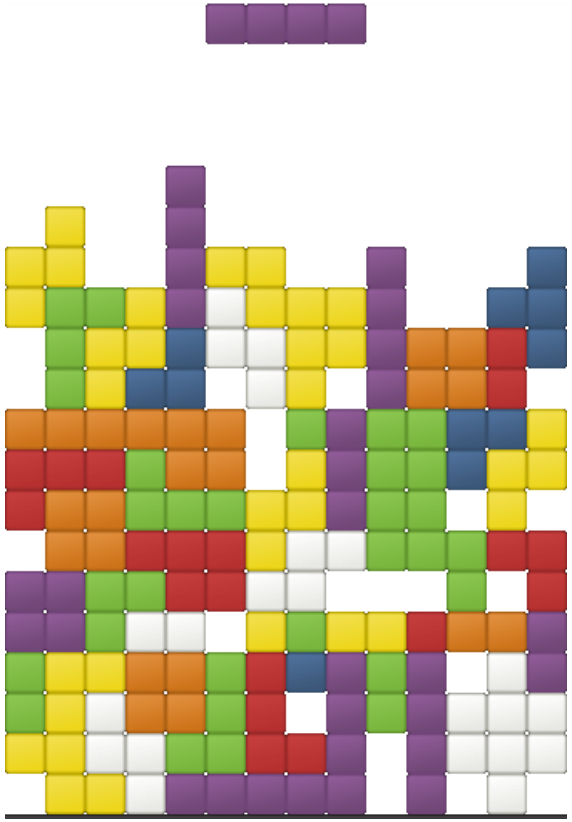
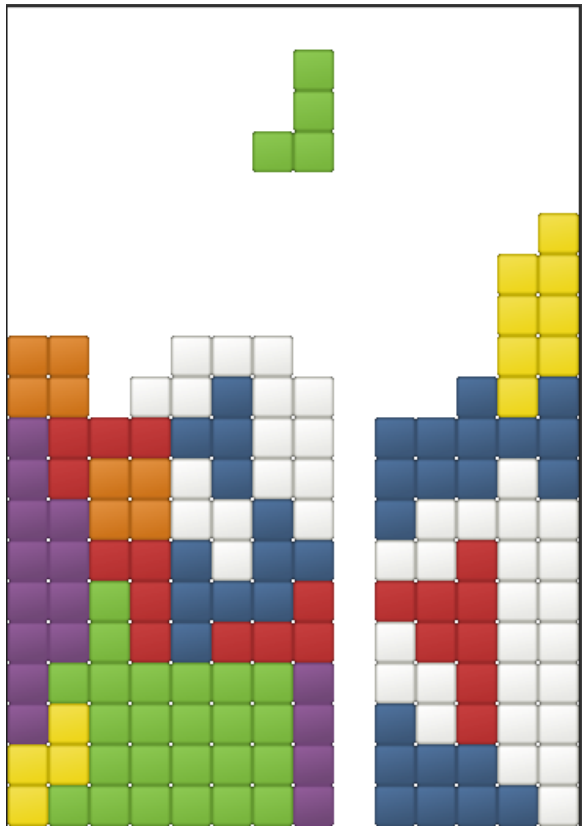
Software Design for Scientific Computing





Course Overview

- 19.04 (12:00 – 14:00) Theory session
- 19.04 Get assignment
- 19.04 Decide if you want to work with a partner, organize in HackMD

- 22.04 (midnight) last time to open a pull request to assignment repository

  This does not have to be complete or perfect!
- 23.04 we will organize reviews

- 26.04 first round of reviews: Get reviewed and give one review (each ½ hour)
- 26.04 get second part of assignment

- 29.04 (midnight) last time to open pull request for second part of assignment
- 30.04 we will organize second round of reviews

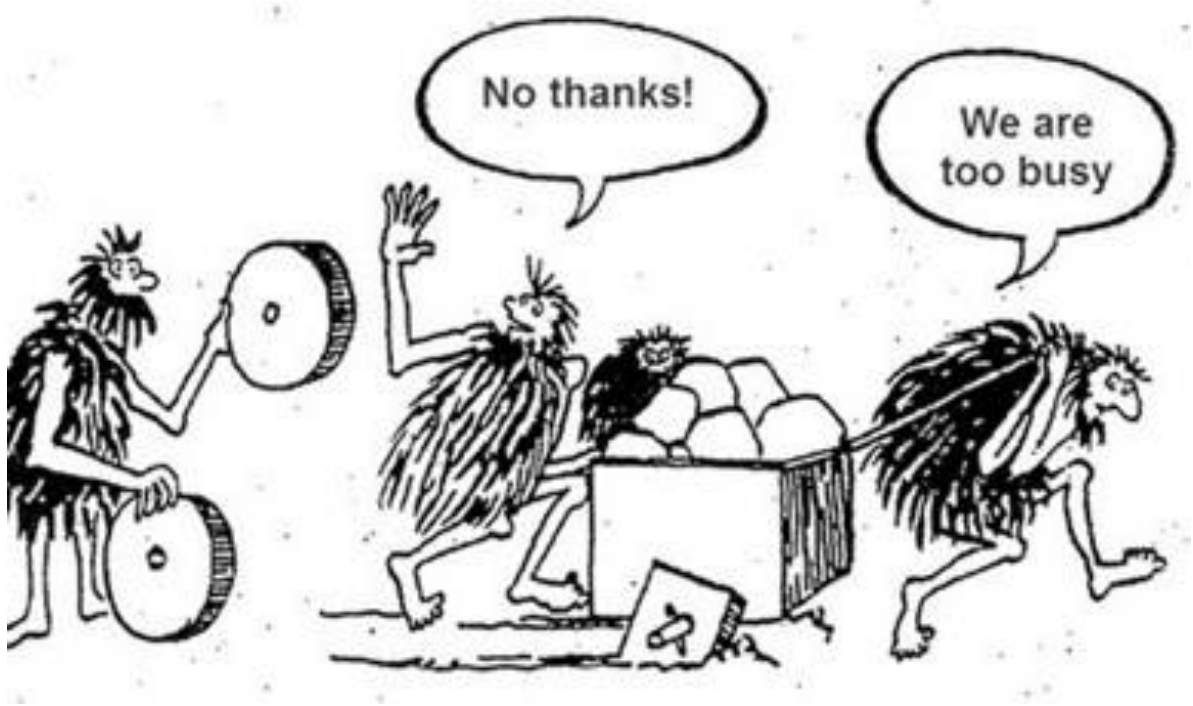
- 03.05 second round of reviews
- 03.05 final feedback session

Overview

- Why code design
- 9 principles of good code design
- How to put it into practice - demo
- Assignment introduction

Why put effort into code design

- Get faster
- Sharable code
- Re-usable code
- Correct code
- Avoid mistakes early
- Don't get stuck midway through implementation
- Guard against code decay



Design Principles

- **Code is only done when it is easy to read.**
- Eliminate small inconsistencies immediately.
- Remove unnecessary code immediately.
- Separate general-purpose from special-purpose.
- Make concise abstractions.
- Make the common use case simple.
- Code should be obviously correct.
- Comments describe things that are not obvious.
- Any time you change the code, you are making design choices.

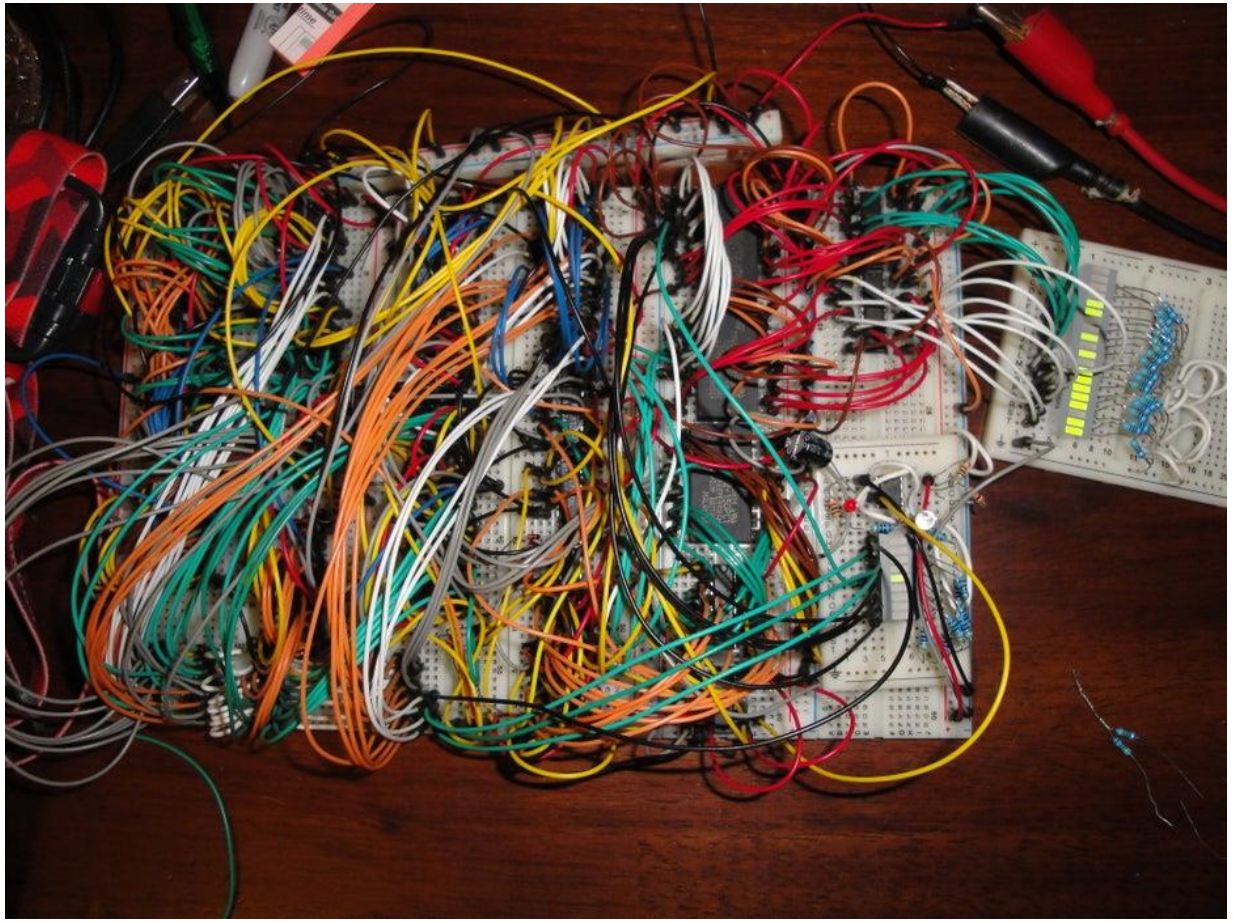
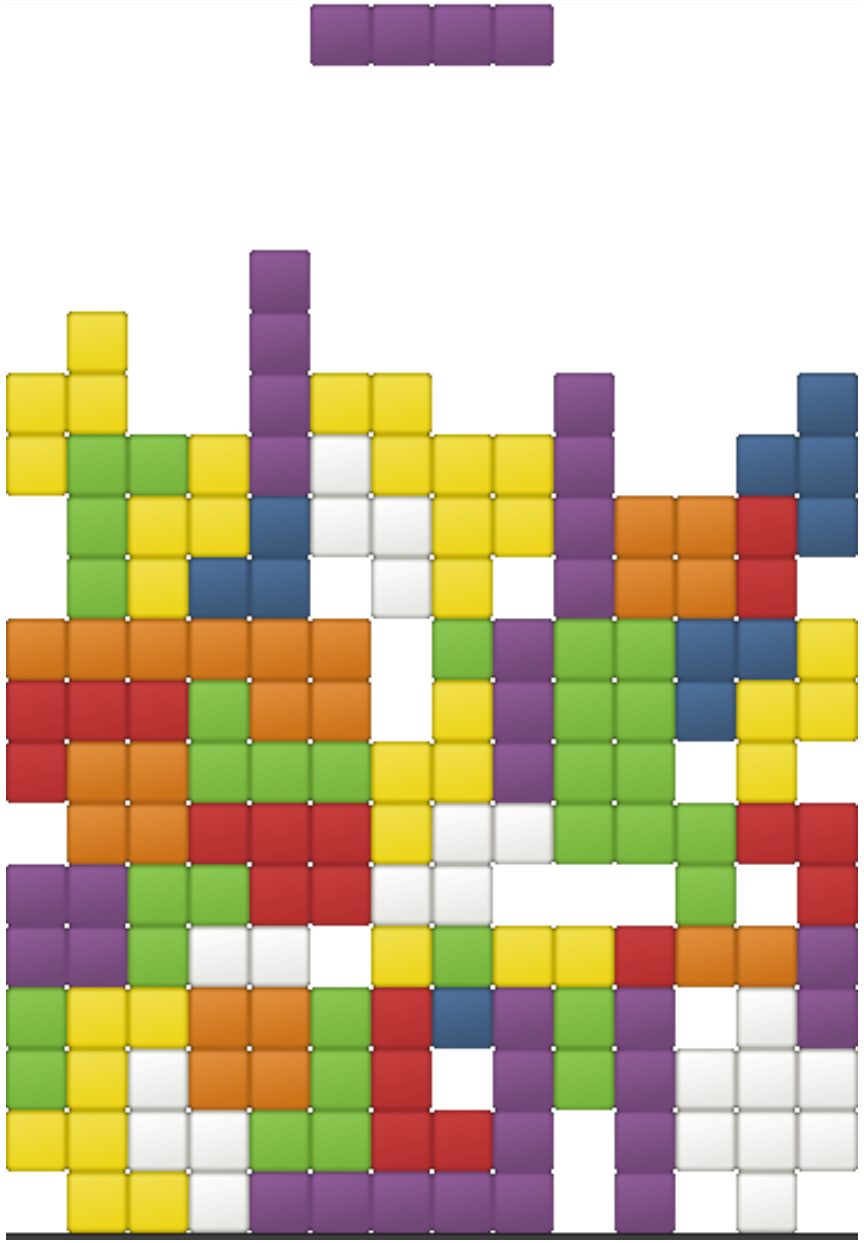


Image from: https://www.instructables.com/id/How-to-Build-an-8-Bit-Computer

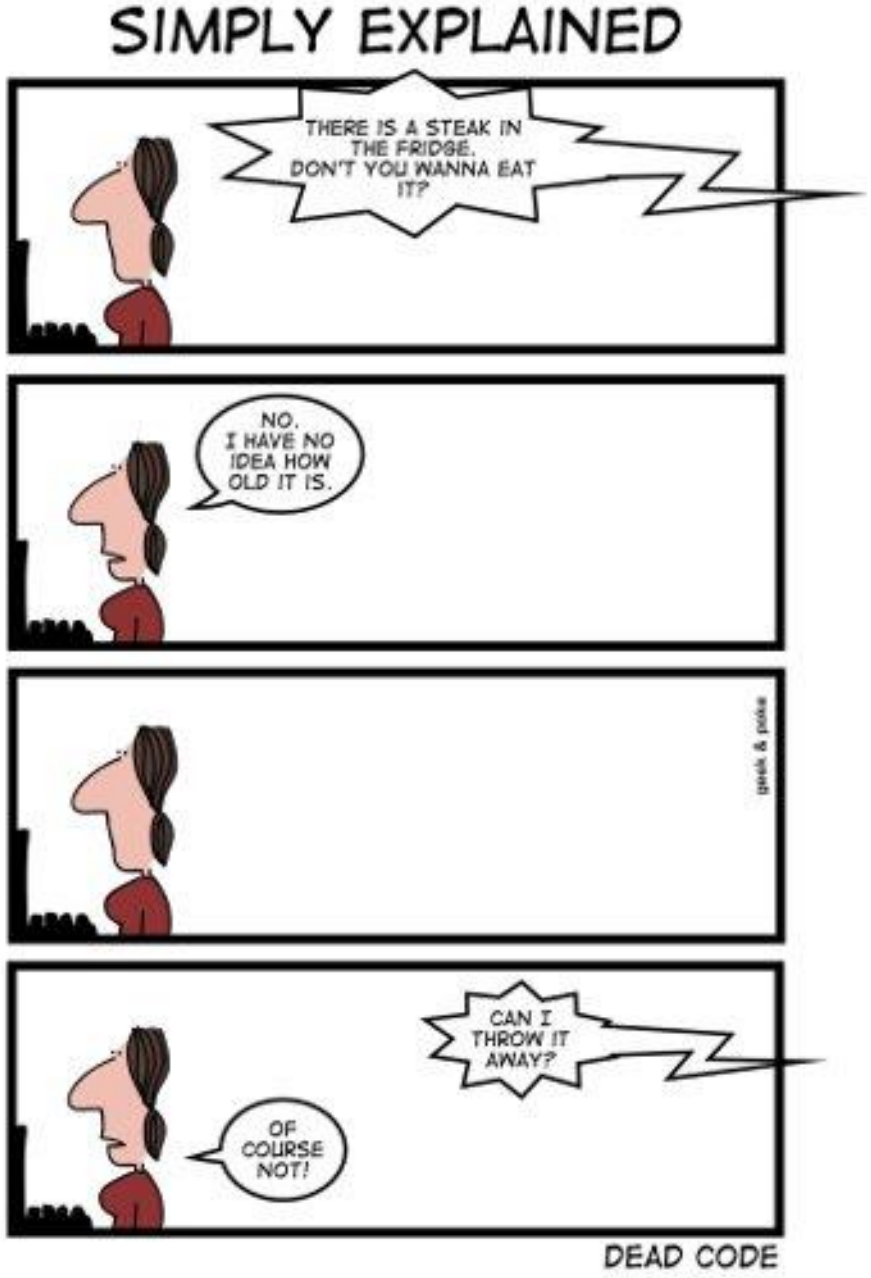
Design Principles

- Code is only done when it is easy to read.
- **Eliminate small inconsistencies immediately.**
- Remove unnecessary code immediately.
- Separate general-purpose from special-purpose.
- Make concise abstractions.
- Make the common use case simple.
- Code should be obviously correct.
- Comments describe things that are not obvious.
- Any time you change the code, you are making design choices.



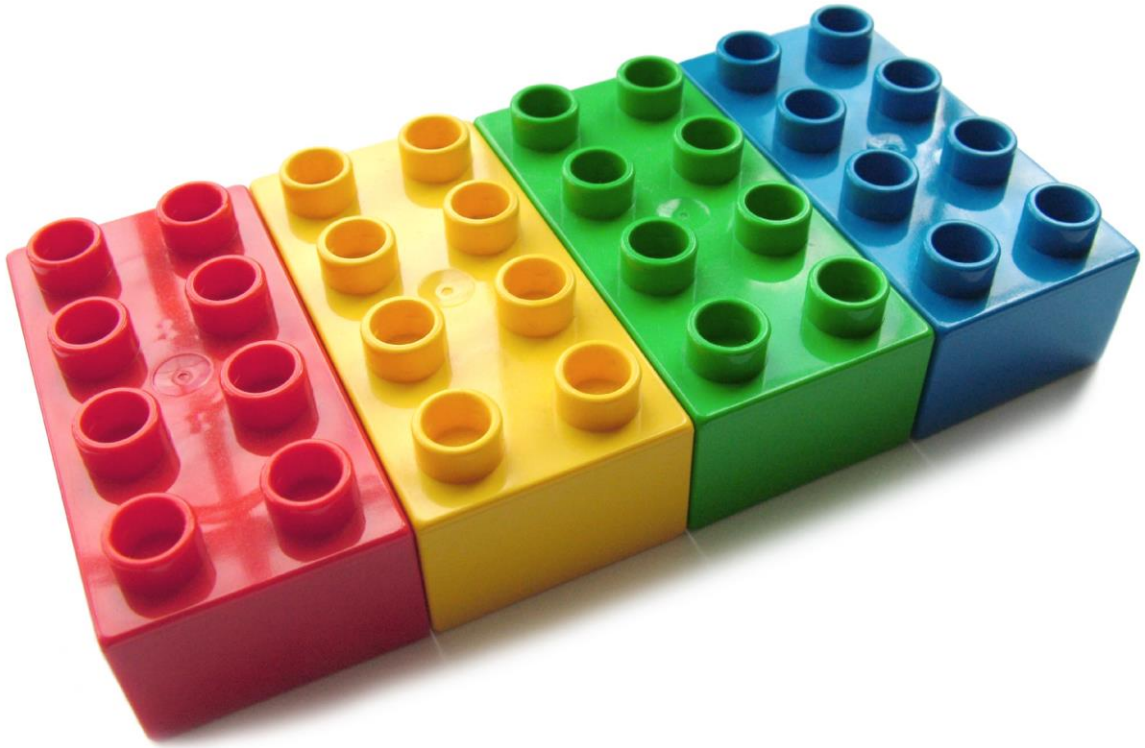
Design Principles

- Code is only done when it is easy to read.
- Eliminate small inconsistencies immediately.
- **Remove unnecessary code immediately**.
- Separate general-purpose from special-purpose.
- Make concise abstractions.
- Make the common use case simple.
- Code should be obviously correct.
- Comments describe things that are not obvious.
- Any time you change the code, you are making design choices.



Design Principles

- Code is only done when it is easy to read.
- Eliminate small inconsistencies immediately.
- Remove unnecessary code immediately.
- **Separate general-purpose from special-purpose.**
- Make concise abstractions.
- Make the common use case simple.
- Code should be obviously correct.
- Comments describe things that are not obvious.
- Any time you change the code, you are making design choices.



Design Principles

- Code is only done when it is easy to read.
- Eliminate small inconsistencies immediately.
- Remove unnecessary code immediately.
- Separate general-purpose from special-purpose.
- **Make concise abstractions.**
- Make the common use case simple.
- Code should be obviously correct.
- Comments describe things that are not obvious.
- Any time you change the code, you are making design choices.

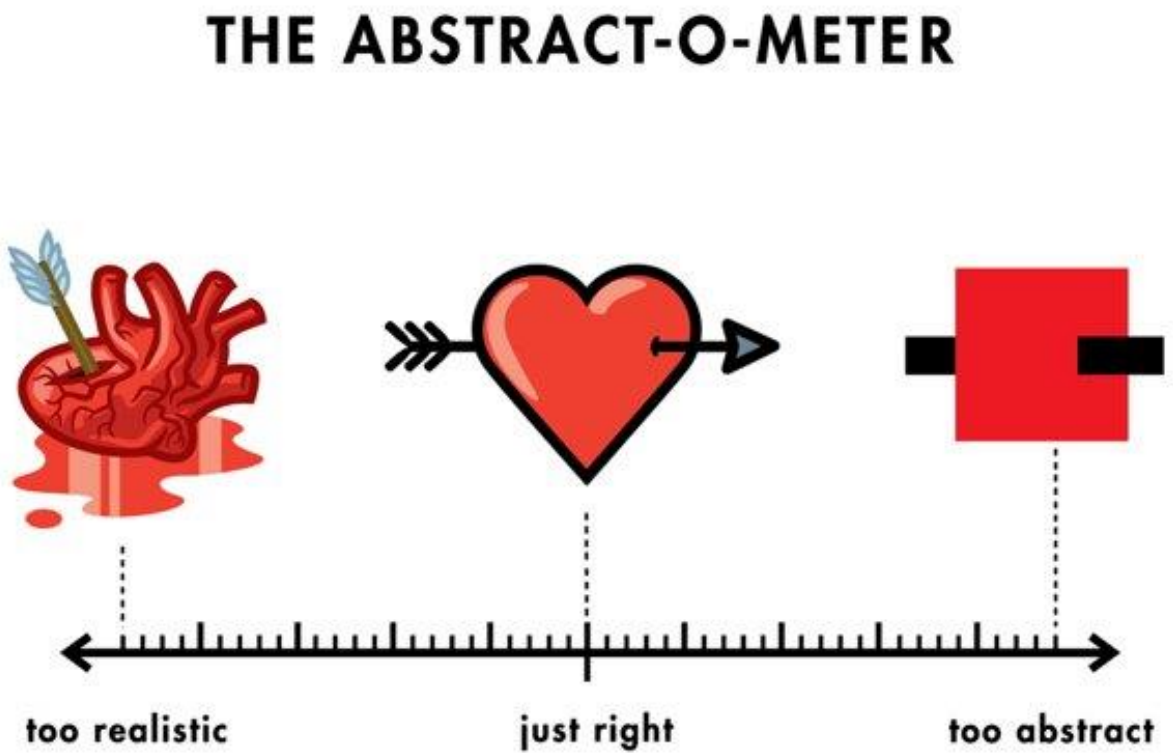


Image from Computer Science Wiki, licensed CC-BY-NC-SA 4.0

Design Principles

- Code is only done when it is easy to read.
- Eliminate small inconsistencies immediately.
- Remove unnecessary code immediately.
- Separate general-purpose from special-purpose.
- Make concise abstractions.
- **Make the common use case simple.**
- Code should be obviously correct.
- Comments describe things that are not obvious.
- Any time you change the code, you are making design choices.

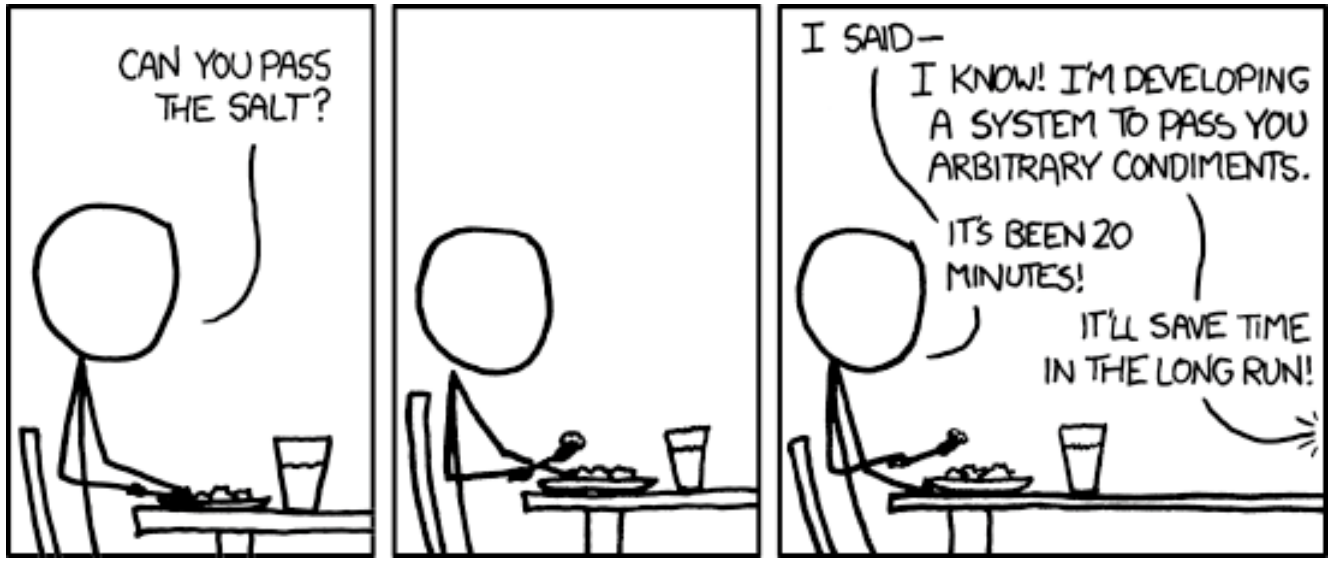
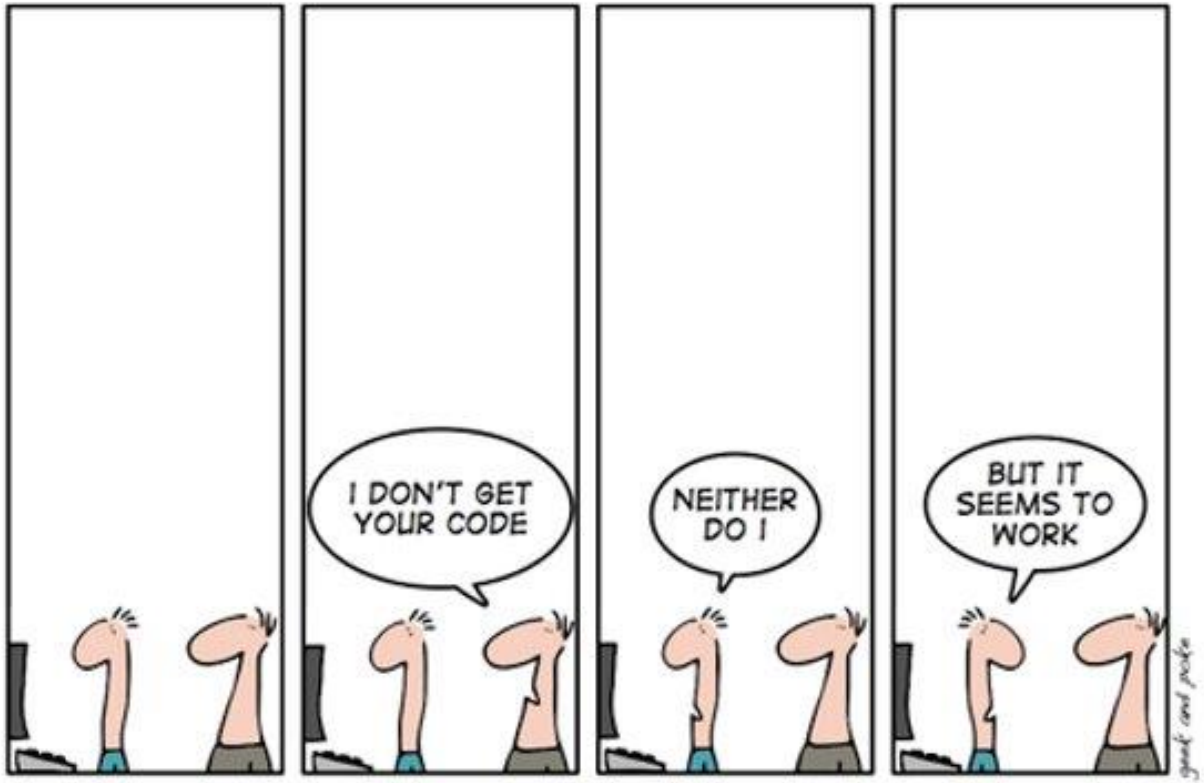


Image by XKCD, licensed under CC-BY-NC 2.5

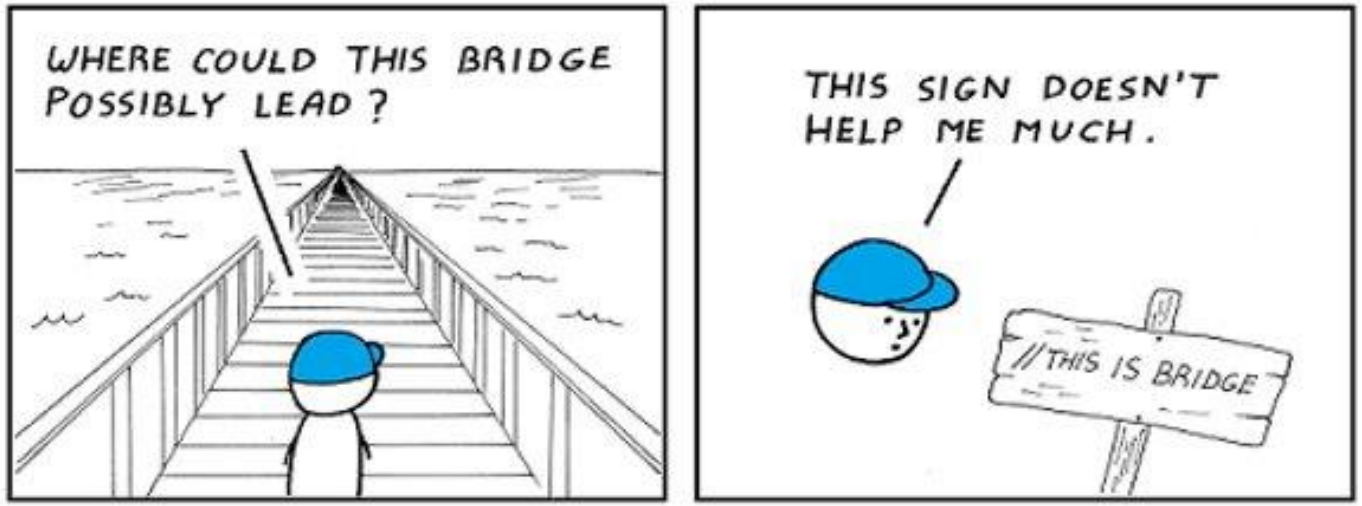
Design Principles

- Code is only done when it is easy to read.
- Eliminate small inconsistencies immediately.
- Remove unnecessary code immediately.
- Separate general-purpose from special-purpose.
- Make concise abstractions.
- Make the common use case simple.
- **Code should be obviously correct.**
- Comments describe things that are not obvious.
- Any time you change the code, you are making design choices.



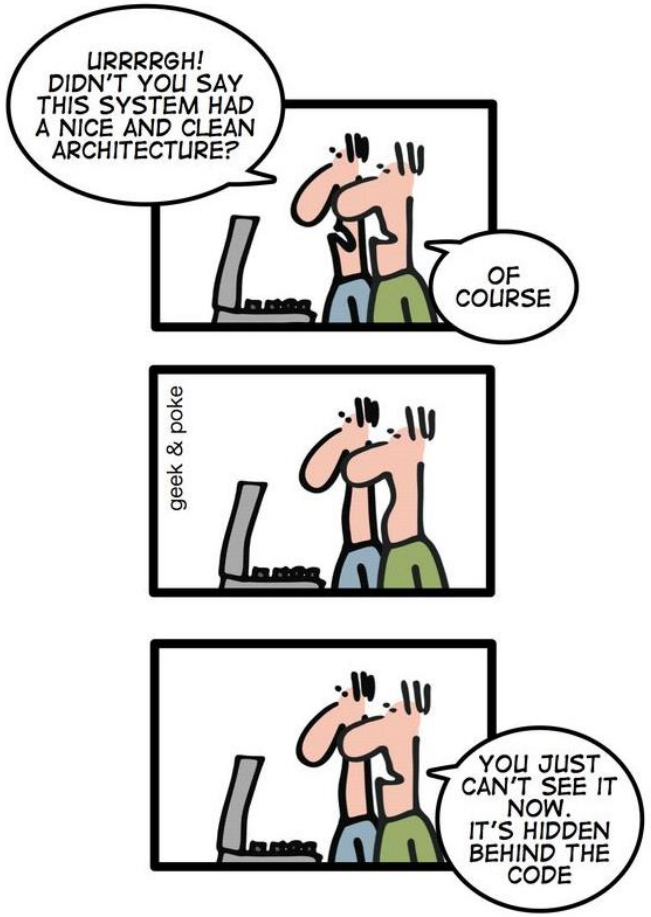
Design Principles

- Code is only done when it is easy to read.
- Eliminate small inconsistencies immediately.
- Remove unnecessary code immediately.
- Separate general-purpose from special-purpose.
- Make concise abstractions.
- Make the common use case simple.
- Code should be obviously correct.
- **Comments describe things that are not obvious.**
- Any time you change the code, you are making design choices.



Design Principles

- Code is only done when it is easy to read.
- Eliminate small inconsistencies immediately.
- Remove unnecessary code immediately.
- Separate general-purpose from special-purpose.
- Make concise abstractions.
- Make the common use case simple.
- Code should be obviously correct.
- Comments describe things that are not obvious.
- **Any time you change the code, you are making design choices.**



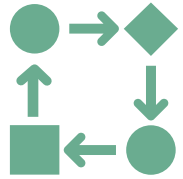
Strategies for designing and writing code - abstraction

Define the problem (on paper, as a picture, etc.)

Decide if the size of the problem, number of users, etc. needs a script or a library

software library: start your design by writing a minimal working example for the most common use case and test cases to go with it

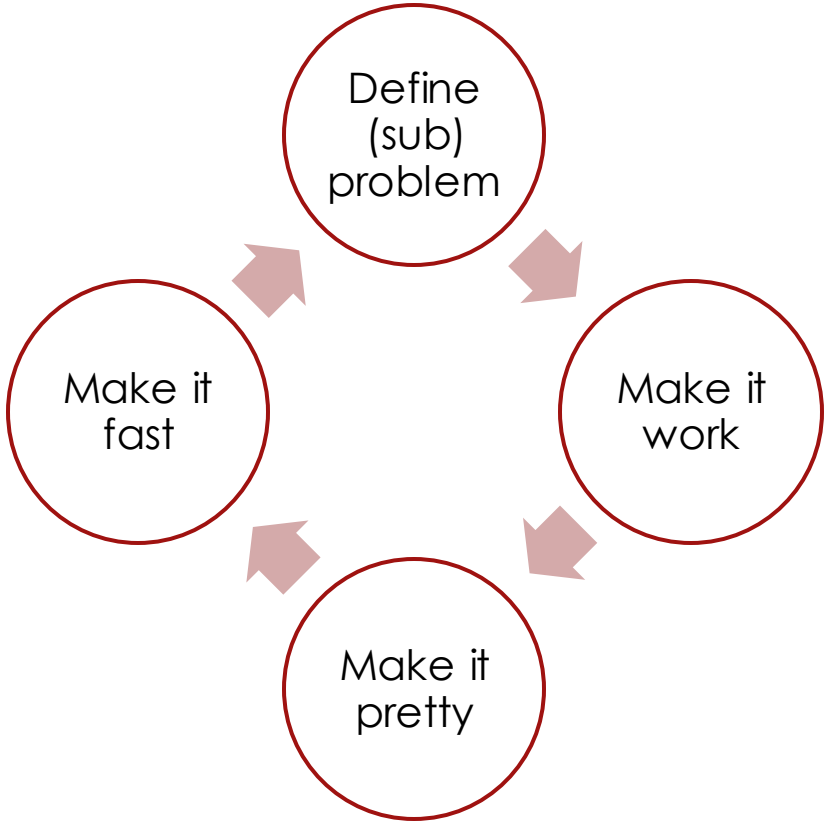
script: start your design by writing the individual steps as comments



Do "implementation loop"

Strategies for designing and writing code – "implementation loop"

Do the implementation loop for every part of the problem:



Assignment

FIND THE ASSIGNMENT AT:
HTTPS://VERSION.AALTO.FI/GITLAB/MERZS1/CDWASSIGNMENT

Assignment – how to solve

- You decide: implement with a partner or alone (You can organize in HackMD now)
- If you do the implementation with a partner, employ pair programming:
  - Meet at the start to discuss the design
  - Implement potential difficult bits using pair programming like we demonstrated.
  - Each person implements the rest of the assignment on their own. If further difficulties come up, meet again to solve it using pair programming.
  - You will get a joined review, during which we will discuss your implementations side by side.
- However you decided, on Thursday 24:00 (midnight) at the latest put in a pull request to the assignment repository. It's no problem if it's not finished, just submit whatever you have. If you worked together tell us whether you want to be reviewed together or separately.
- We assign each of you someone else's code to review.
- We assign each of you two time-slots: one in which you receive a review of your own code and one in which you present your review of someone else's code.
- Same plan for the second week (you can re-decide on working together or alone)