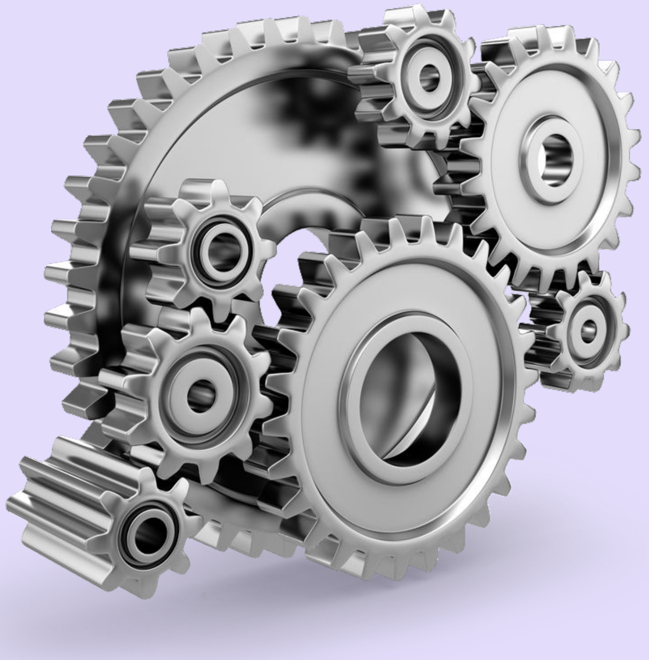
Lecture 31

Software Engineering II



CS61B, Spring 2024 @ UC Berkeley

Background

Dominic Conricode (he/him)

- Senior majoring in CS
- 5th Semester on staff
- Thinks space is cool

Vanessa Teo (she/her)

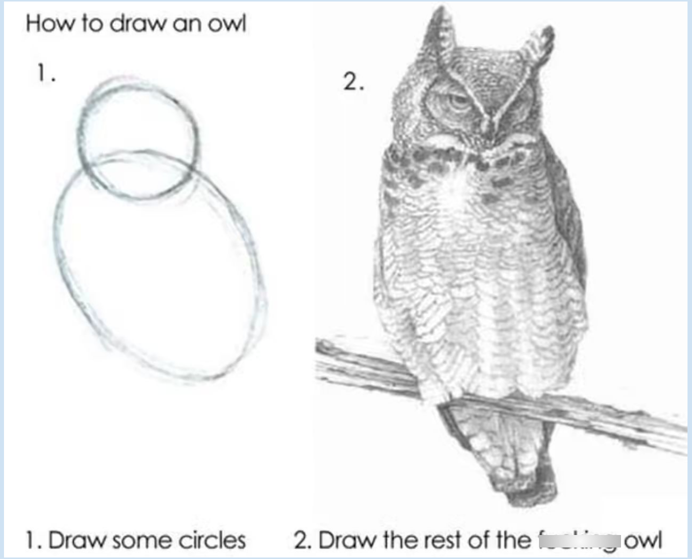
- Junior majoring in CS + Environmental Econ & Policy
- 4th semester on staff
- Has two cats

Today's lecture will be about the building a project from the ground up. The concepts we'll discuss today can be used right now! They'll also apply forever :)

General Q&A will be at the end.

Credit for many of these slides goes to:

- Josh Hug
- Stella Kaval
- Noah Adhikari
- (and us)



Design

Lecture 31, CS61B, Spring 2024

Why Design?

"Instead of wasting time by designing my program, I can save time by starting to code now" - Person who will regret their decision soon™

Design is an important first stage in development:

- Ensures consideration of all requirements and constraints
- Considers multiple alternatives and their advantages / disadvantages
- Identifies potential costs or pitfalls while change is cheap
- Allows multiple minds to weigh in and leverages diverse perspectives

Design Documents

Design documents do not need to follow a specific form, but they should give the reader easily accessible information about your project.

A design document should:

- Quickly bring the reader up to speed
- Include what alternatives were considered and why the trade-offs were made
- Be just as much about the "why" as it is about the "how"
- Update as the project progresses (be alive!)

Design Documents

Often, design documents will be reviewed before work on the project even begins.

When onboarding to an existing project, the first question someone will ask is "where's the design doc?"

Chromium: https://www.chromium.org/developers/design-documents/

Design in Project 3

Think about each class you want to create and how they will fit together

- Putting everything into one file will make you sad :(
- Ideally each class should have one role

Consider what methods are required to meet each class's functionality

The major algorithms you'll need to think about are room and hallway generation

You are making something new!

Managing Complexity

Lecture 31, CS61B, Spring 2024

Who Cares?

- You probably don’t want your work to end up like Space Cadet Pinball

Technical Debt

- Your work will be available to other people.
  - People are going to build off of it, or at least see it.
    - Maybe this is your future self!
  - Ideally, it should be flawless and well-documented.
- Deadlines exist.
  - School deadlines, grants, conferences, reviews, etc.
  - Prototypical, hastily-written code is inevitable.

There is pressure to achieve both quality *and* quantity.

This is inarguably a trade-off—prioritizing one will negatively affect the other.

This doesn’t just apply to code—it applies to life.

How do you deal with this? See Strategic vs Tactical Programming from SWE I.

Importance

- What is “good code”? Three main goals:
  - Readability: The code is easy to read and understand.
  - Simplicity: The solution is straightforward, without unnecessary complexity
  - Testability: The code is easy to test and debug.
- Why is good code important?
  - Facilitates easier maintenance and debugging
  - Enhances collaboration and code readability
  - Directly impacts the success and scalability of project

Bad Style

- Poor and Inconsistent Formatting
    - Multiple styles mixed in the same project
    - `snake_case, camelCase, SCREAMING_SNAKE_CASE, flatcase, kebob-case`

- Over-Complexity
    - simple tasks are over-complicated by unnecessary steps or convoluted logic
    - repetitive code that re-implements the same logic

IMPORTANT NOTE!

Refactoring is changing your code to make things more organized without affecting functionality.

Before refactoring, always **add and commit your code**! This way, you can restore your old work if necessary.

Bad Code Example - What’s wrong with this?

```java
static void print3DArray(String[][][] arrays) {
   System.out.print("[");
   for (int i = 0; i < arrays.length; i++) {
       if (arrays[i] == null) {
           continue;
       }
       System.out.print("[");
              for (int j = 0; j < arrays[i].length; j++) {
                     if (arrays[i][j] == null) {
                            continue;
                     }
           System.out.print("[");
           for (int k = 0; k < arrays[i][j].length; k++) {
               System.out.print(arrays[i][j][k]);
                     if (k < arrays[i][j].length - 1) {
                            System.out.print(", ");
                     }
           }
              System.out.print("]");
           if (j < arrays[i].length - 1) {
               System.out.print(", ");
           }
       }
       System.out.print("]");
              if (i < arrays.length - 1) {
                  System.out.print(", ");
              }
   }
   System.out.print("]\n");
}
```

- I have no idea what’s going on
    - Probably prints 3D arrays?

Bad Code Example - Test to verify correctness as you refactor

```java
public static void main(String[] args) {
   String[][][] arrays = new String[][][] {
       new String[][] {
           new String[] {"1", "2", "3"},
           new String[] {"4", "5", "6"},
           new String[] {"7", "8", "9"}
       },
       null,
       new String[][] {
           new String[] {"10", "11", "12"},
           new String[] {"13", "14", "15"},
           new String[] {"16", "17", "18"}
       }
   };
   print3DArray(arrays);
}
```

- Small test
  - `[[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [[10, 11, 12], [13, 14, 15], [16, 17, 18]]]`
  - Hey, it works! Why change it?

Bad Code Example - Use the tools of the language

```
static void print3DArray(String[][][] arrays) {
   System.out.print("[");
   for (String[][] array : arrays) {
       if (array == null) {
           continue;
       }
       System.out.print("[");
       for (String[] arr : array) {
           if (arr == null) {
               continue;
           }
           System.out.print("[");
           for (String a : arr) {
               System.out.print(a + ", ");
           }
           System.out.print("], ");
       }
       System.out.print("], ");
   }
   System.out.print("]\n");
}
```

- Enhanced-for loops exist
- Still bad, but much better, even with terrible variable names
  - Small test
    - `[[[1, 2, 3][4, 5, 6][7, 8, 9]][[10, 11, 12][13, 14, 15][16, 17, 18]]]`
    - Hey, you broke it! Change it back!

Bad Code Example 2 - Avoid “goto” statements where possible

```
static void print3DArray(String[][][] arrays) {
   System.out.print("[");
   for (String[][] array : arrays) {
       if (array != null) {
           System.out.print("[");
           for (String[] arr : array) {
               if (arr != null) {
                   System.out.print("[");
                   for (String a : arr) {
                       System.out.print(a + ", ");
                   }
                   System.out.print("], ");
               }
           }
           System.out.print("], ");
       }
   }
   System.out.print("]\n");
}
```

- In general, this is a good idea, but it didn’t help much here…
  - Braces and indentation are even worse than before
- Also, the code is still broken :(

Bad Code Example - Separate logic into simple methods

```java
static void print3DArray(String[][][] arrays) {
   System.out.print("[");
   for (String[][] array : arrays) {
       print2DArray(array);
       System.out.print(", ");
   }
   System.out.print("]\n");
}
static void print2DArray(String[][] array) {
   if (array != null) {
       System.out.print("[");
       for (String[] arr : array) {
           print1DArray(arr);
           System.out.print(", ");
       }
       System.out.print("]");
   }
}
static void print1DArray(String[] arr) {
   if (arr != null) {
       System.out.print("[");
       for (String a : arr) {
           System.out.print(a + ", ");
       }
       System.out.print("]");
   }
```

- Much better!
- Our test still doesn't work.

  -
    ```
    [[[1, 2, 3, ], [4, 5, 6, ], [7, 8, 9, ], ],
    [[10, 11, 12, ], [13, 14, 15, ], [16, 17, 18,
    ], ], ]
    ```
  - Weird comma behavior for the last element of arrays.
    - Notably, we have this problem for every nested array
- We have identical print statements for the commas and square brackets everywhere, with similar issues for all of them.
  - Something like generic types would probably come in handy here, but these methods are **`static`**…

Bad Code Example - Code generically; don’t repeat yourself

```java
static void print3DArray(String[][][] arrays) {
   printCommaSeparated(arrays);
   System.out.print("\n");
}
static void print2DArray(String[][] array) {
   printCommaSeparated(array);
}
static void print1DArray(String[] arr) {
   printCommaSeparated(arr);
}

static <T> void printCommaSeparated(T[] arr) {
   if (arr != null) {
       System.out.print("[");
       for (int i = 0; i < arr.length; i++) {
           System.out.print(arr[i]);
           if (i < arr.length - 1) {
               System.out.print(", ");
           }
       }
       System.out.print("]");
   }
}
```

- You actually can have generic static methods in Java.
- `[[[Ljava.lang.String;@e9e54c2, null, [[Ljava.lang.String;@65ab7765]`
    - ?!

Bad Code Example - Don’t reinvent the wheel

```
static void print3DArray(String[][][] arrays) {
   printCommaSeparated(arrays);
   System.out.print("\n");
}
static void print2DArray(String[][] array) {
   printCommaSeparated(array);
}
static void print1DArray(String[] arr) {
   printCommaSeparated(arr);
}

static <T> void printCommaSeparated(T[] arr) {
   if (arr != null) {
       System.out.print(Arrays.toString(arr));
   }
}
```

- `[[[Ljava.lang.String;@e9e54c2, null, [[Ljava.lang.String;@65ab7765]`
  - This occurs due to the nested-ness of our problem.
  - This is actually pretty annoying to resolve - an exercise for the brave amongst you is to fix it without using the library solution.

Better Code Example - ~~Cheat~~ Don’t reinvent the wheel, again

```
static <T> void printArray(T[] arr) {
   System.out.print(Arrays.deepToString(arr));
}
```

- `[[[1, 2, 3], [4, 5, 6], [7, 8, 9]], null, [[10, 11, 12], [13, 14, 15], [16, 17, 18]]]`
  - Yes, we have that **`null`** in there, but it’s totally a feature, not a bug
- This now works for any amount of nested arrays. And any (non-primitive) type of array!

Original vs Refactored

```java
static void print3DArray(String[][][] arrays) {
   System.out.print("[");
   for (int i = 0; i < arrays.length; i++) {
       if (arrays[i] == null) {
           continue;
       }
       System.out.print("[");
              for (int j = 0; j < arrays[i].length; j++) {
                      if (arrays[i][j] == null) {
                             continue;
                      }
           System.out.print("[");
           for (int k = 0; k < arrays[i][j].length; k++) {
               System.out.print(arrays[i][j][k]);
                      if (k < arrays[i][j].length - 1) {
                             System.out.print(", ");
                      }
           }
              System.out.print("]");
           if (j < arrays[i].length - 1) {
               System.out.print(", ");
           }
       }
       System.out.print("]");
              if (i < arrays.length - 1) {
                  System.out.print(", ");
              }
   }
   System.out.print("]\n");
}
```

```java
static <T> void printArray(T[] arr) {
    System.out.print(Arrays.deepToString(arr));
}
```

Bonus: Quake III - Fast Inverse Square Root (with original comments)

```c
float q_rsqrt(float number)
{
  long i;
  float x2, y;
  const float threehalfs = 1.5F;

  x2 = number * 0.5F;
  y  = number;
  i  = * ( long * ) &y;                       // evil floating point bit level hacking
  i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
  y  = * ( float * ) &i;
  y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
  // y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

  return y;
}
```

Good Style and Clean Code Guidelines

- Every company has preferences, and you must follow their conventions
- Google Style Guide
- 61B's Style Guide!

- Not a complete list of guidelines!
    - Add and commit before and after a refactor
    - Understand what your code needs to do
    - Avoid unnecessarily complex ways of doing things
    - Use descriptive method and variable names, and document your code well
    - Separate logic into simpler methods
    - Don't reinvent the wheel
    - Google and StackOverflow are your friend

Writing Clean Code, Summary

- Refactoring is hard.
    - No immediate clear gains—it’s all investment in the future
    - The more that is built on top of your bad code, the worse the problem becomes
        - It’s hard to tell what’ll break, and more will break the longer you wait
    - Code cleanly from the start!

Documentation

Lecture 31, CS61B, Spring 2024

Documentation Intro

- **Documentation** in technology refers to various written materials that accompany software or systems
- It covers a broad spectrum, including:
  - Code Comments
  - API Documentation
  - User Manuals
  - Design Specifications
  - Test Documents
- We'll concentrate on practical documentation strategies that you can directly apply to your projects today

Why Documentation?

"We don't have enough time"

- Quick documentation now avoids longer explanations later

"No-one would read it anyway"

- I highly doubt it
- Serves as a guide for future developers and users of the code
- Essential for maintaining and scaling projects
- In Project 3, your partner will benefit from reading it

"The code should be self-documenting"

- Would you rather read 100 lines of someone's code or a short summary?
- Documentation explains *why* and *the context*, not just *how*

Documentation Example

What can we improve about this documentation?

```java
/**
* Changes location of the trees in the world
* We use java.util.Random.
*/
public boolean movePlayer(int x, int y) {...}
```

- Are we moving trees? Are we moving the player?
- Why are we including `java.util.Random`?
- What are `x` and `y`?

Documentation Example

```java
/**
* Moves the player to a specified location within the game world.
* Validates the destination and checks for bounds.
*
* @param x The x-coordinate of the destination (horizontal position).
* @param y The y-coordinate of the destination (vertical position).
* @return 'true' if the move is successful, 'false' otherwise.
* @throws OutOfBoundsException if coordinates are outside the world.
* @usage To move the player to (5, 10), call movePlayer(5, 10).
*/
public boolean movePlayer(int x, int y) {...}
```

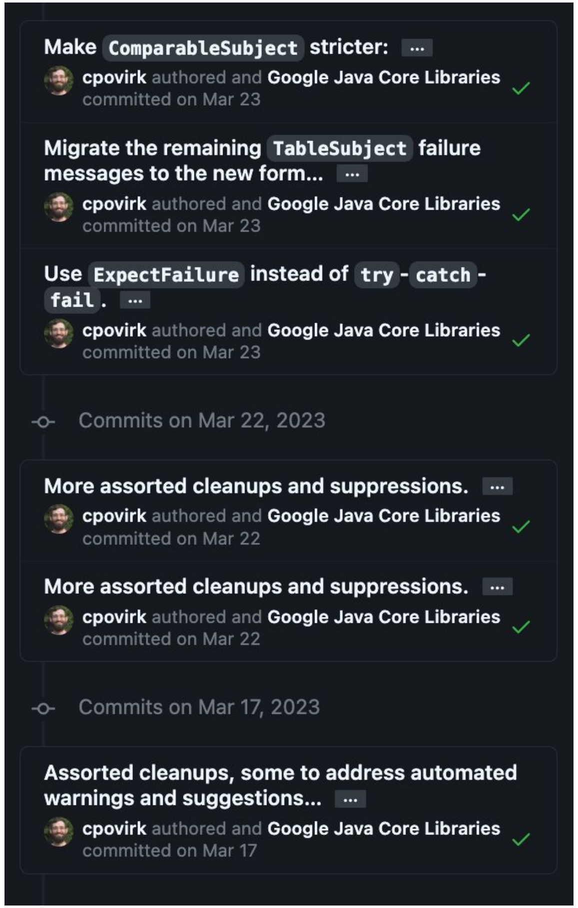
Writing Good Git Commits

There are well-established conventions:

- Capitalize the subject line
- Use imperative mood in the subject line
  - If applied, this commit will <*subject line here*>
- Use the body to explain **what and why** vs how
  - Code already says how, but we need why
  - If it is complex, add source comments
- Limit subject line to 50 characters (soft rule)
  - Ensures that commits are readable
  - If you hard time summarizing, you might be committing too many changes at once.

Rules: https://cbea.ms/git-commit/

Example: Google Truth Library (We use for writing tests!)


Make `ComparableSubject` stricter: …
cpovirk authored and Google Java Core Libraries committed on Mar 23 ✓

Migrate the remaining `TableSubject` failure messages to the new form... …
cpovirk authored and Google Java Core Libraries committed on Mar 23 ✓

Use `ExpectFailure` instead of `try`-`catch`-`fail`. …
cpovirk authored and Google Java Core Libraries committed on Mar 23 ✓

Commits on Mar 22, 2023

More assorted cleanups and suppressions. …
cpovirk authored and Google Java Core Libraries committed on Mar 22 ✓

More assorted cleanups and suppressions. …
cpovirk authored and Google Java Core Libraries committed on Mar 22 ✓

Commits on Mar 17, 2023

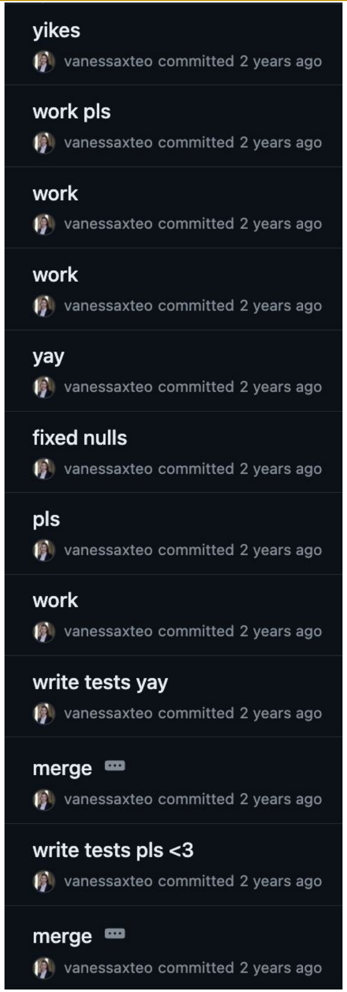
Assorted cleanups, some to address automated warnings and suggestions... …
cpovirk authored and Google Java Core Libraries committed on Mar 17 ✓

Writing Good Git Commits

Good Commit Messages:

- **`Add`**` room generation logic`
- **`Fix`**` overlapping rooms`
- **`Make`**` world generation tests`
- **`Document`**` world generation algorithm`
- **`Start`**` persistence logic`
- **`Refactor`**` persistence logic to use RandomUtils`
- **`Reformat`**` game homepage text to use more whitespace`
- **`Rearrange`**` buttons so OK is on the lower right`
- **`Revise`**` link to update it to the new URL`

Some Bad Commit Messages


yikes
vanessaxteo committed 2 years ago

work pls
vanessaxteo committed 2 years ago

work
vanessaxteo committed 2 years ago

work
vanessaxteo committed 2 years ago

yay
vanessaxteo committed 2 years ago

fixed nulls
vanessaxteo committed 2 years ago

pls
vanessaxteo committed 2 years ago

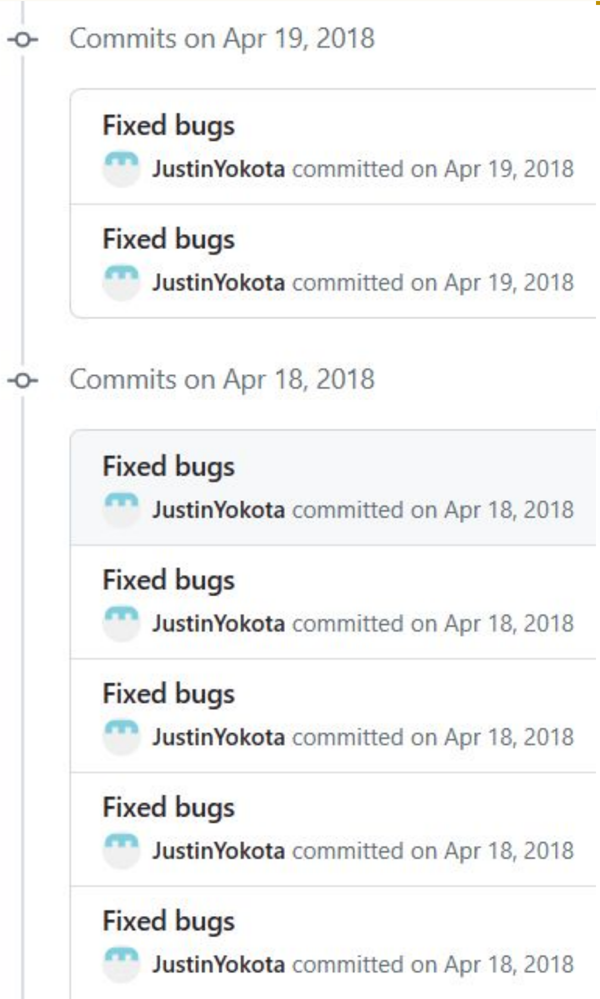
work
vanessaxteo committed 2 years ago

write tests yay
vanessaxteo committed 2 years ago

merge
vanessaxteo committed 2 years ago

write tests pls <3
vanessaxteo committed 2 years ago

merge
vanessaxteo committed 2 years ago


Commits on Apr 19, 2018

Fixed bugs
JustinYokota committed on Apr 19, 2018

Fixed bugs
JustinYokota committed on Apr 19, 2018

Commits on Apr 18, 2018

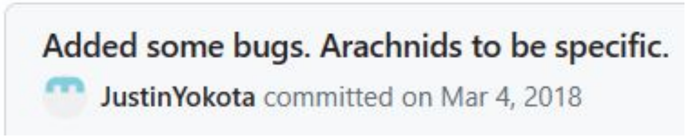
Fixed bugs
JustinYokota committed on Apr 18, 2018

Fixed bugs
JustinYokota committed on Apr 18, 2018

Fixed bugs
JustinYokota committed on Apr 18, 2018

Fixed bugs
JustinYokota committed on Apr 18, 2018

Fixed bugs
JustinYokota committed on Apr 18, 2018


Added some bugs. Arachnids to be specific.
JustinYokota committed on Mar 4, 2018

Overview

Poor documentation can impact productivity in three main areas:

- Searchability
    - Time lost in deciphering and navigating code
    - Might end up rewriting the same code multiple times
- Onboarding
    - Everyone’s implementations are different
    - In office hours, you will be onboarding course staff to your project
    - Industry examples
- Collaboration
    - You and a partner will be writing code that both of you should understand

Working in a Team

Lecture 31, CS61B, Spring 2024

Project 3: BYOW

Project 3 is a team project.

- This semester, as in most of the recent past, we're doing teams of 2.

Two main reasons:

- Get practice working on a team.
- Get more creativity into the project since it's so open ended.

Ancillary reason: Also reduces programming workload per person, but the project is small enough that a single person can handle it.

Teamwork is Hard

In the real world, some tasks are much too large to be handled by a single person.

When faced with the same task, some teams succeed, where others may fail.

Last semester, ~9% of partnerships thought their partnership did not work well for them.

Individual Intelligence

In 1904, Spearman very famously demonstrated the existence of an “intelligence” factor in humans. As described in Woolley (2010):

- “People who do well on one mental task tend to do well on most others, despite large variations in the tests’ contents and methods of administration.” This mysterious factor is called “intelligence.”
- Intelligence can be quickly measured (less than an hour).
- Intelligence reliably predicts important life outcomes over a long period of time, including:
    - Grades in school.
    - Success in occupations.
    - Even life expectancy.

Note: There is nothing in Spearman’s work that says that this factor is genetic.

Group Intelligence

In the famous "Evidence for a Collective Intelligence Factor in the Performance of Human Groups", Woolley et. al investigated the success of teams of humans on various tasks.

They found that performance on a wide variety of tasks is correlated, i.e. groups that do well on any specific task tend to do very well on the others.

- This suggests that groups do have "group intelligence" analogous to individual intelligence as demonstrated by Spearman.

Group Intelligence

In the famous "Evidence for a Collective Intelligence Factor in the Performance of Human Groups", Woolley et. al investigated the success of teams of humans on various tasks.

Studying individual group members, Woolley et. al found that:

- Collective intelligence is not significantly correlated with average or max intelligence of each group.
- Instead, collective intelligence was correlated with three things:
  - Average social sensitivity of group members as measured using the "Reading the Mind in the Eyes Test" (this is really interesting).

Group Intelligence

In the famous “Evidence for a Collective Intelligence Factor in the Performance of Human Groups”, Woolley et. al investigated the success of teams of humans on various tasks.

Studying individual group members, Woolley et. al found that:

- Collective intelligence is not significantly correlated with average or max intelligence of each group.
- Instead, collective intelligence was correlated with three things:
  - Average social sensitivity of group members as measured using the “Reading the Mind in the Eyes Test” (this is really interesting).
  - How equally distributed the group was in conversational turn-taking, e.g. groups where one person dominated did poorly.

Group Intelligence

In the famous "Evidence for a Collective Intelligence Factor in the Performance of Human Groups", Woolley et. al investigated the success of teams of humans on various tasks.

Studying individual group members, Woolley et. al found that:

- Collective intelligence is not significantly correlated with average or max intelligence of each group.
- Instead, collective intelligence was correlated with three things:
  - Average social sensitivity of group members as measured using the "Reading the Mind in the Eyes Test" (this is really interesting).
  - How equally distributed the group was in conversational turn-taking, e.g. groups where one person dominated did poorly.
  - Percentage of females in the group (paper suggests this is due to correlation with greater social sensitivity).

Project 3

Presumably, learning habits that lead to greater group intelligence is possible.

- We hope that project 3 helps with this.

Recognize that teamwork is also about relationships!

- Treat each other with respect.
- Be open and honest with each other.
- Make sure to set clear expectations.
- … and if those expectations are not met, confront this fact head on.

Reflexivity

Important part of teamwork is “reflexivity”.

- “A group’s ability to collectively reflect upon team objectives, strategies, and processes, and to adapt to them accordingly.”
- Recommended that you “cultivate a collaborative environment in which giving and receiving feedback on an ongoing basis is seen as a mechanism for reflection and learning.”
  - It’s OK and even expected for you and your partner to be a bit unevenly matched in terms of programming ability.

You might find this description of best practices for team feedback useful, though it’s targeted more towards larger team projects.

- Some key ideas from this document follow.

Feedback is Hard

Most of us have received feedback from someone which felt judgmental or in bad faith.

- Thus, we’re afraid to give even constructive negative feedback for fear that our feedback will be misconstrued as an attack.
- And we’re conditioned to watch out for negative feedback that is ill-intentioned.

Feedback is Hard

Feedback can also feel like a waste of time:

- You may find it a pointless exercise to reflect on your partnership during the project. What does that have to do with a programming class?
- In the real world, the same thing happens. Your team has limited time to figure out “what” to do, so why stop and waste time reflecting on “how” you’re working together?

Feedback is Hard

Feedback can simply be difficult to produce.

- You may build incredible technical skills, but learning to provide useful feedback is hard!
- Without confidence in ability to provide feedback, you may wait until you are forced to do so by annual reviews or other structured time to provide it.
    - If you feel like your partnership could be better, try to talk about it without waiting until you have to review each other at the end of next week.

Strategies for Collaboration: Pair Programming

Pair programming is a technique where two programmers work on a single station

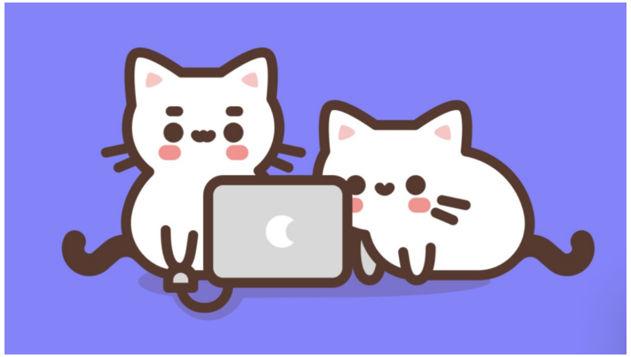
- Driver: focus on implementation
- Navigator: consider design, accuracy

Benefits:

- Reduce errors
- Improve code quality
- Increase productivity

Tips:

- Clearly define roles
- Swap often (15-30 mins)



You can do this remotely too!
IntelliJ: Code With Me
VS Code: Live Share

Strategies for Collaboration: Communication

In industry, you will likely have frequent stand-up meetings to discuss:

- What you are working on
- If anything is blocking your progress
- If you need feedback or advice from a team member

Keep each other up-to-date with progress!

Q&A

Lecture 31, CS61B, Spring 2024