Parth Kapur

CS441 – Software Engineering

Professor. Zheng

Assignment 1

1.1)

Fred Books identified four essential difficulties of software systems: complexity, conformity, changeability, and invisibility. Out of these four essential difficulties, I feel that invisibility is the most challenging difficulty and changeability is the least challenging.

Invisibility, in my opinion, is the most difficult as software is invisible and visualizable. Due to this, in order to write software, we must utilize techniques that are a step by step process. However, these techniques differ from blueprints or scale drawings as these scale drawings are not planar or hierarchical. As a result, as the article as said, our mind’s powerful conceptualization tools are unable to be utilized. Due to the lack of conceptualization, writing software can have a thought process but end up dramatically changing in the way the software is written. Additionally, management will also need to somewhat understand software as well. Conceptualization difficulties, once again, makes it exceptionally difficult to transfer knowledge from one mind to another mind (employee to manager). High level languages can help with conceptualization. However, transferring knowledge from employee to manager is still quite difficult.

Changeability, in my opinion, is the least challenging difficulty of software development. Fred Brooks maintains that software entity is constantly subject to pressures for change since it is pure thought, which inherently, is infinitely malleable. However, as programmers, we are taught to write code in a way where it is designed to be changed with ease. We are taught scope, we are taught constants, we are taught functions, classes, objects, inheritance. All of these tools are designed knowing that software changes. So, unlike invisibility, changeability is something that software engineers have more tools to combat against. Thus, as long as these tools are utilized, the difficulty in changing or updating software is not nearly as challenging as some of the other difficulties that Fred Brooks has mentioned.

1.2)

Object-oriented programming is one of the software methods or tools that I have used. I believe that object-oriented programming is a promising attack on the essential difficulty of changeability. Object-oriented programming allows the use of functions, objects, and object specific functions. This can be quite useful if software is subject to change often, which, it is. As a result, the programmer can write the software utilizing object-oriented programming. Then, if changeability is required, instead of making changes to thousands of lines of codes, the programming can make changes to the object as well as that object’s functions using the object’s class. As a result, the software may still change frequently. However, thru the use of object-oriented programming in combination with high level languages, these factors help to strengthen code re-usability as well as adaptability.

2)

A prototype software system, according to Fred Books, “is a system that simulates the important interfaces and performs the main functions of the intended system, while not necessarily being bound by the same hardware speed, size, or cost constraints.”. So, a prototype, in layman’s terms is a system designed for the client to test it for consistency and usability. A prototype will capture the main ideas and intentions of a client without focusing on exceptional tasks, user input errors, etc. Fred Books states this is important because a client never clearly knows what he wants and has never thought in detail about the solution to the problem he is asking. So, it is useful to prototype before developing the final software.

Incremental development, according to Fred Books, “is the idea that any software system should first be made to run, even if it does nothing useful except call the proper set of dummy subprograms. Then, bit by bit, it should be fleshed out, with the subprograms in turn being developed – into actions or calls to empty stubs in the level below.”. In layman’s terms, this means that incremental development is to build the basic skeleton or shape of the program first and run it, even if the program does nothing. After, focus on the individual parts or functions of the program.

So, the difference between prototyping and incremental development is that a prototype is a developed program that captures the main ideas and intentions of a client while not focusing on input errors, etc. In incremental development, there is also a running program. However, the difference between the prototype is the running program is not complete with the main functions and ideas of a client. Instead, the program is incrementally developed with subprograms that complete the client’s main ideas and intentions. The key-term is incremental. Functionality in incremental development is built piece by piece. It is like creating a car body, then developing the drivetrain, transmission, engine, wheels, etc. Each of the sub-parts (subprograms in computer science terms) incrementally add to the function of a car, which is to take a person from point A to point B. Eventually, when all these subprograms are incrementally developed, the car will be a complete and working model that captures the main ideas and intentions of the client (manufacturer).

3)

The software application idea I have is a website. This website is a ride sharing website which differs from ordinary rideshare services like Uber or Lyft. This website’s purpose will allow users around the world to rideshare with each other. The defining difference between Uber or Lyft is, Uber/Lyft does not fulfill the ability to plan ridesharing ahead of time. Uber, Lyft, and other rideshare companies offer on the spot ridesharing. Sometimes, however, that is not what is desired by the user. People may wish to plan their trips out ahead of time for a variety of reasons including following an itinerary or cost saving. Their only option currently is to book a taxi which can be a costly investment and defeats the purpose of planning. My rideshare website will accomplish and fulfill the needs of this target market (planned rideshare). The tools that I am planning to use to build this website include: HTML, CSS, JavaScript, jQuery, Bootstrap, PHP, MYSQL, Google Maps API, AJAX, and JSON. I may not utilize all these tools, but I can imagine seeing a need for each of these languages. The website will work by prompting a user to sign up or log in. If user needs to sign up, upon successful signup the user will receive a push email to active his or her account. A user should also be able to change his email address or reset his password if required. Once this is done, the user should be able to put a start and end destination into the Google Maps API. When this action is completed, there will be a called to the MYSQL database to query all drivers willing to drive from the start destination to the end destination. This query will display drivers name, vehicle, cost per seat, departure time, phone number, etc. The user will also be allowed to add a trip, edit a trip, or delete a trip if the user himself/herself wants to offer a ride sharing service from a destination to an end destination. Additionally, a user should have the ability to update his profile information with pictures, details, etc. As more and more users sign up, I hope to cover the entire area of San Diego!