Documentation

CS 346: Application Development

Writing code isn’t enough

We want to build maintainable software, that can be modified, update and remain useful over a long period of time.

However, writing well-structured code isn’t enough!

- The person writing the code may not be the person maintaining it.
- Even if you are maintaining code that you wrote, you will not remember the reasons for design decisions you made 6+ months ago.
- Code isn’t accessible to everyone in your organization! What about sales? Marketing? You can’t expect them to ”read the source code” to understand how something works.

Documentation is essential for long-lived projects.

Effective documentation is critical for the communication of complex ideas.

Types of documentation

Project documentation

Tracking project details to help us remember our project constraints. Useful for planning later phases. e.g., Issues lists; Milestones; Project plans; Gantt charts.

Design documentation

Why we made specific design decisions; materials to help new developers understand rationale. e.g., UML diagrams; design documents.

Code documentation

Inline documentation (code comments) to explain peculiarities of an implementation.

User documentation

Help users understand how something works! e.g., how to install; what features exist; what has changed in a new release.

Docs as Code

Documentation as Code (*Docs as Code*) is the philosophy that you should be writing documentation with the same tools you use to author and maintain code.

- Issue tracking – use this to track doc changes.
- Version control (Git) – version your docs with your code.
- Prefer plain text documents – you can diff them, use Git.
- Code reviews – docs should be included in feature reviews.
- Automated tests – unit test your docs!

"This [also] means following the same workflows as development teams and being integrated in the product team. It enables a culture where writers and developers both feel [collective] ownership of documentation and work together to make it as good as possible."

Docs as code does NOT mean that "your code is the documentation".

WRITE THE DOCS

www.writethedocs.org

Markup Languages (Markdown)

Authoring documents using plain text.

What is a markup language?

A markup language is a system of annotating a document to describe its structure and presentation. It uses tags or codes to define elements such as headings, paragraphs, lists, images, links, and more. Examples include HTML, AsciiDoc, reStructuredText and Markdown.

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>
<ul>
    <li>item 1</li>
    <li>item 2</li>
</ul>

</body>
</html>
```

My First Heading

My first paragraph.

- item 1
- item 2

HTML is useful, but do you really want to write your docs in it?

What is Markdown?

Markdown is a simple markup language that allows you to add formatting elements to a text file. Markdown was designed with a focus on generating HTML (see this blog post from 2004).

In its original form, Markdown is both:

- A formatting specification, and
- A tool for converting markdown files to HTML for publication.

In recent years, Markdown has become the *defacto* standard for technical documentation. It is less complete than other markup languages (e.g., AsciiDoc) but is simpler to use.

Using mdbook
See the [mdbook guide](https://rust-lang.github.io/mdBook/for_developers/index.html) for information on using `mdbook`.

There are several methods for navigating through the chapters of a book.

* The sidebar on the left provides a list of all chapters. Clicking on any of the chapter titles will load that page.
* The arrow buttons at the bottom of the page can be used to navigate to the previous or the next chapter.

This site supports the following keyboard shortcuts:

* `Arrow-Left`: Navigate to the previous page.
* `Arrow-Right`: Navigate to the next page.
* `t`: Jump to the top of the current page.
* `s`: Jump to the search bar (`ESC` to cancel).

Using mdbook

See the mdbook guide ↗ for information on using `mdbook`.

There are several methods for navigating through the chapters of a book.

- The sidebar on the left provides a list of all chapters. Clicking on any of the chapter titles will load that page.
- The arrow buttons at the bottom of the page can be used to navigate to the previous or the next chapter.

This site supports the following keyboard shortcuts:

- `Arrow-Left` : Navigate to the previous page.
- `Arrow-Right` : Navigate to the next page.
- `t` : Jump to the top of the current page.
- `s` : Jump to the search bar (`ESC` to cancel).

The course website is generated from Markdown! It's also used for documentation on GitLab, GitHub etc.

Basic Syntax

Symbol	Meaning
#	Heading 1
##	Heading 2
###	Heading 3
text	*Emphasis*
text	*Emphasis alt.*
text	**Embolden**
* item	Bulleted list
1. item	Numbered list
(title)[URL]	Link to a URL
!(title)[URL]	Embed an image

Why would we use Markdown?

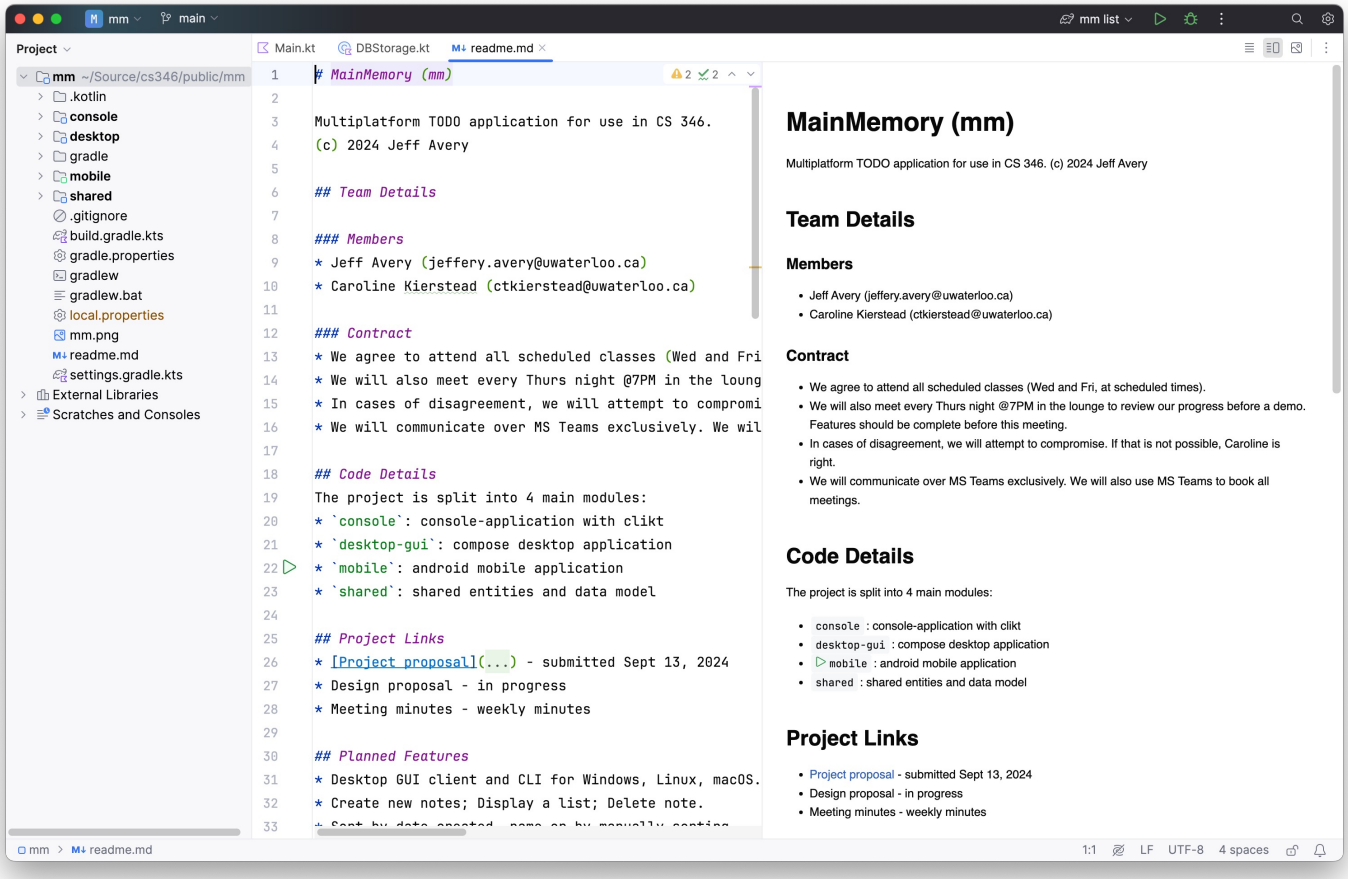
- You can write documentation in any text editor.
- **Text**, so you can version control it, diff it etc.
- VS Code, most IDEs, GitHub, **GitLab** support it.
- Defacto standard for software development.

Why not use Markdown?

- There is no standard specification (GitHub and a few organizations have produced extensions).
- Missing support for important features:
  - Footnotes
  - References
  - Floating images
  - Columns
- Works best at generating simple-HTML docs.

How do I use it?

- Editing Markdown
  - VS code, IntelliJ IDEA and most editors have support for Markdown.
- Integrating into your code/documents:
  - Online sites like GitLab, GitHub have built-in support i.e., you can enter text as markdown, and it will be shown “pretty-printed” when possible.
  - You can even embed diagrams into MD in your code projects!
- Generating HTML?
  - Tools like `pandoc` and `Marked` can convert markdown to HTML.
  - Static site generators: Jekyll, Hugo, Retype all generate websites from markdown.



Most development tools will work with Markdown. IntelliJ IDEA for example has support for Markdown syntax and will even pretty-print the output.

Tools > Mermaid.js

How do we generate diagrams-as-code?

Diagramming

- Documentation requires diagrams.
- We can imagine adding many different types of diagrams and charts to our documentation, including:
  - Gantt charts to project management documents.
  - Timeline charts to show milestones and your delivery schedule.
  - UML diagrams for design, and to document implementation details.
    - Component diagrams, class diagrams, sequence diagrams, state diagrams…
  - Flowcharts, and requirements diagrams to explain features to customers.
  - Pie charts to show results.

Diagramming Tools

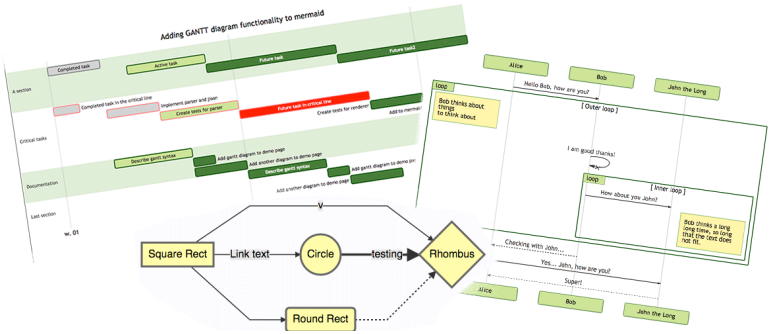
There are many types of diagramming tools:

1. **Pixel manipulation tools**
   - Produce image formats e.g., PNG. Poor for diagrams; large files, don't scale well.
2. **Vector drawing tools**
   - Produce SVG files or a similar format, which you can embed as images.
   - Very precise; complete control over the results!
   - e.g., Affinity Designer, Adobe Illustrator
3. **Markup-based drawing tools**
   - You use a markup language to describe your diagram.
   - A diagram "engine" decides on format, layout etc., so it's less precise.
   - e.g., PlantUML, Mermaid.js

Mermaid.js

Mermaid is a JavaScript based diagramming and charting tool that renders Markdown-inspired text definitions to create and modify diagrams dynamically.

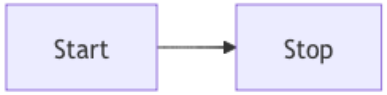
-- Mermaid.js.org



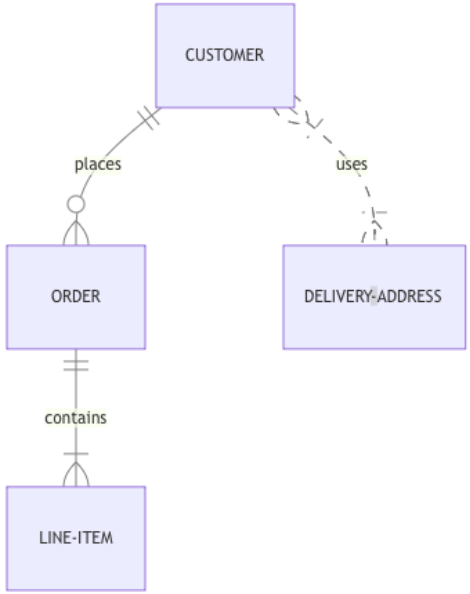
Mermaid supports a HUGE range of diagrams, including all UML diagrams, project charts, etc.

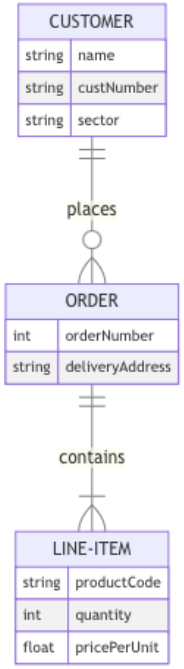
Mermaid.js Diagram Syntax

```mermaid
flowchart
    LR Start --> Stop
```

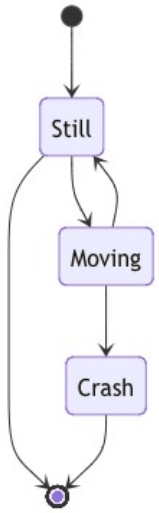



```mermaid
erDiagram
    CUSTOMER ||--o{ ORDER : places
    ORDER ||--|{ LINE-ITEM : contains
    CUSTOMER }|..|{ DELIVERY-ADDRESS : uses
```

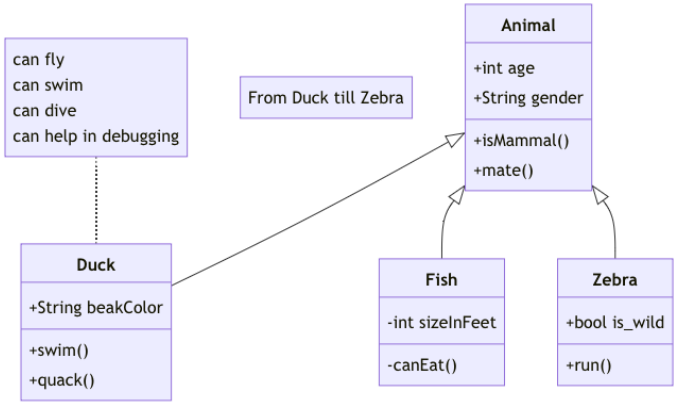




er diagram

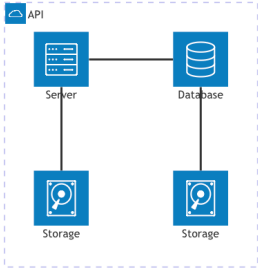


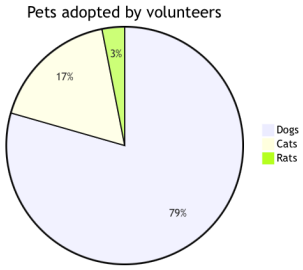
state diagram



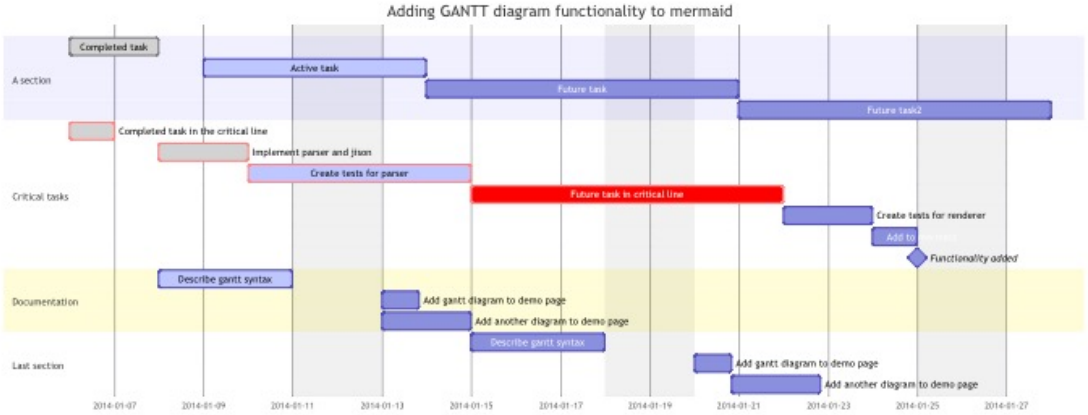
class diagram

See [Mermaid.js documentation] for examples





pie chart

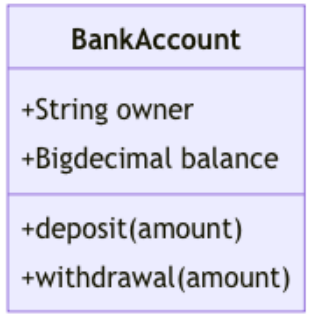


Gantt chart

Mermaid.js + Markdown

- Most environments that support Markdown also support Mermaid.
- This includes GitLab, GitHub, VS Code, IntelliJ IDEA, pandoc, …

```` mermaid
classDiagram
    class BankAccount
    BankAccount : +String owner
    BankAccount : +Bigdecimal balance
    BankAccount : +deposit(amount)
    BankAccount : +withdrawal(amount)
```




You can wrap Mermaid expressions in code blocks in your Markdown documents, and they will be rendered inline. This works in GitLab too!

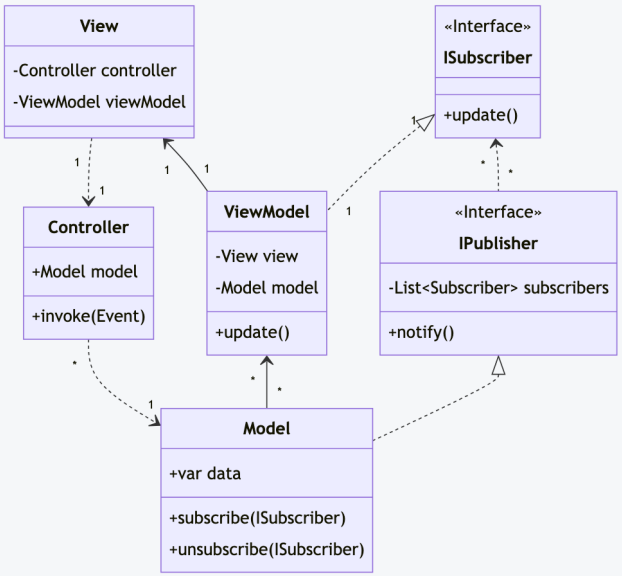
```
44
45  ## Design
46
47  > You need the JetBrains Mermaid plugin installed
48
49  * **Dependencies**: flow from View (top) to Model
50  * **Data flow**: notifications flow bottom to top
51
52  ```mermaid
53  classDiagram
54      View "1" ..> "1" Controller
55      Controller "*" ..> "1" Model
56
57      ISubscriber "1" <|.. "1" ViewModel
58      IPublisher <|.. Model
59      ISubscriber "*" <.. "*" IPublisher
60
61      View "1" <-- "1" ViewModel
62      ViewModel "*" <-- "*" Model
63
64      class View {
65          -Controller controller
66          -ViewModel viewModel
67      }
68
69      class ISubscriber {
70          <<Interface>>
71          +update()
```

- 1.3. Desktop GUI support. Split project into modules.
- 1.4. Android support. Added mobile project.

## Design

> You need the JetBrains Mermaid plugin installed to show this diagram.

- **Dependencies**: flow from View (top) to Model (bottom).
- **Data flow**: notifications flow bottom to top via interfaces.




Most tools support Mermaid diagrams in Markdown documents. IntelliJ IDE above shows this diagram inline with Markdown documentation.

# Tools > Figma

Generating screen mockups

# Prototyping

A **prototype** is a mock-up of your solution, that is built to demonstrate functionality and elicit feedback from your users.

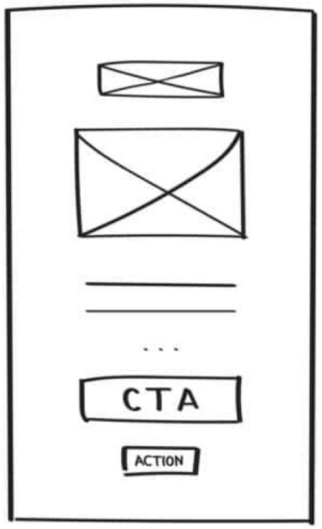
- Helps you determine
  - Which features are interactive, and how the user can utilize them.
  - What input is required for a screen, what output makes sense.
  - The order of screens! How the user will navigate (does it make sense?)
- Useful to show to your user and walk through all the features to get initial “buy in”.
- You will not get everything right! The goal is *early feedback from users*.

# Fidelity

Fidelity refers to the level-of-detail.

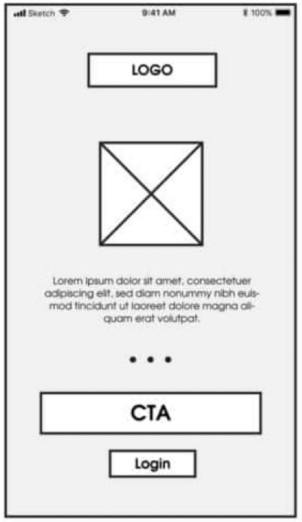
Low-fidelity prototypes are deliberately simple, low-tech, and represent a minimal investment.

- You can sketch on paper.
- Many online tools help you build wireframe diagrams that you can demo e.g., Figma.
- Higher fidelity can be semi-interactive to test progression through the interface.

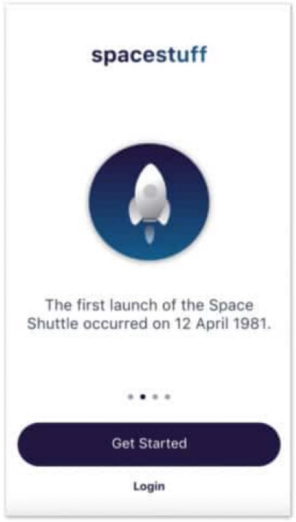



Low

X




Medium




High

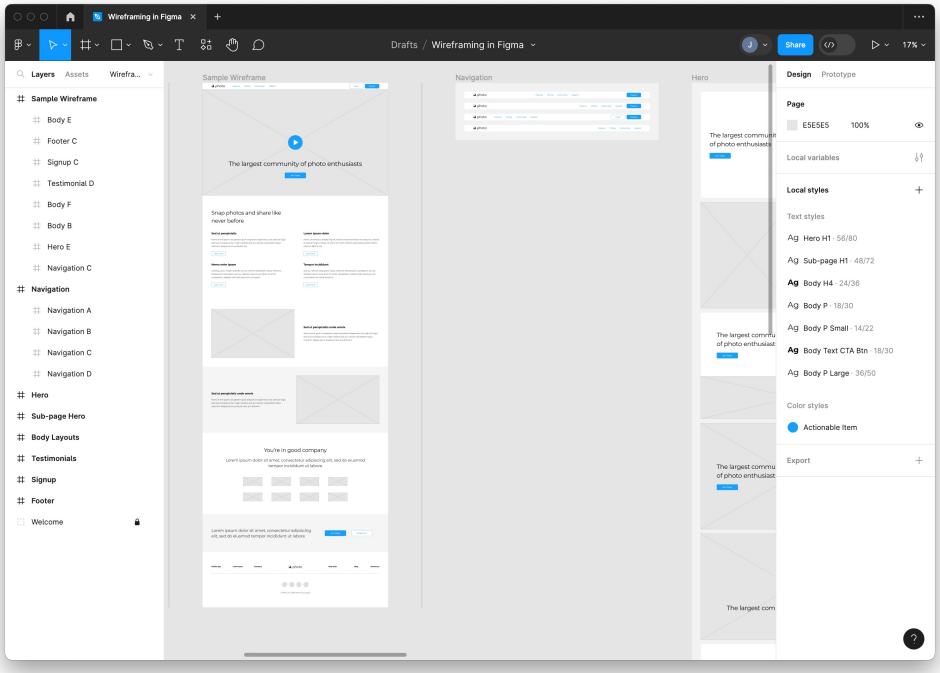
# Figma

Figma is a very popular prototyping tool (desktop, mobile, web).

- Mock screens & interactions.
- Can build specific UI designs (e.g., iPhone, iPad, desktop).
- Iteration is much easier compared to paper prototyping.

Other options

- Omnigraffle (mac)
- Balsamiq (win, mac, web)
- Hand-drawn diagrams




https://www.figma.com/