Introduction to “No Silver Bullet” and its relevance to software engineering.

- Explain the identified key challenges.

简介“没有银弹”以及其与软件工程的相关性

解释已确定的关键挑战

“没有银弹” 是Frederick P. Brooks, Jr.在 1987 年发表的论文 No Silver Bullet: Essence and Accidents of Software Engineering中提出的论断。布鲁克斯认为软件工程本质难题的根源在于软件本身的性质，这使生产力、可靠性或简易性难以实现数量级的提升。尽管过去的高级语言、分时技术等创新解决了意外难题，但布鲁克斯断言，软件工程没有通解，也就是说，不存在任何单一的技术或管理方法能够像银弹一样彻底解决软件开发的本质问题[1]。

"No Silver Bullet" is an assertion by Frederick P. Brooks, Jr. in his 1987 paper No Silver Bullet: Essence and Accidents of Software Engineering. Brooks argued that the root of software engineering's essential difficulties is the nature of the software itself, which makes it challenging to achieve orders of magnitude increases in productivity, reliability, or simplicity. Although past innovations such as high-level languages and time-sharing techniques have solved unexpected problems, Brooks asserts there is no general solution to software engineering. That is, no single technical or managerial approach can completely solve the essential difficulties of software development like a silver bullet [1].

这一观点与现代软件工程仍息息相关，即便过了几十年，软件工程仍在努力应对布鲁克斯所指出的那些挑战。随着技术进步，虽然开发工具和方法不断更新，但软件复杂性依旧在增加，例如数十年后提出的敏捷开发虽然使开发过程更加灵活，但随着需求不断变更，代码也会逐渐变得复杂[2]。布鲁克斯在文章中提醒开发者解决根本的复杂性需要系统性、渐进式的努力，而非革命性的捷径，因此像要避免过度依赖某一种新技术来解决所有问题 。

This view is still relevant to modern software engineering. Even after decades, software engineering still struggles to cope with the challenges Brooks identified. As technology advances, although development tools and methods are constantly updated, software complexity is still increasing. For example, agile development proposed decades later makes the development process more flexible, but as requirements change, the code will gradually become more complex[2]. In the article, Brooks reminds developers that solving fundamental complexity requires systematic and incremental efforts rather than revolutionary shortcuts. Therefore, they should avoid over-reliance on a single new technology to solve all their problems.

已确定的关键挑战主要是本质困难，包括复杂性、一致性、可变性和不可见性。

The key challenges identified are mainly essential difficulties, including complexity, conformity, changeability, and invisibility.

复杂性

软件系统由于其非重复性和具有大量状态的特性，比其他任何人造结构都复杂。软件组件以非线性方式交互，这会导致复杂性随着系统规模的扩大而呈指数级增长。这种复杂性阻碍了开发团队之间的沟通，无论是更新还是维护都变得非常困难，造成了巨大的学习和理解负担。

Complexity

Software systems are more complex than any other artificial structure due to their non-repetitive nature and the large number of states they have. Software components interact in a non-linear way, which causes complexity to grow exponentially as the size of the system increases. This complexity hampers communication between development teams and makes updates and maintenance complex, burdening learning and understanding.

一致性

软件通常必须适应任意外部施加的要求，例如适配其他接口，法规要求等，由于这些要求是不同的人定义且毫无规律，这会导致代码变得复杂，开发人员也会优先考虑兼容性而非代码的简洁优雅，这往往会导致架构脆弱，仅靠对软件的重新设计无法消除这种复杂性。

Conformity

Software usually has to adapt to arbitrary externally imposed requirements, such as adapting to other interfaces, regulatory requirements, etc., which can lead to code complexity because these requirements are defined by different people and are not regular. Developers also prioritize compatibility over code simplicity and elegance, which often leads to architectural fragility. This complexity cannot be eliminated by simply redesigning the software.

可变性

与生产出来后很少修改的实体产品不同，软件可以被相对轻易的改变。用户需求不断变化，硬件平台的更新以及新的监管要求等会导致软件系统需要不断且频繁的修改。这种可塑性虽然有利，但如果底层架构缺乏弹性，每次更改都可能带来不稳定的风险。

Changeability

Unlike physical products that are rarely modified after production, software can be changed relatively easily. Changing user needs, updates to hardware platforms, and new regulatory requirements will lead to constant and frequent changes to software systems. While this malleability is beneficial, each change can pose a risk of instability if the underlying architecture is not resilient.

不可见性

软件缺乏自然的几何表示，因此难以可视化和沟通。软件结构是抽象且多维的，尽管在简化和限制软件结构方面取得了一定进展，但软件本质上仍然是不可视的，这种不可见性阻碍了设计的一致性和团队协作，例如在大型分布式系统中，开发人员在没有详尽文档的情况下难以追踪跨服务交互。

Invisibility

The software lacks a natural geometric representation, making it difficult to visualize and communicate. Software structures are abstract and multidimensional, and despite some progress in simplifying and restricting software structures, software is still essentially invisible. This invisibility hinders design consistency and team collaboration. For example, in large distributed systems, developers have difficulty tracking cross-service interactions without detailed documentation.

过去的技术突破已经解决了意外困难，但本质困难依然存在，且无法被单一 “银弹” 攻克。

Past technological breakthroughs have solved accidental difficulties, but the essential difficulties still exist and cannot be overcome by a single "silver bullet".

[1]

[1] Brooks, “No Silver Bullet Essence and Accidents of Software Engineering,” *Computer*, vol. 20, no. 4, pp. 10–19, Apr. 1987, doi: [10.1109/MC.1987.1663532](https://doi.org/10.1109/MC.1987.1663532).

[2] V. Antinyan, M. Staron, W. Meding, P. Österström, H. Bergenwall, J. Wranker, J. Hansson, and A. Henriksson, “Monitoring evolution of code complexity in agile/lean software development: A case study at two companies,” in Proc. 13th Symp. on Programming Languages and Software Tools (SPLST), 2013, pp. 1–15.