PPT01-Intro

Outline

What is software?

Why software engineering? (重要性)

What is software engineering?

SE body of knowledge

SE code of ethics

计算机软件是计算机程序和相关数据的集合，提供指令告诉计算机做什么和如何去做。

软件工程的重要性。

软件工程是建立并使用健全的工程原则，来生产开销合宜且在实际机器能有效工作的软件。

软件工程的原则：系统，开销合宜，高质量。

从系统规范的早期阶段到它被使用的维护阶段：

需求：确定、分析并详细说明用户需求。

设计：定义应该如何构建系统以满足需求。

编码：编写程序软件。

测试：核实、验证系统。

维护：在操作过程中维护软件。

软件工程的成本分配：依赖于过程模型和软件类型。

系统的集成和测试成本最高：规范和设计-40%；实现-20%；集成和测试-40%。

系统演变可能比开发成本高出三四倍。

软件需求(Software Requirements) 软件设计(Software Design)

软件构造(Software Construction) 软件测试(Software Testing)

软件维护(Software Maintenance) 软件配置管理(Software Configuration Management)

软件工程管理(Software engineering management) 软件工程过程(Software engineering process)

软件工程工具和方法(Software engineering tools and methods) 软件质量(Software quality)

一旦我写完程序并让它运行，工作就完成了。 VS 60-80%的精力消耗在软件交给用户之后。

得到程序运行才能评估其质量。 VS 最有效的软件质量保证机制即正式技术评审应该从工程早期就开始应用。

一个成功项目要交付的产品仅仅只有能有效工作的程序。 VS 程序只是软件配置的一部分，它还包括了许多元素，例如文档。

软件工程使我们创建大量不必要的文档，会延缓我们的开发速度。 VS 软件工程并不是创建文件，而是保证质量。高质量才能减少返工，从而能更快交货。

一个大致目标描述足以开始写程序，之后再补充详细信息。 VS 一个全面稳定的需求描述往往不可能，但模棱两可的需求描述是灾难。明确需求需要客户和开发人员有效和持续的沟通。

项目需求不断变化，软件是活的，所以可以很容易适应变化。 VS 需求变化会引起设计框架的变化，需要额外的资源和设计修改，且变化对成本的影响随时间增加。

我们有一本关于构建软件的标准和程序的书，它能为用户提供一切他们需要知道的。 VS这本书很可能存在，它是现在的吗？软件实践人员意识到它的存在吗？它反映了现代软件工程实践吗？它完整吗？它适用吗？它流线型地改善交货时间同时保持对质量的关注吗？在许多情况下？所有这些问题的答案是否定的。

如果落后于计划，可以添加更多的程序员并赶上。 VS软件开发不是一个机械过程，加人需要花时间教他们工作从而导致开发时间减少，尤其不能在软件开发后期加人。

如果把软件项目外包给第三方，我可以放松，让公司构建它。 VS一个组织往往不了解软件项目内部的管理和控制。

软件工程涉及更广泛的责任，而不仅仅是技术的应用。如果要成为受人尊敬的专业人士，软件工程师必须表现出诚实和道德责任。工程师的职业责任：

保密：应该尊重雇主或客户的保密无论是否正式签订保密协议。

能力：工不应该歪曲自己能力水平，不应该接受超出自身能力的工作。

知识产权：应该确保雇主或客户的知识产权保护。

PPT02-Project

All great changes are preceded by chaos.

**软件项目管理：**Project Management

确保软件交付：预算、按时、质量

保护伞活动：测量及指标、评估、风险、时间表、跟踪和控制

人是主元。

(招聘、选拔、绩效管理、培训、薪酬、职业发展、组织和工作设计和团队/文化发展)

涉众(利益相关者)：(双赢策略)

高级经理，项目(技术)经理，从业人员，客户，最终用户

团队：分工、信任、荣辱与共。 组长：激励、责任

沟通与协调：

正式客观的方法、正式的人际交往过程、非正式的人际交往过程、电子通讯、人际关系网络

W5HH Plan：why、what、when、who、where、how、how much

Scheduling(项目调度)：定义所有项目任务、明确关键路径和里程碑、明确可交付成果、定位过程以确保延迟可识别且可控制。

**任务分解结构：**Work Breakdown Structure

把任务分解成可的元素：产品、过程、组织

自顶向下细化、分析和组织整体范围、确保一丝不落

责任分配矩阵

**任务网络：**(活动网络，Tasks Network)

任务集Task Set是项目的任务分解结构。没有一个任务集可以适合所有的项目和过程模型。任务集应该提供足够的原则来实现高的软件质量，但不能担负不必要的工作。

任务网络是项目的任务流的图形表示，它描述了任务长度、序列、并发性和依赖性。

关键路径(Critical Path)：任务网络上从开始到结束的一条路径；如果整个项目需要按期完成，其上的任务必须按期完成，否则影响整个项目的进度；它决定了项目的最短工期。

关键路径法(CPM)：指定个人活动->确定活动顺序->画网络图->为每个活动估计完成时间->确定关键路径->随着项目进展更新CPM图。

它提供了项目的图形视图，预测项目完成时间，显示至关重要的活动，确定项目是否按进度。

**关键路径计算**：任务网络里最长的，如果不推迟项目，其上的活动不能推迟。如果要缩短工期，只需减少关键路径耗费的时间。

EST-earliest start time：最早开始时间（其先例活动必须先完成）

EFT-earliest finish time：最早完成时间（EST+活动持续时间）

LFT-latest finish time：最晚完成时间（保证不推迟项目）

LST-latest start time：最晚开始时间（LFT-活动持续时间）

松弛时间Slack Time = LST – EST = LFT – EFT

关键路径上的每个节点：Slack Time = 0，即LST = EST，EFT = LFT。

**时间安排图表：**Timeline Chart

也叫甘特图（Gantt Chart）：任务名称、开始时间、结束时间、工期、前置任务、进度线。

列出完成项目所需步骤，估计每一步所需时间，步骤列在图左侧，时间列在底部，在图中用横线表示每一步骤，开始日期为起点，结束日期为终点，长度显示持续时间。

多步骤发生在相同时间段，意味着任务的并发性。

在步骤必须完成的地方用一条虚线来表示。milestone的持续时间为零。

**挑战：**challenges

影响项目进度的因素：项目规模、潜在用户数量、任务临界、应用长久性、需求稳定性、客户/开发人员沟通方便、适用技术的完备性、性能的限制、(非)嵌入式特性、项目人员。

调度问题：估计开发成本是困难的、生产力并不与任务工作人数成正比、由于通信开销在后期项目加人反而更慢、计划要允许应急以防意料之外的事。

40-20-40的精力分配：

建议整个软件过程的努力分布是40%(分析和设计)，20%(编码)，40%(测试)‏。

项目计划2-3%、需求分析10-25%、软件设计20-25%、编码15-20%、测试和后续调试30-40%

添加程序员赶进度：实际上导致进度下滑：新人必须学习系统，在教学中，教的人没有完成工作，而且沟通渠道和固有延迟会增加。

精力应用于交货时间存在非线性关系：交货时间缩短则精力迅速增加。

推迟交付项目可以显著降低成本：**E = L3/(P3t4)**

E：每人每月的工作量

L：交付的源码的行数

P：生产力参数(2000到12000)‏

t：项目持续时间(日历月)

项目调度的基本原则：Principles

足够的区分度、确定的相关性、时间安排合理、工作分配合理、明确的结果、定义里程碑。

PPT03-Process

**过程：Process**

软件生命周期：需求、规范(分析)、设计、实现、集成、维护、退休。

软件系统的质量受到用于开发和演变它的过程的支配。

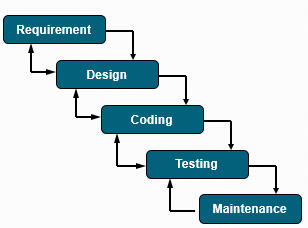
软件过程模型：

传统过程模型（瀑布、演化）；现代过程模型（RUP：Rational Unified Process统一软件过程）。

**瀑布过程模型：**

是一种系统的、有序的软件开发方法：需求分析和定义，系统和软件设计，实现和单元测试，集成和系统测试，操作和维护。

批评：构建软件本质上是一个迭代式的社会学习过程、变化可能引起混乱、很难在项目开始阶段就明确所有需求、程序在项目后期才能有效工作。



**演化过程模型：**

使得软件工程师可以越来越多地迭代开发更完整的软件版本：增量式，原型化，螺旋式。

**增量式开发：**应用线性序列交错的方式，每个线性序列产生软件的可交付“增量”。

**原型化开发：**原型是软件系统的初始版本（演示概念，试验设计选项，找出问题及解决方案）。

演化式原型法（产生初始原型并通过许多步骤不断改善它，直到最后的版本）。

丢弃式原型法（产生实际的原型来帮助发现需求问题，丢弃它，再用其他开发过程开发系统）

**螺旋式开发：**过程被表示成螺旋而不是回溯的活动序列。螺旋的一次循环代表过程的一个阶段。没有固定的规范或设计等阶段，螺旋的循环选择取决于什么是必须的。整个过程中明确地评估并解决风险。目标设置、风险评估、开发和验证、规划。

**统一软件过程：**

UML给出了面向对象方法(OO methodlogies)的标准表示，进行面向对象分析和设计(OOAD)。如果使用UML作为建模语言，则RUP是推荐的软件过程。

RUP工作流：

核心工作流：业务建模，需求捕获，分析和设计，实现，测试，部署。

辅助工作流：配置和变更管理，项目管理，环境。

RUP生命周期阶段：

开始阶段（明确项目的范围，开发业务案例）

细化阶段（规划项目，详细列举特性，基线架构）

建设阶段（构建项目）

过渡阶段（过渡项目给用户）

**重型进程：**Heavy-weight Process

SEI：软件工程研究所

CMM：能力成熟度模型 （在过程改进工作上很有影响力）

软件过程的CMM 5-levels：

level1-initial：没有有效的管理步骤、依赖于个人开发、过程和过程是不可预测的。

Level2-Repeatable：有正式的管理、没有正式的过程模型、依赖个别管理人员。

Level3-Defined：有明确的基于定性的过程改进的过程、有正式的步骤来确保已明确的过程。

Level4-Managed：有明确的过程和定性数据集的正式程序、过程和产品指标被收集。

Level5-Optimizing：致力于持续的过程改进。

问题：该模型只关注项目管理而不是产品开发，它不考虑一个组织的技术使用，不把风险分析和解决作为一项关键的过程技术。没有定义模型的适用域。

**轻型进程：**Light-weight Process

个体和交互重于过程和工具

工作的软件胜于全面的文档

客户协作重于合同谈判

响应变化胜过遵循计划

也就是说，虽然有价值的物品在右边，我们更看重左边。

新方法学：

自适应而不是预测（重方法倾向于花很多时间详细地计划软件过程的很大一部分，所以抵制变化。而轻方法欢迎变化，适应变化，甚至改变自己。）

以人为本而不是面向流程（明确提出遵循人性而非抵制，强调软件开发应该是愉快的活动。）

敏捷的原则：

最高优先级是满足客户，通过早期和持续的有价值的软件的交付。

欢迎需求变更,，即便是在开发后期。

频繁交付可工作的软件，从几个星期到几个月，偏好较短的时间跨度。

业务人员和开发人员必须每天一起工作。

围绕被激励起来的个人构建项目。给他们他们需要的环境和支持,相信他们完成工作。

传达信息的最有效的和有效的方法,在一个开发团队是面对面的交谈。

为被激励的个人构建项目，给他们他们需要的环境和支持，相信他们完成工作。

在一个开发团队，传达信息最有效的方法是面对面的交谈。

可工作的软件是进度的主要衡量。

敏捷的过程提倡可持续发展。

持续关注技术优势和好的设计提高敏捷性。

未完成的工作量最大化是至关重要的。

最好的架构、需求和设计产生于自组织的团队。

每隔一定时间，团队反思如何更有效，然后相应地调整其行为。

PPT04-Analysis-1

outline

Why requirements engineering is critical?

What are the key issues of requirements engineering?

What are the process and activities of requirements engineering?

Requirements elicitation and analysis需求抽取和分析

Requirements specification需求规范

Requirements validation需求验证

Requirements management需求管理

需求工程为设计和结构建立了一个桥梁。

需求：系统必须拥有的功能，或是系统必须使客户满足并接受的约束条件。系统目标是正确的、完整的、一致的和可验证的。

需求工程：软件需求的采集、分析、规范、验证和管理。

功能需求：描述系统应提供的服务、系统应如何应对特定输入和系统在特定情况应如何行为。

非功能需求：约束系统提供的服务或功能，如时间限制，限制开发流程，标准，等等。

域需求：来自系统的应用程序域和反映该域特征的需求。

PPT04-Analysis-2

Outline

Requirements Elicitation需求获取

Uniform Modeling Language (UML) 统一建模语言

UML Use Case Diagram UML用例图

UML：

统一建模语言：图形化的用来列举、可视化、构建和记录软件系统的语言。

UML用例PPT04-2 P25



PPT05-Design-1

Outline

**What is design and why design is important?什么是设计？为什么设计很重要？**

**Software design principles软件设计原则**

Object-oriented design with UML modeling tool面向对象设计(使用UML模型工具)

Design patterns设计模式

Conclusion

Design Principles：

eep It Simple Stupid尽量简单

Modularity模块化(内聚和耦合)

Separation of Concern关注点的分离(横切关注点)

Design for CHANGE 为了改变的设计

Design for REUSE 为了重用的设计

PPT06-Design-2

Outline

What is design and why design is important

Software design principles

**Object-oriented design with UML modeling tool**

**Design patterns**

Object-Oriented Design：

面向对象编程Object-Oriented Programming：互动的、增量的软件开发，按单元设计软件，像构建系统一样开发软件。

OO = Object + Classification + Inheritance + Communication with messages

（对象+分类+继承+信息沟通）

抽象数据类型Abstract Data Type (ADT)：数据及其相关操作的集合，精确地独立于任何特定实现。

Class Analysis：

面向对象设计与分析，标识类。

Example: The ATM System

CRC Card：Class-Responsibility-Collaborator Card代表类的标准索引卡片的集合

UML Package and Class Diagram：

**包图(包关系)、类图(类关系)、接口、对象图**

UML Sequence Diagram：

交互图（序列图）

PPT07-Design-3

Object-Oriented Design

组合胜于继承

Software Design Pattern

封装变化、设计模式(创建型模式、结构模式、行为模式)

About Creation：工厂方法

About Algorithm：模板

About Object：状态模式

About Interaction：观察者模式

What makes a good OO design? Design for Change/ Design for Reuse

PPT08-Design-4

Software Architecture Design软件架构设计

PPT10-Testing

Outline

Software Quality Assurance软件质量保证

Testing Fundamentals测试基础知识

White Box Testing Techniques白盒测试技术

Black Box Testing Techniques黑盒测试技术

Integration and System Testing集成和系统测试

测试基础知识：

检验 & 有效性验证

软件审查：静态检验

软件测试：动态测试

测试文氏图

测试覆盖、测试有效性、测试用例、测试充分性

黑盒测试（基于规格说明）、白盒测试（基于代码）