Lecture 31: Software Engineering 2

11/6/2020

Build Your World

Part 1: World Generation

- Given a random seed (long), generate a 2D world (TetTile[][]) with rooms and hallways
    - N: Create new world
    - 343434: Random seed
    - S: End of seed marker

Part 2: Interactivity

- In part 2, you'll add:
    - An interactive keyboard mode
    - The ability for the avatar to move around in the world
        - Must also be able to handle movements given via `interactWithInputString`

Ousterhout's Take on Complexity

- There are two primary sources of complexity:
    - **Dependencies**: When a piece of code cannot be read, understood, and modified independently
    - **Obscurity**: When important information is not obvious

Modular Design

Hiding Complexity

- One powerful tool for managing complexity is to design your system so that programmer is only thinking about some of the complexity at once
    - Using helper methods (i.e. `getNeighbor(WEST)` and helper classes (`InputDevice`)) hide complexity

Modular Design

- In an ideal world, system would be broken down into modules, where every module would be totally independent
    - Here, "module" is an informal term referring to a class, package, or other unit of code
    - Not possible for modules to be entirely independent, because code from each module has to call other modules
        - e.g. need to known signature of methods to call them
- In modular design, out goal is to minimize dependencies between modules

Interface vs. Implementation

- As we've seen, there is an important distinction between Interface and Implementation

    - Map is an interface
    - HashMap, TreeMap, etc. are implementations
- "The best modules are those whose interfaces are much simpler than their implementation"
    - A simple interface minimizes the complexity the module can cause elsewhere
        - If you only have a `getNext()` method, that's all someone can do
    - If a module's interface is simple, we can change an implementation of that module without affecting the interface
        - If `List` had an `arraySize` method, this would mean you'd be stuck only being able to build array based lists

Interface

- A Java interface has both a formal and an informal part:
    - Formal: The list of method signatures
    - Informal: Rules for using the interface that are not enforced by the compiler
        - Example: If your iterator requires hasNext to be called before next in order to work properly, that is an informal part of the interface
        - Example: If your add method throws an exception on null inputs, that is an informal part of the interface
        - Example: Runtime for a specific method, e.g. `add` in `ArrayList`
        - Can only be specified in comments
- Be wary of the informal rules of your modules as you build project 3

Modules Should be Deep

- Ousterhout: "The best modules are those that provide powerful functionality yet have simple interfaces. I use the term *deep* to describe such modules"
- For example, the KdTree is a deep module
    - Simple interface:
        - Constructor takes a list of inputs
        - `nearest` method that takes a point and returns closest point in set
        - Nothing informal the user needs to know
    - Powerful functionality:
        - List converted into a complex searchable binary tree
        - Nearest method has complex and subtle pruning rules for efficiency

Information Hiding

- The most important way to make your modules deep is to practice "information hiding"
    - Embed knowledge and design decision in the module itself, without exposing them to the outside world
- Reduces complexity in two ways:
    - Simplifies interface
    - Makes it easier to modify the system

Information Leakage

- The opposite of **information hiding** is **information leakage**

- Occurs when design decision is reflected in multiple modules
    - Any change to one requires a change to all
    - Information is embodied in two places, i.e. it has "leaked"
- Ousterhout:
  - "Information leakage is one of the most important red flags in software design"

Temporal Decomposition

- One of the biggest causes of information leakage is "temporal decomposition"
- In temporal decomposition, the structure of your system reflects the order in which events occur

BYOW Suggestions

- Some suggestions for BYOW:
  - Build classes that provide functionality needed in many places in your code
  - Create "deep modules", e.g. classes with simple interfaces that do complicated things
  - Avoid over-reliance on "temporal decomposition" where your decomposition is driven primarily by the order in which things occur
    - It's OK to use some temporal decomposition, but try to fix any information leakage that occurs
  - Be strategic, not tactical
  - Most importantly: Hide information from yourself when unneeded!

Teamwork

Project 3

- Project 3 is a team project
- Two main reasons:
  - Get practice working on a team
  - Get more creativity into the project since it's so open ended

Teamwork

- 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

Individual Intelligence

- In 1904, Spearman very famously demonstrated the existence of an "intelligence" factor in humans:
  - "People who do well on 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
    - Life expectancy
- Note: There is nothing in Spearman's work that says this factor is genetic

Group Intelligence

- Wooley 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
- Studying individual group members, Wooley 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"
    - 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)

Teamwork and Project 3

- Presumably, learning habits that lead to greater group intelligence is possible
- 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 to "cultivate a collaborative environment in which giving and receiving feedback on an ongoing basis is seen as a mechanism for reflection and learning"

Feedback is Hard: Negativity

- Most of us have received feedback from someone which felt judgmental or in bad faith
  - Thus, we're afraid to given 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: Can Seem Like a Waste of Time

- Feedback also can feel like a waste of time:
  - You may find it a pointless exercise to rate each other twice 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: Coming Up With Feedback is Tough

- Feedback can simply 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

Team Reflection

- We are going to have you reflect on your team's success twice:
  - This partnership has worked well for me
  - This partnership has worked well for my partner
  - Estimate the balance of work between you and your partner
    - Contribution in a more general sense
  - If applicable, give a particularly great moment from your partnership
  - What's something you could do better?
  - What's something your partner could do better?
- In extreme cases, we will not be penalizing partners who contributed less