南京大学
NANJING UNIVERSITY

软件工程概论

——从程序设计到软件工程

张 天

SEG - Software Engineering Group

ztluck@nju.edu.cn

2022年 秋季

软件工程所面对的问题

- **大规模、复杂软件系统的开发和维护**
  - 现实问题的复杂性
- **如何高效地开发可信的软件系统？**
  - 开发的过程可控、质量可控

Warm up

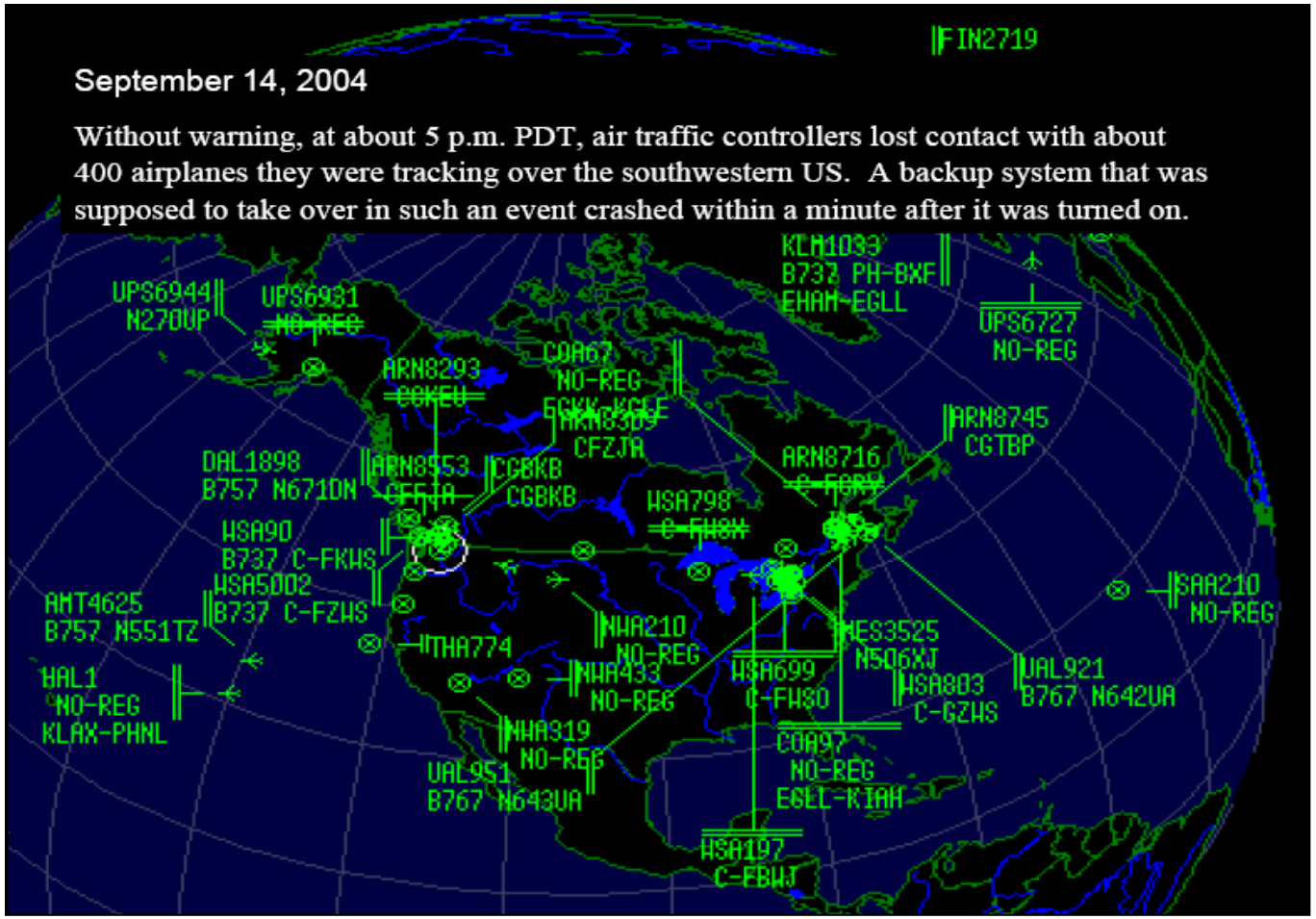
- **从68、69年“软件危机”的爆发，产生了“软件工程”这样一个专门的领域开始，历经了近半个世纪，如今的软件系统依旧脆弱和不可信！**

- *让我们简单回顾一下……*

美国电力监测与控制管理系统

多计算机系统试图同时访问同一资源引起的**软件失效**

美国空管软件

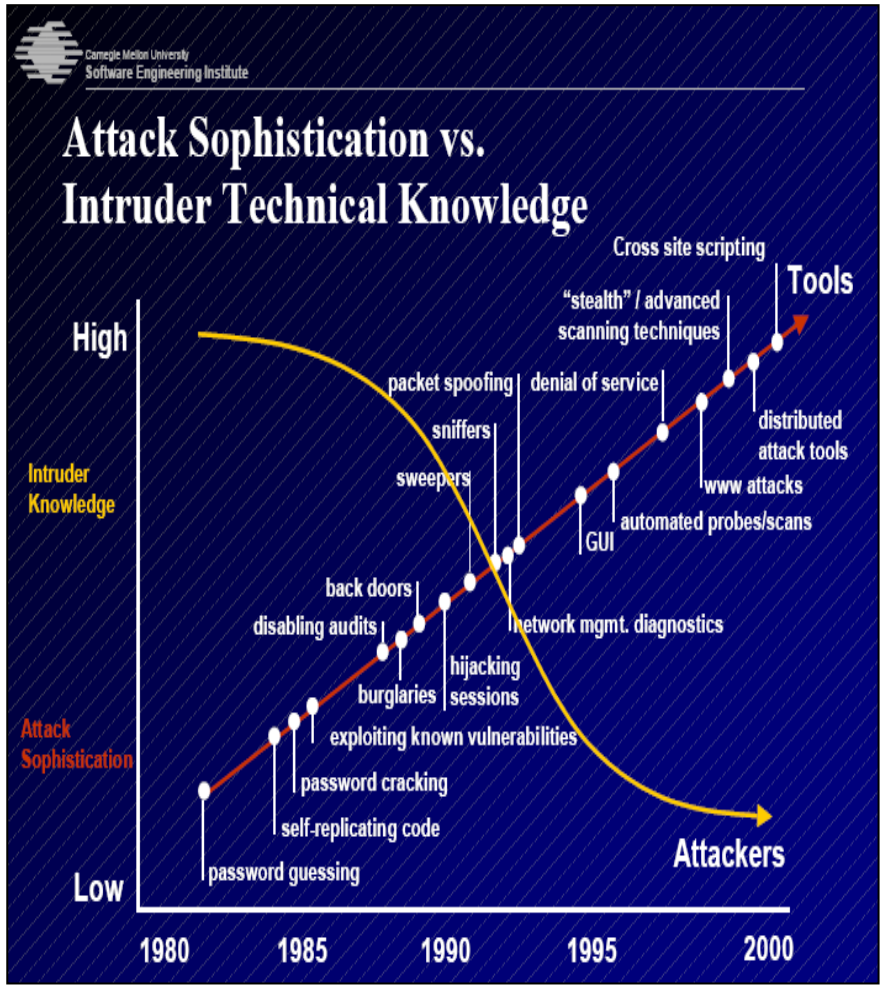


原因是**空管软件时钟缺陷**

东京证券交易软件

2005年11月1日，东京证券交易所因为**软件升级出现系统故障**，导致早间股市“停摆”。

互联网软件安全的几个数据



- 约80%的家庭用户感染了Spyware
- 美国2004年网络犯罪非法谋利105亿美元
- 50%以上的安全漏洞是由软件缺陷引起的

误杀百万电脑 诺顿致歉

由于误将中文Windows XP中的系统文件当作木马程序屏蔽掉，2007年5月17日和18日，诺顿杀毒软件导致全部安装了该软件的计算机系统瘫痪。根据保守估计，被误杀的电脑在中国内地不下百万台。昨天，诺顿产品的母公司赛门铁克公司发表声明，向用户致歉并发布了官方解决方案。

Microsoft的承诺

- 就间接损害不赔付责任：
  - 在法律所允许的最大范围内，Microsoft Corporation或其他供应商绝不就因使用或不能使用本“软件产品”所发生的其他损害负赔偿责任，即使Microsoft Corporation事先被告知该损害发生的可能性。

- 若为Windows的每一次Bug出现赔偿1美分，比尔盖茨早已一贫如洗。

为什么？

- 软件系统的规模和复杂性正在不断超出人类驾驭的范围。
- 软件产品没有“质保”，没有“三包”。
- 任何机构和个人都无法确保所开发的软件一定没有问题。

怎么办？

操作系统、程序设计语言、结构化分析与设计、面向对象分析与设计、UML、体系结构、数据库、设计模式、分析、测试、验证、度量、配置、CMMI、开源、过程管理、敏捷开发 ……

没有银弹！

现 状

直到今天，我们仍在摸索中前进！

“软件工程”是个合集

“软工”初步印象

如果说我们真的摸到了什么，那么**软件工程**无疑是人类在“高效率、低成本地开发高质量软件”的征途上所集齐的整套装备！

思考以下说法

- 我们已经有了一本写满软件开发标准和规程的宝典，难道不能提供我们所需要的所有信息吗？
- 如果我们未能按时交付计划，可以通过增加程序员的人数而赶上进度。
- 如果决定将软件外包给第三方公司，就可以放手不管，完全交给第三方公司开发。
- 当我们完成程序并将其交付使用之后，我们的任务就完成了。
- 对于一个成功的软件项目，可执行程序是唯一可交付的工作成果。

思考以下说法

- 软件工程将导致我们产生大量无用文档，并因此降低工作效率。
- 直到运行开始，才能评估其质量。
- 有了对项目目标的大概了解，便足以开始编写代码，可以在之后的项目开发过程中逐步充实细节。
- 虽然软件需求不断变更，但是因为软件是弹性的，因此可以很容易的适应变更。

软件神话

- 以上说法就是所谓“软件神话”
- 软件神话
  - 关于软件及其开发过程的一些被人们盲目相信的说法
  - 今天，大多数有经验的软件工程师已经意识到软件神话的本质，它实际上误导了管理者和从业人员，有时甚至产生非常严重的后果
  - 但是直到今天，由于习惯和态度的根深蒂固，软件神话的遗风犹存

软件工程的诞生

- 20 世纪60年代以前
  - 软件设计往往只是为了一个特定的应用（数学计算性的）
  - 采用密切依赖于计算机的机器代码或汇编语言
  - 设计软件往往等同于编制程序
- 60年代中期
  - 大容量、高速度**计算机**的出现；**高级语言**开始出现；**操作系统**的发展引起了计算机应用方式的变化；大量数据处理导致第一代**数据库管理系统**的诞生
  - 软件系统的规模越来越大，复杂程度越来越高，软件可靠性问题也越来越突出，开始爆发众所周知的**软件危机**
- 1968、1969年，最初的软工大会
  - 为了解决问题，北大西洋公约组织（NATO）在1968、1969年连续召开两次著名的NATO会议，并同时提出**软件工程**的概念

思考：体会软件危机背后的深层次原因

软件的特殊性

1. 软件是设计开发的，而不是传统意义上生成制造的
2. 软件不会磨损（但是有失效）
3. 虽然整个工业向着基于构件的构造模式发展，然而大多数软件仍是根据实际的顾客需要定制的

Key Points:

- 在指定解决方案之前要理解问题
- 设计是一项关键的软件工程活动
- 质量和可维护性都来自于好的设计

计算机的世界 vs 现实世界

- 帮助人类解决问题的工具
  - 计算机科学是解决问题的科学
  - 只能识别2个符号：0，1
- 人类大脑的拓展和延伸
  - 时间
  - 空间
- 计算机面对的问题
  - 不能解决的问题
  - 需要无限资源才能解决的问题（例如500年）
  - 利用有限资源就能解决的问题
- 计算机在软件的指挥下解决问题

软件的复杂性是这样来的

软件工程两个定义

- [Nau69]（软件工程是）建立和使用一套合理的工程原则，以便经济地获得可靠的、可以在实际机器上高效运行的软件。

软工概念产生之初

- [IEEE93]软件工程是：

  (1) 将系统化的、规范的、可量化的方法应用于软件的开发、运行和维护，即将工程化方法应用于软件。

  (2) 在(1)中所述方法的研究。

软工的发展相对成熟

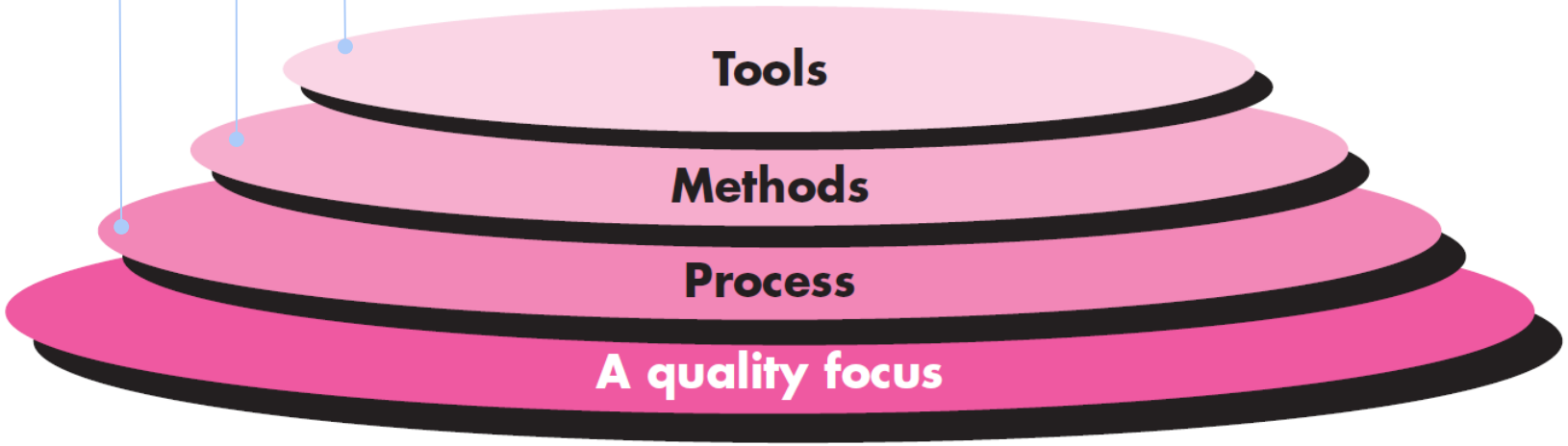
软件工程到底是什么

- 软件工程是一种层次化的技术

过程是工作产品构建时所执行的一系列活动、动作和任务的集合

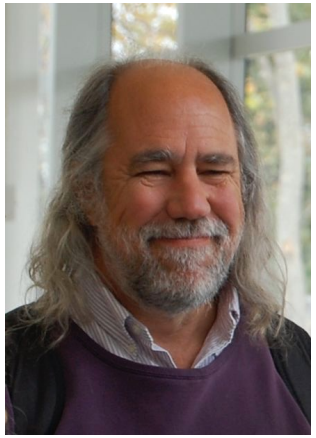
方法为构建软件提供技术上的解决方法（“如何做”）

工具为过程和方法提供自动化或半自动化的支持



复杂系统的5个属性

- 复杂系统的5个共同属性[OOAD07]
  - 层次结构(Hierarchic Structure)
  - 相对本原(Relative Primitives)
  - 分离关注(Separation of Concerns)
  - 共同模式(Common Patterns)
  - 稳定的中间形式(Stable Intermediate Forms)



Grady Booch

理解这个5个属性非常重要，可以更深刻地理解软工发展的原因和机理。

层次结构

“复杂性常常以层次结构的形式存在。复杂的系统有一些相关的子系统组成，这些子系统又有自己的系统，如此下去，直到达到某种最低层次的基本组件。” [Courtois85]

- 复杂系统的架构是它所有组件以及这些组件之间的层次结构的函数

相对本原

“直到达到某种最低层次的基本组件”

- 关于复杂系统中基础组件的实质，经验表明：
  - “选择哪些作为系统的基础组件，相对来说比较随意，这在很大程度上取决于观察者的判断”
  - 对于一个观察者来说很基础的东西，对另一个可能具有很高的抽象层次！

分离关注

“组件内的联系通常比组件见的联系更强，这一事实实际上将组件中的高频率动作（涉及组件的内部结构）和低频率动作（涉及组件见的相互作用）分离开来。” [Ibid]

- 组件内部作用和组件见作用的差异，让我们在系统的不同部分之间实现“分离关注”
- 让我们能够以相对隔离的方式来研究每个部分。

共同模式

“层次结构通常只是由少数不同类型的子系统按照不同的组合和安排方式构成的。”

- 换而言之，复杂系统具有共同的模式
- 这些模式可能涉及小组件的复用，如细胞、或者大一些的结构如脉管系统，在植物和动物中都存在

稳定的中间形式

“复杂系统毫无例外都是从能工作的简单系统演变而来的……从头设计的复杂系统根本不能工作，也不能通过打补丁的方式使其工作。必须从头开始，从能工作的简单系统开始。” [Gall86]

- 随系统的演化，曾经认为是复杂的对象就变成了基础对象，在这些对象的基础上构建更复杂的系统。
- 可以将演化的过程看做是从一个“相对稳定的中间形式”进化到另一个“相对稳定的中间形式”

最主要的手段“分而治之”

- “控制复杂性的技巧我们从远古时代就知道了，即**分而治之**”

- 两种主流的分解方式
  - 算法分解和面向对象分解

- 思考：对复杂系统的分解，按算法分解和按面向对象分解，哪一种更好？
  - 这个问题带有欺骗性!
  - 本课程将会从两个不同思路去介绍软件工程的实践

从混沌到有序

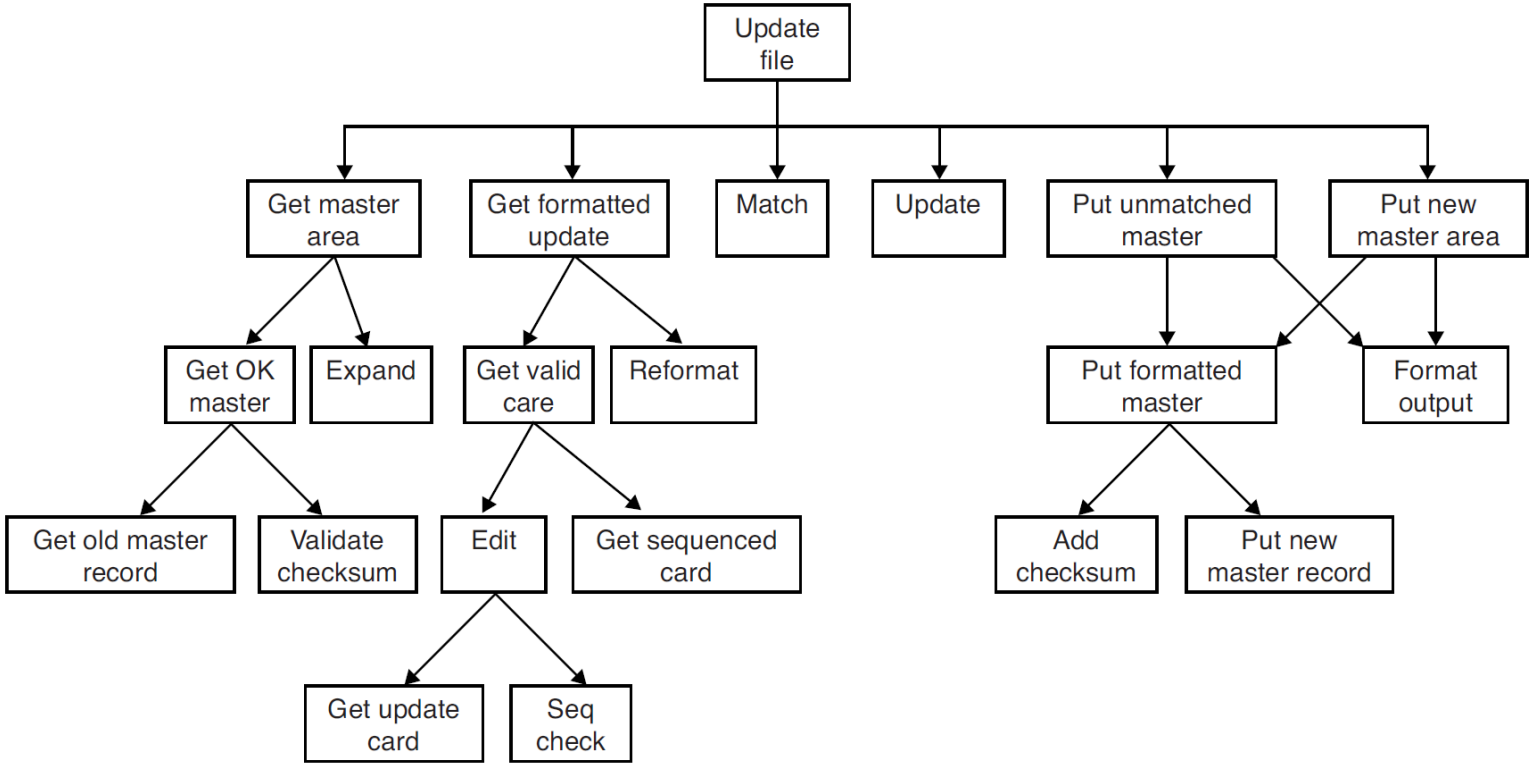
- 不要指望“英雄”帮我们解决所有问题
- 必须考虑通过一些行之有效的方式来控制复杂性
  - 分解的手段（分而治之）
    - *算法分解 vs 面向对象分解*
  - 抽象的手段
    - *如果不能全面掌握，就选择忽略非本质的细节，转而处理一般化、理想化的模型*
  - 层次结构的手段
    - 组合关系、继承关系

算法分解

- 将问题的解决看作是由一系列步骤通过**控制流程**形成的过程
  - 所分解得到的系统中的每个模块，代表了整个过程中的一个步骤

- 由此而形成了**结构化的分析与设计**
  - 我们也常称之为传统分析与设计，对于的软件工程为传统软件工程
  - 与之对应的就是面向对象分析与设计，以及面向对象软件工程

算法分解示例



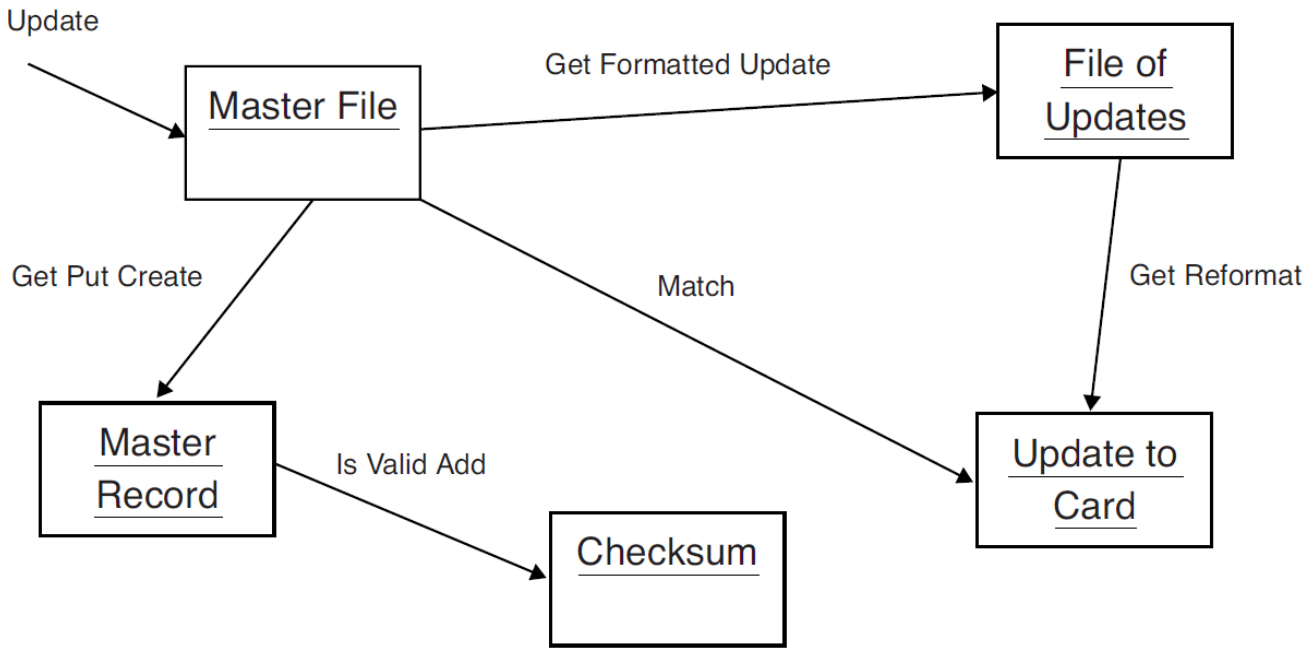
思考：如果让你实现这个功能，你准备用什么语言，如何实现？

面向对象的分解

- 对于同样的问题，还存在着另一种可能的分解方式
  - 根据问题领域中的关键**抽象概念**对系统进行分解
  - 我们把世界看成是一组自动化的代理，他们相互协作，执行某种高级的行为
  - 面向对象的分解，关键在于深刻把握“**对象模型**”的概念

- 这是两种**相当不同**的处理方式，但他们可以处理**相同的问题**！

面向对象分解示例



思考：如果让你实现这个功能，你准备用什么语言，如何实现？

从语言的演化看软工 (极简)

- 从FORTRAN开始，高级程序设计语言的发展基本经历了多次演化过程

- 在这一系列的语言换代中，每种语言所支持的抽象机制发生了变化

第一代语言

- First-generation languages (1954–1958)
  - FORTRAN I　　数学表达式
  - ALGOL 58　　数学表达式
  - Flowmatic　　数学表达式
  - IPL V　　数学表达式

- 第一代语言主要应用于科学和工程应用，这个问题领域的词汇几乎全是数学
- 典型代表FORTRAN I，让程序员可以写成数学公式，从而不用面对汇编语言或机器语言中的一些复杂问题

第二代语言

- Second-generation languages (1959–1961)
  - FORTRAN II　　子程序、单独编译
  - ALGOL 60　　块结构、数据类型
  - COBOL　　数据描述、文件处理
  - Lisp　　列表处理、指针、垃圾收集（注：J. McCarthy设计并实现了第一个以lambda演算为模型的语言）

- 第二代语言中，重点是算法抽象
- 关注的焦点主要是告诉计算机做什么：先读入一些记录，然后进行排序，最后打印报表。
- 向着问题空间有靠近一步，离底层计算机又远了一步

第三代语言

- Third-generation languages (1962–1970)
  - PL/1　　FORTRAN + ALGOL + COBOL
  - ALGOL 68　　ALGOL 60的严格继承者
  - Pascal　　ALGOL 60的简单继承者
  - Simula　　类、数据抽象

- 演进到了支持数据抽象
- 这是，程序员可以描述相关数据的意义（它们的类型），并让程序设计语言强制确保这些设计决策

断代

- The generation gap (1970–1980)

  人们发明了许多不同语言，但很少存活下来。但是，下面的语言值得一提：

  - C　　Efficient; small executables
  - FORTRAN 77　　ANSI standardization

- 以C语言为代表，从语言机制上并没有创新，但面向实际应用需求，更具有实用性
  - 值得一提的是，C语言是和UNIX系统共同发展的

面向对象兴盛

- Object-orientation boom (1980–1990, but few languages survive)
  - Smalltalk 80　纯面向对象语言
  - C++　从C和Simula发展而来
  - Ada83　强类型，收到Pascal的很大影响
  - Eiffel　从Ada和Simula发展而来

框架的出现

- Emergence of frameworks (1990–2000)

  出现了许多语言活动、新版本和标准化工作，导致了程序设计框架的出现.

  - Visual Basic　简化了Windows应用的图形用户界面(GUI)开发
  - Java　Oak的后续版本，其设计意图是实现可移植性
  - Python　Object-oriented scripting language
  - J2EE　基于Java的企业计算框架
  - .NET　Microsoft’s object-based framework
  - Visual C#　.NET架构下Java的竞争者
  - Visual Basic　针对微软.NET框架的Visual Basic

- 中间件（middle-ware）的概念逐渐登上历史舞台，并成为一个重要的竞争领域

新的发展趋势

- 动态脚本类语言随着Web应用的发展受到越来越多的关注
  - Javascript、Python
  - 典型的有松本行弘的Ruby语言，随着Rails的推广而受到关注（Martin Fowler给予非常高的评价）

- 函数式编程随着并行技术的发展得到更多关注
  - Scala语言、Google Go语言
  - Java 8的lambda表达式

软件工程的基本内容

- 软件设计方法论
- 软件过程
- 软件工具
- 软件工程标准和规范
- 软件工程管理
- 软件工程理论

软件工程的基本原理

- 严格按照计划进行管理
- 坚持进行阶段评审
- 实行严格的产品控制
- 采用现代的程序技术
- 结果要能清晰地审计
- 开发小组人员素质要好，数量不宜多
- 要承认不断改善软件工程实践的必要性

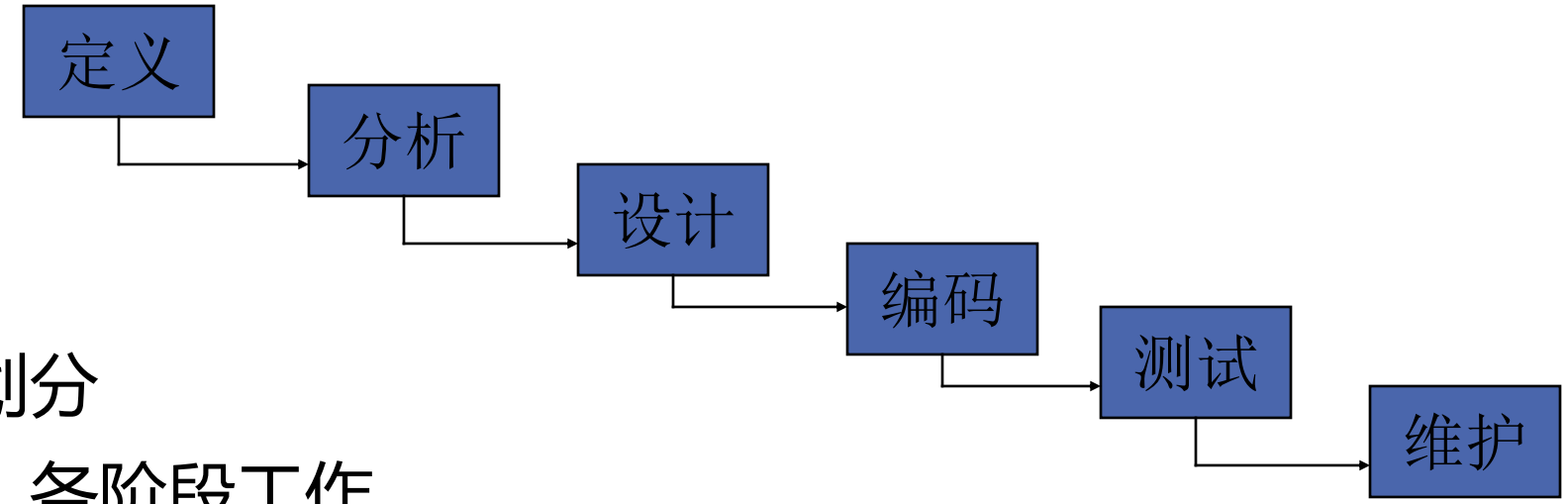
软件生存期模型

软件生存期（过程）模型：

软件生存期是软件产品或系统一系列相关活动的全周期。从形成概念开始，经过研制，交付使用，在使用中不断增补修订，直到最后被淘汰，让位于新的软件产品的过程。对软件生存期的不同划分，形成了不同的软件生存期模型。

瀑布式软件生存期模型

瀑布式软件生存期模型



强调阶段的划分
及其顺序性、各阶段工作
及其文档的完备性，是一种严格线性的、
按阶段顺序的、逐步细化的开发模式。

软件质量要素

软件质量要素：

- 正确性：软件产品准确执行软件规格说明中所规定的能力。
- 健壮性：在异常条件下软件仍能运行的能力。
- 可靠性：软件在给定的时间内和规定的环境条件下，按规格说明的规定成功地运行的概率。可靠性理解为正确性和健壮性之和。

面向对象vs传统方法

- 很多主流文献都提到面向对象的方法具有更明显的优势 [Schach11]
  - 在软工没有得到广泛实践时，采用的普遍是传统范型，那时只是“编写”软件。随着软件产品的规模增长，结构化技术的不足日益显露出来，而面向对象范型成为更好的替代物。
  - 面向对象范型是目前可用的最好方法。

传统软件工程

- 传统范型（或结构化范型）
  - 在1975年以前，大多数软件组织没有使用专门的开发技术，每个人都以自己的方式工作
  - 75-85年，所谓传统或结构化范型的发展使软工在这个方面有了突破性进展
- 传统范型的技术
  - 结构化系统分析
  - 数据流分析
  - 结构化编程
  - 结构化测试