Homework 1

Nicolas Broeking
CSCI 5828 - Foundations of Software Engineering

August 28, 2015

Question 1. Define the term essential difficulties as it is used by Brooks. 5*points*

Essential difficulties are difficulties in software engineering that exist just because of the essential nature of software. These difficulties are born out of the essential properties of software that make software projects unique compared other engineering projects. Essential difficulties take form in four different forms: complexity, conformity, changeability, and invisibility. The nature of software is much different than that of other types of engineering fields and thus introduces difficulties that don't appear in many other fields.

Question 2. Define the term accidental difficulties as it is used by Brooks. 5*points*

Accidental difficulties are difficulties that arise not from the nature of software but from problems in the tools, engineer and software process. A great example that Brooks uses in the reading is syntax errors. These errors come from the programmers inexperience with a language, tools unable to properly show the problem, and other similar issues. However, the nature of software itself did not cause the syntax problems.

Question 3. List and briefly describe the four essential difficulties of developing software systems that Brooks identifies. 12*points*

Complexity - Software problems are inherently complex, Brooks argues "orders of magnitude more complex than hardware, systems. He does say though that this doesn't mean that other systems lack complexity. The problem with software is that as the size of a software system increases linearly the number of parts as well as the types of parts increases exponentially. This leads to software becoming very very hard and complex very quickly.

Conformity - The next essential difficulty mentioned by Brooks is conformity. What makes hardware hard is that software must conform to human laws not physical laws. Other forms of engineering are bound by physical laws. The question of if a problem can be solved or not comes down to physical laws. However, software is not bound to the physical world and thus must conform to human defined laws.

Changeability - Changeability is the third essential difficulty and sprouts from the fact that software is easy to change. With tangible projects one is forced to spend a lot of money to re build the object. However, software is just an abstract idea. You can change and modify it without having to redo the whole project. This then adds extra pressure to software to change as its environment changes. This extra pressure is the problem though. It forces software engineers to not only build the current project but also keep in mine everything that the current goals could turn into over time, consequentially make software extremely difficult.

Invisibility - The final essential difficulty is invisibility. Software is invisible and intangible. This however makes the nature of software development very difficult. Other forms of engineering allow for a physical representation to not only help the client but help the developers as well visualize how everything will work. Humans have adapted to this over years and have developed powerful imagery tools in our brain. Software's lack of visibility prevents us from being able to use these tools thus making software more difficult.

Question 4. Define what Brooks means by a silver bullet and reconstruct his argument as to why he believes there is no silver bullet for software engineering. 10*points*

When Brooks talks about a silver bullet he means ONE tool/practice/solution that if adopted will provide a order of magnitude increase in productivity to all software projects. He argues though that no such magical silver bullet exists for Software Engineering. The reason is because if you look at all the tools and methods in the world today they all target accidental difficulties not essential ones. So in order to have one tool be able to increase productivity by an order of magnitude then two things must be true. Accidental difficulties must make up over 90% of all difficulties in a software project and this technique must be able to eliminate all accidental difficulties. He argues however that neither are true. Accidental difficulties take up much less than 90% of all difficulties and that it is impossible to remove all accidental difficulties from software projects.

Question 5. In lecture, software engineering's relationship to computer science was described by analogy by discussing the differences between a chemist (chemistry) and a chemical engineer (chemical engineering). Define software engineering and its relationship to computer science; make use of the chemist vs. chemical engineer analogy when answering this question. 6*points*

The difference between computer science and software engineering is slight but important. First, lets look at the chemist example. A chemist is a person that is researching possibilities. He will find what chemicals react and turn into what and is only concerned with showing that certain reactions can be possible. A chemical engineer on the other hand takes the results from the chemist and must turn them into a product that is cost effective and reliable and something that provides benefit to humanity as a whole. The same is true for software engineers vs computer scientists. Computer scientists can develop new algorithms and discover new ways to solve theoretical problems. The software engineer needs

to take these algorithms and create a software system the benefits people while minimizing memory use and increasing performance to create a outstanding user experience. Its not just enough to show that you can use an algorithm for a software engineer. They are more concerned with how to use the algorithm in an efficient way to be able to create something useful.

Question 6. In lecture, we discussed the importance of the following concepts to software engineers: abstractions, conversations, specification, translation, and iteration. Define each of these concepts as they are related to software engineering and discuss their importance. 12*points*

These five things are the most important tools used in a software project. If you are fluent in all five areas then you are doing everything you can for a successful software project.

Abstraction - All software problems are complex but by abstracting the problem into layers it enables the engineer to solve the problem using simple pieces. For example opening a file. Instead of having to open the file manually at the machine code level. We have provided file abstractions that allow use of files to become much simpler. Ex: file.open("file.txt")

Communication - Communication is critical to all software projects. Without it everyone ends up on different pages with a different project, and a different goal. Communication allows for the creation of specifications and results to allow for a successful project. You need communication between the designers and the engineers, the team and the clients, and pretty much everyone involved. Communication can show up in many ways but a good example is with testing. Testing provides a way for clients and team to decide when the project is completed and deliverable.

Specification - Specification is the list of rules, plans, requirements, designs, and life cycle chosen for a specific project. This allows you preparation to map out what needs to be done and how for a successful project.

Translation - The ability to move from one abstraction to another. With more and more abstractions allowing us to solve more complex problems the need for these abstractions to work together becomes even greater. Translation is the ability for an engineer to move between these abstractions and use them together without a lot of extra work.

Iteration - Iteration is the means to the end for software projects. We iterate through all the steps and features, step by step until the project is done. Iteration is the structure to how we complete software projects and without any structure then projects would never get completed.