**COMP-SCI 5555 Software Methods and Tools**

**Assignment 1**

# What are four essential difficulties of software systems discussed in Fred Brooks’s paper? Explain each using your own words.

“*No Silver Bullet – Essence and Accidents of Software Engineering*” is a widely discussed paper on Software Engineering written by Turing Award winner Fred Brooks in 1986. According to Brooks “no single software can be built in a single go”. He also states that we cannot expect the same growth rate for software as that of hardware.

In olden days, werewolves are the most dangerous monsters which will fill the nightmares of people. According to our folklore only a silver bullet can kill those werewolves. Brooks compares software system with werewolves. Werewolves are other than human beings which will change their shape enormously whenever they are angry. Similarly, softwares can cause disasters due to missed schedules, blown budgets and flawed products. So, Brooks depicts softwares as werewolves and a great programmer as silver bullet. But in Brooks opinion there is no such silver bullet programmer who can develop a reliable software system in a single go.

According to Brooks, there are two different types of complexity in software systems:

* Essential Complexity: It is the complexity involved in designing the complex structures of software application. This complexity is inherent in the nature of the software.
* Accidental Complexity: It is the complexity involved in the representation of the complex designs of software in programming languages and optimizing these representations. This complexity is not inherent in nature.

Brooks states that, even if we overcome all the accidental complexity, it cannot give us an order of magnitude improvement. We should also take care of essential complexity.

Brooks believe that the hard part of building a software is its specification, design and testing. The syntax errors that we come across during the development stage can be removed easily. Brooks proposed four inherent difficulties in achieving the essence of the software system.

**Complexity:**

Software components are more complex because they are built by different programmers. No two programmers will have the same set of ideas. If two software components are same, we will mark them as one entity. Thus, these software entities will have large complexity when compared with buildings, vehicles etc..,

Digital computers will have complex construction. But software systems will have orders of magnitude complexity when compared with digital computers. So it is always to describe, develop and test the softwares. Scaling-up of a software is not just repeating the elements in larger size. It is always increasing the number of software elements. Thus, scaling – up a software will always have a non-linear complexity. Many of software developing problems comes from this essential complexity and nonlinear increase of its size. If we develop a software without considering the future requirements, it will cause increase in the complexity of software. This is because, when we want to scale-up the software, as we didn’t consider the future requirements, it will cause a very difficult situation to scale the system. Sometimes the software may become invalid for the situation. In such cases, programmers will prone to develop inefficient temporary solutions to resolve the issue. This will lead to removal of systematic structure of software, thereby increasing the complexity of software components and their communication. This will in turn lead to product flaws, cost overruns, and schedule delays.

This complex construction of a software system will not only cause technical problems but also managerial problems such as difficulty in system overview, delays in software delivery targets. It also reduces the turnover per personnel as each one of the team has to spend tremendous amount of time in understanding this complex system.

**Conformity:**

One of the main tasks of the software is to interact with the hardware in order to perform some tasks. There are multiple arbitrary environments on which the software has to run with the same expected results. These arbitrary environments might be completely differ from each other. So it is the software engineer responsibility to be familiar with all of these diversified environments.

Let us consider, Microsoft Windows operating system. We cannot say that the OS is operational by merely testing it on a single computer. It is the responsibility of software developer to create the software in such a way that it is compatible with all the available environments and yields the exact required outputs. Microsoft adopted this “conformity” principle and developed Windows 10 operating system. Their aim is “One software, multiple platforms”. That means, the developed software should be capable of running on Laptop, mobile, tablets by dynamically adopting the environment properties.

**Changeability:**

In today’s world, change is the most common requirement and phenomenon in the society. But at what rate the change is happening will tell us the complexity. If we consider, cars and laptops, there won’t be a tremendous changes in the past decade. This is not the case with respect to software entity.

The main purpose of software is to achieve the functionality as required by the client. Softwares are infinitely malleable. That means, we can constantly change or improve the functionality of the software. But in case of buildings, the “change” can be achieved with high costs. All successful softwares should satisfy the below two properties.

Firstly, the software should be 100% working for what it is designed for. It should satisfy the basic required functionality. The end users will make use of the software in all the possible ways and in the end it is them who will show interest in extending the functionality of the software.

Second, a software will said to be successful, if it lasts beyond the normal lifetime of the machine for which it is written. It should also be functional when an old hardware part is replaced with an new hardware like new hard disk drive, new displays etc..,

To be short, software is part of the infinitely changing world. The changes in the entities of the world are forcing the software to be changed indefinitely.

**Invisibility:**

We already know that software systems will have orders of magnitude of complexity in terms of their design, development. We can represent a building floor plan on a piece of paper. We can draw a person’s face. We can draw mechanical parts of a vehicle. A geometric reality is possible in all the above cases. But it is not possible when it comes to software systems. We cannot realize the software structure on a paper.

Almost all the elements on the Earth can be represented in space. In case of software system, we will be able to represent some elements of system in graphical manner like data flow, database storage structure etc.., but we cannot visualize the entire architecture of software system in space.

# Pick one software method or tool that you used before and specifically explain whether or not you think this method or tool is a “promising attack” on the essential difficulties mentioned above.

First let’s briefly go through each of these “Promising attacks”.

Brooks proposed four essential parts which we need to address in order to minimize or completely remove the Essential complexities. Those are…

* **Do buy, Do not build:**

The most suitable solution for constructing a software is not to construct it from scratch. We need to use all the available resources (like libraries, plug-ins, tools and so on). This will help us in on-time delivery of products, cost reduction.

These days it becomes easier, as variety of vendors are offering more better and diversified software products. Also, the vendors will provide the support for the applications, which will help us in resolving the live issues on time. Consider a piece of software costing $100,000. Purchasing this software will might cost the salary of an employee/year, but the product delivery is immediate.

The cost of a software will always be the development cost. Even if there is a replication cost, it will be very less when compared with the development cost. So, if we want to use the software in multiple locations, purchasing the software with *n* number of replications will radically cut the total cost. Even more, if the software is purchased by more than one person, then the per-person cost will be very less. Moreover, replicating the software will improve the productivity.

Back in 1950’s, the study shows that the users are not willing to purchase these softwares. This is because the diversified requirements of different users are not satisfied by the softwares. The buyer of a million dollar machine can afford one or two thousand dollars to get a special customized functionality of the machine. But what about the low level business people? They do not tend to purchase these ready-made softwares.

But that is not the case these days. Each and every person is capable of handling and diagnosing his own computer at least in the basic level. Today, the softwares are being built in such a way that they will have lot of customizing options that suit the client requirements. Also these customizations are in such a way that the single user can implement them without any hassle.

* **Rapid Prototyping:**

The most difficult and important part of building a software is to designing the software components and relating them. In order to build a perfect software design, one need to have iterative meetings with the client, retrieve, refine the exact product requirements before the beginning phase of code development. This will ensure less or no redesign of software architecture, less code reworks and increases the productivity.

In some cases, client might not know completely, precisely and correctly what functionality exactly he wants. In such cases rapid prototyping will help the software designer to simulate a basic prototype of the final product and demonstrate it to the client to get a clear view of what exactly is required to build. This prototyping will not include the functionalities like input data validation, displaying proper error messages etc.., but it will help the client to come up with more accurate functional requirements.

Much of current software project acquisition procedures rests upon the assumptions that one can come up with a satisfactory system in advance. One who gets the bids for this software development will build it and installs it in the client environment. But Brooks will not agree with such process. According to Brooks, this procedure will NOT yield “I built exactly what you wanted”. Brooks suggests to have recurring meetings with the client, have revise the functional requirements by simulating the prototypes of the final products. This will reduce the Essential complexity of the softwares.

* **Grow the software:**

Construction of a house involves a lot of parallel tasks like, building walls, slab, and windows and so on. This parallel construction will cause the house to be not in the correct shape, or at least not with the required dimensions. So, we need to demolish the entire house and build it again. What Brooks suggests with respect to software is, building a software will cause more and more problems in future. He wants each software to be grown, not to be built.

According to Harlan Mills, any software system should be grown by incremental development. To do so, first the system should be made to run, even there is no exact functionality of the sub programs. Then we need to build the subprograms step-by-step. This will gradually give the final shape to the final product. This process should go as top-down design. This process will resembles the rapid prototyping process as at first we will have just the draft version of the final product and in the end we will have the final product itself. This technique will help us in backtracking whenever we have an error in the software. The main advantage of growing a software is, the developer will have a running system, even it is in its early stages. The developer will be more interested in developing the application as at every stage he will have a working system.

* **Great Designers:**

Software designers are the important key personnel in developing a software. A good designer may create a sustainable software, but these days, a sustainable software is not enough. The software should have more efficiency to cope with the infinitely increasing the demand. So, great designers has to be evolved from the society. A great designer will have the ability to foresee the future requirements which will help him in designing the future-proof softwares.

In Brooks’s opinion, good designers are scarce. Great designers are very rare. Most of the organizations are keen to find the good managers, but they are not showing the considerable interest in finding the great designers. Brooks suggest that great designers are as much important as great managers to an organization. They both should have same salary, recognition in the office, support and so on.

Each and every organization should show interest in creating the great designers. This can be done by

* Systematically identify top designers as early as possible.
* Assign a career mentor responsible for the designer’s development of prospect and a good career.
* Encourage the designers to learn from mistakes by taking short courses.
* Provide opportunities for growing designers to exchange ideas and experiences with the other designers.

The above mentioned four factors will impact the Essential complexity of the Software systems.

**My usage of these “promising attacks” in project work:**

1. **Do Buy, Do Not Build:**

We’ve used different tools in order to provide the client required functionalities. One of them is **Intec InterconnecT**.

**Client requirement:** Rating the call detail records (CDRs) received from multiple telephone exchange switches and creating final bill to the operators.

The purchased tool “Intec InterconnecT” will be used for rating the CDRs. So, we purchased the already available tool which will implement the some part (Rating the CDRs) of the client requirement. Bill generation logic was developed by us.

Purchasing this tool helped us in delivering the client requirements on time. Also we will get exclusive 24\*7 support to resolve the issues with the tool. So buying a tool which is already available in the market is definitely a promising attack on essential difficulties of software systems. This tool is compatible with multi-variant operating systems and also they provide new versions of this tool to support latest operating systems. This helped us in achieving the conformity of the software system. Thus this tool acted as a promising attack on “**conformity**” difficulty of software system.

1. **Rapid Prototyping:**

We’ve used this prototyping method to design the solution for the client requirement. The requirement is to scale the existing application in order to make it future-proof. At first attempt, our client is not satisfied with the provided solution. This is because the client even don’t know the functionality of existing system. So, we had more recurring meeting meetings with the client and provided multiple solutions. These different solutions helped us in understanding the exact client requirement. We made changes to the solution design and the client accepted the final design. This prototype design of the solution helped us in preventing “code rewrites”. This is because we got a clear view of client requirements by repeatedly developing the prototypes of the functionality workflow. Using this method helped us to remove two essential difficulties.

1. **Complexity**: As we’ve a clear view of what needs to be build, we took all the required precautions to make the “billing system” as simple as possible so that a new employee will be able to understand the architecture and data flow of the system.
2. **Changeability**: As we’ve multiple recursive meetings with the client, we came to know that the possible future requirements. This helped us to develop the application in such a way that the application can be changed to tackle the future requirements.
3. **Grow the Software:**

Growing the software is the primary task of our software development process. No one in our project is a silver bullet. So, we used to build the skeleton of the portion of software. And then we replaced the dummy code with the required functionality. We also did the unit testing of each functionality. In the next stage, we assembled all these software chunks and perform a system testing with multiple test cases. This growing technique helped us in finding the errors and exceptions very easily as we write the code in incremental development mode. So, I strongly believe that growing the software will act as the best promising attack on essential complexities of software system. This technique helped us a lot in reducing the **complexity** of software development process as we created the sub modules of the software and then integrated these submodules.

1. **Great Designers:**

We’ve a designer in our project but as he don’t have a good grip on our domain (Telecom domain) technology, he is still trying to improve his skills to be a “Great Designer”. He is responsible for understanding the client requirements. He is also responsible for providing the direct or alternative solutions to the client requirements. But our luck is, our client has enormous domain knowledge, so he helped us in creating the solutions. These solutions are future-proof and we didn’t get any problems while developing the software as the software design was done by our great designer (in this case it is our client☺). I strongly agree that, having a great designer will help the organization because designing the software clearly will give first best impression from client. Client will show more interest to give more projects to the organization. So, having a great designer will be a promising attack on essential difficulties of software development process. **Having a great designer helped us in attacking 3 of the essential complexities** (complexity, Conformity, changeability). Having a great designer also helped us in creating the system design documents, version control documents, bug tracking document and so on. This helped us in **visualizing** the life cycle of our software.

# Make a class schedule for this course using Microsoft Project 2013. Your schedule chart must include the following elements:

# • A Project Summary Task named “Software Methods and Tools”

# • Summaries and Tasks (e.g. Planning, Design, Implementation, Testing)

# • Recurring Tasks (e.g. Lectures/Labs)

# • Milestones (e.g. Assignments, Exams)

# • Relationships between the above elements (e.g. the Planning task generates Assignment 1)

