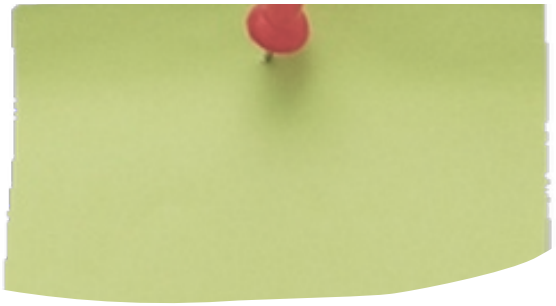


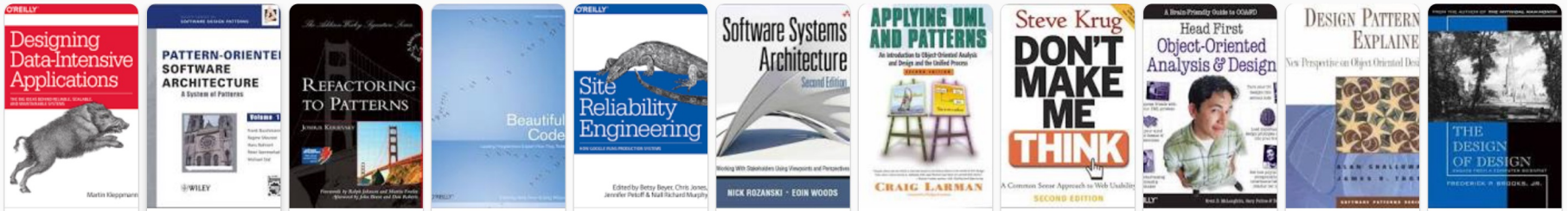
COMP 303 Winter 2022

Introduction to Software Design

Jin L.C. Guo

Why are we here?

Design Patterns: Ele… 1994 | Domain-driven design Eric J. Evan… | Code Complete Steve McCo… | Head First Design Patte… 2004 | Refactoring 1999 | Clean Architecture:… Robert Cecil… | Agile Principles, P… 2002 | Code: The Hidden Lang… 1999 | A Philosophy of Software … John Ouster… | Building Microservice… Sam Newma… | The Algorithm Design Manual Steven Skie…



Designing Data-Intensi… Martin Klepp… | Pattern-oriented Soft… 1995 | Refactoring to Patterns Joshua Keri… | Beautiful Code 2007 | Site Reliability Engineering:… 2016 | Software Systems Arc… 2005 | Applying UML and patterns Craig Larma… | Don't Make Me Think Steve Krug, … | Head First Object-Orien… Brett McLau… | Design patterns expl… Alan Shallo… | The Design of Design Fred Brooks,…

… …

Why are we here?

The Verge

GitHub and OpenAI launch an AI Copilot tool that generates its own code

GitHub and OpenAI have launched a technical preview of a new AI tool called Copilot, which lives inside the Visual Studio Code editor and...

Jun 29, 2021

TechTalks

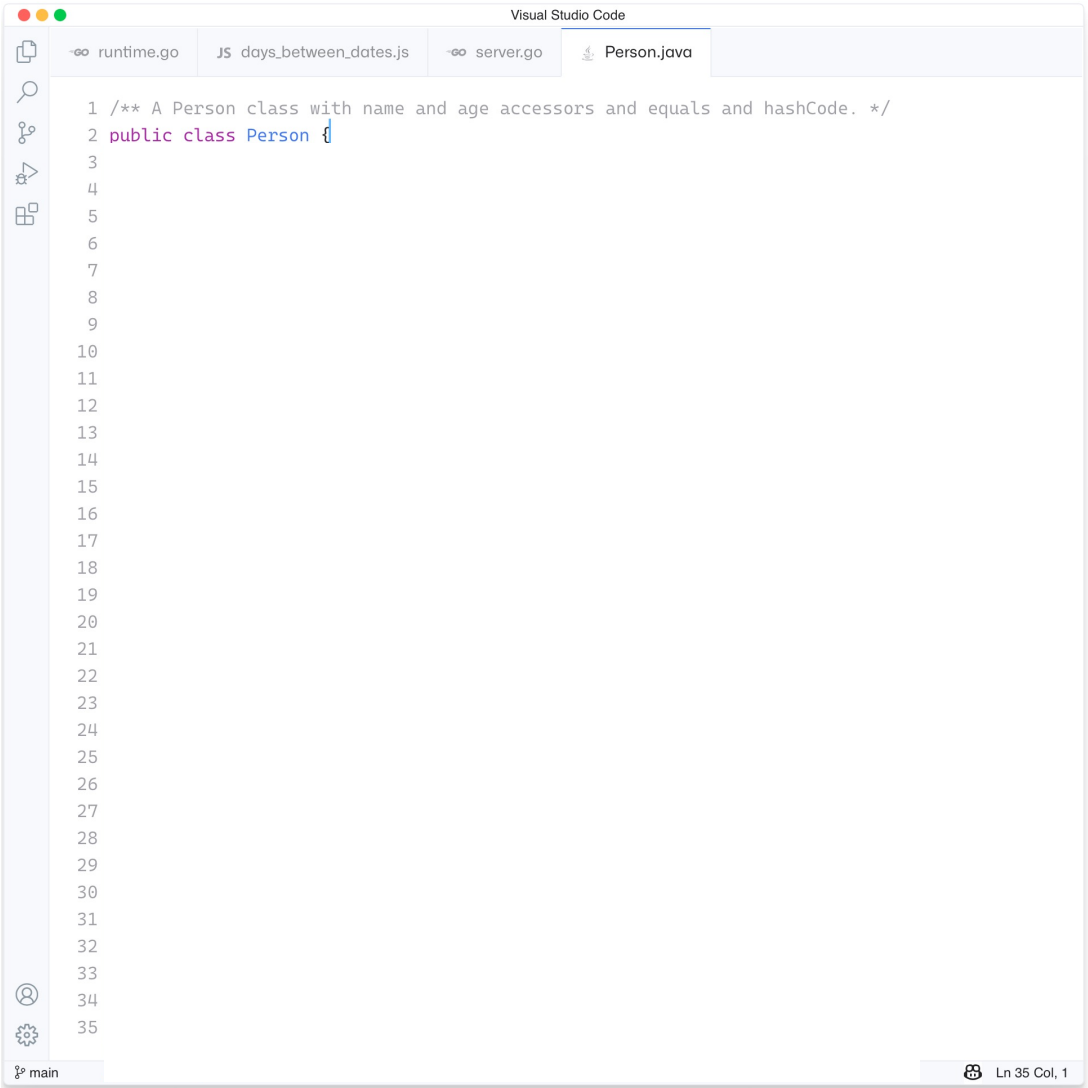
What OpenAI and GitHub's "AI pair programmer" means for the software industry

OpenAI has once again made the headlines, this time with Copilot, an AI-powered programming tool jointly built with GitHub.

Jul 5, 2021

Why are we here?

"GitHub Copilot is powered by Codex, the new AI system created by OpenAI. GitHub Copilot understands significantly more context than most code assistants. So, whether it's in a docstring, comment, function name, or the code itself, GitHub Copilot uses the context you've provided and synthesizes code to match. Together with OpenAI, we're designing GitHub Copilot to get smarter at producing safe and effective code as developers use it."



https://copilot.github.com

https://youtu.be/SGUCcjHTmGY

Activity - Software Design is about?

INTRODUCE YOURSELF TO OTHER MEMBERS IN THE BREAKOUT ROOM AND TELL HEM WHY YOU ARE HERE.

PICK AN OBJECT IN YOUR HOUSE (OR WHEREVER YOU ARE) THAT YOU CONSIDERED AS AN EXAMPLE OF GOOD DESIGN. EXPLAIN WHY.

WRITE DOWN FIVE THINGS/ACTIVITIES RELATED TO SOFTWARE DESIGN.

Good design is

Inclusive

Intuitive

Stress-free

sustainable

A problem-solver

Blends into the environment

……

Software Design is about?

Managing COMPLEXITY

Image source: https://sourcemaking.com/files/sm/images/spagett.jpg

Software Design is about?

Complexity leads to

change amplification,

cognitive load, and

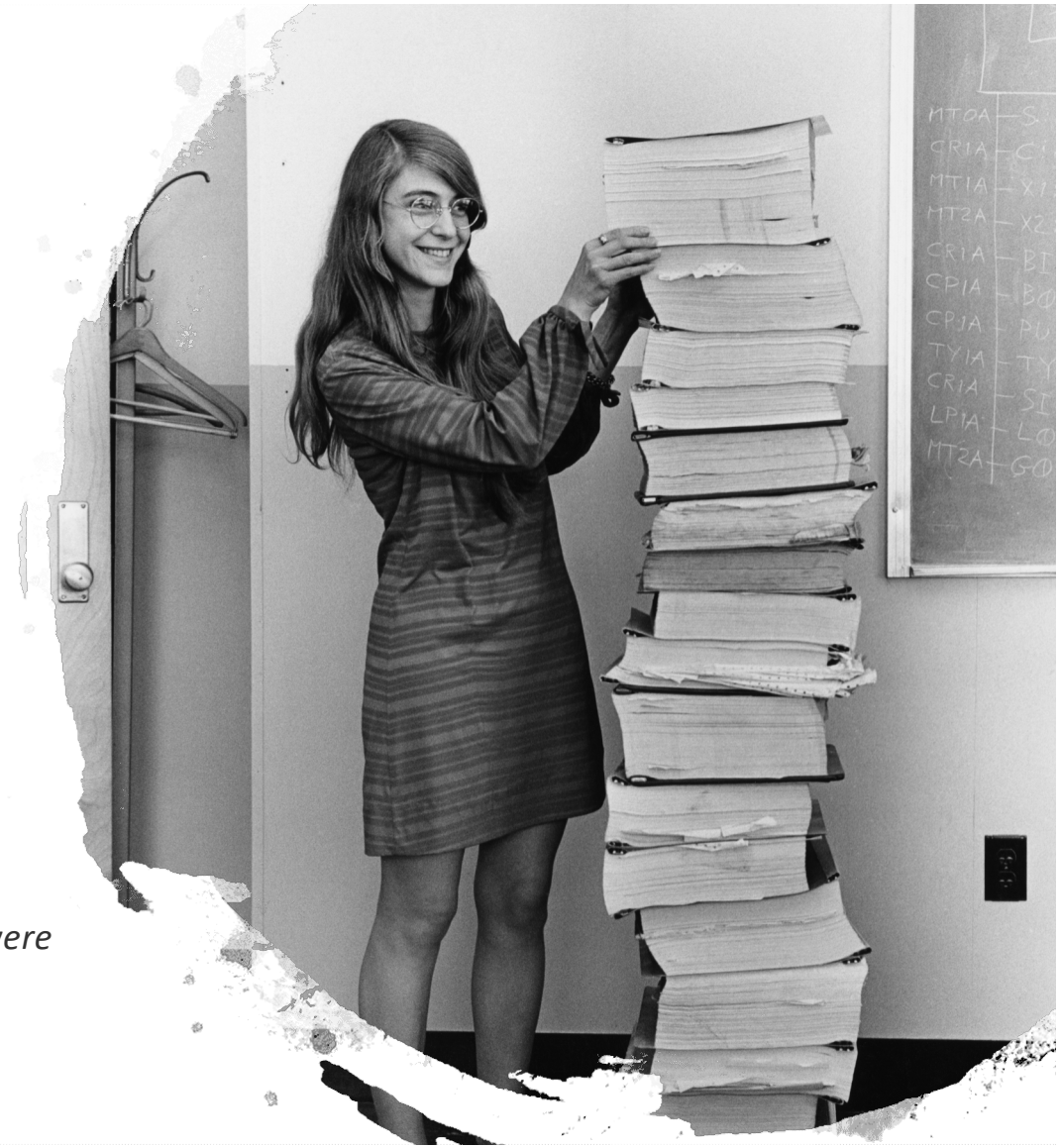
unknown unknown

Software Design is about?

Coping with imperfect world --

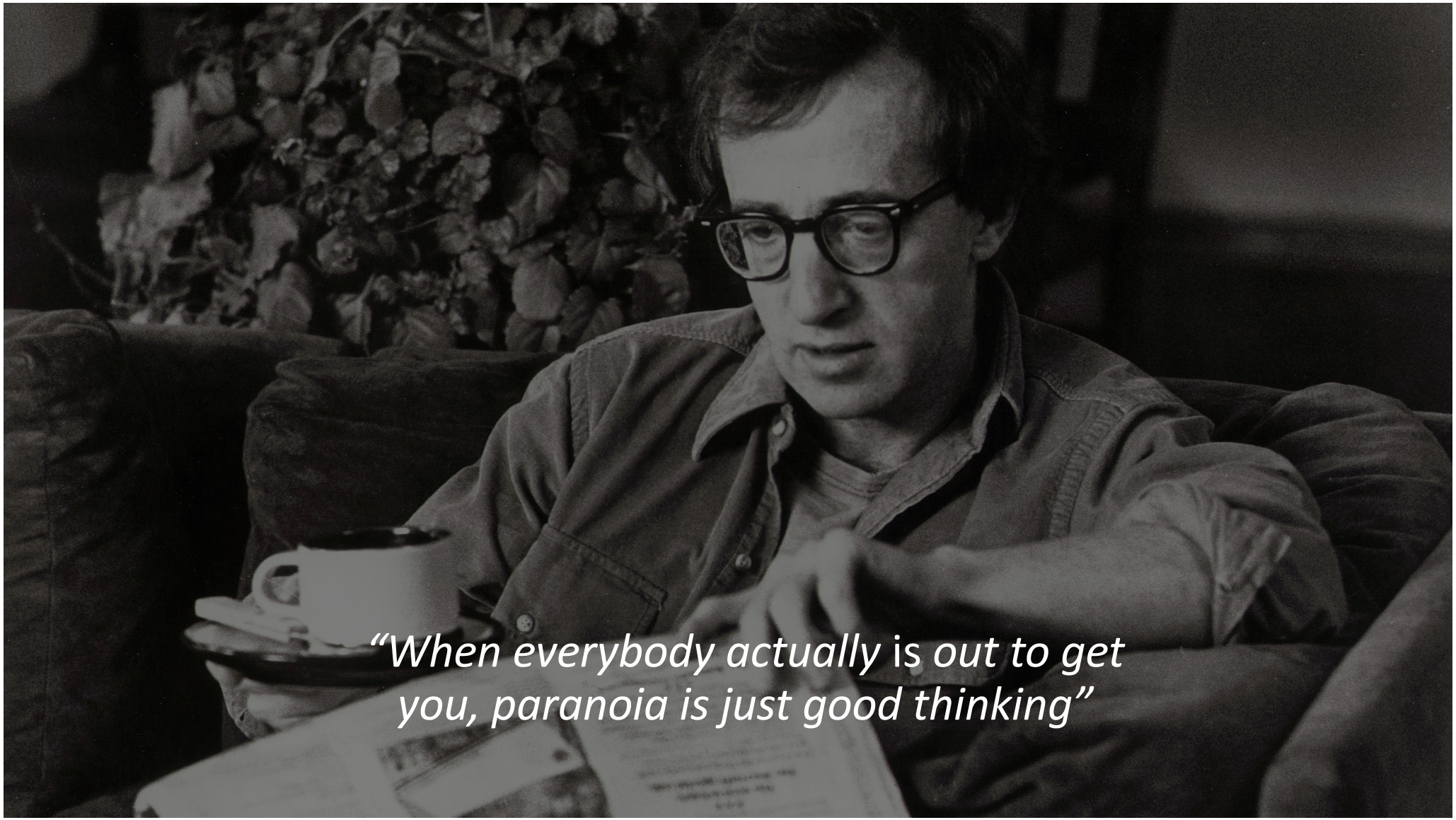
the potential defects from

- *Client*
- *Developer/User*
- *Environment*
- *Software*



NASA: "Astronauts would not make any mistakes. They were trained to be perfect."

Plenary Session by Margaret Hamilton @ ICSE 2018
https://youtu.be/ZbVOF0Uk5lU


“When everybody actually is *out to get you, paranoia is just good thinking”*

Software Design is about?

Communication with

Computers

PEOPLE

"... the ratio of time spent reading versus writing is well over 10 to 1" – Robert C. Martin

Definition of Software Design

(As a process) the construction of abstractions of data and computation and the organization of these abstraction into a working software application.

- Abstractions – variables, classes, objects, etc.
- Organization – modularized in a flexible and maintainable manner
- Working – correctly functioning (specification, testing)

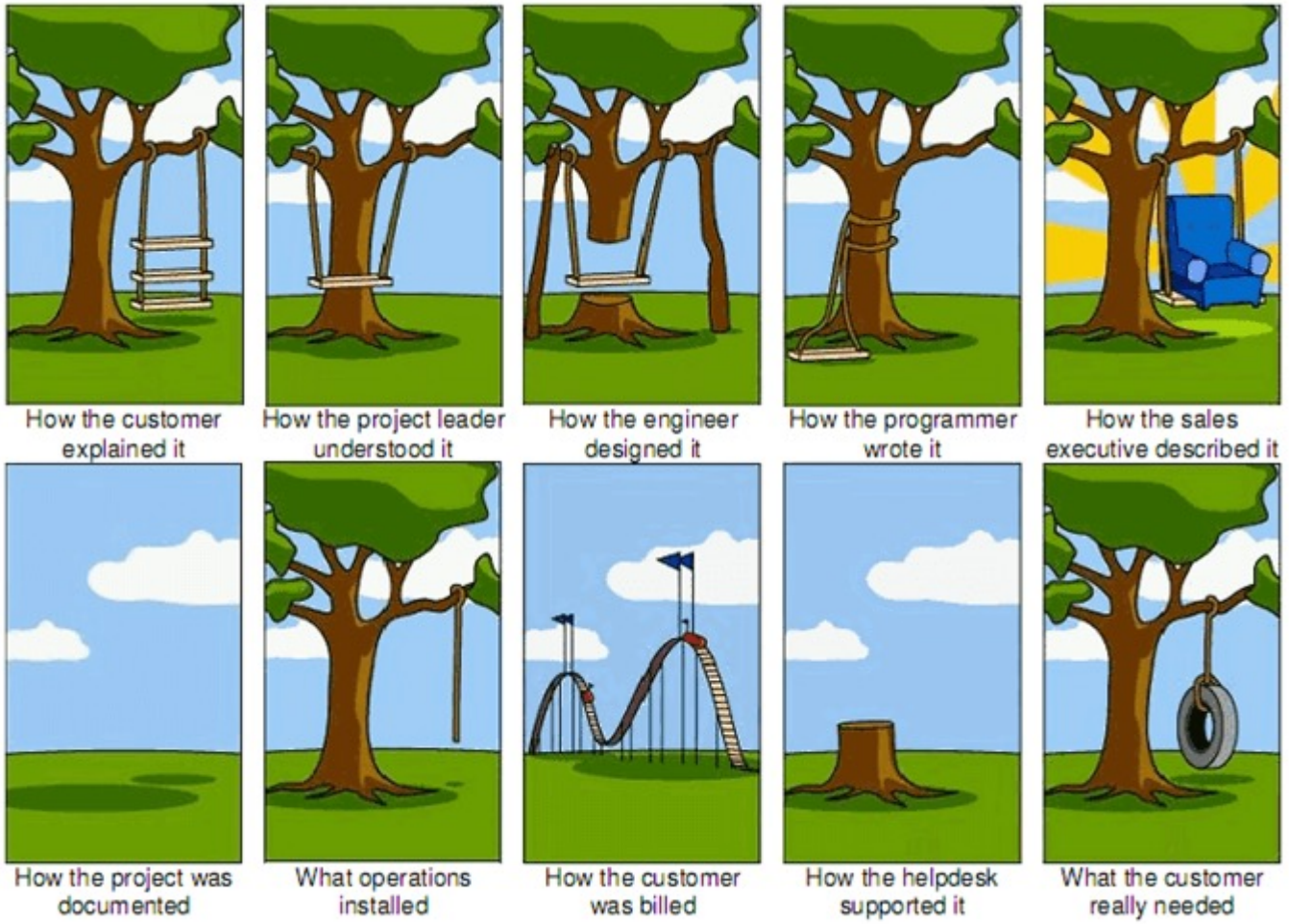


Image Source: http://tamingdata.com/wp-content/uploads/2010/07/tree-swing-project-management-large.png

Design in Software Engineering

- **NATO Software Engineering Conferences** (1968 and 1969)

  Key questions:

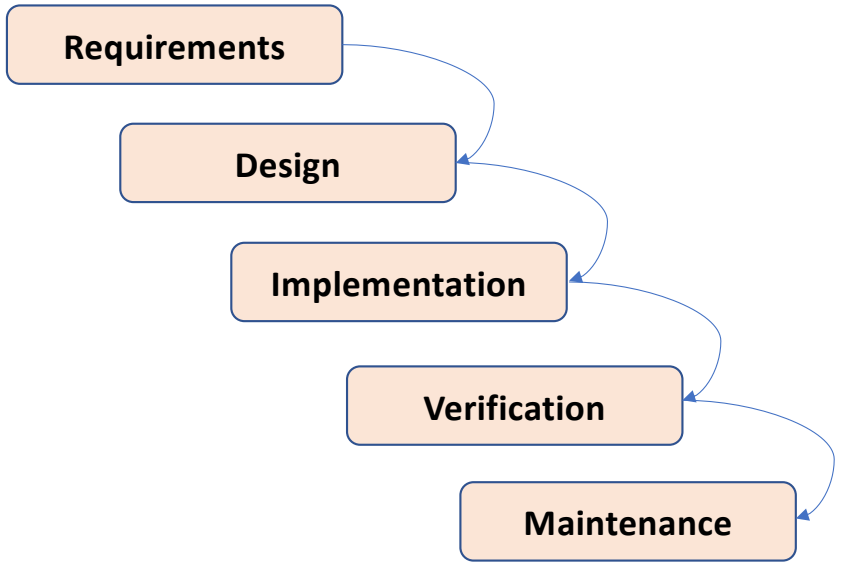
  How to keep code maintainable?

  How to satisfy extensive and changing requirements?

  How to work on code as a team?

Design in Software Engineering

Software development process



Design in Software Engineering

Software development process

Requirements

Elicit *Analyze* *Document*

Quality Attribute: **Maintainability, Portability, Reliability, Scalability, Flexibility, Auditability, Documentation, Performance, Security, Usability, …**

Image Source: https://www.outsystems.com/blog/truth-about-non-functional-requirements.html

Design in Software Engineering

Software development process

Conceptual Design

Technical Design

*Recognize the **components**, **connections**, and **responsibility** of the software product.*

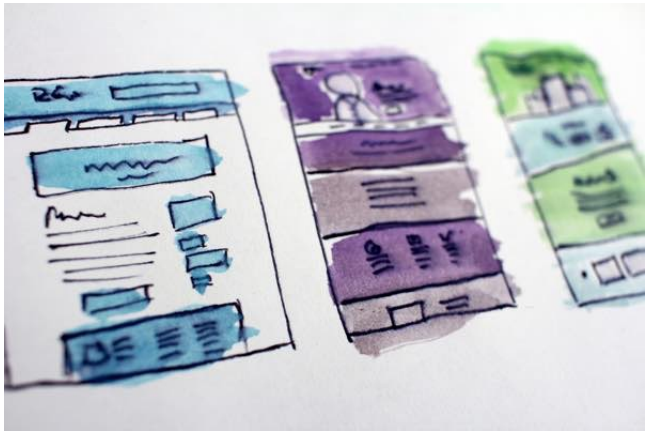


Image source: http://maryshaw.net/wp-content/uploads/ui-design-course.jpg

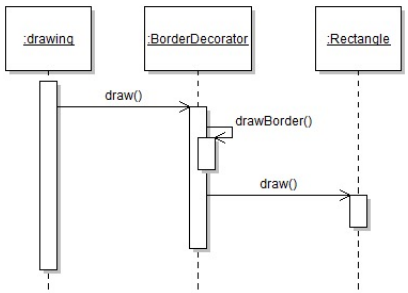
Design in Software Engineering

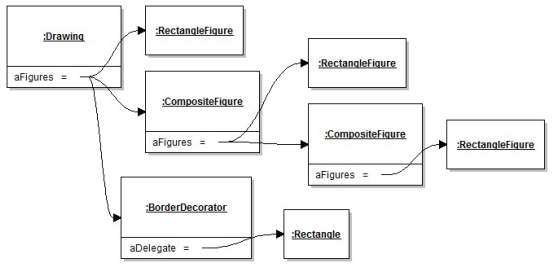
Software development process

Conceptual Design

Technical Design

describe how the responsibilities are met.



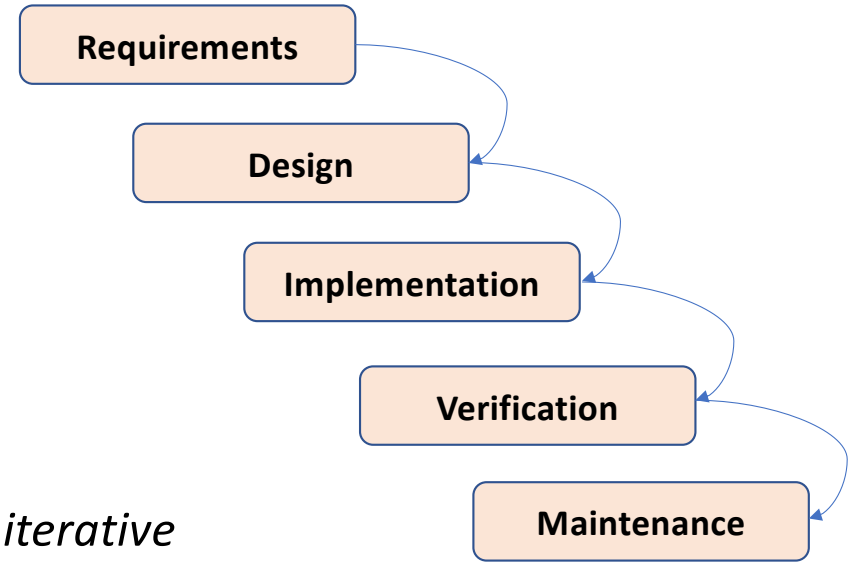


Design in Software Engineering

Software development process

Waterfall

Agile

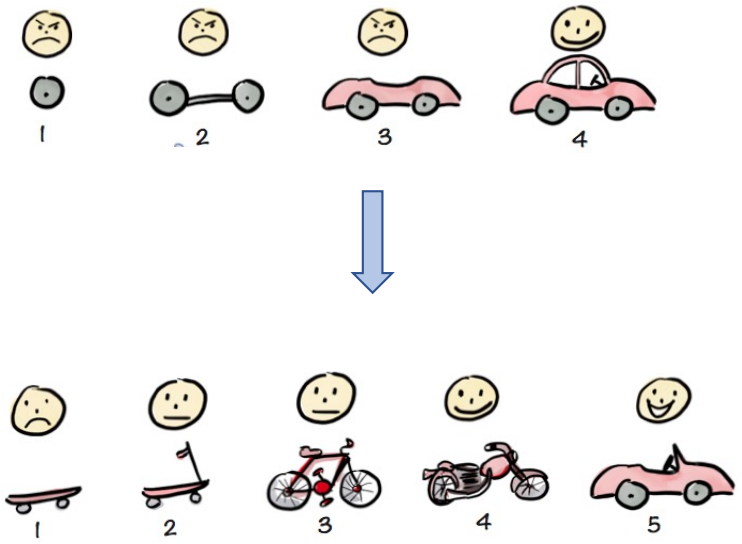


Emphasize incremental and iterative development and delivery

Agile Practices

Waterfall

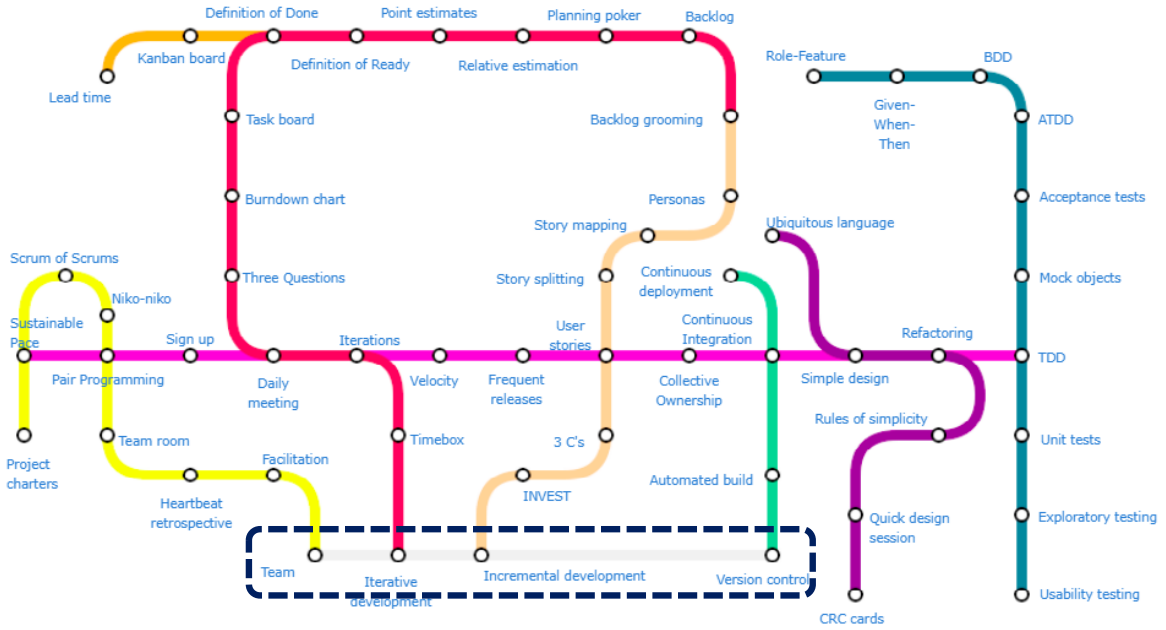
Agile



Emphasize incremental and iterative development and delivery

Image source: https://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp

Agile Practices



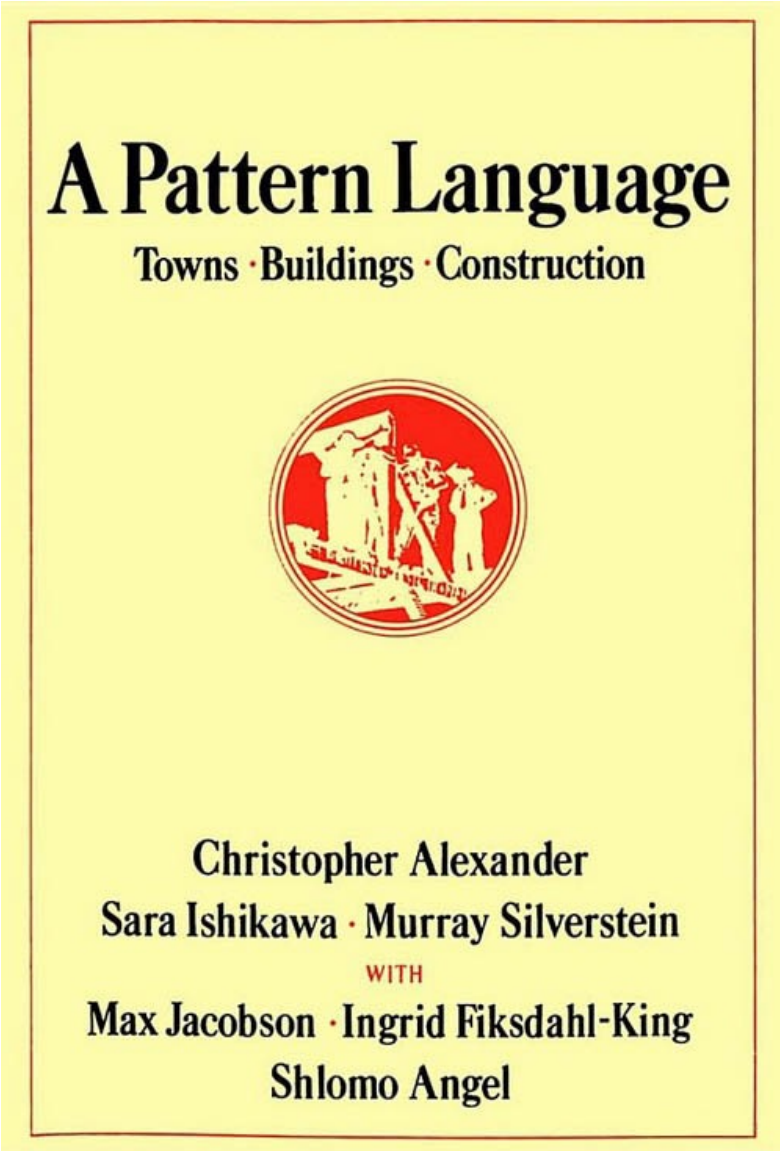
Lines represent practices from the various Agile "tribes" or areas of concern:

Extreme Programming
Teams
Lean
Scrum
Product management
Devops
Design
Testing
Fundamentals

Image source: https://www.agilealliance.org/agile101/subway-map-to-agile-practices/

Storing and Sharing Design knowledge

- Interval -- Human head
- External
  - Source code, including identifier, comments
  - Design documents, including diagrams, blog post
  - Discussion platforms and version control system
  - Design Patterns: name, problem, solution, consequences



Storing and Sharing Design knowledge

- *"The elements of this language are entities called patterns. Each pattern describes a problem that occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."* — Christopher Alexander

- How to achieve greater beauty and livability (Wholeness) – within the built environment

1 Levels of Scale

2 Strong Centers

3 Boundaries

4 Alternating Repetition

5 Positive Space

6 Good Shape

7 Local Symmetries

8 Deep Interlock

9 Contrast

10 Graded Variation

11 Roughness

12 Echoes

13 The Void

14 Inner Calm

15 Not-Separateness

Storing and Sharing Design knowledge

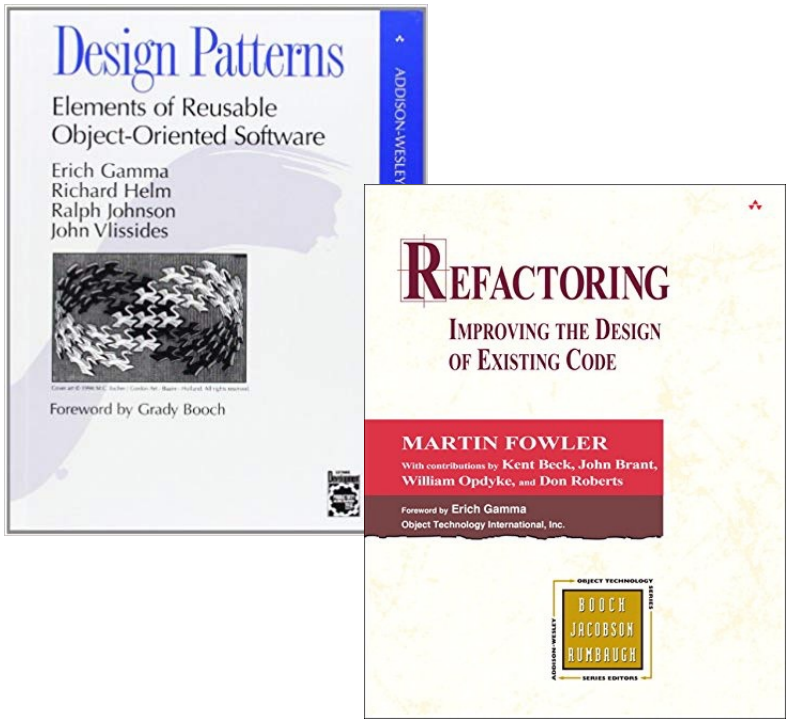
- Design pattern
  - Name
  - Problem
  - Solution template
  - Consequences

Storing and Sharing Design knowledge

- Design pattern
  - Name
  - Problem
  - Solution template
  - Consequences

- Code smell or Antipatterns
  - God classes
  - Primitive Obsession

  ……



This course is about?

Effectively *Use*

- **Programming language mechanisms**: typing, enumerated types, scopes, access modifiers, assertions, Java interfaces, interface implementation, null references, final variables, optional types, nested classes, closures, unit testing frameworks, JUnit, metaprogramming, annotations, aggregation, delegation, cloning, inheritance, subtyping, downcasting, object initialization, super calls, overriding, overloading, abstract classes, abstract methods, final classes, final methods, application framework, event loop, graphical user interface (GUI).

This course is about?

Properly *Explain* and *Apply*

- **Design Concepts and Principles**: Encapsulation, information hiding, abstraction, immutability, interface, reference sharing, escaping references, polymorphism, loose coupling, reusability, extensibility, separation of concerns, interface segregation, concrete and abstract object states, object life cycle, object identity, object equality, object uniqueness, state space minimization, unit testing, regression testing, quality attributes of unit tests, test suites, test coverage, divide and conquer, law of Demeter, code reuse, extensibility, Liskov substitution principle, inversion of control, model–view–controller (MVC) decomposition, callback method.

This course is about?

Effectively *Adopt*

- **Design Techniques**: class definition, object diagrams, immutable wrappers, reference copying, copy constructors, design by contract, decoupling behavior from implementation, interface-based behavior specification, class diagrams, function objects, iterators, state diagrams, test suite organization, use of test fixtures, testing with stubs, testing private structures, use of test coverage metrics, testing exceptional conditions, sequence diagrams, combining design patterns, inheritance-base reuse, class hierarchy design, adapter inheritance, event handling, GUI design, behavior composition, functions as data sources, interface segregation with first-class functions, pipelining, map–reduce.

This course is about?

Properly *Adopt or Identify*

- **Patterns and Antipatterns:** Primitive Obsession*, Inappropriate Intimacy*, Iterator, Strategy, Switch Statement*, Speculative Generality*, Temporary Field*, Long Method*, Null Object, Flyweight, Singleton, Duplicated Code*, God Class*, Message Chain*, Composite, Decorator, Prototype, Command, Template Method, Pairwise Dependencies*, Observer, Visitor.

Background Knowledge

- You have taken COMP 206 and COM 250;

- You are able to
  - Understand and use basic data structures (such as arrays, hash tables, trees and lists);
  - Understand the basic notions of object-oriented programming (such as objects, references, self, interfaces, and subclassing);
  - Write Java programs to solve small and well-defined problems (given the specification);
  - Use a revision control system to organize your work;
  - Use a debugger to trace through execution and inspect run-time values.

- Self Assessment

Logistics

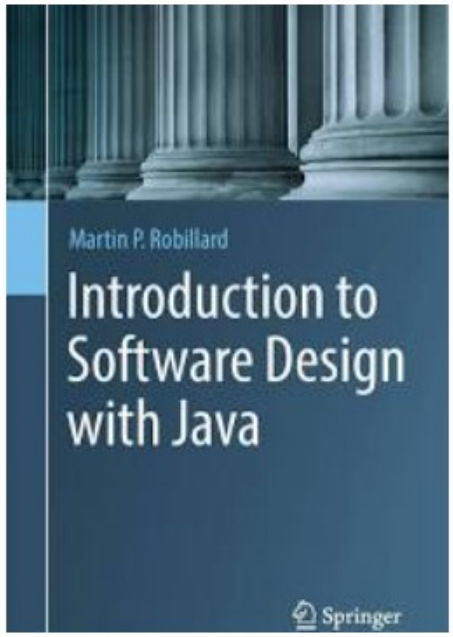
- Syllabus

- QA Forum: Ed Discussion

Textbook

- *Introduction to Software Design with Java*

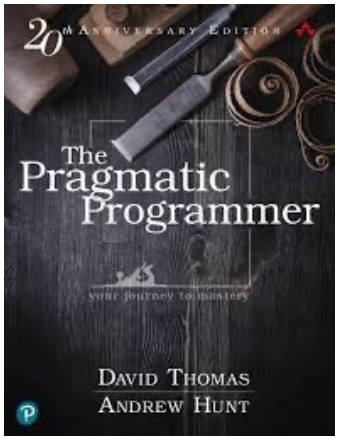
by Martin Robillard (SD)



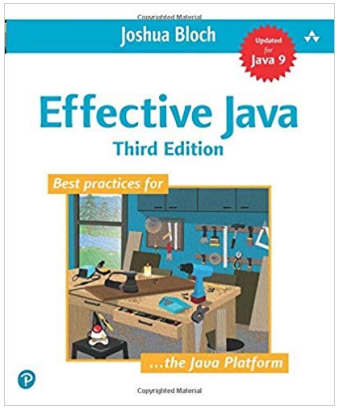
- Companion website: https://github.com/prmr/DesignBook

Other Reference Material

- *The Pragmatic Programmer* by Andrew Hunt and David Thomas, Addison-Wesley, 2019. (PP)



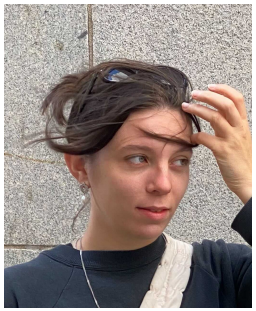
- *Effective Java* by Joshua Bloch, 2nd ed., Addison-Wesley, 2008; or 3rd ed. 2018. (EJ)



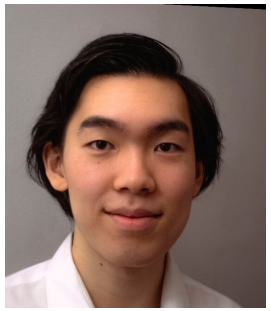
TA Team

Deeksha Arya

Avinash Bhat



Béatrice Duval



Peter Rong

Violet Shi

Jiahao Chen

Beyza Yıldırım

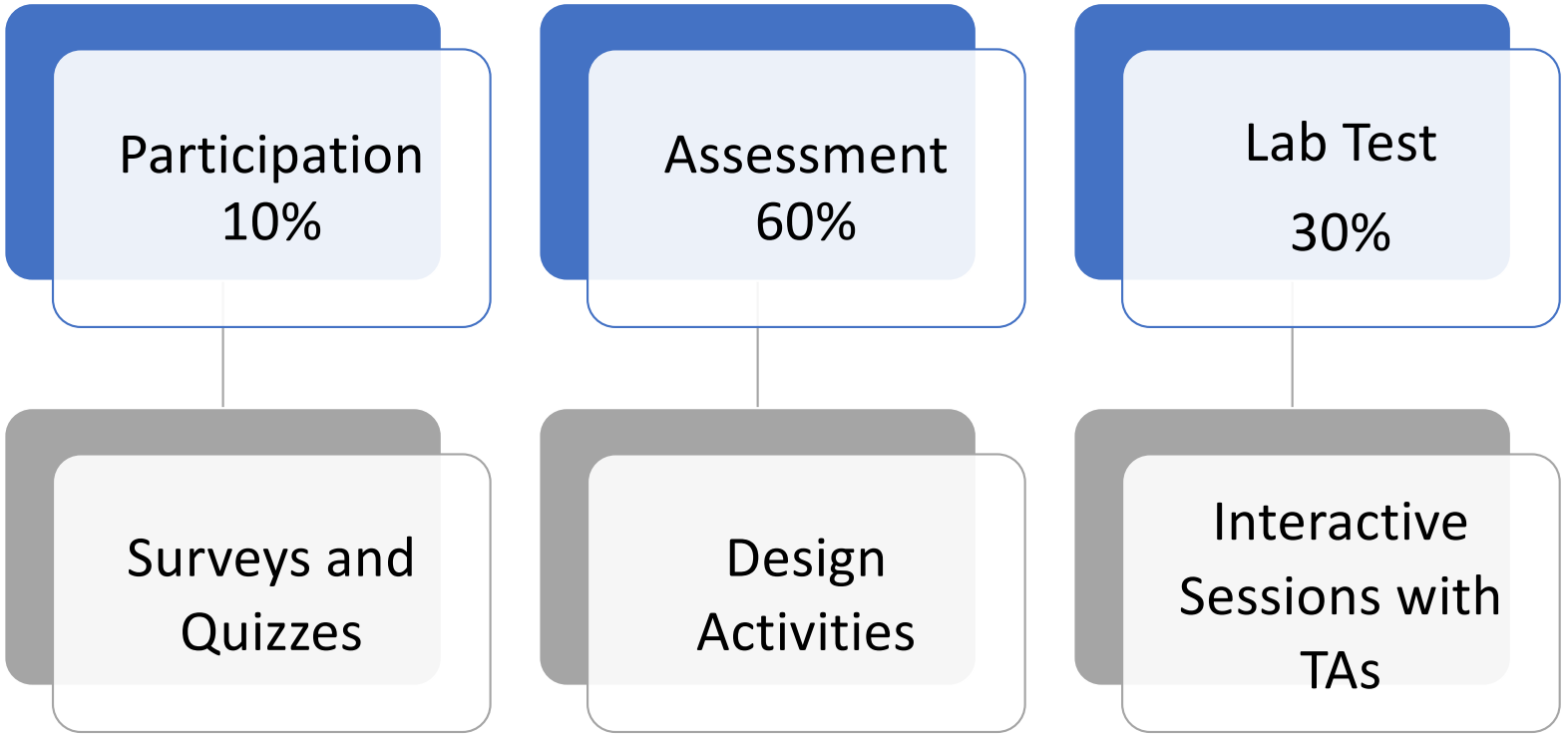
Anthony Ong

Linhui (Malinda) Huang

Shawn Hu

Xinran Xiong

Assessment and Evaluation



Ethics & Integrity

- ACM Code of Ethics and Professional Conduct

https://ethics.acm.org

- Academic Integrity

https://www.mcgill.ca/students/srr/academicrights/integrity

To-do list

- Finish the first survey
  - https://forms.gle/b3dX84Dp8h5SACPd7
- Reading:
  - ACM Code of Ethics and Professional Conduct
  - SD: Chapter 1