The four essential difficulties discusses in “No Silver Bullet: Essence and Accidents of Software Engineering”

1. **Complexity:**

Software development is complex, when some components are alike, it will be developed as a function, which everyone can use. Where as in other domains like civil engineering, hardware industry, we can find repeated elements more, which make other domains not that much complex.

Complexity is an essential property of the software, where as in other domains like Math’s, physics the complexity phenomenon can be ignored as it is not the essential property. So, in other domains they are able to build the complex system (where complexity is not essence, it has quite a lot of properties) in to a simplified models and able to propose solutions for complex ones.

Because of complexity so many issues arise like:

1. We ignore some cases, which we cannot think off and which causes a security vulnerability and which opens a door for software hackers to hack the software.
2. Understanding differences because of complexity in the software between team members, which makes developers in team in ambiguous situation and develop a software which is a flaw and not useful to anyone. Example like FMS (Flight Management System) a complex software developing has taken years because of its complex nature. It has taken years because to understand such a huge system takes time and to make all people to understand will take time.
3. Because of complexity involved in software development, a lot many projects are not completing all the phases and come to user for their usage. Ex: In R&D centers, a lot many ideas will come up, but only few ideas comes to user for their usage, a lot many will have their racks in R&D centers storage areas.
4. Because of this complexity, it happens to miss the deadlines in a project, cause budget overrun and software developments get halted.
5. This causes, not only technical complexity, but also non-technical complexity, because non-technical managers will not be able to under the complexity and cannot have proper plan and design and they cannot have overview of the entire software, as something’s can be missing.
6. Complexity in functionality design, which makes harder to use the functionality.
7. Because of the complexity, the already existing software’s cannot be reused and extended, because of its complexity, members involved in design are not able to consider future requirements and able to fit in to currsent software.
8. Because of its complex nature, the growth in software industry is not that much, when compared to other domains.
9. As software complexity increases, number of states also linearly increases.
10. **Conformity:**

A lot of complexity in software developing is arbitrariness.

Suppose there is a change in banking domain happening, then software has to adapt to that arbitrary changes in that domain. It is not possible to consider arbitrary changes in to current software and develop.

The most complexity comes from confirmation with other interfaces which are not standardized.

And also, one more thing is software is recently came in to picture and also it is invisible and it cannot be perceived, where as in others it is confirmed by perceiving it. Ex: Building can be perceived by touching, Cars can be perceived by driving it.

1. **Changeability:**

Software is the one that is constantly under pressure for change. Why only software has this much pressure because, it is the one allows changes to the functionality. Whereas other domains like automobile, manufacturing industry are not that much under the pressure, because once they are manufactured, to make a change on it is not easy, as they cannot adopt it. They are superseded by other new models. As, software has chance to allow changes easily in to it, it is under constant pressure for changeability forever. And one major thing for changeability is software is developed from human brains, so if thinking changes and get new thoughts, it can be easily fit in to it, as software is easily malleable to changes.

And lastly, software is continuously changing because software is used across all domains, whenever there is a change in requirement in that domain, software is under pressure to change.

Ex: Suppose in Medical industry software is being used, then suppose new method comes to cure the disease using hardware, sensors and some techniques then software needs to be changed for that system. In Medical industry Laser eye treatment has taken a lot of changes, it is because of software and hardware changes only.

1. **Invisibility:**

Software is invisible, as it does not have any geometric representation like chips have diagrams, computers have connectivity schemas, cars have models etc., and one more thing is software does not embed into space. When we try to draw diagram for software structures we have many diagrams like representation for structures, data flow diagrams, control flow diagrams,

Activity diagrams and dependency diagrams. It is difficult to get everything in a single diagram.

There is a lack of visualization in the software, because of this human brains are not using their visualization skills. Humans get more through visualization, whereas software is invisible, so it has become difficult for developers and teams to work on.

2.

I have followed Waterfall Model software method during my project work tenure in Honeywell.

There are some attacks on this waterfall software method.

In the first phase, we have collected the requirements from the customer. After the requirement collection we understand the requirements and if any issues discuss with customer regarding requirements. Later we document the requirements and have a review of it. Then progress to next phase.

In this phase we design the software in accordance with requirements. But whereas requirements collected from customer would be one line and give very less clarity. So the design we have developed may not be robust and may not be feasible with later changes from customer.

Software is represented through many diagrams like class diagrams etc., through UML modeling in design phase. Because of these many diagrams in UML it makes tough for us to draw these diagrams and have complete understanding of the entire software.

Because of less clarity on entire system, documenting the requirements and design in an abstract manner causes the software more complex. From this complexity nature, we faced difficulty to understand the requirements and code them during coding phase. Because of this, when the product is released we got some security issues and loop holes in system.

In the coding phase, we code the software for requirements. But as requirements are in an abstract manner, we code them as per understanding. But because of this abstract manner, we might not be understanding all the requirements as per customer perspective. And when software is shown to customer, he may not be satisfied with software. So changing the software design at that point makes more complex.

During testing, testers have faced a lot of issues like they wouldn’t have clear understanding on system and because of this some bugs would be rolled out to customer. As they need to test the software with a lot of test cases, because of complex nature of software, some scenarios they might not have tested and because of this some issues will not be uncovered and the software is released with some bugs.

In testing, if environment is dynamic and when we get the values from different interfaces, testing also becomes complex. Because if it is dynamic nature, some corner cases can be missed out by testers, and can have a flaw in product.

Also, the continuous change in requirements from customer in later phase’s cause’s problems in design and architecture of system. As some requirements may not fit into current design and then the entire design needs to be changed for this. Thus, the changeability is a continuous attack on software development in this method.

Because of complexity in the system, there would be some security vulnerability issues in system.

Ex: In a protocol development in our project, while developing we have not considered multiple call backs from same instance at same time. Because of this it had caused a lot of changes in the software design and coding. Because of this our project is not able to meet deadline for one time.

Because of the changes and complexity of software, it can cause project delay, budget over run and it makes system complex, which is already having a lot of complexity.