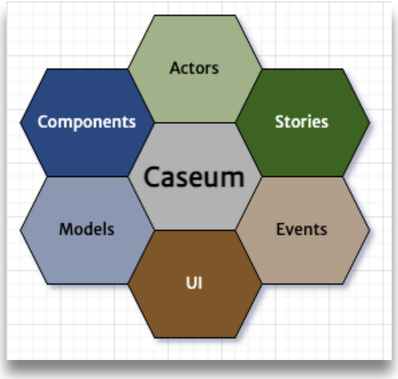


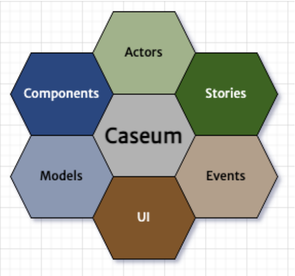
Caseum

Introducing a lightweight approach to software architecture

Leo Simons, November 2023

https://caseum.org

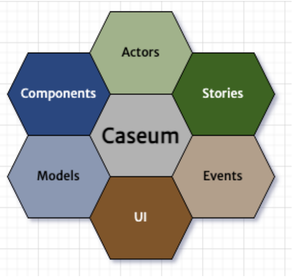
Key concepts



Caseum is simple

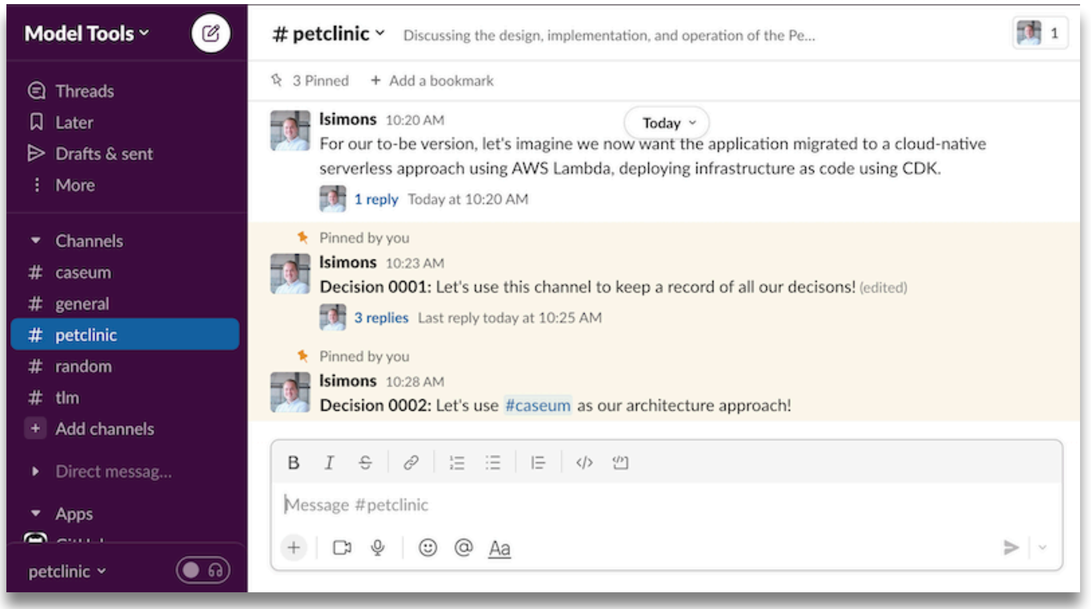
- **Multiple views:**
  - Use C4 for component diagrams
  - Include other views for other important information
- **Lightweight approach:**
  - Don’t use a digital diagram when a whiteboard picture will do
  - If you do need structured models, use lightweight text-based syntax
  - Focus on interactions over processes and tools
- **User-centric:**
  - Include UI designs since they help users understand
  - Include Actor personas and scenarios since they drive the design
  - Events are a good way for users to understand systems
- **Standardise common best practices:**
  - Embeds established practices like Event Storming and Domain-Driven Design
  - Agile development context expected

Architecture = communication



Use simple, accessible tools

- Architecture records design decisions
- Designs are recorded for people working on and with the software
- Prefer tools that untrained non-technical people can use
  - Slack & Email *over* Documents
  - Wiki/SharePoint *over* Version Control
  - Readable text files *over* Formal Methods



Lightweight approach

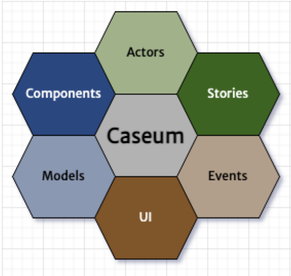
Three stages as projects get bigger

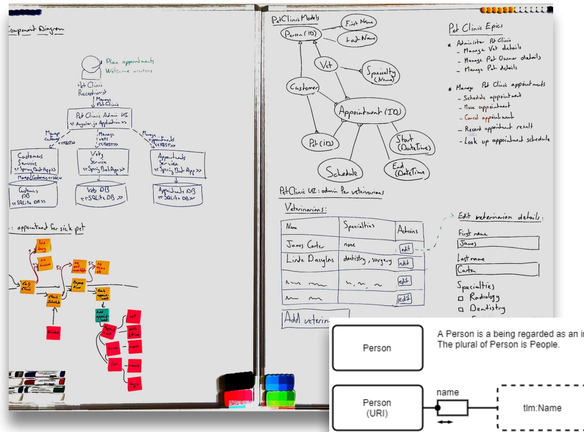
1. Brainstorming & whiteboarding

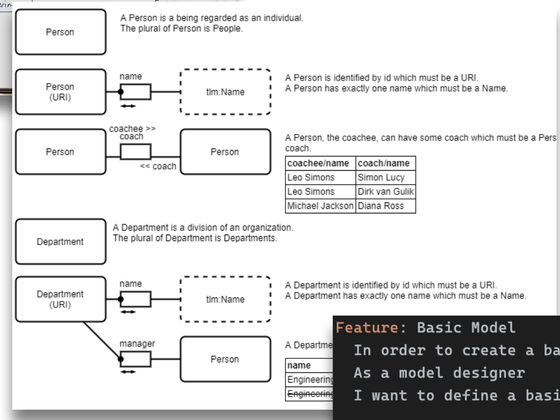
2. Digital diagrams & decision records

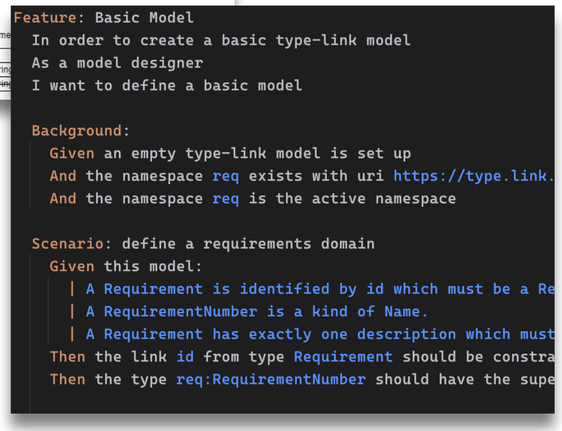
3. Models as code & executable specifications

...use only what you really need.

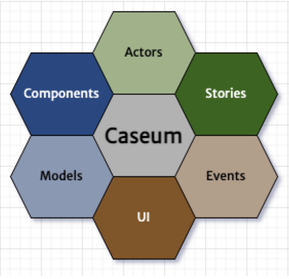






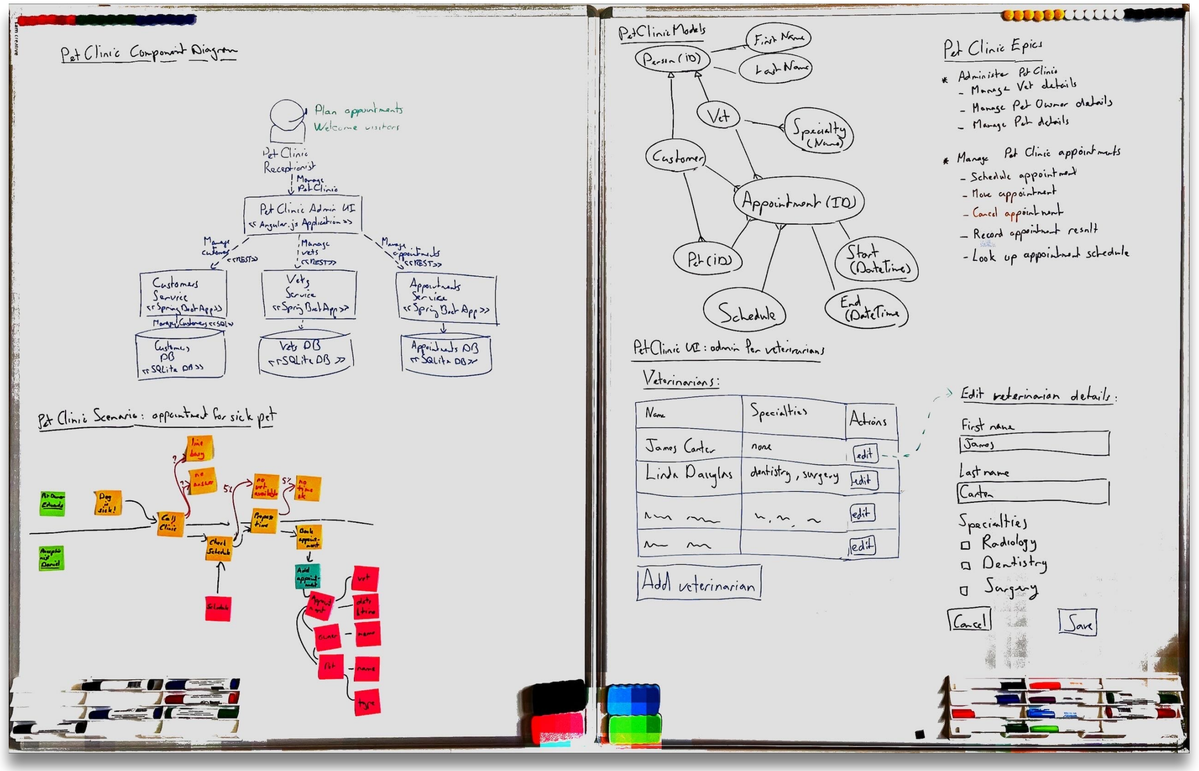


Stage 1: whiteboarding

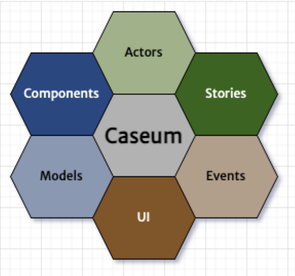


Design is an activity: draw again and again

- Draw the software design on a whiteboard using multiple different views
- Take pictures of the whiteboard and add them to decision records
- Drawing a picture while talking about it is a powerful way to share designs
- Caseum shows how to draw different views on the whiteboard

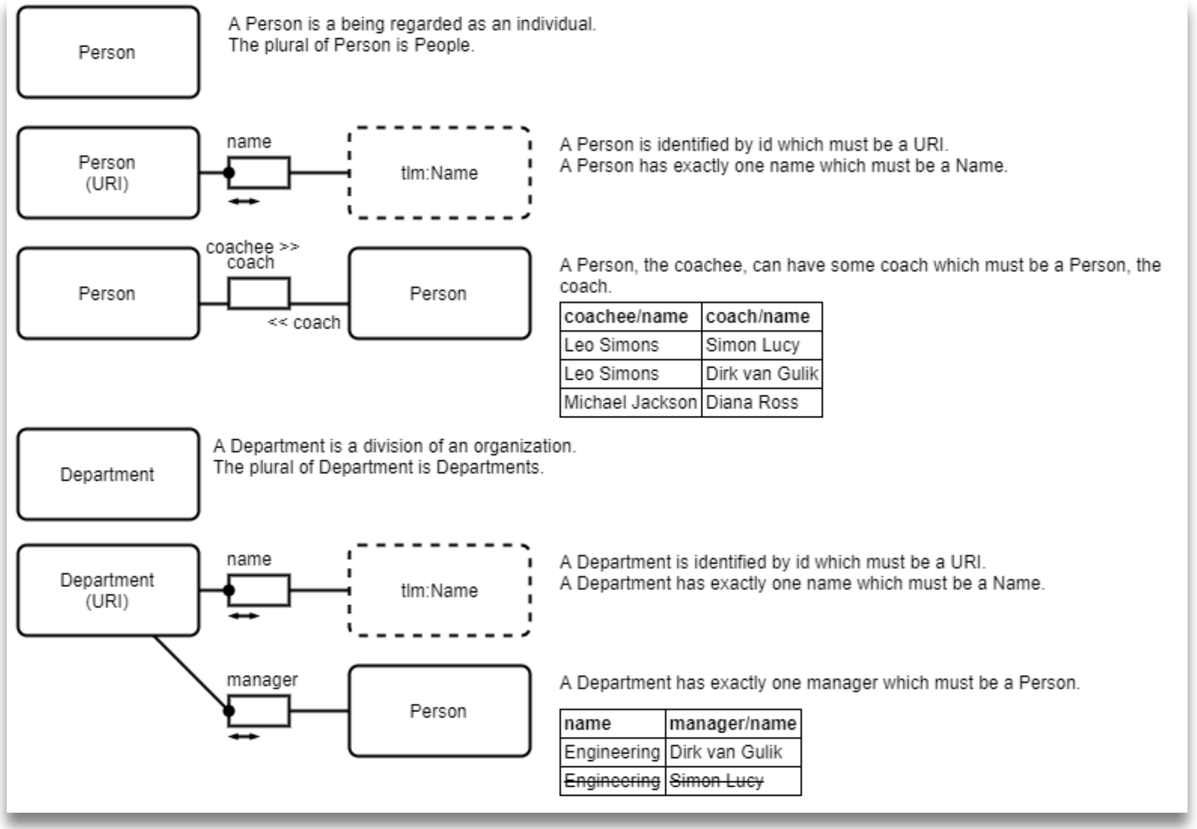


Stage 2: digital diagrams

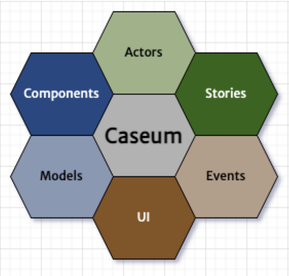


Use draw.io for high quality visuals

- When whiteboarding stops working...
- Create high-fidelity digital versions of the different Caseum views
- Use the draw.io libraries and templates Caseum provides
- Start keeping a digital record of your design decisions
- Use the simplest and most accessible digital tools so everyone can contribute to the design



Stage 3: models as code



Create structured text versions of views

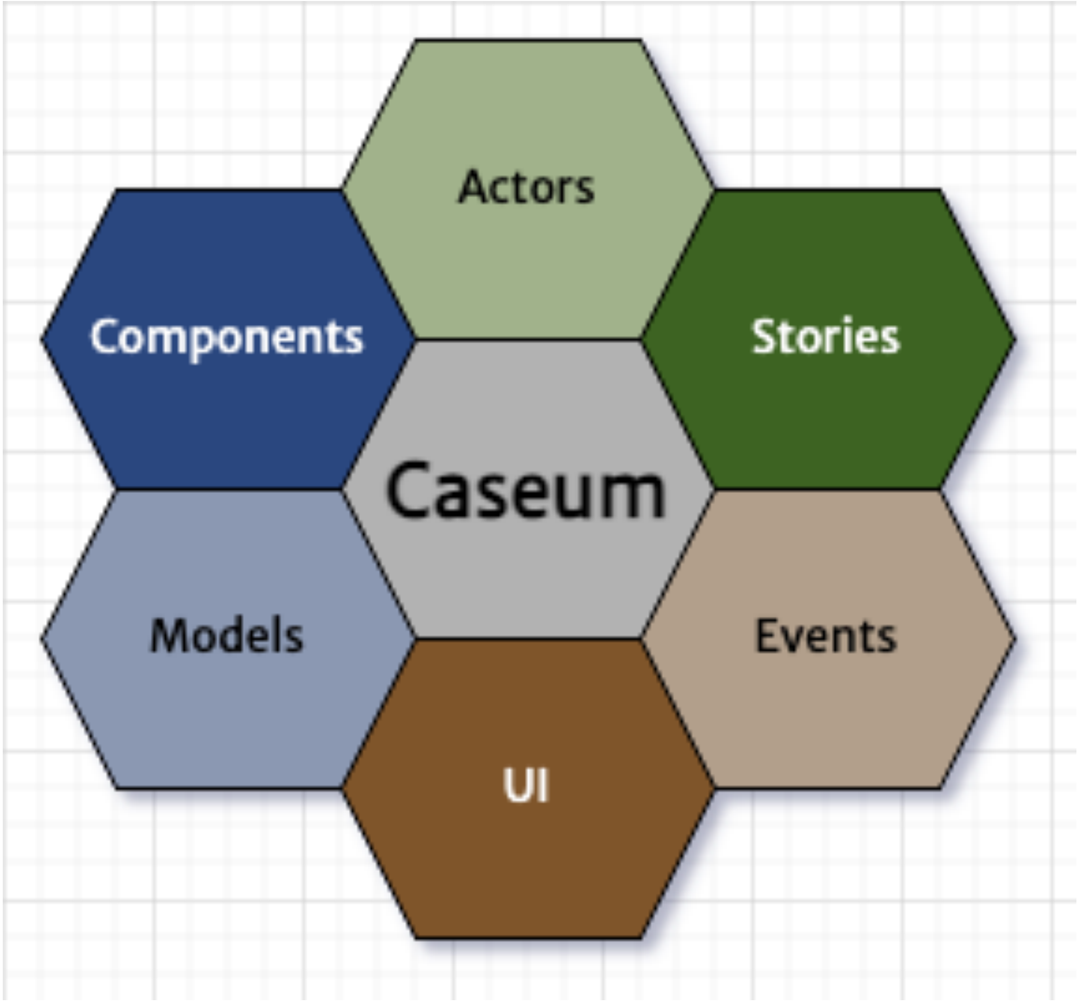
- Overkill for most projects
- Especially if you have modular architecture & independent teams
- When investing in codifying your architecture, focus on gaining back other benefits, such as
  - Automated testing
  - Generating code and docs
  - API contracts

```
Feature: Basic Model
  In order to create a basic type-link model
  As a model designer
  I want to define a basic model

  Background:
    Given an empty type-link model is set up
    And the namespace req exists with uri https://type.link.model.tools/ns/sample-requirements/
    And the namespace req is the active namespace

  Scenario: define a requirements domain
    Given this model:
      | A Requirement is identified by id which must be a RequirementNumber. |
      | A RequirementNumber is a kind of Name.                               |
      | A Requirement has exactly one description which must be a string.    |
    Then the link id from type Requirement should be constrained to values of type RequirementNumber
    Then the type req:RequirementNumber should have the supertype xs:Name
```

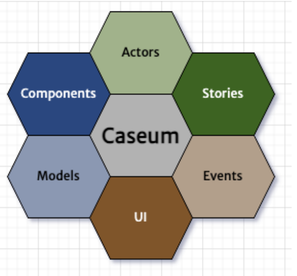
The 6 views in Caseum

Caseum is a mnemonic:

- **C**omponents using C4
- **A**ctors using roles
- **S**tories using Gherkin
- **E**vents using event storming
- **U**I using wireframes
- **M**odels using TLM

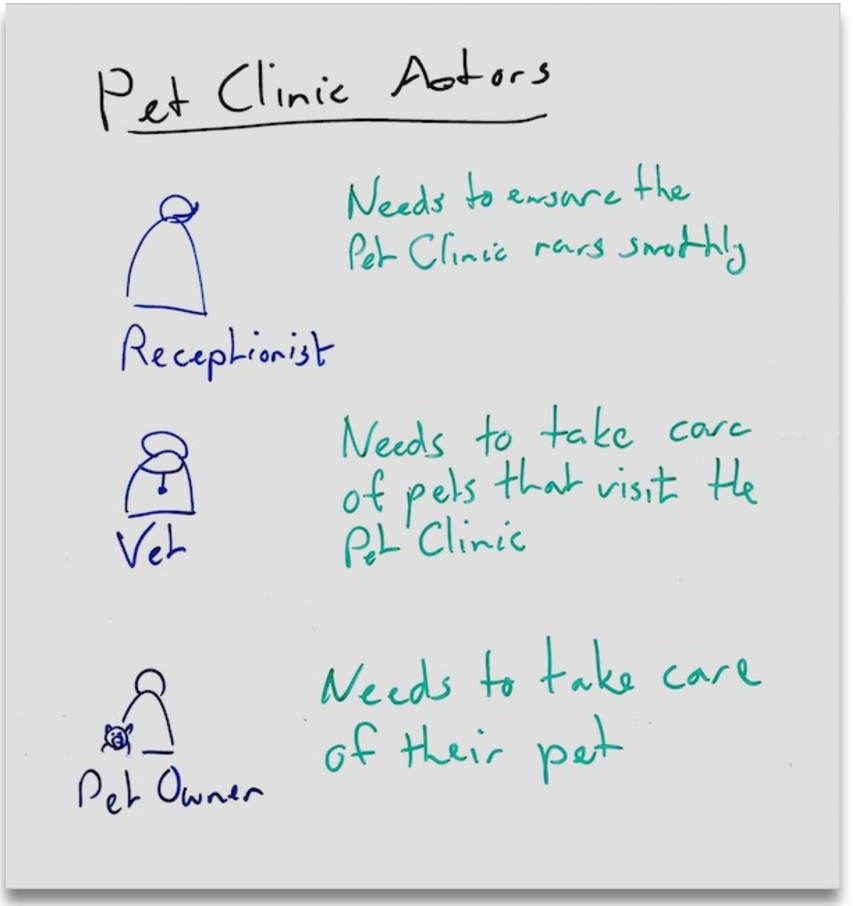


Actors

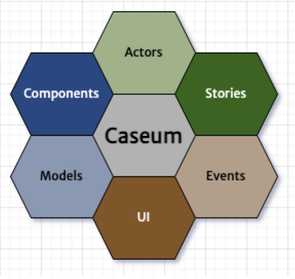


Caseum adopts role descriptions for the key actors

- Having a clear picture of your users in mind helps to make software accessible
- Deciding on your key users may be hard but helps to build the right thing
- Always capture at least the role (or system) **name** and their key **need** from the software you are designing

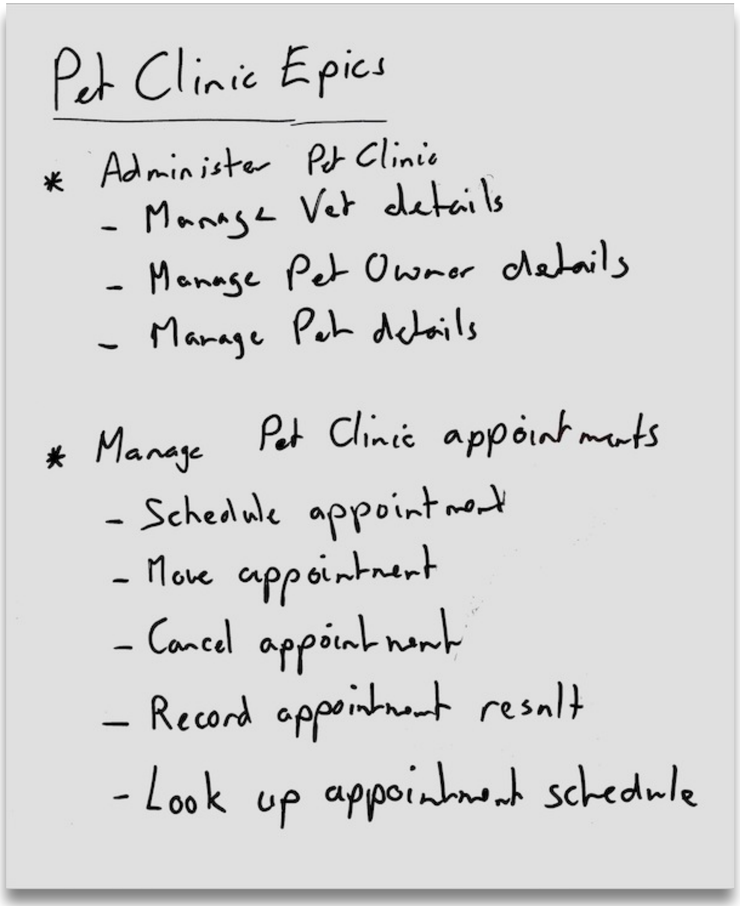


Stories

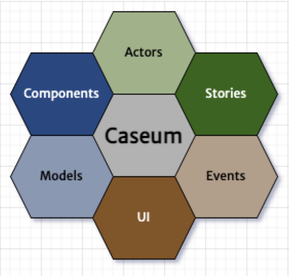


Caseum adopts user stories for functionality

- Describing functionality with and in the language of your users helps to make software that is useful
- Write just enough detail so everyone involved understands what is needed and add other Caseum views for more detail
- Implementing software one user story at a time is sometimes an option but Caseum does not recommend it



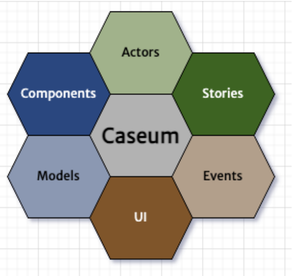
UI



Caseum adopts wireframes for the user interface

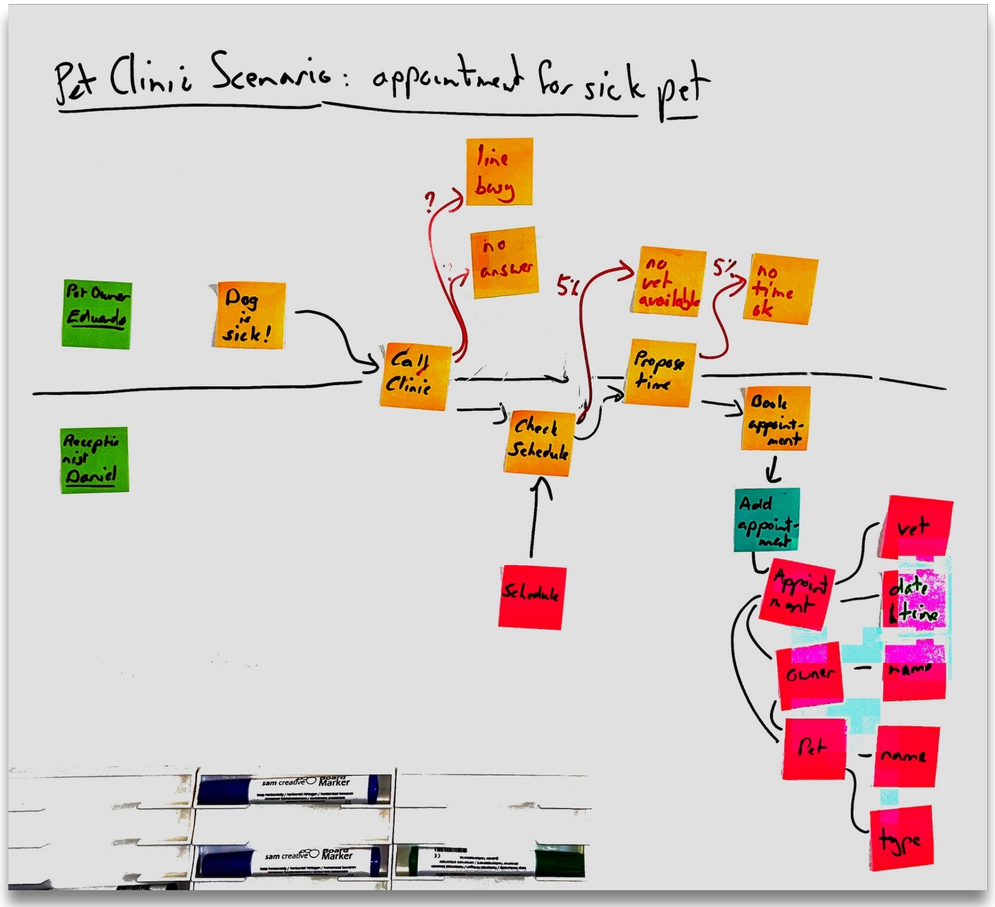
- Talking through how the UI will look and function with users and stakeholders helps clarify the other views
- Because wireframes limit detail they are easier to discuss and change
- Involve skilled UI/UX designers when possible

Events

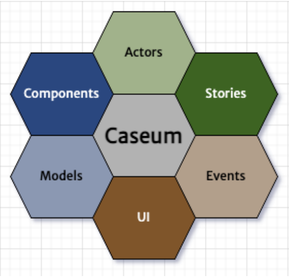


Caseum adopts Event Storming for business domain events

- Talking through the processes that software supports is a great way for developers to gain understanding of the business domain
- Having the whole process in one picture helps to make good choices for how to break large systems into pieces (by “bounded context”)

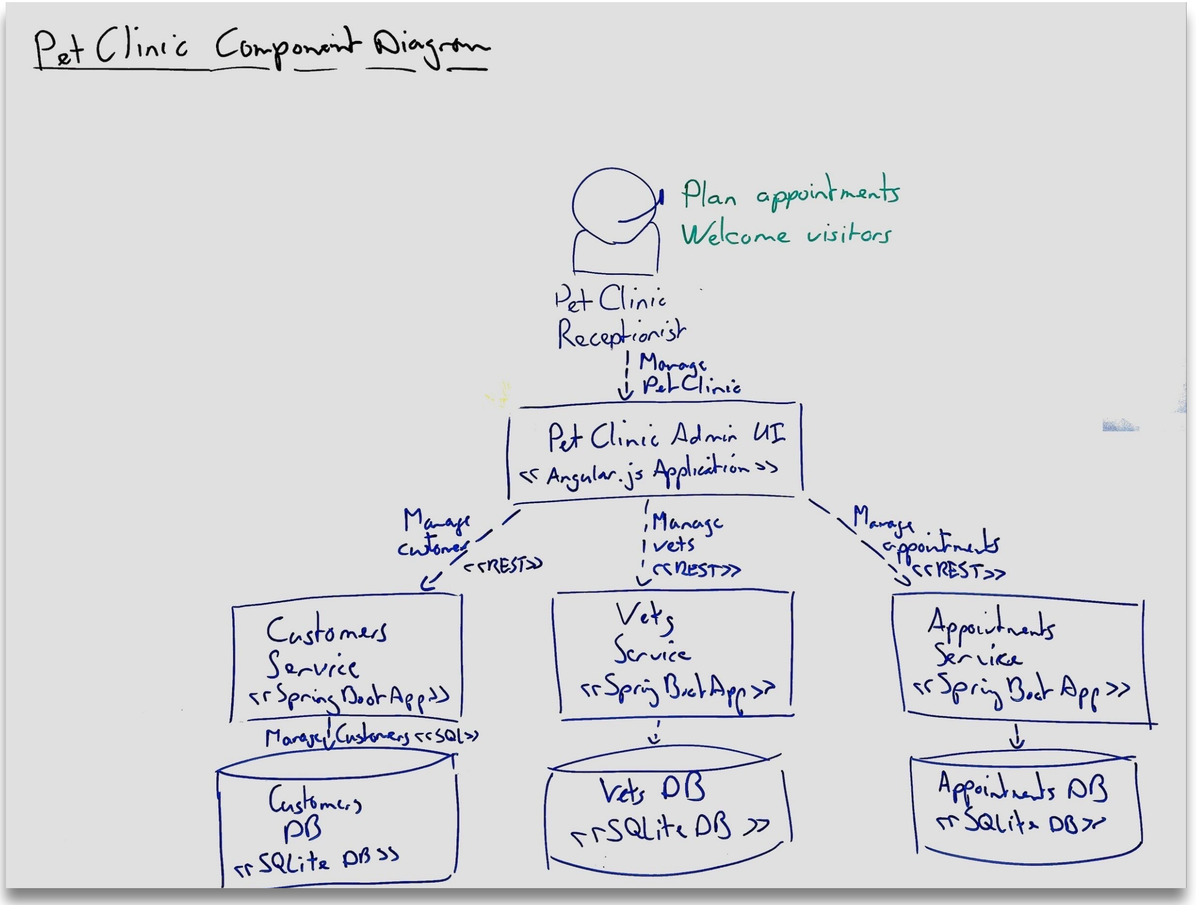


Components

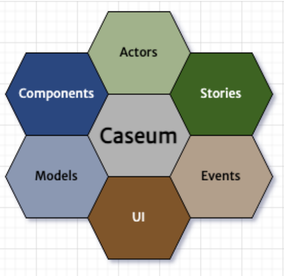


Caseum adopts C4 for components

- Focus on telling the story of your architecture *over* following the C4 model exactly
- The context and component diagrams are the most important, container diagrams can come later
- Avoid structurizr / models-as-code as long as you can

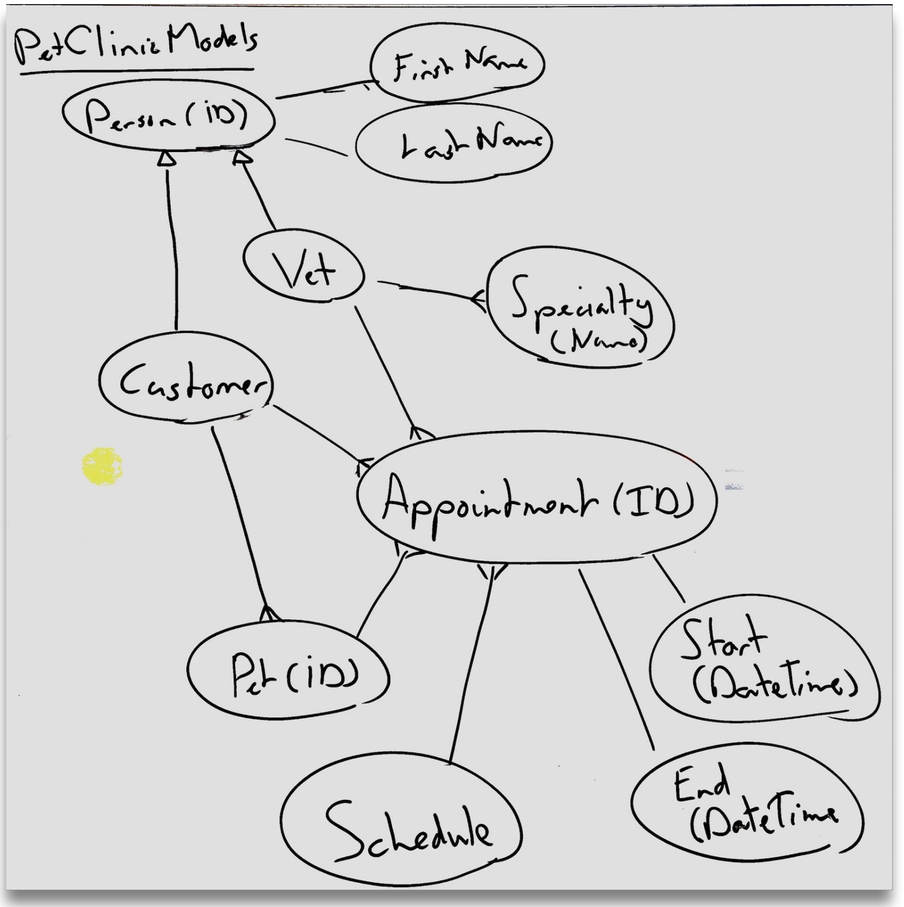


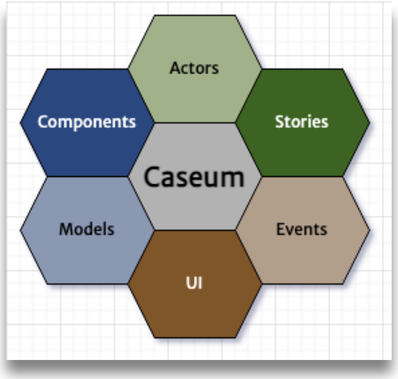
Models



Caseum adopts fact-based modelling for types

- Capture just the key facts in designing your data models
- Decide the physical structure of your data as late as possible, focus on the logic first
- Add examples of facts to make the model clear





Caseum

The simplest software architecture approach that could possibly work

Leo Simons, November 2023

https://caseum.org