SE 577
Software Architecture

Architectural Modeling

Modeling is a technique used to tame complexity

- Models are used to capture abstractions of complex systems to make them easier to reason about.
  - What are the key questions that you need to be able to answer
  - Identify the optimal level of abstraction at which to model
  - Identify what "objects" will be modeled and the types of interactions you will model
  - Identify the possible states of the system including both system states and the state of their runtime environment
  - Identify how interactions will change (if at all) over time

So what are software architecture models?

- Architecture models capture the foundational design decisions about a system.
  - An architecture model is an **artifact** that captures these decisions
  - These models serve as documentation that can be consumed by various stakeholders

Why is the ability to document architecture important?

Its important to be able to document an architecture to reason about its design for a number of reasons…

- **Program Understanding** – we need to make modifications to the system, where would the modifications go, how long would they take, how much would they cost
- **Explaining the operation of the system to stakeholders** – how does the system work, what are its major features, why should you use this system versus a competitors solution
- **Providing a common document** to capture the high-level design of the system that can be used to make important management and technical decisions
- **Measuring technical health and technical debt** – how far does the "as built" system deviate from the "as designed" architecture?

Modeling choices are important!

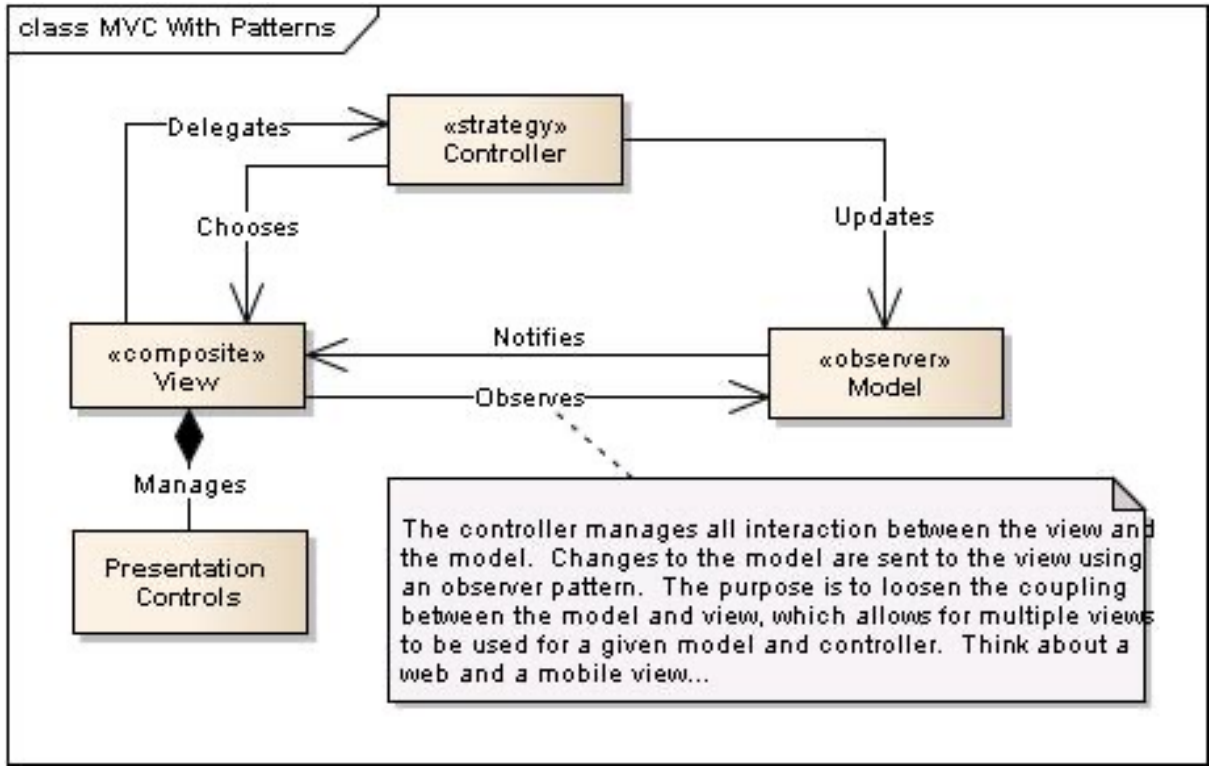
- The choice of what to model is important!
  - This activity takes time and costs money so picking what to model becomes a critical activity
- The choice of how detailed your models are are important!
  - What stakeholders are you trying to influence?
  - How much detail is needed to describe the architecture to the stakeholders?
  - Does the model describe the design, or is the model intended to influence a decision?
- The choice of what notation is used to document your architecture is important!
  - Should you use formal, or semi-formal notations

Modeling is about documenting key architecture and design decisions!

- A **model** is an **artifact** that captures a number of different design decisions used to establish the overall system architecture
  - This activity takes time and costs money so picking what to model becomes a critical activity
- Key things to consider when modeling
  - What architectural decisions and concepts should be modeled,
  - At what level of detail, and
  - With how much rigor or formality

What do we document in a model?

- Components
- Connectors
- Interfaces (including constraints)
- Decisions and Rational
- Structural, Runtime and Behavioral Constraints

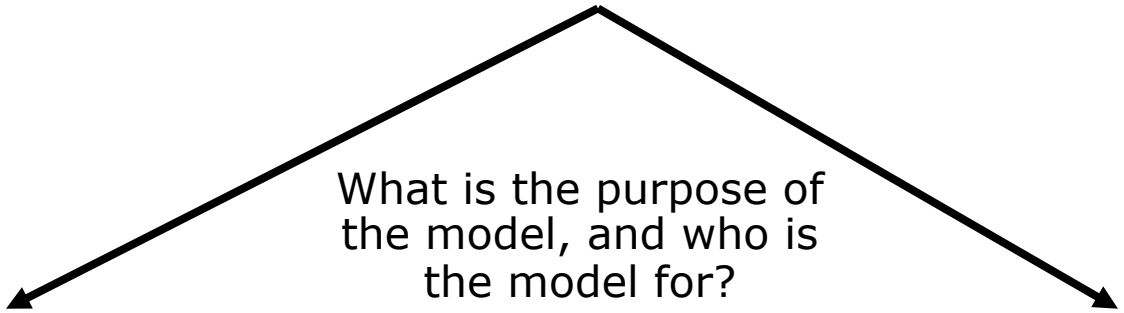


Remind you of the definition of software architecture?

Thinking about models…

To experiment
To clarify
To understand
To analyze
To evaluate

Reason for Modeling



What to Model

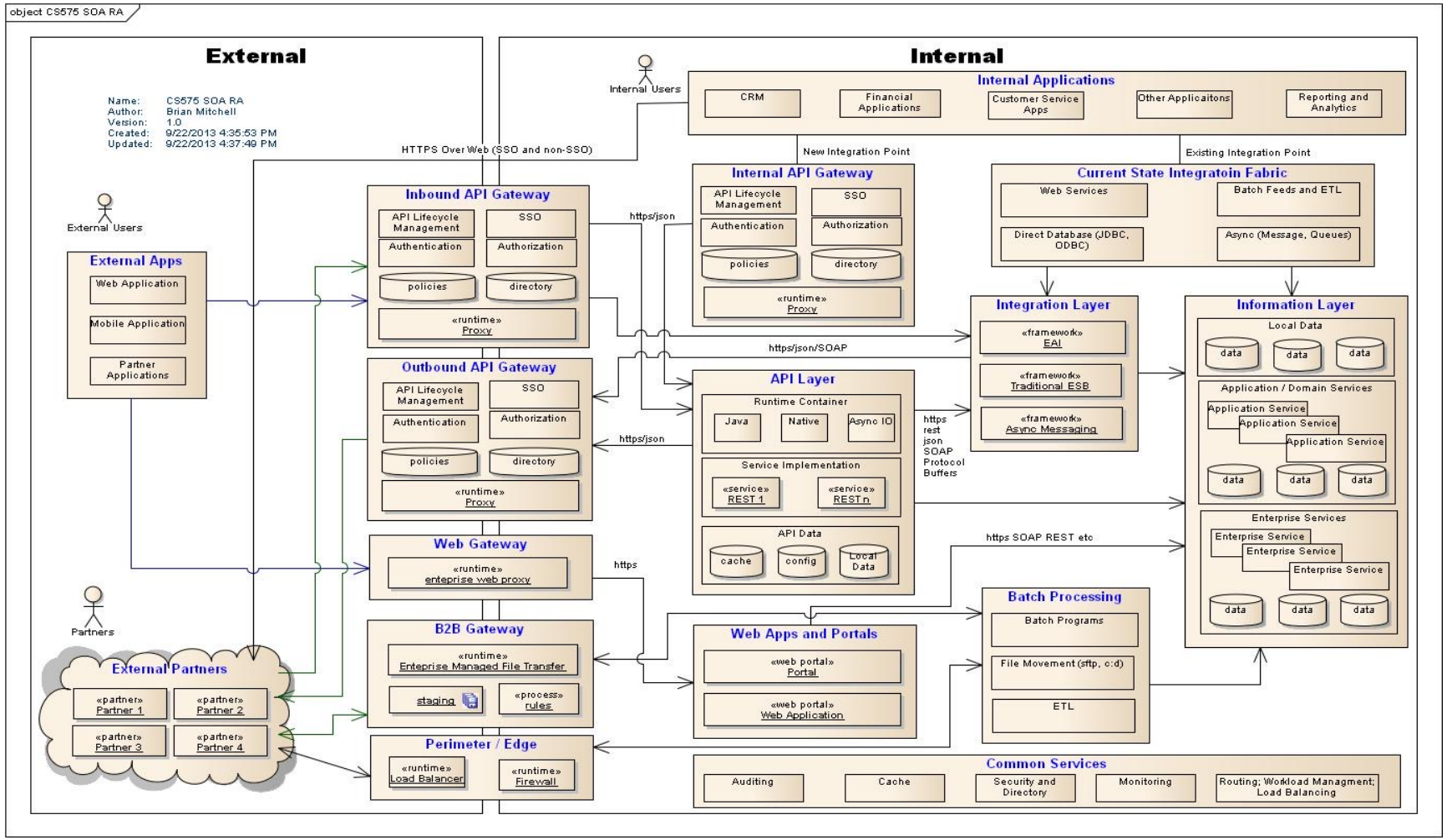
Structure
Transformations
State
Inputs and Outputs

How to Model

Textual
Graphical
Formal / Mathematical

Adapted From: https://www.cl.cam.ac.uk/teaching/1112/SWDesign/softwaredesign01.pdf

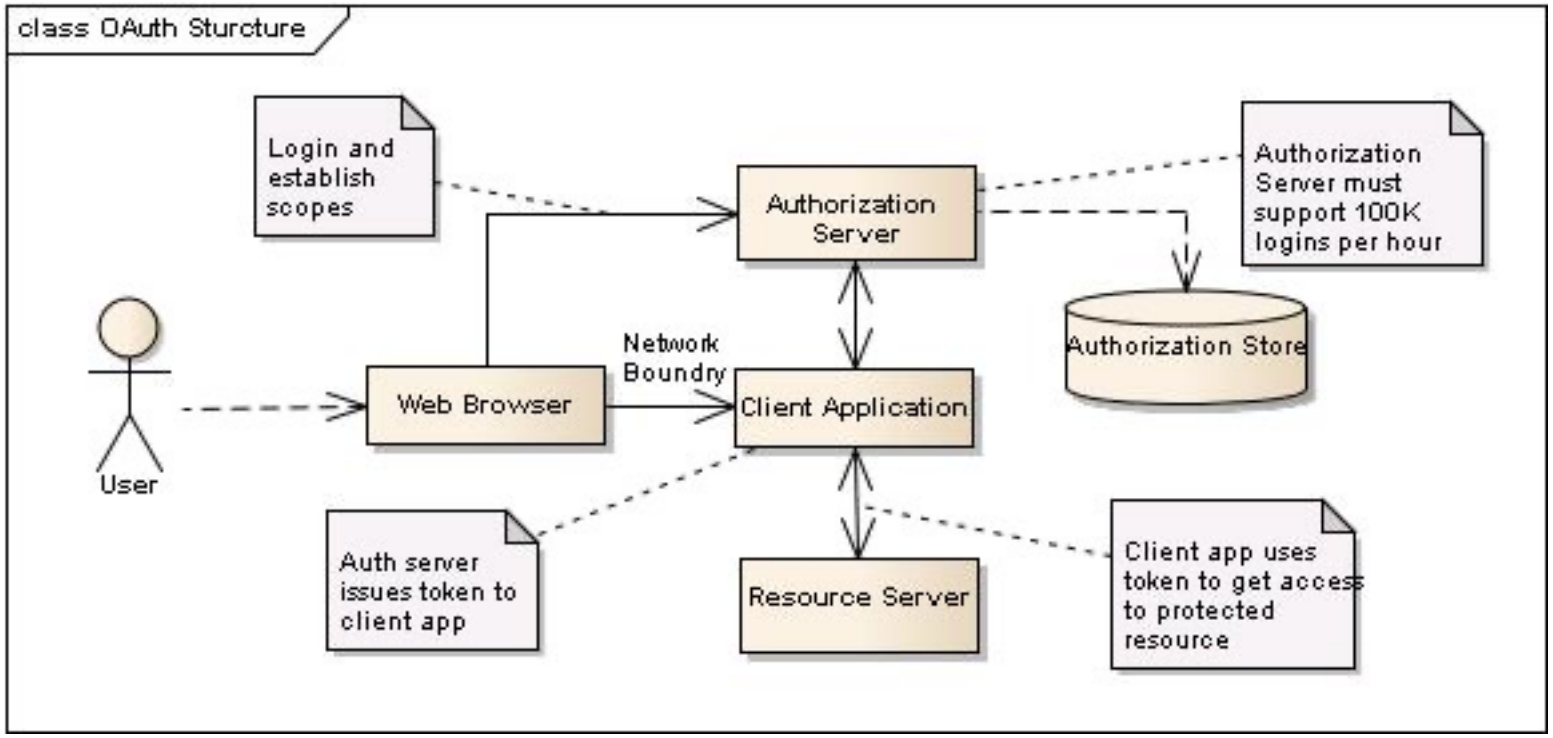
Example: What are some of the key aspects of this model?



Modeling the different views – Static and Dynamic

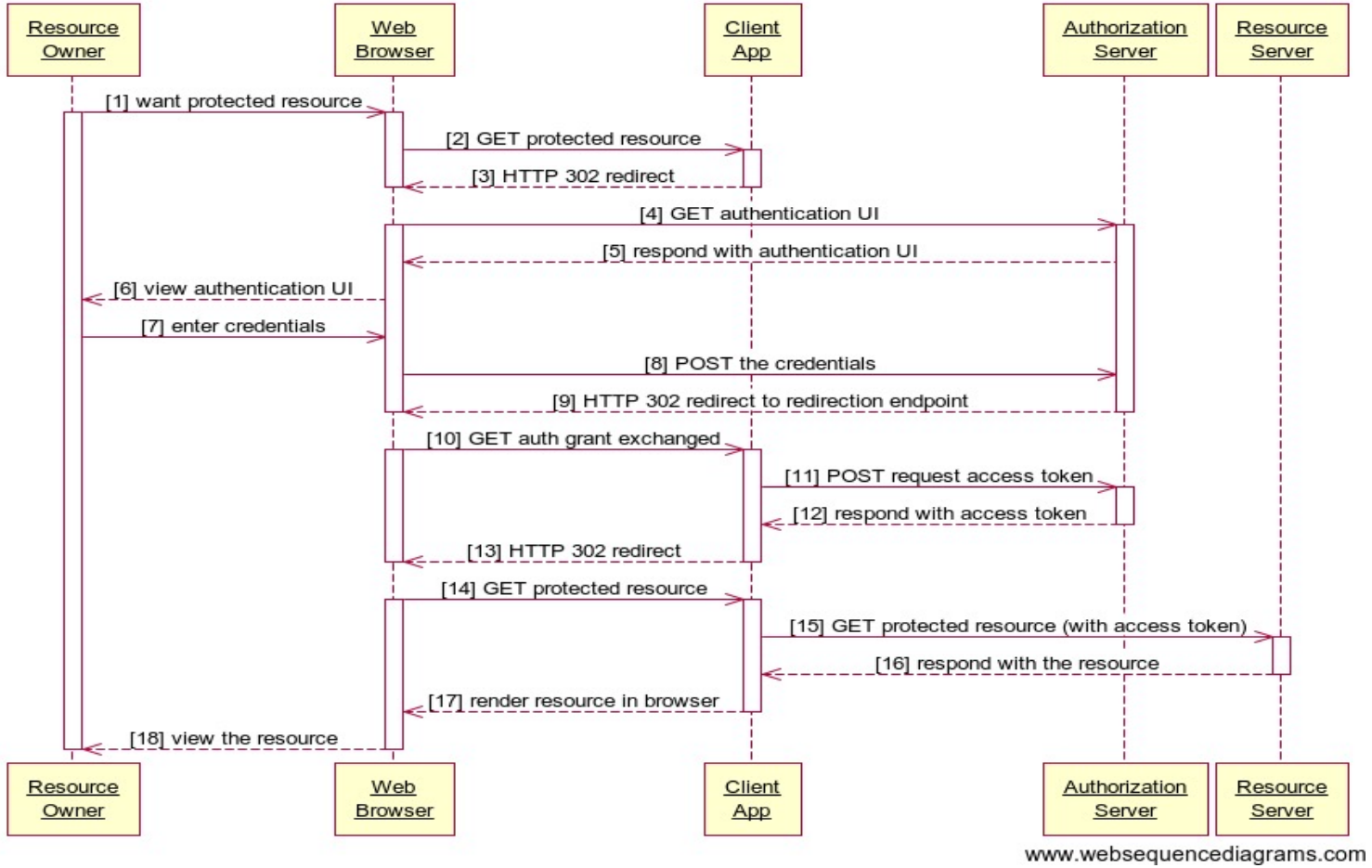
- Static models describe the structure of the system
  - These structures generally do not change over time
  - Capture the topology of the system components:
    - can be software component relations – uses/inherits
    - Can be runtime component relations – sends message to over HTTP
    - Can be deployment component relation – this software is deployed to this server…
- Dynamic models capture the behavior of the system during runtime
  - The important thing to capture is change is being managed over time
  - State of the components
  - Data flow over the connectors

Static Model View (Example: oAuth)



Is this view useful to discuss the components associated with managing the oAuth protocol (functional and non-functional)?

Dynamic Model View (Example: oAuth)



Is this view useful to discuss how the components implementing the oAuth protocol work together?

Possible problems using multiple views to model software architectures

- Modeling a realistic software architecture using a single view is not practical
  - Too complex
  - Different concerns that don't belong together – static, dynamic, deployment
- Key challenge is to ensure that all the different views are consistent
  - Might be hard to find these issues
    - **Direct** – one diagram states two servers, another states three servers
    - **Indirect** – one view is at a higher level than another and the refinement introduced inconsistencies
    - **Structure vs Dynamic** – structures don't support dynamic requirements
    - **Functional vs Non-Functional** – The structure does not support the non-functional requirements

Modeling (Documentation) Objectives

- Useful now, represents the best knowledge of key architecture and design decisions
- Is contextual to the key problems or design decisions that need to be made
- Is appropriate to a targeted stakeholder
- Longer term, captures key decisions made that are useful for maintenance
- As little as possible, documentation requires maintenance

Where to Start? Consider what needs to be modeled!

The hardest part of reconstructing an architecture is selecting what needs to be modeled …

- **Components** – hierarchical description of the major subsystems
- **Connectors** – connections between the components
- **Interfaces** – the protocols governing the connectors, or the properties managed by the clients
- **Patterns and Styles** – are there any interesting patterns or styles used in the architecture that should be documented?
- **Rationale** – reasoning behind decisions of the aspects we chose to model – are they important to stakeholders? are they critical to the understanding of the system?
- **Constraints** – what dependencies don't we want to have in the solution

Where to Start? Consider what needs to be modeled!

Think about what is important to show to promote an understanding of the architecture…

- Static aspects are things that do not change as the system runs, and
- Dynamic aspects are things that do change, manage important state, or are sequence dependent
- Platform aspects are things where the runtime or deployment decisions play an important part of the architecture – load balancers, proxy servers, etc.

What about redocumenting existing systems from the source code

How do we find things that we want to model…

- Look at the code
  - Source code analysis tools
  - Class/package structure
  - Build / configuration management information
  - Test cases – what is being tested – that must be important
  - Directory structures
  - Naming conventions
- What do we know about the system
  - What are the main features
  - How are they exposed to the users
  - Build / configuration management information
  - Test cases – what is being tested – that must be important
  - Directory structures
  - Naming conventions
  - Non-functional aspects

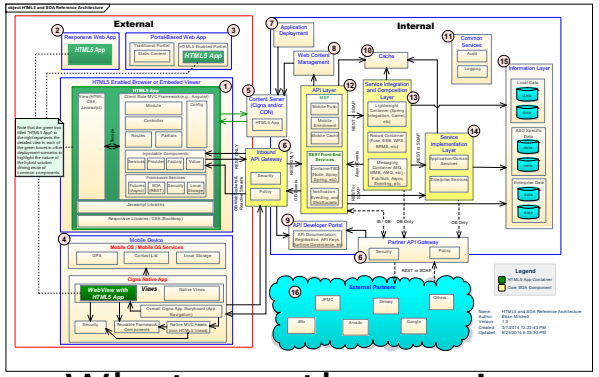
Think about what story that you want to tell to identify important things that need to be modeled

- Think about how you were taught to write stories in elementary school:
  - **Who?** Who is your stakeholder or target audience
  - **What?** What is the key message you are trying to get across
  - **Why?** Why is it important
  - **Where?** Where are the most important parts – think scale, security, etc
  - **When?** When do key events happen?
  - **How?** How does it work?
  - **How Much?** How much impact does the system have on an existing landscape?

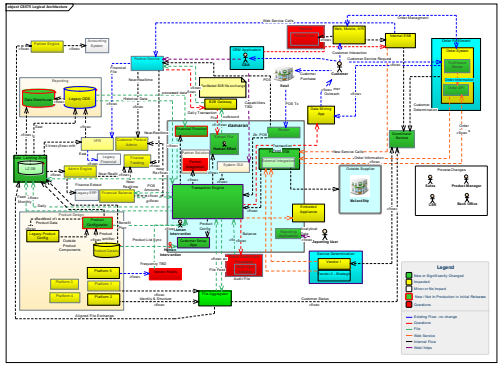
And the outcome you want to achieve

- Why do I create a Model / View:
  - Drive stakeholder clarity
  - Making quality / informed decisions
  - Forcing others to make decisions
  - Being transparent around the solution – what it is and what it is not
  - Being opinionated around constraints
  - Show alignment, and possible misalignment with enterprise or industry standards

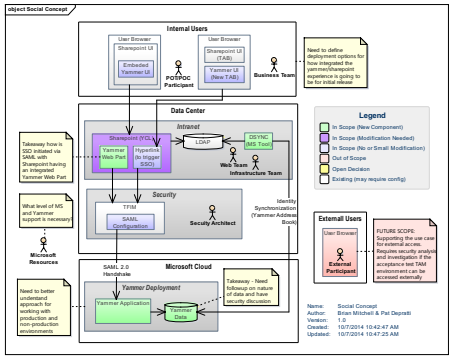
Be clear on the outcome you are trying to accomplish



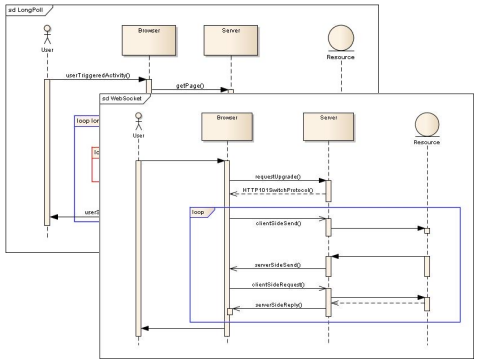
What are the major Components?



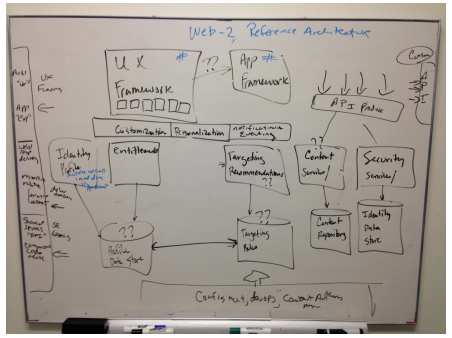
What is being impacted?



Who are the important stakeholders?



How does this complex behavior work?



How do we message what needs to change?

Is this even feasible?

MOST IMPORTANT CONCEPT IN THIS LECTURE

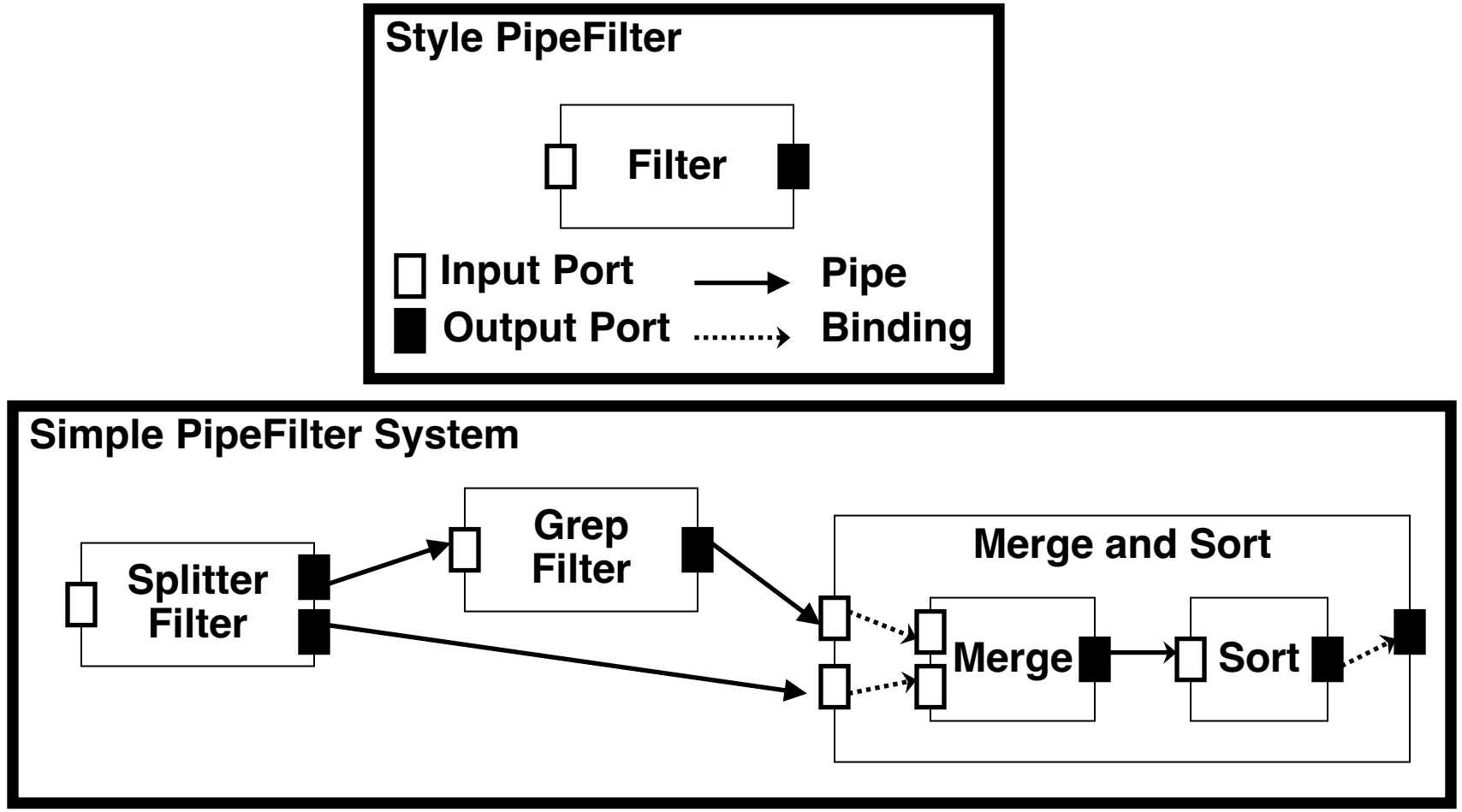
At the end of the day, a model is a quasi-useless artifact unless:

1. It helps **you as the architect, or your architecture team decompose the complexity of the problem domain**, or
2. Its **purposefully targeted at one or more key stakeholders** to facilitate alignment on:
    - Decision that have, or will need to be made on the solution to acheive its functional or non-functional requirements
    - Aspects around tradeoffs driven by constraint balancing that require agreement
    - Agreement related to technical debt that require socialization
    - Agreement around technical stack or deployment options that impact or constrain engineering staff
    - Alignment around which decisions need to be made when, even if those decisions have not been made yet.

Describing Architectures

- Formal Approaches
  - Architectural Description Languages
  - Many to date
- Informal/Semi-Informal Approaches
  - UML Notation – Simple and known by a broader community
    - UML is missing architecture semantics
  - Lines and Boxes (complemented) with natural English
  - Other Domain Specific Notations – C4

Method 1 – Formal Architecture Description Languages; Pipe and Filter Example



Lets see how we can formally specify this next ->

Method 1 – Formal Architecture Description Languages; Pipe and Filter Example

```
Family PipeFilter = {
  Port Type OutputPort;
  Port Type InputPort;
  Role Type Source;
  Role Type Sink;
  Component Type Filter;
  Connector Type Pipe = {
    Role src : Source;
    Role snk : Sink;
    Properties {
      latency : int;
      pipeProtocol: String = …;
    }
  };
};

System simple : PipeFilter = {
  Component Splitter : Filter = {
    Port pIn : InputPort = new InputPort;
    Port pOut1 : OutputPort = new OutputPort;
    Port pOut2 : OutputPort = new OutputPort;
    Properties { … }
  };
  Component Grep : Filter = {
    Port pIn : InputPort = new InputPort;
    Port pOut : OutputPort = new OutputPort;
  };
  Component MergeAndSort : Filter = {
    Port pIn1 : InputPort = new InputPort;
    Port pIn2 : InputPort = new InputPort;
    Port pOut : OutputPort = new OutputPort;
    Representation {
      System MergeAndSortRep : PipeFilter = {
        Component Merge : Filter = { … };
        Component Sort : Filter = { … };
        Connector MergeStream : Pipe = new Pipe;
        Attachments { … };
      }; /* end sub-system */
      Bindings {
        pIn1 to Merge.pIn1;
        pIn2 to Merge.pIn2;
        pOut to Sort.pOut;
      };
    };
  };
  Connector SplitStream1 : Pipe = new Pipe;
  Connector SplitStream2 : Pipe = new Pipe;
  Connector GrepStream : Pipe = new Pipe;
  Attachments {
    Splitter.pOut1 to SplitStream1.src;
    Grep.pIn to SplitStream1.snk;
    Grep.pOut to GrepStream.src;
    MergeAndSort.pIn1 to GrepStream.snk;
    Splitter.pOut2 to SplitStream2.src;
    MergeAndSort.pIn2 to SplitStream2.snk;
  };
}; /* end system */
```

Thoughts on Formal Architecture Modeling

How does this….

```
Family PipeFilter = {
  Port Type OutputPort;
  Port Type InputPort;
  Role Type Source;
  Role Type Sink;
  Component Type Filter;
  Connector Type Pipe = {
    Role src : Source;
    Role snk : Sink;
    Properties {
      latency : int;
      pipeProtocol: String = …;
    }
  };
};

System simple : PipeFilter = {
  Component Splitter : Filter = {
    Port pIn : InputPort = new InputPort;
    Port pOut1 : OutputPort = new OutputPort;
    Port pOut2 : OutputPort = new OutputPort;
    Properties { … }
  };
  Component Grep : Filter = {
    Port pIn : InputPort = new InputPort;
    Port pOut : OutputPort = new OutputPort;
  };
  Component MergeAndSort : Filter = {
    Port pIn1 : InputPort = new InputPort;
    Port pIn2 : InputPort = new InputPort;
    Port pOut : OutputPort = new OutputPort;
    Representation {
      System MergeAndSortRep : PipeFilter = {
        Component Merge : Filter = { … };
        Component Sort : Filter = { … };
        Connector MergeStream : Pipe = new Pipe;
        Attachments { … };
      }; /* end sub-system */
      Bindings {
        pIn1 to Merge.pIn1;
        pIn2 to Merge.pIn2;
        pOut to Sort.pOut;
      };
    };
  };
  Connector SplitStream1 : Pipe = new Pipe;
  Connector SplitStream2 : Pipe = new Pipe;
  Connector GrepStream : Pipe = new Pipe;
  Attachments {
    Splitter.pOut1 to SplitStream1.src;
    Grep.pIn to SplitStream1.snk;
    Grep.pOut to GrepStream.src;
    MergeAndSort.pIn1 to GrepStream.snk;
    Splitter.pOut2 to SplitStream2.src;
    MergeAndSort.pIn2 to SplitStream2.snk;
  };
}; /* end system */
```

Help Achieve any of this?

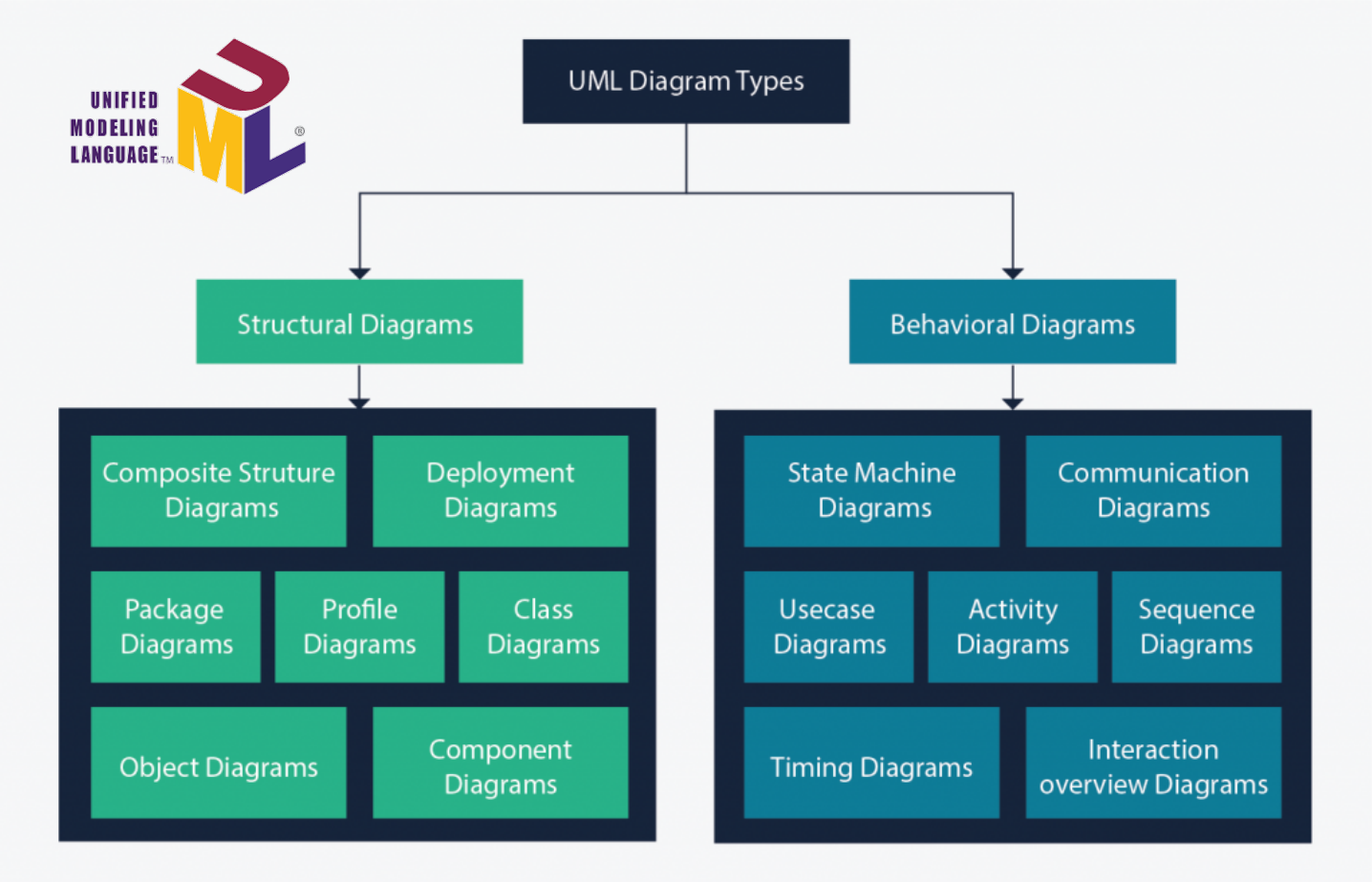
MOST IMPORTANT CONCEPT IN THIS LECTURE

At the end of the day, a model is a quasi-useless artifact unless:

1. It helps **you as the architect, or your architecture team decompose the complexity of the problem domain**, or
2. Its **purposefully targeted at one or more key stakeholders** to facilitate alignment on:
   - Decision that have, or will need to be made on the solution to acheive its functional or non-functional requirements
   - Aspects around tradeoffs driven by constraint balancing that require agreement
   - Agreement related to technical debt that require socialization
   - Agreement around technical stack or deployment options that impact or constrain engineering staff
   - Alignment around which decisions need to be made when, even if those decisions have not been made yet.

My personal view – I have never found any instance where formal methods for architecture modeling have made sense. However, I think the sweet spot for formal methods is in design verification, especially for safety critical systems. I like the work of Wane Hillel in this space, ref: https://www.hillelwayne.com/tags/formal-methods/. See his work on TLA+ and Alloy - https://www.hillelwayne.com/projects/

UML as a Modeling Language for Architecture

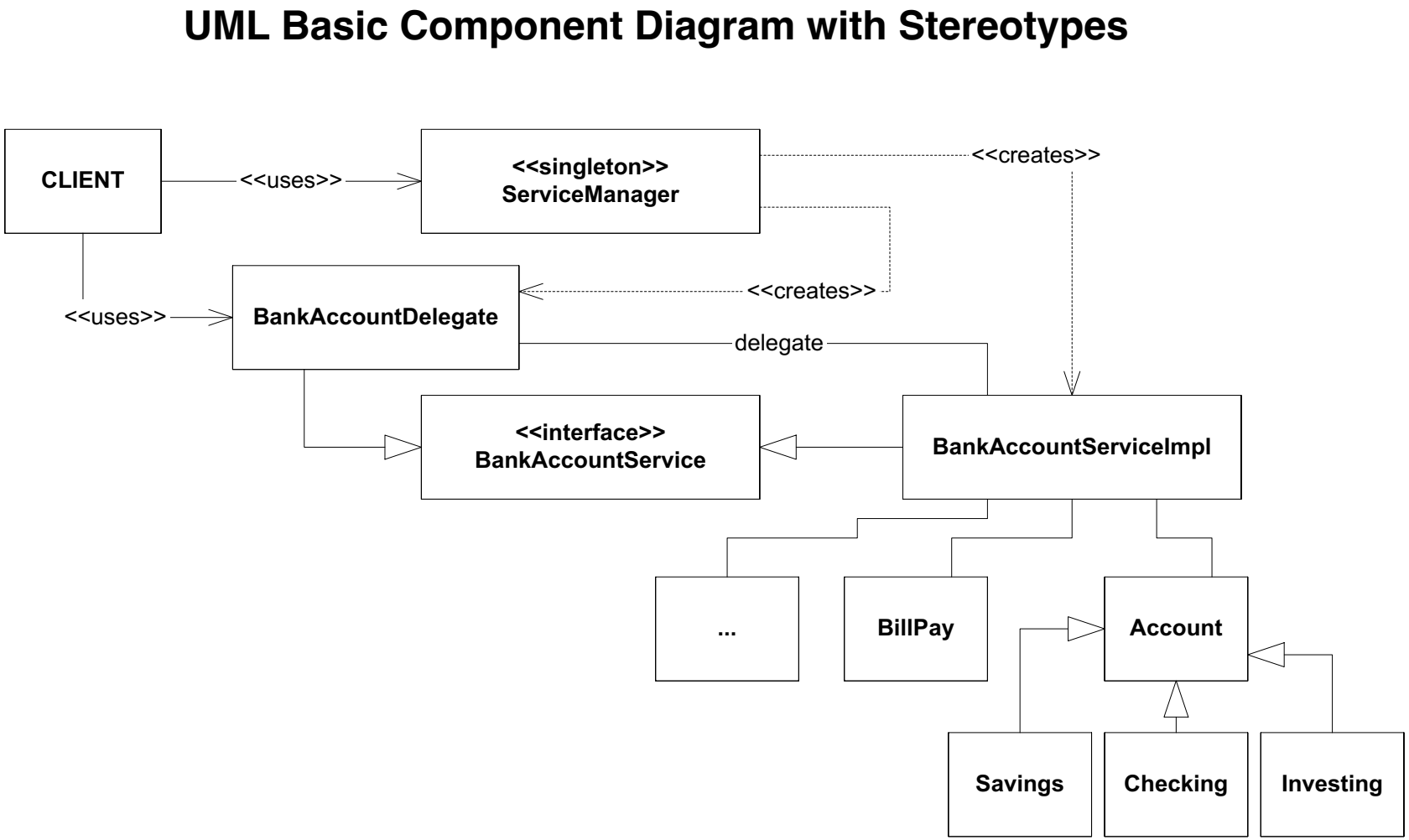


UML is a semi-formal visual language which specifies 14 different types of diagrams that cover both static and dynamic modeling needs

While UML is expressive enough to support architecture modeling, its primary abstractions are geared towards source code components and connectors vs architectural ones.

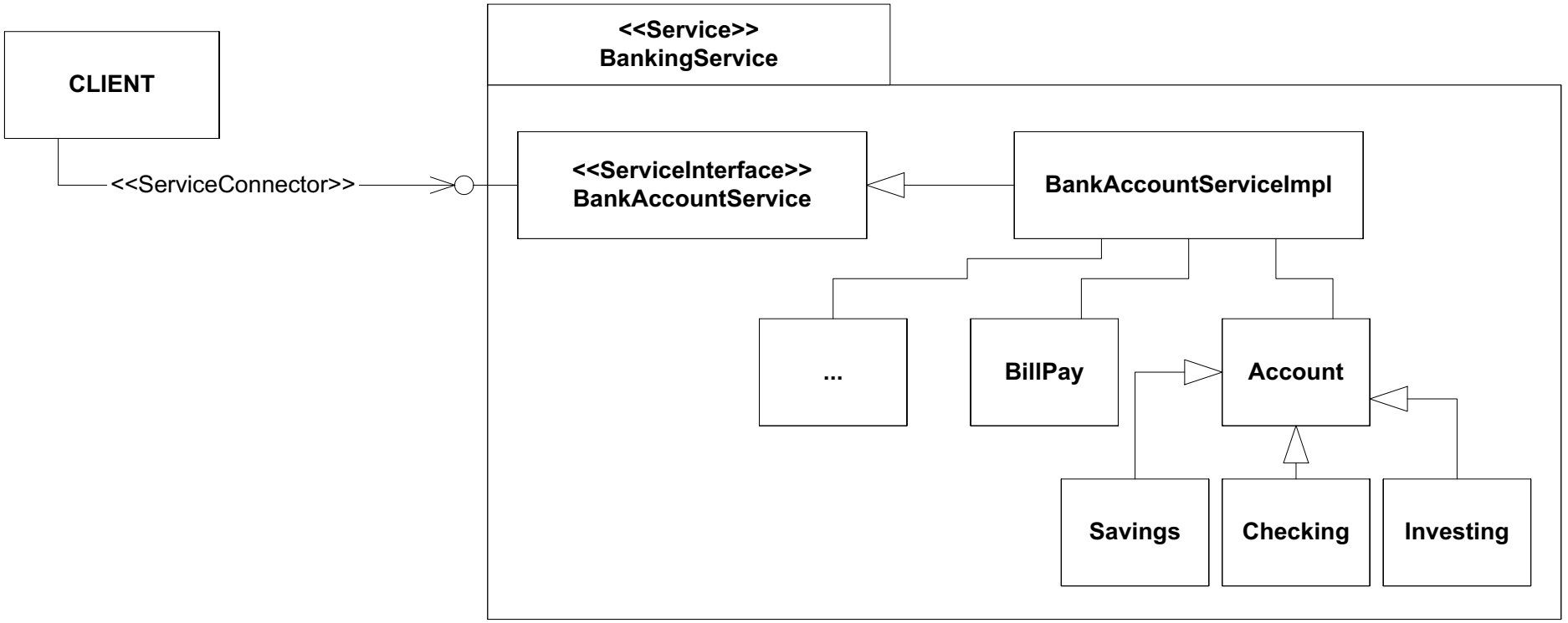
https://creately.com/blog/diagrams/uml-diagram-types-examples/

Example: UML to model a banking service

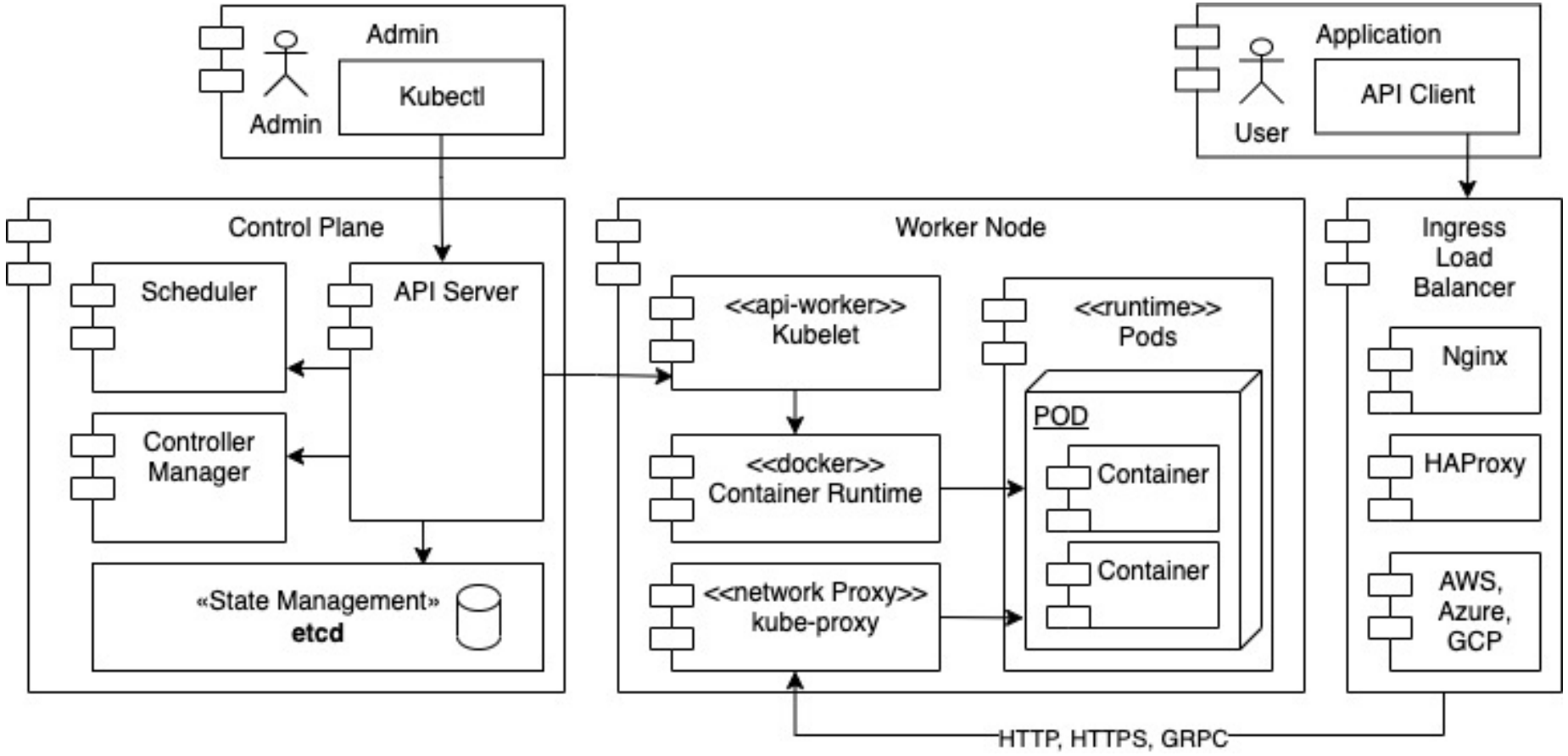


Another UML Example

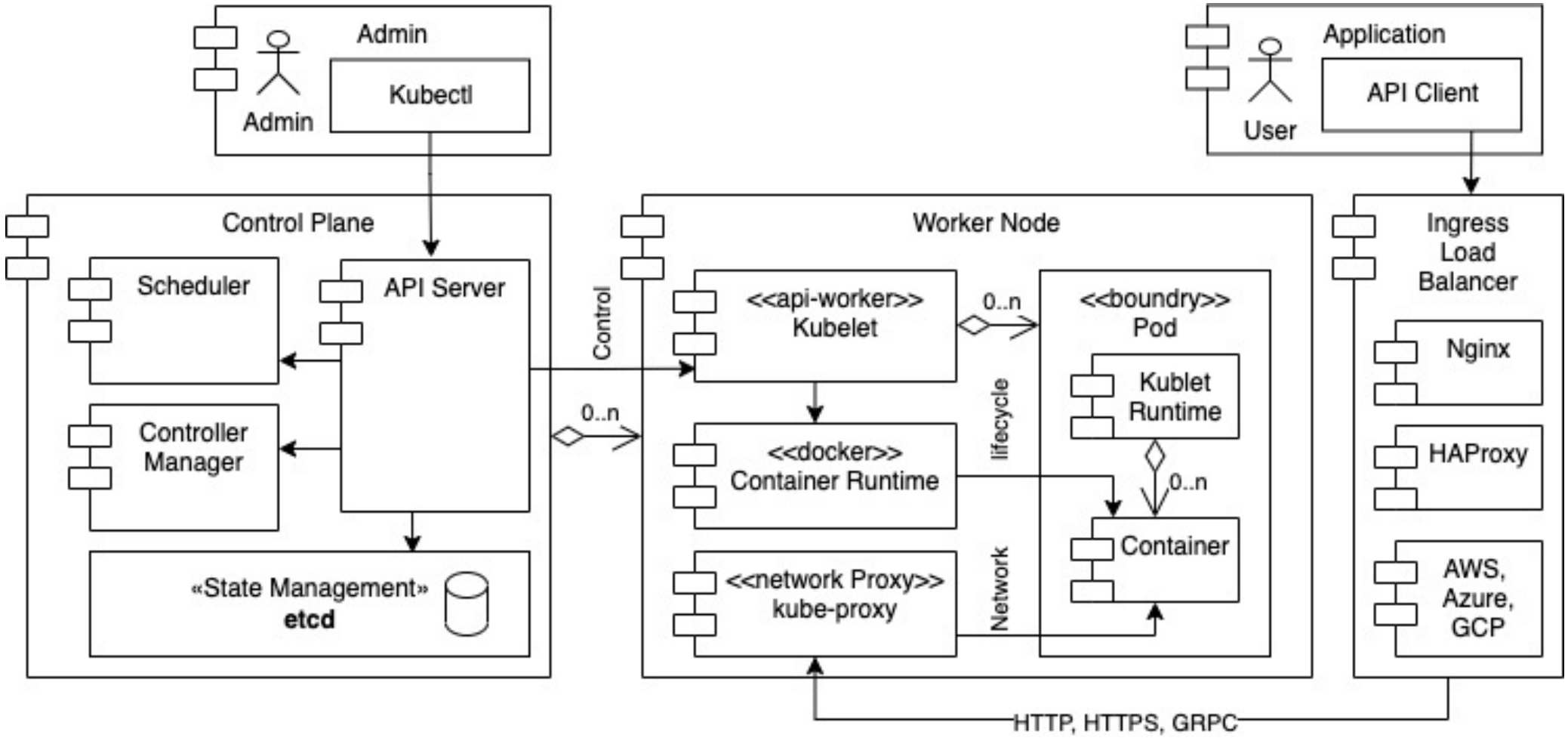
Adding Custom Stereotypes – Service, Service Connector, Service Interface



Another UML Example – Something more realistic Kubernetes v1

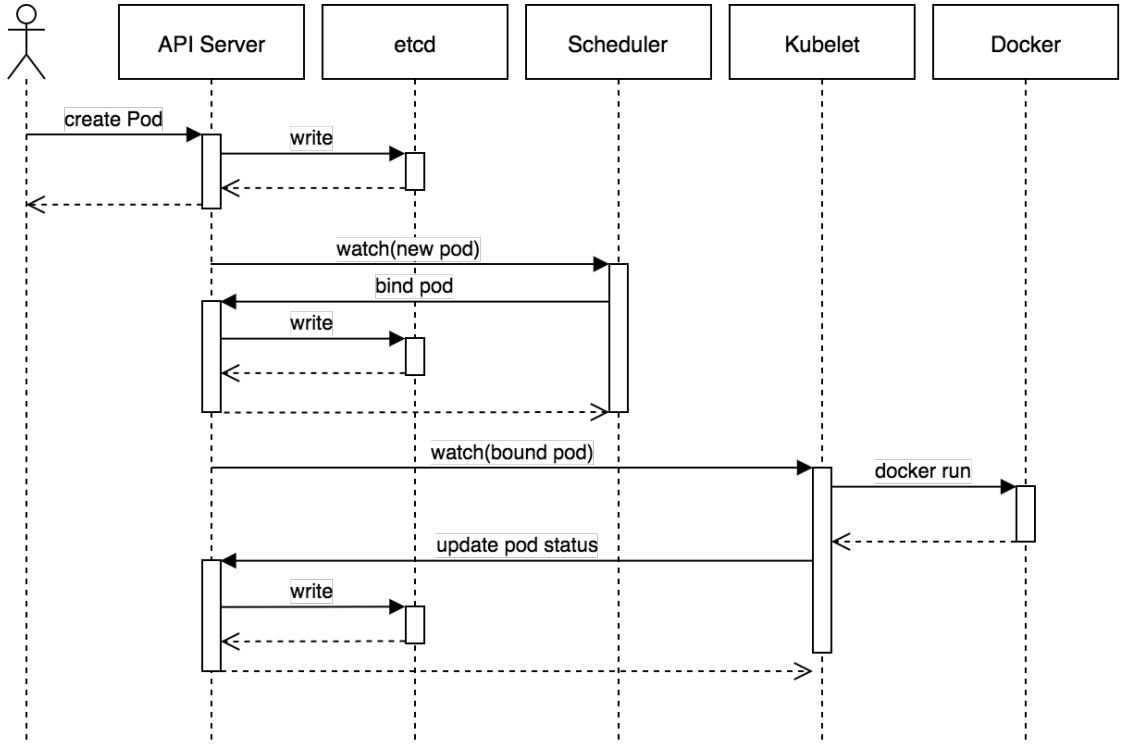


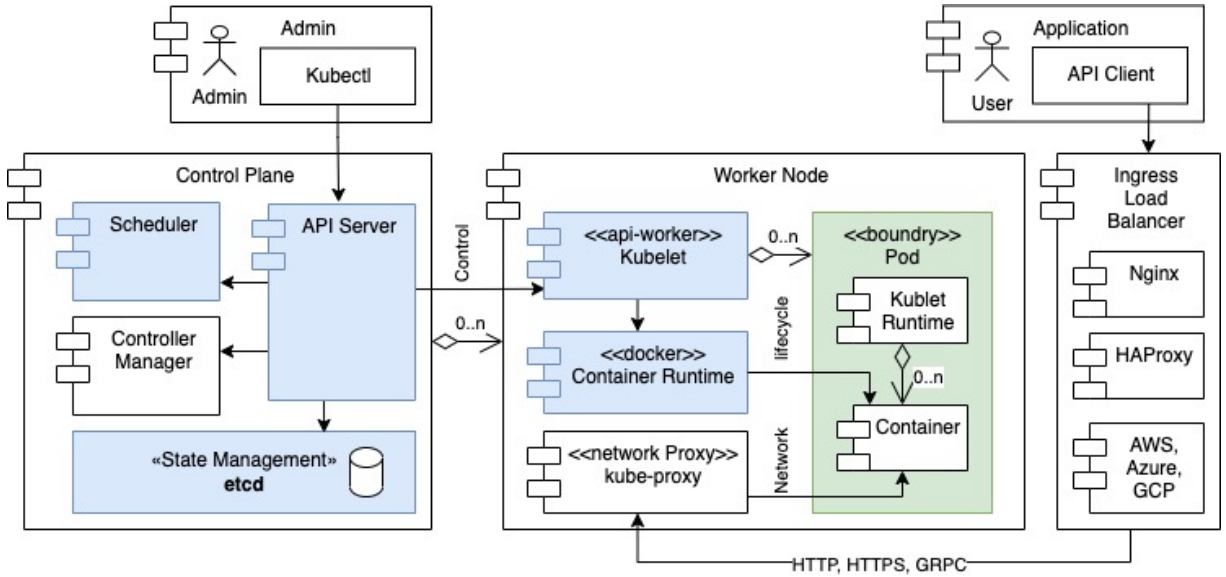
Another UML Example – Kubernetes v2



Notice the similarity to the first version of the model. Just some notation changes. How are they helpful to promote understanding of the architecture?

Kubernetes Pod Lifecycle – Behavioral Example



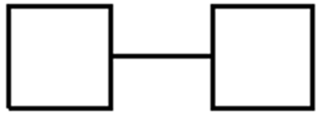


Don’t forget about modeling dynamic behavior, the how the system will behave at runtime is also very important

For UML to be used for Architecture Modeling, Team Standards must be established

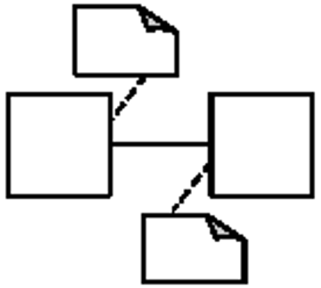
- With UML there are multiple ways the same abstractions can be represented

generic (line-and-box)



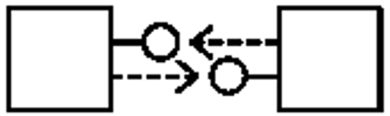
Option 1

ports as annotations

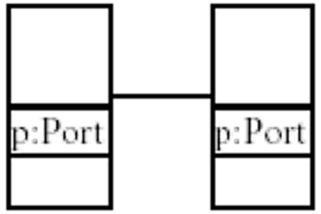


Option 2

ports as interfaces

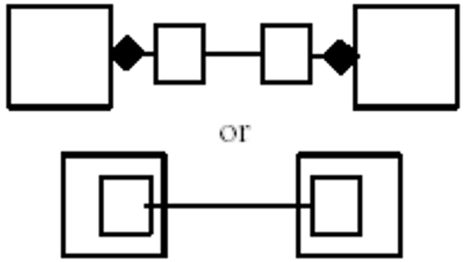


Option 3



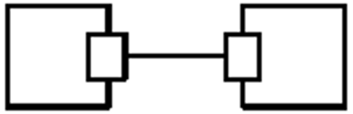
Option 4

ports as class attributes



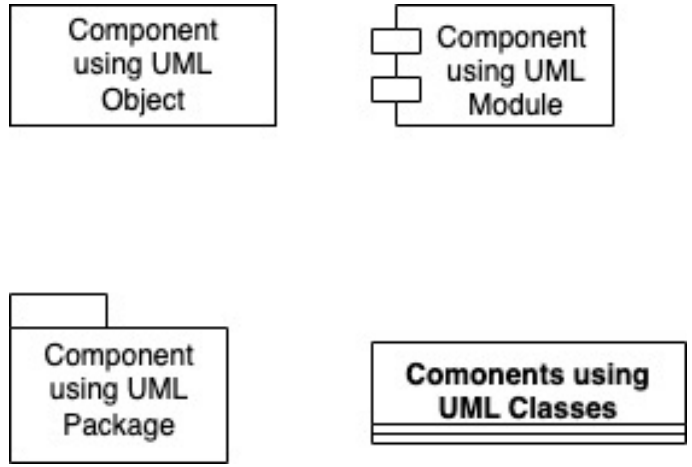
Option 5

ports as classes



Option 6

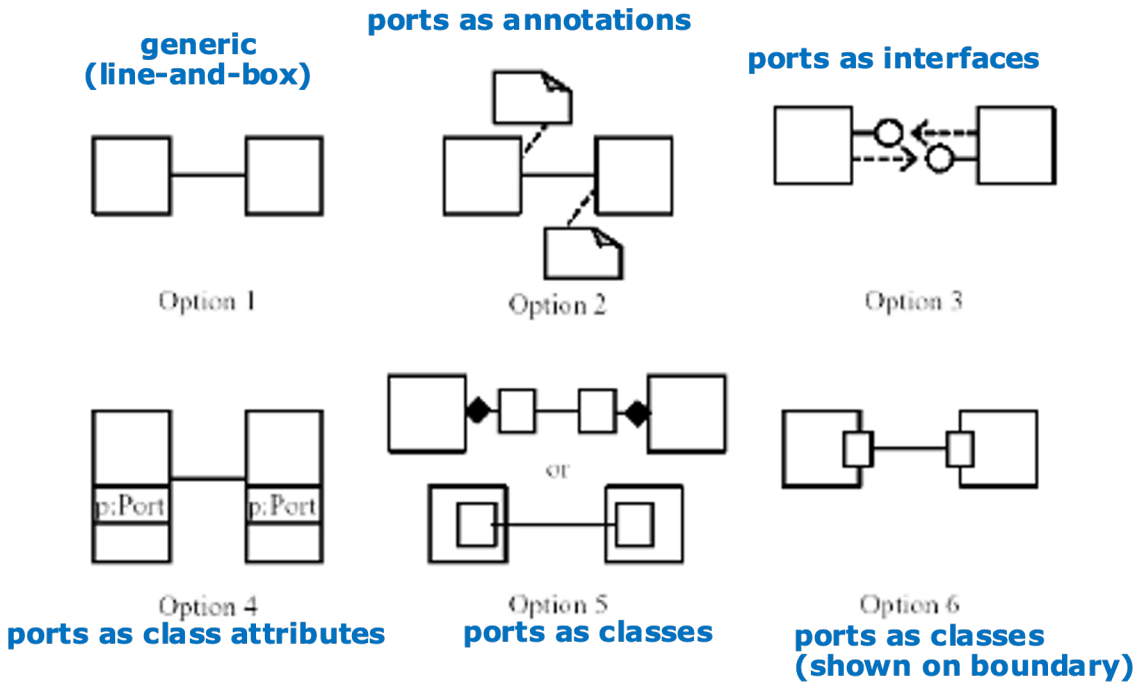
ports as classes (shown on boundary)

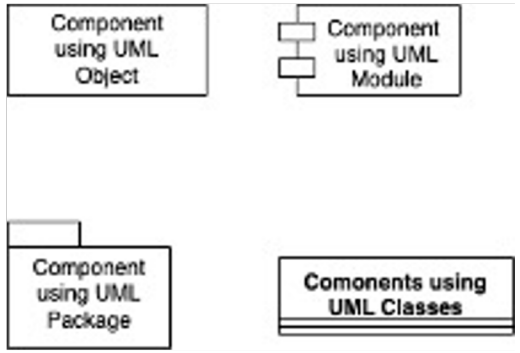


4 Different Ways to Represent Architecture Components

For UML to be used for Architecture Modeling, Team Standards must be established

- Need to agree on notational conventions
- Does this make UML better/worse that informal modeling techniques?





4 Different Ways to Represent Architecture Components

UML as a Modeling Language for Architecture

The Good

- UML is "somewhat-widely" known
- There is commercial tool support for UML
- UML is extensible

The Not-So Good

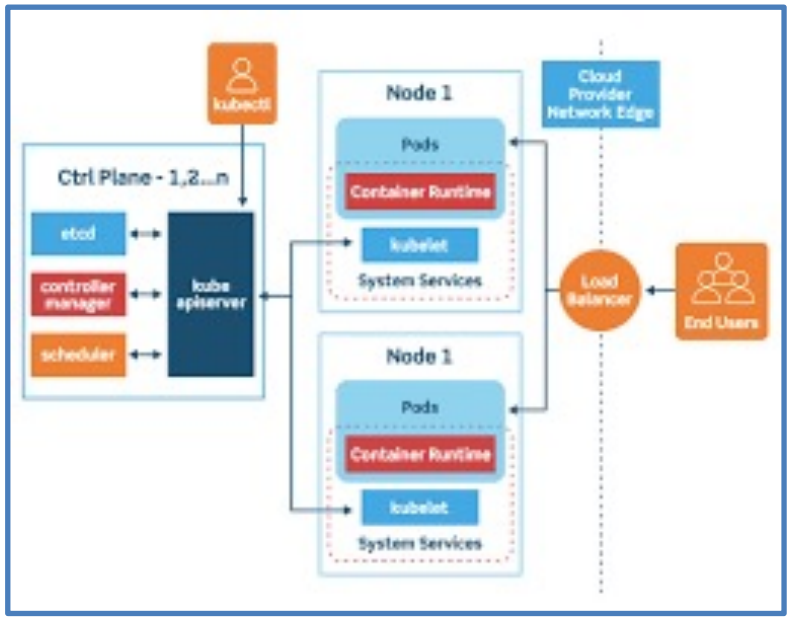
- UML is a language of design, and not architecture
  - Many of UML abstractions are really constructs that you see in object-oriented programming languages
- UML has many options to do basically the same thing, how to you get consistency?

Using informal lines and boxes to model architecture

- In this approach a diagraming tool such as Microsoft Visio, Draw.io or Omnigraffle is used to document the architecture views
- Tends to create the "best looking" views given that there is no limitation to the artwork that can be applied to the components and connectors.
- Probably the most popular approach to document architectures
- But there are challenges:
  - There is a lack of rigor over the component and connector vocabulary (Legends can help)
  - They tend to be pictures, and difficult to get value over managing the views in a model repository – for example – there is limited opportunity to share metadata between diagrams
  - Hard to deal with versioning and change – "Is this the latest view?"
  - Impossible to detect inconsistencies between models

So why are informal line-and-box drawings so popular for modeling architectures?

How does this….



Help Achieve any of this?

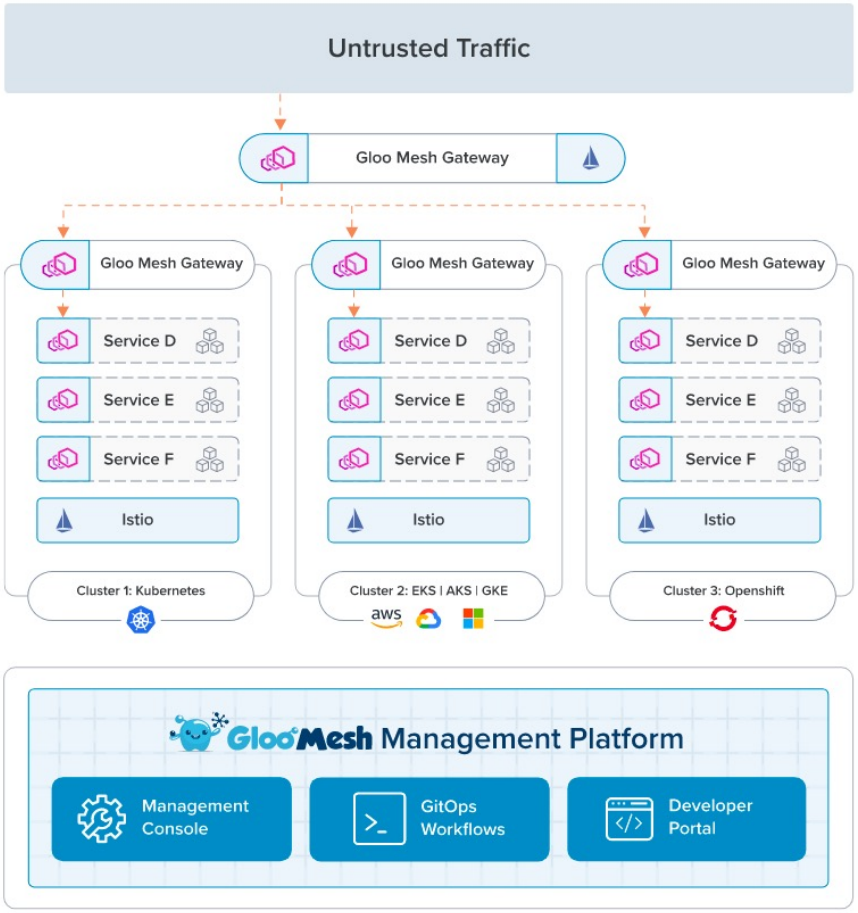
MOST IMPORTANT CONCEPT IN THIS LECTURE

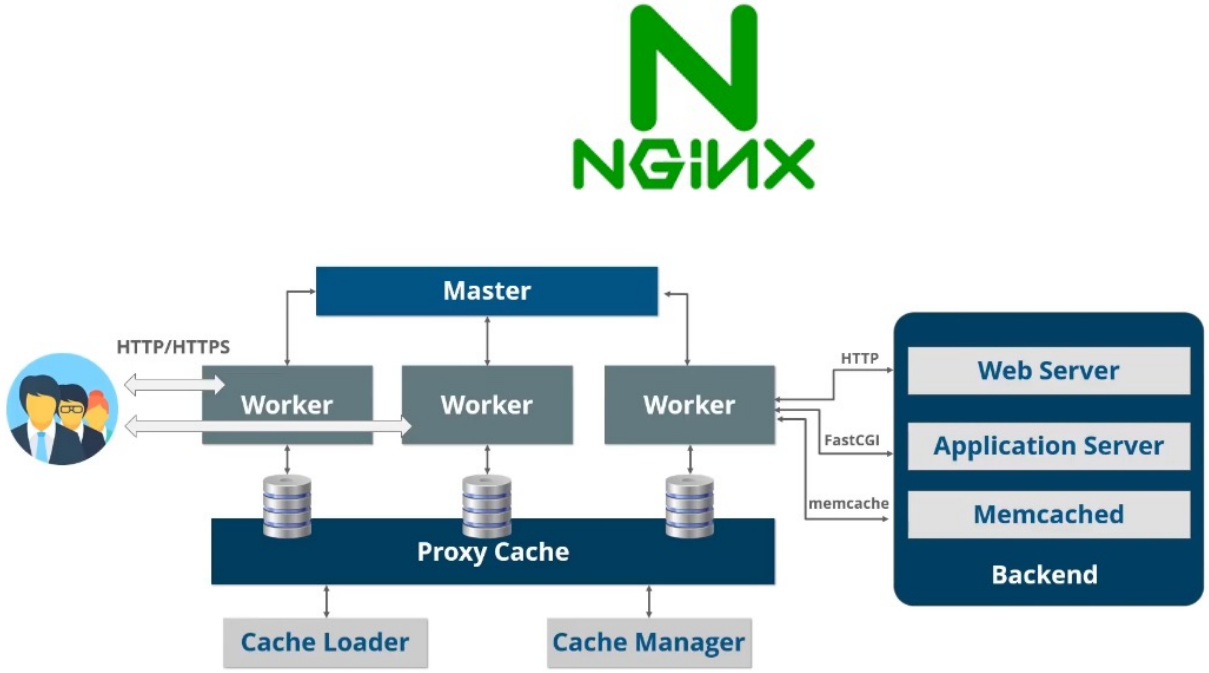
At the end of the day, a model is a quasi-useless artifact unless:

1. It helps **you as the architect, or your architecture team decompose the complexity of the problem domain**, or
2. Its **purposefully targeted at one or more key stakeholders** to facilitate alignment on:
   - Decision that have, or will need to be made on the solution to acheive its functional or non-functional requirements
   - Aspects around tradeoffs driven by constraint balancing that require agreement
   - Agreement related to technical debt that require socialization
   - Agreement around technical stack or deployment options that impact or constrain engineering staff
   - Alignment around which decisions need to be made when, even if those decisions have not been made yet.

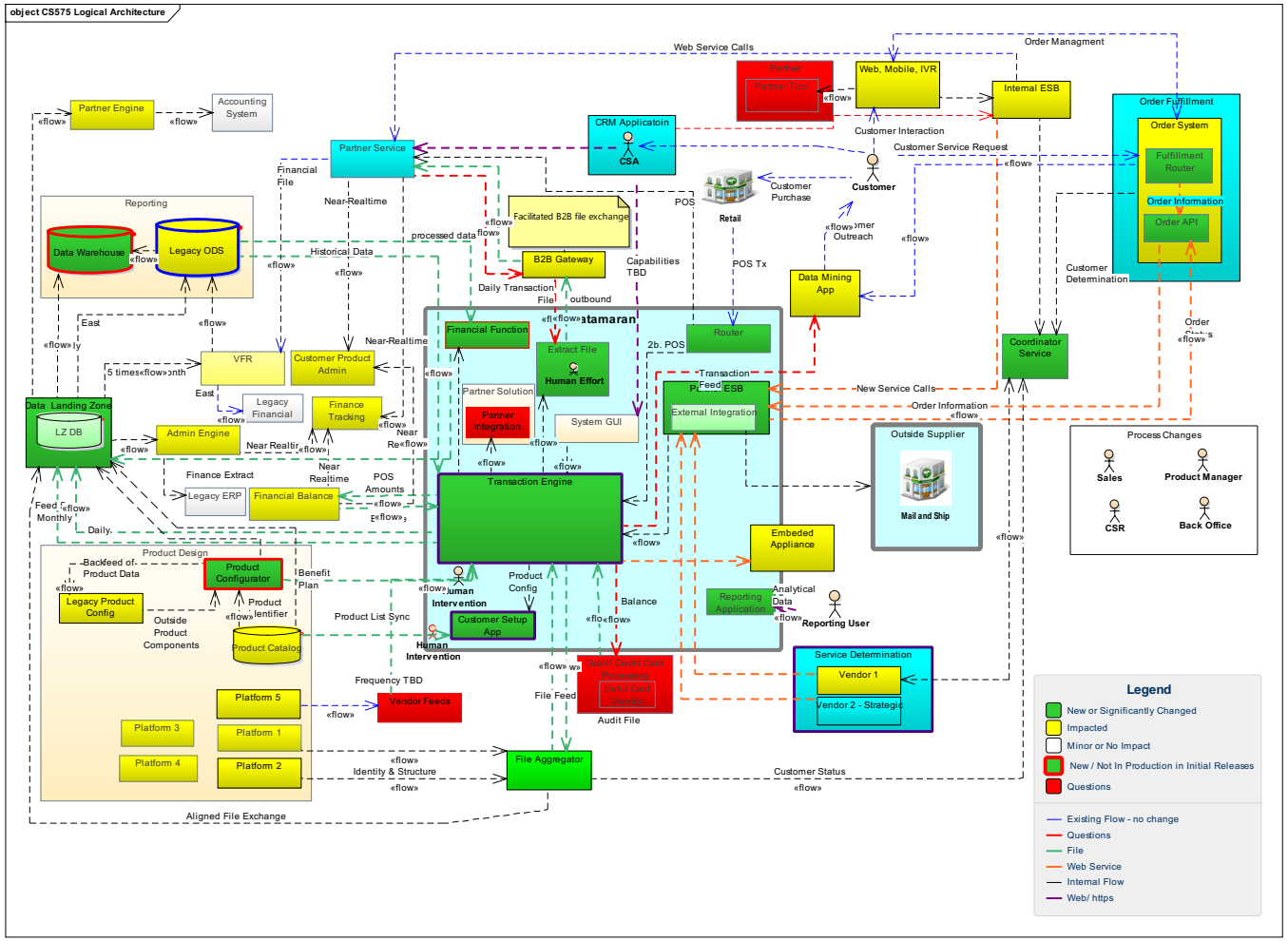
Unlike formal methods, informal lines and boxes can be heavy customized to meet the objective of the model. That said, its basically just drawing and care must be placed on the objective of the model, or else it will end up being quasi-useless architecture art

Informal Line and Box Examples – "Marketecture"





Informal Line and Box Examples – Architecture



The key objective of this architecture model wasn't to really show anything of technical architecture significance…

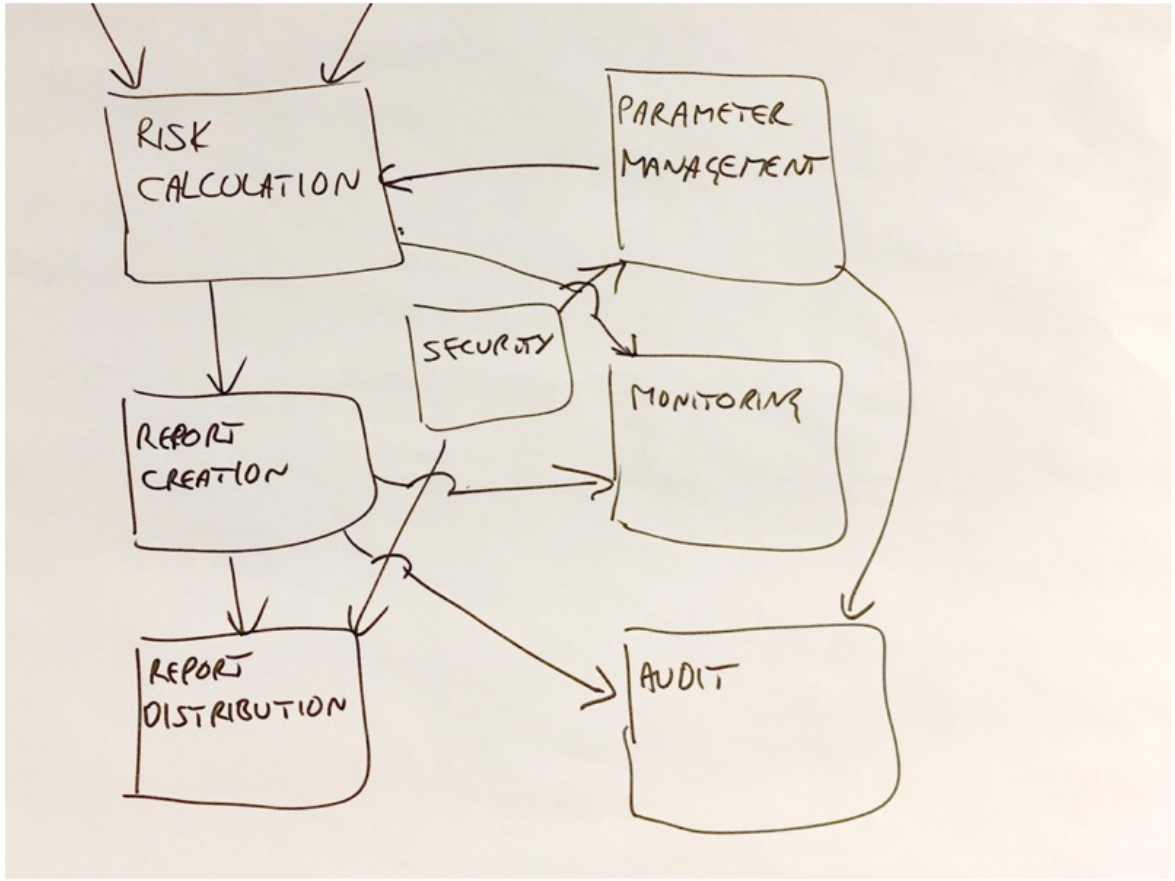
Its real objective was to to highlight to executive management the size of the impact of implementing this solution.

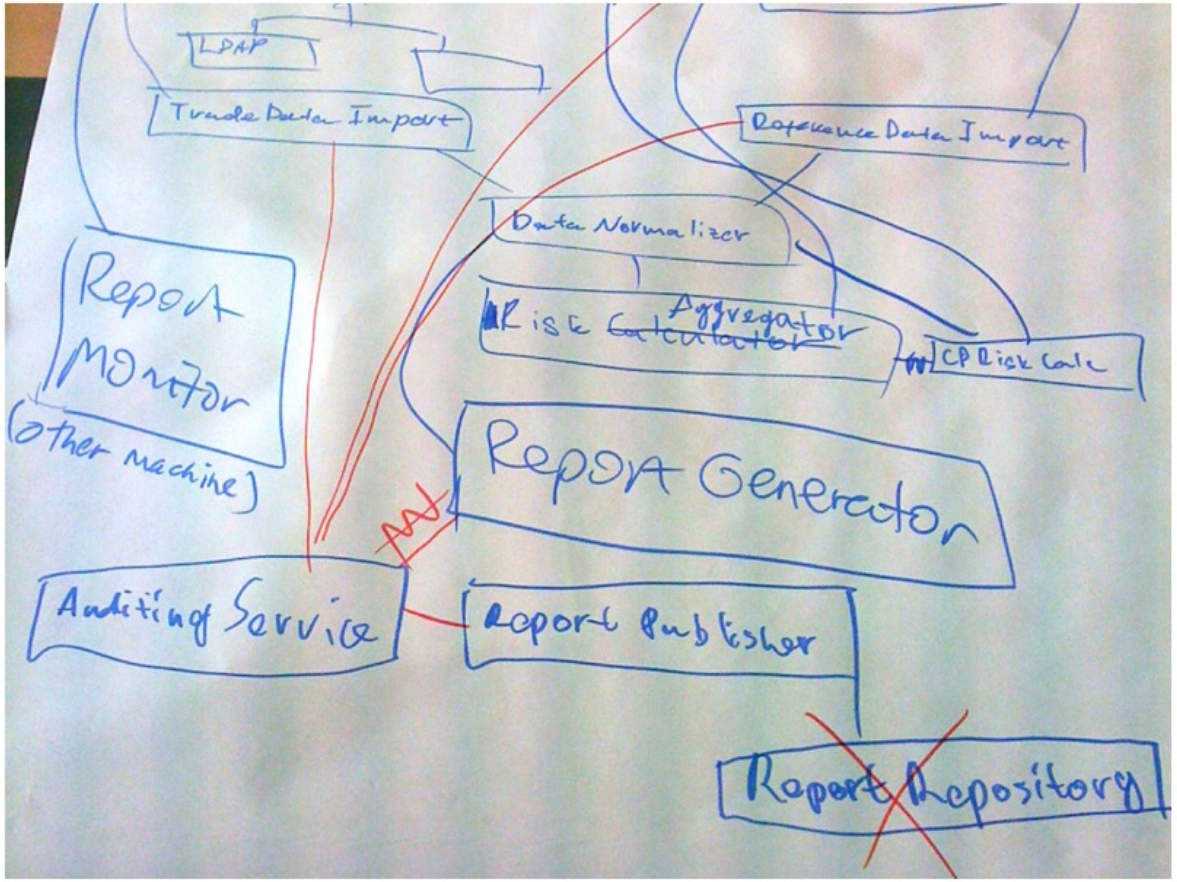
The key to this model is the legend to highlight what is new, changed or deprecated.

This model also uses other techniques such as the size of the boxes to highlight relative magnitude

Terrible model to convey architecture, great model to convey to a stakeholder why what we are doing is complex and expensive

Informal Line and Box (Bad) Examples

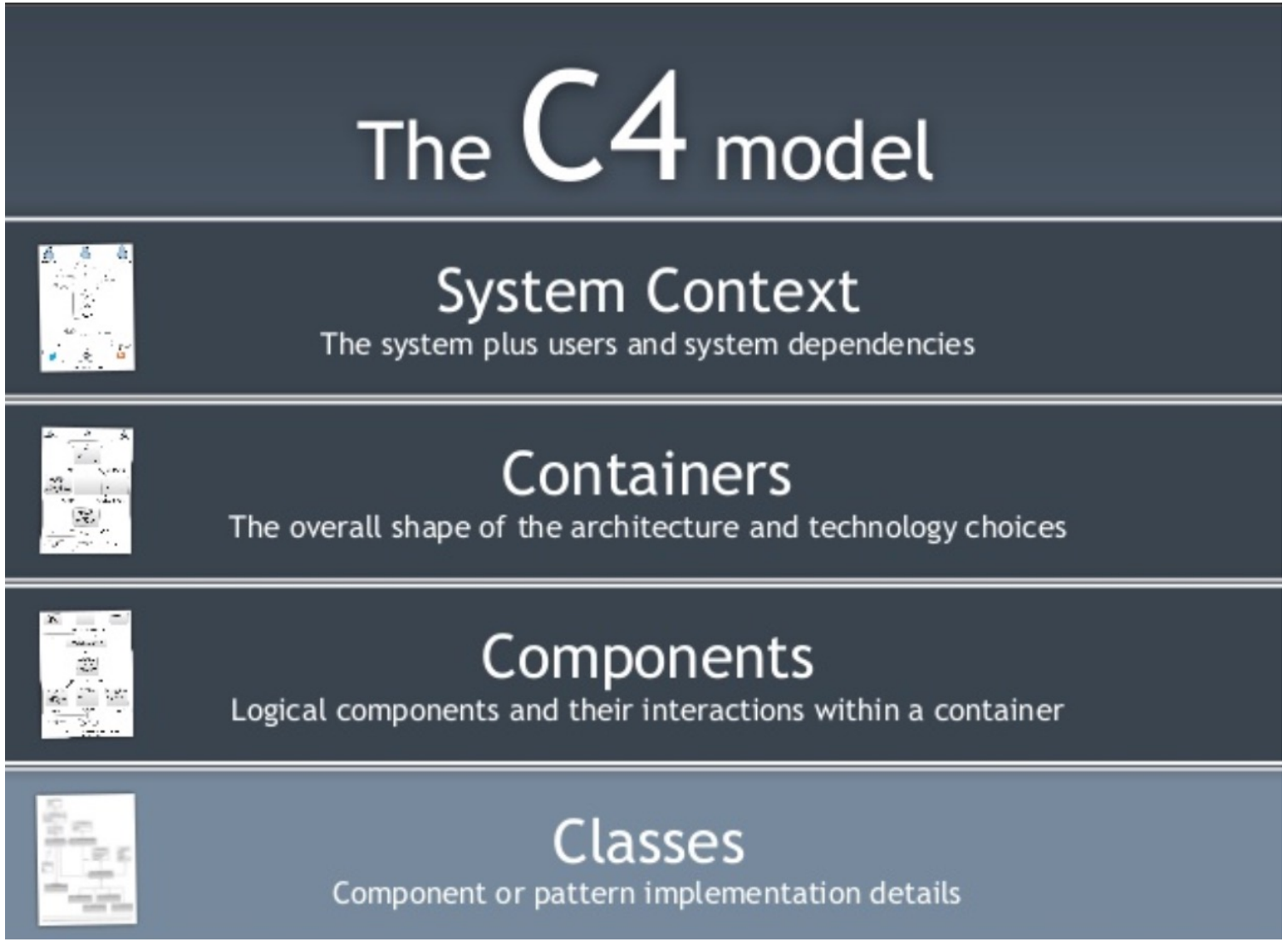




Summary: Using informal lines and boxes to model architecture

- Clearly the most popular approached used to model architectures
- Have to very clearly focus on the intended value or impact of the model at the **risk of it becoming just artwork**
- Remember that things that help improve understanding should be incorporated into the model:
  - Legend to highlight the lines and the boxes meaning or any special color coding
  - Callout text boxes, or annotate important areas of the model with callouts like (1) and then supplement with a textual description.
  - Avoid distribution over email, or via shared drives. We have found value creating and meaning these artifacts in shared wikis like Confluence or via GitDocs.
  - Make sure that important diagrams are tagged so that they can be searched for and located

Finding the balance - the C4 model

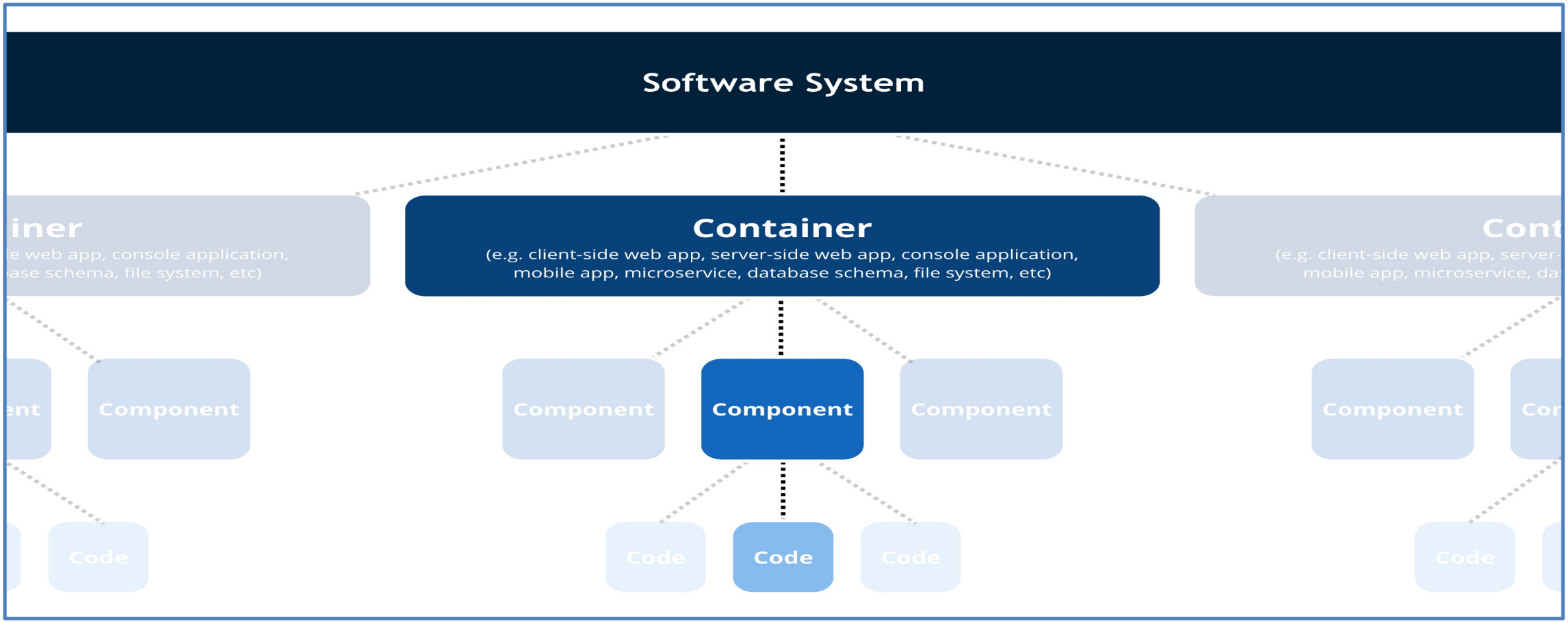


https://c4model.com/

https://simonbrown.je/

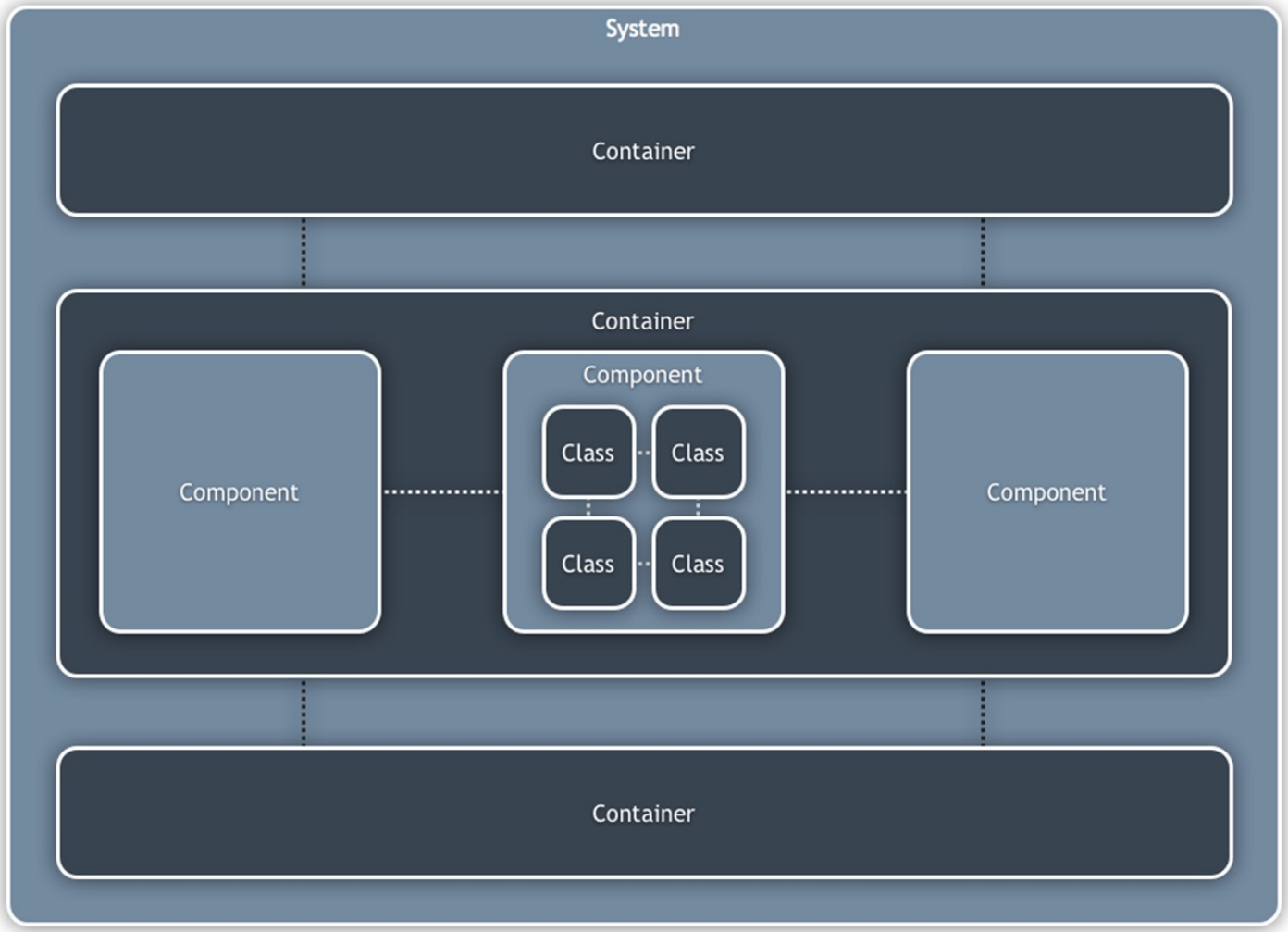
Material on C4 is derived from the many online resources created by its inventor – Simon Brown

C4 imposes an opinionated structure on modeling

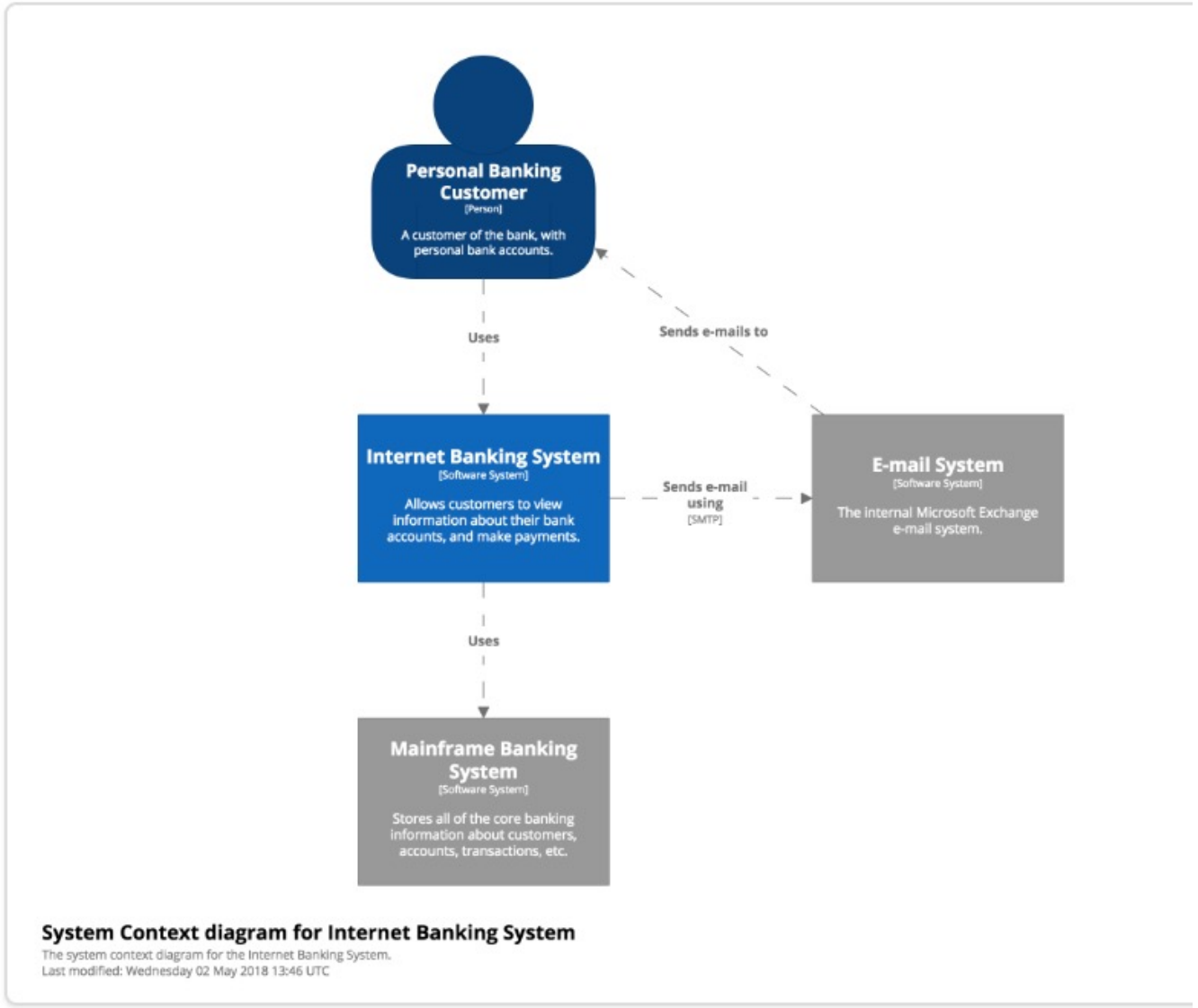


*A **software system** is made up of one or more **containers** (web applications, mobile apps, desktop applications, databases, file systems, etc), each of which contains one or more **components**, which in turn are implemented by one or more **code elements** (e.g. classes, interfaces, objects, functions, etc).*

The key attributes in C4 are related to each other



C4 - Context

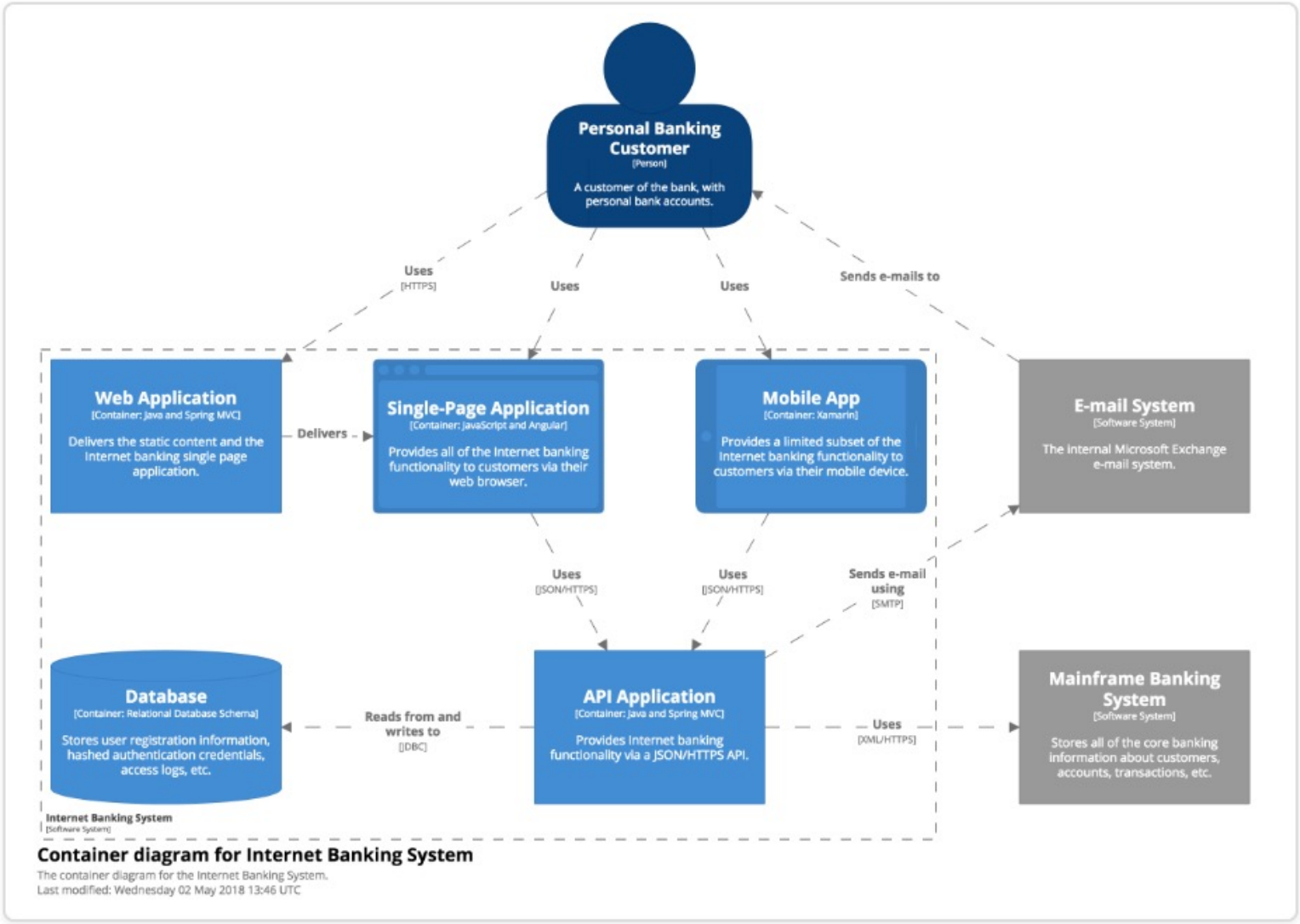


System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Last modified: Wednesday 02 May 2018 13:46 UTC

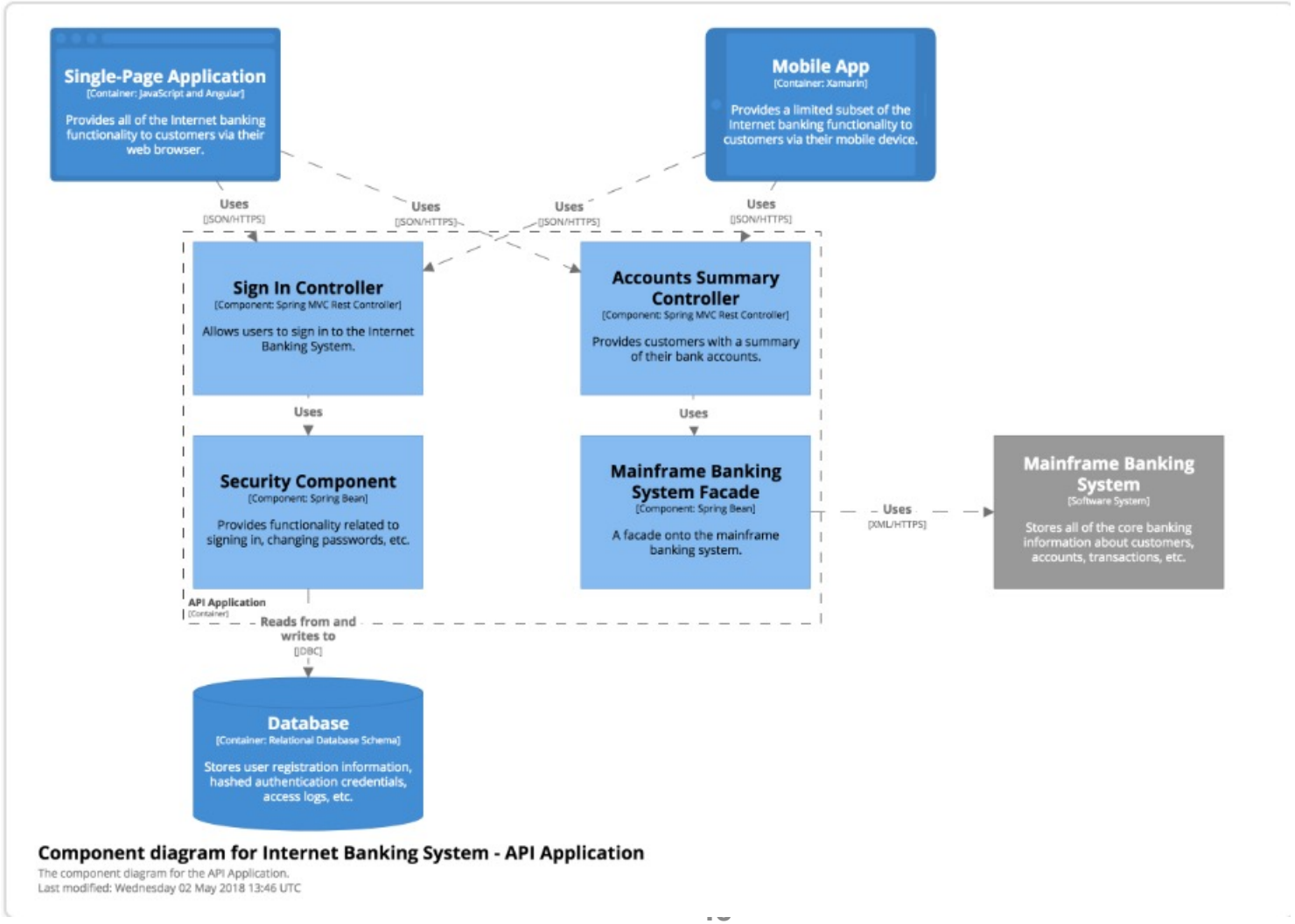
Ref: Simon Brown
https://c4model.com/

C4 - Container



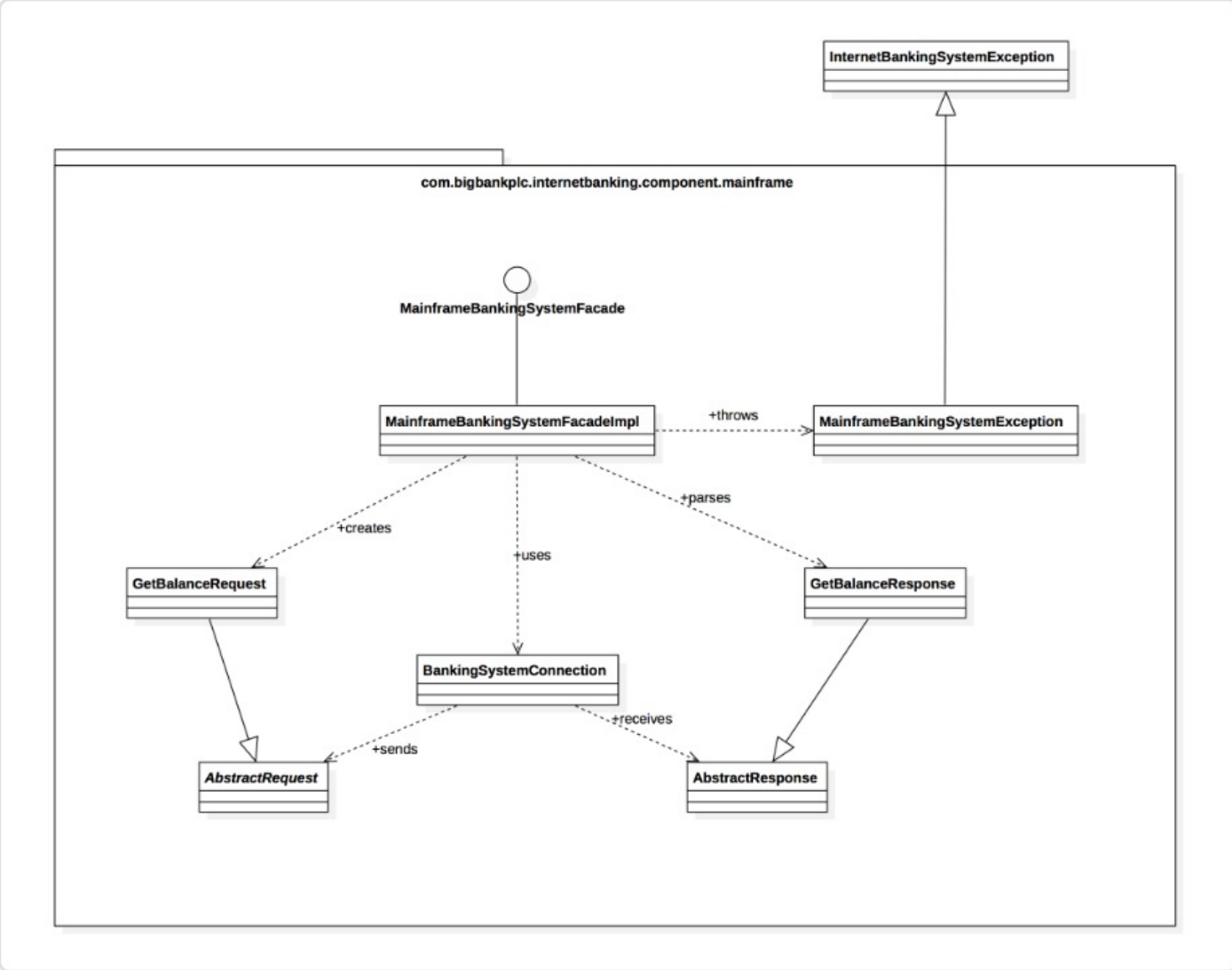
Ref: Simon Brown
https://c4model.com/

C4 - Components



Ref: Simon Brown
https://c4model.com/

C4 – Code (Optional)



Ref: Simon Brown
https://c4model.com/

Architecture Decision Records

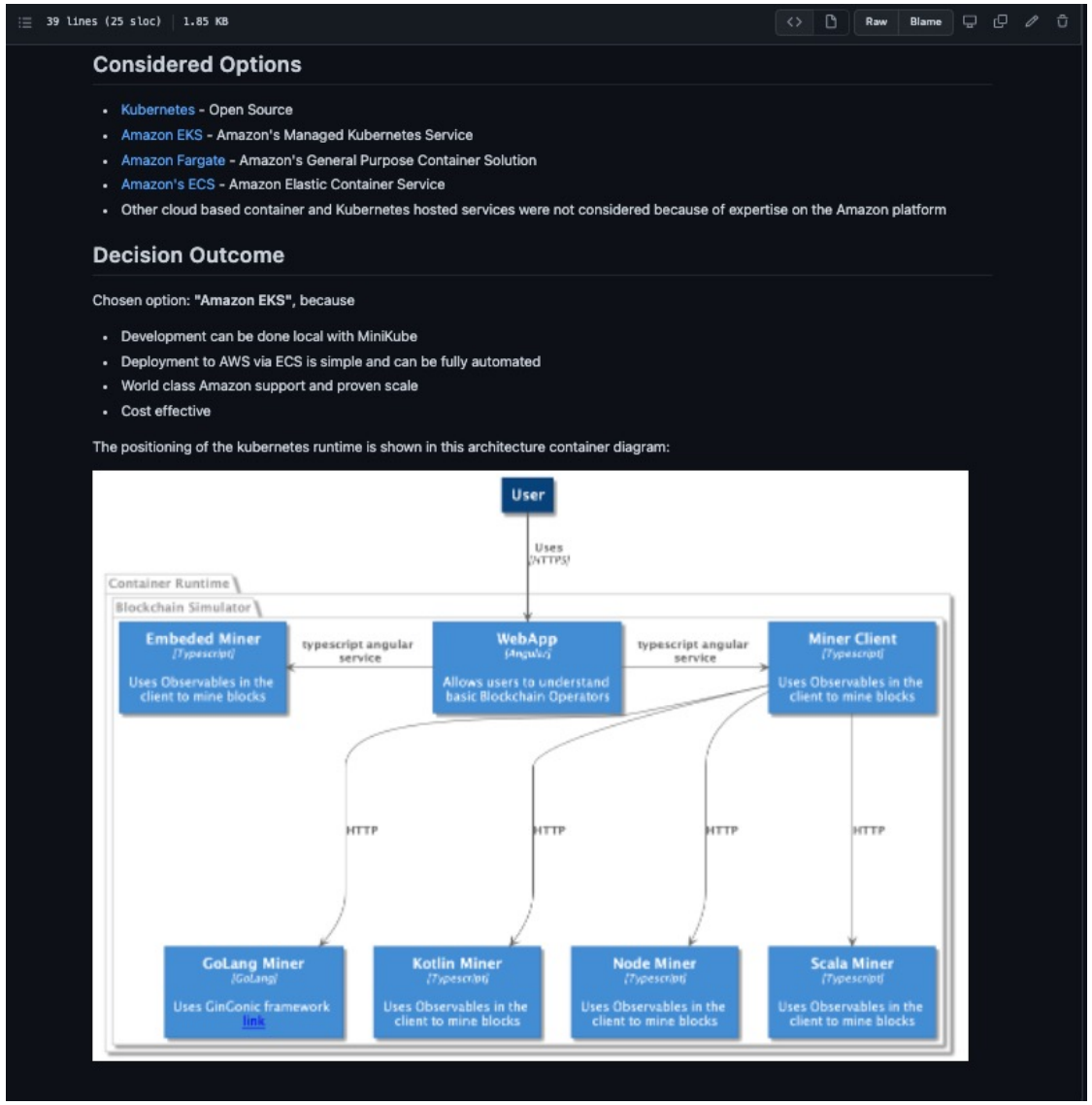
http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions
Seminal Reference (2011) – Michael Nygard - https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions

- One of the best ways to document an architecture is to think about all of the decisions that need to be made
  - Not all decisions need to be made at once
  - Not all decisions need solutions prior to a project beginning
  - We constantly learn new things, thus, decisions may change
- Also, keeping decisions close to the code, with collaboration capabilities is desirable: https://github.com/ArchitectingSoftware/CS575-SoftwareDesign/tree/master/Lectures/Architecture-Documentation-Demo/doc
- ADR
  - Numbered monotonically, no number ever repeats
  - Clear title, status and keywords for search
  - Context and Problem Statement
  - Considered Options
  - Decision Outcome
  - Supporting architecture diagrams

DEMO AT:
https://github.com/ArchitectingSoftware/CS575-SoftwareDesign/tree/master/Lectures/Architecture-Documentation-Demo/doc

Architecture Decision Records

- Think of ADRs as "Architecture as Code"
  - Lend themselves very well to be managed and versioned in git
  - Indexable/ searchable
  - Ephemeral – Records history of changes over time as new things are learned
- ADRs complement visual models – they do not replace them.
  - Records alternatives and models help support the rationale



More on C4 Styling

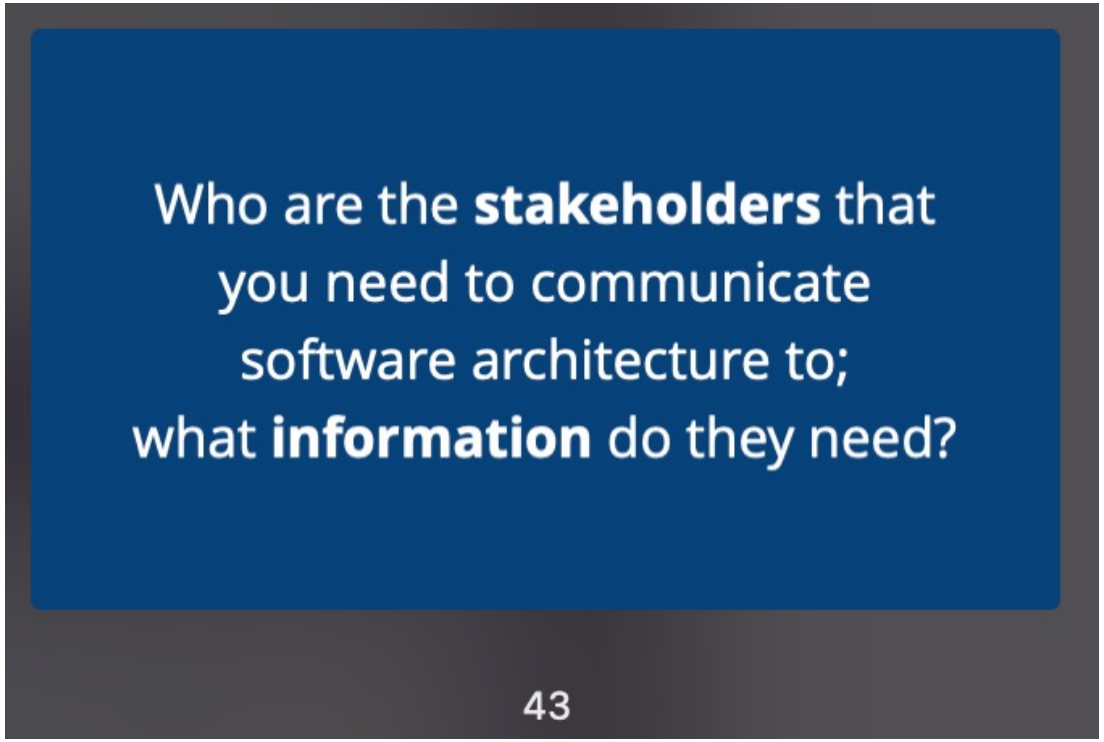
See:

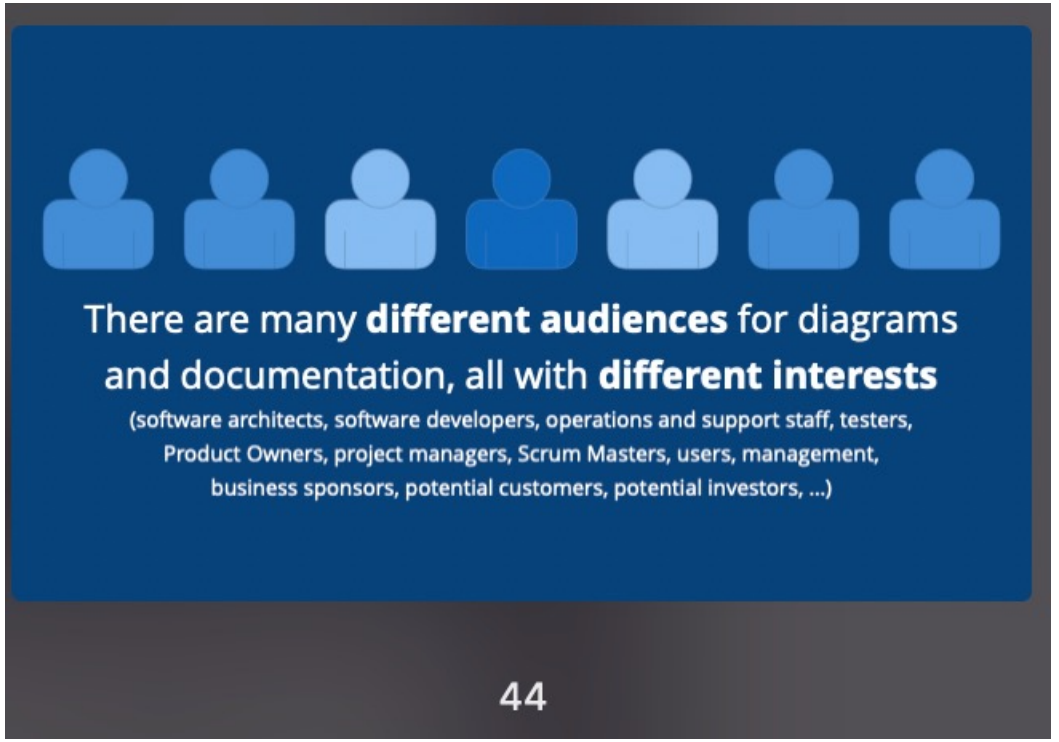
https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf

Lots of great suggestions on C4 best practices from Simon Brown himself

Some callouts from the attachment

https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf

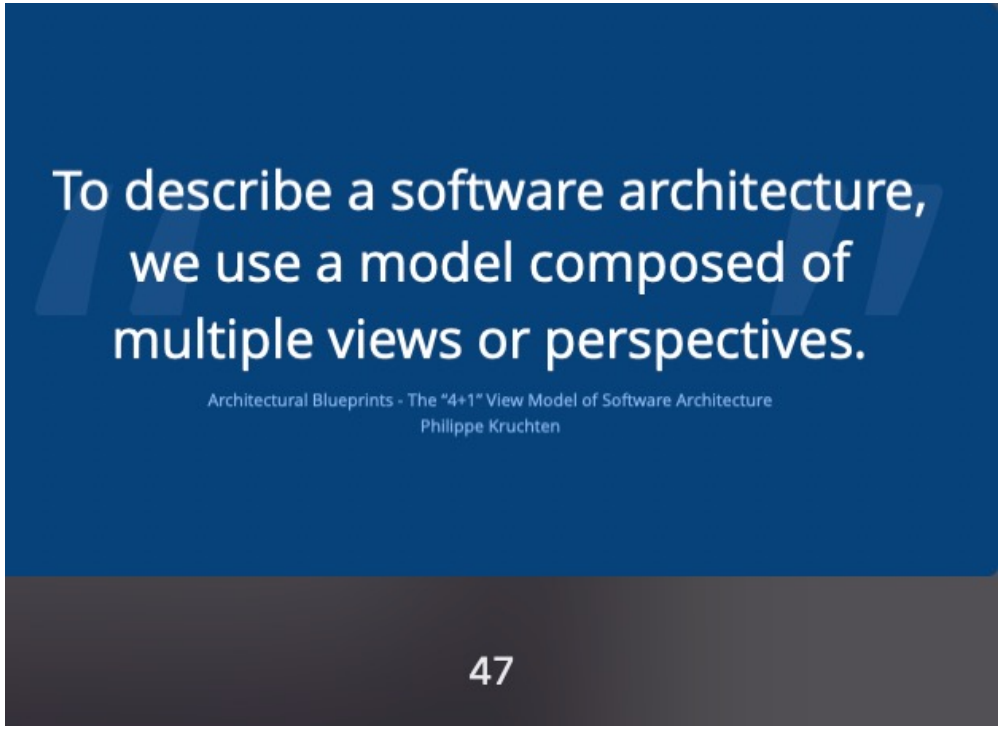


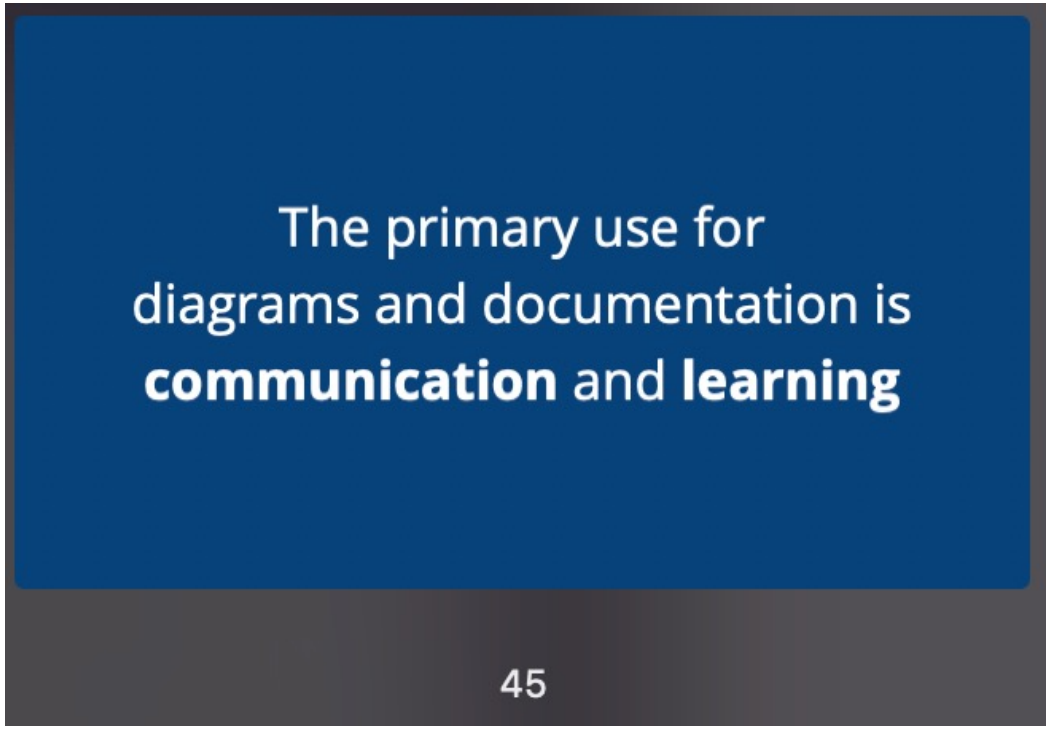


I have mentioned this MANY times

Some callouts from the attachment

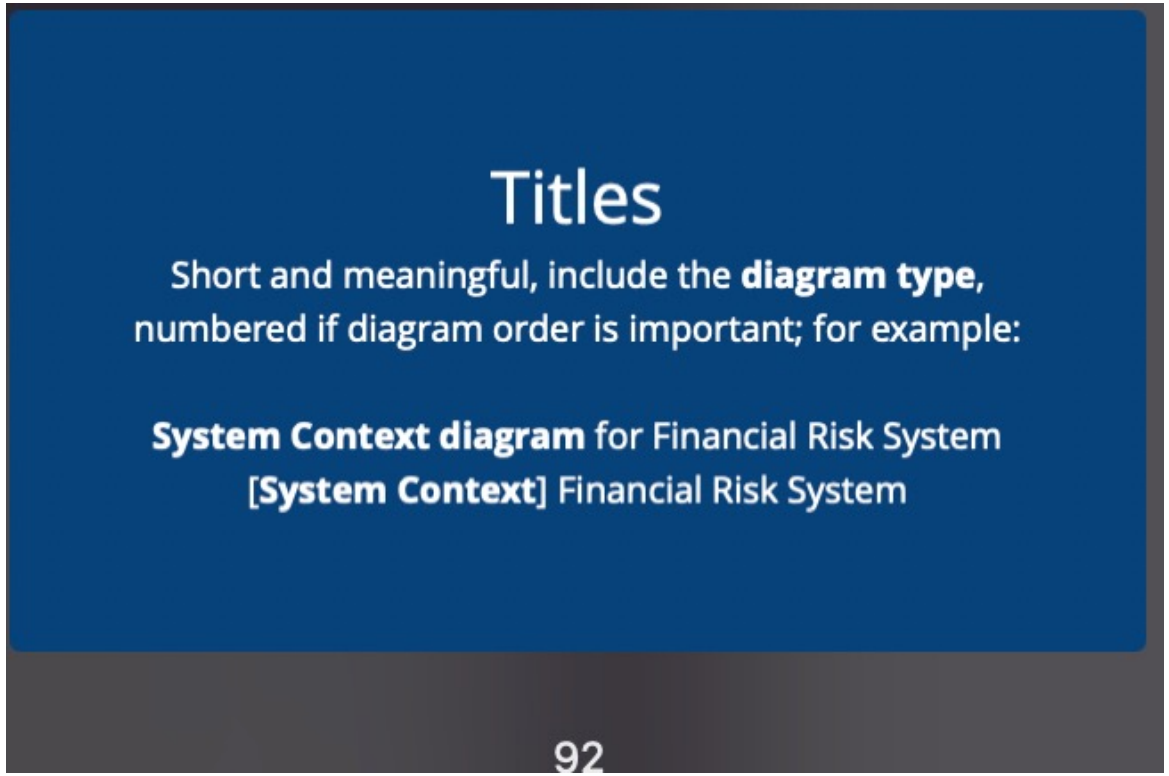
https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf

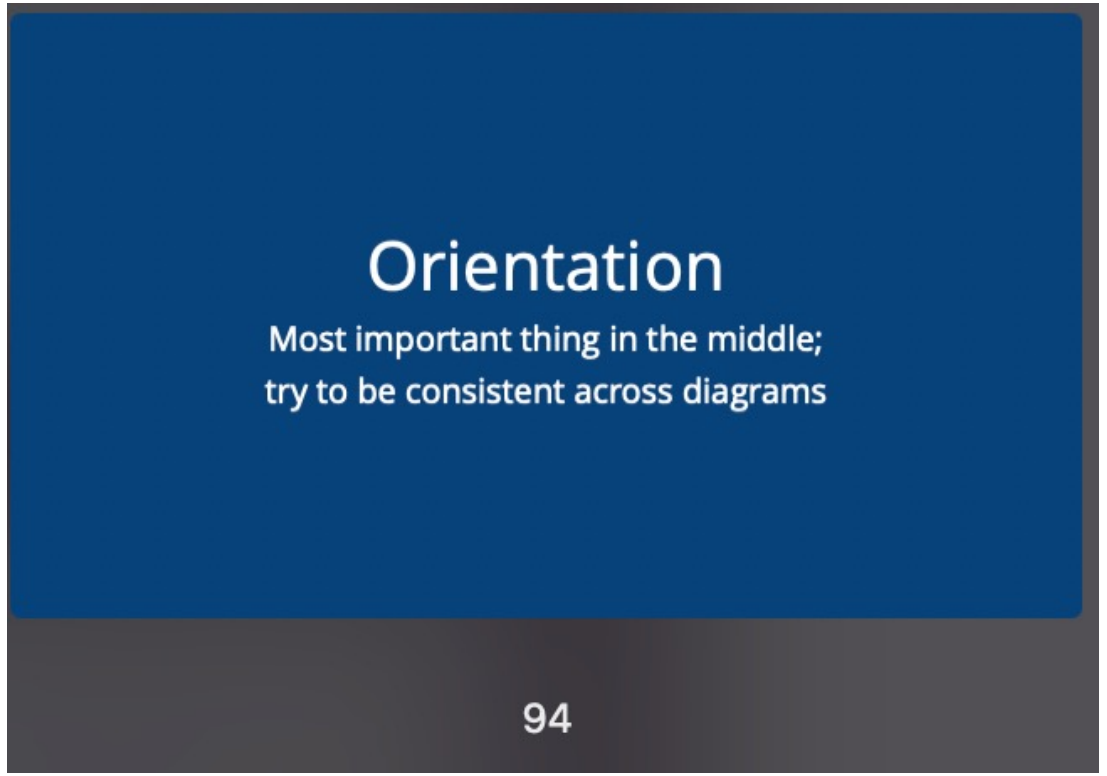




Some callouts from the attachment

https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf





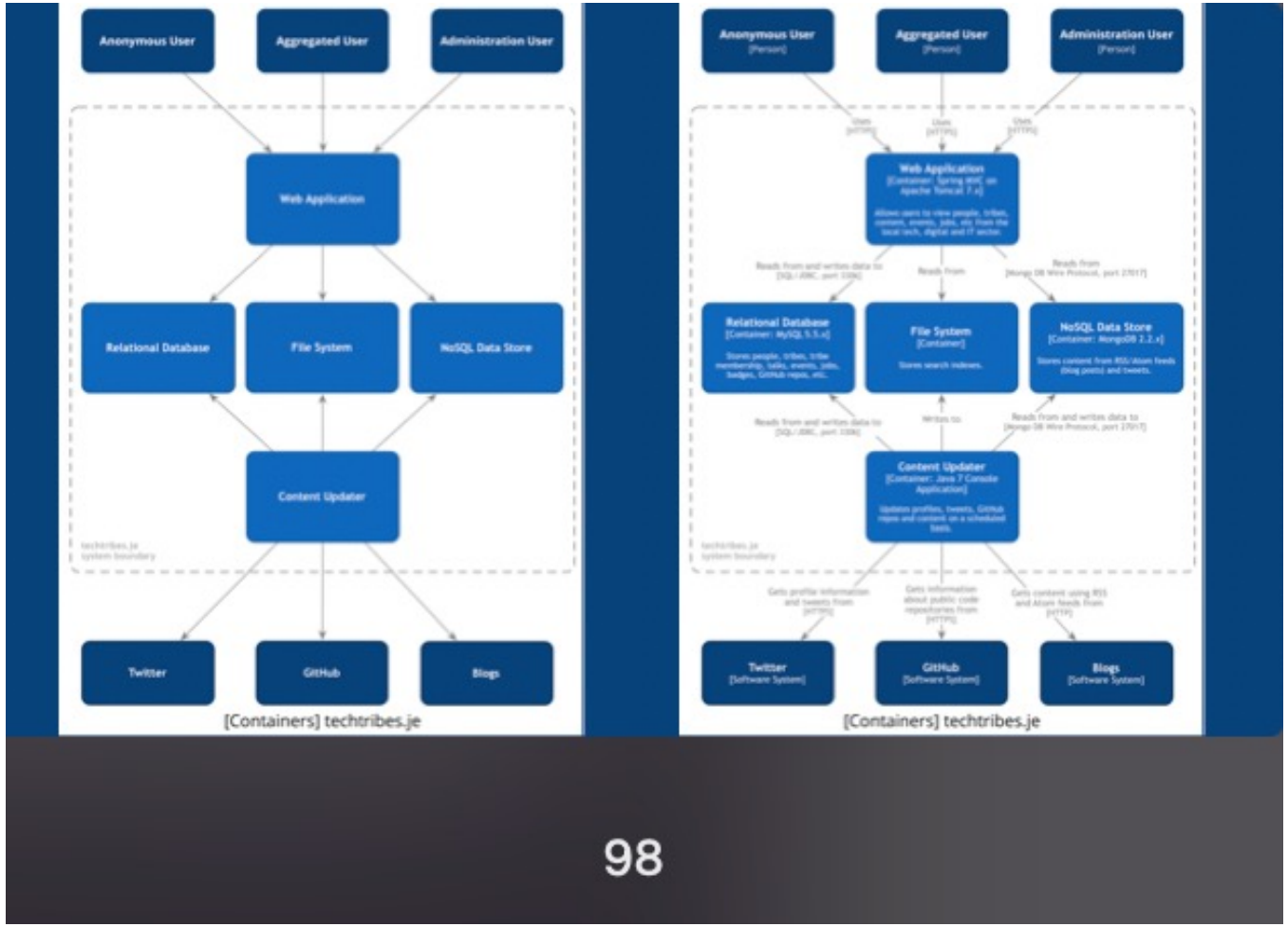
Some callouts from the attachment

https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf

Elements

Start with simple boxes containing the element name, type, technology (if appropriate) and a description/responsibilities

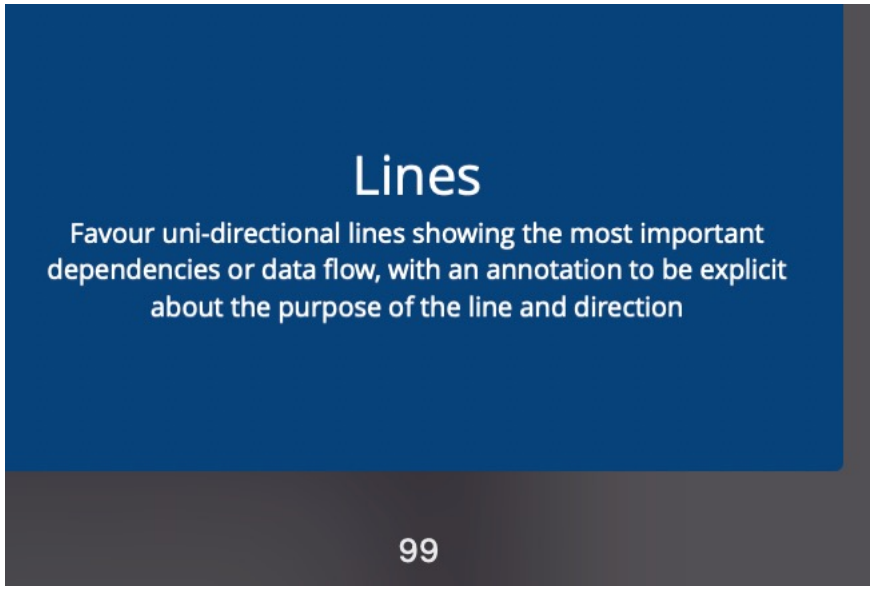
96



98

Some callouts from the attachment

https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf



Slides 99-106 Have Great Guidence

Some callouts from the attachment

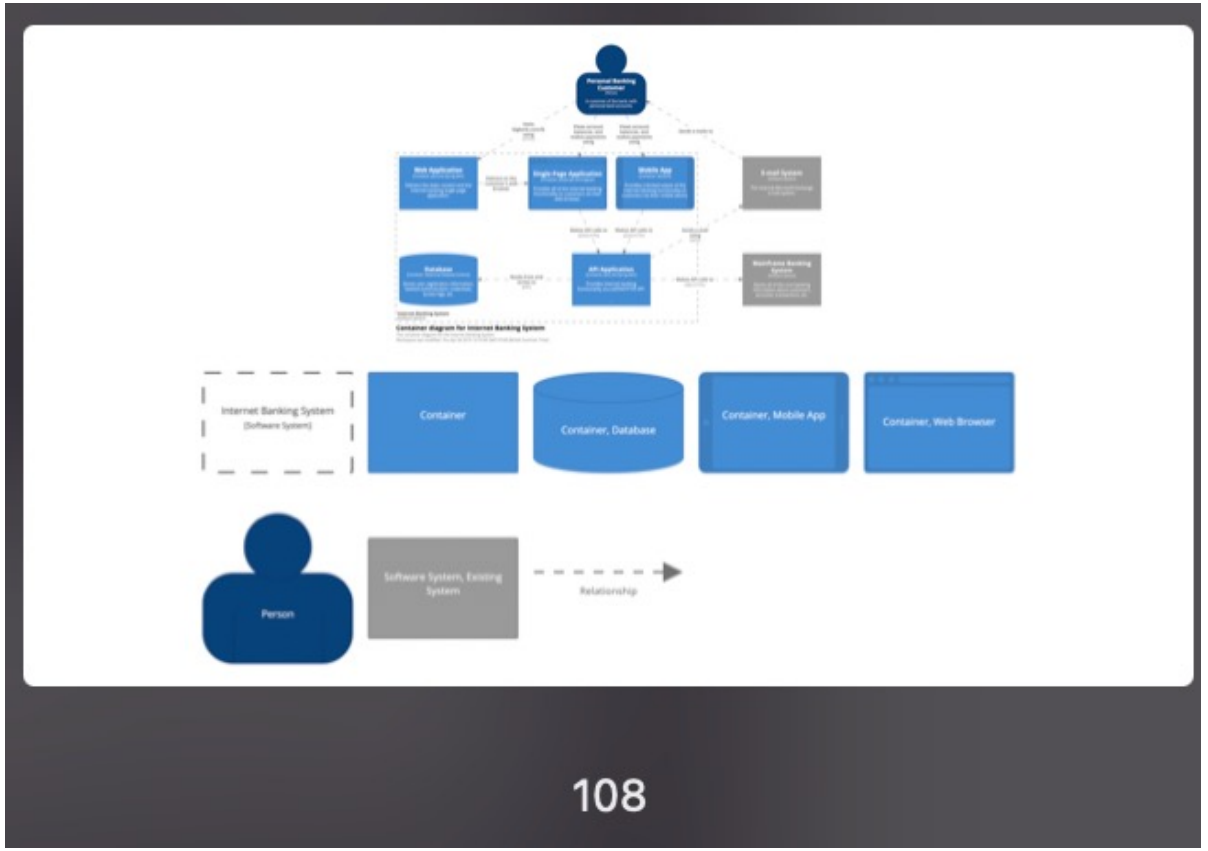
https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf

Key/legend

Explain shapes, line styles, colours, borders, acronyms, etc

... even if your notation seems obvious!

107



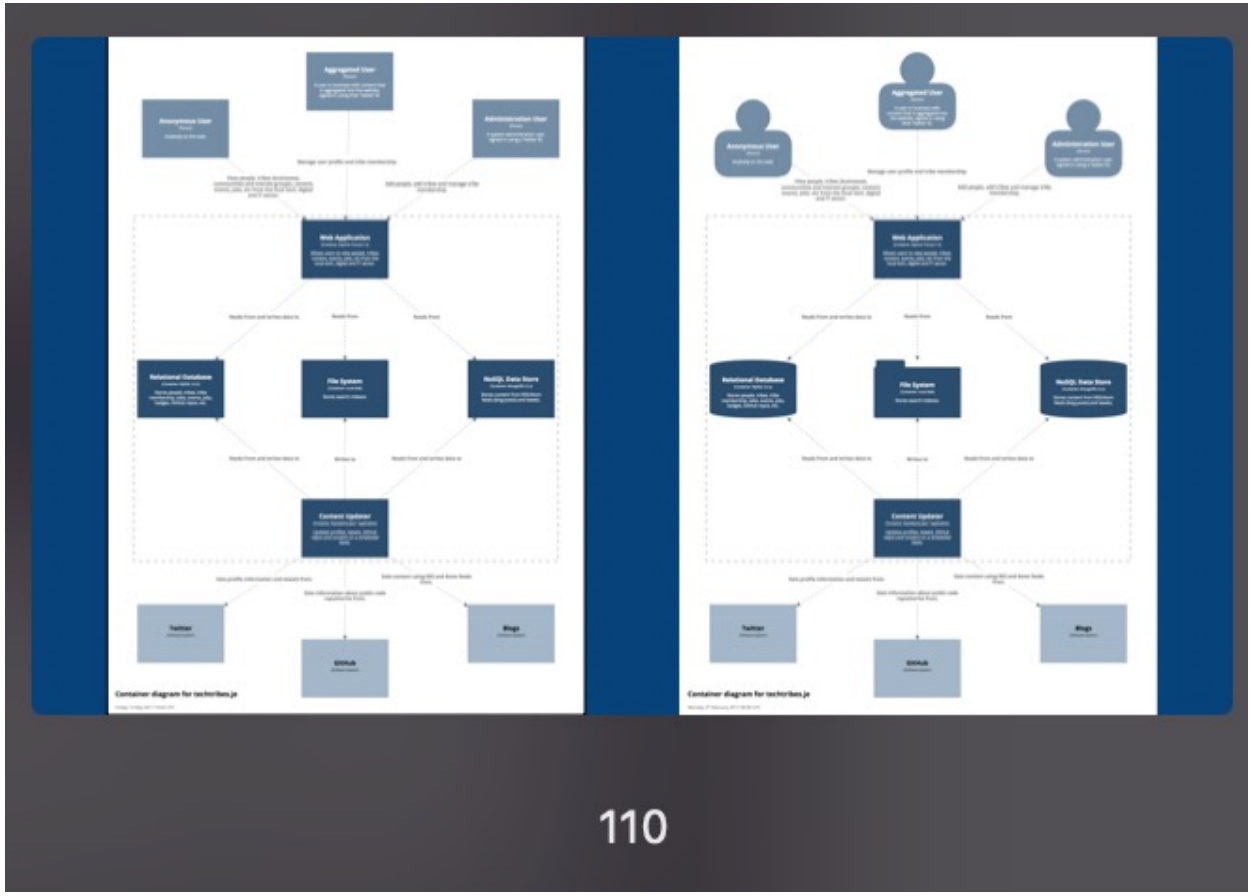
108

Some callouts from the attachment

https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf

Use shape, colour and size to **complement** a diagram that already makes sense

109



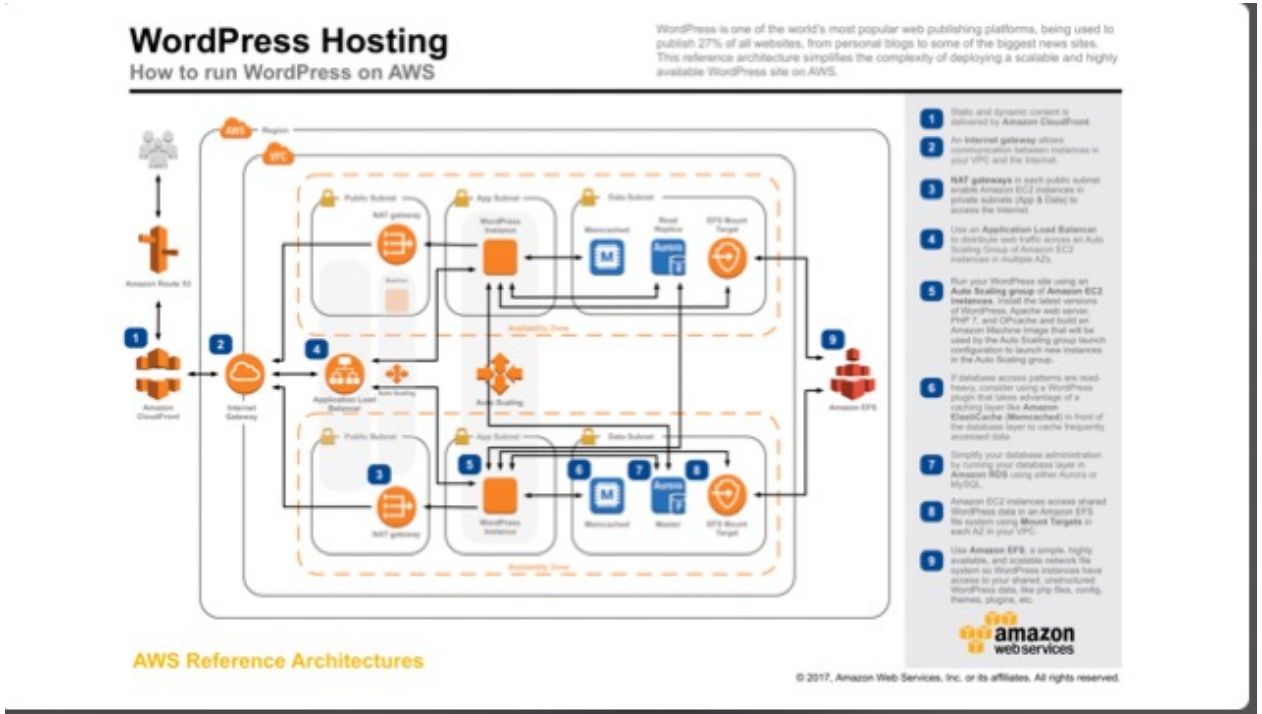
110

Some callouts from the attachment

https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf

Be careful with **icons**

111



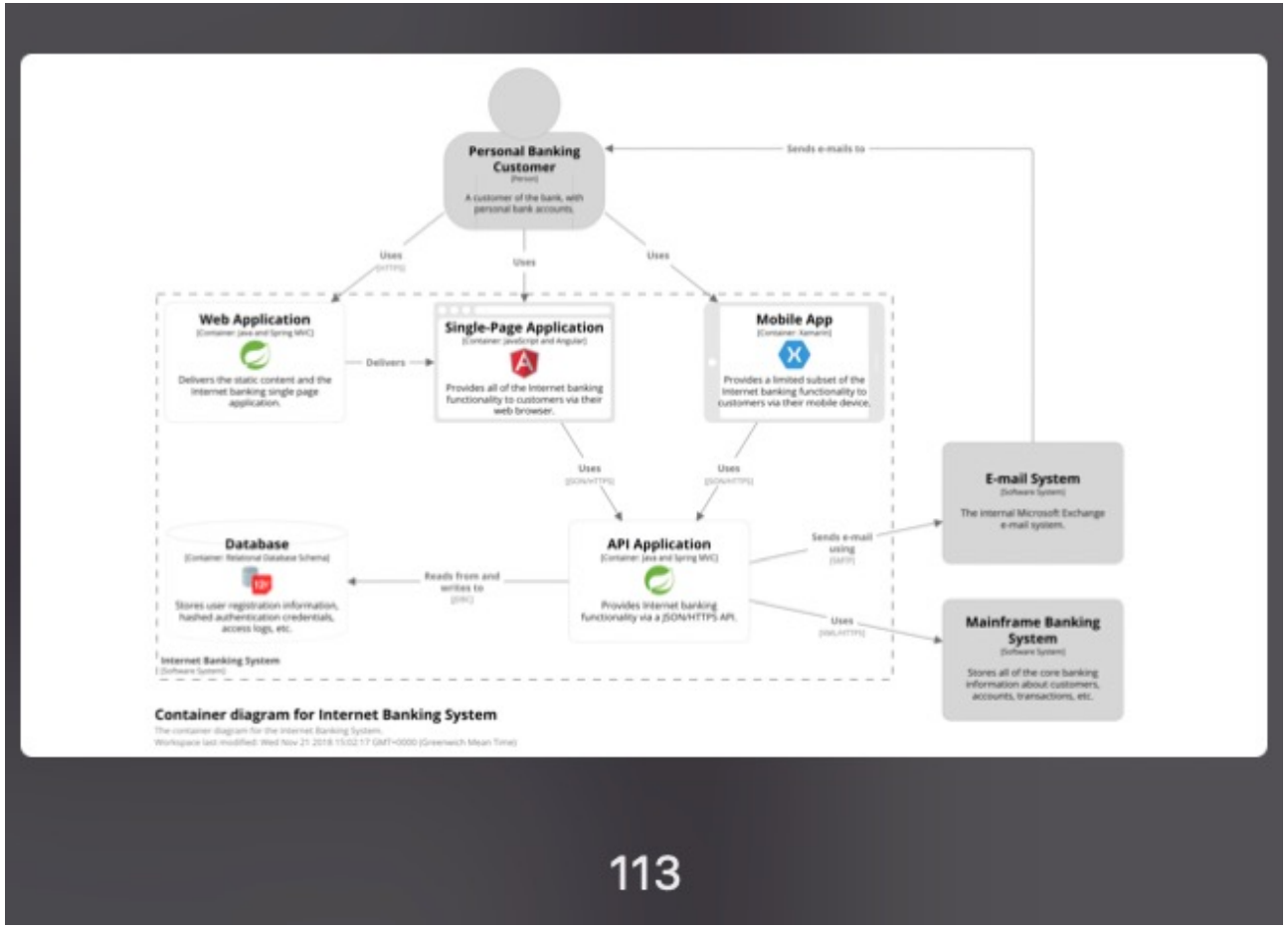
112

Some callouts from the attachment

https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf

Use icons to supplement text, not replace it

114



113

Some callouts from the attachment

https://github.com/ArchitectingSoftware/SE577-SoftwareArchitecture/blob/main/lectures/visualising-software-architecture-with-the-c4-model.pdf