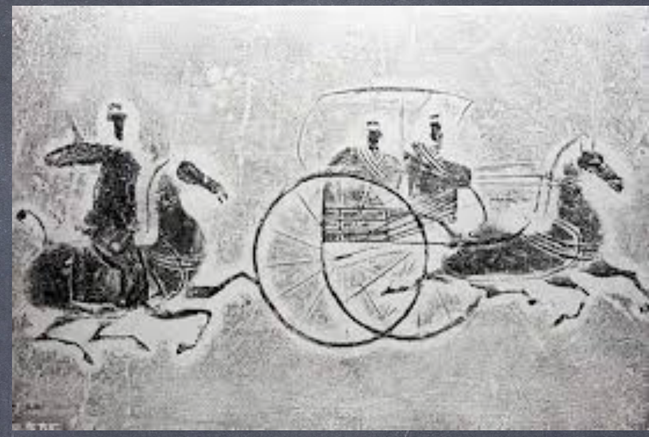
# 软件工程的过去，现在与将来

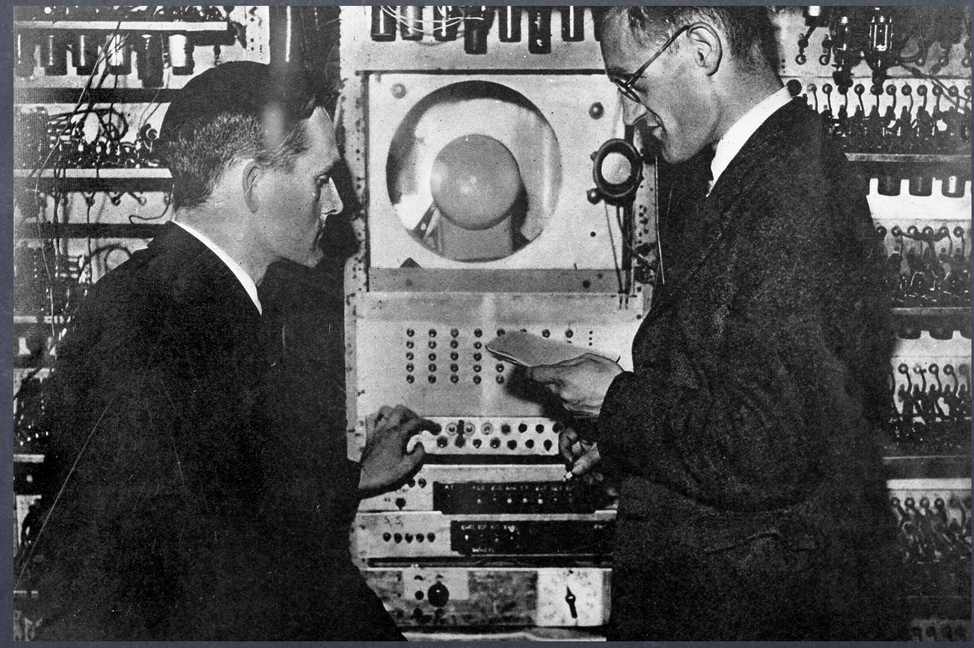
# 工程由来已久



- 工程（Engineering）是一个古老的话题
  - 工程的历史可以追溯到轮子的发明（甚至更早）。
  - "工程师"一词来源于14世纪，当时"engine'er"指的是制造军事机器（如投石机和其他攻城器械）的人
- "Engine"一词源自拉丁语"ingenium"（约1250年），意为"天赋，特别是智力，因此引申为巧妙的发明"。
  - 这一词源反映了工程师的核心能力：利用智力和创新来解决问题和发明新技术。

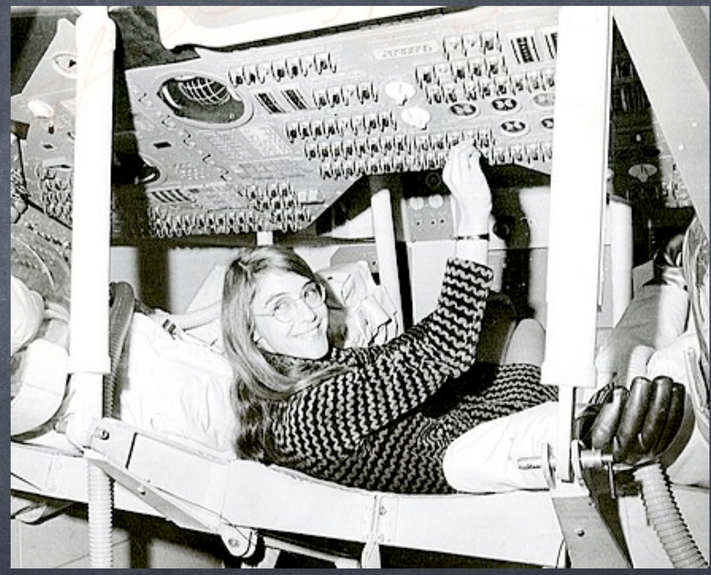
# 软件的诞生

- 软件首次亮相普遍认为于1948年，并且直到1952年才广泛被称为"软件"。
  - 软件第一次被存储程序计算机成功加载并执行是在1948年6月21日上午11点，地点是曼彻斯特大学的曼彻斯特名为"Baby"的小规模实验机（Manchester Mark 1的前身）。这段程序由汤姆·基尔本（Tom Kilburn）编写，旨在计算整数$2^{18}$的最大因数。



# 软件工程

- 1963年到1964年间，在为阿波罗计划开发导航和制导系统的过程中，计算机科学家和系统工程师玛格丽特·汉密尔顿（Margaret Hamilton）创造了"软件工程"（Software Engineering）这个术语。



- 汉密尔顿认为软件开发人员应当被称为工程师，因为他们的工作复杂性和重要性与传统工程学科相当。

# 软件危机

- 1965年，"软件危机"（Software Crisis）开始显现，原因是软件开发无法跟上硬件发展的步伐。这一时期暴露出许多软件开发中的问题，包括预算超支、进度延误、需要大量调试、不满足用户需求、大量维护需求（即使有可能维护），或者干脆从未完成。

PRODUCTION

## Software gap—a growing crisis for computers

**Shortage of programmers—and the fruits of their solitary art—is stunting growth of computer use and costing industry hard cash**

**The computer,** man's most complex industrial product, can be cranked out in quantity by mass-production techniques. But it is powerless to solve problems, sort data, or store information without instructions.

The process of writing instructions—or programs—is a new human intellectual art, not a mechanical or electronic skill. And this factor is setting limits on the usefulness of the computer far below those imposed by electronic technology.

David B. Hertz, a consultant at McKinsey & Co., summed up the problem at an American Management Assn. meeting earlier this year: "The overriding issue is people—specifically, skilled computer personnel . . . Already, the supply is far short of the demand, and the gap is widening inexorably. For the foreseeable future, there is literally no possibility that we shall have enough trained people to go around."

The implication of this gap for business and science, he added, is that "use of computer systems five years hence will be seriously hobbled." There are only about 120,000 programmers in the U. S.—and right now there's a shortage of 55,000 or more of these new professionals, according to some estimates.

**Special type.** Programming is a young profession, born only a dozen years ago when computer makers first began to turn the fuzzy outlines of business problems into the precise electronic language of the machine. Since then, the number, power, and widespread application of computers has far outstripped the supply of programmers. Delays and extra costs have been the price to both users and computer makers.

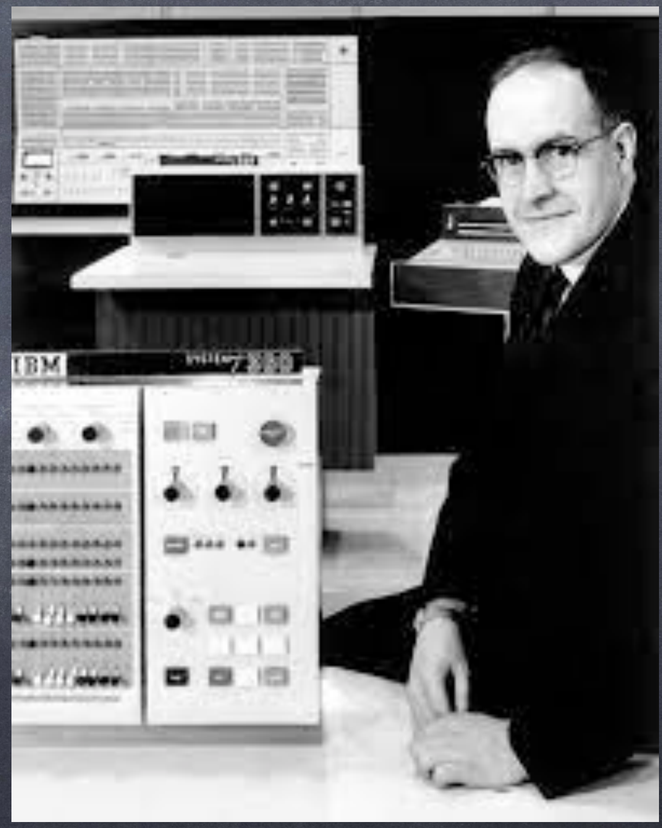
To top it off, the computer has gotten so complex it is demanding

**Flow-chart** shows the program steps computer will perform.

**Debugging** on software company's own computer works out flaws, misstatements in program, so it will perform jobs client wants when run on his computer.

# 软件危机例子

- 成本和预算超支：OS/360操作系统是一个经典的例子。这个持续了十年的项目最终产出了当时最复杂的软件系统之一。
- OS/360是最早的大型软件项目之一，参与开发的程序员多达1000人。弗雷德·布鲁克斯在《人月神话》一书中声称，他犯了一个数百万美元的错误，即在开发开始前没有制定一个连贯的体系结构。



# 软件危机例子

- 软件缺陷可能致命：一些用于放射治疗设备的嵌入式系统发生了严重故障，导致患者接受了致命剂量的辐射。这些失败中最著名的是Therac-25事件。

# 软件危机例子

- 功能无法完成：丹佛国际机场行李处理系统曾被希望设计成世界上最先进的行李系统，通过自动化技术处理整个机场的行李。
- 然而，该项目以惨败告终，使得机场在其他部分已经准备就绪后，仍无法运行达16个月之久，预算超支5.6亿美元，最终系统仅实现了预期功能的一小部分。

# 软件工程会议

- 1968年，首次北约软件工程会议（NATO Software Engineering Conference）召开。一年后，第二次会议也如期举行。这些会议旨在解决软件危机问题，并为软件开发建立指导方针和最佳实践。
- 从此，"软件工程"成为人所共知的主题！

Peter Naur和Brian Randell主编的会议报告中这样写道："我们特意选择'软件工程'这个颇具争议性的词，是为了暗示这样一种意见：软件的生产有必要建立在某些理论基础和实践指导之上

没有银弹！

# 软件工程迅速发展

- 软件危机持续存在，而软件工程师们则努力纠正这一局面。从上世纪70年代到90年代软件工程迅速发展，新的理念、编程语言和工具相继出现。
  - 每一项工作都曾经被寄托成为解决软件危机的"银弹"！

# 形式化方法

- 形式化方法（Formal methods）：将形式化的工程方法应用于软件开发，主张证明所有程序都是正确的。
  - 模型检查（Model Checking）：将程序建模为状态机，系统地探索所有可能的系统状态，以发现潜在的错误（Spin、NuSMV）
  - 定理证明（Automated theorem proving）：给出程序（甚至是芯片）的形式规约，然后用自动化定理证明技术证明程序的正确性（以及可以帮助合成正确的软件）
  - 抽象解释（Abstract interpretation）：通过对程序行为的过近似（over-approximation）来分析和验证程序的特性，旨在解决无法枚举所有可能的程序状态下的程序分析

# 形式化方法

- 形式化方法的问题：
  - 构建一个普通程序的形式规约需要时间（以及需要专业领域的专家）
  - 证明本身是复杂度非常高的方法，一般需要在一个指数级的搜索空间中进行搜索，因此一般用于安全攸关领域
  - 缺乏一个统一的业界标准（工业界一般很难评估其价值）

# 编程范型

- 新的编程范型（Programming paradigm）不断被提出
  - 当人们开始大量依赖"Goto"语句时，产生的代码越来越难维护和修正（Go To Statement is considered harmful by Dijkstra），为改变这一现状，很多新的程序范型被提出
    - 结构化编程：用if，else，while等控制语句块来替代goto（FORTRAN，C）
    - 面向对象编程：将数据和函数统一抽象管理（Java、Python、C++）
    - 声明式编程：屏蔽过程，只要描述问题，更高的抽象（SQL、Datalog）

# 编程范型

- 然而编程范型只能减少程序员发生错误，无法避免错误发生
  - 实际上随着编程范型越来越高级，编程的门槛变低了，更多的人可以利用编程解决问题，随之而来的就是越来越多的bugs！

# 管理

- 软件的危机本质是"人"会犯错，因此应该加强人的管理
  - 规范代码的编写（文档、各种代码标准）
  - 加强软件过程生产的过程管理：
    - 需求分析、详细设计、实现、测试、维护
    - 每个过程都有持续的追踪和定量的分析，可以不断反馈修正（如CMM模型）

# 管理

- 与范型一样，管理同样无法避免错误的发生
  - 并且增加了开发的成本（小型作坊完全无法支付起一个管理团队）

# 没有银弹

- 1986年，弗雷德·布鲁克斯发表了《没有银弹》一文，主张没有任何单一的技术或实践能在10年内使生产率提高10倍。对于银弹的争论在接下来的十年中持续激烈。最终，几乎所有人都认同，永远找不到银弹。
- "没有银弹"不是软件工程的失败，反而说明证明软件工程最终已经成熟（其他职业也没有银弹）
  - 其指明了寻找单一成功关键的努力不会奏效。所有已知的技术和实践只能对生产率和质量进行渐进式改进

# 大数据的挑战："卷"

# 互联网时代来袭

- 蒂姆·伯纳斯-李（Tim Berners-Lee）于1990年开发了World Wide Web，这是第一个网页浏览器。他还创建了HTTP、HTML以及第一个网页，描述了他所创建的内容，互联网开始兴起！
- 与此对应，1990年代先后诞生了Python、Java和Javascript，分别是现如今网络爬虫、应用后端、前端的主流语言

# 互联网时代来袭

- 从此，全球范围内的信息显示和电子邮件系统的需求快速增长。程序员需要以前所未有的速度处理多媒体数据如插图、地图图像，音频流，视频流等
  - 而当时几乎没有什么著名的方法来优化海量的数据的存储和分析
  - 比如，到了2001年，此时的Google需要处理13亿个网页的检索，传统的软件解决方案无法应对！

# 移动互联网

- 移动智能设备（手机、平板等）的出现加剧了数据的产生，至2023年全球互联网用户：全球约有53.5亿互联网用户，每人每天可能产生约15.87 TB的数据。
  - Facebook：每天产生约4,000 TB的数据
  - TikTok：平均每天产生约7.35 TB的数据。
  - YouTube：2023年每天托管超过72万小时的视频，相当于约4.3 PB的数据。

# 大数据时代下的软件工程

- 软件开发的架构发生了巨大的变化
  - 硬件上通过增加更多的服务器或节点来提高系统的处理能力，而不是依赖单个服务器的性能。
  - 架构逐步转向分布式架构：如分布式计算平台Map Reduce、Haddop，存储HDFS (Hadoop Distributed File System)、NoSQL 数据库（Cassandra、MongoDB），实时数据流处理架构（Kafka、Flink），使用Redis等缓存系统来减少数据访问的延迟和频率

# 大数据时代下的软件工程

- 程序应用的交互开发逻辑也发生了改变，个人单机已经无法高效处理大数据的应用
  - 云计算的开发模式占据主导：显示前端和业务后段进行分离
  - "计算即是服务"的理念深入人心：基础设施即服务(IaaS)、平台即服务(PaaS)、软件即服务(SaaS)的出现让软件开发变得越来越容易！

# 大数据时代下的软件工程

- 随着计算资源的普及，许多小型组织对软件需求的不断扩大，对廉价软件解决方案的需求促使了更简单、更快速的软件开发方法的发展
  - 敏捷开发
  - DevOps

# 轻量级开发

- 敏捷软件开发（Agile software development）指导软件开发项目在不断变化的期望和竞争市场中快速演进。
  - 敏捷开发的支持者认为，像 TickIT、CMM和ISO 9000 这样的重文档驱动过程正逐渐失去重要性。轻量级开发的理念如极限编程（Extreme Programming）成为软件开发的主流模式
  - 其主张以人为本、迭代和增量的开发方法，利用测试驱动的方式快速交付高质量的软件，并适应不断变化的需求

# 轻量级开发

- DevOps：将"敏捷"拓展到测试之外：部署和维护也同样需要敏捷的原则
  - 其将开发和运维整合成一个无缝、协作的过程，彻底改变了传统的软件开发生命周期。通过专注于自动化、持续集成和共享责任，使团队能够更快速、更可靠和更高效地交付软件。
  - 一些技术如持续集成、持续部署、微服务架构在不断优化软件开发的流程

# 智能软件时代

# 人工智能时代

- 多种计算平台的发展促进了人工智能不同方面的进步，从专家系统到深度学习，再到具有开放接口的机器人平台
  - 近年来，深度神经网络和分布式计算的进步催生了大量软件库，包括Deeplearning4j、TensorFlow、和PYTorch。

# AI for SE

- AI在各种层面可以辅助软件的开发和维护
  - 自动化编程：AI能够自动生成代码，减少人工编写代码的工作量。例如，GitHub Copilot 使用自然语言处理技术，从开发人员输入的注释或代码片段中生成完整的代码。
  - 误检测和修复：AI系统可以自动检测代码中的错误并建议修复方案。例如，DeepCode使用机器学习模型来分析代码，提供优化和错误修复建议。
  - 代码审查：AI可以自动进行代码审查，检测潜在的安全漏洞和性能问题。例如，CodeGuru通过机器学习模型分析代码，提供审查和优化建议。

# AI for SE

- AI在各种层面可以辅助软件的开发和维护
  - 自动化测试生成：AI可以根据代码自动生成测试用例，提高测试覆盖率和效率（如Diffblue Cover）
  - 自动化运维：AI可以自动监控系统性能，预测潜在问题并自动采取修复措施（如Dynatrace）。

# SE for AI

- AI本身也是软件，同样也会存在缺陷问题，因此软件工程的一些方法也适用于保障AI的开发和维护
  - 目前还处于前沿的研究，如何利用软件工程的方法来评测和提高AI的安全性、功能正确性、鲁棒性、准确率以及可解释性等
  - 常见的方法如蜕变测试、形式化建模、N-version construction等

# 大模型时代的软件工程

- OpenAI的ChatGPT等大模型使的成功无疑是目前软工世界甚至是计算机世界（可能也是现实世界）最为热门的事件
  - 其问题求解的精度远超以往任何一个机器学习模型
  - 软件工程以往的幻想都逐步成为了现实：工业可用的自动代码生成、修复等等无不在告诉我们，未来已来！

未来

# 未来

- 在可见的未来，软件工程将持续围绕着大数据和人工智能进行不断进行优化和改进
  - 但每次软工的"大"的发展都是随着新的需求出现，而已有工程方法难以满足才会催生新的技术和新的模式
    - 比如量子计算、VR等都可能会成为下一个中心

# Thanks!