SOFTWARE TESTING

Chapter 1

Overview of Software Engineering

School of Data & Computer Science
Sun Yat-sen University

Approaches & Technologies

OUTLINE

- 1.1 软件与软件危机
- 1.2 软件开发与软件工程
- 1.3 软件生命周期模型
- 1.4 软件质量标准
- 1.5 敏捷开发
- 1.6 软件生命周期过程

- 计算机硬件体系结构
  - 冯·诺依曼结构
  - 基本属性
    - 机内数据表示
    - 寻址方式
    - 寄存器组织
    - 指令系统
    - 存储系统
    - 中断机构
    - 输入输出结构
    - 信息保护

计算机软件体系结构

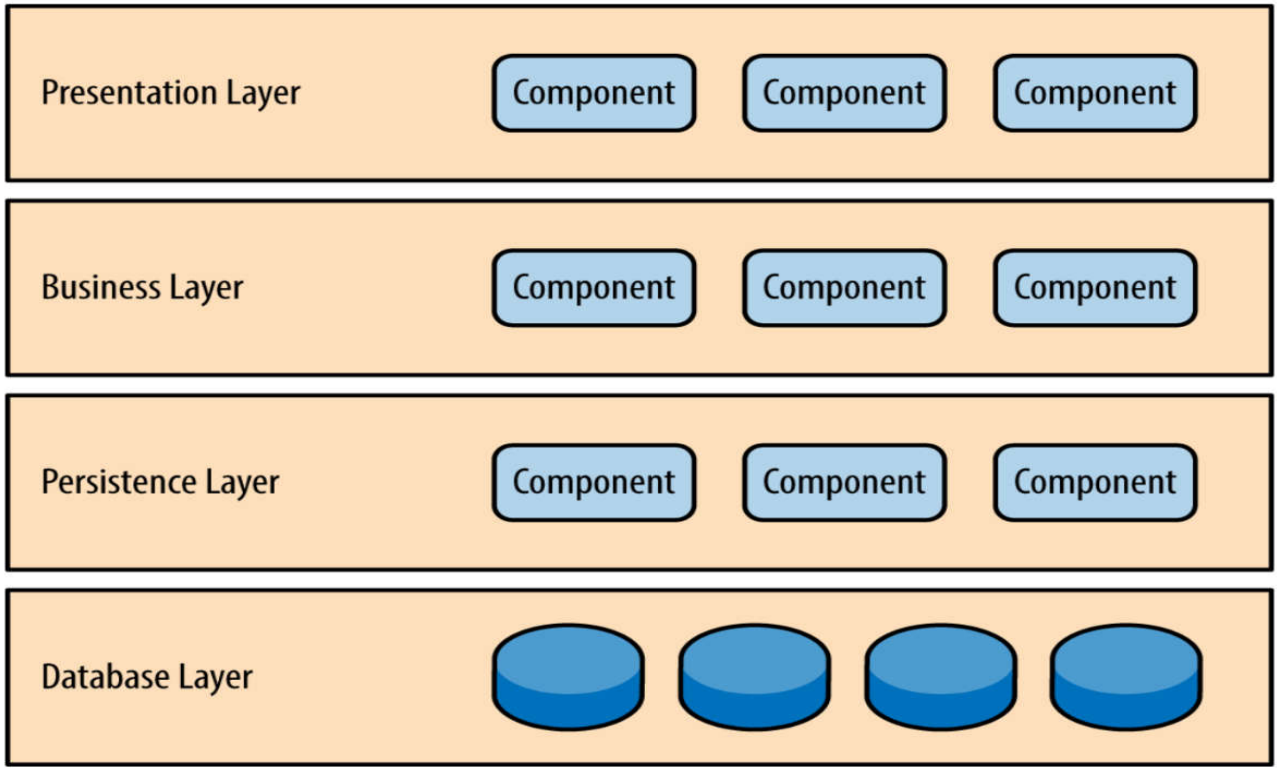
- 软件体系结构是计算机软件系统 (下简称系统) 的结构、行为和属性的高级抽象，描述构成系统的元素、元素的相互作用、元素集成的模式以及有关这些模式的约束。软件体系结构给出系统的组织结构和拓扑结构，规定系统需求和构成系统的元素之间的对应关系，并提供设计决策的基本原理。
- Some Prevalent Software Architectures
  - Layered Architecture
  - Event-Driven Architecture
  - Microkernal Architecture
  - Microservices Architecture Pattern
  - Space-Based Architecture

计算机软件体系结构

- Some Prevalent Software Architectures
  - Layered Architecture
    - The layered architecture pattern, otherwise known as the n-tier architecture pattern, is the most common architecture pattern. It closely matches the traditional IT communication and organizational structures found in most companies, making it a natural choice for most business application development efforts.
    - Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application. Most layered architectures consist of four standard layers: presentation, business, persistence, and database. In some cases, the business layer and persistence layer are combined into a single business layer, particularly when the persistence logic is embedded within the business layer components. Thus, smaller applications may have only three layers.

计算机软件体系结构

- Some Prevalent Software Architectures
  - Layered Architecture
    - Layered architecture pattern with four layers.



计算机软件体系结构

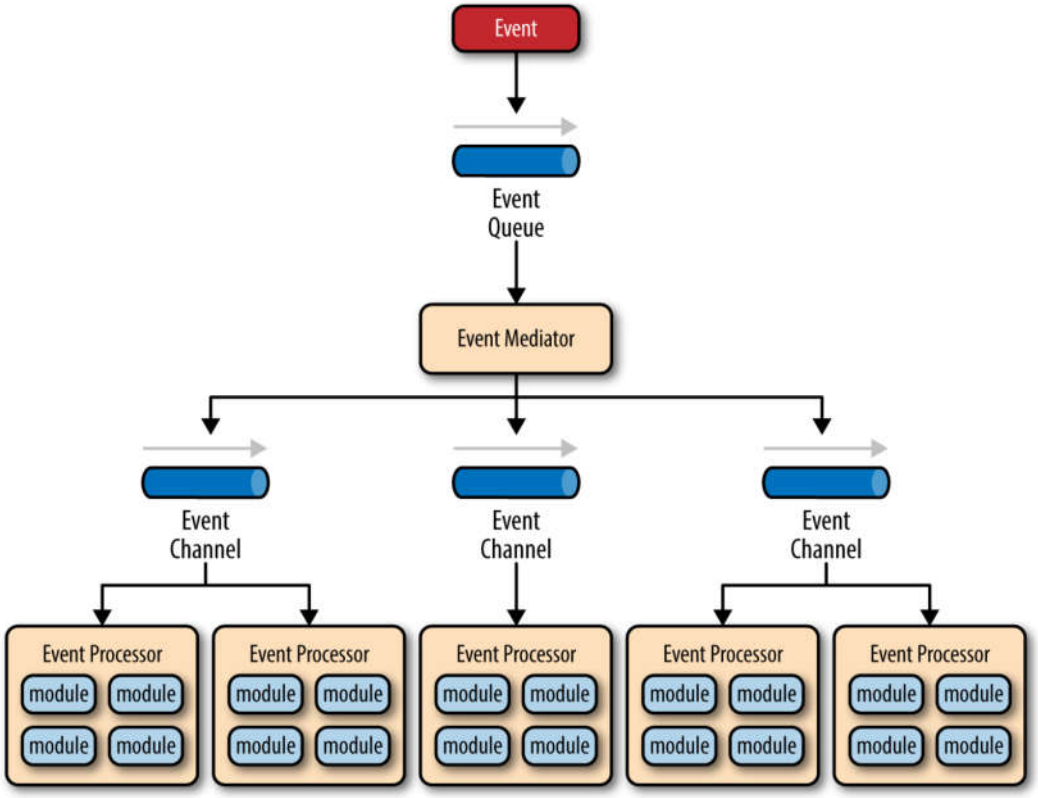
- Some Prevalent Software Architectures
  - Layered Architecture
    - One of the powerful features of the layered architecture pattern is the *separation of concerns* among components. Components within a specific layer deal only with logic that pertains to that layer.
      - For example, components in the presentation layer deal only with presentation logic, whereas components residing in the business layer deal only with business logic.
    - This type of component classification makes it easy to build effective roles and responsibility models into your architecture, and also makes it easy to develop, test, govern, and maintain applications using this architecture pattern due to well-defined component interfaces and limited component scope.
    - The most important is the layers of isolation concept.

计算机软件体系结构

- Some Prevalent Software Architectures
  - Event-Driven Architecture
    - The event-driven architecture pattern is a popular distributed asynchronous architecture pattern used to produce highly scalable applications. It can be used for small applications and as well as large, complex ones. The event-driven architecture is made up of highly decoupled, single-purpose event processing components that synchronously receive and process events.
    - The event-driven architecture pattern consists of two main topologies, the mediator and the broker. The mediator topology is commonly used when you need to orchestrate multiple steps within an event through a central mediator, whereas the broker topology is used when you want to chain events together without the use of a central mediator.

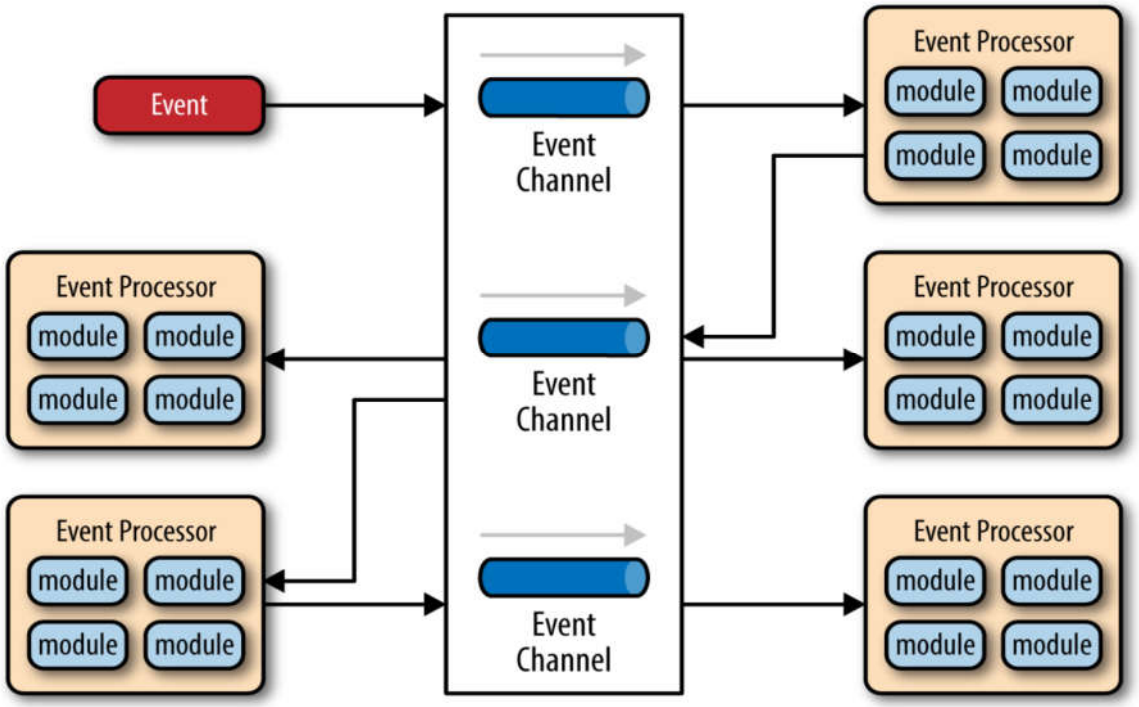
计算机软件体系结构

- Some Prevalent Software Architectures
  - Event-Driven Architecture
    - The event-driven architecture mediator topology



计算机软件体系结构

- Some Prevalent Software Architectures
  - Event-Driven Architecture
    - The event-driven architecture broker topology

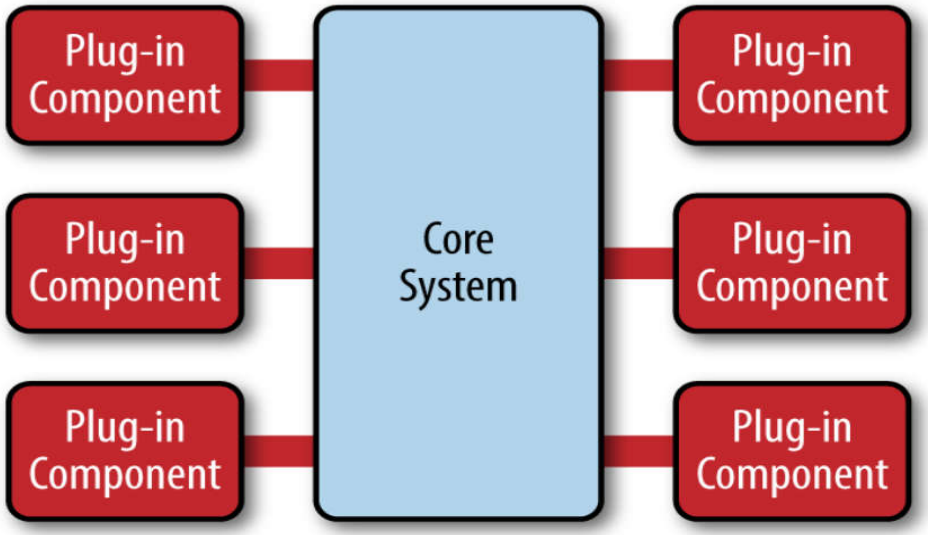


计算机软件体系结构

- Some Prevalent Software Architectures
  - Microkernal Architecture
    - The microkernel architecture pattern (sometimes referred to as the plug-in architecture pattern) is a natural pattern for implementing product-based applications. A product-based application is one that is packaged and made available for download in versions as a typical third-party product. The microkernel architecture pattern allows you to add additional application features as plug-ins to the core application, providing extensibility as well as feature separation and isolation.
    - The microkernel architecture pattern consists of two types of architecture components: a *core system* and *plug-in modules*. Application logic is divided between independent plug-in modules and the basic core system, providing extensibility, flexibility, and isolation of application features and custom processing logic.

计算机软件体系结构

- Some Prevalent Software Architectures
  - Microkernal Architecture



计算机软件体系结构

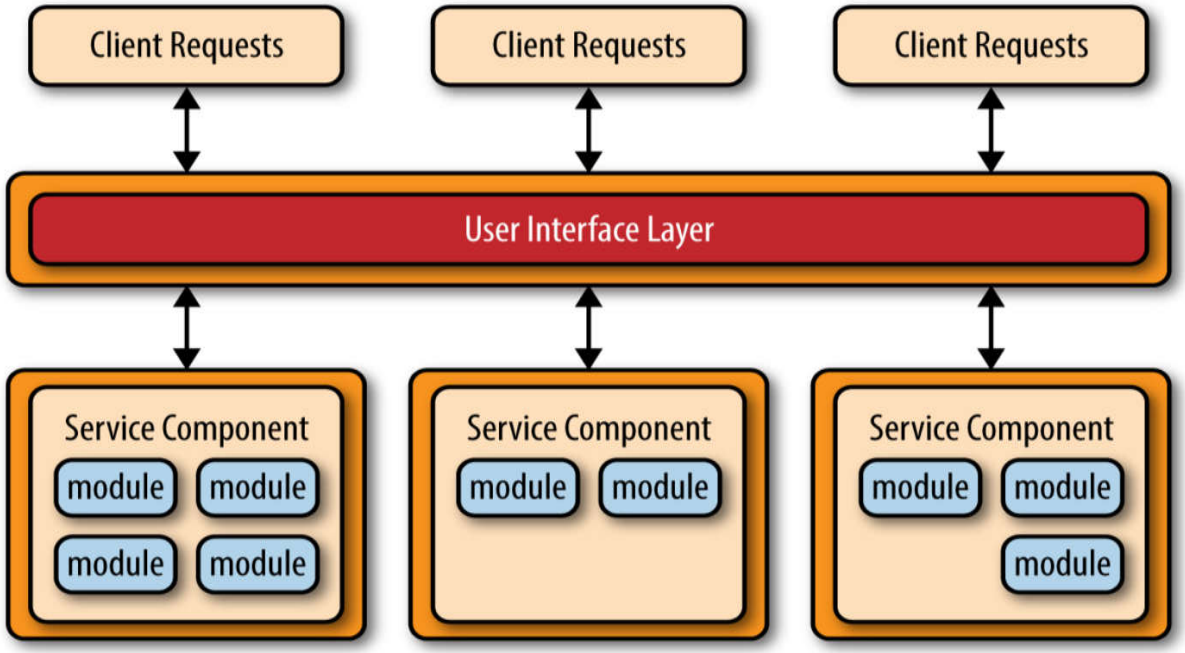
- Some Prevalent Software Architectures
  - Microservices Architecture Pattern
    - The microservices architecture pattern is quickly gaining ground in the industry as a viable alternative to monolithic applications and service-oriented architectures. The microservices architecture style naturally evolved from two main sources: monolithic applications developed using the layered architecture pattern and distributed applications developed through the service-oriented architecture pattern.
    - Core concepts of the microservices architecture pattern.
      - *Separately Deployed Units*. Each component of the microservices architecture is deployed as a separate unit, allowing for easier deployment through an effective and streamlined delivery pipeline, increased scalability, and a high degree of application and component decoupling within your application.

计算机软件体系结构

- Some Prevalent Software Architectures
  - Microservices Architecture Pattern
    - Core concepts of the microservices architecture pattern.
      - *Service Components*. Service components, varying in granularity, contain one or more modules that represent either a single-purpose function or an independent portion of a large business application. Designing the right level of service component granularity is one of the biggest challenges within a microservices architecture.
      - *Distributed Architecture*. All the components within the architecture are fully decoupled from one other and accessed through some sort of remote access protocol. The distributed nature of this architecture pattern is how it achieves some of its superior scalability and deployment characteristics.

计算机软件体系结构

- Some Prevalent Software Architectures
  - Microservices Architecture Pattern
    - Basic microservices architecture pattern

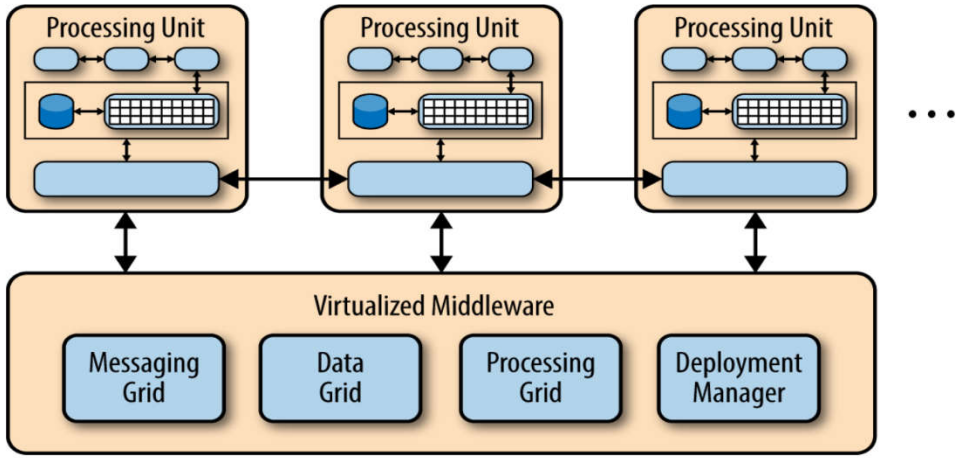


计算机软件体系结构

- Some Prevalent Software Architectures
  - Space-Based Architecture
    - The space-based architecture pattern is specifically designed to address and solve scalability and concurrency issues. It is also a useful architecture pattern for applications that have variable and unpredictable concurrent user volumes. Solving the extreme and variable scalability issue *architecturally* is often a better approach than trying to scale out a database or retrofit caching technologies into a non-scalable architecture.
    - The space-based pattern (also sometimes referred to as the cloud architecture pattern) minimizes the factors that limit application scaling. This pattern gets its name from the concept of *tuple space,* the idea of distributed shared memory. High scalability is achieved by removing the central database constraint and using replicated in-memory data grids instead.

计算机软件体系结构

- Some Prevalent Software Architectures
  - Space-Based Architecture
    - Application data is kept in memory and replicated among all the active processing units. Processing units can be dynamically started up and shut down as user load increases and decreases, thereby addressing variable scalability. Because there is no central database, the database bottleneck is removed, providing near-infinite scalability within the application.



- 计算机软件体系结构
  - 描述软件体系结构的模型
    - 结构模型 Structural Model
      - 以体系结构的构件、连接件和其他概念来刻画软件系统结构，力图通过结构反映系统的重要语义内容，包括系统的配置、约束、隐含的假设条件、风格、性质等，是一类最为直观、普遍的建模方法。
    - 框架模型 Frame Model
      - 侧重于描述软件系统的整体结构而非结构内容的细节。
    - 动态模型 Dynamic Model
      - 描述软件系统的“大粒度”的行为性质，例如系统的重新配置或演化。
      - 动态可以指软件系统总体结构的配置、建立或拆除通信通道、计算过程等。

- 计算机软件体系结构
  - 描述软件体系结构的模型 (续)
    - 过程模型 Process Model
      - 过程模型研究构造软件系统的步骤和过程，因而系统结构是遵循某些过程脚本的结果。
    - 功能模型 Functional Model
      - 软件体系结构由一组功能构件按层次组成，由下层向上层提供服务。功能模型可以看作是一种特殊的框架模型。

软件

- 软件 = 程序 + 数据 + 文档 (+ 服务)
  - 软件是能够完成预定功能和性能的、可执行的计算机指令；
  - 软件是使得程序能够适当地处理信息的数据结构；
  - 软件是描述程序的操作和使用的文档。
  - *Algorithms + Data Structures = Programs*, Nicklaus Wirth 1976.

- 软件
  - 软件的逻辑抽象性
    - 软件是逻辑实体 (抽象性)，不是物理实体。
      - 软件是逻辑的、知识性的产品集合，是对物理世界的一种抽象，或者是某种物理形态的虚拟化。
        - 软件工程与社会工程存在复杂的交叉。
        - 软件的复杂程度持续上升。
      - 软件是智力成果，开发成本昂贵，可以复制 (具备低边际成本特性)。
      - 软件的“问题”都是在软件开发和修改过程中引入的；软件的开发至今尚未完全摆脱手工作坊式的低效率的开发方式。

- 软件
  - 软件的分类
    - 系统软件
      - 操作系统、数据库管理系统、设备驱动程序、通信处理程序等。
    - 应用软件
      - 事务软件、实时软件、工程和科学软件、嵌入式软件、娱乐软件、个人计算机软件、人工智能软件等。
    - 工具软件
      - 文本编辑软件、文件格式化软件、数据传输软件、程序库系统以及支持需求分析、设计、实现、测试和支持管理的软件等。
    - *可重用软件

- 软件产品的组成
  - 开发周期的软件产品组成
    - 客户需求文档 - Customer Requirements
    - 市场需求文档 - MRD (Marketing Requirement Documents)
    - 软件规格说明书 – Specifications
    - 技术设计文档 – Technical Design Documents
    - 测试文档 Test Documents
    - 在线帮助 - Online Help
    - 产品发布注释 - Release Notes / Read Me
    - 产品软件包 - Release Packages

- 软件产品的组成
  - 销售周期的软件产品组成
    - 帮助文件 Help Files
    - 示例 Samples and Examples to Illustrate Points
    - 产品支持文档 Product Support Information
    - 错误信息 Error Messages
    - 安装手册 Setup and Installation Instructions
    - 用户手册 User Manual (s)
    - 产品标签 Label and Stickers
    - 产品广告或宣传材料 Ads and Marketing Material ...

- Software Crisis 软件危机
  - 早期软件开发的特点
    - 软件规模相对较小
    - 程序设计是一门技艺
    - 缺少软件开发工具的支持
    - 缺乏软件开发管理的理论与方法
    - 缺乏有效的软件开发后的维护

Software Crisis

- The causes of the software crisis were linked to the overall complexity of hardware and the software development process. The main cause is that improvements in computing power had outpaced the ability of programmers to effectively utilize those capabilities.
- The crisis manifested itself in several ways: (软件危机的表现形式)
  - Projects running over-budget
  - Projects running over-time
  - Software was very inefficient
  - Software was of low quality
  - Software often did not meet requirements
  - Projects were unmanageable and code difficult to maintain
  - Software was never delivered

- 软件危机的成因
  - 软件危机的根源
    - 软件的大量需求与软件生产力效率之间的矛盾
    - 软件系统的复杂性与软件开发方法之间的矛盾
  - 软件本身的特点
    - 软件是一种抽象逻辑
    - 软件是开发人员的智力劳动成果
    - 软件具备强烈的个性化特征
    - 软件规模日趋庞大，实现的业务逻辑与流程复杂

- 软件危机的成因
  - 软件开发的客观因素
    - 系统需求分析不足
    - 开发周期管理不善
    - 开发过程缺乏规范
      - 软件开发 =？程序编写
    - 质量控制标准规程滞后
    - 软件维护计划被忽视
  - 软件开发的产业因素
    - 软件企业的作坊式管理
    - 软件企业规模的急剧膨胀

- 软件危机的解决途径
  - 正确认识计算机软件的内涵。
  - 采用工程项目管理方法实施软件开发的组织管理。
    - 软件开发应该是一种组织良好、管理严密、协同配合的工程活动。
  - 采用成熟的软件开发技术和方法，开发和使用适当的软件工具。

软件开发的基本过程

- 计划
- 需求分析
  - 根据客户的要求，清楚了解客户需求中的产品功能、特性、性能、界面和具体规格等，分析确定软件产品所能达到的目标。
- 设计
  - 根据需求分析的结果，考虑如何在逻辑上实现所定义的产品功能、特性。设计可分为概要设计和详细设计两个实施阶段，也可分为数据结构设计、软件体系结构设计、应用接口设计、模块设计、界面设计等部分。

软件开发的基本过程

- 实现
  - 编写程序将设计结果转换成计算机可读的形式并获得运行。
- 测试
  - 确认用户需求、对设计和实现结果进行验证的过程。
- 维护
  - 维持软件运行，修改软件缺陷、增强已有功能、增加新功能、升级等。

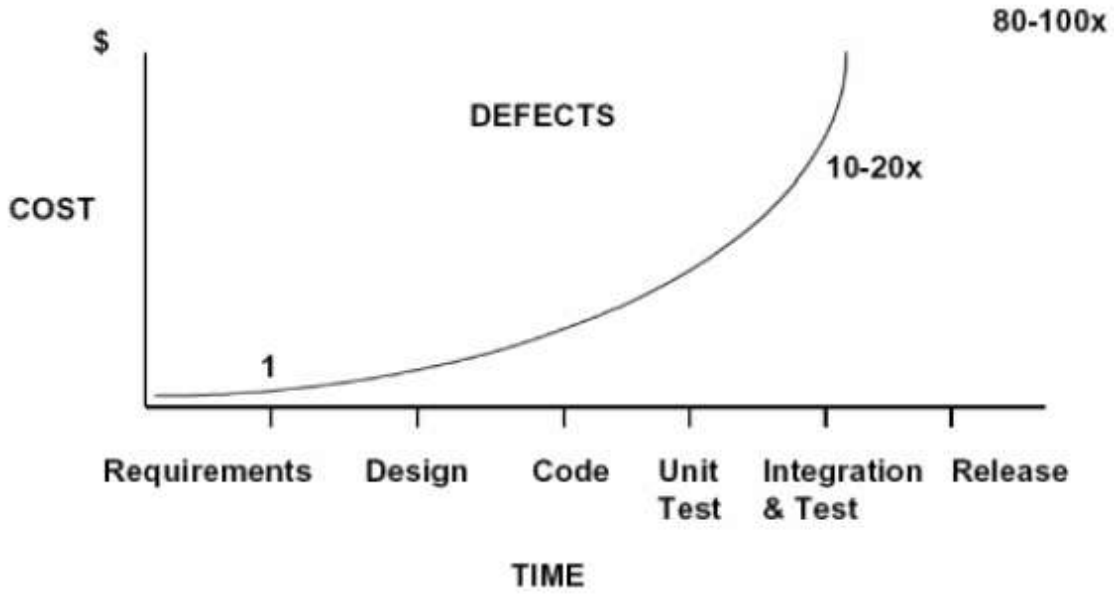
软件开发成本

软件开发成本分布

软件类型	开发成本的阶段分布 %		
	需求与设计	实现	测试
控制软件	46	20	34
航空航天软件	34	20	46
操作系统	33	17	50
科技计算软件	44	26	30
商业应用软件	44	28	28

软件开发成本

- 软件的修改成本随时间变化的趋势



软件工程概念的提出

- 为了克服软件危机，北约组织成员国的软件工作者于1968年和1969年两次召开会议 (NATO 学术会议)，提出了“软件工程”概念，著名的瀑布模型也同期出现。
  - 采用工程的概念、原理、技术和方法来开发与维护软件。
  - 把经过时间考验的管理技术和当前能够得到的最好的技术方法结合起来。
- 软件工程进展
  - 软件工程理论与实践水平不断发展。
  - 软件工程学的重要分支包括软件开发方法学、软件需求工程、软件过程控制与改进、软件质量工程、软件可靠性工程、软件能力成熟度 (CMM)、软件测试与认证以及软件可信计算等。

软件工程要点

(1) 软件开发是一个有计划、分阶段、严格按照标准或规范进行的工程活动。

- 软件工程是指导计算机软件开发和维护的工程学科。
- 软件工程方法是结合了工程、管理和技术的方法。

(2) 软件工程将系统的、规范的、可度量的方法应用于软件的开发、运行和维护的过程。

- 将工程化应用于软件开发过程中，对上述方法进行研究

(3) 采用适当的软件开发方法和支持环境以及编程语言来表示和支持软件开发各阶段的各种活动，并使开发过程条令化、规范化，使软件产品标准化，开发人员专业化。

(4) 用工程学的观点进行费用估算，制定进度和计划；用管理科学中的方法和原理进行软件生产的管理；用数学的方法建立软件开发中的各种模型和各种算法。

软件工程过程

- 软件工程过程是为了获得最终满足需求且达到工程目标的软件产品，在软件工具的支持下由软件工程师完成的一系列软件工程活动。
- 生产一个软件产品所需要的基本步骤：

(1) 软件规格说明 Specification

- 问题分析：
  - 需求获取和定义
- 需求分析：
  - 生成软件需求规格说明文档
  - 规定软件的功能及其使用限制

软件工程过程

- 生产一个软件产品所需要的基本步骤：(续)

(2) 设计与实现 (开发) Development

- 概要设计：建立整个软件的体系结构，包括子系统、模块以及相关层次的说明、每一模块的接口定义等。
- 详细设计：产生程序员可用的模块说明，包括每一模块中数据结构说明及加工描述 。
- 把设计结果转换为可执行代码，产生满足规格说明的软件。

(3) 确认 Validation

- 通过对产品的有效性验证，确保软件满足用户的要求。

(4) 演进 Evolution

- 为了满足客户的变更要求，软件必须在使用过程中进行不断地改进和完善。

- 软件工程原则
  - 采取适宜的开发模型
    - 比如能够适应易变的用户需求
  - 采用合适的设计方法
    - 需要软件模块化、抽象与信息隐藏、局部化、一致性以及适应性等，需要合适的设计方法的支持。
  - 提供高质量的工程支持
    - 软件工程过程需要软件工具和环境的支持
  - 重视开发过程的管理
    - 资源的有效利用、产品的目标管理、软件生产率的提高等

软件工程目标

- 软件工程的目标是生产具有正确性、可用性以及开销适宜、进度保证、并且项目成功的软件产品。软件工程目标决定了软件工程过程、过程模型 (软件周期模型) 和工程方法的选择。
  - 正确性
    - 软件产品达到预期功能及性能的程度。
  - 可用性
    - 软件基本结构、实现及文档为用户可用的程度。
  - 经济性
    - 软件开发、运行的整体耗费 (包括资金耗费和时间耗费) 满足用户要求的程度。
  - 软件项目成功标识
    - 开发成本低、功能与性能满足需求、易于移植、维护方便、按时完成开发任务并及时交付软件产品。

- 软件工程学
  - 软件开发技术研究
    - 软件开发技术是完成软件生命周期各阶段的任务所必须具备的技术手段，包括：
      - 软件开发方法学
      - 软件开发工具
      - 软件工程环境
  - 软件工程管理研究
    - 软件工程管理学
    - 软件经济学

软件开发方法

- 软件开发方法研究如何在规定的投资和时间内，开发出符合用户需求的高质量的软件产品。
- 软件开发方法是一种使用预先定义的技术集及符号来组织软件生产过程的方法。
  - 包括：项目计划与估算、软件系统需求分析、数据结构、系统总体结构的设计、算法的设计、编码、测试以及维护等。
- 软件开发方法主要有：
  - 面向数据流的结构化程序开发方法
  - 面向数据结构的开发方法 (*Jackson* 方法)
  - 基于模型的方法
  - 面向对象的开发方法

- 软件开发方法
  - 面向数据流的结构化程序开发方法 (始终关注程序结构)
    - 指导思想
      - 自顶向下，逐步求精
    - 基本原则
      - 功能的分解与抽象
    - 适用性
      - 适合数据处理领域的问题

软件开发方法

- 面向数据结构的开发方法 (*Michael A. Jackson*, 1975)
  - JSP (*Jackson* Structured Programming)
    - 描述问题的输入、输出数据结构，分析其对应性。
    - 设计相应的程序结构，从而给出问题的软件过程描述。
    - 以数据结构为驱动。
  - JSD (*Jackson* System Development)
    - 标志实体与行为
    - 生成实体结构图
    - 创建软件系统模型
    - 扩充功能性过程
    - 施加时间约束
    - 系统实现

- 软件开发方法
  - 基于模型的方法 (支持程序开发的形式化方法)
    - 维也纳方法 (VDM, IBM Laboratory Vienna, 1972)
      - 将软件系统当作模型来给予描述，把软件的输入、输出看作模型对象，把这些对象在计算机内的状态看作该模型在对象上的操作。
  - 面向对象的开发方法 (OOD)
    - 指导思想
      - 尽可能按照人类认识世界的方法和思维方式来分析和解决问题。
    - 面向对象方法包括
      - 面向对象分析
      - 面向对象设计
      - 面向对象实现

软件开发工具

- 软件开发工具是为了支持软件人员开发和维护目标系统而使用的软件工具。
  - 软件开发工具为软件开发方法提供自动的或半自动的软件支撑环境，是完成软件开发任务的重要辅助手段。
  - 例：微软的 Visual Studio Team Suite 是基于 Microsoft Solution Framework 的支持工具套件，通过 Team Foundation Server 对项目研发过程中的项目管理、版本管理、工作项跟踪、报告、软件构件等内容进行管理，并提供相应的测试工具，是一类软件开发生命周期或应用生命周期管理工具 (SDLC/Application Life Cycle Management)。
    - http://msdn.Microsoft.com/zh-cn/teamsystem/default.aspx
  - 例：IBM/JAZZ、HP/BTO、MicroFocus/Open ALM 等平台。

- 软件工程环境
  - 软件工程环境是支持软件开发的整个生存周期的资源集合。
    - 用以提高软件开发效率，提高软件质量，降低软件开发成本。
- 计算机辅助软件工程 CASE
  - CASE 是一类为软件开发提供一组优化集成的且能大量节省人力的软件开发工具。
    - 实现软件生存期各环节的自动化并使之成为一个整体。

Thank you!