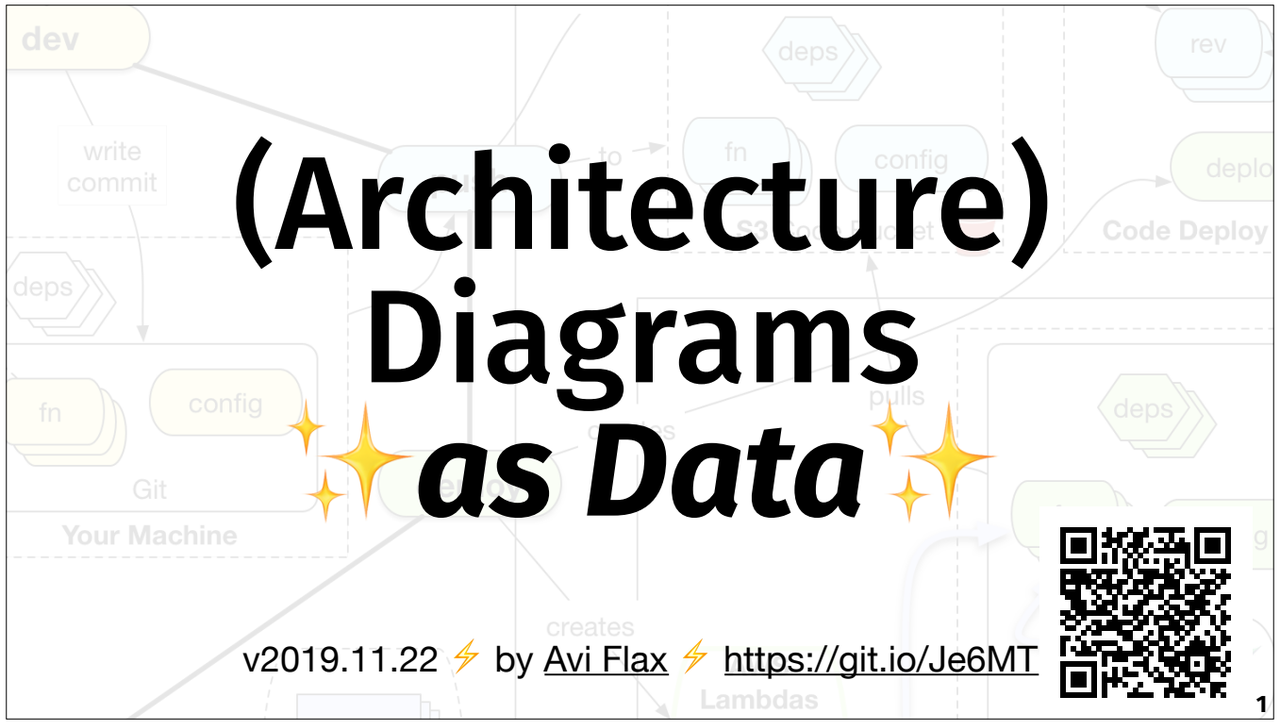


Introduce the talk

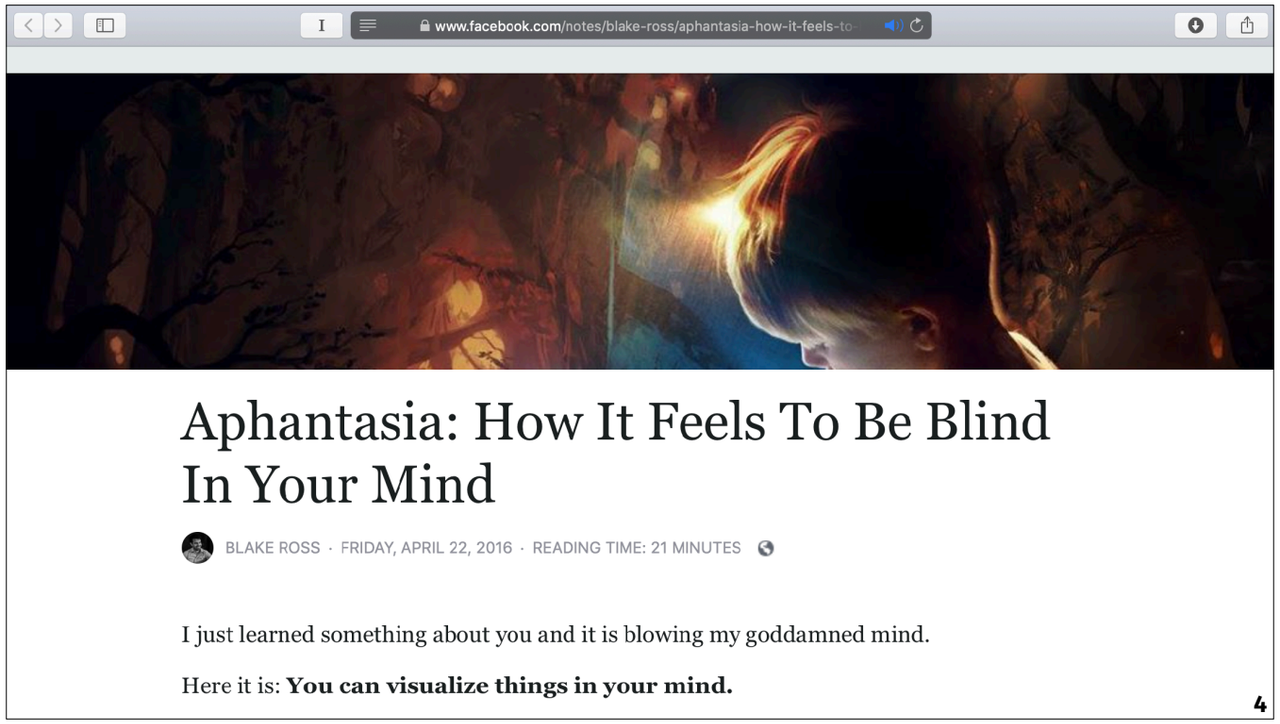
- I'm Avi, I'm here to talk about Software Architecture Diagrams —
    - I'm not sure how to pronounce the sparkles — ***as Data***
- Before I start:
    - Isn't this conference fantastic?
    - How great is this?
    - Such amazing people!
    - I love all of you!
- Also, there's a really cool talk happening next door…
    - If you realize at any point that that talk is a better fit for you,
        - please don't hesitate to get up and head over there;
        - I won't be offended.

2

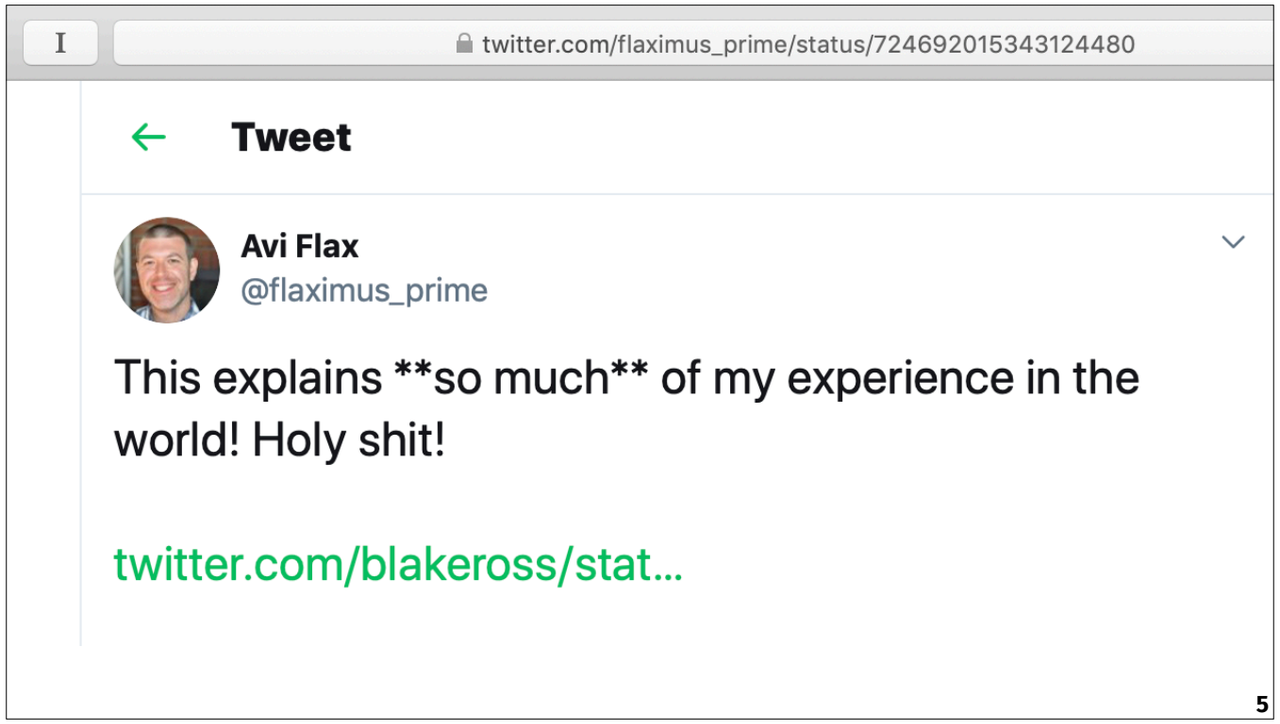
- In the spring of 2016,
    - I learned something that blew my mind
- More specifically,
    - I learned something mind-blowing
        - about *how my mind works*

- I was scrolling through Twitter when I came across this tweet by Blake Ross.
- {read the tweet}
- I read this,
    - Did a double take,
    - Said something like “holy shit”
    - And immediately clicked that link


twitter.com/blakeross/status/724321050985517056
Tweet
Blake Ross
@blakeross
I'm 30 and I just realized that people can actually "picture" things in their mind. I can't. It's called Aphantasia: facebook.com/notes/blake-ro...
3:36 PM · Apr 24, 2016

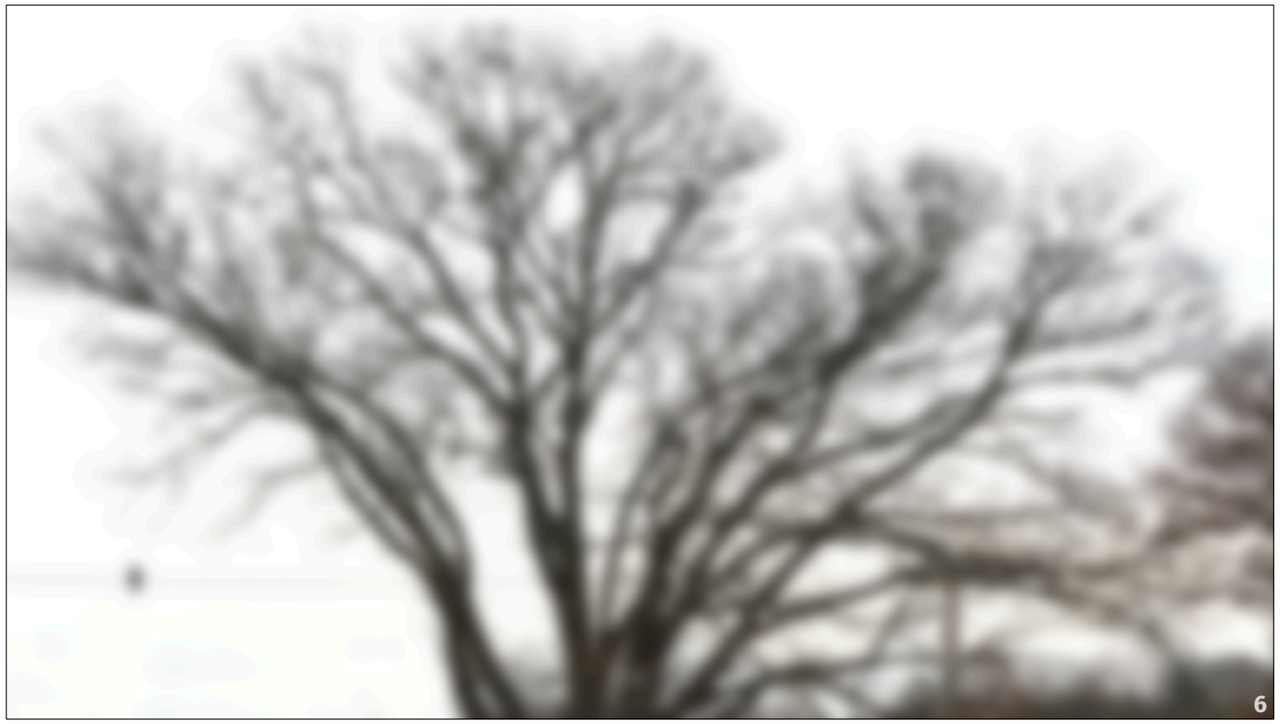


- The link led me to this essay he'd just written.
- I read the essay,
    - and I was astounded.
- That's how I found out that I have aphantasia.
    - A neurological variation wherein
    - I don't have a functioning "mind's eye"
    - I don't — *can't* — visualize images in my head

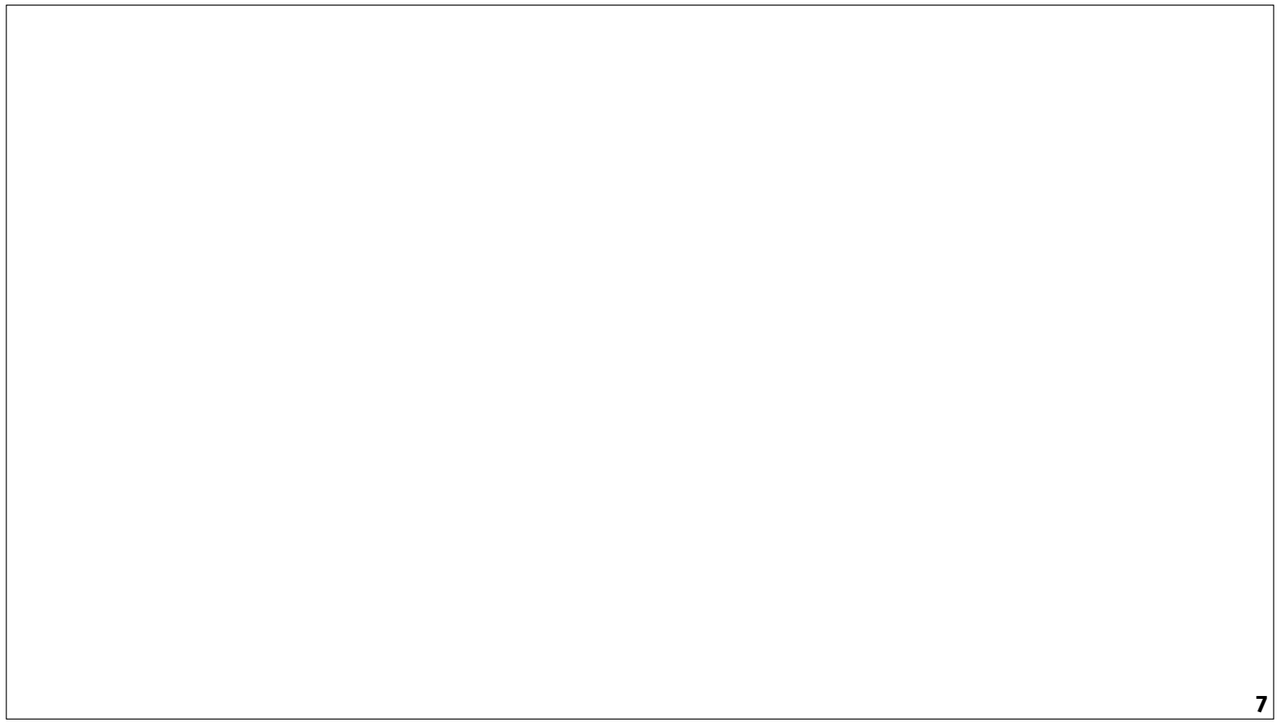


- This is how I felt at that moment.

- I don’t have time to go deep on aphantasia,
- But this explains *many* strange things about me.
- In particular,
    - it may explain some quirks of my memory →

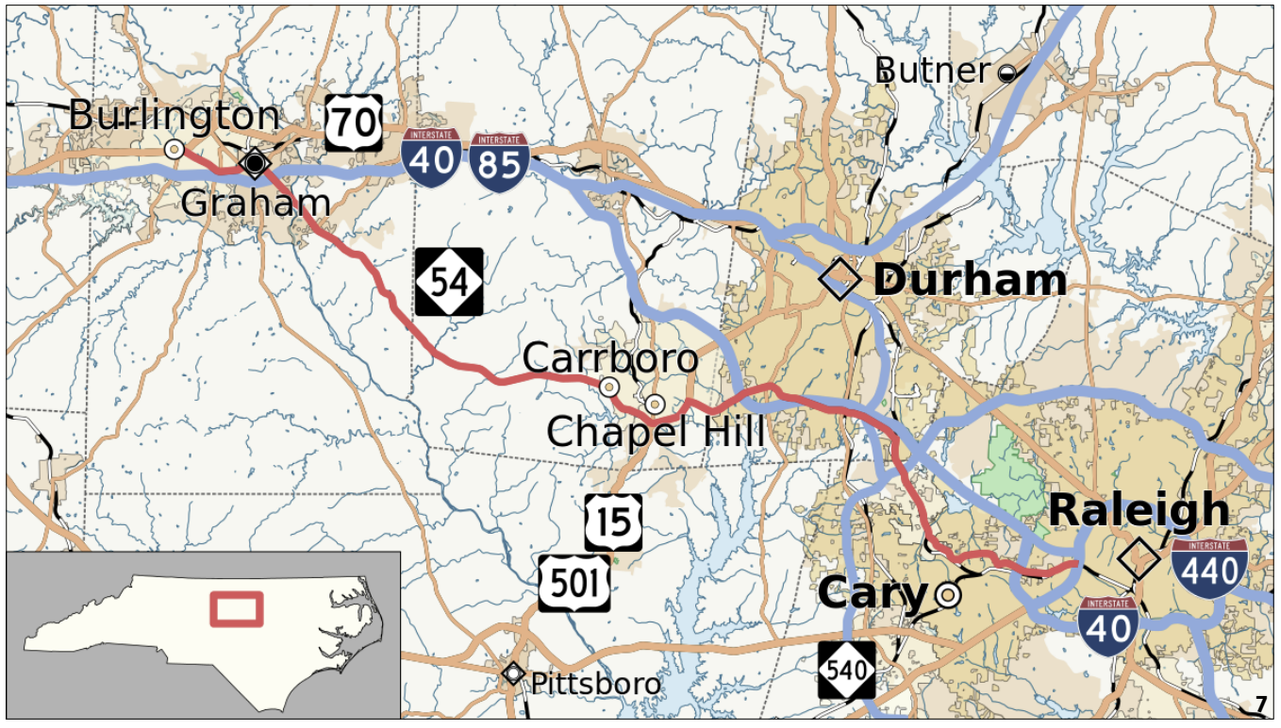


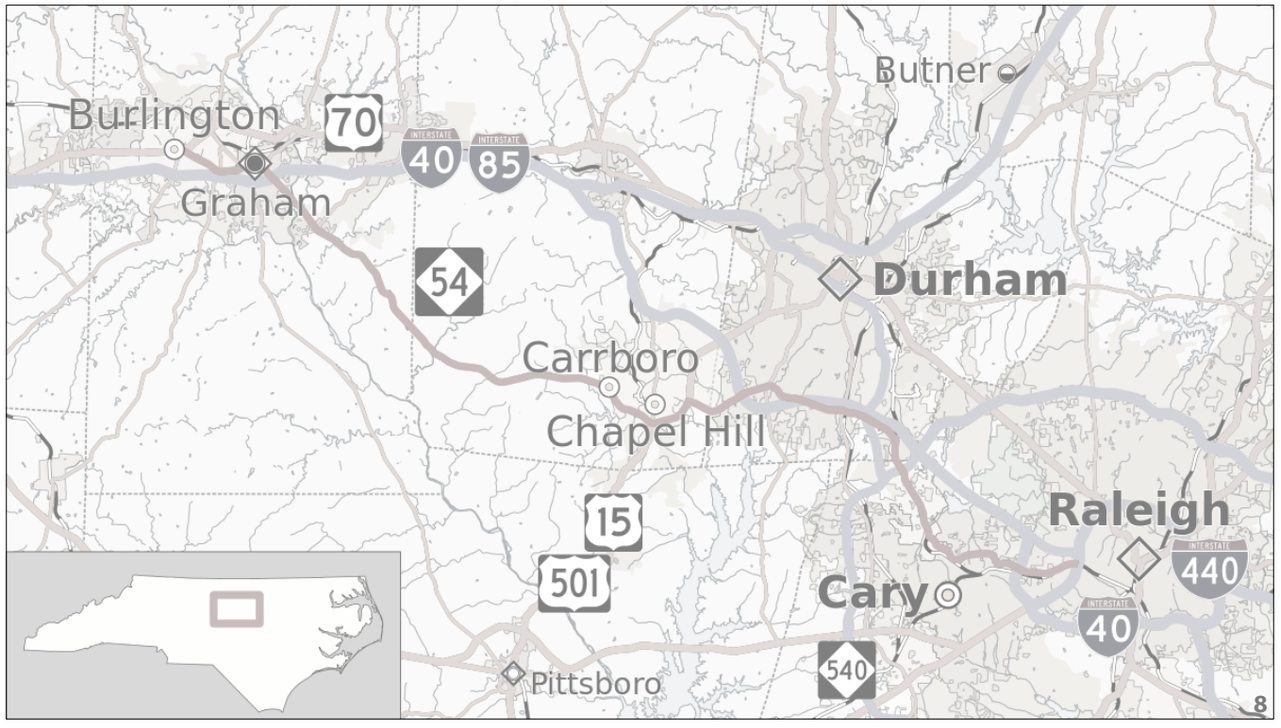
- My memory seems to work differently than most people's,
    - In various ways
- For example:
    - I'm mostly incapable of memorization
- It's not that I can't learn new information,
    - and not that can't access information I've previously learned.
- I can learn.
    - Just not through memorization.
    - In fact, I have no idea how to even begin to try to memorize anything.
- Instead, the only way for me to retain anything usefully is to **really** learn it.
    - To learn it as a system, as a network of connected information, a web, etc.
    - To **understand** it.
- The thing is, though:
    - to truly understand something takes time — often a lot of time.
- In a world that often equates memory with intelligence,
    - I have developed habits and strategies to compensate for my inability to memorize
    - For example, I rely heavily on reference documentation while writing programs →



- And not just while writing programs
- When I’m operating in any context that I don’t understand deeply
  - For example
    - When I’m in a new context,
    - a blank slate
    - I need documentation
      - *especially* reference documentation
      - to operate in that context
- For example, this is my first visit to North Carolina
  - For this situation, the reference docs I need
  - (reveal)
  - are maps.

Image: https://commons.wikimedia.org/wiki/File:NC_54_map.svg

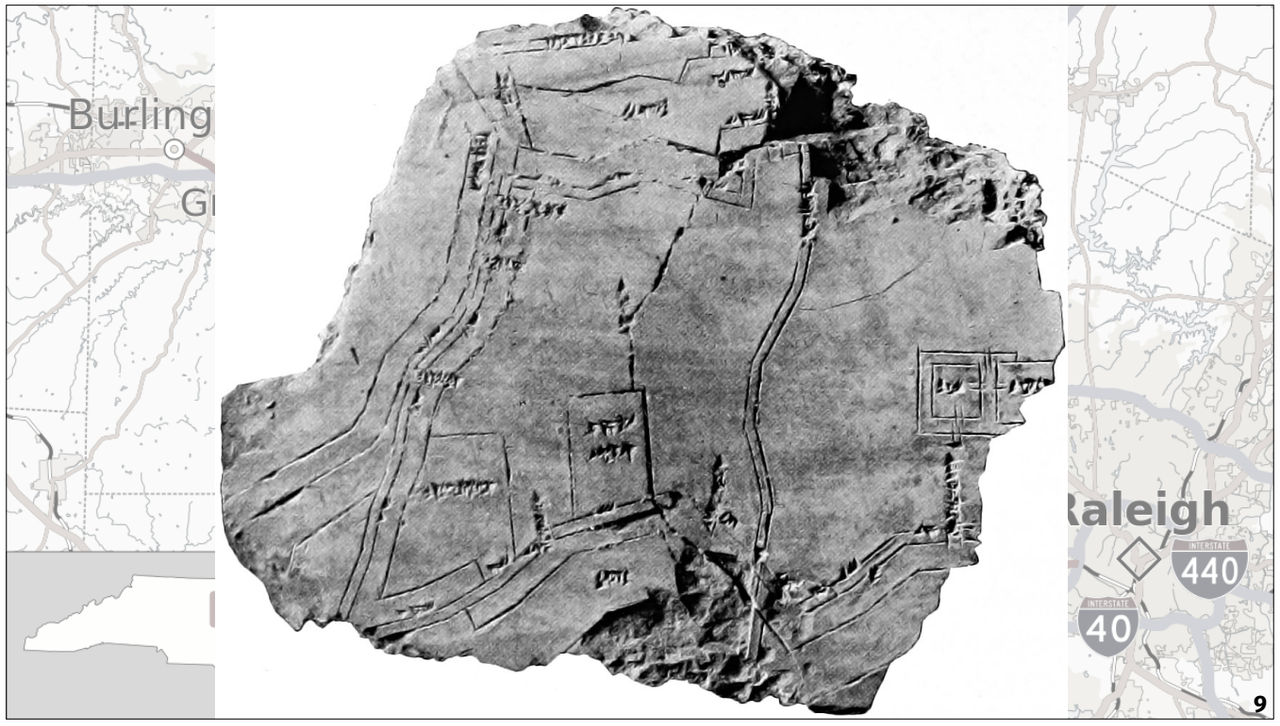

Burlington
70
Graham
INTERSTATE
40
INTERSTATE
85
Butner
54
Durham
Carrboro
Chapel Hill
15
501
Raleigh
INTERSTATE
440
Cary
INTERSTATE
40
540
Pittsboro
7



- Many of us use these visualizations to orient ourselves,
    - locate ourselves,
    - and navigate within unfamiliar environments;
    - and to *learn* those environments.
- But if you think about it,
    - we don't stop using maps once we've learned the geography of an area.
- Think about that.
    - Even after we've learned that geography,
        - after we "know" it,
        - we continue to refer to maps.
- Even those of us that have visual memories,
    - continue to refer to maps,
    - even if maybe they're doing it in their heads.
- Why?
    - Why do we continue to use maps even after we've learned the geography of an area?
    - Because the maps serve as a sort of *external memory*.
        - They help us organize and access the information we have in our heads.
        - So maybe we use them as both
            - an *external index* and an *external dataset.*
        - Maybe we load some modest portion of that index and dataset in our heads,
            - to varying degrees,
            - And then when we refer to a map,
            - when we *look* at a map,

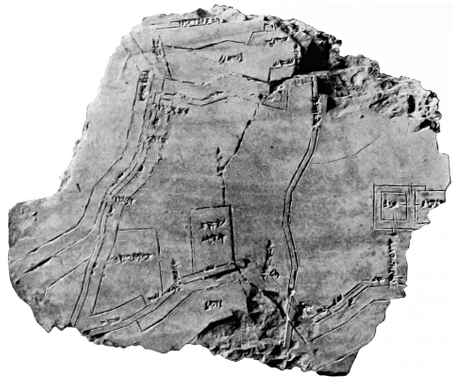
        - we're "joining" the index and dataset in our heads with those in the map.
- So, to recap:
    - Maps are extremely rich with information and structure.
    - Someone who's not familiar with an area;
        - can use a map to get oriented,
        - get familiar,
        - and then dig deeper, learn more.
    - Once a person is familiar with an area,
        - they'll continue using that map,
        - to fill in the details that they couldn't commit to memory,
        - to learn more and more over time.
- Maps are a rich, sophisticated, powerful visualization medium.
- But they didn't start out that way.

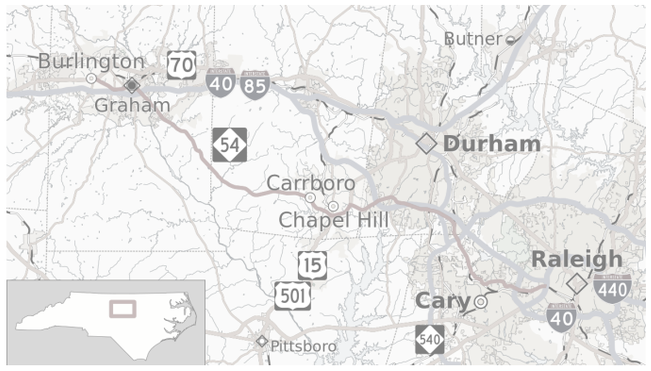
Image: https://commons.wikimedia.org/wiki/File:NC_54_map.svg



- This is a map of the Babylonian city of Nippur.
    - It’s about 3,500 years old.

Image: https://commons.wikimedia.org/wiki/File:Clay_tablet_containing_plan_of_Nippur_(Hilprecht_EBL_1903).jpg

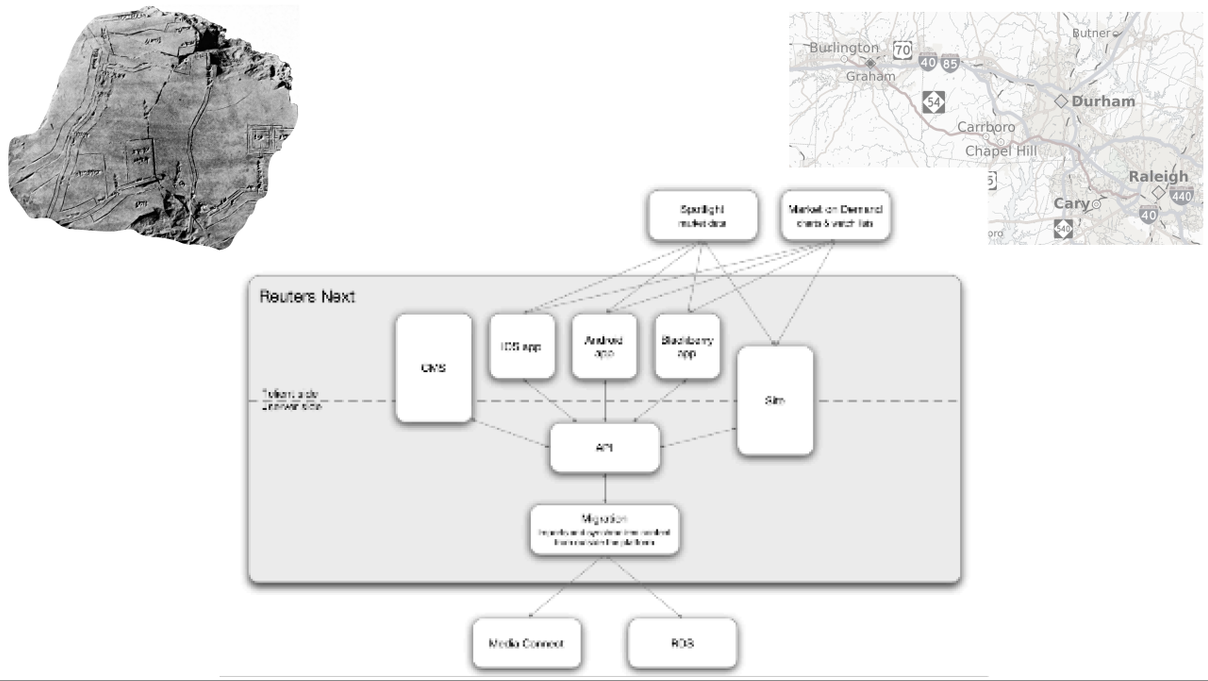




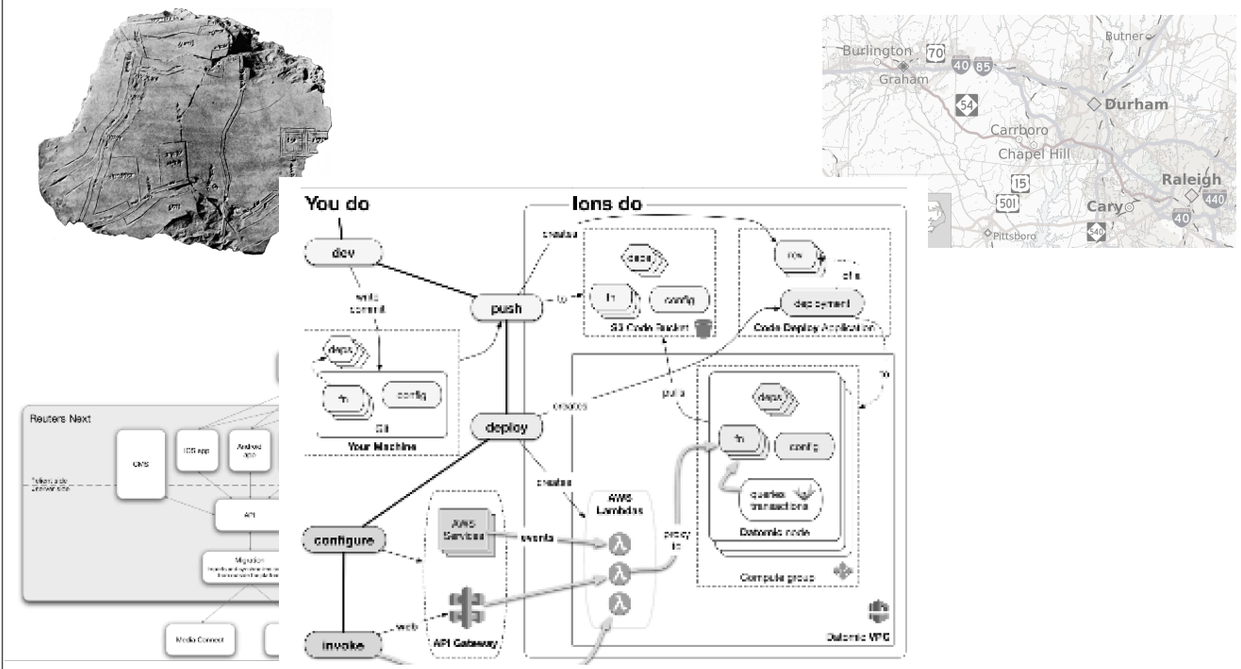
- That means it took us 3,500 years
    - to go from simplistic maps like the one on the left,
        - of Nippur,
    - to rich, sophisticated maps like the one on the right,
        - of central North Carolina.
- Now let's take a look at a similar visual medium;
    - one that has great potential,
    - but is very young,
        - just as geographical mapping was a young medium 3,500 years ago
    - software architecture diagramming →

Images:

- https://commons.wikimedia.org/wiki/File:Clay_tablet_containing_plan_of_Nippur_(Hilprecht_EBL_1903).jpg
- https://commons.wikimedia.org/wiki/File:NC_54_map.svg

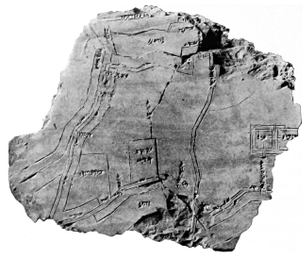


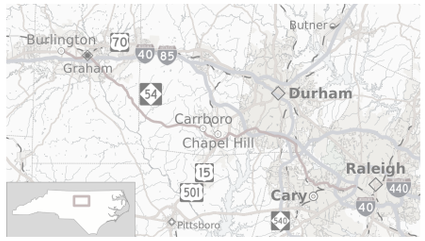
- This is an architecture diagram I created about 7 years ago.
- I think it’s OK, but fairly primitive.
- Closer to the Nippur map than the North Carolina map.

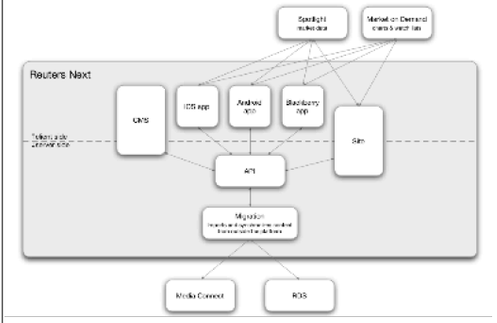


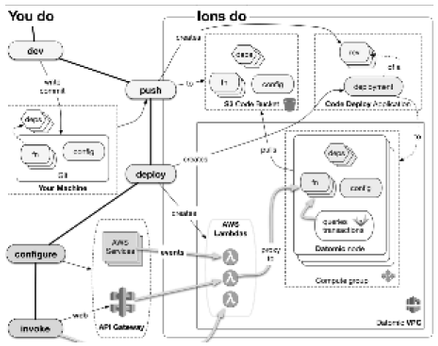
- Here's a much better architecture diagram,
    - created in the past year or two.
    - Some of you might recognize this.
- This is an excellent diagram,
    - A good way to illustrate the deep *potential* of the medium;
    - Which is to engender powerful, effective, efficient, **learning**;
        - both quickly, up front, **and** over time.
- I've found that, for me,
    - a diagram of a software system is akin to a map:
    - a way for me to get my bearings in that system, quickly,
    - so I can get right to work and be effective
    - well before I've attained a comprehensive understanding of that system.
- Because I can and will eventually commit structures like these to memory;
    - one way or another,
    - I do learn them,
        - and I can recall the elements and their relationships.
    - Not visually, but semantically.
- I think this is why I find diagrams so crucial when I'm in a new context;
    - they enable fast and effective reference,
    - and fast and effective learning.
- This may be especially true for me,
    - and maybe for others with aphantasia.
- But I suspect this also holds true for neurotypical folks.

- To summarize:
    - diagrams are a fast and effective medium for reference and learning,
    - they can help people be more effective faster,
    - in new contexts and over time.
- So, coming back to this excellent example.
- It *is* excellent.
- But if we use it to discern the development of the *medium*,
    - well, this medium is still not as far along as geographical maps are.
- I'd put it somewhere in the middle →









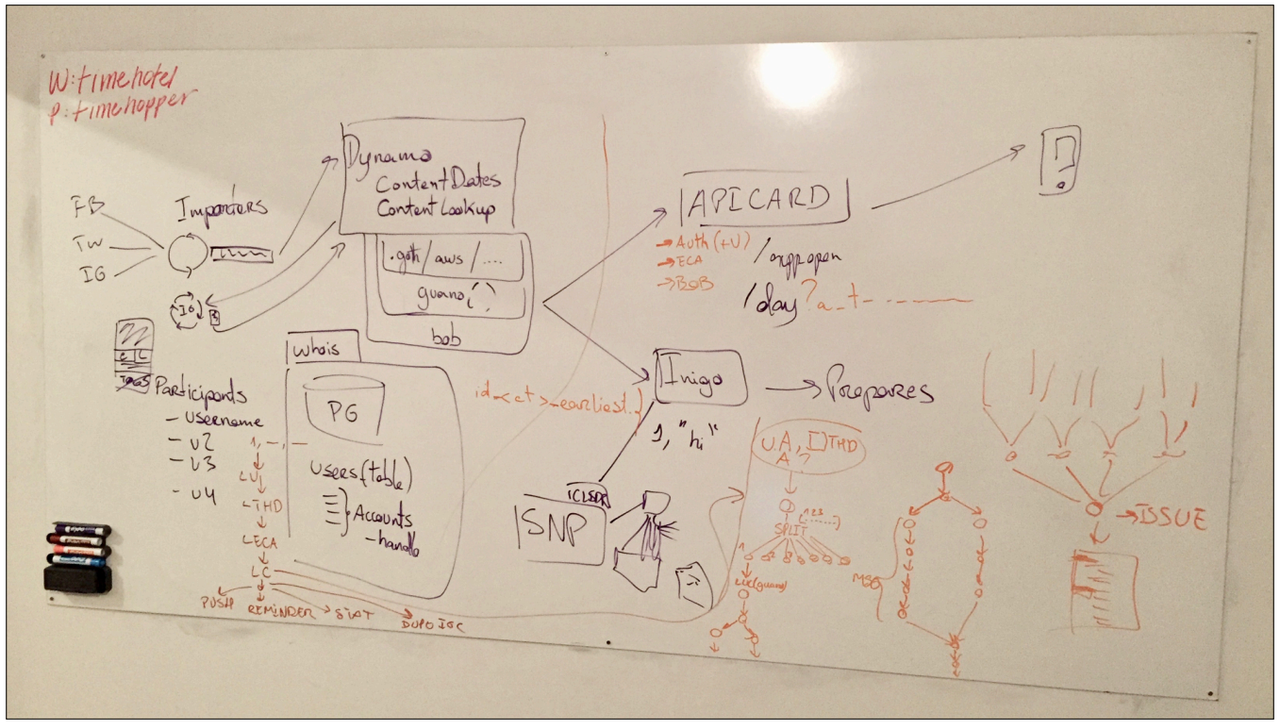
- So there's still some work to do,
    - some problems to solve,
    - to develop the medium to fulfill its potential.
    - To move us further to the right.
- As someone who creates a lot of diagrams,
    - my personal obsession is with *how* we create them.
    - The tools, techniques, processes, etc.
- And with these — well, there are some problems.

The What → 

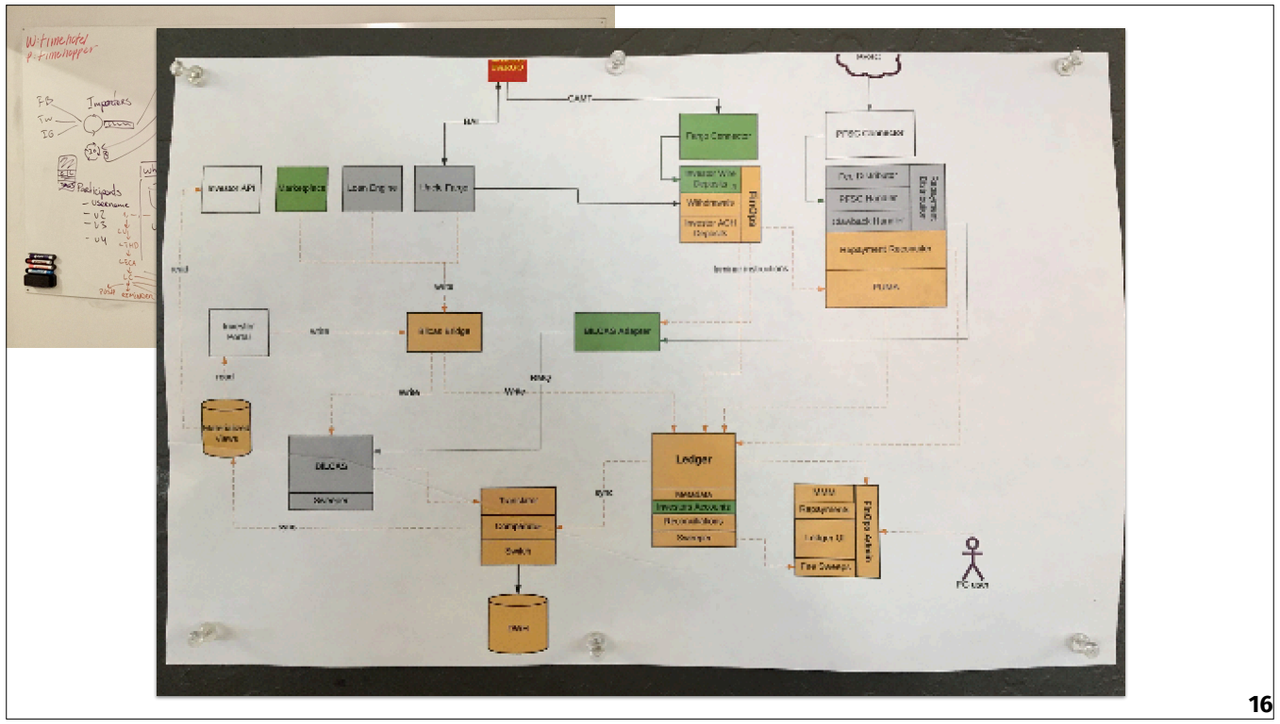
- **What is**
- Architecture diagrams — the “what” — are great.
- The tools, techniques, processes, and patterns we employ to create and maintain architecture diagrams —
    - the “how” —
    - (reveal)
    - not great.
- See, over the years,
    - I’ve encountered a whole bunch of problems,
    - and I’d like to share some of those experiences with you →

The What → 🤩

The How → 🤦

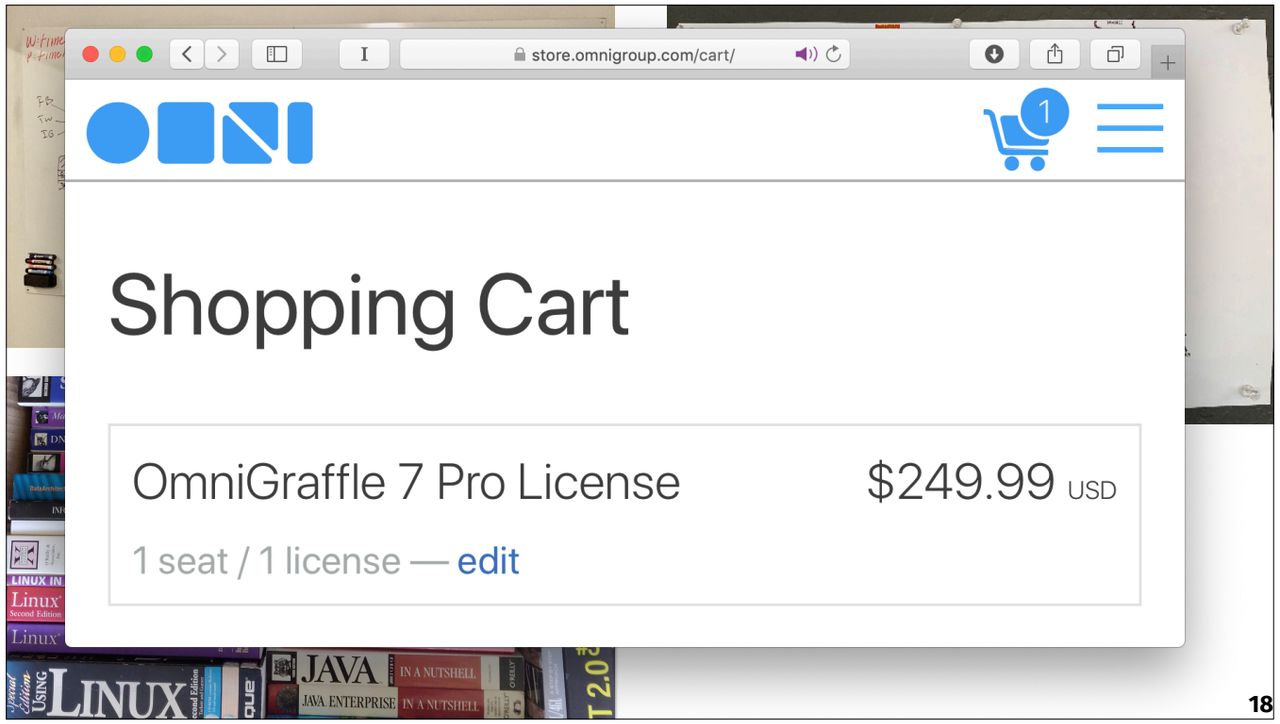


- There were the many times I joined a new team,
    - and asked someone if there were any architecture diagrams,
    - and the person I asked lead me over to a whiteboard
    - and started scribbling furiously,
    - with a stream-of-consciousness narration.
    - The results were generally less than coherent, and certainly not correct.

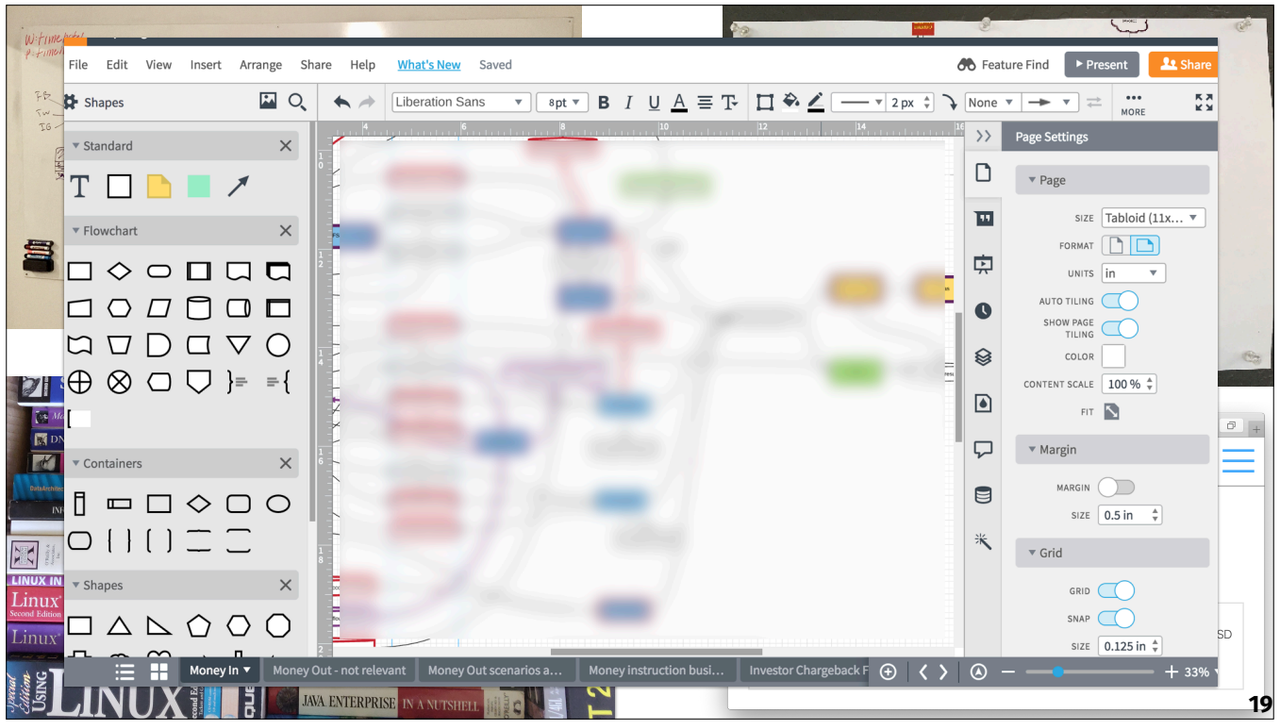


- There was the time I joined a new team,
    - and started looking for diagrams.
- And I found some, printed out and pinned up to the wall.
- They were pretty good, but needed updating.
- When I asked who made them and when,
    - I just got shrugs.
    - And “I’m not sure, but I think they left last year.”

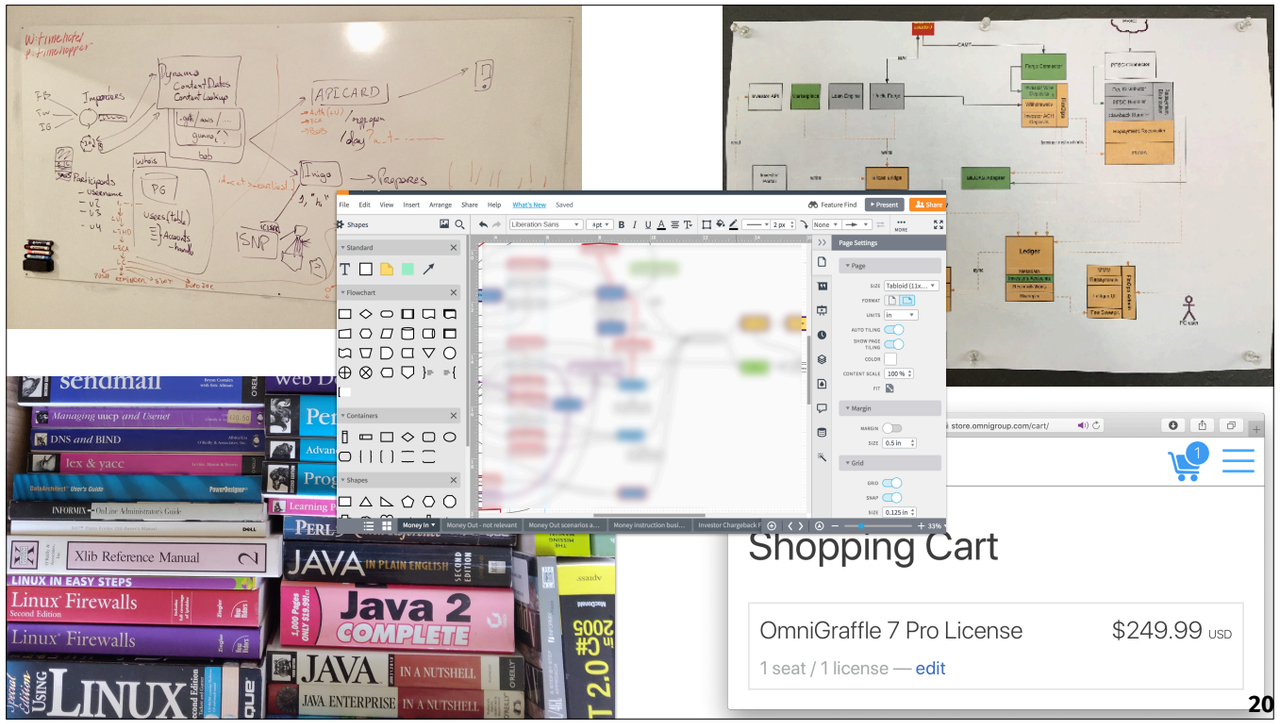
- There was the time I asked someone to work with me to update a diagram,
    - and they declined because,
    - and this is a quote,
    - “documentation is always out of date”



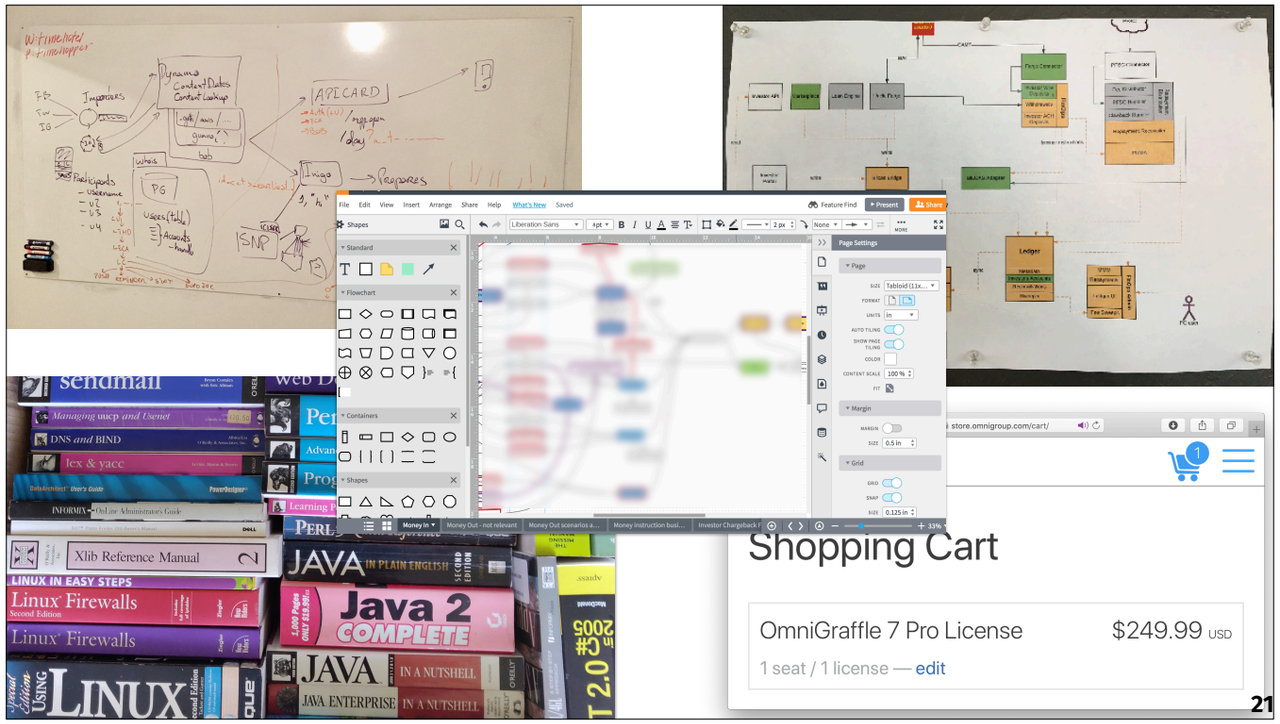
- There was the time I wanted to update a diagram that someone else had created,
    - some mystery person.
- And, against all odds, I actually found the source file for the diagram!
- But then I was blocked,
    - for lack of a proprietary and expensive software license.



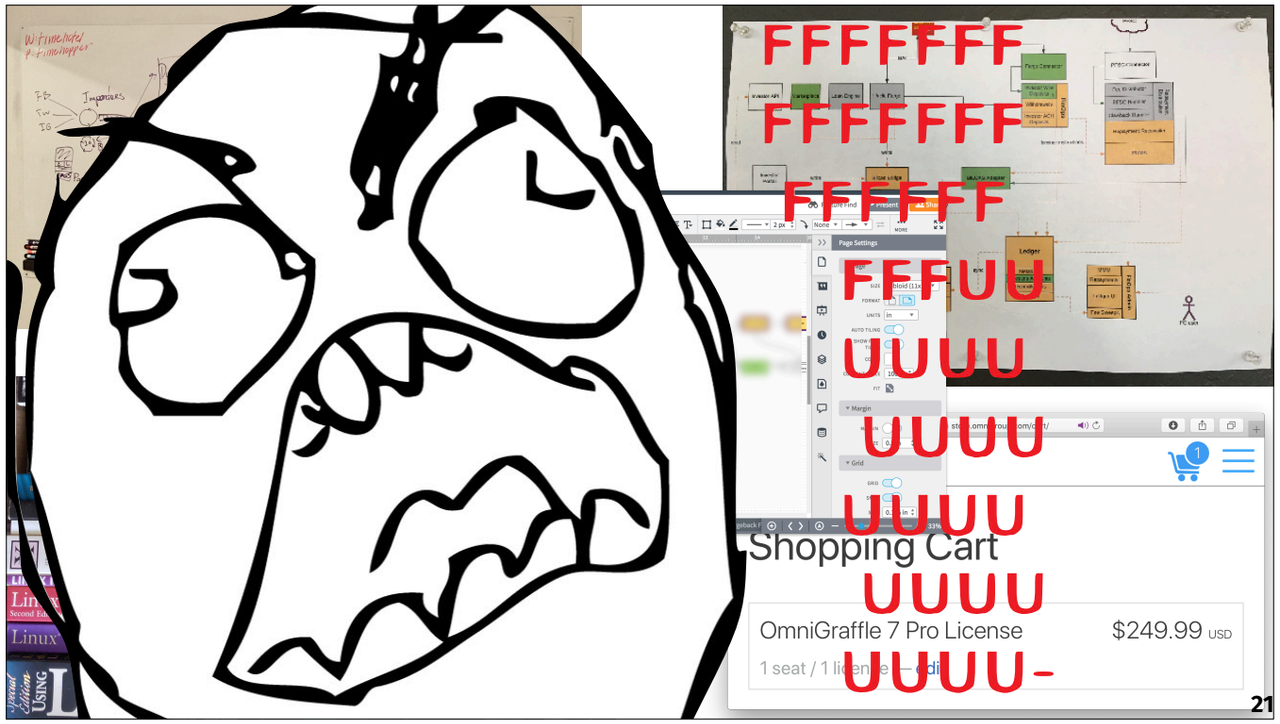
- There were the many, many hours I wasted fiddling with diagrams
    - to get their elements laid out evenly and consistently.
- I hated every moment of those hours,
    - because what the hell,
    - computers were supposed to eliminate this kind of drudgery years ago?



- To summarize:
- Many diagrams are ephemeral and ad-hoc,
    - when they should be artifacts that are maintained and improved, thoughtfully and carefully, over time.
- Many diagrams are created as just a bunch of boxes and lines,
    - without any specific *method* or *conceptual guidelines* for,
    - for example, what should be included and what shouldn't.
- Most diagrams are created and stored in individual people's devices or accounts,
    - when they should be in a single shared corpus,
    - where anyone in the org could discover which diagrams exist and which don't,
    - and could find the source of each diagram,
    - and the change history for each diagram.
- Most diagrams use file formats that are proprietary,
    - and can be edited with only the tool that was used to create them,
    - when they should use file formats that are open and interoperable.
- Most diagrams are created in GUIs that are basically illustration tools,
    - which often require authors to spend a lot of time on layout, positioning, and styling,
    - when they should use tools that are specifically for diagramming and therefore handle a lot of that stuff automatically.

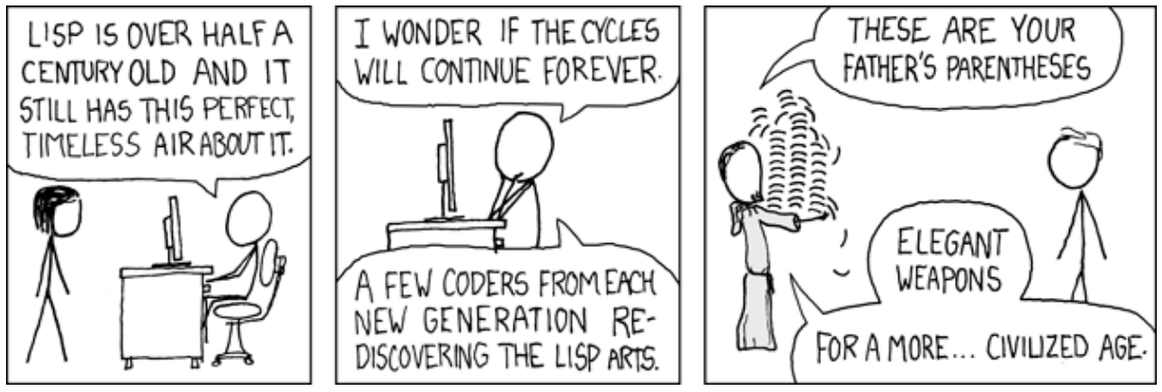


- When I think about this set of problems,
    - this is how I feel **(reveal)**
    - I've done my fair share of ranting and raving
- Turns out, there's a solution;
    - A technology we're quite familiar with →

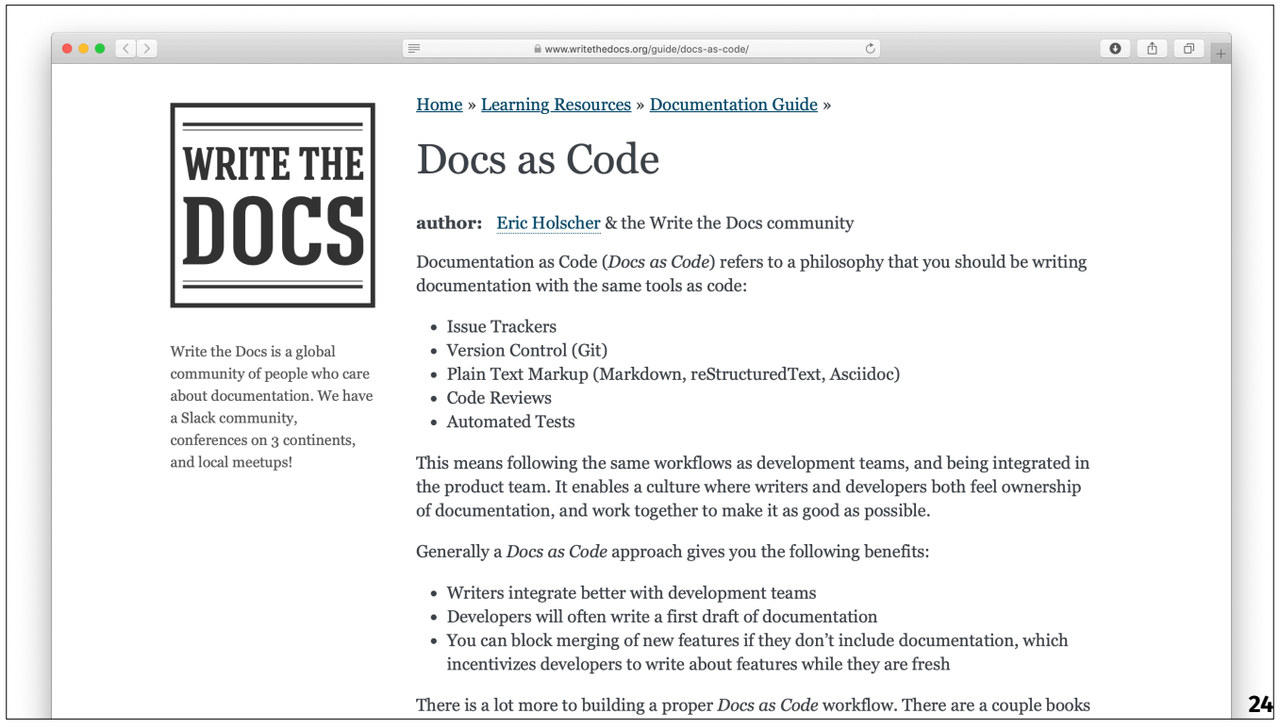

FFFFFFF
FFFFFFF
FFFFFF
FFFUU
UUUU
UUUU
UUUU
UUUU
UUUU-
Shopping Cart
OmniGraffle 7 Pro License
$249.99 USD
1 seat / 1 license — edi
Linux

:data-in-text-files

- That’s right, it’s **data in text files!**



- As as many of us might already know, in Clojure, and all Lisps, code is data.
- We represent our code in text files using our language's native data structure literals.
- This makes it trivial to generate code and trivial to parse code — and once it's parsed you can manipulate it, analyze it, project it, transform it, translate it, graph it, etc.
- You can do all sorts of fun things!
- Wouldn't it be cool
    - if we could express our architecture diagrams as data in text files
    - so that we could then do some of these fun things with our diagrams?
    - I think that would be super cool!
- Using *Data in text files* would also enable us to use the same workflow for authoring for our documentation that we use for authoring our code.
    - This is a Very Good Idea
    - It's called *Docs as Code* →

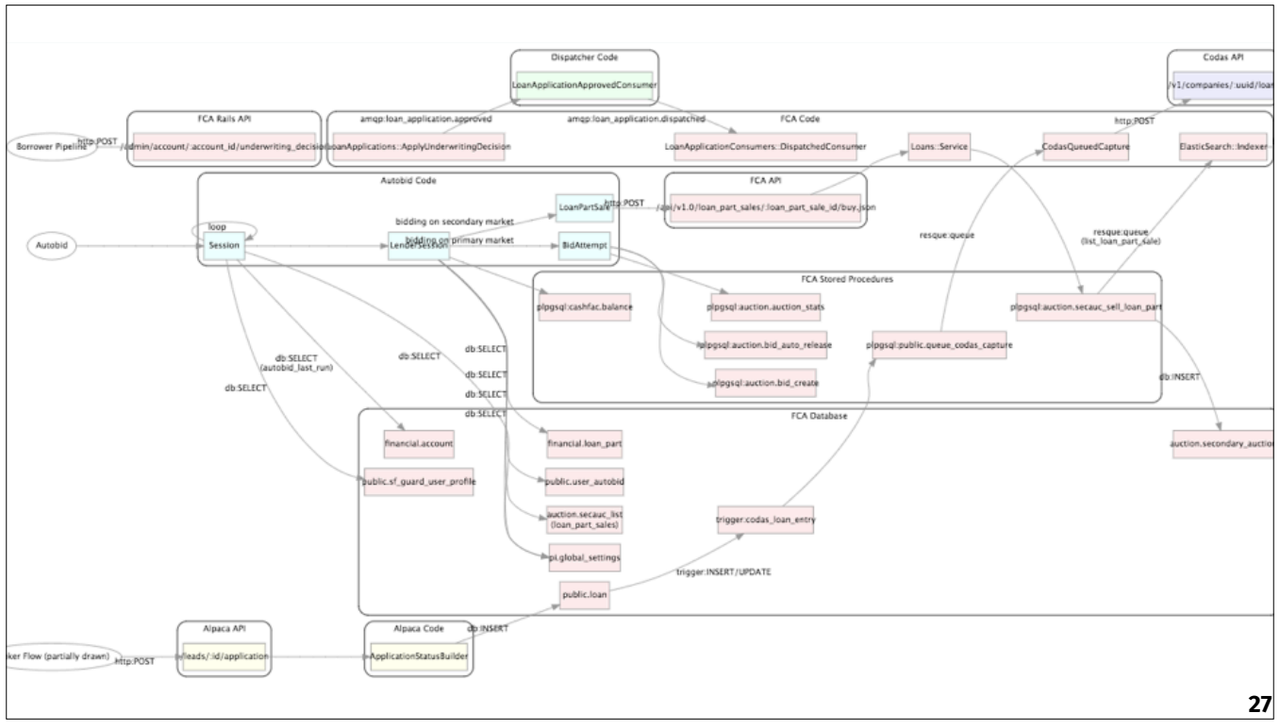


- Something about Write the Docs
- Using *Data in text files* would also enable us to leverage all the rich, robust tools and workflows that have been developed over decades to facilitate and support collaboration via text files:
    - version control systems (VCSs) such as Git
    - and collaboration systems built on those VCSs
        - such as GitHub, GitLab, BitBucket, Gerritt, Phabricator, etc.
- Those systems are fantastic for collaboration, but they're optimized for collaborating on *text files*
    - they can do very little with the proprietary binary file formats used by many diagramming tools.

- As great a technology as *data in text files* is,
    - it's not quite enough to solve our problems.
- (reveal)
- We also need tools that will render that data into images.

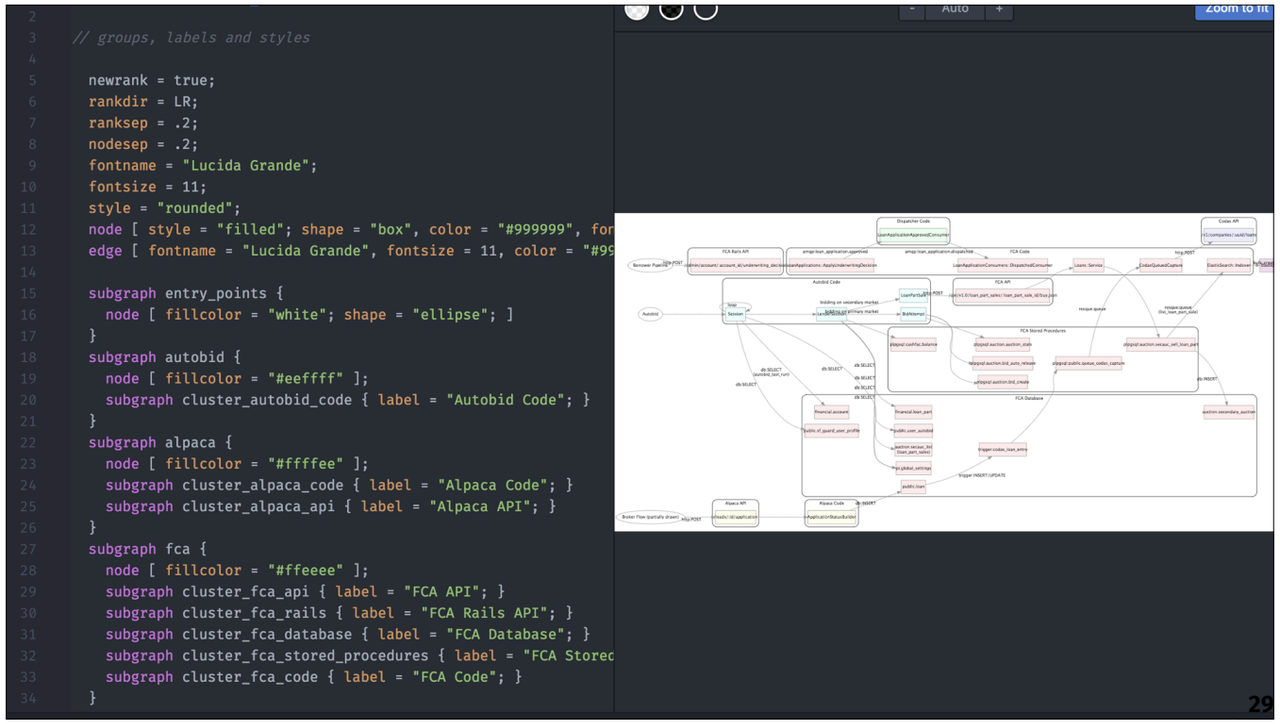
```
(+ :data-in-text-files
   :rendering-tools)
```

- **What is**
- Now, you might be thinking:
    - Don’t we already have tools that can render diagrams from data in text files?
    - Tools like Graphviz, PlantUML, Mermaid, etc?
- Well, yes, those tools do exist.
    - I’ve tried many over the years.
    - Unfortunately, none of them have ever felt like a good fit for software architecture.
- There are a few reasons for this.

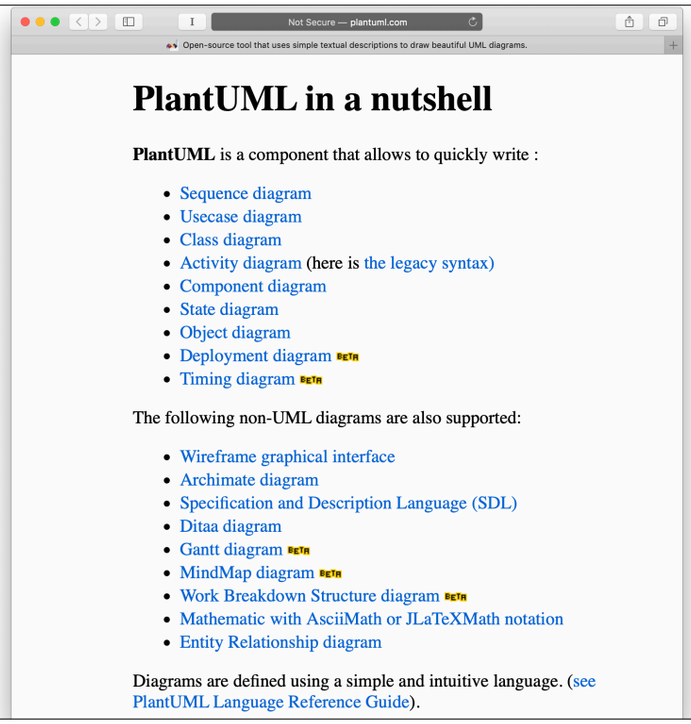


- For example, Graphviz is a very well known tool.
    - And it’s an excellent tool.
- Unfortunately, Graphviz is all about graphs!
    - *Mathematical* graphs.
- So it makes composition and nesting quite awkward.

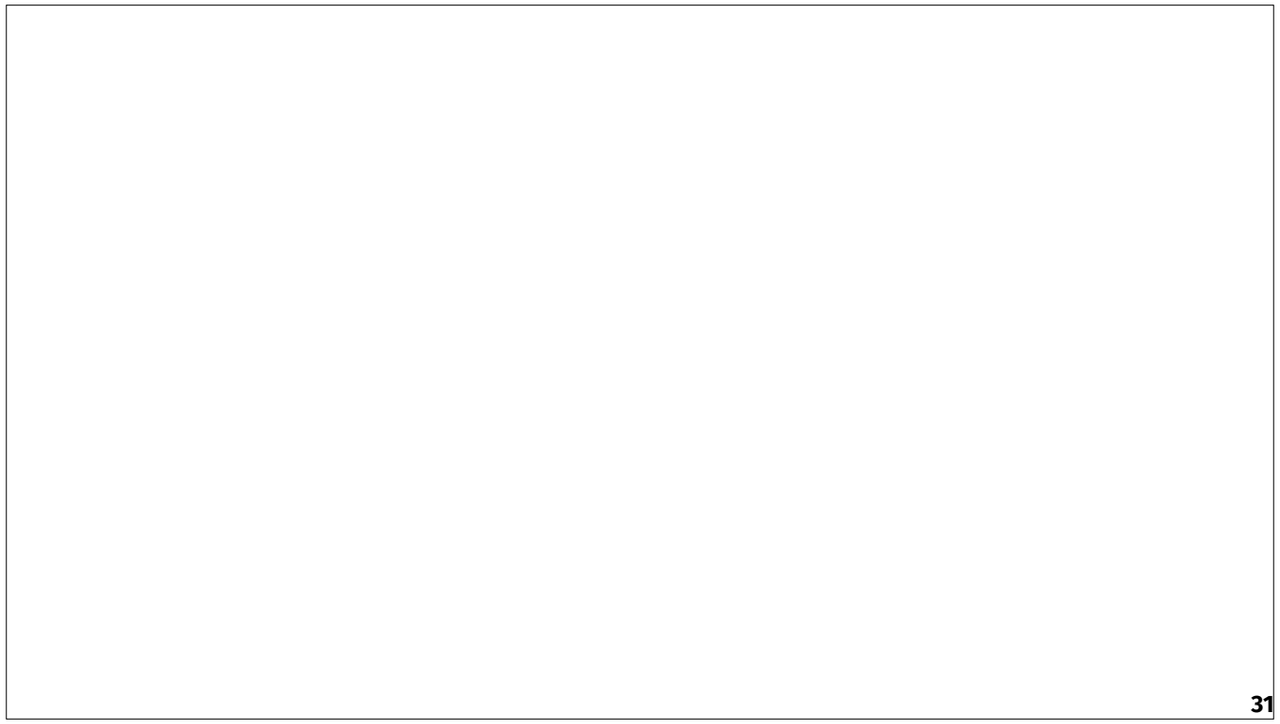
- Which is unfortunate,
    - because nested elements are an extremely useful notation to include in an architecture diagram!



- All or most of these tools support only algorithmic layouts when rendering diagrams
    - meaning the tool decides where to position all the elements, how to route the relationship lines, etc.
- This is great when one has a large dataset on hand
    - and one is creating a visualization
    - in order to explore that data, to seek new insights.
- But it's not at all great when one is *authoring* the data specifically in order to create a visualization designed to *convey* a coherent *story* about about those elements.
- It turns out that when one *is* trying to convey a coherent story about a set of elements,
    - it's extremely useful to be able to specify the relative positioning of those elements.
- Whenever I tried Graphviz, Cypher, things like that:
    - I'd find myself wrestling with its algorithms
    - just to get things laid out clearly and usefully.



- Finally, there's the problem of how general-purpose these tools are.
- Mermaid alone supports:
    - flowcharts, sequence diagrams, class diagrams, state diagrams, Gannt charts, and Pie charts.
    - PlantUML supports at least 18 kinds of diagrams.
- The generality of these tools means that:
    - they have no specific notations or conventions specifically for software architecture diagrams.
- In other words, they lack a *conceptual model* for *how* to author architecture diagrams.
- In my experience, the lack of a clear, robust, well-defined conceptual model is a major impediment to people getting involved with creating and maintaining architecture diagrams
    - it can be daunting, and a lot of work to have to figure out on one's own how to craft a good diagram.
    - Then everyone else has to learn how to "read" your diagram!
- (To be clear, those tools are great for lots and lots of use cases. I just don't think they're great for this very specific use case: software architecture diagrams.)

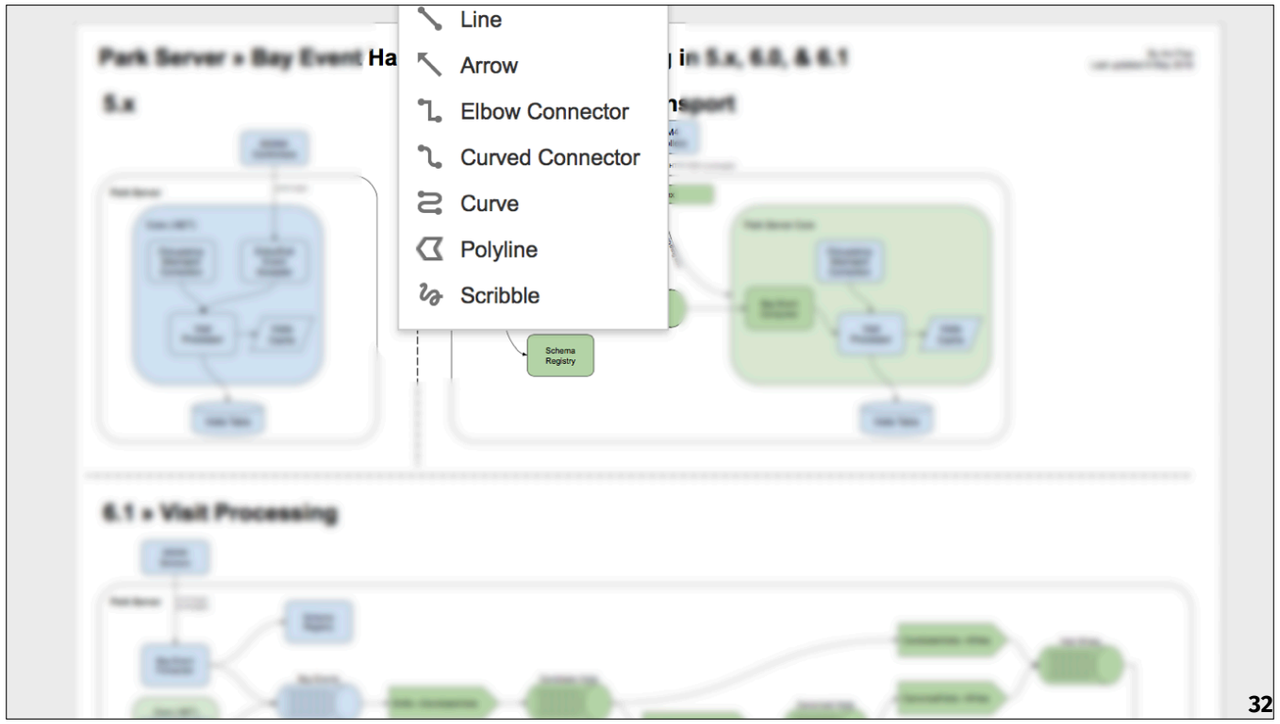


Summarize the problems with existing tools

- Just to summarize the problems with the existing tools
- I've made this table (reveal)
- *Because there's just so much information here.*
- These are our needs,
    - And here's how the existing tools do (reveal)

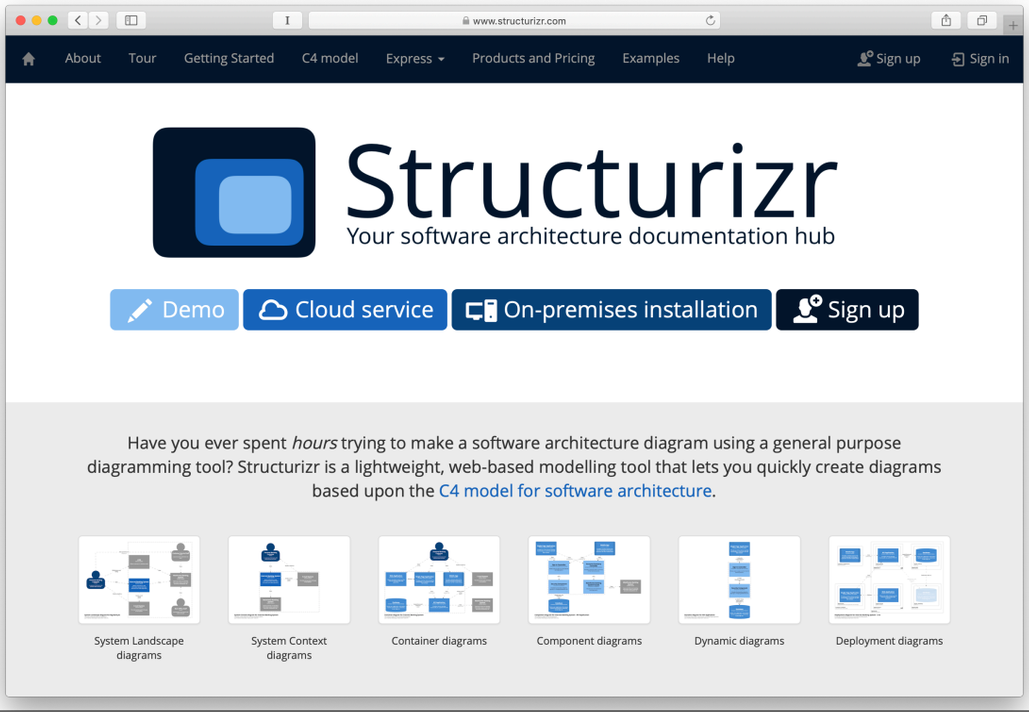
	Composition, nesting, encapsulation	User-specified layout	Specific support for software architecture

	Composition, nesting, encapsulation	User-specified layout	Specific support for software architecture
Existing tools			



- About two years ago I started a new job, at Funding Circle
  - I decided I should create a set of architecture diagrams
  - to get my bearings in a new context,
    - and help others.
- I once again tried a few tools that could render diagrams from data in text files.
- But I was once again stymied
  - and once again fell back to using GUIs with proprietary, closed file formats
  - I was bummed,
    - and I kept randomly searching the web for diagramming tools…
- And then I found one!

- **What could be**
- And then I stumbled across Structurizr.

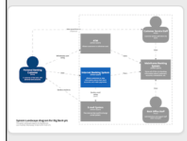


- This is structurizr.com.
- I'd like to read you the description right there in the center of the page:
    - (zoom)
    - "Have you ever spent *hours* trying to make a software architecture diagram using a general purpose diagramming tool? Structurizr is a lightweight, web-based modeling tool that lets you quickly create diagrams based upon the *C4 model for software architecture*."
- Oooh, the C4 model… what's that?

Your software architecture documentation hub

Demo | Cloud service | On-premises installation | Sign up

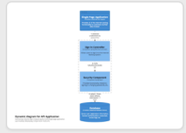
Have you ever spent *hours* trying to make a software architecture diagram using a general purpose diagramming tool? Structurizr is a lightweight, web-based modelling tool that lets you quickly create diagrams based upon the C4 model for software architecture.


System Landscape diagrams

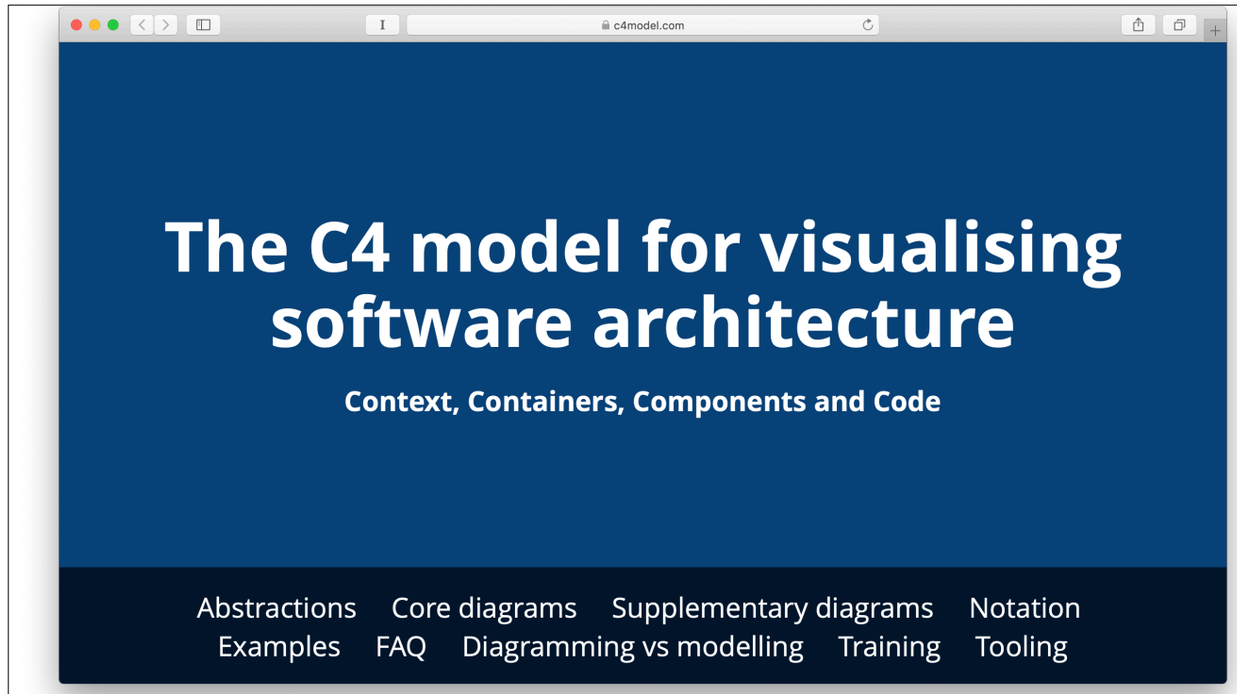

System Context diagrams


Container diagrams


Component diagrams


Dynamic diagrams


Deployment diagrams

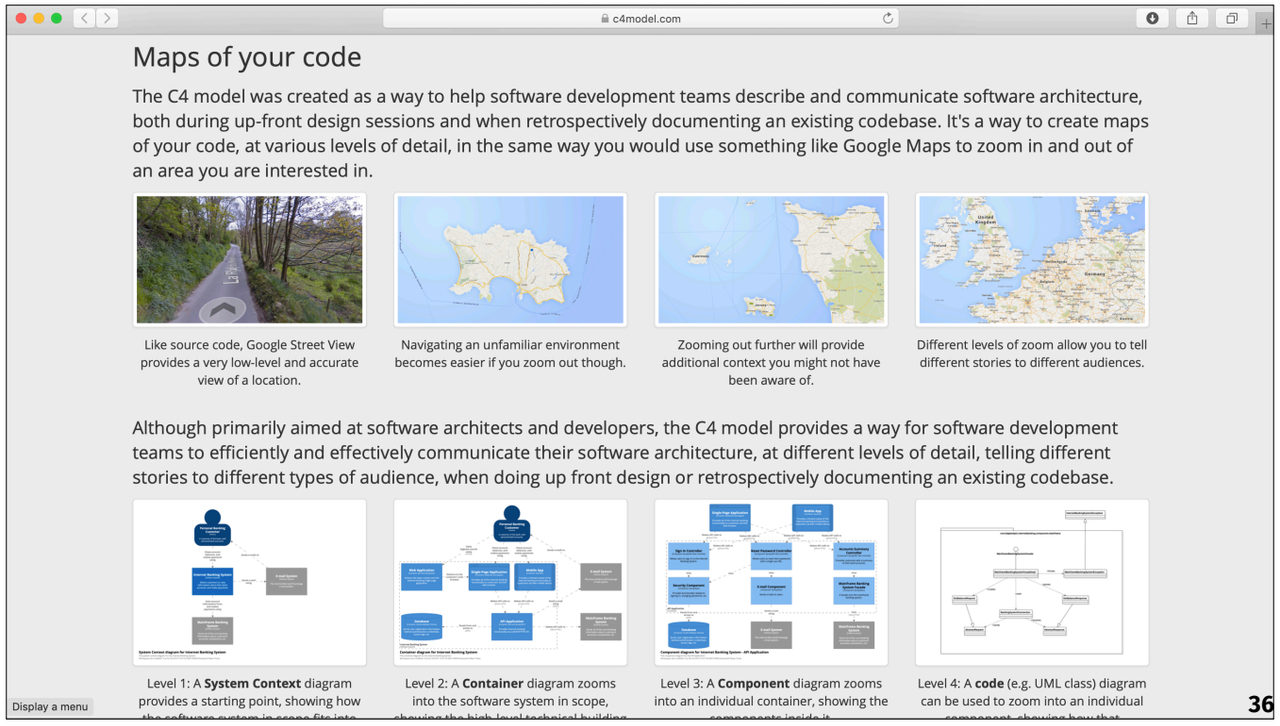


35

- Turns out, I wasn't the only one who'd been frustrated with this situation
- But Simon Brown had actually *done something* about it.
- He created Structurizr,
    - And he also created the C4 model for visualizing software architecture.
- This is that conceptual model I mentioned earlier!
- The name C4 comes from the 4 primary types of diagrams
    - that the framework defines —
    - which all start with the letter C.
- (zoom)
- (read list)
- Brown has done some wonderful work here,
    - in defining and documenting this model.
- If you scroll down on this page,
    - you'll pretty quickly come across this section: →

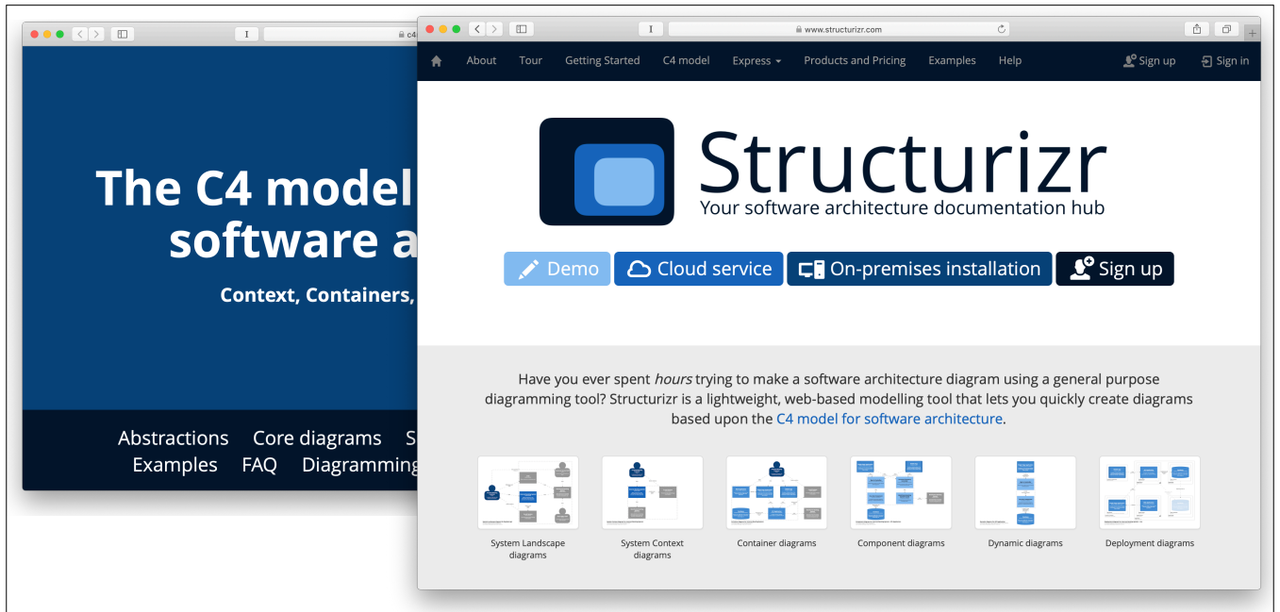
C4 model for visuali
oftware architectur

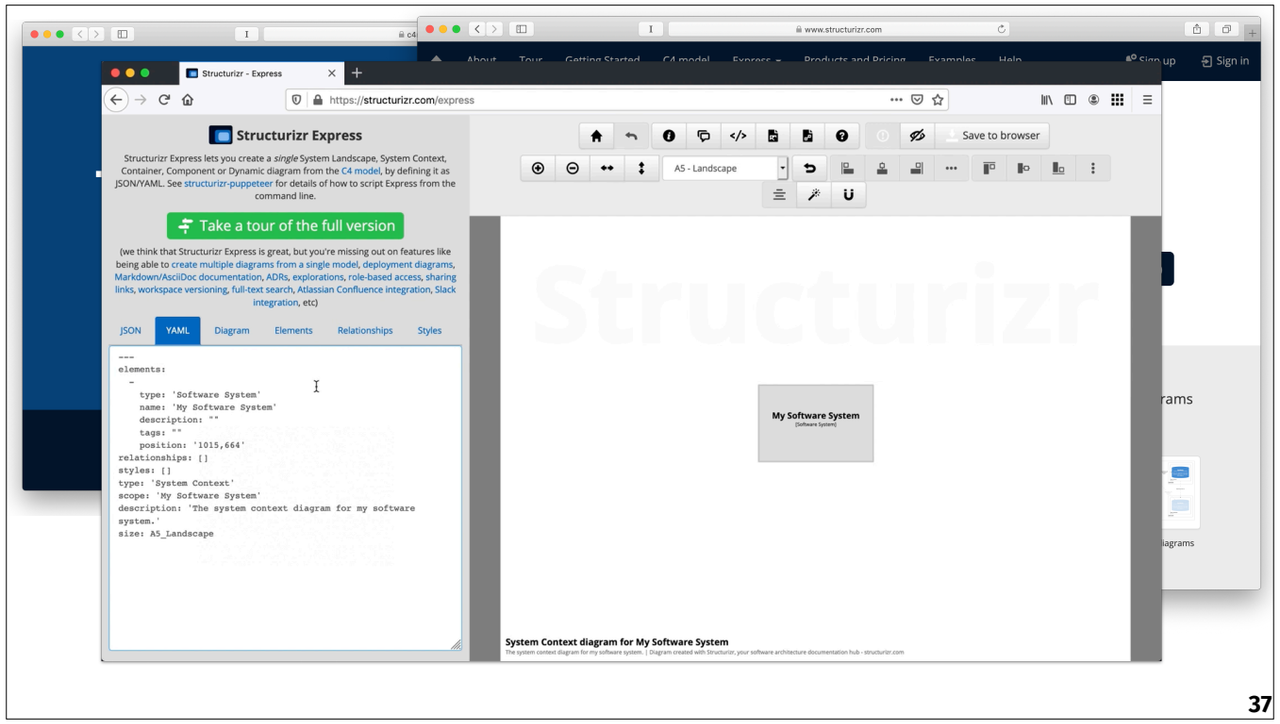
Context, Containers, Components and Code

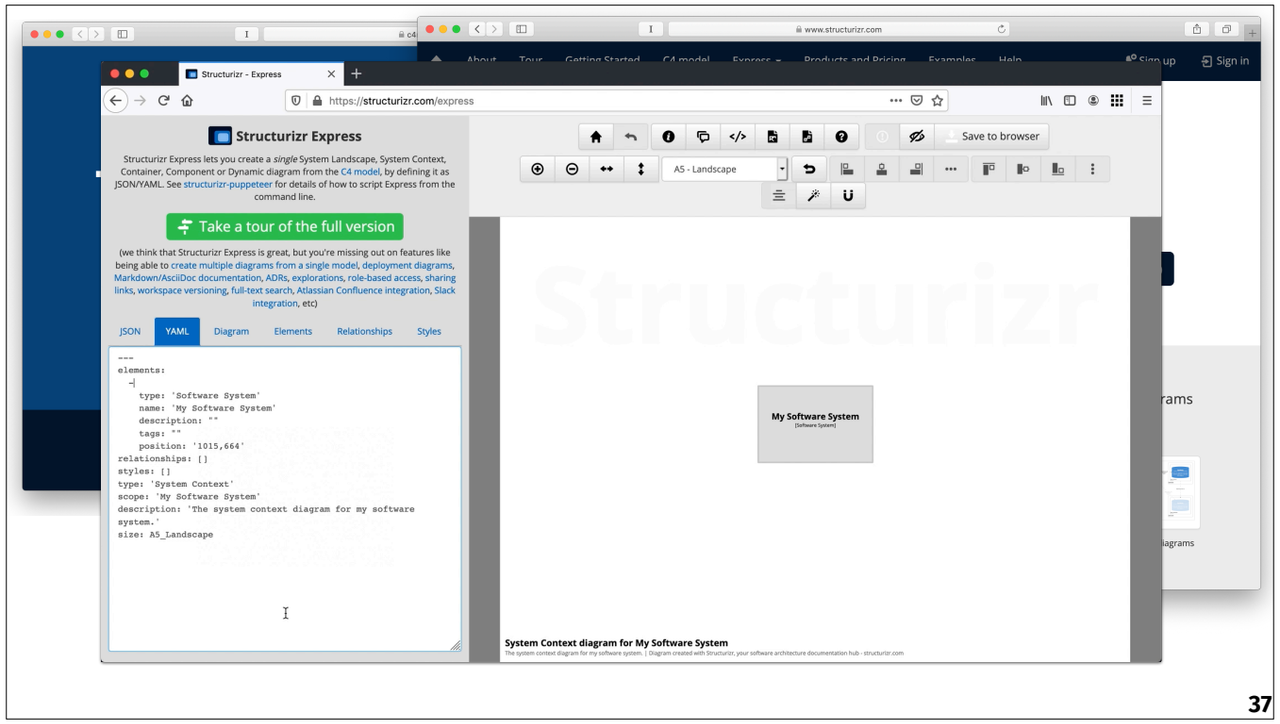


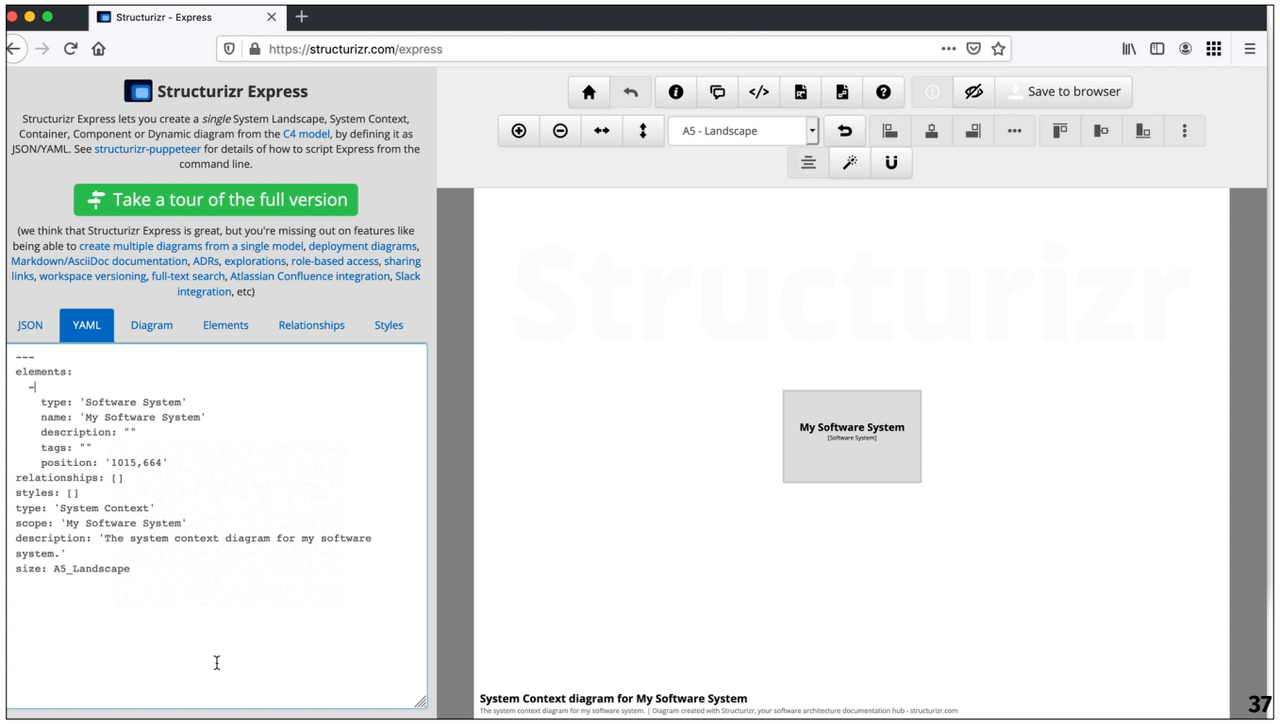
- Brown also saw a correspondence between maps and diagrams.
- In particular,
    - he’s organized this conceptual model around different zoom levels.
- Just as maps at different zoom levels are useful at different times,
    - for different purposes;
    - sometimes you need to get a high-level overview of an area,
    - sometimes you need a block-by-block street map,
    - so too with diagrams.

- So Simon Brown has done some amazing work.
- He created Structurizr
    - a Web-based system for authoring, hosting, browsing, and publishing C4 diagrams
- And he created Structurizr Express.
- SE is a single-page Web app that:
    - Defines a simple data model for C4 diagrams
    - enables authoring C4 diagrams as YAML or JSON data as text
    - enables authoring C4 diagrams graphically
    - and it keeps them in sync!
        - editing the data updates the graphic, and vice-versa!
- Here’s a quick demo of using SE.


The C4 model
software a
Context, Containers,
Abstractions
Core diagrams
Examples
FAQ
Diagramming
www.structurizr.com
About
Tour
Getting Started
C4 model
Express
Products and Pricing
Examples
Help
Sign up
Sign in
Structurizr
Your software architecture documentation hub
Demo
Cloud service
On-premises installation
Sign up
Have you ever spent *hours* trying to make a software architecture diagram using a general purpose diagramming tool? Structurizr is a lightweight, web-based modelling tool that lets you quickly create diagrams based upon the C4 model for software architecture.
System Landscape diagrams
System Context diagrams
Container diagrams
Component diagrams
Dynamic diagrams
Deployment diagrams

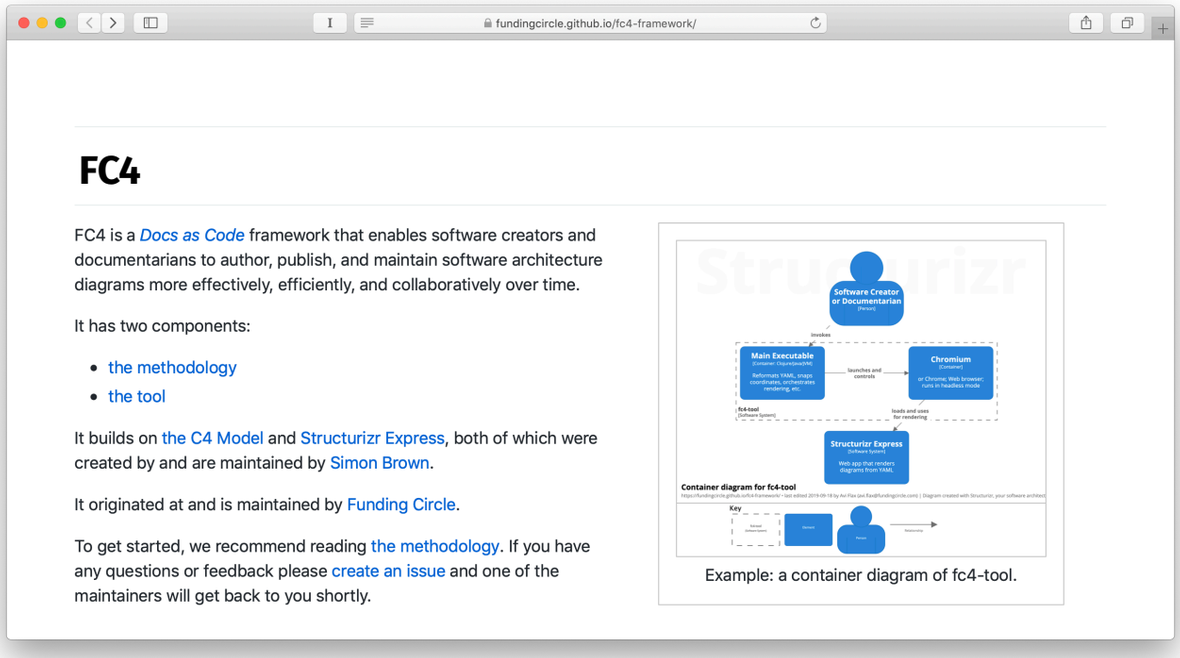

Structurizr - Express
https://structurizr.com/express
Structurizr Express
Structurizr Express lets you create a single System Landscape, System Context, Container, Component or Dynamic diagram from the C4 model, by defining it as JSON/YAML. See structurizr-puppeteer for details of how to script Express from the command line.
Take a tour of the full version
(we think that Structurizr Express is great, but you're missing out on features like being able to create multiple diagrams from a single model, deployment diagrams, Markdown/AsciiDoc documentation, ADRs, explorations, role-based access, sharing links, workspace versioning, full-text search, Atlassian Confluence integration, Slack integration, etc)
JSON
YAML
Diagram
Elements
Relationships
Styles
---
elements:
- type: 'Software System'
name: 'My Software System'
description: ""
tags: ""
position: '1015,664'
relationships: []
styles: []
type: 'System Context'
scope: 'My Software System'
description: 'The system context diagram for my software system.'
size: A5_Landscape
Save to browser
A5 - Landscape
Structurizr
My Software System
[Software System]
System Context diagram for My Software System
www.structurizr.com
About
Tour
Getting Started
C4 model
Products and Pricing
Examples
Help
Sign up
Sign in


Structurizr - Express
https://structurizr.com/express
Structurizr Express
Structurizr Express lets you create a single System Landscape, System Context, Container, Component or Dynamic diagram from the C4 model, by defining it as JSON/YAML. See structurizr-puppeteer for details of how to script Express from the command line.
Take a tour of the full version
(we think that Structurizr Express is great, but you're missing out on features like being able to create multiple diagrams from a single model, deployment diagrams, Markdown/AsciiDoc documentation, ADRs, explorations, role-based access, sharing links, workspace versioning, full-text search, Atlassian Confluence integration, Slack integration, etc)
JSON
YAML
Diagram
Elements
Relationships
Styles
---
elements:
- type: 'Software System'
name: 'My Software System'
description: ""
tags: ""
position: '1015,664'
relationships: []
styles: []
type: 'System Context'
scope: 'My Software System'
description: 'The system context diagram for my software system.'
size: A5_Landscape
Save to browser
A5 - Landscape
Structurizr
My Software System
[Software System]
System Context diagram for My Software System
The system context diagram for my software system. | Diagram created with Structurizr, your software architecture documentation hub - structurizr.com
www.structurizr.com
About
Tours
Getting Started
C4 model
Express
Products and Pricing
Examples
Help
Sign up
Sign in

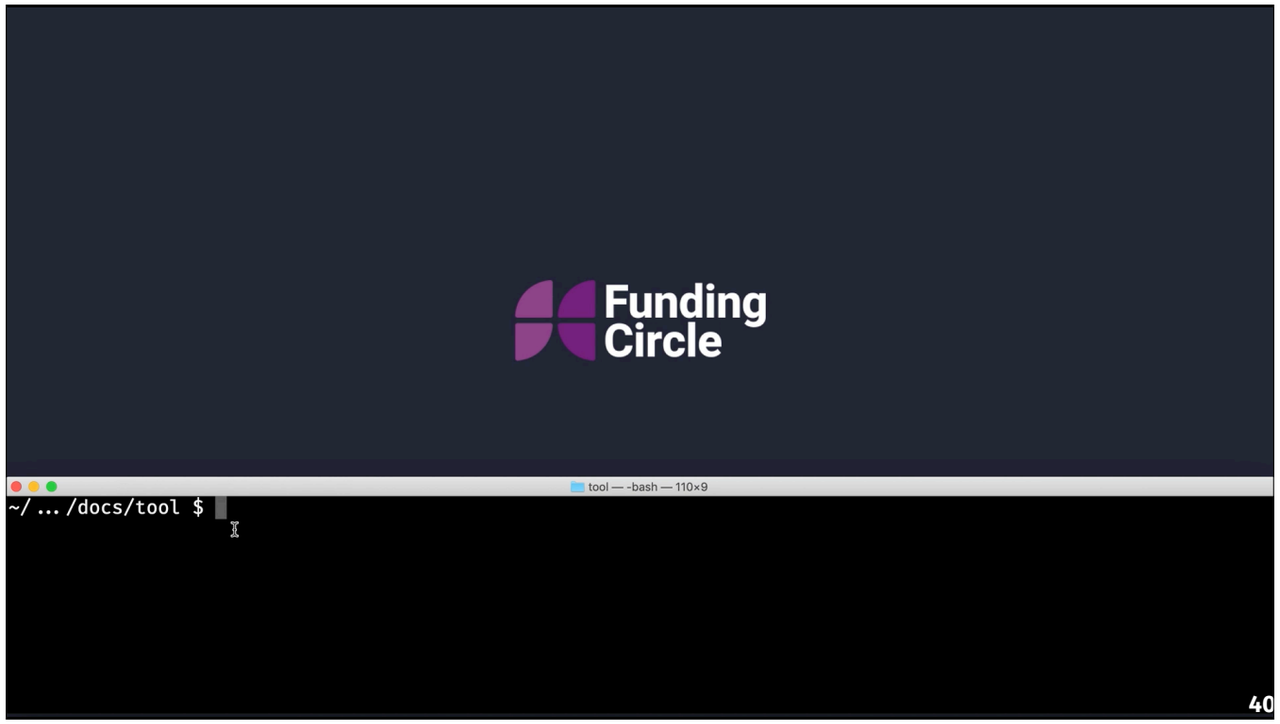

Structurizr - Express
https://structurizr.com/express
Structurizr Express
Structurizr Express lets you create a single System Landscape, System Context, Container, Component or Dynamic diagram from the C4 model, by defining it as JSON/YAML. See structurizr-puppeteer for details of how to script Express from the command line.
Take a tour of the full version
(we think that Structurizr Express is great, but you're missing out on features like being able to create multiple diagrams from a single model, deployment diagrams, Markdown/AsciiDoc documentation, ADRs, explorations, role-based access, sharing links, workspace versioning, full-text search, Atlassian Confluence integration, Slack integration, etc)
JSON
YAML
Diagram
Elements
Relationships
Styles
---
elements:
-
type: 'Software System'
name: 'My Software System'
description: ""
tags: ""
position: '1015,664'
relationships: []
styles: []
type: 'System Context'
scope: 'My Software System'
description: 'The system context diagram for my software system.'
size: A5_Landscape
Save to browser
A5 - Landscape
Structurizr
My Software System
[Software System]
System Context diagram for My Software System
The system context diagram for my software system. | Diagram created with Structurizr, your software architecture documentation hub - structurizr.com
37

```yaml
type: System Context
scope: FC4
description: Helps create C4 diagrams.

elements:
  - type: Person
    name: Software Documentarians
    position: '725,50'
  - type: Software System
    name: Structurizr Express
    description: Web app; renders diagrams
    position: '700,900'
  - type: Software System
    name: FC4
    description: CLI tool; wraps SE
    position: '700,500'

relationships:
  - source: Software Documentarians
    description: invoke
    destination: FC4
  - source: FC4
    description: renders diagrams with
    destination: Structurizr Express

styles:
  - type: element
    tag: Element
    background: '#2486DC'
    color: '#ffffff'
    shape: RoundedBox
```

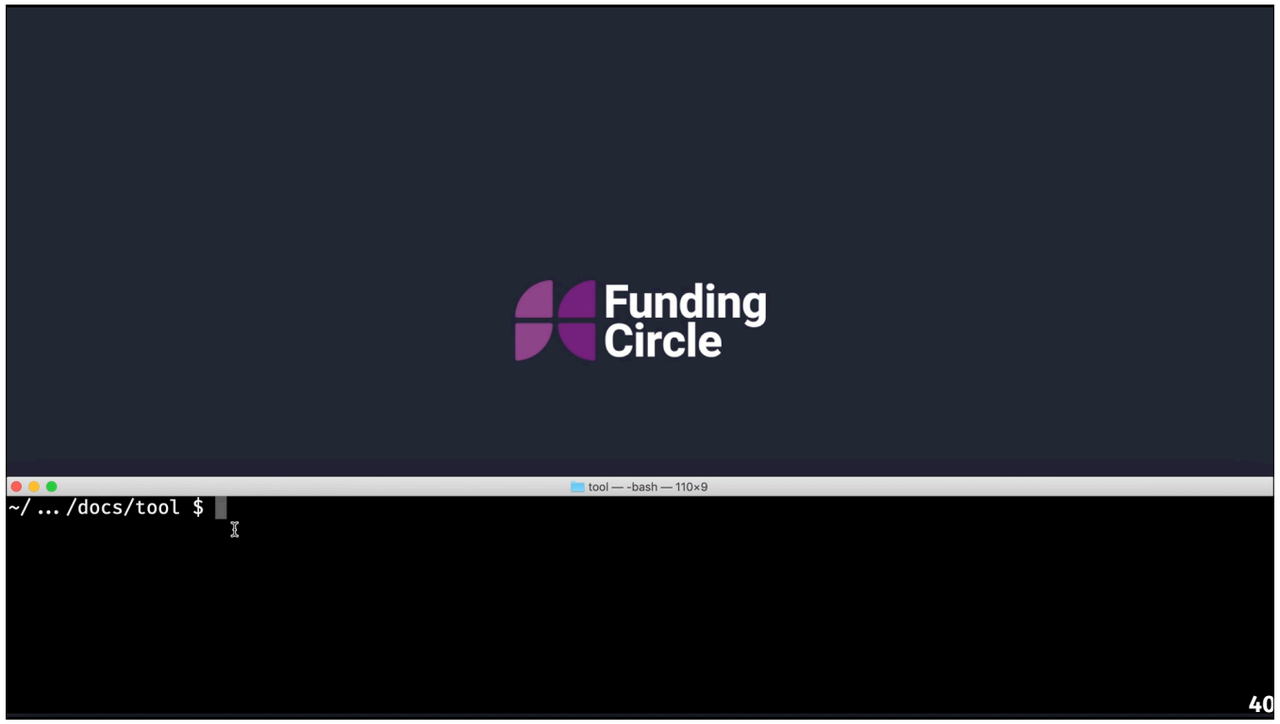

- Here's an example of the data structure that Simon Brown defined for Structurizr Express.
- SE is fantastic,
    - but it wasn't *quite* complete, from my perspective
- I wanted a file-based workflow
    - so I could follow the *Docs as Code* philosophy
- So I've created a tool that wraps SE
    - and facilitates that file-based workflow

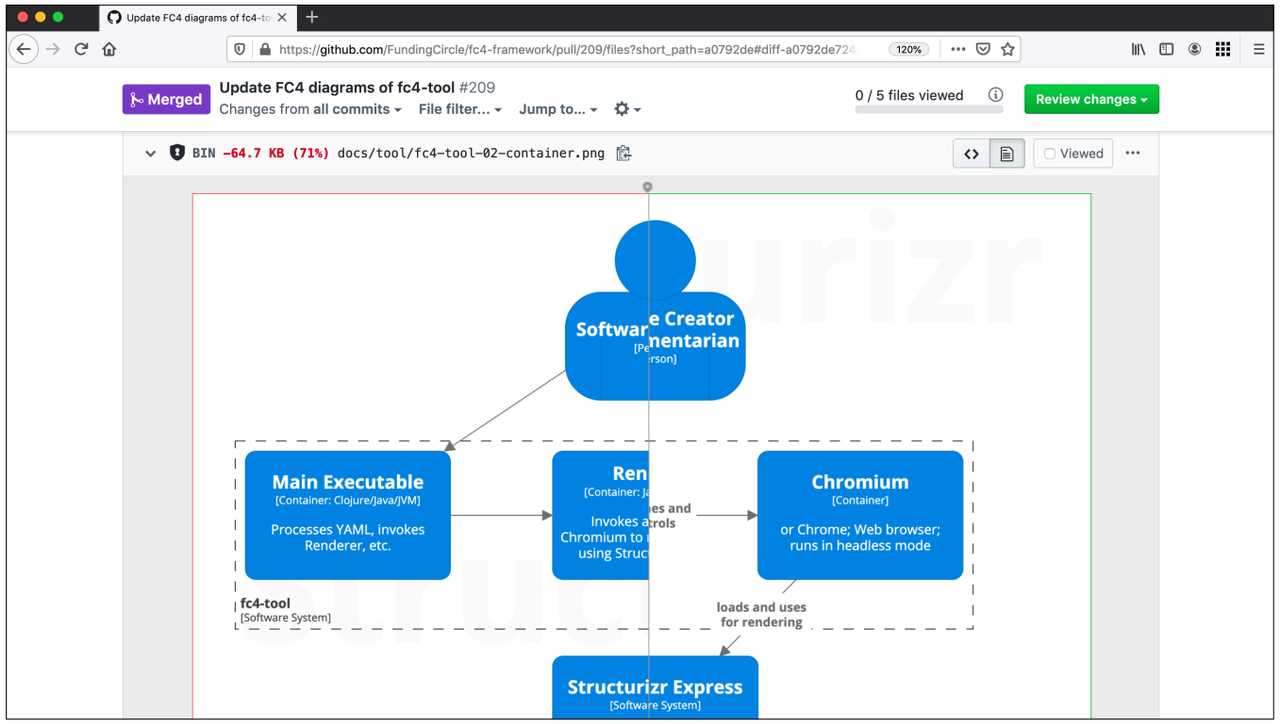


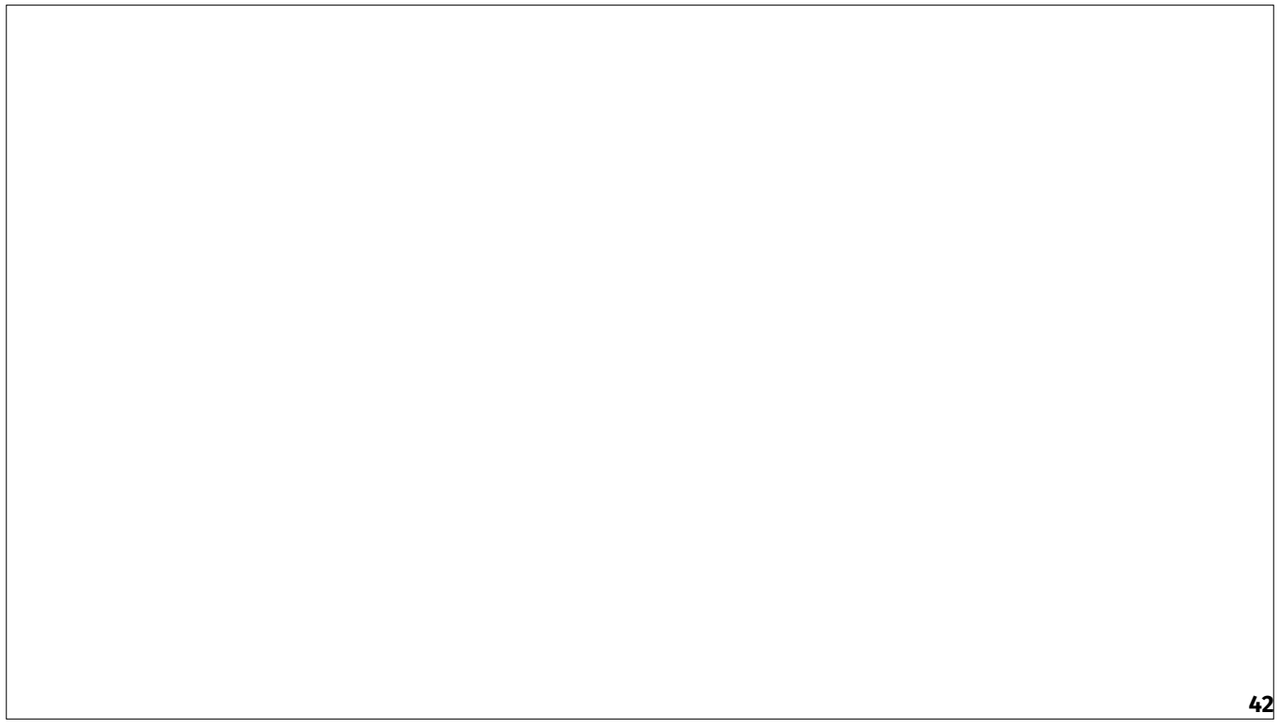
- It's called FC4
- It facilitates and supports
    - maintaining diagrams as files in a git repo
- Via these features:
    - reformatting and normalizing the diagram source files to prevent noisy diffs
    - automatically snapping the elements in a diagram to a virtual grid
        - which eliminates the drudgery of lining things up in a fiddly GUI
    - automatically rendering diagrams YAML→images
        - It's sort-of like a compiler for the diagrams
- It's implemented with Clojure,
    - it's open source with a BSD license,
    - and it's been supported by Funding Circle.



- Here’s a quick demo of an authoring session using FC4
- (narrate the video)


Funding
Circle
tool — -bash — 110×9
~/ ... /docs/tool $

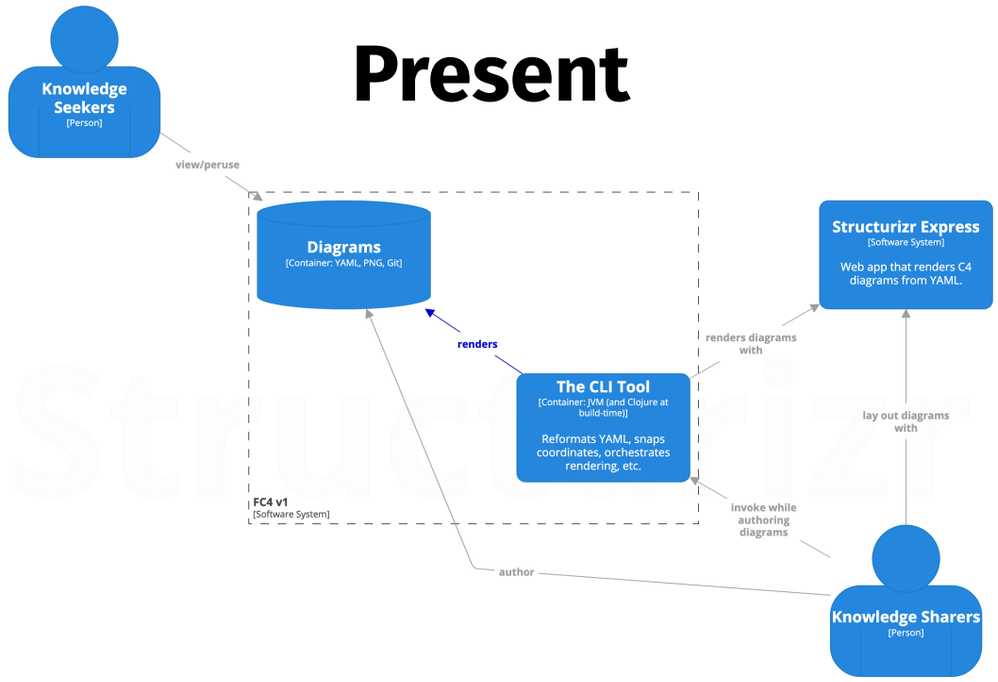

Update FC4 diagrams of fc4-to
https://github.com/FundingCircle/fc4-framework/pull/209/files?short_path=a0792de#diff-a0792de724
120%
Merged
Update FC4 diagrams of fc4-tool #209
Changes from all commits
File filter...
Jump to...
0 / 5 files viewed
Review changes
BIN -64.7 KB (71%) docs/tool/fc4-tool-02-container.png
Viewed
Main Executable
[Container: Clojure/Java/JVM]
Processes YAML, invokes Renderer, etc.
Chromium
[Container]
or Chrome; Web browser; runs in headless mode
fc4-tool
[Software System]
loads and uses for rendering
Structurizr Express
[Software System]



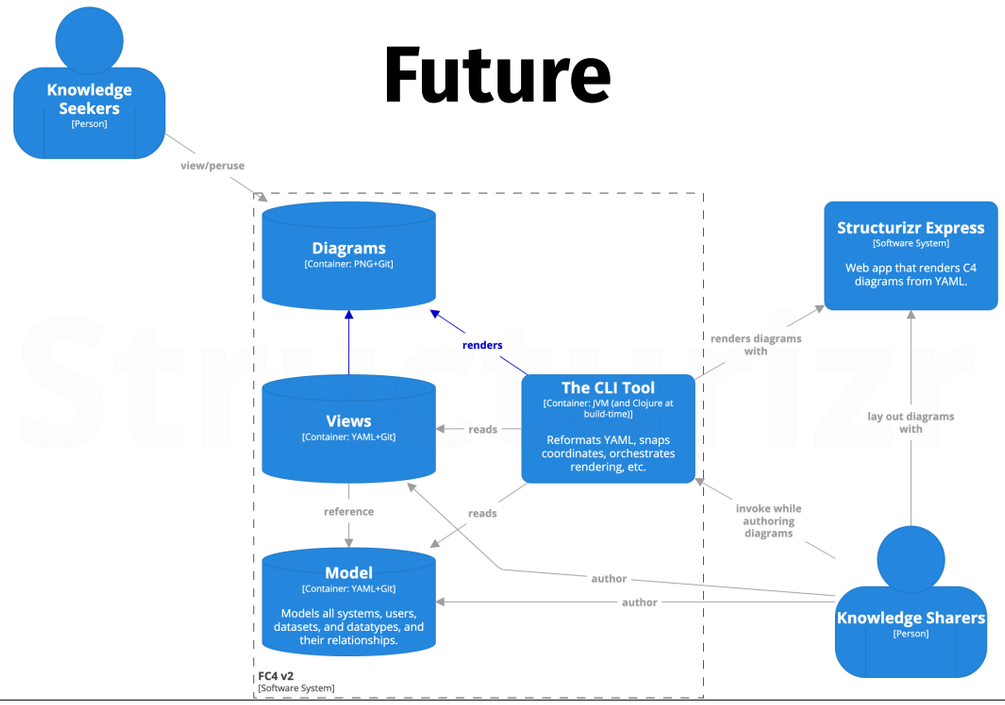
- **What is**
- If you'll indulge me, I'd like to talk about the future of FC4.
    - I'm hoping this isn't *too* self-indulgent;
    - Basically, FC4 has become the vehicle for realizing my vision for software architecture diagramming
    - So as my vision for diagramming evolves,
        - my vision for FC4 evolves correspondingly.
    - So describing my vision for FC4 is a way to convey my vision for software architecture diagramming,
        - in a somewhat concrete fashion.
- To talk about FC4's future,
    - I first need to point out a flaw in its current state.
- (reveal) This is the current state of FC4.
    - You can see there's basically two elements that compose FC4;
        - the CLI tool, and the corpus of diagrams.
- Over the past 2 years,
    - My colleagues and I have created a lot of diagrams using FC4.
- As we built up a substantial corpus of these FC4 diagrams,
    - in my org,
    - I observed increasing duplication and fragmentation in the data.
- Each diagram definition is completely self-contained.
- Therefore some elements are defined repeatedly,
    - in multiple places.
- It's been difficult to keep those definitions in sync across those files.
- We've been doing so entirely manually,

    - which is very tiresome;
    - so tiresome that we often put it off or neglect to do it.
- So I’ve been working on the next iteration of FC4 →

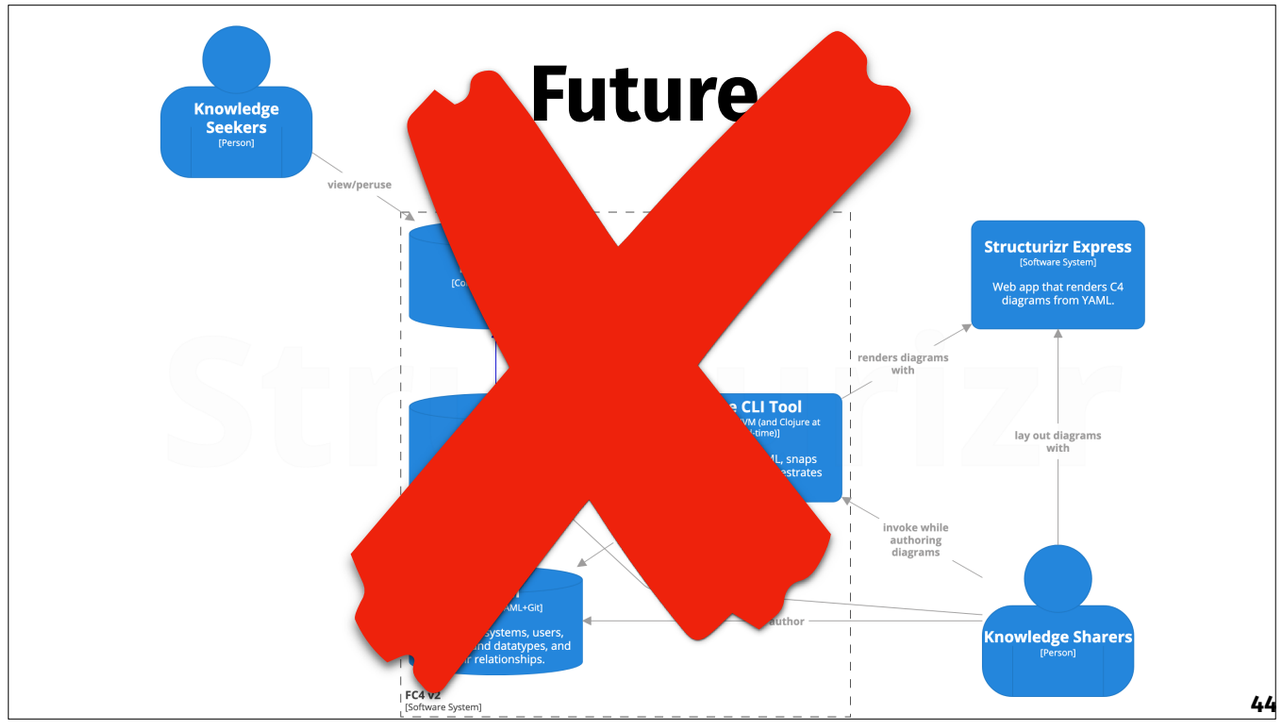
Present


Knowledge Seekers
[Person]
view/peruse
Diagrams
[Container: YAML, PNG, Git]
renders
The CLI Tool
[Container: JVM (and Clojure at build-time)]
Reformats YAML, snaps coordinates, orchestrates rendering, etc.
FC4 v1
[Software System]
renders diagrams with
Structurizr Express
[Software System]
Web app that renders C4 diagrams from YAML.
lay out diagrams with
invoke while authoring diagrams
author
Knowledge Sharers
[Person]

Future



- **What could be**
- We're going to change the data model,
    - So that a given FC4 corpus will consist of
    - A single unified model of all of the elements and their relationships
    - And a set of views that reference that model.
    - The views will yield diagrams.
    - This will reduce duplication, data drift, and toil, significantly.
- I've been working on this,
    - on and off, on the side,
    - for a while.
- But recently I've realized that this plan doesn't go quite far enough →

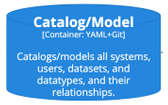


- So I’ve revised the plan for the future of FC4 →

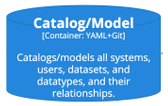
Future

- I've realized that,
    - as powerful as diagrams can be,
    - the *model* of the systems, *as data* —
    - that's actually *more valuable* than the diagrams.
- See, it's not just a *model,*
    - it's also a catalog. *(reveal)*
- Turns out:
    - that a centralized semantic catalogue of our systems has massive value.
- In fact, I've come up with a variation of Greenspun's Tenth Rule;
    - I hope you'll bear with me here *(reveal, read, hide)*
- Sufficiently large orgs need a centralized source of truth for this data
    - And a good way to maintain and improve it over time
- So I'm taking FC4 in a new direction,
    - So it'll be that single source of truth.
- Instead of being a diagramming tool,
    - It's going to become a toolkit
    - for cataloguing, modeling, and documenting software systems.
- Diagrams will be *one* of the kinds of documentation
    - that can be authored, published, and maintained via the toolkit.
- But the central focus of the toolkit
    - will be that centralized unified dataset
- There'll be various things one can do with that data →

Future

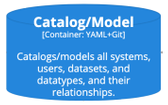

Catalog/Model
[Container: YAML+Git]
Catalogs/models all systems, users, datasets, and datatypes, and their relationships.

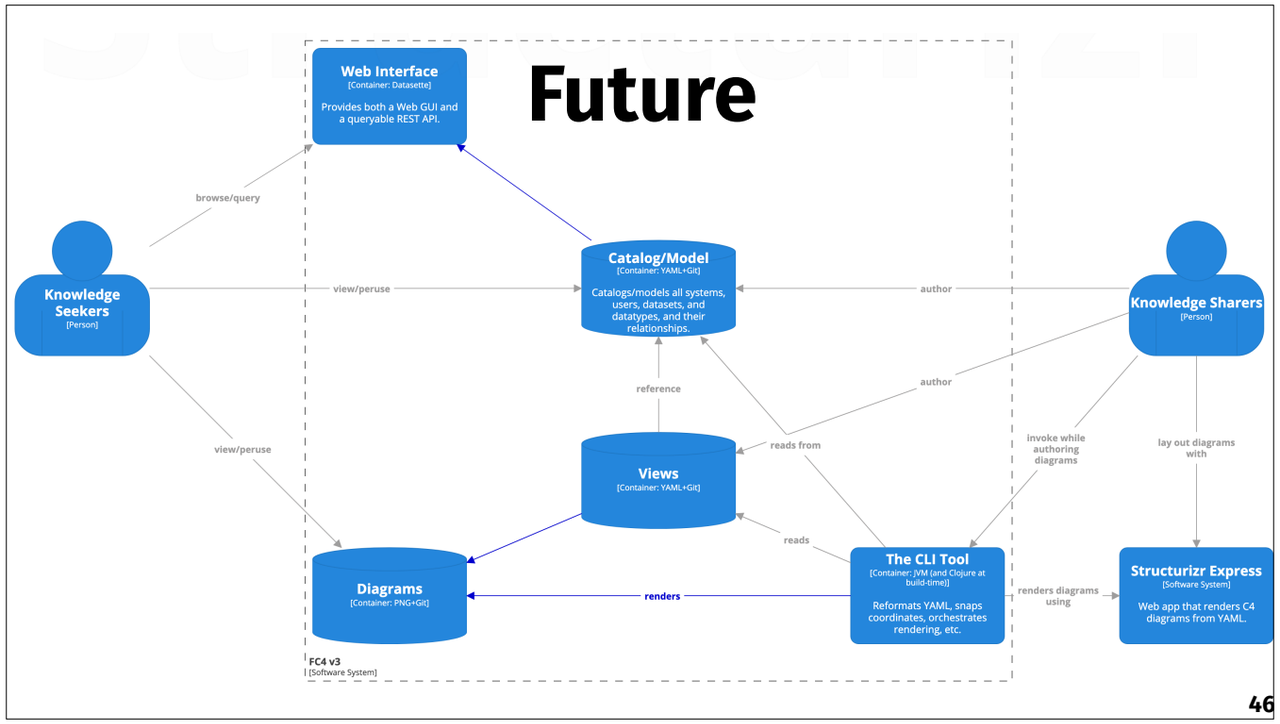
Future



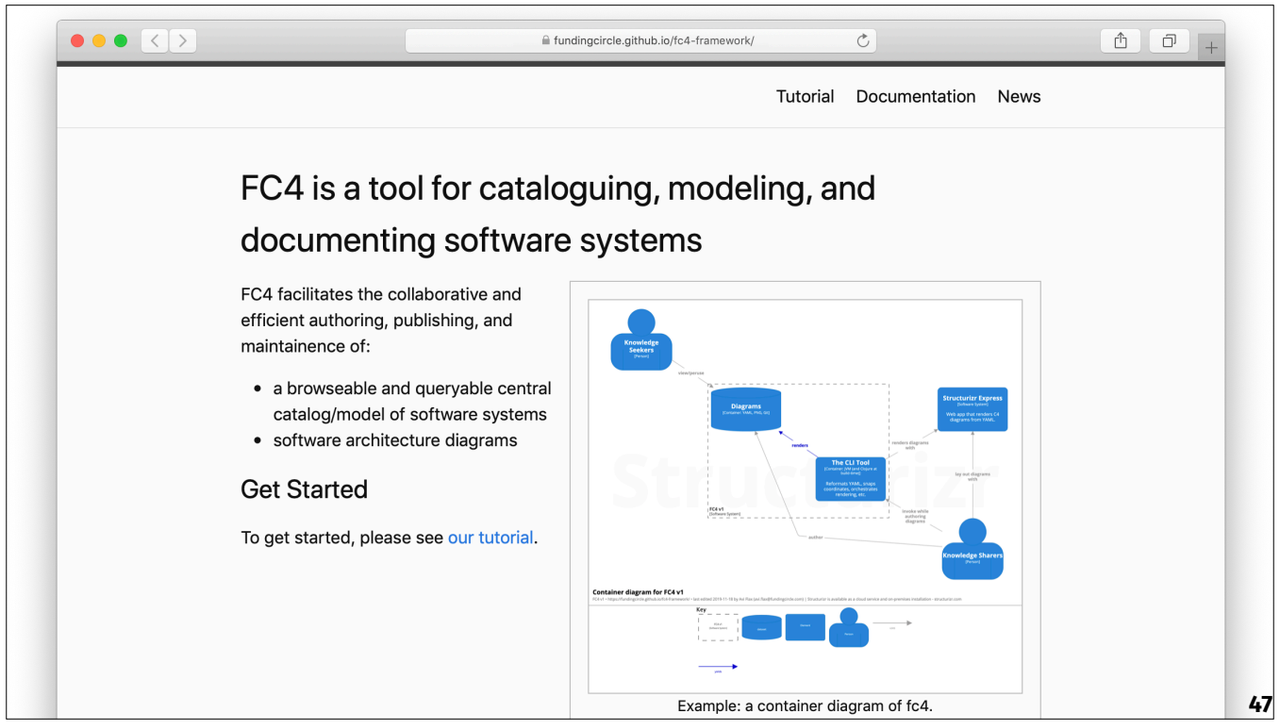
Each department of a sufficiently large software-driven organization contains an ad-hoc, informally specified, error-ridden, inaccessible, incomplete catalog of the organization's software systems.

Future


Catalog/Model
[Container: YAML+Git]
Catalogs/models all systems, users, datasets, and datatypes, and their relationships.

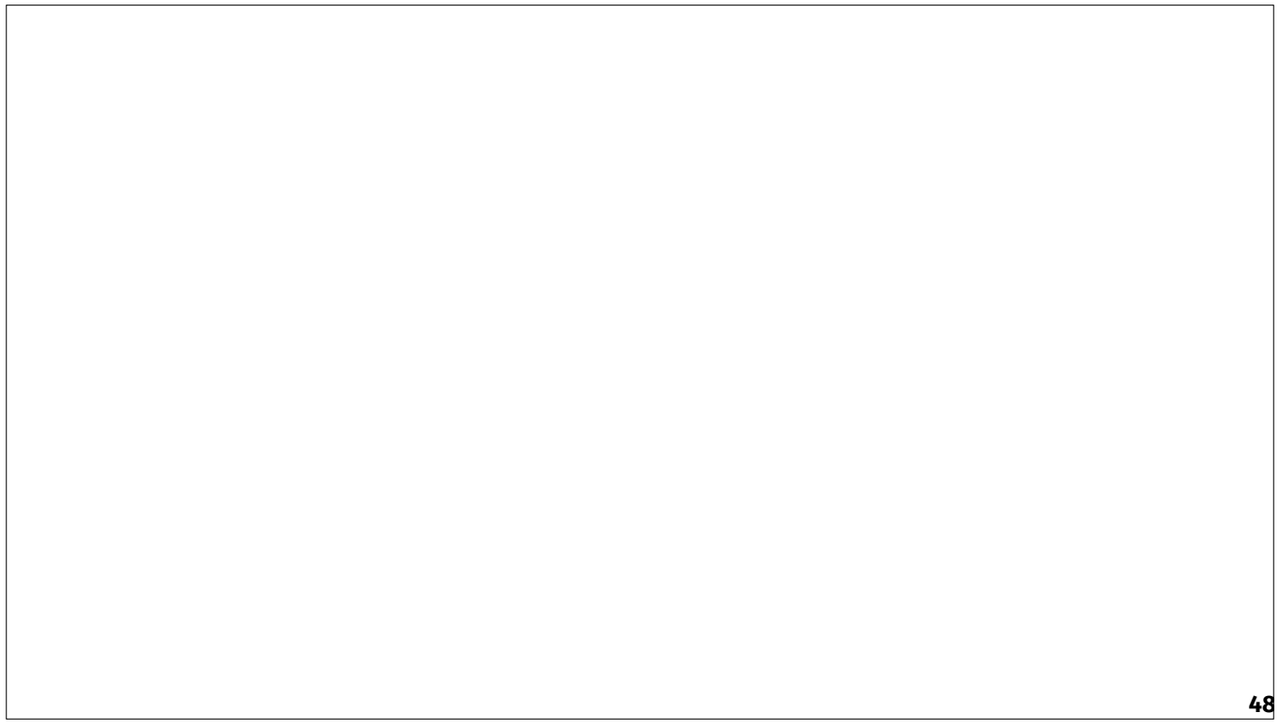


- You'll be able to author views
    - And render diagrams from those views
- You'll also be able to run a single command to produce a read-only Website
    - that provides for interactive browsing and querying of the data via a browser,
    - and for programmatic querying via HTTP



Example: a container diagram of fc4.

- Here's a preview of an upcoming version of the website
- (read the masthead)



- Since we're going to be decomplecting the model and the views,
    - we need a new data structure.
- If you recall,
    - SE's data structure is for self-contained diagrams,
    - that contain the full definition of every element and their relationships.
- So we need some new data structures;
    - one for the model and one for the views.
- I've designed these new data structures not just as data structures,
    - but also as a DSL,
    - a Domain Specific Language.
- Here's a model file. *(reveal)*
- *(quick walk-through of the file)*

```
systems:
  Structurizr Express:
    is-a: Web app
    that: renders diagrams from YAML
  FC4:
    is-a: CLI tool
    that: wraps SE
    uses:
      Structurizr Express:
        to: render diagrams
people:
  Software Documentarians:
    are: those who document software
    use:
      FC4:
        to: format, align, and render diagrams
```

```
systems:
  Structurizr Express:
    is-a: Web app
    that: renders diagrams from YAML
  FC4:
    is-a: CLI tool
    that: wraps SE
    uses:
      Structurizr Express:
        to: render diagrams
people:
  Software Documentarians:
    are: those who document software
    use:
      FC4:
        to: format, align, and render diagrams
```


- Now let’s move that over a little,
    - Make some room.
- So I can show you an FC4 view
- That’s it, that’s a complete view.
- Only two elements,
    - but that’s enough to show you the structure

```
systems:
  Structurizr Express:
    is-a: Web app
    that: renders diagrams from YAML
  FC4:
    is-a: CLI tool
    that: wraps SE
    uses:
      Structurizr Express:
        to: render diagrams
people:
  Software Documentarians:
    are: those who document software
    use:
      FC4:
        to: format, align, and render diagrams
```

```
subject: FC4
other-elements:
  people:
    Software Documentarians: [100, 100]
  systems:
    Structurizr Express: [600, 100]
```

Summary

- Just to summarize:
    - (reveal + read the slide)
- Finally,
    - as sort of meta-take-away,
    - I’ve come up with a variation on the last element of Python’s PEP 20, by Tim Peters: →

```
user⇒ (+ data-in-text-files
         rendering-tools)
("awesome diagrams"
 "peer reviews"
 "same workflow for docs and code"
 "generate diagrams from other data"
 "and more!")
```

Data orientation is one honking great idea — let's do more of that!

- (read the words)
- Just in case it's not clear;
    - it's not a coincidence that my work has become
    - more and more data-oriented over the past few years;
    - Clojure and the Clojure community have been instrumental
    - in spurring me to adopt this mindset.
    - For which I'm grateful.

Call to action

- So:
    - Next time you need to create an architecture diagram
    - Before you break out Visio, draw.io, LucidChart, OmniGraffle, etc
    - Try this:
        - Pick a tool
        - Whether it's Mermaid, PlantUML, SE, FC4, or something else
            - When choosing a tool, try to choose one that has a data format that appeals to you.
        - Author your diagram as *data in a text file*
        - Render an image using the tool
        - Commit the two files — the diagram source and the rendered image — to a git repo
        - Use the same peer review workflow you use for your code
- **New bliss**
    - I think you will find this approach has lots of benefits
    - And it opens the door to new, adjacent possibilities, that are enabled by authoring your diagrams as *data in text files*

Image: https://pixabay.com/photos/video-production-video-movie-film-4223885/

Thank You!

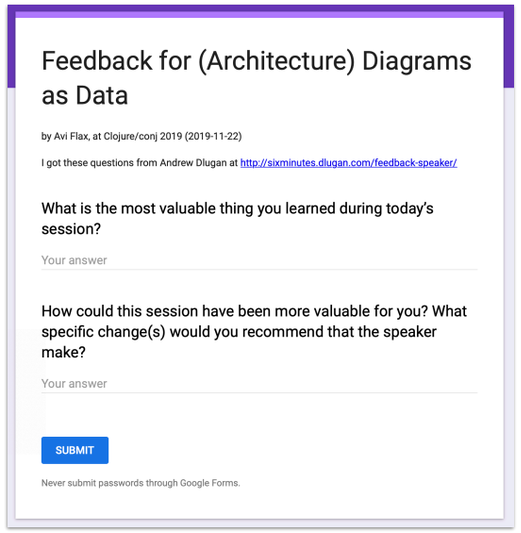
- Thank you to all of you for your time.
- Thank you to the creators of,
    - and contributors to, Clojure,
    - and to the entire community,
    - for all of your contributions to our field in general,
    - and to my personal and professional development.
- I’m truly grateful to all of you.
- And I’d be *even more* grateful if…

Feedback Please!

https://forms.gle/dTvYWENLRthMD7JZ9

Feedback Please!

https://forms.gle/dTvYWENLRthMD7JZ9



Feedback Please!

https://forms.gle/dTvYWENLRthMD7JZ9

Let's Talk!

```
{:electronic-mail
 "avi@aviflax.com"
 :clojureverse
 "@avi"
 :clojurians
 "@avi"
 :twitter
 "@flaximus"}
```


- *If there's time for questions:*
    - I have a few minutes for questions,
    - so if you have a question,
        - please raise your hand.
    - Please don't feel obligated to stick around;
        - the talk is over,
        - Everyone should feel free to head out without feeling bad about it.
    - We can always talk later.