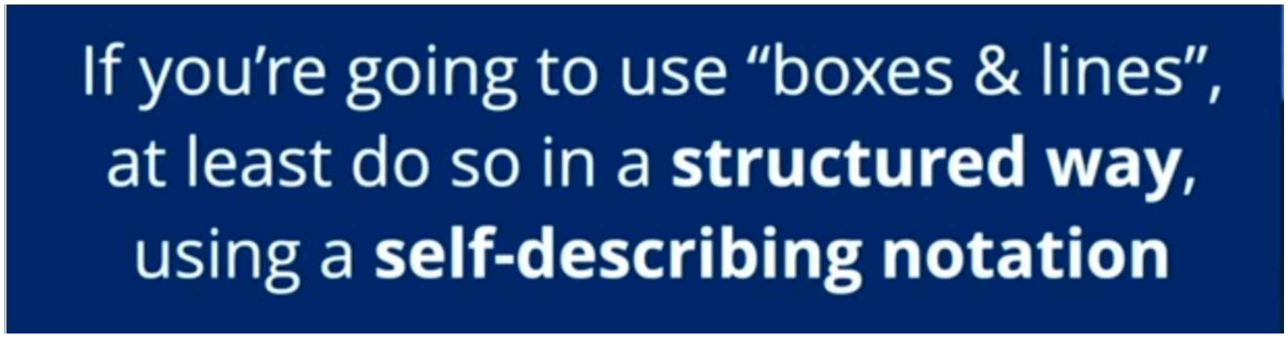
Knowfa Mallity

#2 "Not everybody else on the team knows it."
#3 "I'm the only person on the team who knows it."
#36 "You'll be seen as old."
#37 "You'll be seen as old-fashioned."
#66 "The tooling sucks."
#80 "It's too detailed."
#81 "It's a very elaborate waste of time."
#92 "It's not expected in agile."
#97 "The value is in the conversation."



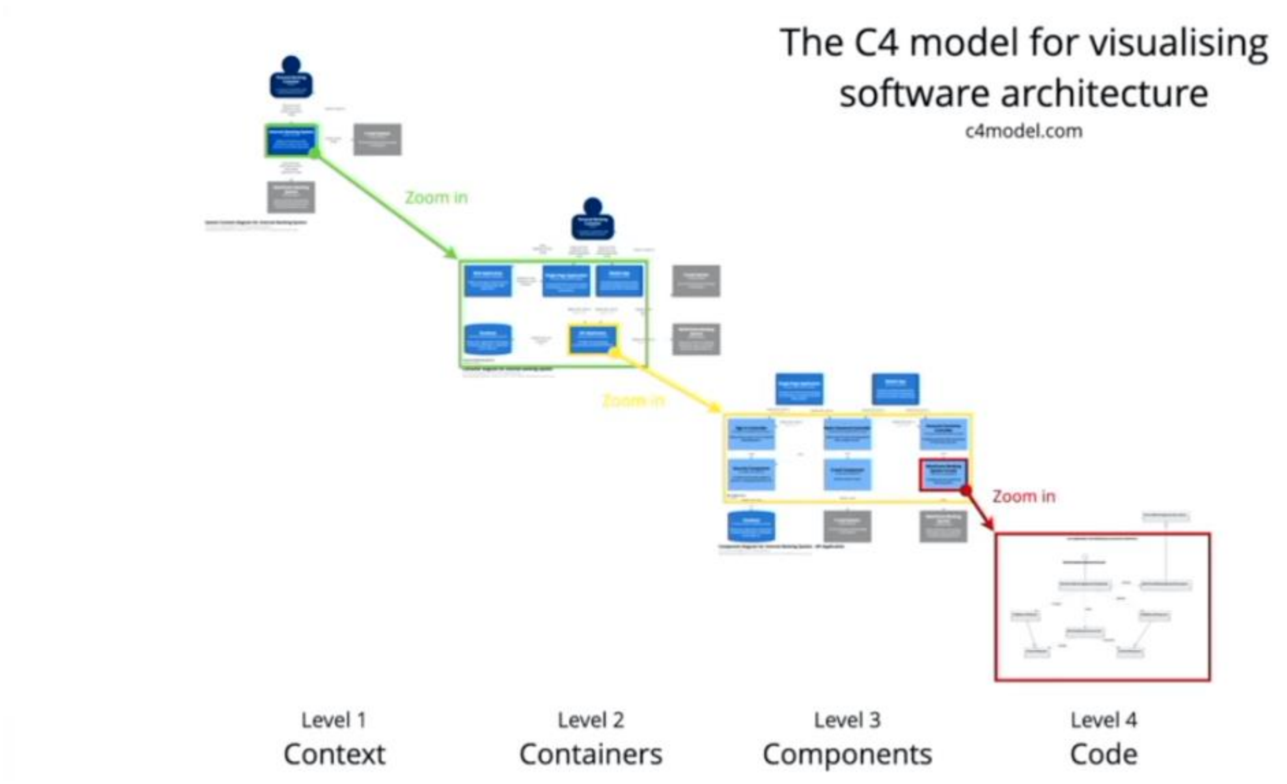
"just use a whiteboard;
the value is in the conversation"




If you're going to use "boxes & lines", at least do so in a **structured way**, using a **self-describing notation**

C4

c4model.com


The C4 model for visualising software architecture

c4model.com

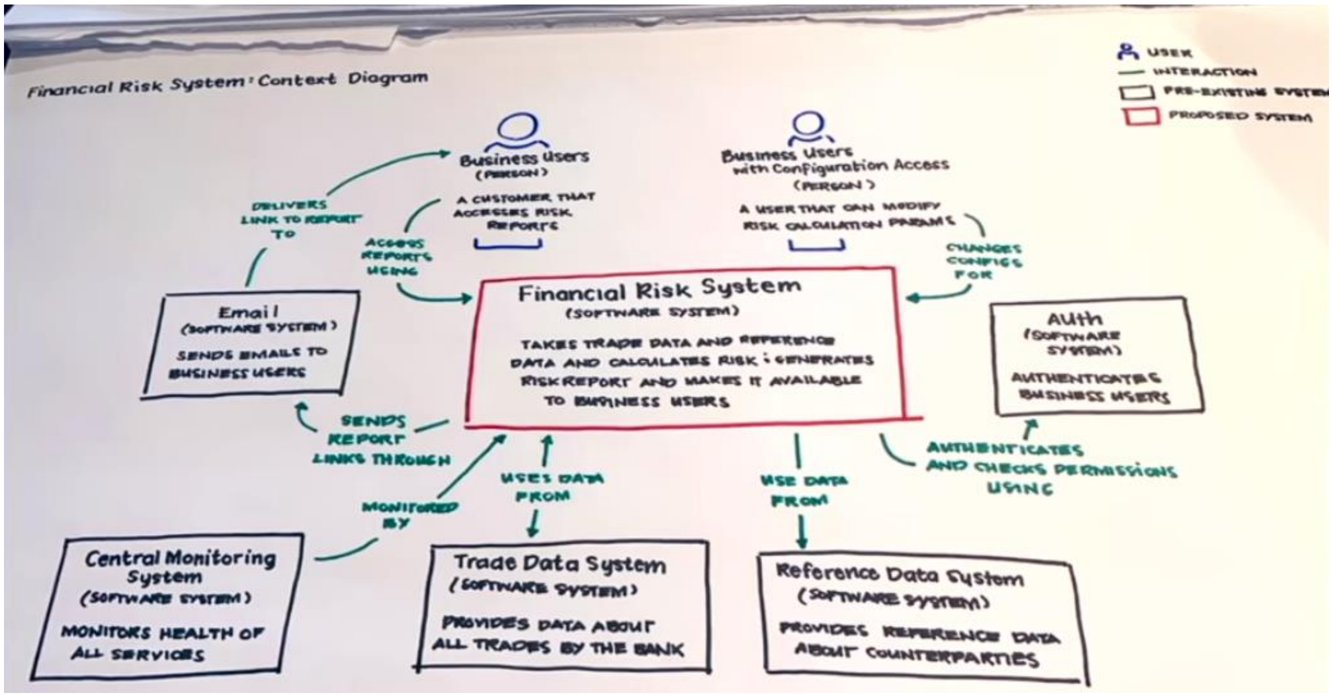
C4 model is a set of hierarchical sequence diagrams that allow you to tell different stories/details to different audiences.

Diagrams are maps

that help software developers navigate a large and/or complex codebase

System Context diagram

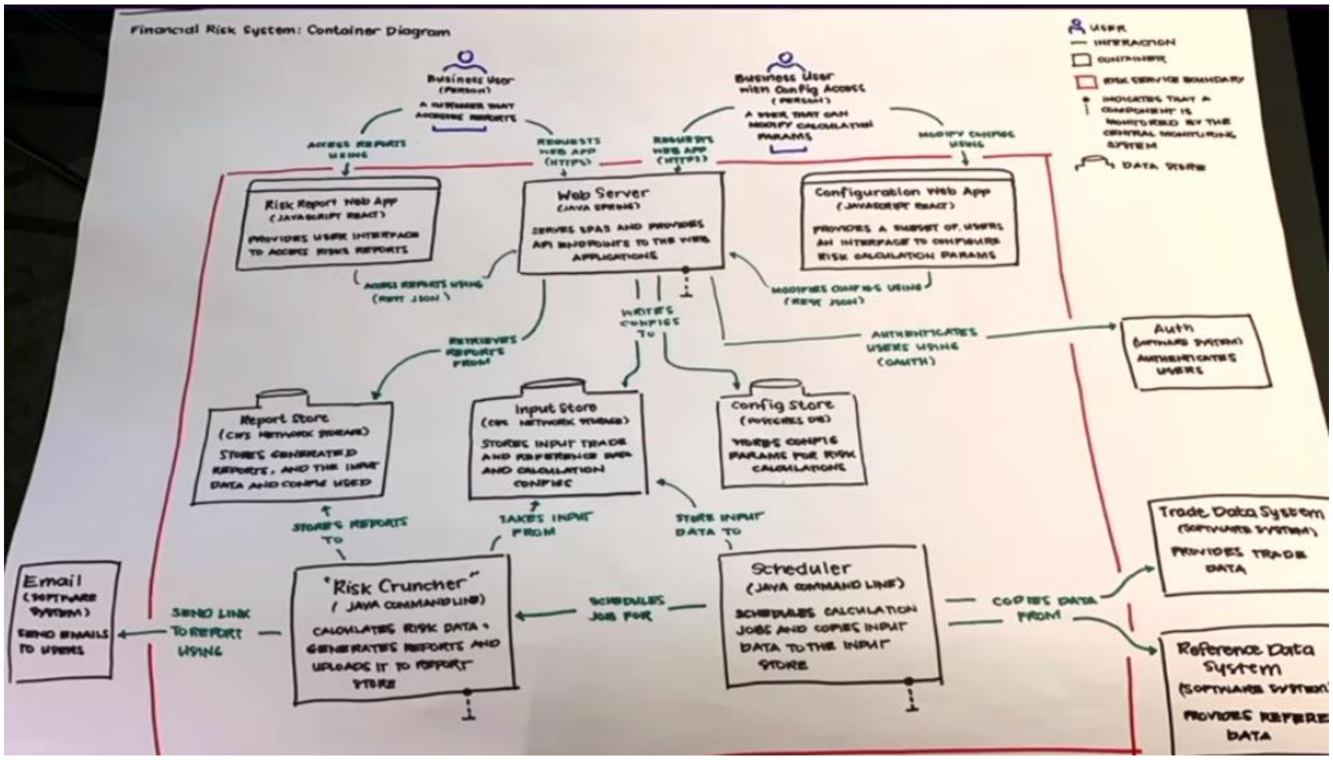
What is the scope of the software system we're building?
Who is using it? What are they doing?
What system integrations does it need to support?



Financial Risk System: Context Diagram
USER
INTERACTION
PRE-EXISTING SYSTEM
PROPOSED SYSTEM
Business Users (PERSON) A CUSTOMER THAT ACCESSES RISK REPORTS
Business Users with Configuration Access (PERSON) A USER THAT CAN MODIFY RISK CALCULATION PARAMS
DELIVERS LINK TO REPORT TO
ACCESS REPORTS USING
CHANGES CONFIGS FOR
Email (SOFTWARE SYSTEM) SENDS EMAILS TO BUSINESS USERS
Financial Risk System (SOFTWARE SYSTEM) TAKES TRADE DATA AND REFERENCE DATA AND CALCULATES RISK; GENERATES RISK REPORT AND MAKES IT AVAILABLE TO BUSINESS USERS
Auth (SOFTWARE SYSTEM) AUTHENTICATES BUSINESS USERS
SENDS REPORT LINKS THROUGH
MONITORED BY
USES DATA FROM
USE DATA FROM
AUTHENTICATES AND CHECKS PERMISSIONS USING
Central Monitoring System (SOFTWARE SYSTEM) MONITORS HEALTH OF ALL SERVICES
Trade Data System (SOFTWARE SYSTEM) PROVIDES DATA ABOUT ALL TRADES BY THE BANK
Reference Data System (SOFTWARE SYSTEM) PROVIDES REFERENCE DATA ABOUT COUNTERPARTIES

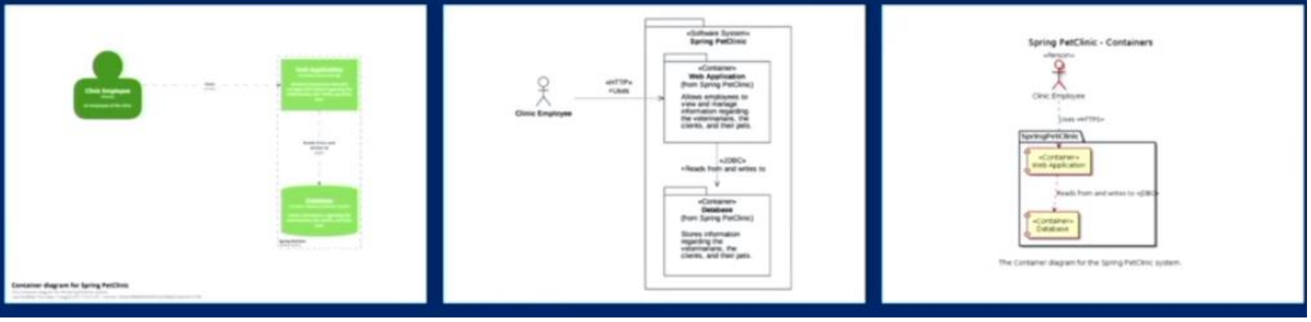
Container diagram

What are the major technology building blocks?
What are their responsibilities?
How do they communicate?



Financial Risk System: Container Diagram
Business User (Person) A business that accesses reports
Business User with Config Access (Person) A user that can modify calculation params
Access reports using
Requests web app (HTTPS)
Requests web app (HTTPS)
Modify configs using
Risk Report Web App (JavaScript React) Provides user interface to access risk reports
Web Server (Java Spring) Serves SPAs and provides API endpoints to the web applications
Configuration Web App (JavaScript React) Provides a subset of users an interface to configure risk calculation params
Access reports using (REST JSON)
Modifies configs using (REST JSON)
Writes configs to
Retrieves reports from
Authenticates users using (OAuth)
Auth (Software System) Authenticates users
Report Store (CIFS Network Storage) Stores generated reports, and the input data and config used
Input Store (CIFS Network Storage) Stores input trade and reference data and calculation config
Config Store (Postgres DB) Stores config params for risk calculations
Stores reports to
Takes input from
Store input data to
Email (Software System) Send emails to users
Send link to report using
"Risk Cruncher" (Java command line) Calculates risk data + generates reports and uploads it to report store
Schedules job for
Scheduler (Java command line) Schedules calculation jobs and copies input data to the input store
Copies data from
Trade Data System (Software System) Provides trade data
Reference Data System (Software System) Provides reference data
User
Interaction
Container
Risk service boundary
Indicates that a component is monitored by the central monitoring system
Data store

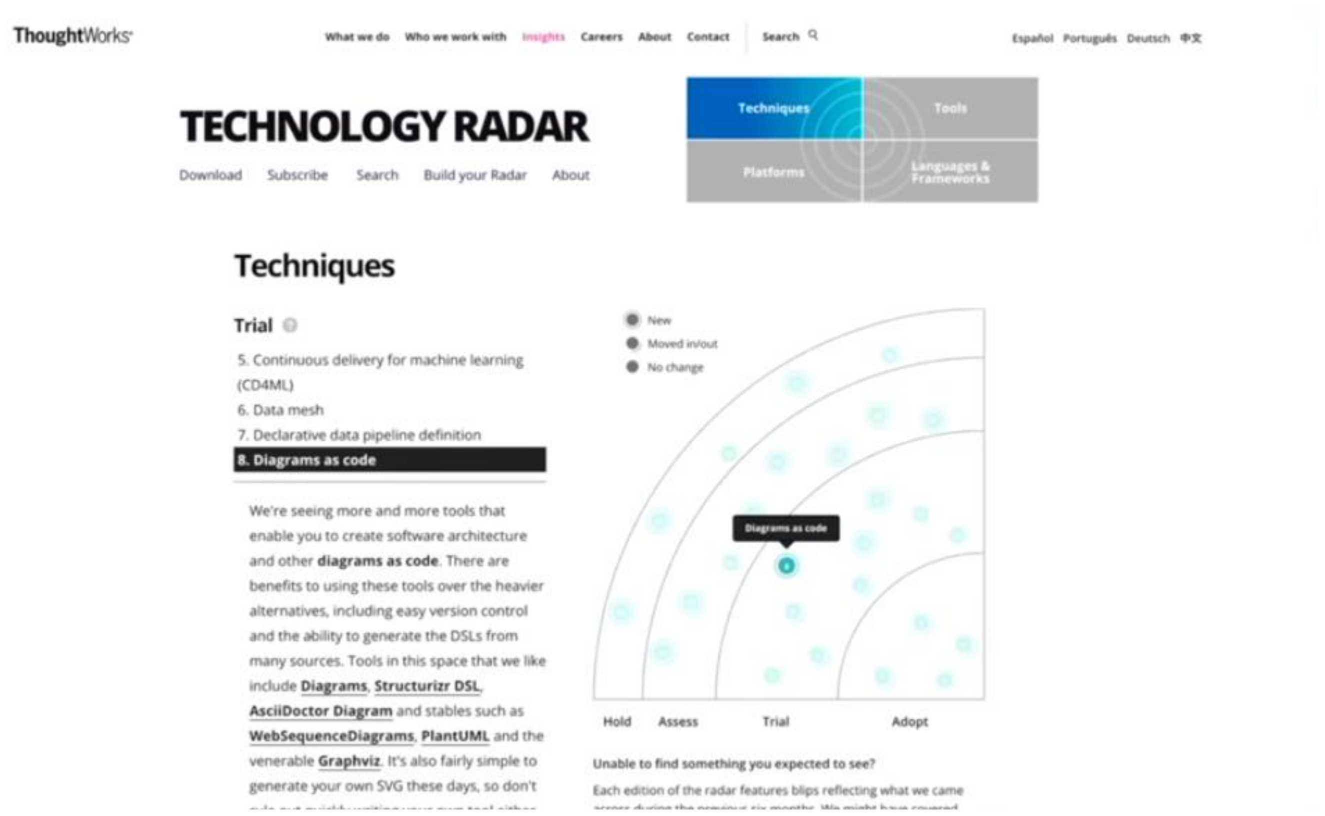
The C4 model is **notation independent**



Container diagram for Spring PetClinic
Clinic Employee
Web Application
Database
«Software System» Spring PetClinic
«Container» Web Application
Allows employees to view and manage information regarding the veterinarians, the clients, and their pets.
«HTTP» Uses
«JDBC» Reads from and writes to
«Container» Database
Stores information regarding the veterinarians, the clients, and their pets.
Spring PetClinic - Containers
«Person» Clinic Employee
Uses «HTTPS»
SpringPetClinic
«Container» Web Application
Reads from and writes to «JDBC»
«Container» Database
The Container diagram for the Spring PetClinic system.

A **common set of abstractions** is more important than a common notation

Tooling?



Techniques

Trial

5. Continuous delivery for machine learning (CD4ML)
6. Data mesh
7. Declarative data pipeline definition
8. Diagrams as code

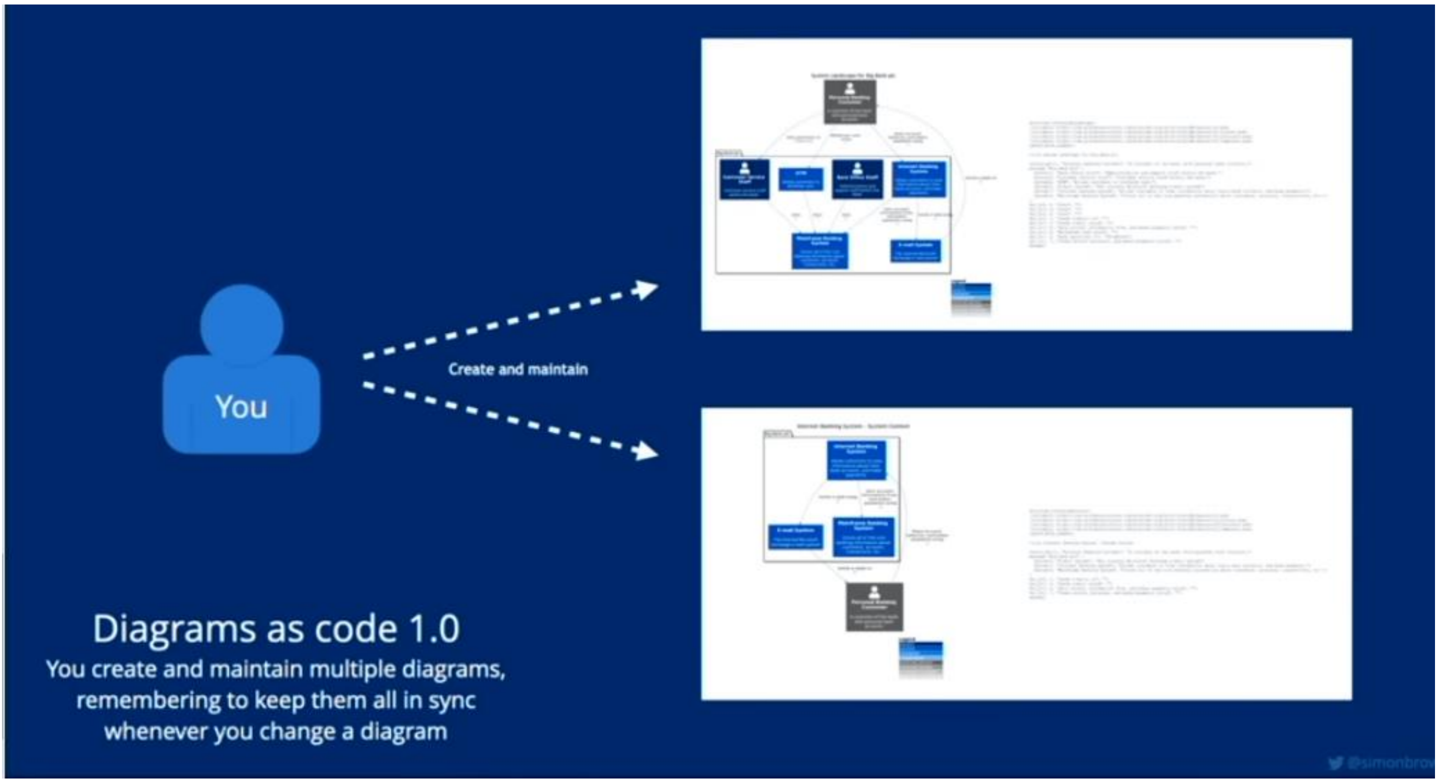
We're seeing more and more tools that enable you to create software architecture and other **diagrams as code**. There are benefits to using these tools over the heavier alternatives, including easy version control and the ability to generate the DSLs from many sources. Tools in this space that we like include **Diagrams**, **Structurizr DSL**, **AsciiDoctor Diagram** and stables such as **WebSequenceDiagrams**, **PlantUML** and the venerable **Graphviz**. It's also fairly simple to generate your own SVG these days, so don't

Unable to find something you expected to see?

Each edition of the radar features blips reflecting what we came

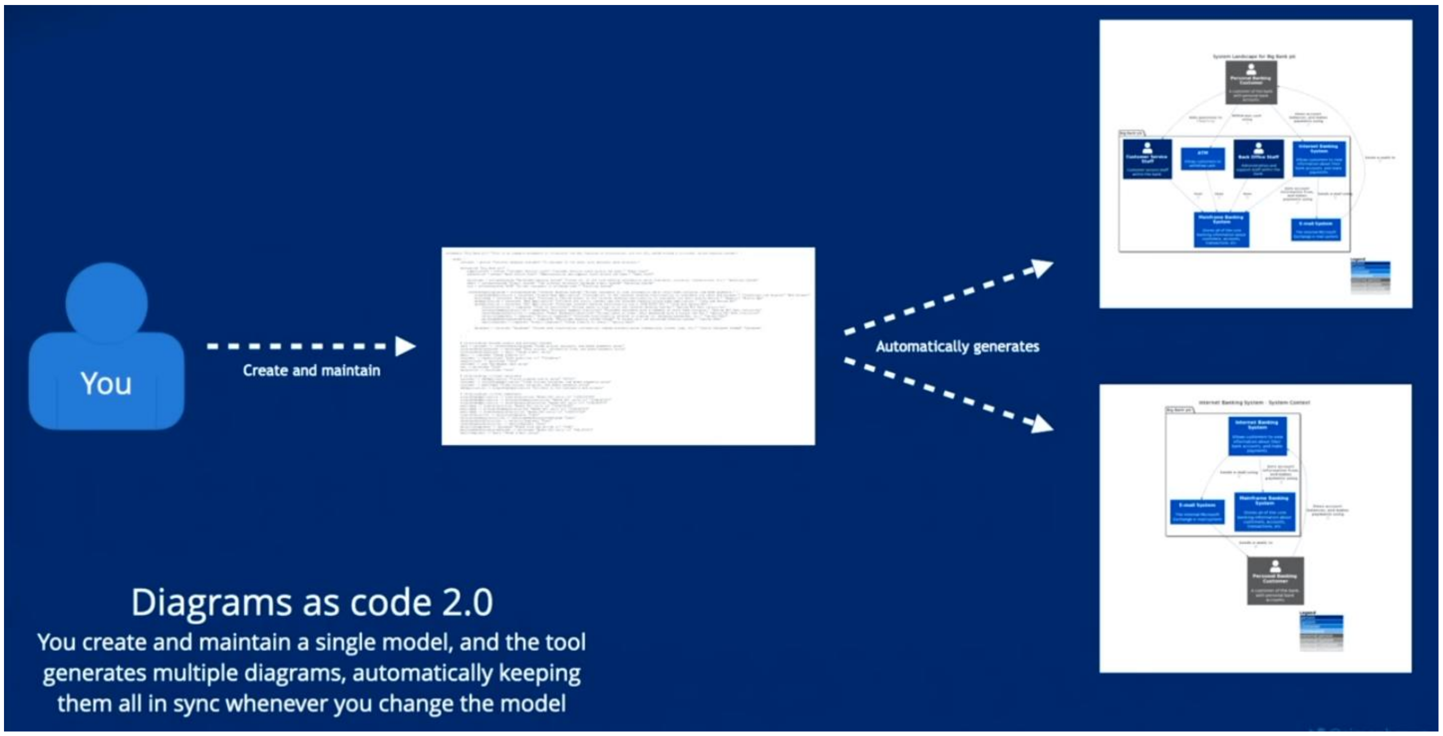
"Diagrams as code" is easy to author, diff, version control, collaborate on, integrate into CI/CD, etc

Diagramming
vs
modelling



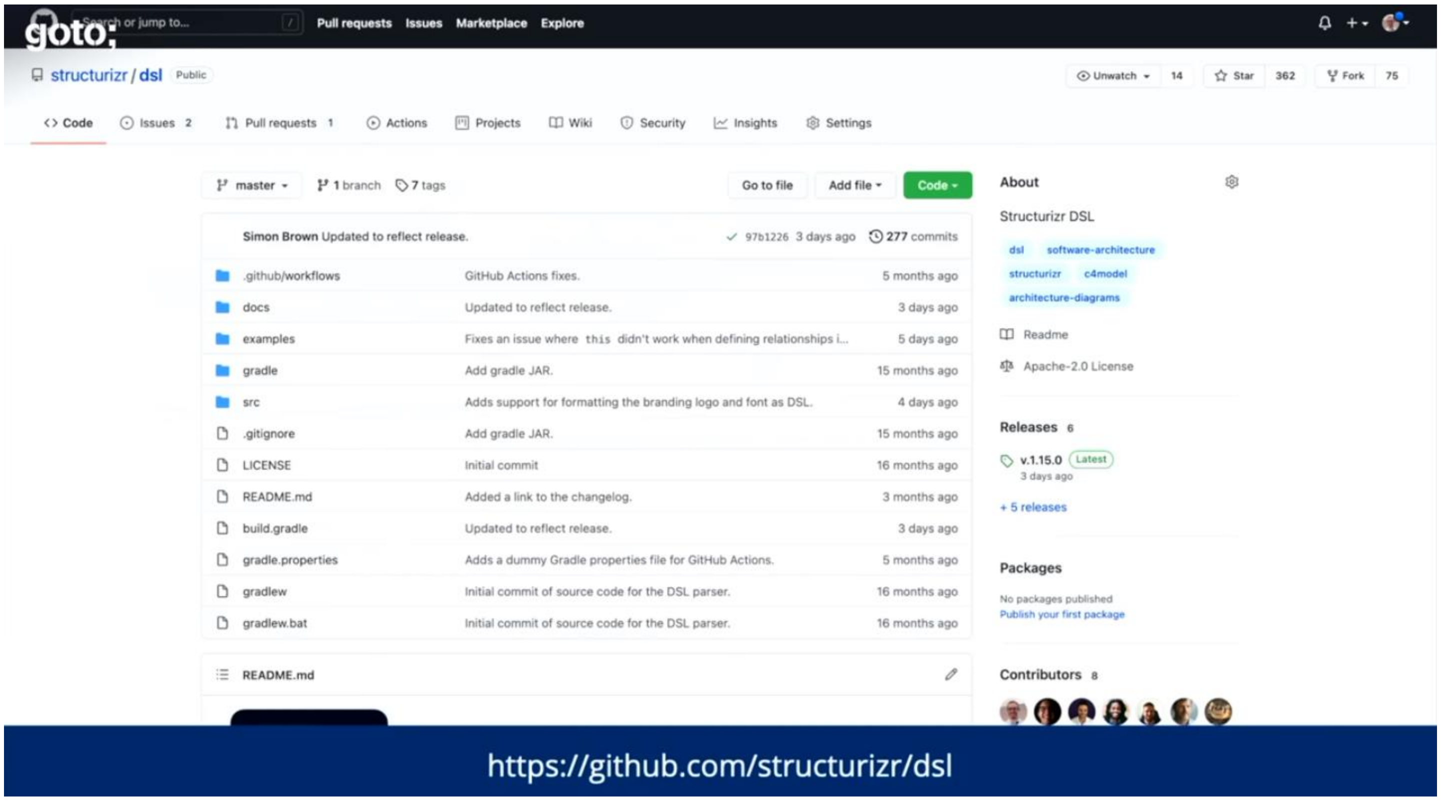
Diagrams as code 1.0

You create and maintain multiple diagrams, remembering to keep them all in sync whenever you change a diagram



Diagrams as code 2.0

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. ✓ 97b1226 3 days ago 277 commits

.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

README.md

About

Structurizr DSL

dsl software-architecture structurizr c4model architecture-diagrams

Readme

Apache-2.0 License

Releases 6

v.1.15.0 Latest
3 days ago

+ 5 releases

Packages

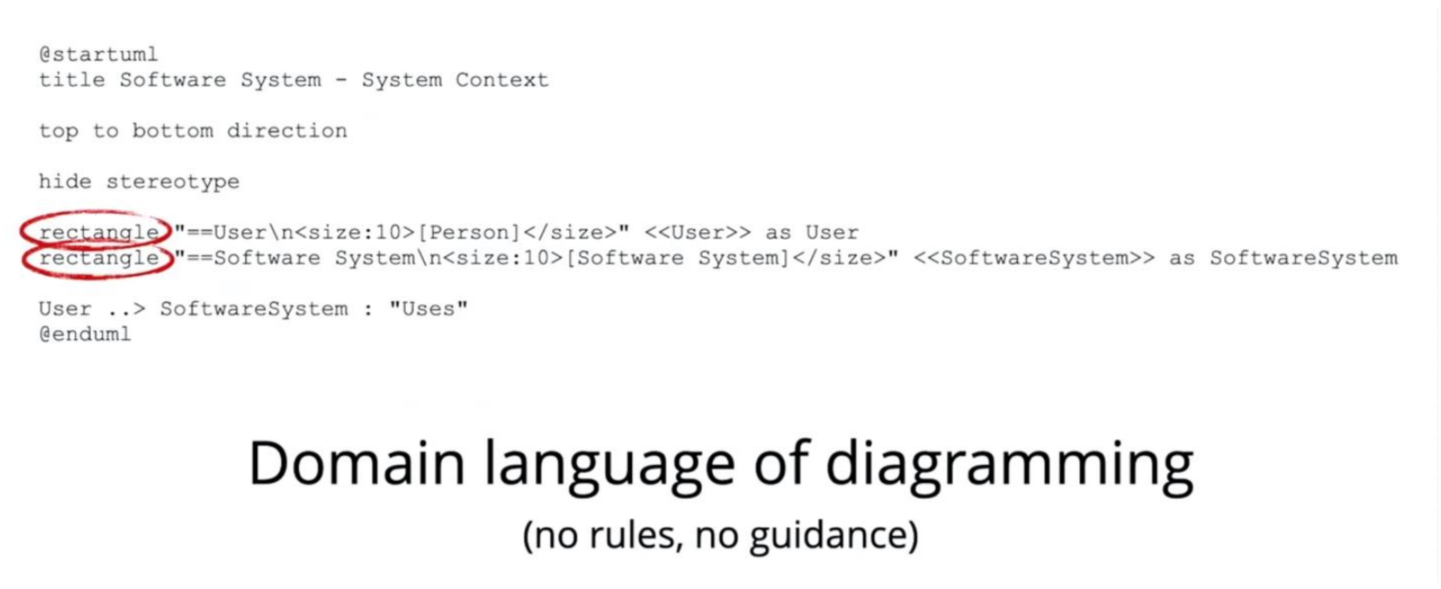
No packages published
Publish your first package

Contributors 8

https://github.com/structurizr/dsl

Domain concepts

(not "boxes and lines")



```
@startuml
title Software System - System Context

top to bottom direction

hide stereotype

rectangle "==User\n<size:10>[Person]</size>" <<User>> as User
rectangle "==Software System\n<size:10>[Software System]</size>" <<SoftwareSystem>> as SoftwareSystem

User ..> SoftwareSystem : "Uses"
@enduml
```

Domain language of diagramming

(no rules, no guidance)

```
workspace {

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

        user -> softwareSystem "Uses"
    }

    views {
        systemContext softwareSystem {
            include *
            autoLayout
        }
    }

}
```

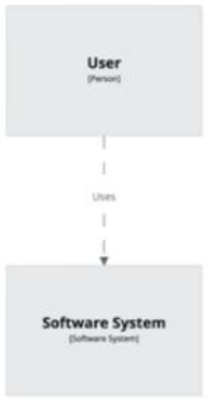
Domain language of software architecture

(metamodel and rules)

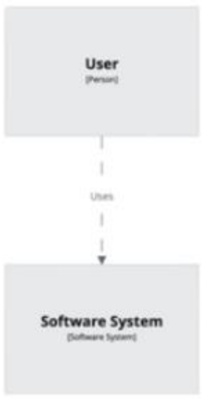
Model-based

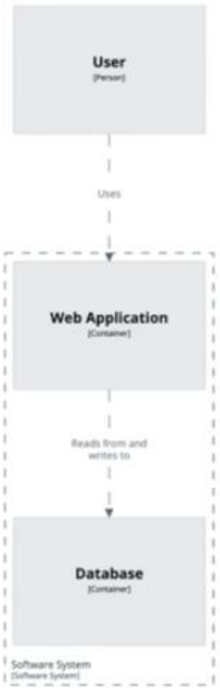
(DRY)

```
workspace {

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



        user -> softwareSystem "Uses"

    }

    views {
        systemContext softwareSystem {
            include *
            autoLayout
        }



    }

}
```



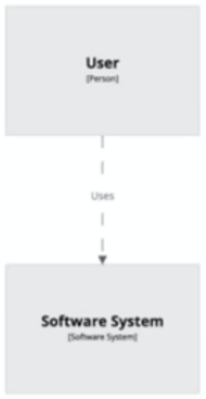
```
workspace {

    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
        }
    }

}
```



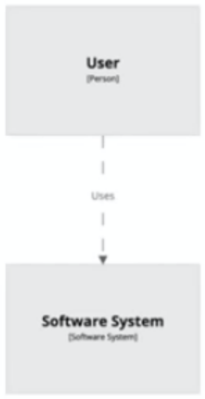


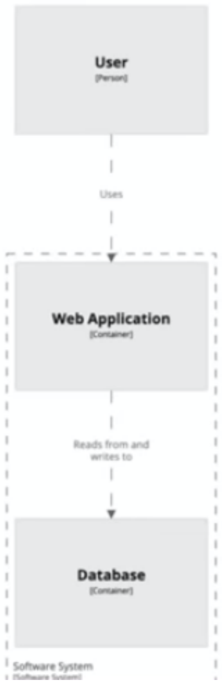
Unspecified relationships can be implied from the model

```
user -> softwareSystem "Uses"
```



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





Implied relationships can be disabled using:

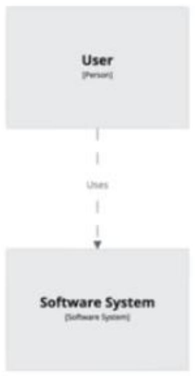
```
!impliedRelationships false
```

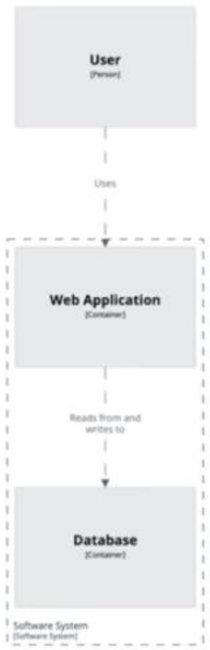
Separation of content and presentation

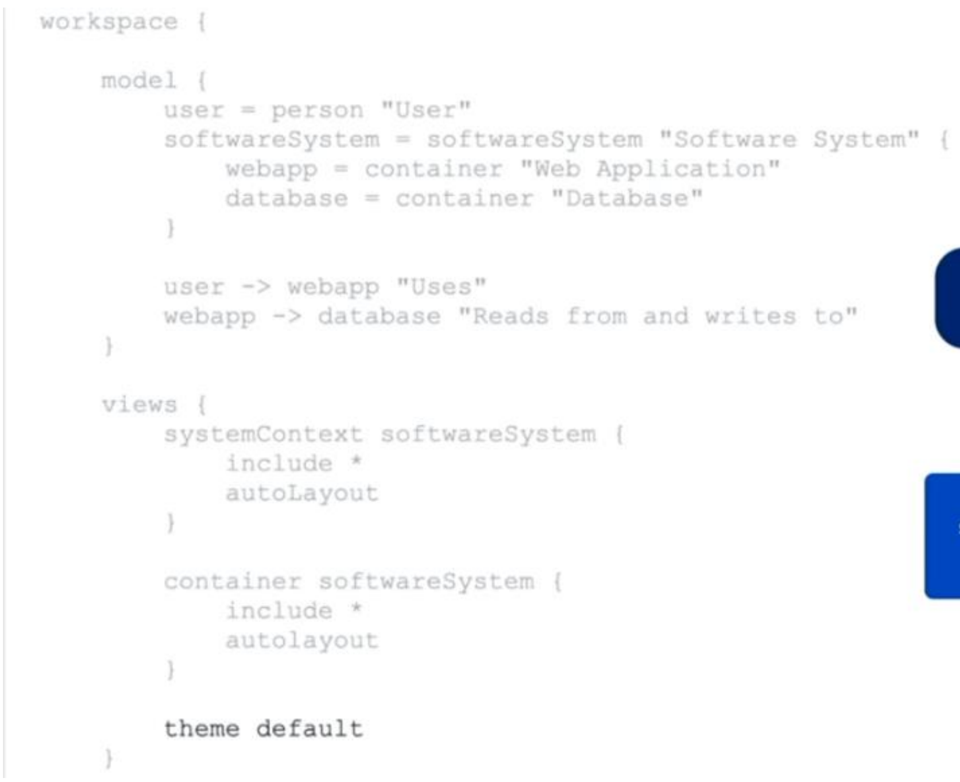
HTML & CSS

the Zen of CSS design
visual enlightenment for the web
by dave shea and molly e. holzschlag

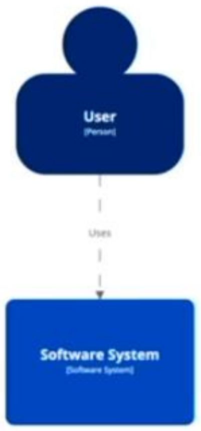
```
workspace {

    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
        }
    }

}
```

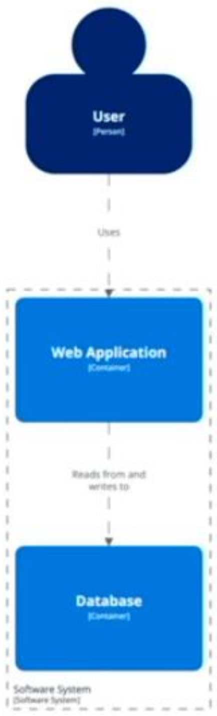


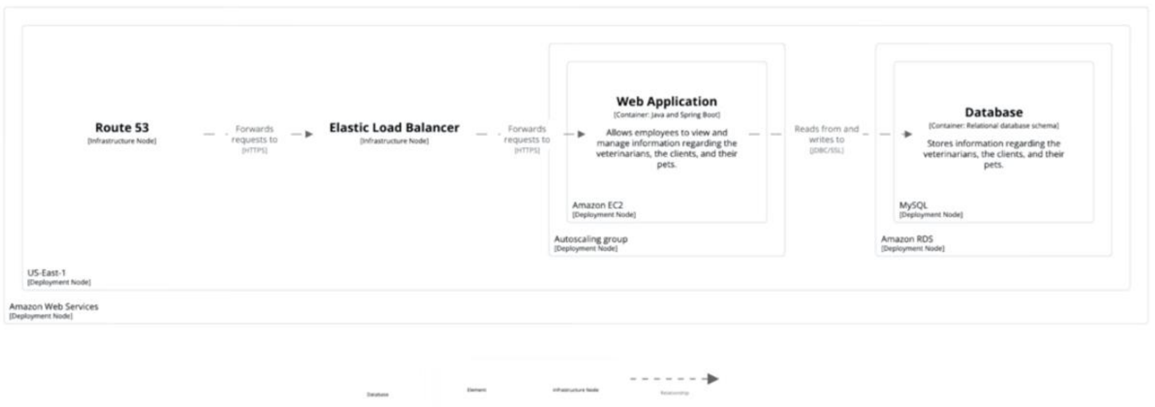


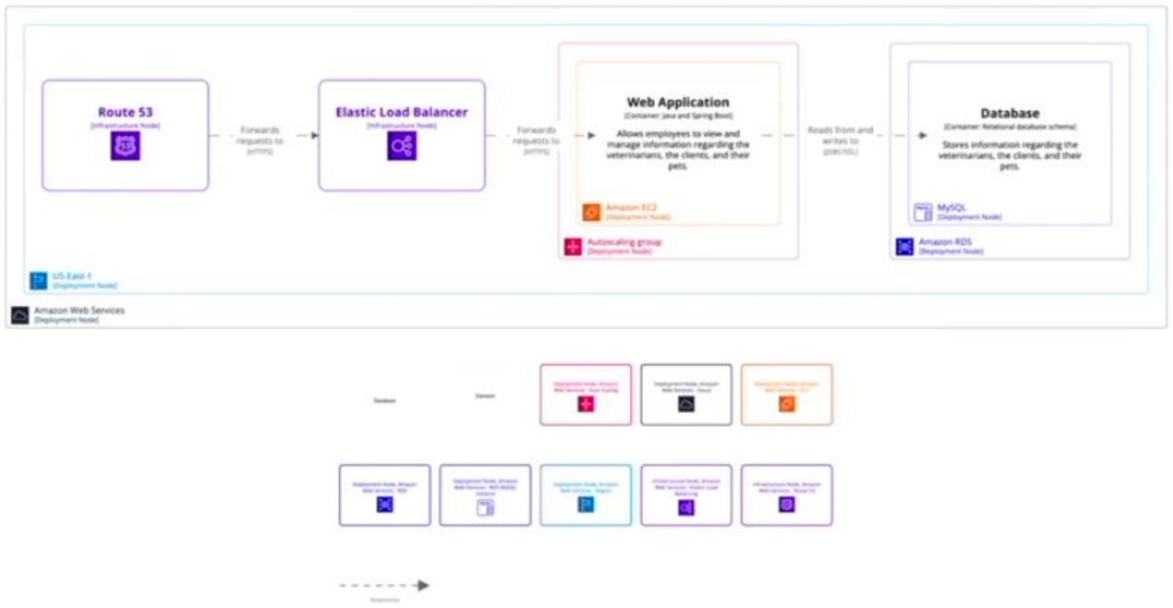


```
workspace {

    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
    }
```





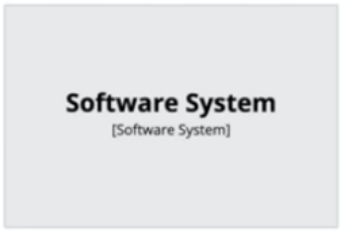




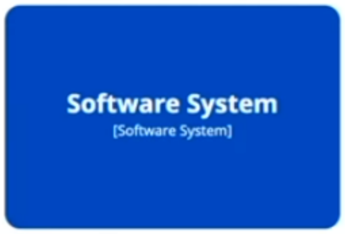
These use the AWS icon theme

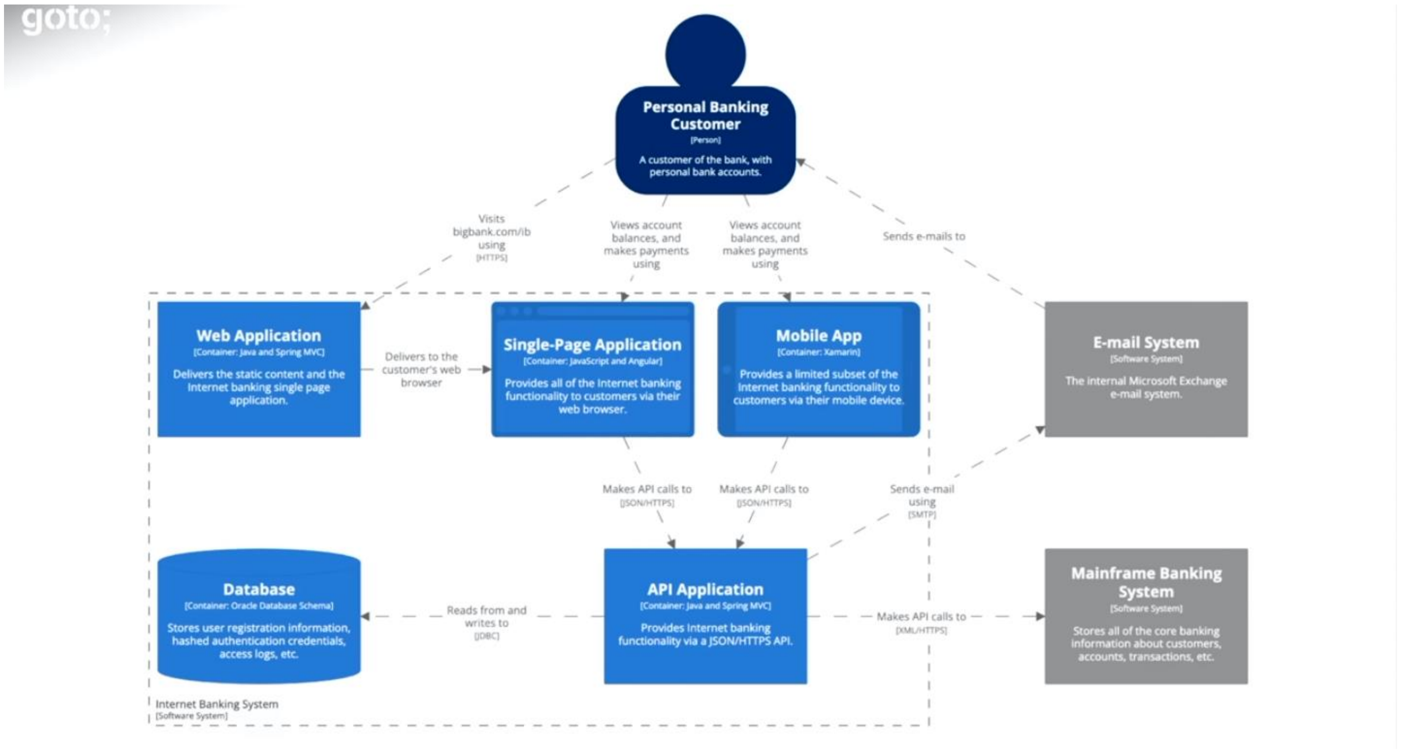
Styling of elements and relationships is achieved via tags

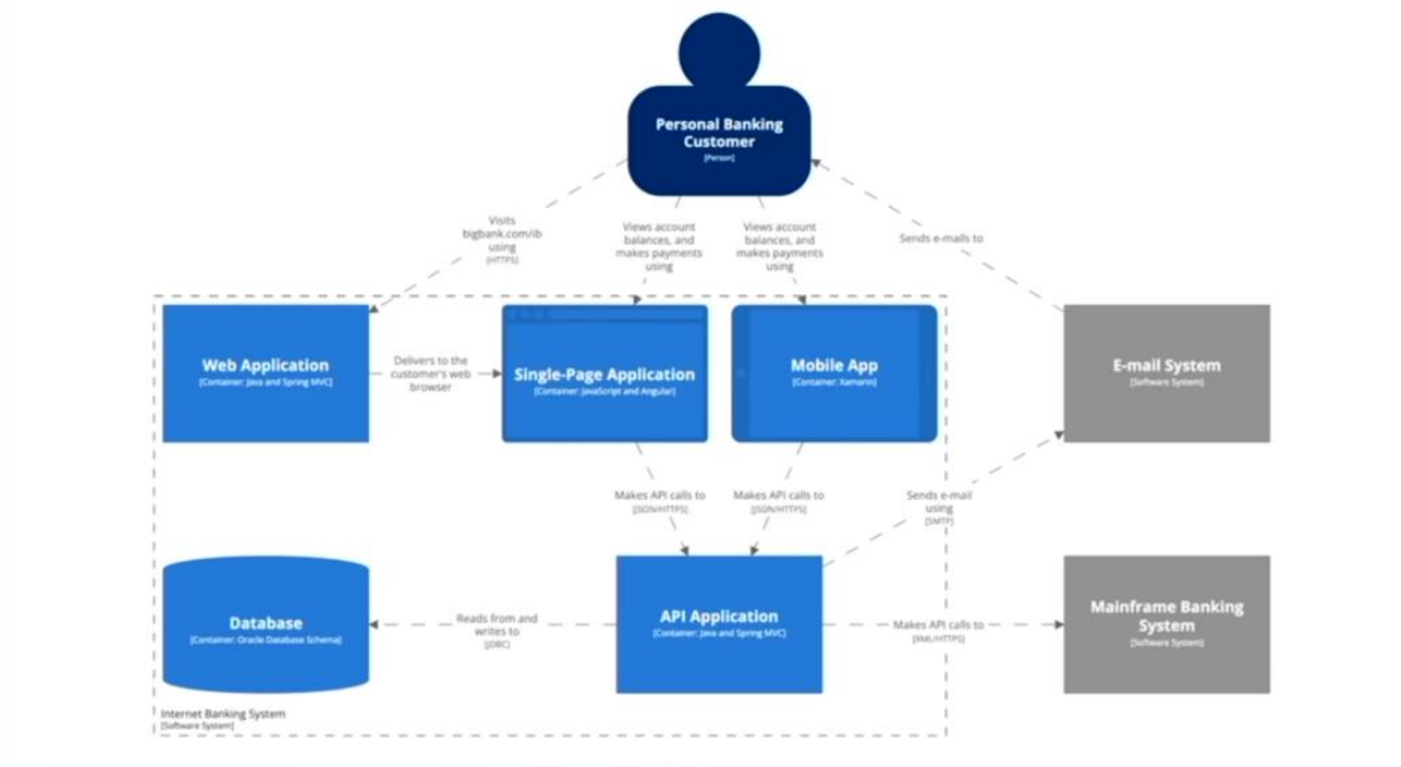
```
workspace {

    model {
        softwareSystem "Software System"
    }

    views {
        systemLandscape {
            include *
            autolayout
        }








    }

}
```



```
workspace {

    model {
        softwareSystem "Software System"
    }

    views {
        systemLandscape {
            include *
            autolayout
        }

        styles {
            element "Software System" {
                background #1168bd
                color #ffffff
                shape RoundedBox
            }
        }
    }

}
```




goto;
Personal Banking Customer
[Person]
A customer of the bank, with personal bank accounts.
Visits bigbank.com/ib using
[HTTPS]
Views account balances, and makes payments using
Views account balances, and makes payments using
Sends e-mails to
Web Application
[Container: Java and Spring MVC]
Delivers the static content and the Internet banking single page application.
Delivers to the customer's web browser
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.
E-mail System
[Software System]
The internal Microsoft Exchange e-mail system.
Makes API calls to
[JSON/HTTPS]
Makes API calls to
[JSON/HTTPS]
Sends e-mail using
[SMTP]
Database
[Container: Oracle Database Schema]
Stores user registration information, hashed authentication credentials, access logs, etc.
Reads from and writes to
[JDBC]
API Application
[Container: Java and Spring MVC]
Provides Internet banking functionality via a JSON/HTTPS API.
Makes API calls to
[XML/HTTPS]
Mainframe Banking System
[Software System]
Stores all of the core banking information about customers, accounts, transactions, etc.
Internet Banking System
[Software System]



You can also turn off the text in a diagram

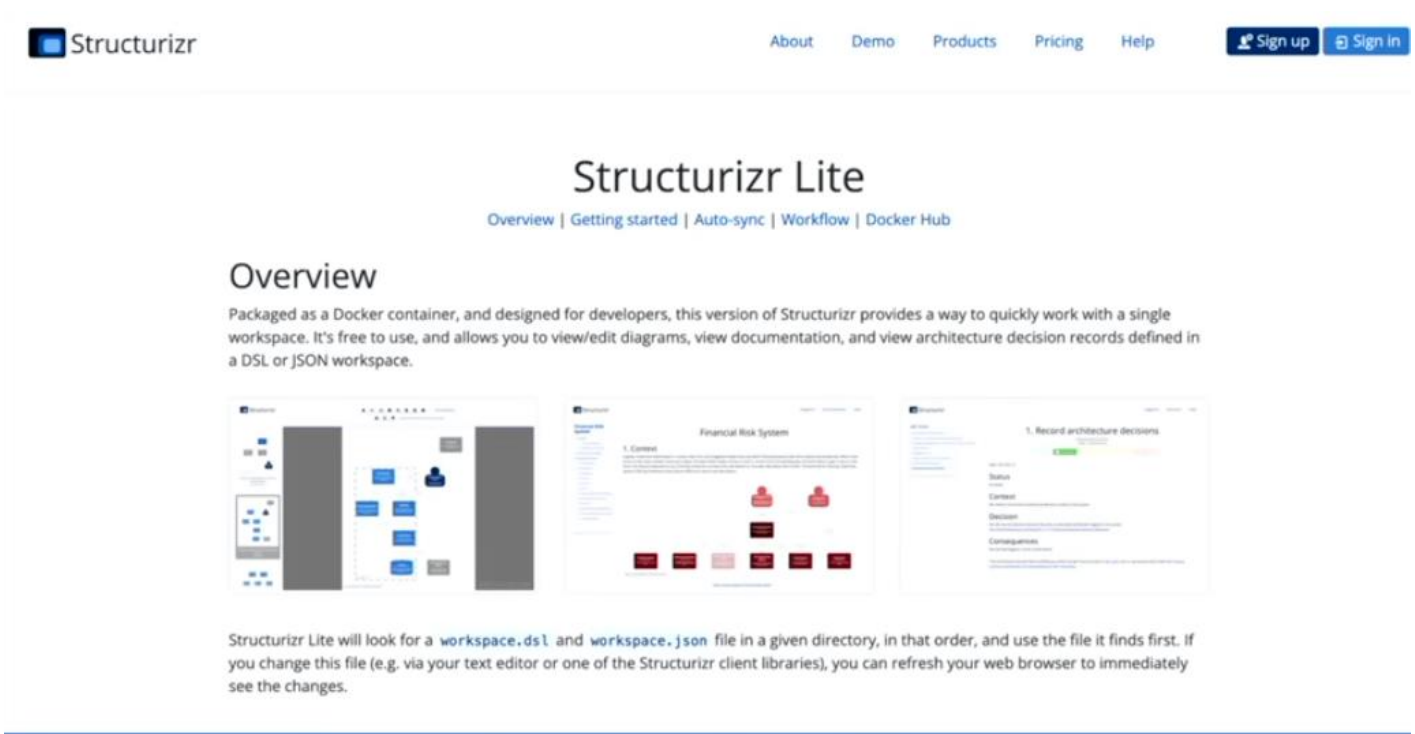
Rendering tool independent

The structurizr tool doesn't draw the diagram for you, it lets you define the model and structure using the DSL

“Diagrams as code 1.0”

PlantUML, Mermaid, etc are **input formats**

Automatic layout
vs
manual layout?



Structurizr Lite

Overview | Getting started | Auto-sync | Workflow | Docker Hub

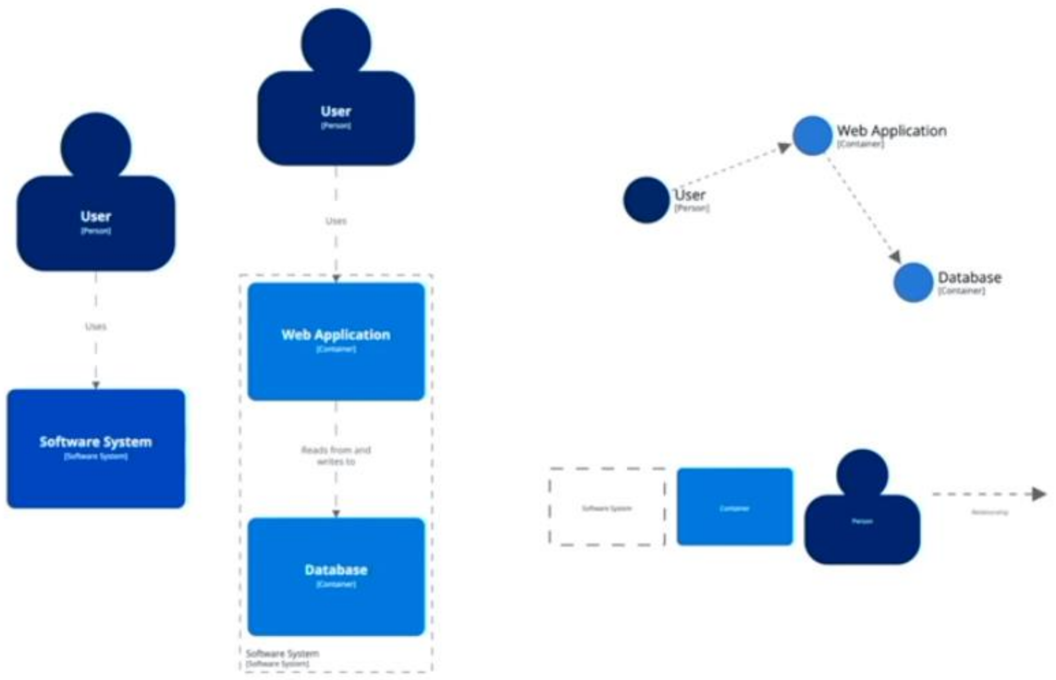
Overview

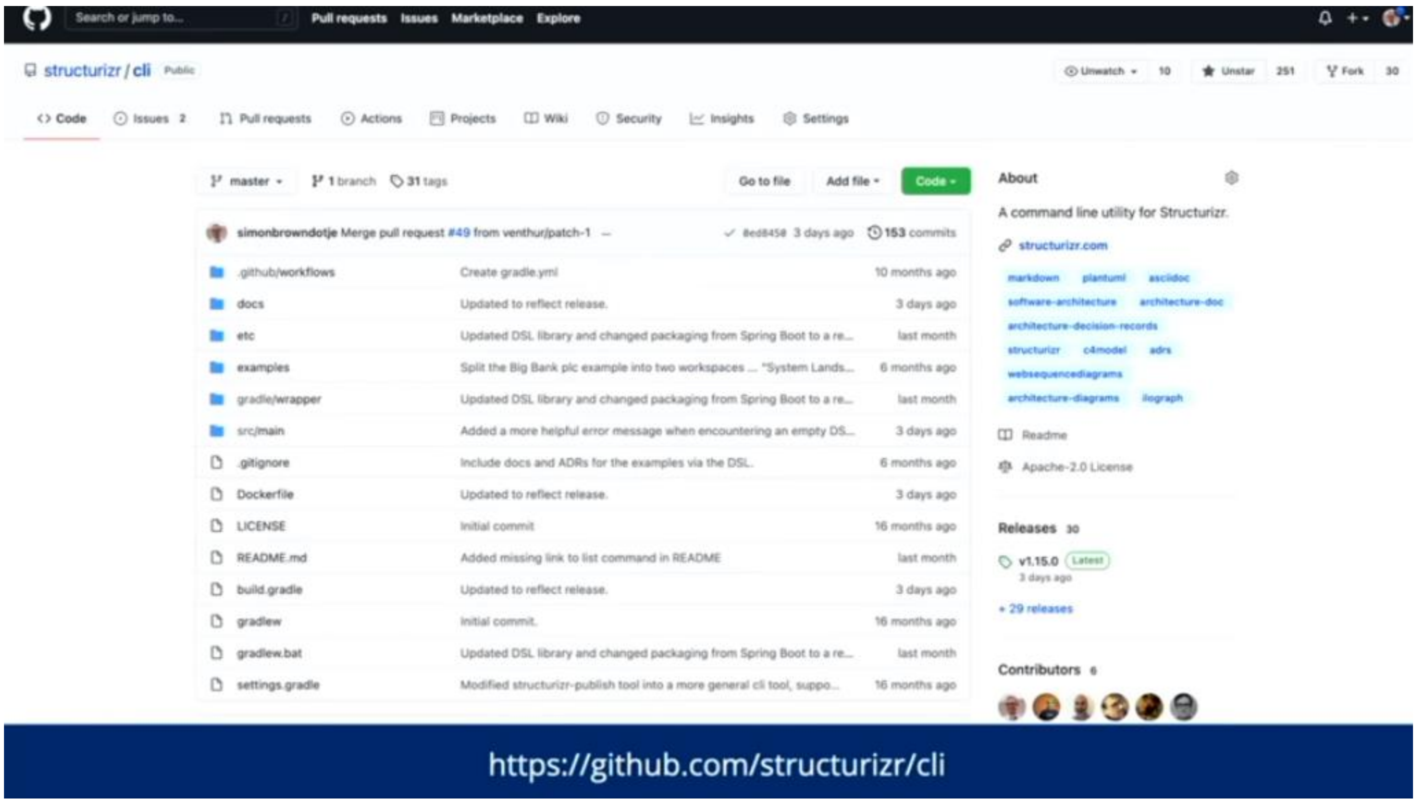
Packaged as a Docker container, and designed for developers, this version of Structurizr provides a way to quickly work with a single workspace. It's free to use, and allows you to view/edit diagrams, view documentation, and view architecture decision records defined in a DSL or JSON workspace.

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

This is a free version of the Structurizr tool that is available as a Docker image from Docker Hub. This is a little web app that you can spin up, point it to your folder that contains the Structurizr DSL definition files, open up localhost:8080 in the browser and it will create a bunch of diagrams for you like below.




Search or jump to... Pull requests Issues Marketplace Explore

structurizr / cli Public

Unwatch 10 Unstar 251 Fork 30

Code Issues 2 Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 31 tags Go to file Add file Code

simonbrowndotje Merge pull request #49 from venthur/patch-1 ... 8ed8458 3 days ago 153 commits

.github/workflows	Create gradle.yml	10 months ago
docs	Updated to reflect release.	3 days ago
etc	Updated DSL library and changed packaging from Spring Boot to a re...	last month
examples	Split the Big Bank plc example into two workspaces ... "System Lands...	6 months ago
gradle/wrapper	Updated DSL library and changed packaging from Spring Boot to a re...	last month
src/main	Added a more helpful error message when encountering an empty DS...	3 days ago
.gitignore	Include docs and ADRs for the examples via the DSL.	6 months ago
Dockerfile	Updated to reflect release.	3 days ago
LICENSE	Initial commit	16 months ago
README.md	Added missing link to list command in README	last month
build.gradle	Updated to reflect release.	3 days ago
gradlew	Initial commit.	16 months ago
gradlew.bat	Updated DSL library and changed packaging from Spring Boot to a re...	last month
settings.gradle	Modified structurizr-publish tool into a more general cli tool, suppo...	16 months ago

About

A command line utility for Structurizr.

structurizr.com

markdown plantuml asciidoc software-architecture architecture-doc architecture-decision-records structurizr c4model adrs websequencediagrams architecture-diagrams ilograph

Readme

Apache-2.0 License

Releases 30

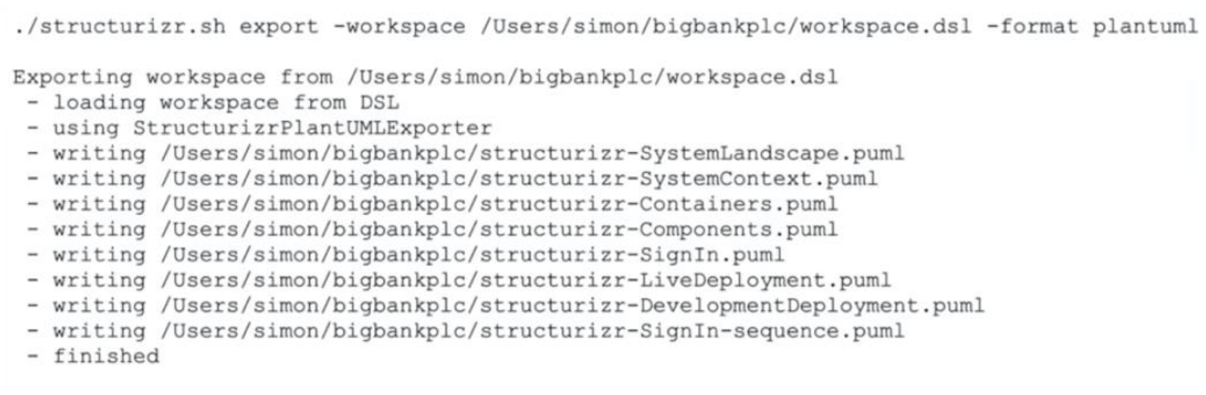
v1.15.0 Latest
3 days ago

+ 29 releases

Contributors 6

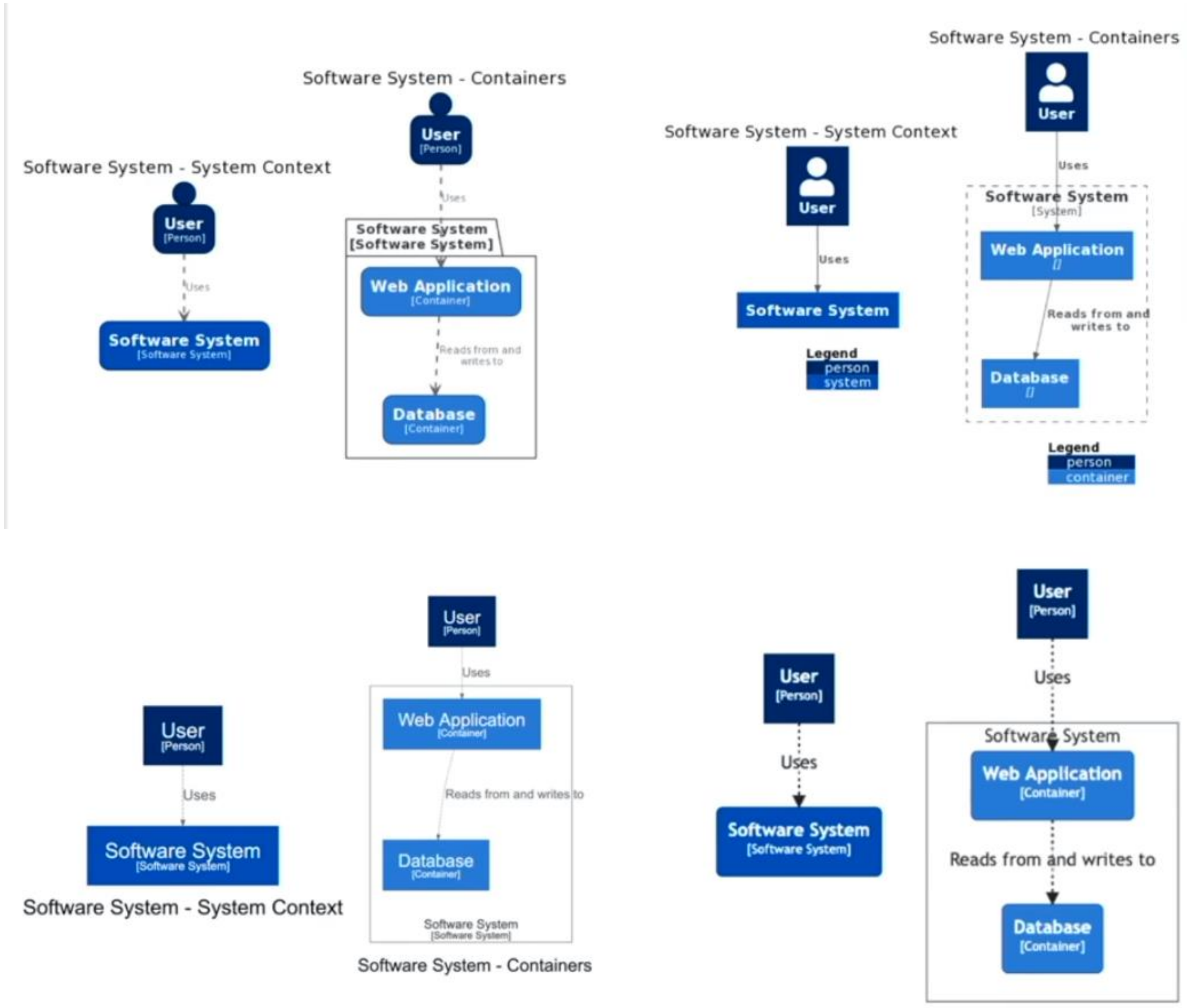
https://github.com/structurizr/cli

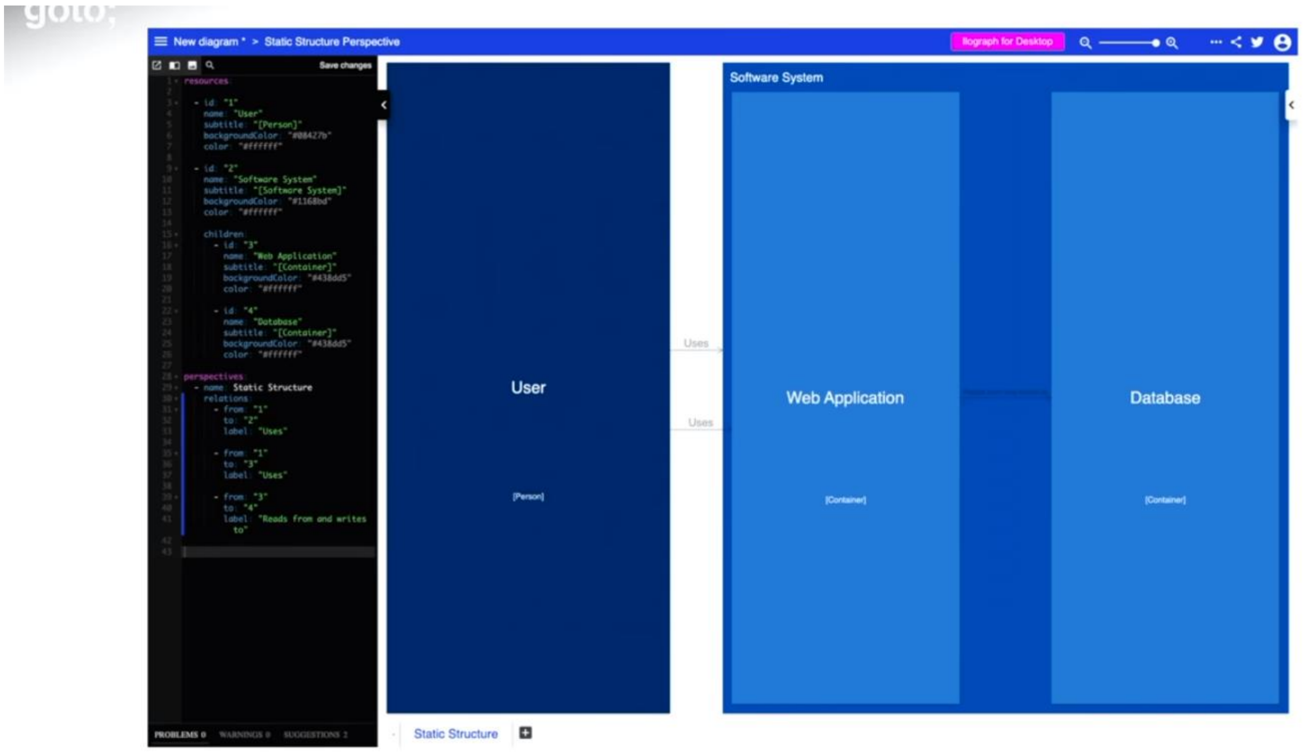
The CLI tool allows you to export the views you define in your Structurizr DSL definitions to various exort formats like PlantUML, Mermaid, etc.



```
./structurizr.sh export -workspace /Users/simon/bigbankplc/workspace.dsl -format plantuml

Exporting workspace from /Users/simon/bigbankplc/workspace.dsl
 - loading workspace from DSL
 - using StructurizrPlantUMLExporter
 - writing /Users/simon/bigbankplc/structurizr-SystemLandscape.puml
 - writing /Users/simon/bigbankplc/structurizr-SystemContext.puml
 - writing /Users/simon/bigbankplc/structurizr-Containers.puml
 - writing /Users/simon/bigbankplc/structurizr-Components.puml
 - writing /Users/simon/bigbankplc/structurizr-SignIn.puml
 - writing /Users/simon/bigbankplc/structurizr-LiveDeployment.puml
 - writing /Users/simon/bigbankplc/structurizr-DevelopmentDeployment.puml
 - writing /Users/simon/bigbankplc/structurizr-SignIn-sequence.puml
 - finished
```

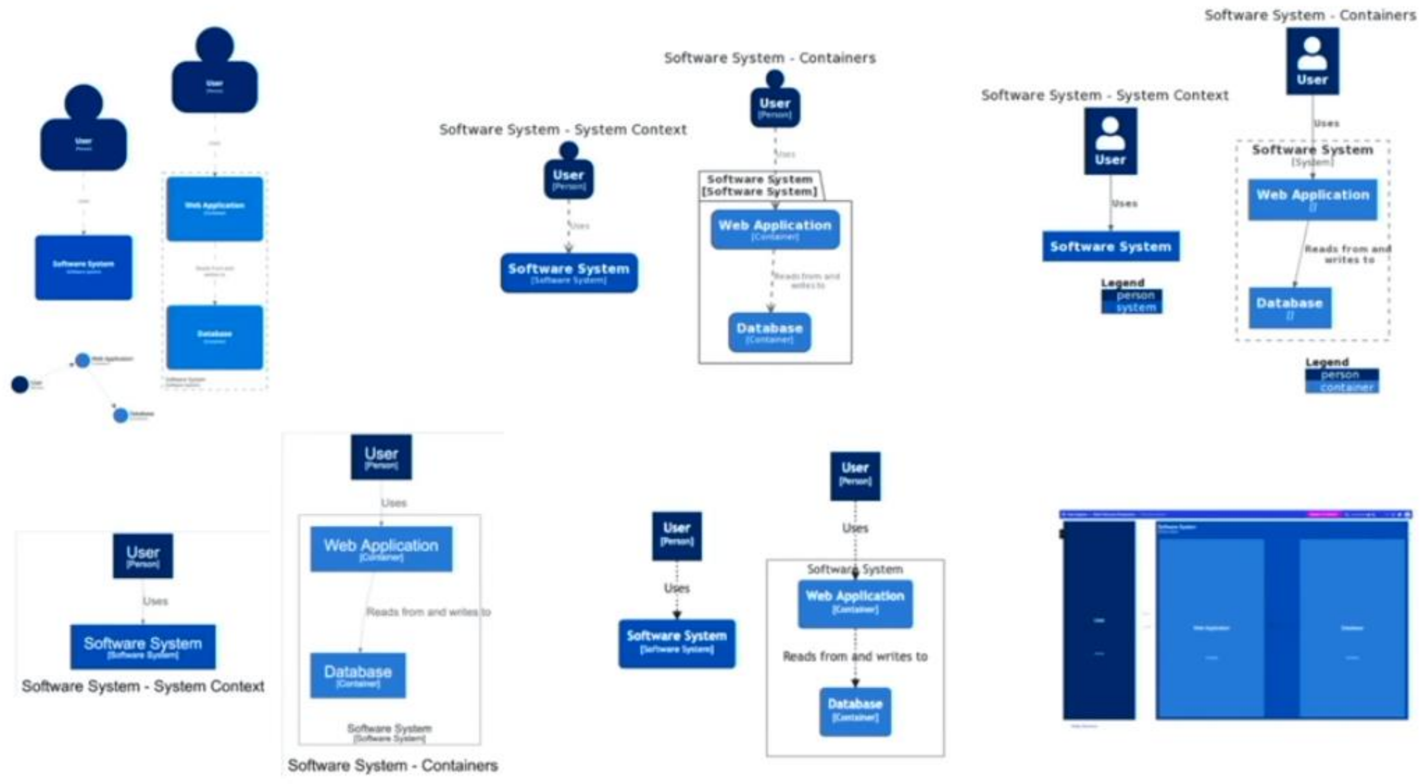
So, you spin up the tool, point it to a workspace definition and export in a format like PlantUML as above to get the files.





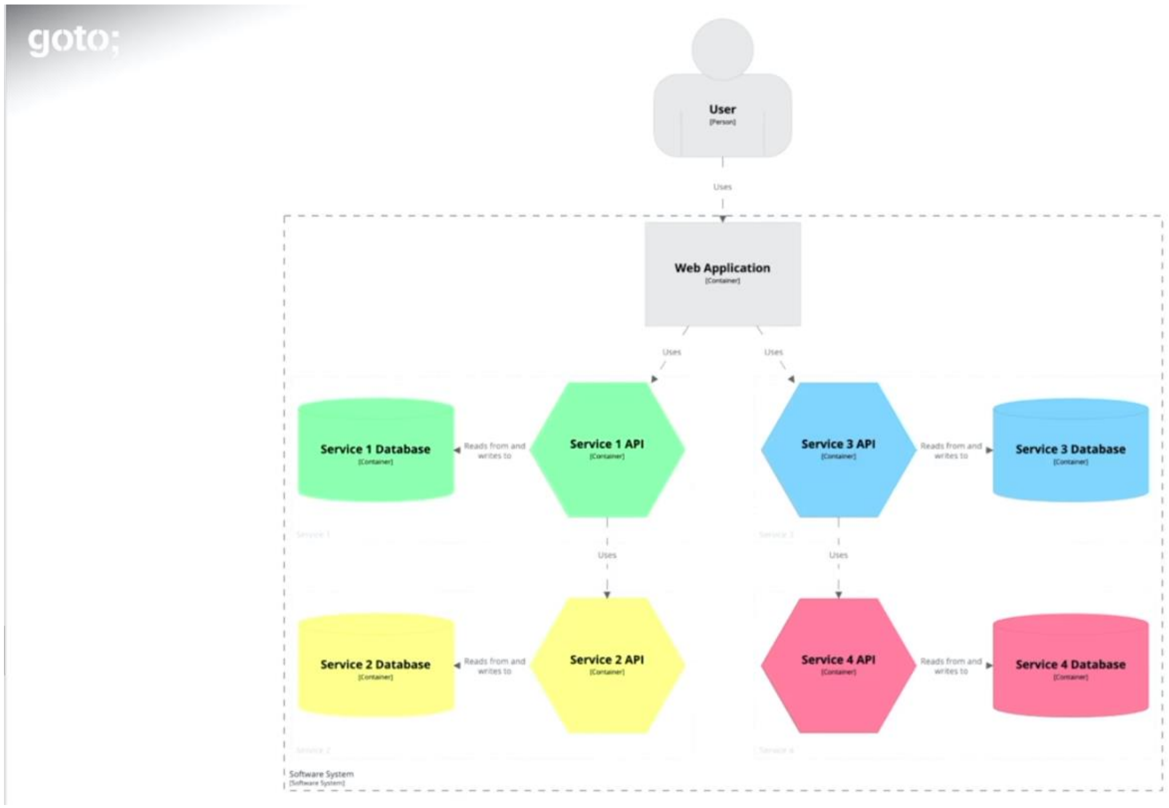
```yaml
resources:

  - id: "1"
    name: "User"
    subtitle: "[Person]"
    backgroundColor: "#08427b"
    color: "#ffffff"

  - id: "2"
    name: "Software System"
    subtitle: "[Software System]"
    backgroundColor: "#1168bd"
    color: "#ffffff"

    children:
      - id: "3"
        name: "Web Application"
        subtitle: "[Container]"
        backgroundColor: "#438dd5"
        color: "#ffffff"

      - id: "4"
        name: "Database"
        subtitle: "[Container]"
        backgroundColor: "#438dd5"
        color: "#ffffff"

perspectives:
  - name: Static Structure
    relations:
      - from: "1"
        to: "2"
        label: "Uses"

      - from: "1"
        to: "3"
        label: "Uses"

      - from: "3"
        to: "4"
        label: "Reads from and writes
          to"

```

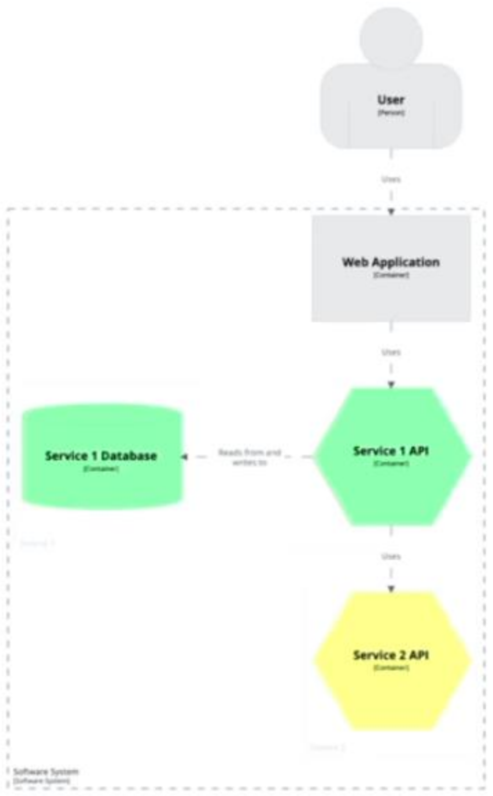
This is a sample Heliograph

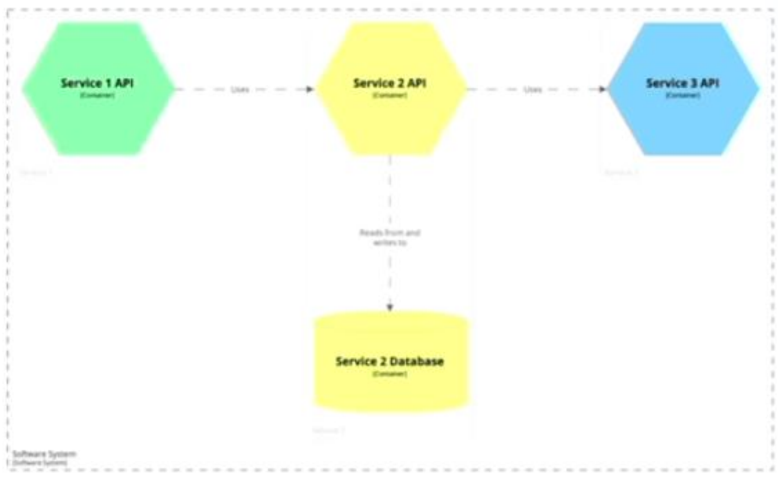


More advanced features

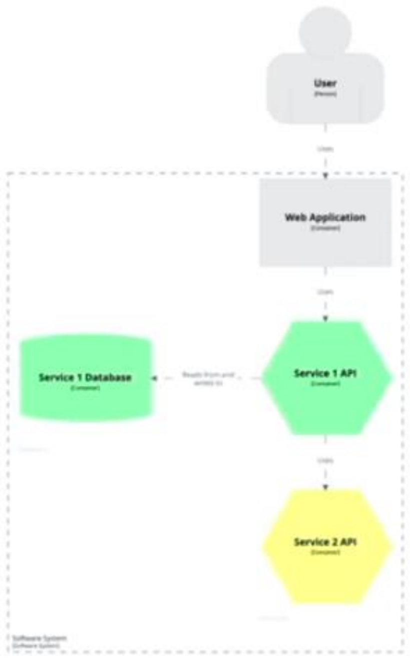
How do you diagram large and complex software systems?

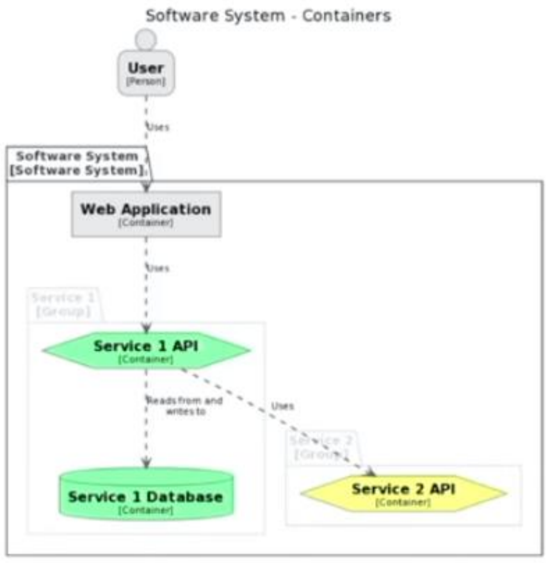




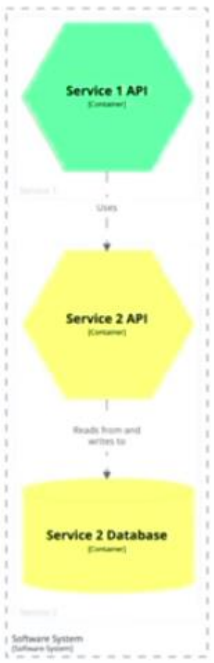


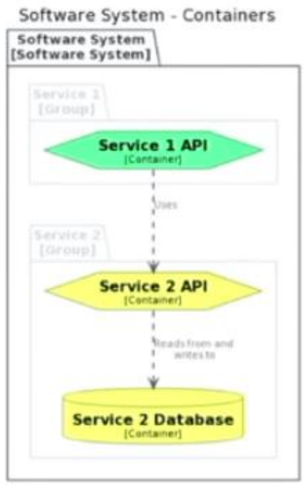
```
container softwareSystem {
    include user ->service1->
    autolayout
}
```



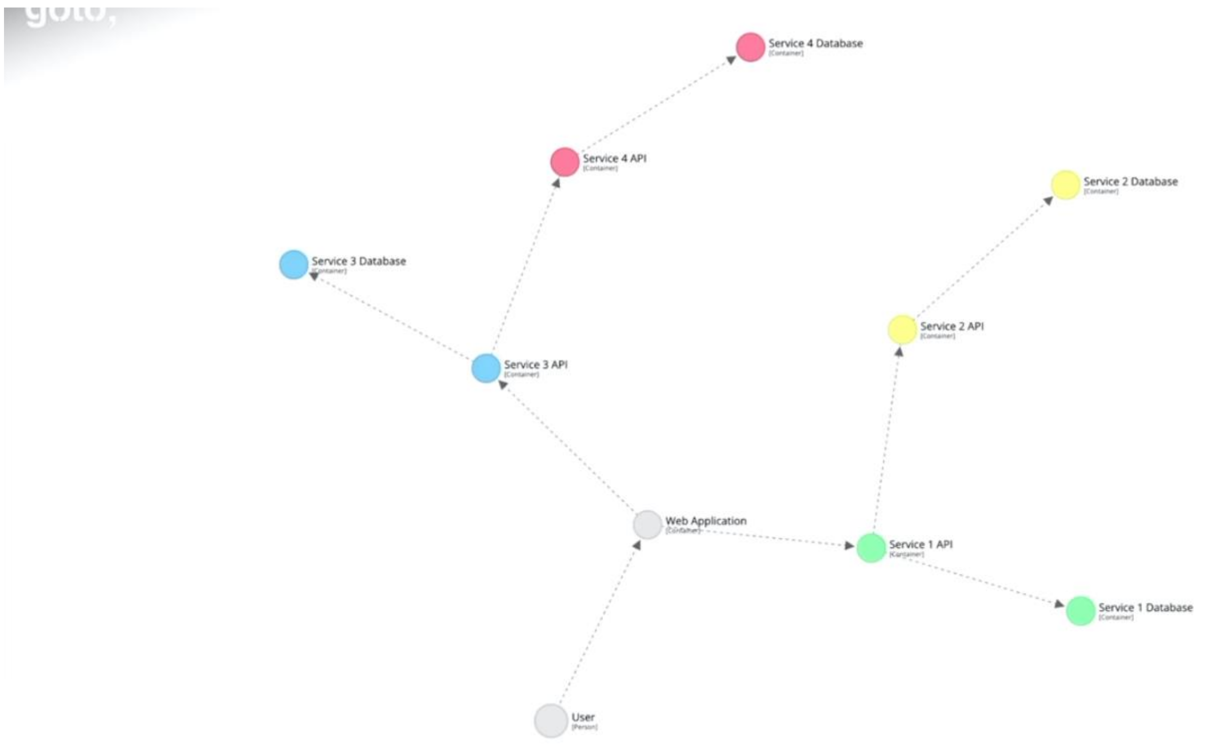


```
container softwareSystem {
    include ->service2->
    autolayout
}
```



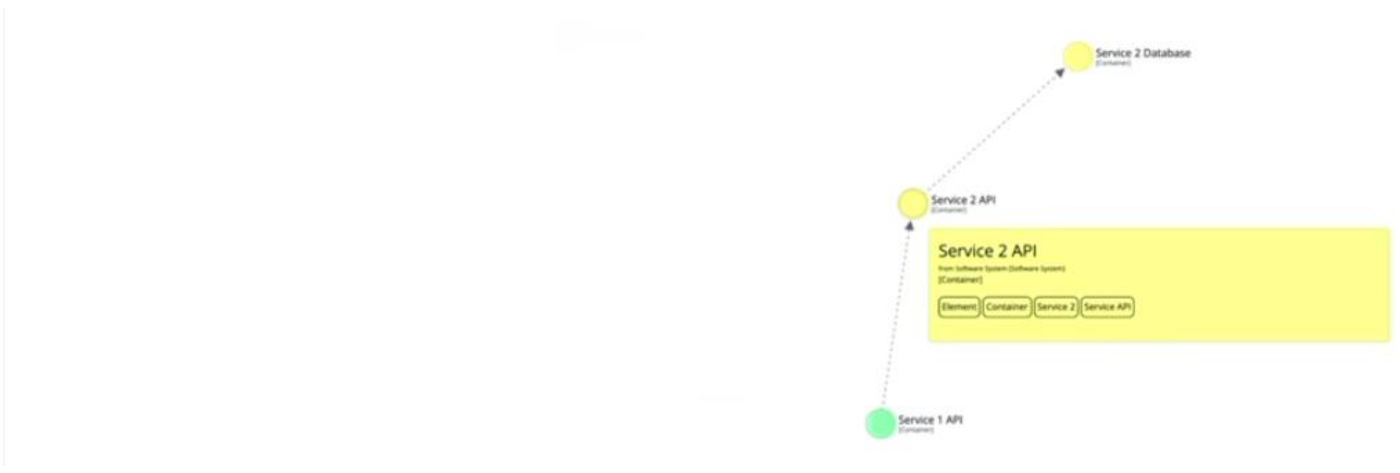


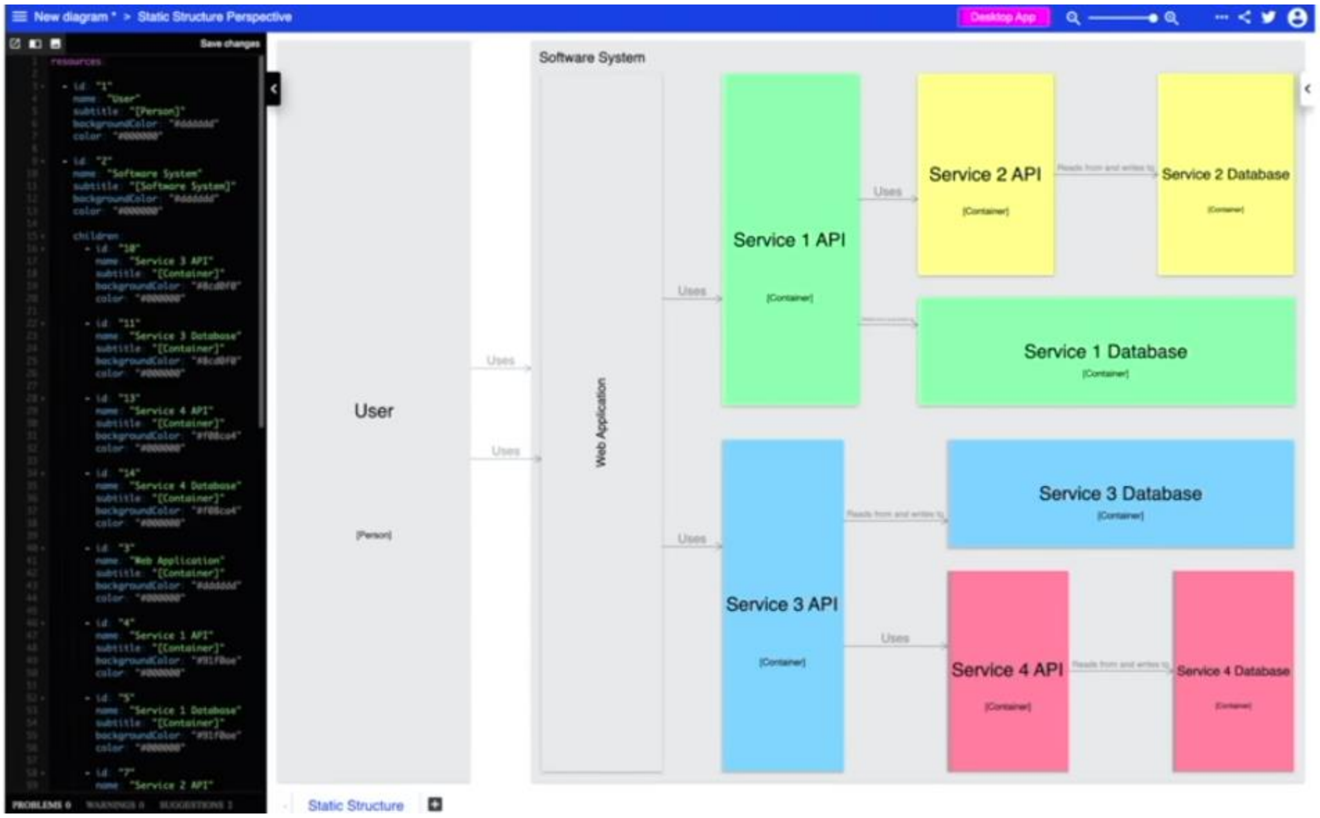
This is one way



Another approach is to use the D3.js view as a force-directed graph above, you can click on anode to see it as below

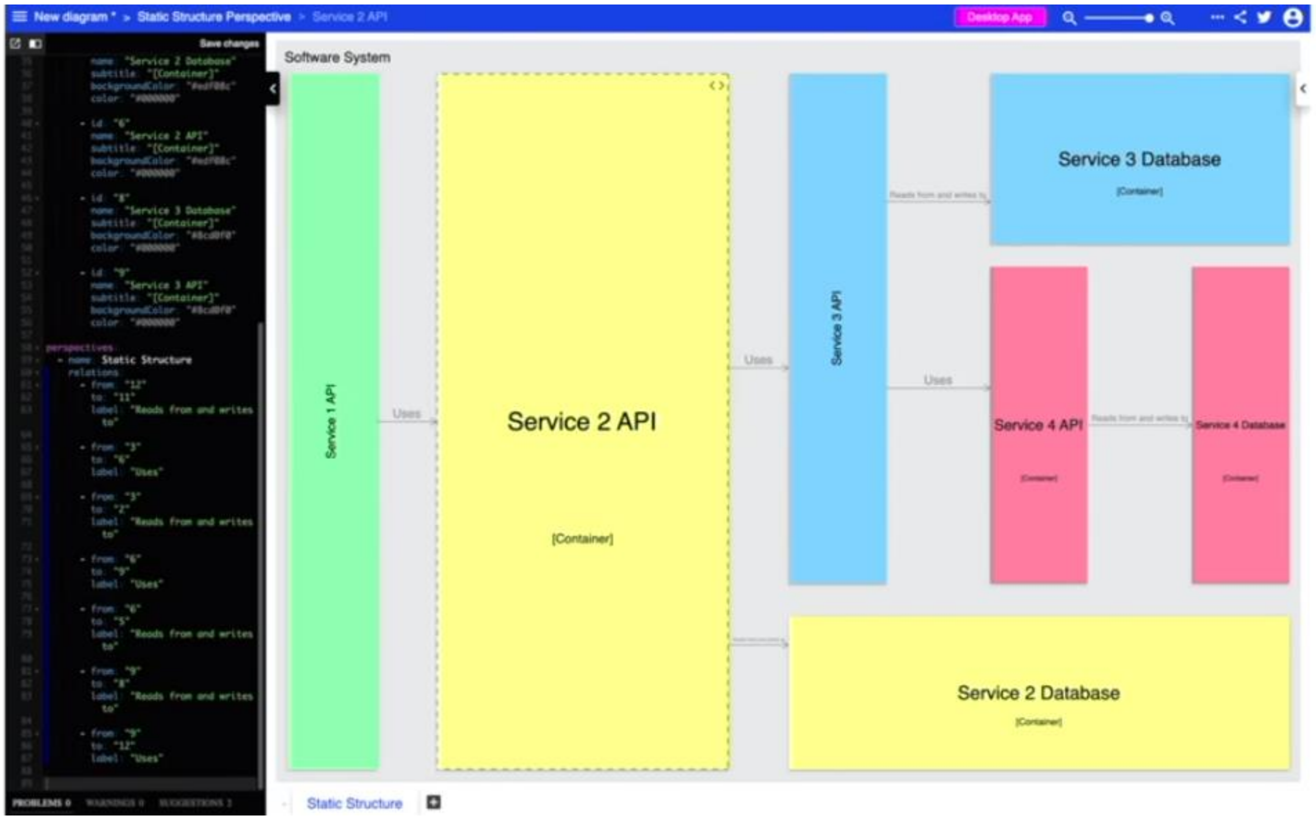
Service 2 Database
[Container]
Service 2 API
[Container]
Service 1 API
[Container]





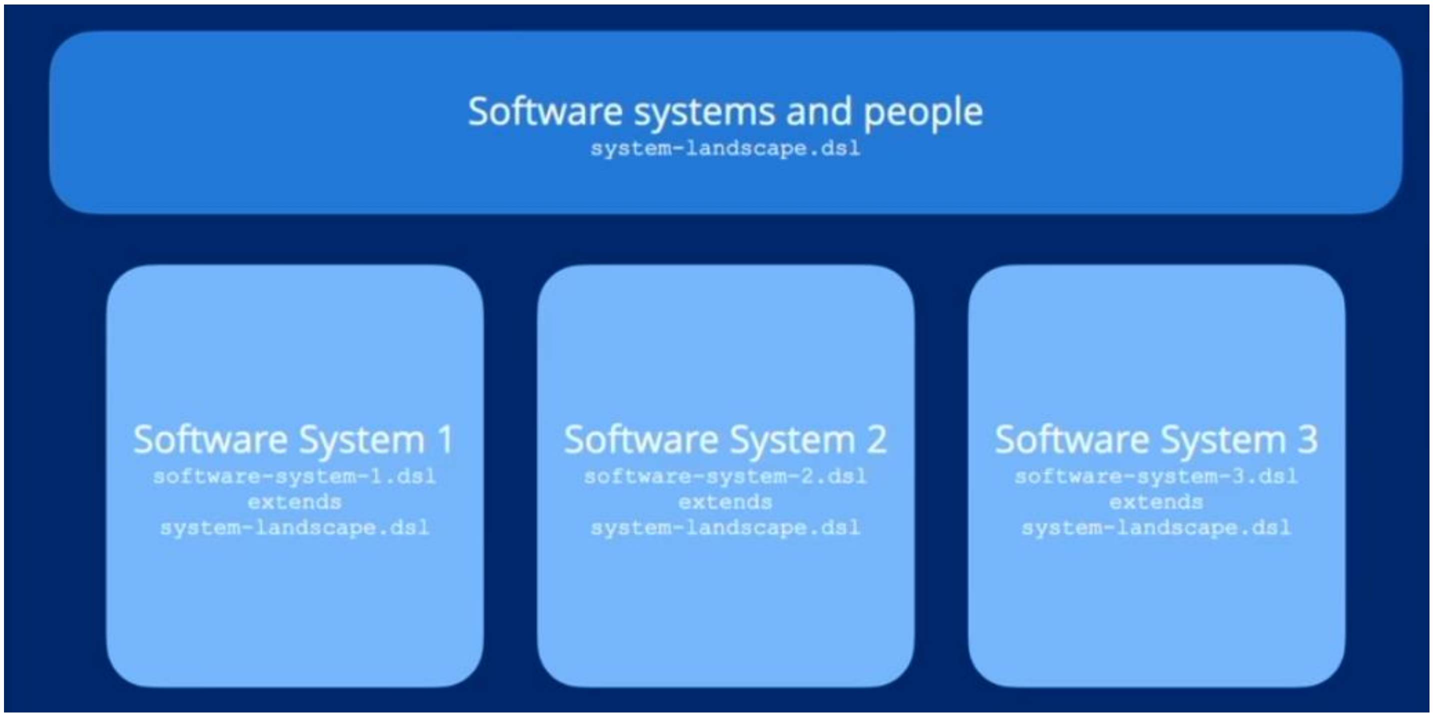
```
resources:

  - id: "1"
    name: "User"
    subtitle: "[Person]"
    backgroundColor: "#dddddd"
    color: "#000000"

  - id: "2"
    name: "Software System"
    subtitle: "[Software System]"
    backgroundColor: "#dddddd"
    color: "#000000"

    children:
      - id: "10"
        name: "Service 3 API"
        subtitle: "[Container]"
        backgroundColor: "#8cd0f0"
        color: "#000000"

      - id: "11"
        name: "Service 3 Database"
        subtitle: "[Container]"
        backgroundColor: "#8cd0f0"
        color: "#000000"

      - id: "13"
        name: "Service 4 API"
        subtitle: "[Container]"
        backgroundColor: "#f08ca4"
        color: "#000000"

      - id: "14"
        name: "Service 4 Database"
        subtitle: "[Container]"
        backgroundColor: "#f08ca4"
        color: "#000000"

      - id: "3"
        name: "Web Application"
        subtitle: "[Container]"
        backgroundColor: "#dddddd"
        color: "#000000"

      - id: "4"
        name: "Service 1 API"
        subtitle: "[Container]"
        backgroundColor: "#91f0ae"
        color: "#000000"

      - id: "5"
        name: "Service 1 Database"
        subtitle: "[Container]"
        backgroundColor: "#91f0ae"
        color: "#000000"

      - id: "7"
        name: "Service 2 API"
```

You can also load it back into Heliograph to see the above view



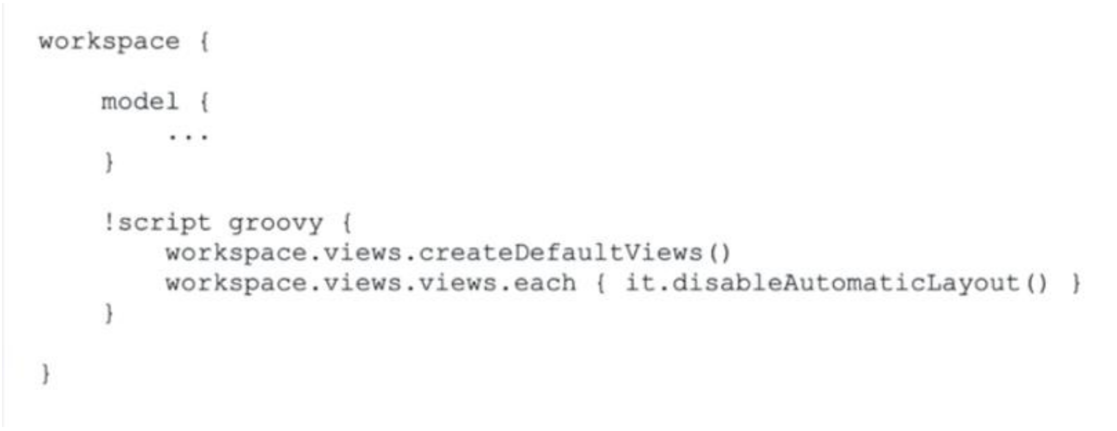
```
        name: "Service 2 Database"
        subtitle: "[Container]"
        backgroundColor: "#edf08c"
        color: "#000000"

      - id: "6"
        name: "Service 2 API"
        subtitle: "[Container]"
        backgroundColor: "#edf08c"
        color: "#000000"

      - id: "8"
        name: "Service 3 Database"
        subtitle: "[Container]"
        backgroundColor: "#8cd0f0"
        color: "#000000"

      - id: "9"
        name: "Service 3 API"
        subtitle: "[Container]"
        backgroundColor: "#8cd0f0"
        color: "#000000"

perspectives:
  - name: Static Structure
    relations:
      - from: "12"
        to: "11"
        label: "Reads from and writes
          to"

      - from: "3"
        to: "6"
        label: "Uses"

      - from: "3"
        to: "2"
        label: "Reads from and writes
          to"

      - from: "6"
        to: "9"
        label: "Uses"

      - from: "6"
        to: "5"
        label: "Reads from and writes
          to"

      - from: "9"
        to: "8"
        label: "Reads from and writes
          to"

      - from: "9"
        to: "12"
        label: "Uses"
```

Enterprise-wide modelling?



Scripting support

(via JSR-223: Java Scripting API)



```
workspace {

    model {
        ...
    }

    !script groovy {
        workspace.views.createDefaultViews()
        workspace.views.views.each { it.disableAutomaticLayout() }
    }

}
```

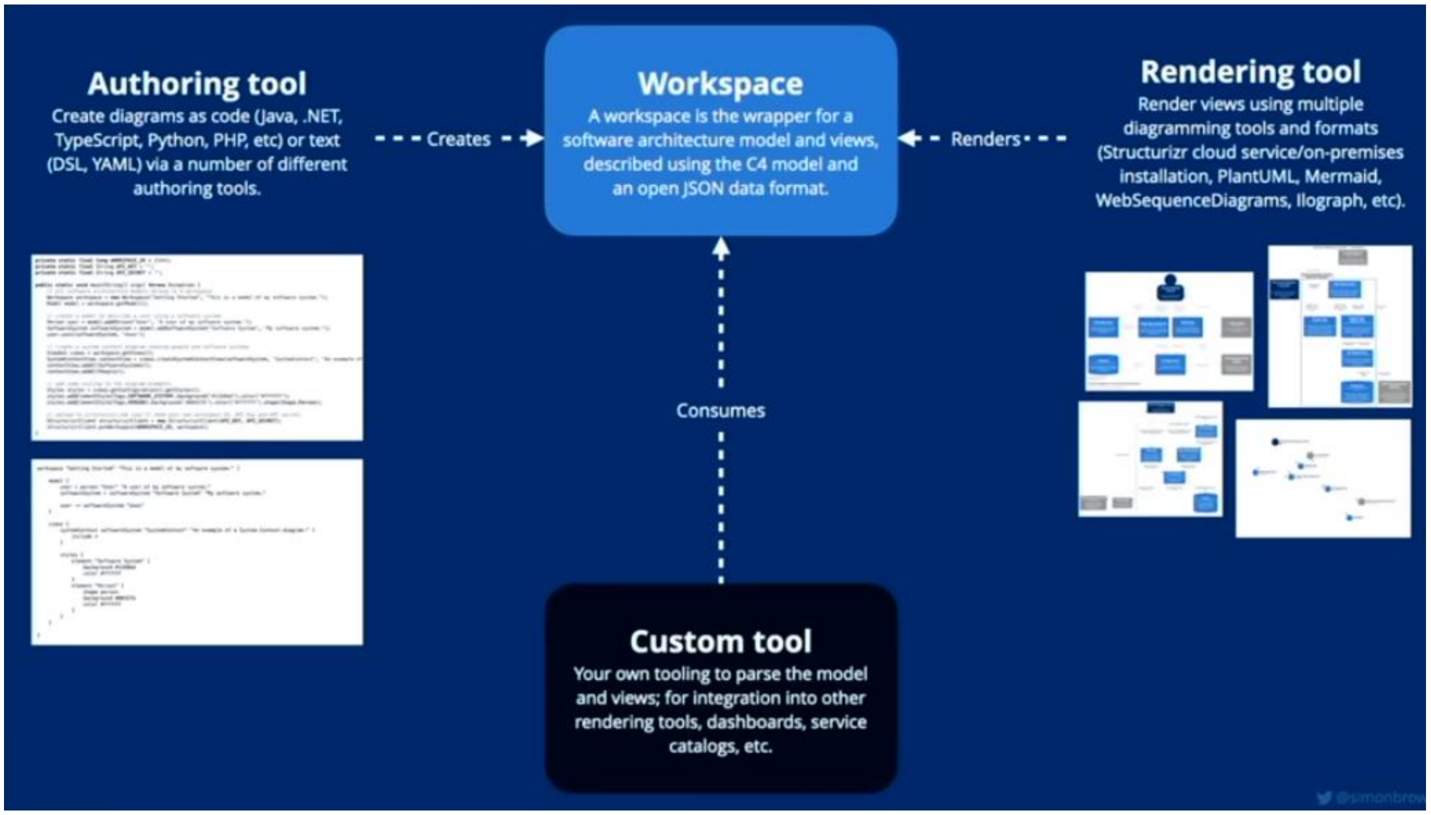
Plugin support

(via Java)

```
workspace {

    model {
        s = softwareSystem "Software System" {
            webapp = container "Web Application"
            database = container "Database" {
                webapp -> this "Reads from and writes to"
            }
        }
    }

    views {
        systemContext s {
            include *
            autoLayout lr
        }

        container s {
            include *
            autoLayout lr
        }
    }

}
```

```
StructurizrDslParser parser = new StructurizrDslParser();
parser.parse(new File("workspace.dsl"));

Workspace workspace = parser.getWorkspace();
Container webApplication = workspace.getModel()
        .getSoftwareSystemWithName("Software System”)
        .getContainerWithName(“Web Application");

// add components manually or via automatic extraction
...

// add a component view
ComponentView componentView = workspace.getViews()
        .createComponentView(webApplication, "Components", "Description");
componentView.addDefaultElements();
componentView.enableAutomaticLayout();
```

Custom tooling



Authoring tool
Create diagrams as code (Java, .NET, TypeScript, Python, PHP, etc) or text (DSL, YAML) via a number of different authoring tools.

Workspace
A workspace is the wrapper for a software architecture model and views, described using the C4 model and an open JSON data format.

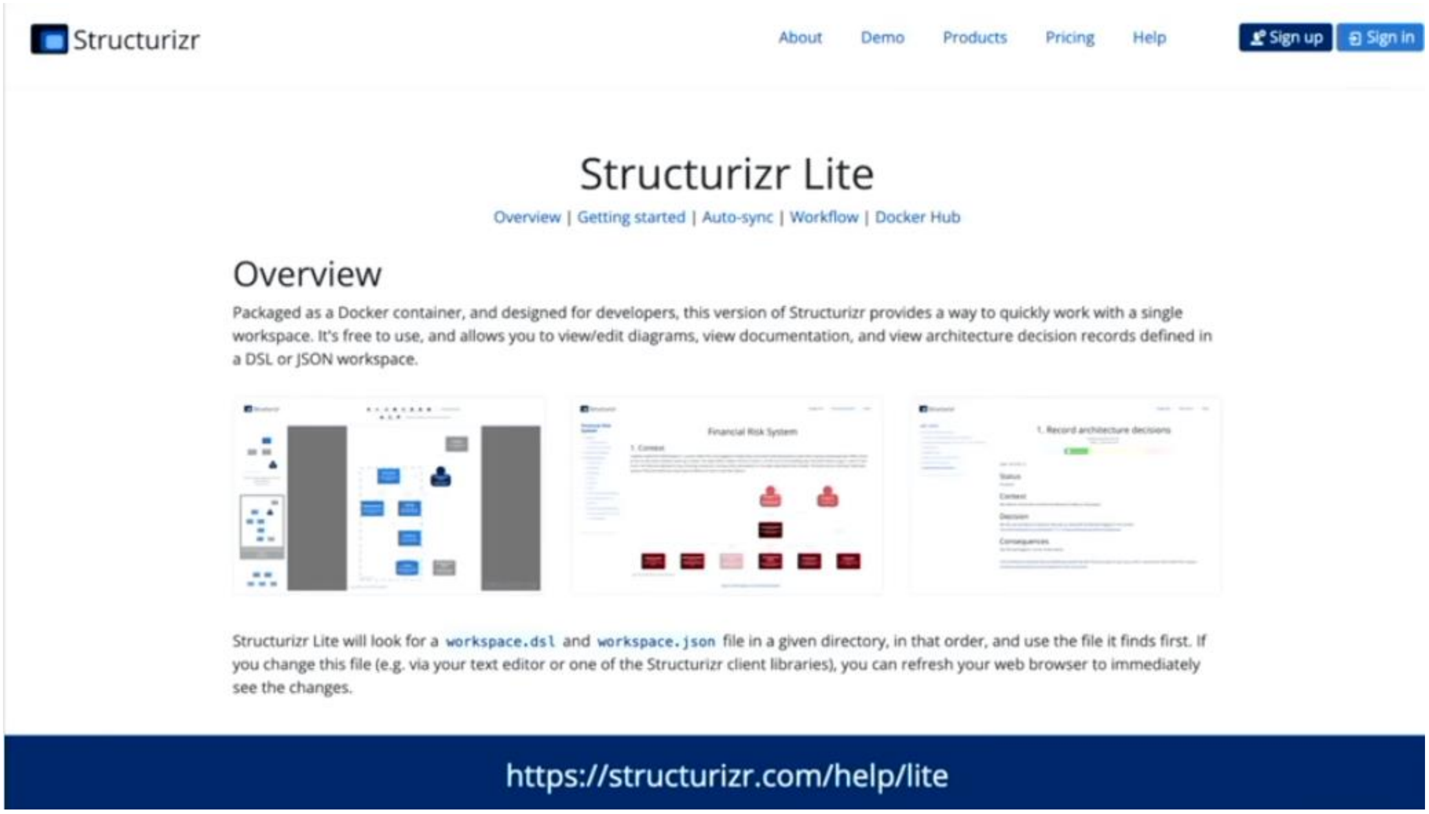
Rendering tool
Render views using multiple diagramming tools and formats (Structurizr cloud service/on-premises installation, PlantUML, Mermaid, WebSequenceDiagrams, Ilograph, etc).

Custom tool
Your own tooling to parse the model and views; for integration into other rendering tools, dashboards, service catalogs, etc.

Usage scenarios

Static diagrams
(e.g. PNG/SVG)

Interactive diagrams
(e.g. browser-based)



Structurizr Lite

Overview | Getting started | Auto-sync | Workflow | Docker Hub

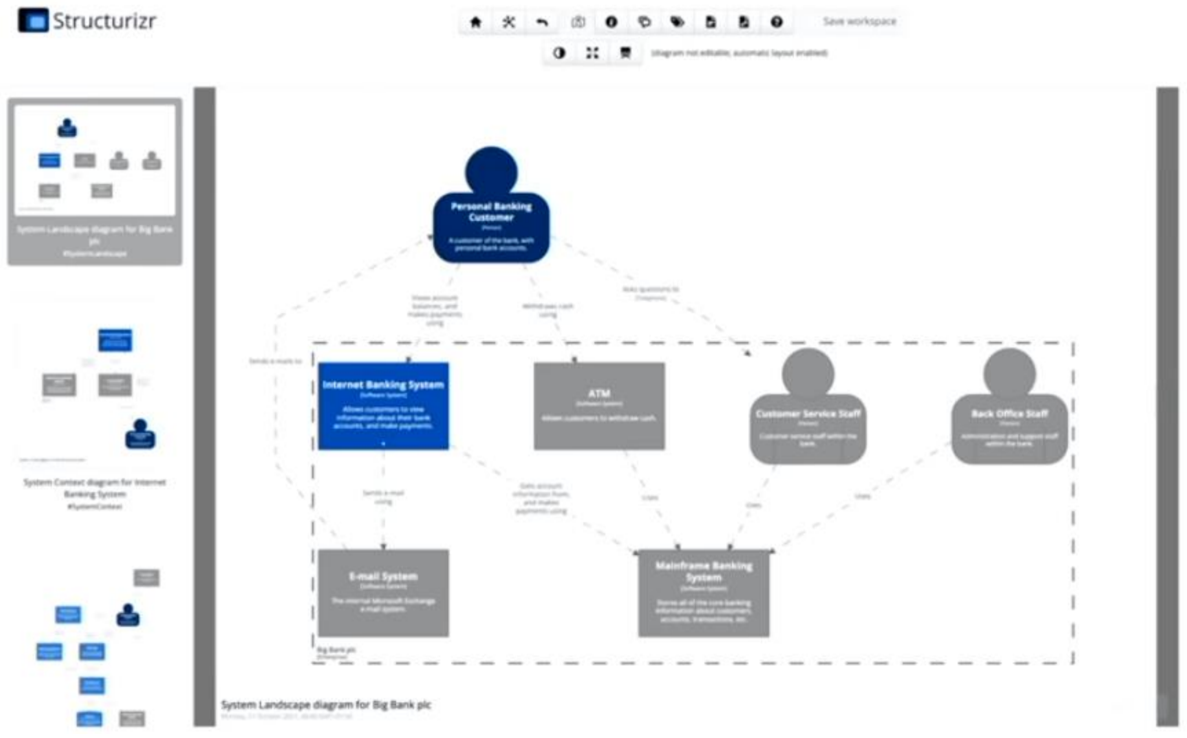
Overview

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.

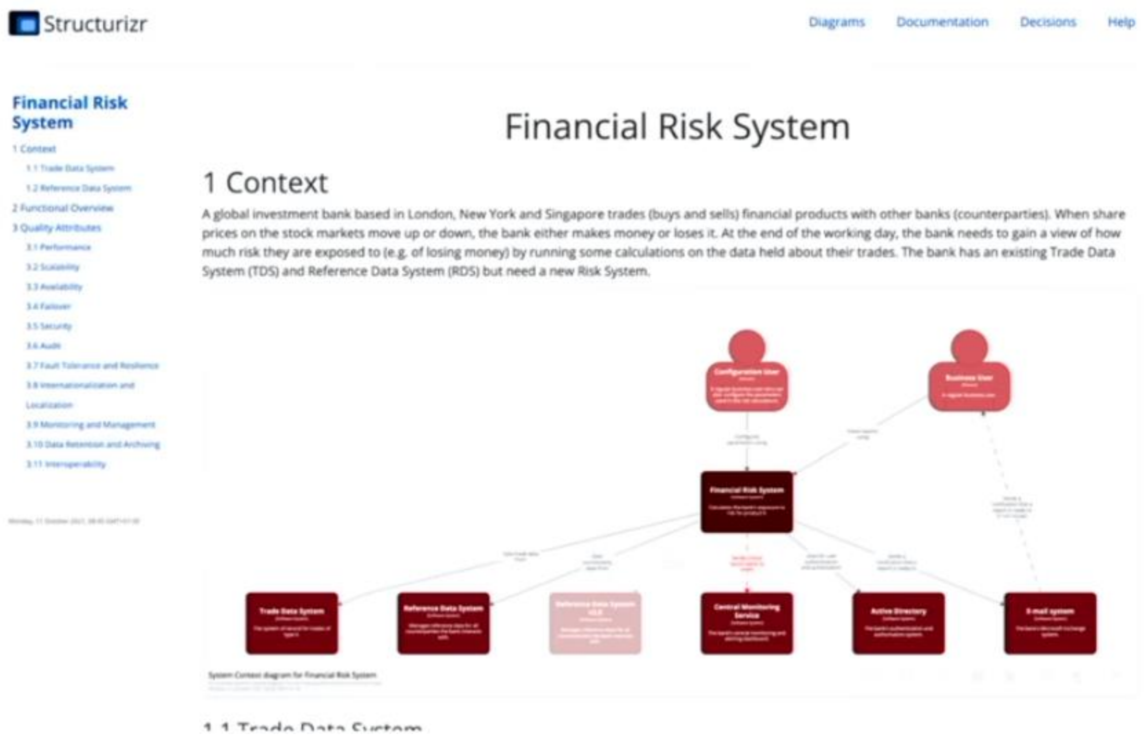
https://structurizr.com/help/lite

```
docker run -it --rm -p 8080:8080 -v /Users/simon/bigbankplc/:/usr/local/structurizr structurizr/lite
```



Structurizr
Save workspace
(diagram not editable; automatic layout enabled)
Personal Banking Customer
Internet Banking System
ATM
Customer Service Staff
Back Office Staff
E-mail System
Mainframe Banking System
Big Bank plc
System Landscape diagram for Big Bank plc
System Context diagram for Internet Banking System

```
!docs <directory name>
```



Structurizr

Diagrams Documentation Decisions Help

Financial Risk System

1 Context
- 1.1 Trade Data System
- 1.2 Reference Data System

2 Functional Overview

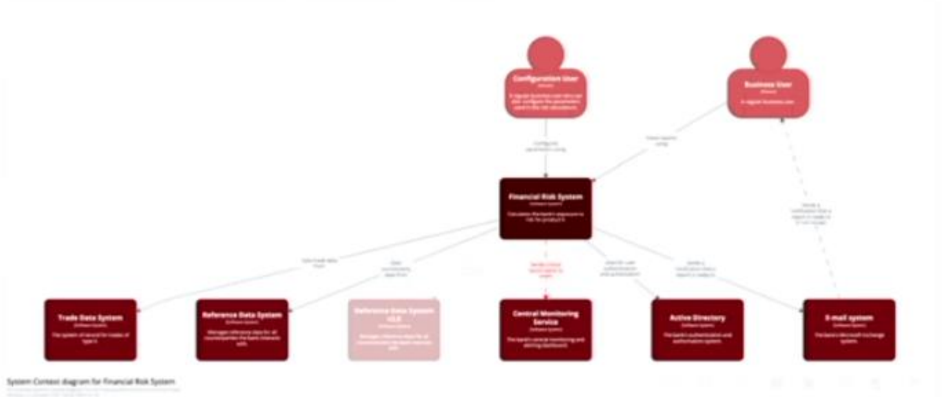
3 Quality Attributes
- 3.1 Performance
- 3.2 Scalability
- 3.3 Availability
- 3.4 Failover
- 3.5 Security
- 3.6 Audit
- 3.7 Fault Tolerance and Resilience
- 3.8 Internationalization and Localization
- 3.9 Monitoring and Management
- 3.10 Data Retention and Archiving
- 3.11 Interoperability

Monday, 11 October 2021, 08:45 GMT+01:00

Financial Risk System

1 Context

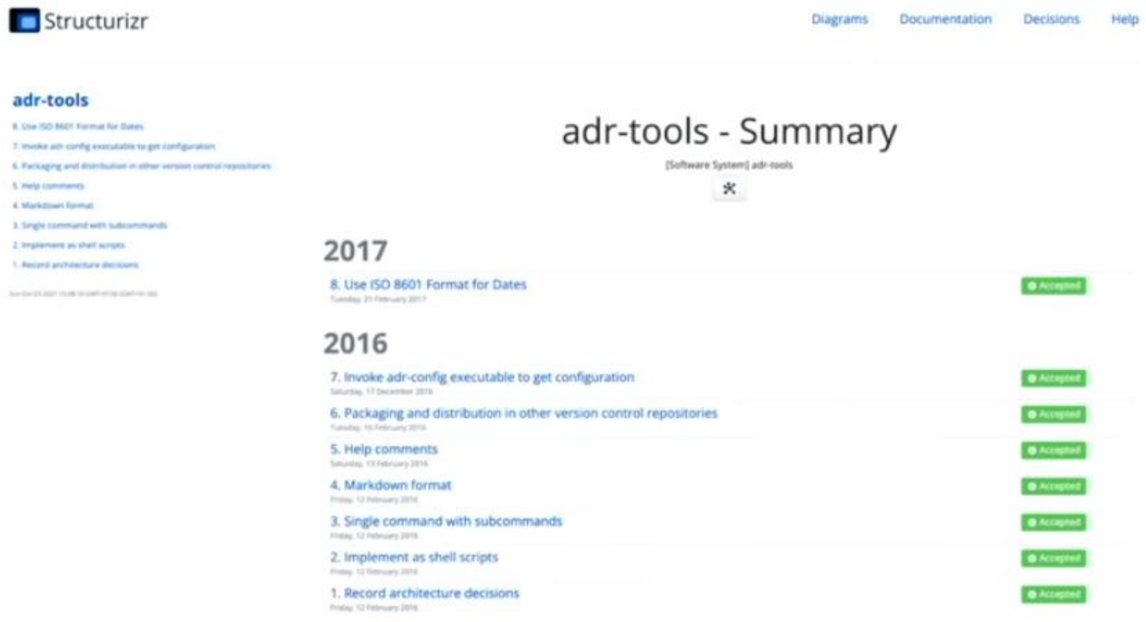
A global investment bank based in London, New York and Singapore trades (buys and sells) financial products with other banks (counterparties). When share prices on the stock markets move up or down, the bank either makes money or loses it. At the end of the working day, the bank needs to gain a view of how much risk they are exposed to (e.g. of losing money) by running some calculations on the data held about their trades. The bank has an existing Trade Data System (TDS) and Reference Data System (RDS) but need a new Risk System.



System Context diagram for Financial Risk System

1.1 Trade Data System

```
!adrs <directory name>
```



Structurizr

Diagrams Documentation Decisions Help

adr-tools

8. Use ISO 8601 Format for Dates
7. Invoke adr-config executable to get configuration
6. Packaging and distribution in other version control repositories
5. Help comments
4. Markdown format
3. Single command with subcommands
2. Implement as shell scripts
1. Record architecture decisions

[illegible]

adr-tools - Summary

[Software System] adr-tools

2017

8. Use ISO 8601 Format for Dates — Accepted
Tuesday, 21 February 2017

2016

7. Invoke adr-config executable to get configuration — Accepted
Saturday, 17 December 2016

6. Packaging and distribution in other version control repositories — Accepted
Tuesday, 16 February 2016

5. Help comments — Accepted
Saturday, 13 February 2016

4. Markdown format — Accepted
Friday, 12 February 2016

3. Single command with subcommands — Accepted
Friday, 12 February 2016

2. Implement as shell scripts — Accepted
Friday, 12 February 2016

1. Record architecture decisions — Accepted
Friday, 12 February 2016

Closing thoughts

"Diagrams as code" is easy to author, diff, version control, collaborate on, integrate into CI/CD, etc

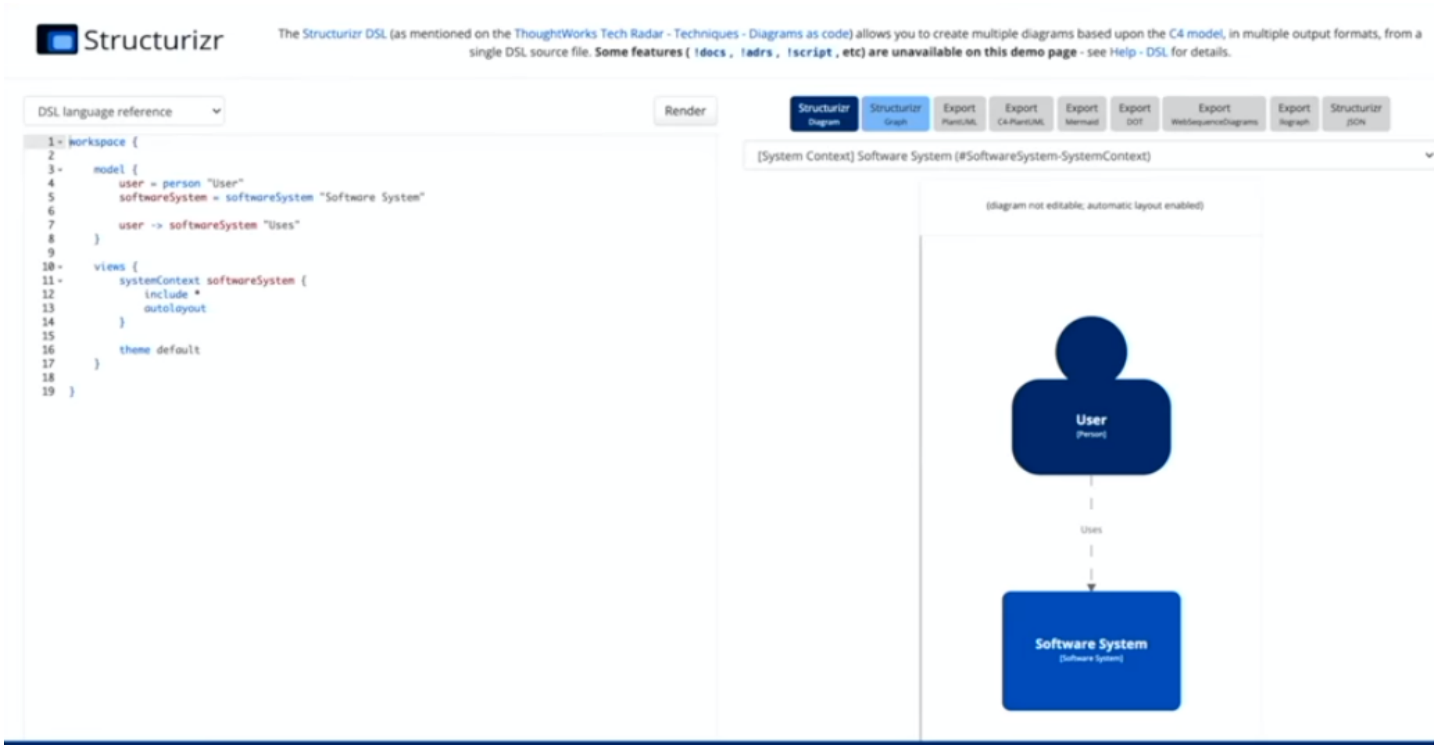
Developers
vs
non-developers?

Store your diagrams and docs
in version control,
next to your source code

"Publish" the diagrams and documentation if necessary

Up front design
vs
long-lived documentation?

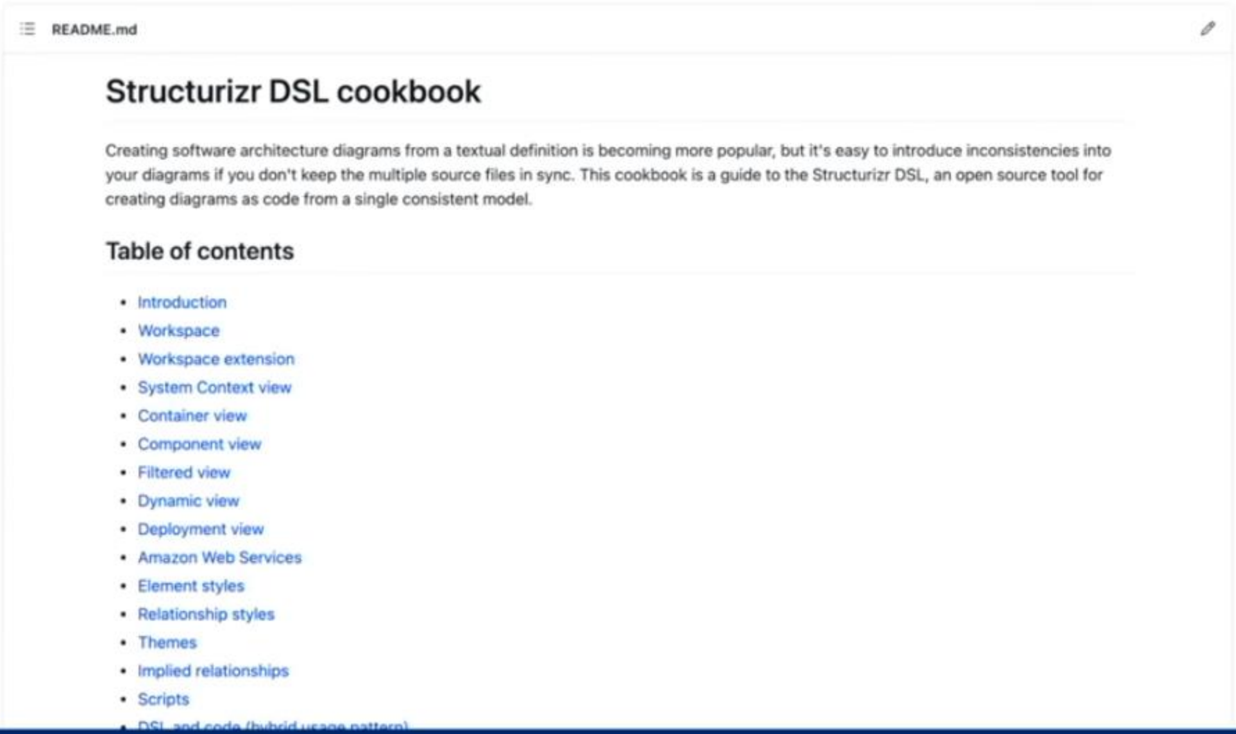
Think about diagrams as being "disposable" artefacts



The Structurizr DSL (as mentioned on the ThoughtWorks Tech Radar - Techniques - Diagrams as code) allows you to create multiple diagrams based upon the C4 model, in multiple output formats, from a single DSL source file. **Some features (`!docs` , `!adrs` , `!script` , etc) are unavailable on this demo page** - see Help - DSL for details.

```
workspace {

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

        user -> softwareSystem "Uses"
    }

    views {
        systemContext softwareSystem {
            include *
            autolayout
        }

        theme default
    }

}
```

https://structurizr.com/dsl



README.md

Structurizr DSL cookbook

Creating software architecture diagrams from a textual definition is becoming more popular, but it's easy to introduce inconsistencies into your diagrams if you don't keep the multiple source files in sync. This cookbook is a guide to the Structurizr DSL, an open source tool for creating diagrams as code from a single consistent model.

Table of contents

- Introduction
- Workspace
- Workspace extension
- System Context view
- Container view
- Component view
- Filtered view
- Dynamic view
- Deployment view
- Amazon Web Services
- Element styles
- Relationship styles
- Themes
- Implied relationships
- Scripts
- DSL and code (hybrid usage pattern)

https://github.com/structurizr/dsl/tree/master/docs/cookbook

Thank you!