2DV603 – Assignment 0
Service Design

Author: *Sarpreet Singh Buttar*

Task 1

Impact: “No Silver bullet” is a classic essay on software engineering which was written more than 30 years ago by Brooks. The author tried to separate the causes of essential and accidental difficulties in software development. He identified complexity, conformity, changeability and invisibility as inherent properties which makes them irreducible. Furthermore, he also described some promising attacks on conceptual essence which have left huge impact on software design in order to reduce its essential complexity. Following are the successful promising attacks which were suggested by the author and has been applied in modern software development.

1. **Buy versus build:** The author suggested to reuse software tools in order to improve the productivity. In result, a huge number of open source tools have been introduced in the past 20 years which supports the idea of reusing the existing software.
2. **Requirements refinement and rapid prototyping:** The author argued that it is not possible to get a complete definition of system in the beginning of development, it is more productive to gather the requirements in iterative cycles. In result, agile development was introduced which supports iterative cycles and it became more successful than the waterfall model.
3. **Incremental development:** The authors suggested the idea of growing software rather than building. In result, mock objects, GUI frameworks, model based development were introduced and become part of the incremental development and more successful over time.
4. **Great designers:** The authors suggested to find creative designers because contracting software is a creative process. In result, agile development introduced and supports the creativity of the best designers.

In conclusion, essential difficulties can only be reduced, not eliminated.

Following are the three Core problems including their possible mitigations:

1. **Complexity**: It is very hard to provide the description of software entity by constructing models because complexity is the essential property which cannot be ignored. From the complexity many problems comes out such as difficulties to communicate with mates, delays, going over budget, product flaws, less understanding and many more.
   Possible Mitigations:
   - If possible buy the software product rather than building it because it is cheaper, provide s immediate delivery, is better documented and more maintained than a homegrown product.
   - Incremental development should be used for growing the software system because it grants painless backtracking and early prototypes.
   - For maintaining an overview of complexity, Time-sharing must be practised because it preserves immediate needs.
2. **Changeability**: All successful softwares are constantly under pressure for change because users demands increase over time which demands extended functionality. Also software life is longer than it’s machine life, so for running in a new machine the software must meet some standards of the new machine.
   Possible Mitigations:
   - Product requirements should be carefully refined in iterating cycles between the designer and the client in order to get a correct definition of the system.
   - Structure should be designed by best designers so that it is smaller, cleaner, simpler and demands less work to add new functionality.
3. **Invisibility:** Software is invisible by nature, difficult to show its picture and details.
   Possible Mitigations:
   - Use UML or Visual modeling
   - Lots of communication among team members

Task 2

Reflection: A design is a created object, independent from the thing being designed from it. The author divides this creative process into three different explicit aspects: the Idea (formulation of the conceptual constructs), the Implementation in the real world, and the Interaction with real users. In software engineering, working with the essence (complexity, conformity, changeability and invisibility) is the mental crafting of the conceptual constructs which means it can be complete before any implementation starts. Solving the accidental difficulties is its implementation process. Interaction takes place when the software is used. All the three steps operate recursively. Hence, the design is the mental imagination. According to the rational model (waterfall model), a software design must know its goal, needs, utility function, limitations and resources in the beginning but the author opposed it as these aspects changes from time to time. He mentioned that designers discover these aspects as they work, and suggested to create a list of known limitations and scan it periodically in order to eliminate or avoid these limitations.

Problems which I recognised:

1. **We Don't Really know the Goal When We Start:** Presently, I'm working part-time as a web scraper in a company (not officially released yet) which is designing a search engine for buying online antiques. In the beginning, they told me to scrap the objects with their title, subtitle, price, description, image, condition and link. After some weeks they added two more attributes - location and time zone because they decided to take objects from international websites. Again, they added an end date for the auction's products. During that time I have not read this article and also felt same as author did during his work in the missile company. But, now I saw that the design goal setting process is iterative process in order to decide what we really want.

   Workaround:
   - So far working in iterations.

2. **The Constraints Keep Changing:** Last year, I have studied a project oriented course in which every team have to design a website with user and admin mode by following the Waterfall model. The goals were fixed and very well known from the beginning but still the design changed at each submission because the limitations kept changing. For instance, somebody found a new way to interact within the classes which resulted in introducing new limitations. There are many more problems during this development but the word limit does not allow to explain them all.

   Workaround:
   - Mostly, by changing the source code in order to work around these new limitations.

3. **The Nodes Are Really Not Design Decisions, but Tentative Complete Designs:** Last summer, I had not much to do and spent my time by re-solving some first year assignments with different approaches. I recognise that each new solution is not a simple choice for the particular problem but one alternative among various simple choices. No workaround found for it, only experience helps.