1. **Introduction**

In the history of software engineering, engineers have constantly searched for a method that could completely solve the challenges of software development. However, in 1986, renowned computer scientist Fred Brooks presented a groundbreaking view in his paper *"No Silver Bullet: Essence and Accidents of Software Engineering"*: there will be no single technology or method capable of increasing software development productivity by an order of magnitude.

A "silver bullet" refers to a mythical weapon that can instantly eliminate all problems with a single shot. The idea of "no silver bullet" means that in software development, there is no magic solution that can instantly remove all difficulties or dramatically improve efficiency. Fred Brooks argued that the greatest challenges in software development stem from the inherent complexity of the problems themselves, not from the limitations of implementation tools. This viewpoint has remained highly influential and is widely regarded as a foundational theory for understanding the true nature of software engineering.

1. **Identified Key Challenges**

In "No Silver Bullet: Essence and Accidents of Software Engineering", Fred Brooks divides the difficulties of software engineering into two categories: essential and accidental. He particularly emphasizes the essential challenges, as these cannot be completely eliminated through technological means. The following are the key challenges in software development discussed in the article:

1. **Complexity**

Software systems are inherently complex because they need to handle a wide variety of states, inputs, logic, and interactions. Each module is highly interconnected, and modifying one part may impact the entire system. Moreover, unlike other engineering products that often have repetitive structures, software design is usually customized, with no universal "template" to rely on. As the system grows in scale, this complexity increases exponentially, significantly raising the difficulty of development, testing, and maintenance.

1. **Conformity**

Software must adapt to various external environments and constraints, such as laws and regulations, business processes, user requirements, and hardware platforms. These external demands are often complex and even conflicting, which means the software must continuously adjust itself to maintain conformity. Because these requirements come from multiple sources, it's nearly impossible to establish a single, unified principle to simplify them. As a result, developers are faced with the difficult task of constantly adapting the software to meet these diverse expectations.