COMP-SCI 5555 Software Methods and Tools

Assignment 1

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.

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

- Essential Complexity: It is the complexity involved in designing the complex structures of software application.
- Accidental Complexity: It is the complexity involved in the representation of the complex designs of software in programming languages and optimizing these representations.

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

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.

2. 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.

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.

2. **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.

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.

4. **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.

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)

Picture is in the next page.

ID	WBS	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	**1**		**Software Methods and Tools**	**32 days**	**Tue 8/25/15**	**Thu 12/10/15**		
2	**1.1**		**Planning**	**4 days**	**Tue 8/25/15**	**Thu 9/3/15**		**Professor,Student**
3	1.1.1		Course Introduction	1 day	Tue 8/25/15	Tue 8/25/15		Professor,Student
4	1.1.2		Software Development Process and Activities	1 day	Thu 8/27/15	Thu 8/27/15	3	Professor,Student
5	1.1.3		TBD	1 day	Thu 9/3/15	Thu 9/3/15	56	Professor,Student
6	**1.2**		**Design**	**11 days**	**Tue 9/8/15**	**Tue 10/13/15**		**Professor,Student**
7	1.2.1		UML Modeling I	1 day	Tue 9/8/15	Tue 9/8/15	5	Professor,Student
8	1.2.2		UML Modeling II	1 day	Tue 9/15/15	Tue 9/15/15	57	Professor,Student
9	1.2.3		Software Architecture and Design I	1 day	Tue 10/6/15	Tue 10/6/15	59	Professor,Student
10	1.2.4		Software Architecture and Design II	1 day	Thu 10/8/15	Thu 10/8/15	9	Professor,Student
11	1.2.5		ArchStudio	1 day	Tue 10/13/15	Tue 10/13/15	10	Professor,Student
12	1.3		Midterm Review	1 day	Tue 10/20/15	Tue 10/20/15	60	Professor,Student
13	1.4		Midterm	1 day	Thu 10/22/15	Thu 10/22/15	12	
14	**1.5**		**Implementation**	**13 days**	**Tue 9/22/15**	**Tue 11/3/15**		**Professor,Student**
15	1.5.1		IDE and Eclipse	1 day?	Tue 9/22/15	Tue 9/22/15	58	
16	1.5.2		Eclipse Plug-Ins I	1 day	Thu 9/24/15	Thu 9/24/15	15	Professor,Student
17	1.5.3		Eclipse Plug-Ins II	1 day	Tue 9/29/15	Tue 9/29/15	16	Professor,Student
18	1.5.4		Emacs	1 day	Tue 10/27/15	Tue 10/27/15	13	Professor,Student
19	1.5.5		TBD	1 day	Tue 11/3/15	Tue 11/3/15	61	Professor,Student
20	**1.6**		**Testing**	**2 days**	**Thu 11/5/15**	**Tue 11/10/15**		**Professor,Student**
21	1.6.1		Testing	1 day	Thu 11/5/15	Thu 11/5/15	19	Professor,Student
22	1.6.2		Junit	1 day	Tue 11/10/15	Tue 11/10/15	21	Professor,Student
23	**1.7**		**Maintenance**	**6 days**	**Tue 11/17/15**	**Thu 12/3/15**		**Professor,Student**
24	1.7.1		Version Control	1 day	Tue 11/17/15	Tue 11/17/15	62	Professor,Student
25	1.7.2		Subversion	1 day	Thu 11/19/15	Thu 11/19/15	24	Professor,Student
26	1.7.3		Thanksgiving Day (no class)	1 day	Tue 11/24/15	Tue 11/24/15	25	
27	1.7.4		Thanksgiving Day (no class)	1 day	Thu 11/26/15	Thu 11/26/15	26	
28	1.7.5		GIT	1 day	Thu 12/3/15	Thu 12/3/15	63	Professor,Student
29	1.8		Course Review	1 day	Thu 12/10/15	Thu 12/10/15	64	Professor,Student
30	**1.9**		**Recurring Tasks**	**32 days**	**Tue 8/25/15**	**Thu 12/10/15**		**Professor,Student**
31	**1.9.1**		**Lectures**	**32 days**	**Tue 8/25/15**	**Thu 12/10/15**		**Professor,Student**
32	1.9.1.1		Course Introduction	1 day	Tue 8/25/15	Tue 8/25/15		Professor,Student
33	1.9.1.2		Software Development Process and Activities	1 day	Thu 8/27/15	Thu 8/27/15		Professor,Student
34	1.9.1.3		TBD	1 day	Thu 9/3/15	Thu 9/3/15		Professor,Student
35	1.9.1.4		UML Modeling I	1 day	Tue 9/8/15	Tue 9/8/15		Professor,Student
36	1.9.1.5		UML Modeling II	1 day	Tue 9/15/15	Tue 9/15/15		Professor,Student
37	1.9.1.6		IDE and Eclipse	1 day	Tue 9/22/15	Tue 9/22/15		Professor,Student
38	1.9.1.7		Eclipse Plug-Ins I	1 day	Thu 9/24/15	Thu 9/24/15		Professor,Student
39	1.9.1.8		Eclipse Plug-Ins II	1 day	Tue 9/29/15	Tue 9/29/15		Professor,Student
40	1.9.1.9		Software Architecture and Design I	1 day	Tue 10/6/15	Tue 10/6/15		Professor,Student
41	1.9.1.10		Software Architecture and Design II	1 day	Thu 10/8/15	Thu 10/8/15		Professor,Student
42	1.9.1.11		ArchStudio	1 day	Tue 10/13/15	Tue 10/13/15		Professor,Student
43	1.9.1.12		Midterm Review	1 day	Tue 10/20/15	Tue 10/20/15		Professor,Student
44	1.9.1.13		Midterm	1 day	Thu 10/22/15	Thu 10/22/15		Professor,Student
45	1.9.1.14		Emacs	1 day	Tue 10/27/15	Tue 10/27/15		Professor,Student
46	1.9.1.15		TBD	1 day	Tue 11/3/15	Tue 11/3/15		Professor,Student
47	1.9.1.16		Testing	1 day	Thu 11/5/15	Thu 11/5/15		Professor,Student
48	1.9.1.17		Junit	1 day	Tue 11/10/15	Tue 11/10/15		Professor,Student
49	1.9.1.18		Version Control	1 day	Tue 11/17/15	Tue 11/17/15		Professor,Student
50	1.9.1.19		Subversion	1 day	Thu 11/19/15	Thu 11/19/15		Professor,Student
51	1.9.1.20		Thanksgiving Day (no class)	1 day	Tue 11/24/15	Tue 11/24/15		Professor,Student
52	1.9.1.21		Thanksgiving Day (no class)	1 day	Thu 11/26/15	Thu 11/26/15		Professor,Student
53	1.9.1.22		GIT	1 day	Thu 12/3/15	Thu 12/3/15		Professor,Student
54	1.9.1.23		Course Review	1 day	Thu 12/10/15	Thu 12/10/15		Professor,Student
55	**1.9.2**		**Labs**	**29 days**	**Tue 9/1/15**	**Tue 12/8/15**		**Professor,Student**
56	1.9.2.1		Lab 1 - Microsoft Project	1 day	Tue 9/1/15	Tue 9/1/15	4	Professor,Student
57	1.9.2.2		Lab 2	1 day	Thu 9/10/15	Thu 9/10/15	7	Professor,Student
58	1.9.2.3		Lab 3	1 day	Thu 9/17/15	Thu 9/17/15	8	Professor,Student
59	1.9.2.4		Lab 4	1 day	Thu 10/1/15	Thu 10/1/15	17	Professor,Student
60	1.9.2.5		Lab 5	1 day	Thu 10/15/15	Thu 10/15/15	11	Professor,Student
61	1.9.2.6		Lab 6	1 day	Thu 10/29/15	Thu 10/29/15	18	Professor,Student
62	1.9.2.7		Lab 7	1 day	Thu 11/12/15	Thu 11/12/15	22	Professor,Student
63	1.9.2.8		Lab 8	1 day	Tue 12/1/15	Tue 12/1/15	27	Professor,Student
64	1.9.2.9		Lab 9	1 day	Tue 12/8/15	Tue 12/8/15	28	Professor,Student
65	**1.10**		**Milestones**	**30 days**	**Tue 9/1/15**	**Thu 12/10/15**		**Student**
66	1.10.1		Assignment 1 - No Silver Bullet	4 days	Tue 9/1/15	Wed 9/9/15		Student
67	1.10.2		Assignment 2	3 days	Thu 9/10/15	Wed 9/16/15		Student
68	1.10.3		Assignment 3	5 days	Thu 9/17/15	Wed 9/30/15		Student
69	1.10.4		Assignment 4	5 days	Thu 10/1/15	Wed 10/14/15		Student
70	1.10.5		Assignment 5	5 days	Thu 10/15/15	Wed 10/28/15		Student
71	1.10.6		Midterm	1 day	Thu 10/22/15	Thu 10/22/15		Student
72	1.10.7		Assignment 6	5 days	Thu 10/29/15	Wed 11/11/15		Student
73	1.10.8		Assignment 7	6 days	Thu 11/12/15	Mon 11/30/15		Student
74	1.10.9		Assignment 8	3 days	Tue 12/1/15	Mon 12/7/15		Student
75	1.10.10		Assignment 9	2 days	Tue 12/8/15	Wed 12/9/15		Student