CS4530 Spring 2021 Review Discussion

Q: Has programming gotten better/easier over time? Or has programming become a bigger problem as programs/computers get bigger?

- Programming has become easier to learn - more resources available, setup has gotten easier
  - Deployment is useful, too!
- Easier to collaborate now - Processes have improved - agile, lean, xp - software as a service for managing teams, etc.
- Downside - our EXPECTATIONS of programs have grown too

Q: What is difference between programming and software engineering?

- SE is about programming at scale - lots of developers, lots of revisions of code (maintain over time)

"From there it is only a small step to measuring 'programmer productivity' in terms of 'number of lines of code produced per month.' This is a very costly measuring unit because it encourages the writing of insipid code, but today I am less interested in how foolish a unit it is from even a pure business point of view. My point today is that, if we wish to count lines of code, we should not regard them as 'lines produced' but as 'lines spent': the current conventional wisdom is so foolish as to book that count on the wrong side of the ledger."

- Technical debt - everything that we produce, we have to maintain

Sample question discussion

Q: Rest design question.

1. HTTP Verb: POST

REST Path: /hotels

Request data (www-url-encoded):

```
      {
       ownerID: string,
```

    hotelName: string,
    city: string,
    state: string,
    rooms: {roomID: string, dailyRate: number}[]
    }
Response data (JSON): {hotelID: string, status: string}

2. HTTP Verb: GET
   REST path: /hotels
   or: /cities/:cityID/hotels <— This prioritizes “cities” as the resource, vs “hotels” which prioritizes “hotels” as the resource
   Request data: Query string ?city=CityName&state=StateName
   Also OK, a variant of: /hotels/state/:stateName/city/:cityName (but we prefer the query string in practice)
   Response: {hotelID: string, hotelName: string}
3. HTTP Verb: GET
   REST path:
      /rooms
      /hotels/:hotelID/rooms (either is OK! But this locks you into querying only for rooms within a single hotel)
   Request data: Query string ?hotelID=someID&date=someDate
   Response: {roomID: string, dailyRate: number}[]
4. HTTP Verb: POST
   REST path: /reservations
   Request data: {userID: string, roomID: string, date: Date}
   Response: {status: string, reservationNumber: number}
5. HTTP Verb: DELETE
   REST path: /reservations
   Request data: JSON body - {userID: string, reservationNumber: number}
   Response: {status: string}

Asynchronous Programming

```
    someRemoteAPICall("callA", () => {
        console.log("A");
        someRemoteAPICall("callB", () => {
            console.log("B");
        })
})
    someRemoteAPICall("callC", () => {
        console.log("C");
        someRemoteAPICall("callD", () => {
            console.log("D");
        })
});
```

1. Could “B” print before “A” - NO
2. Could “C” print before “A” - YES
3. Could “D” print before “A” - YES
4. D (Because “B” can’t print before “A”)

Testing with mocks
Mocks - Replace a function with a canned value
Fakes - A mock that has real logic implemented [thin/grey line between this and mocks - don’t worry]
Spies - Call the actual function, but record that it was called [in some testing frameworks, all spies are mocks, but this isn’t always the case]
[No need to distinguish between the 3 above classes for exam]

1. Mock isAuthenticated, doLogin, doLogout. Reduce flakiness by making tests hermetic: all logic and data for the test is self contained (doesn’t rely on outside authenticate service, or network)
2. Given: An application state where a user is logged in
   When: A user clicks on the logout button
   Then: The doLogout should be called, and the doLogin function should not be called

Distributed systems

1. Tolerate all of them (?). Tolerate the different kinds of errors users might create, and also tolerate different kinds of network failures (partitions vs total outages)
2. Retry operations when they fail.
3. Everyone should see the same thing. There can be inconsistency when two players' view of the world diverges. For example, there could be a lag in watching someone move around the screen. Alternatively, some players might be visible only to some other players, but not to all.
4. Availability means that the system is functioning - unavailable if the server crashes

Inclusive Development

1. Limited groups make it easy to have blind spots (especially in image recognition, voice recognition applications) - think of only a limited set of experiences, leading to a limited set of solutions; cultural diversity brings more creativity; can empathize better with users and hopefully make better software/products
2. GenderMag encourages a persona-based design process, with a diverse group of personas that represent different technology learning and usage styles
3. Blind-> screen readers, deaf -> closed captioning, non-native English users -> iconography, translations
4. Examples: Iconography (good UX design generally), Alt-text (screen readers + low bandwidth + broken photo links), closed captions, GenderMag inclusivity example on MS Academic

Q: One of the goals in testing module was to give examples of non-deterministic testing - is this a good thing, or a bad thing?
A: Have a mix of both. Lots of small unit tests (which are deterministic), some integration/end-to-end tests (which might be non-determinsitic).