

Introduction to Large-Scale Software Design

17-423/723 Designing Large-Scale Software Systems

Lecture 1
Jan 17, 2024

Learning Goals

After today's lecture, you will be able to:

- Describe the importance of design in building successful software products
- Describe the risk-driven approach to software design
- Describe the generate-communicate-evaluate (GCE) paradigm to designing software

Today's Agenda

- Introduction and motivation for software design
- Course logistics
- Introductions

Why design?

What is Design?

Dictionary

Definition

verb

noun

Synonyms

Synonym Chooser

Example Sentences

Word History

Phrases Containing

Entries Near

Show More ▾

Save Word 📌+

design 1 of 2 verb

de·sign (di-'zīn ◀▶)

designed; designing; designs

[Synonyms of design >](#)

transitive verb

1 : to create, fashion, execute, or construct according to plan : **DEVISE, CONTRIVE**

| *design* a system for tracking inventory

2 a : to conceive and plan out in the mind

| he *designed* the perfect crime

b : to have as a purpose : **INTEND**

| she *designed* to excel in her studies

c : to devise for a specific function or end

| a book *designed* primarily as a college textbook

| a suitcase *designed* to hold a laptop computer

How do we construct bridges (in the real world)?

- What are key objectives of this bridge?
- Describe the process step by step



How do we construct bridges (in the real world)?

- What are key objectives of this bridge?
- Describe the process step by step

Iterative process consisting of:

- Requirements Elicitation
- Modelling
- Analysis
- Construction

How do we construct software?

- Describe the process step by step

How do we construct software?

- Describe the process step by step

Often not enough
planning, resulting in
inferior products



How do we construct software?

- Systematic and deliberate design is less common in software industry
- Temptation is to start coding right away
 - “Why bother with design? It’s fine as long as it works!”
- Even if you skip deliberate design, you will end up with an implicit or “default” design anyway
 - But most likely, the default design is not going to have desirable qualities of a successful product

What if we built bridges the way we build software?

1. Take some off-the-shelf parts and glue them together
(aka reuse)
2. Drive cars over the bridge to see if it breaks
(aka testing)
3. If it breaks, wiggle some tape around it and hope it doesn't break again
(aka hotfix)

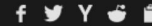
Lack of Design: Risks

- Bad designs can lead to loss of life, damage to property, or loss of money

KILLED BY A MACHINE: THE THERAC-25

by: [Adam Fabio](#)

151 Comments

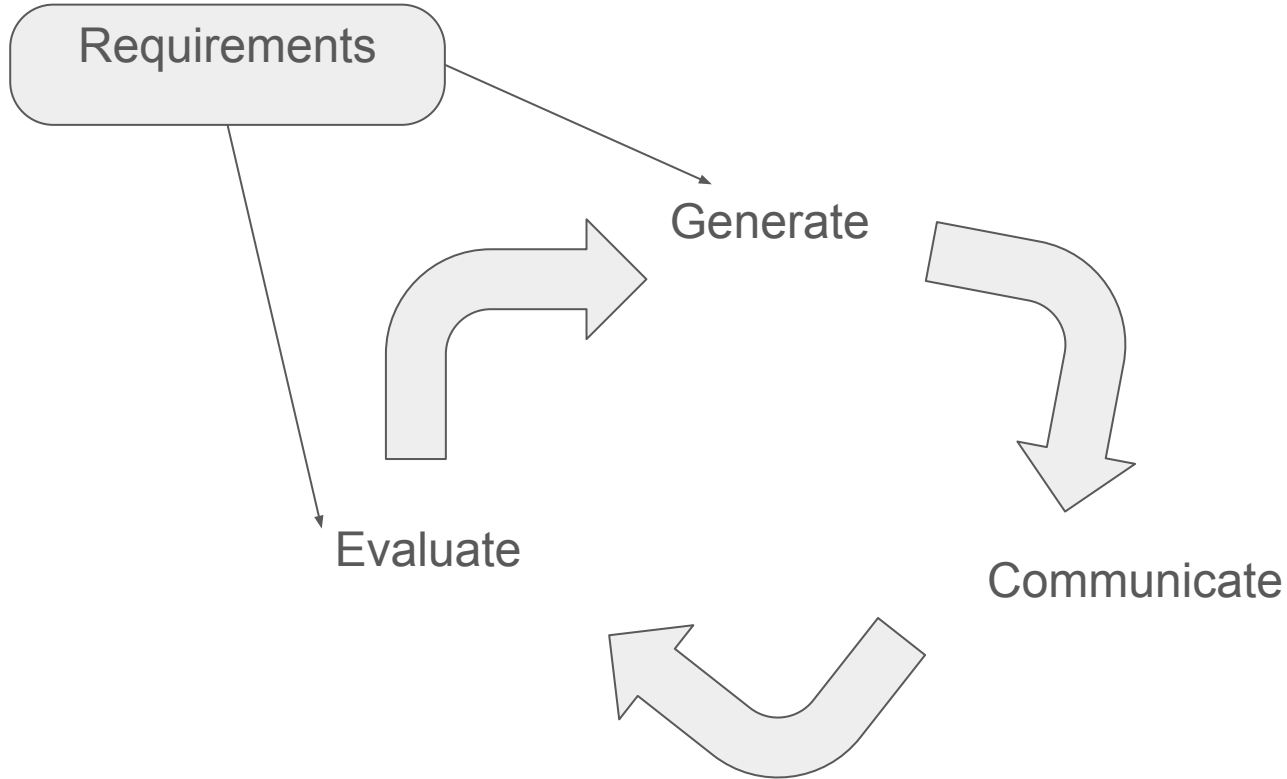


October 26, 2015



The Therac-25 was not a device anyone was happy to see. It was a radiation therapy machine. In layman's terms it was a "cancer zapper"; a linear accelerator with a human as its target. Using X-rays or a beam of electrons, radiation therapy machines kill cancerous tissue, even deep inside the body. These room-sized medical devices would always cause some collateral damage to healthy tissue around the tumors. As with chemotherapy, the hope is that the net effect heals the patient more than it harms them. For six unfortunate patients in 1986 and 1987, the Therac-25 did the unthinkable: it exposed them to massive overdoses of radiation, killing four and leaving two others with lifelong injuries. During the investigation, it was determined that the root cause of the problem was twofold. Firstly, the software controlling the machine contained bugs which proved to be fatal. Secondly, the design of the machine relied on the controlling computer alone for safety. There were no hardware interlocks or supervisory circuits to ensure that software bugs couldn't result in catastrophic failures.

How SHOULD we construct software?



Why do we still build software the “wrong” way?

- **Cost**
 - In some cases the cost of extensive analysis is actually not justified
 - In most cases however clean design is actually not more expensive
- **Misunderstanding of the Agile Manifesto**
 - Agile software development avoids up-front design due to the bad reputation of the “waterfall” model of software development
 - However, the goal should not be to have NO up-front design, just not too much up-front design

Risk-Driven Approach to Design

Why makes design hard?

Reason #1: Design is less well-understood by software engineers

- Design is often considered “art” or “talent” rather than teachable skills
- This course is a rebuff against this



Why makes design hard?

Reason #2: Requirements change frequently

- Your design may become obsolete over time
- But you **can** design systems to be ready for such changes



"The client kept changing the requirements on a daily basis, so we decided to freeze them until the next release."

Why makes design hard?

Reason #3: Tension between upfront design vs. time-to-market

- Pressure to release your product as soon as possible
- Often used as an excuse to avoid any upfront design



Why makes design hard?

Reason #3: Tension between upfront design vs. time-to-market

- Pressure to release your product as soon as possible
- Often used as an excuse to avoid any upfront design



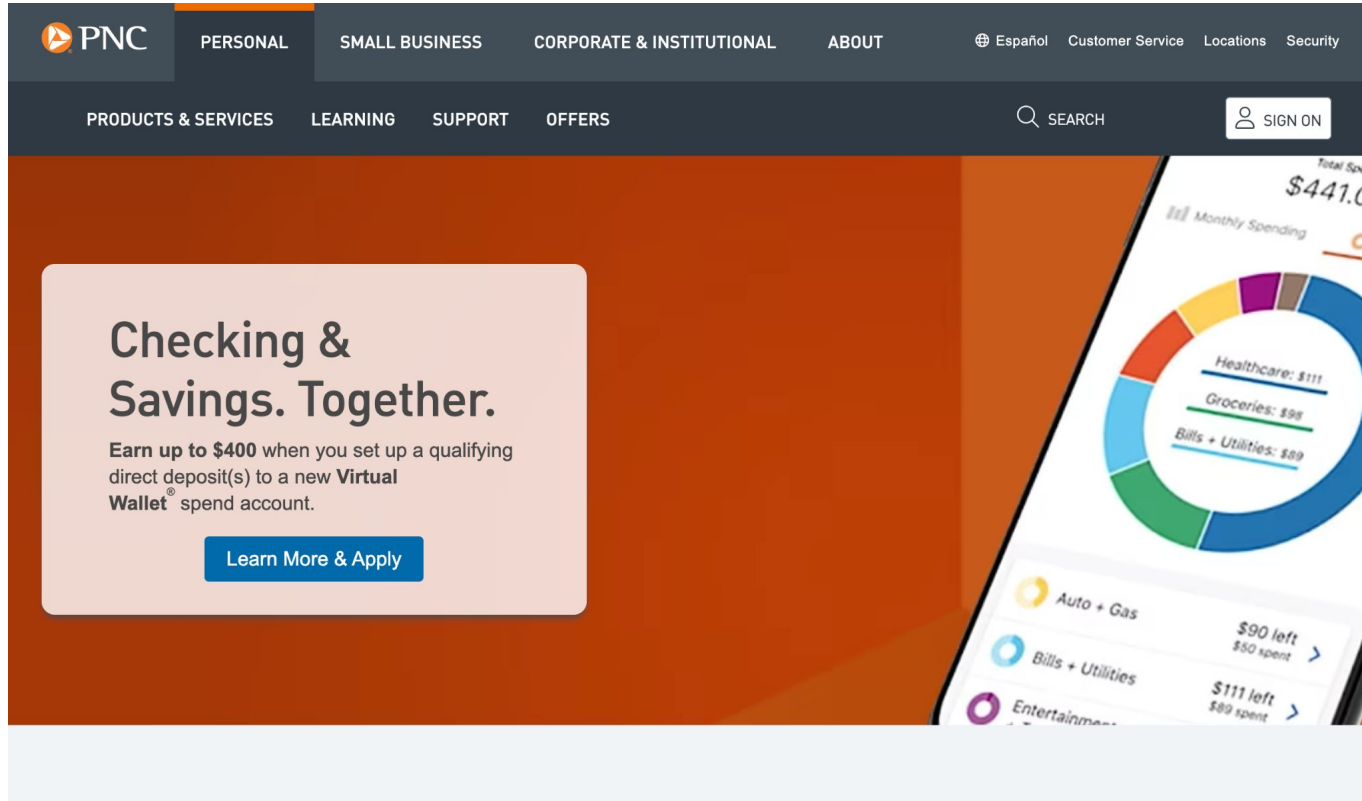
Risk-Driven Approach to Design

- What is the “right” amount of design to do?
- Risk = (cost of failure)*(likelihood of failure)
- **Ask:** What are possible costs to my organization/stakeholders if my product/system fails?

Cost of a Possible Failure?



Cost of a Possible Failure?



The image shows a screenshot of the PNC website's top navigation bar and a promotional banner. The navigation bar includes the PNC logo, menu items for PERSONAL, SMALL BUSINESS, CORPORATE & INSTITUTIONAL, and ABOUT, along with links for Español, Customer Service, Locations, and Security. Below this is a secondary navigation bar with PRODUCTS & SERVICES, LEARNING, SUPPORT, OFFERS, a search bar, and a SIGN ON button. The main banner features a dark orange background with a light beige box on the left containing promotional text and a blue button. On the right, a smartphone displays a financial dashboard with a donut chart and a list of spending categories.

PERSONAL SMALL BUSINESS CORPORATE & INSTITUTIONAL ABOUT

Español Customer Service Locations Security

PRODUCTS & SERVICES LEARNING SUPPORT OFFERS

SEARCH SIGN ON

Checking & Savings. Together.

Earn up to **\$400** when you set up a qualifying direct deposit(s) to a new **Virtual Wallet**® spend account.

[Learn More & Apply](#)

Total Spent
\$441.0

Monthly Spending

Category	Amount
Healthcare	\$111
Groceries	\$98
Bills + Utilities	\$89

Category	Left	Spent
Auto + Gas	\$90 left	\$50 spent
Bills + Utilities	\$111 left	\$89 spent
Entertainment		

Cost of a Possible Failure?



Cost of a Possible Failure?



Risk-Driven Approach to Design

- What is the “right” amount of design to do?
- Risk = (cost of failure)*(likelihood of failure)
- **Ask:** What are possible costs if my product/system fails?
 - The amount of design should be proportional to the level of risks
- Cost of failure: Examples
 - Loss of customers & revenue due to poor availability
 - Loss of customers & revenue due to poor usability
 - Extra development costs and project delays due to poor extensibility
 - Theft of sensitive information due to poor security
 - Injuries or loss of lives due to poor safety
- The goal is not to achieve a perfect product, but to identify & eliminate “bad” designs that are likely to result in high-risk failures

What does designing involve?

Types of Design

- **Conceptual design**
 - What are the key concepts in our system? What does the data model look like?
- **Functional design**
 - How are the key functionalities of the system implemented? What does the business logic look like?
- **Architectural design**
 - What are the key components in the system? What are the interfaces between components look like?
- **Interaction design**
 - How does the user interact with the system? How do we ensure that the system is easy and intuitive for them to use?
- **Performance design**
 - How do we scale the number of users? How do we ensure high service uptime?
- **Security and reliability design**
 - How do we protect the system against malicious actors? How do we make system resilient against possible failures in the network?

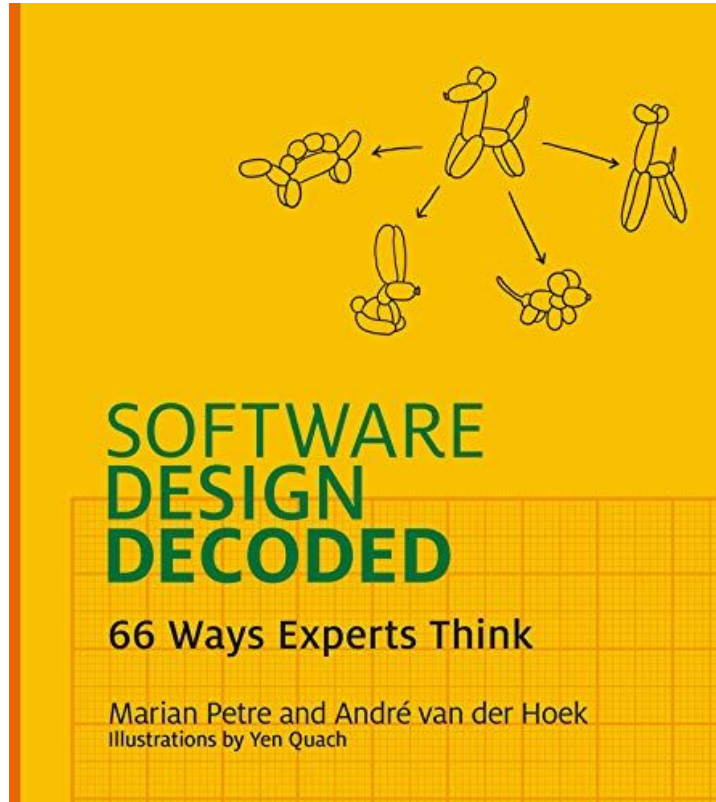
Quality Attributes

- Design is “easy” if the only goal is to build a functional system
- **Quality attributes (QAs)** are what makes design challenging
 - Often these are cross-cutting and in conflict with each other
 - Designing will frequently involve making trade-offs among these quality attributes
- QAs that we will study in this class include:
 - Extensibility
 - Reusability
 - Interoperability
 - Testability
 - Scalability
 - Robustness
 - Security
 - Usability
 - and others...

What do designers do?



What do designers do?



What do designers do?

- **Generate**, brainstorm, and explore a space of candidate design solutions



What do designers do?

- **Generate**
- **Communicate** designs to team members & clients through design sketches, documentation & prototyping

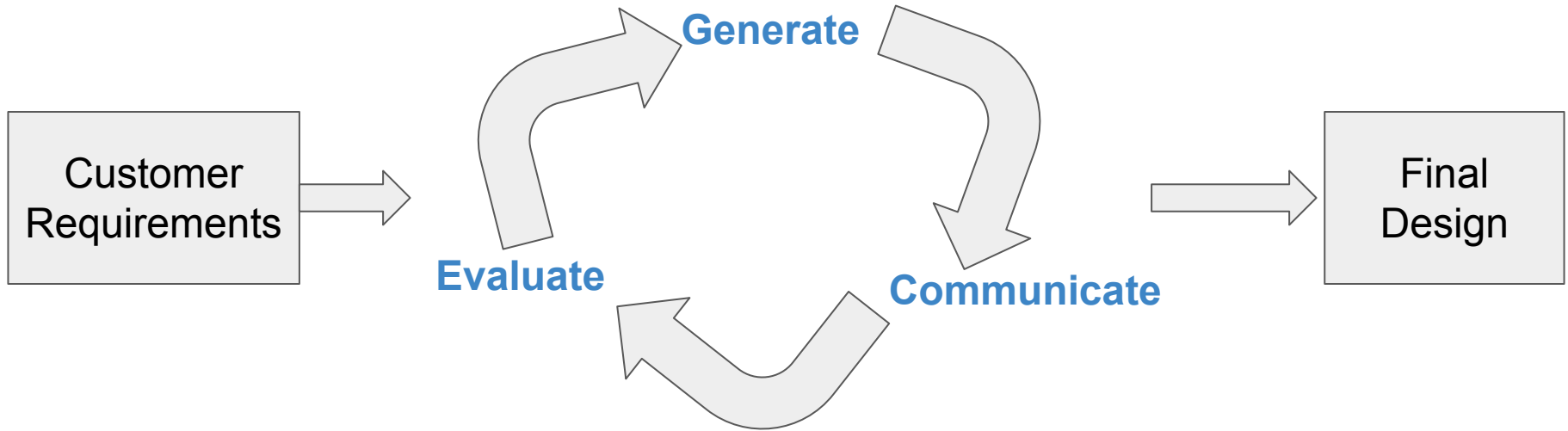


What do designers do?

- **Generate**
- **Communicate**
- **Evaluate** designs for various quality attributes & identify possible flaws

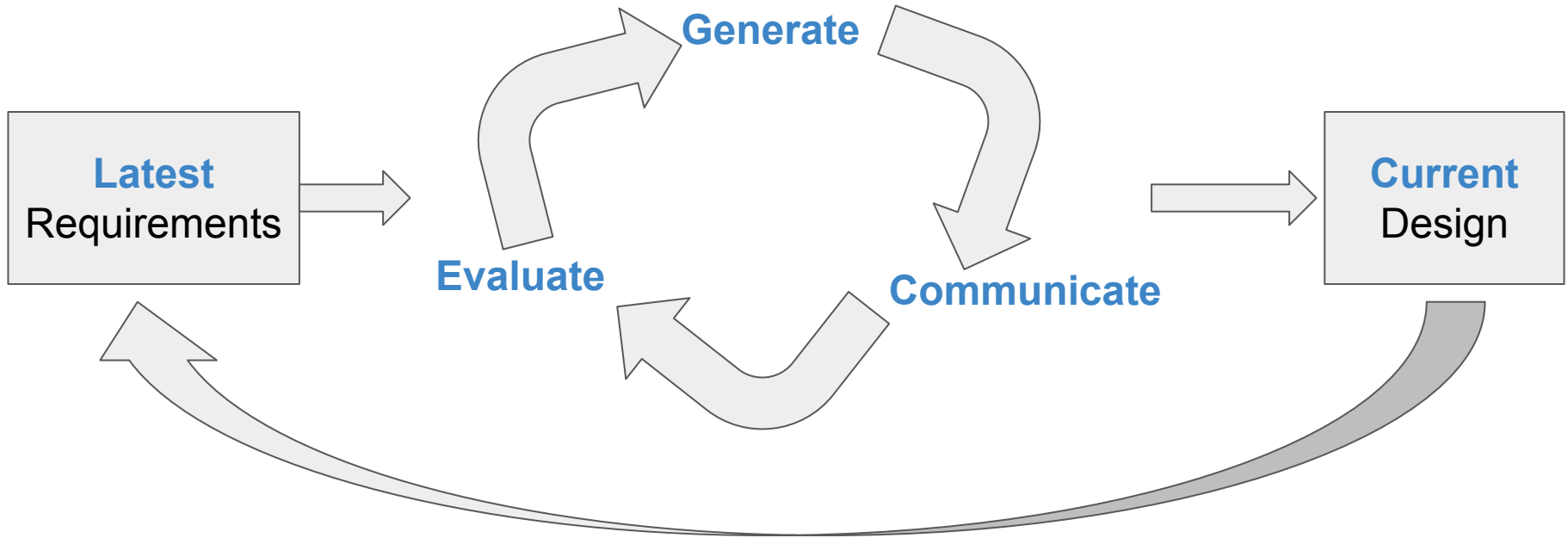


Generate-Communicate-Evaluate (GCE) Paradigm



We will cover a set of principles, techniques, and tools for generating, communicating, and evaluating designs w.r.t. various quality attributes

Generate-Communicate-Evaluate (GCE) Paradigm



Design is never “finished”; it’s a continuous, iterative process!

Designers learn from failures



Throughout the class, we will use case studies of software failures (as well as successes) to extract lessons and design principles

Why take this course?

- Develop skills for a career as a system architect/designer
 - “System design” questions are common in interviews for software engineering positions
- Role of AI/LLMs in software development
 - Coding will be more and more automated
 - Role of software engineers will change & involve making design decisions
 - Skills from this course will become increasingly more valuable
- For PhD & research-oriented students:
 - Software design is still relatively less understood by researchers
 - This course will discuss some open questions and opportunities in software design research

What do we mean by “large-scale”?

- Not just in scalability in the number of users & requests, but also in terms of:
 - Complexity in the problem domain
 - Diversity in functional requirements and quality attributes
 - Number of interacting subsystems and components
 - Period of system evolution over time (potentially indefinite)

Course Logistics

Course Staff



Co-instructor

Tobias Dürschmid
TCS 413
tdurschm@andrew



Co-instructor

Eunsuk Kang
TCS 322
eunsukk@andrew



TA

Parv Kapoor
TCS 413
parvk@andrew

Communication

- Email us or ping us on Slack (invite link on Canvas)
- Post questions on Slack
- All announcements through Slack #announcements
- Submissions through Canvas & Gradescope
- Other non-public materials (readings) on Canvas
- Please use #general or #assignments and post publicly if possible; your classmates will benefit from your Q&A!
- Lecture recordings: We will provide them per request

Disclaimer: New Class

- Expect rough edges & moving parts over the semester
- Please be flexible & patient!
- We welcome your candid feedback! Let us know:
 - Topics that you'd like to us cover in more/less detail
 - Concepts that could be better explained
 - Tools/techniques that you'd like us to cover
 - An assignment that takes too much (little) time or is too hard (easy)
 - Readings that are too boring/obscure/hard to follow
 - And anything else, really!
- You have a chance to shape the future offerings of this course!

Course Learning Goals

After taking this course, you will be able to:

- Design software systems for various quality attributes, including reusability, extensibility, interoperability, robustness, scalability, testability, security, usability
- Explain how to adapt a software design process to fit different domains, such as robotics, web apps, mobile apps, and medical systems
- Identify, describe, and prioritize relevant requirements for a given design problem
- Generate viable design solutions
- Apply appropriate abstractions & modeling techniques to communicate and document design solutions
- Evaluate design solutions based on their satisfaction of common design principles and trade-offs between quality attributes

Course Philosophy

- **Hands-on Experience in a Collaborative Project**
 - Maybe the largest project you have worked on so far!
- **Growth Mindset & Learning from Failures**
 - Learning from real-world case studies
 - Learning from your own mistakes in the project
- **Active Student Participation**
 - We encourage you to ask questions and participate in class discussions
 - Wrong answers support learning! (see growth mindset)

This is a Software Engineering Class!

- Focused on engineering judgment
- Arguments, trade-offs, and justification, rather than a single correct answer
- The answer will often be: "It depends..."
- Practical engagement, building systems, testing, automation
- Strong teamwork component

Pedagogical Principles in this Course

- Spaced Practice / Interleaving of Topics
 - Practicing newly learned material is most effective when you started to forget about it
 - Based on educational research, topics won't be covered in a single lecture, but spread throughout the course
 - Connections between different topics will become more clear this way!
 - Active Learning
 - Lectures are structured with many in-class discussions, think-pair-share, and other activities
 - Research shows that active learning is more effective, when when it doesn't feel like that
- See <https://www.pnas.org/doi/10.1073/pnas.1821936116>

Recitations

- Cover hands-on topics, such as tools, frameworks, best practices, additional exercises to supplement concepts from the lectures
- First recitation this Friday: Teamwork issues & guidelines
 - Important for your project!
- Participation in recitations will also be considered as part of “Class Participation” grade (see later on Grading)

Grade Breakdown

- 20% Homeworks
- 50% Project
- 20% Midterm & final (open book)
- 10% Class participation
 - In-class discussions
 - Exit tickets

Specification Grading

- Goal: Clearly communicate expectations about course deliverables
- We will try to be as explicit as possible about what we expect
- Every assignment/project milestone will be broken down into a set of questions
- Each question will be assigned points and be graded **Pass/Fail**
 - No partial points!
- If your answer satisfies the specification, you will receive Pass

Participation: Exit Tickets

- Goal: Recall and summarize the key ideas from the lecture
 - Also to help us understand how well we conveyed those ideas
- In the last 5 min of each class, we will ask you to answer a couple of quick questions about that day's content (on Canvas)
- Not graded on correctness; any on-topic & “reasonable” answers will be accepted
- 3 free passes throughout the semester
 - Let us know if you have any exceptional circumstances (illness, travel, interviews, etc..)

Team Project

- Goal: Gain experiences designing, implementing, and iteratively improving a complex software system
- Six milestones over the semester
 - M1: Initial design and specification
 - M2: Initial prototype implementation
 - M3: Iterative design for extensibility
 - M4: Design review & critique
 - M5: Iterative design for scalability
 - M6: Final design report & presentation
- More details about the project coming next week

Individual Homeworks

- Goal: Practice applying design principles and techniques not covered by the project milestones
- We expect that these will take no more than 2~3 hours to complete

Grading: Tokens

- Individual & team tokens: 7 each
- Use 1 token to submit a homework assignment/project milestone 1 day late
- Use 3 tokens to redo a homework/milestone
- Unused individual tokens at the end of the semester will count towards your participation grade

Participation: In-class Discussions

- Most lectures will involve case studies & discussions
- Please don't hesitate to contribute your ideas and experiences!

Remember, there's no one "correct" answer to problems in this class

Use of Generative AI

- You are free to use generative AI (e.g., ChatGPT) for homeworks/project
 - We are interested to explore the potential utility of LLMs for software design!
 - Generate design alternatives, generate documentation, synthesize code given a specification, etc.,
- It's your responsible to check the quality of the output from an LLM
 - These models will sometimes hallucinate and generate superficial, bogus output
- In your submissions, clearly document how you've used these tools

Slack Introductions

- Before Friday's recitation, introduce yourself on #introduction channel:
 - Your (preferred) name
 - In 1~2 sentences, your software engineering background and goals (e.g., coursework, internships, work experience)
 - Your favorite programming languages, tools, or frameworks
 - One topic you are particularly interested in learning during this course?
 - A hobby or a favorite activity outside school

Summary

- Systematic design is common in many engineering disciplines.
- Decisions to invest in design should be driven by the amount of risks in the product or system.
- Designers generate, communicate, and evaluate design solutions.
- Designing is a continuous, iterative process.