Writing software is a complex process that involves multiple challenges.

The main difficulties developers encounter can be divided into two groups: some are **accidental** – complexities that are not related to the problem of the software itself, but are usually due to a fault of the programmers. Typical difficulties are linked to the lack of experience and thorough understanding of a language, or a poor organisation resulting in little time to finish the program in a proper way.

According to Brooks, however, most complexities are **essential**, inherent in the nature of building software, and articulated in multiple dimensions: the more functionalities a program includes, the larger it becomes in size.

Several developers are needed to write large systems, which increases complexity, as people produce code with different styles.

Software is easier to modify than hardware, but changes may introduce new faults, although they are necessary to adapt to new machines.

Also, designing a software system is difficult because a program is a hardly visualizable entity, unlike the hardware on which it is installed.

The core of the problem is the deep **difference between progress of hardware and software** in the last decades: over the years, technology improved, transistors doubled in circuits and the hardware cost dropped. Moore’s law may be considered the silver bullet of hardware systems – machines that ten years ago were thought to be too advanced and complex, are reality today – but does such a certainty exist for software?

Brooks suggests a few techniques that attacked the accidental complexities. For example, the logic and conceptual structures of high-level languages are closer to the way developers think, so that they do not need to worry about the machine’s lower levels.

There are hopes for a silver bullet, thanks to the advancement of technologies such as AI, expert systems and automated code generation. Nevertheless, none of this will magically make the production of software easier.

More promising, instead, are solutions that directly address the essence, that is the **difficulty in rendering a business idea into a design**. Among these, the focus on improving requirements and relying on a loop of rapid prototyping and frequent feedbacks, with the help of skilled designers, seems the best technique to grow (rather than build) better software.