第 3 章 – 软件过程

第1讲

涵盖的主题

- 3.1 软件过程模型
- 3.2 过程活动
- 3.3 应对变化
- 3.4 Rational统一流程
    - 现代软件过程的一个例子。

软件过程

- 开发软件系统所需的一组结构化活动。
- 许多不同的软件过程，但都涉及：
    - 规范——定义系统应该做什么；
    - 设计和实施——定义系统的组织和实施系统；
    - 验证——检查它是否满足客户的需求；
    - 进化——改变系统以响应不断变化的客户需求。
- 软件过程模型是过程的抽象表示。它从某个特定的角度描述了一个过程。

软件过程描述

- 当我们描述和讨论流程时，我们通常会谈论这些流程中的活动，例如指定数据模型、设计用户界面等，以及这些活动的顺序。
- 过程描述还可能包括：
    - 产品，是过程活动的结果；
    - 角色，反映参与过程的人员的职责；
    - 前置条件和后置条件，它们是在流程活动实施或产品生产之前和之后为真的陈述。

计划驱动的敏捷流程

- 计划驱动的流程是所有流程活动都预先计划并根据该计划衡量进度的流程。
- 在敏捷流程中，计划是渐进式的，更容易更改流程以反映不断变化的客户需求。
- 在实践中，大多数实际流程都包含计划驱动和敏捷方法的元素。
- 没有正确或错误的软件过程。

3.1 软件过程模型

- 瀑布模型

    - 计划驱动模型。规范和开发的独立和不同的阶段。
- 增量开发
    - 规范、开发和验证是交错进行的。可能是计划驱动的或敏捷的。
- 面向重用的软件工程
    - 该系统由现有组件组装而成。可能是计划驱动的或敏捷的。
- 在实践中，大多数大型系统的开发过程都结合了所有这些模型的元素。

3.1.1 瀑布模型

瀑布模型阶段

- 瀑布模型中有单独的识别阶段：
    - 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
    - 运维
- 瀑布模型的主要缺点是在流程进行之后难以适应变化。原则上，一个阶段必须完成才能进入下一阶段。

瀑布模型问题

- 不灵活地将项目划分为不同的阶段，难以响应不断变化的客户需求。
    - 因此，此模型仅适用于需求已被充分理解且在设计过程中更改相当有限的情况。
    - 很少有业务系统有稳定的需求。
- 瀑布模型主要用于在多个站点开发系统的大型系统工程项目。
    - 在这些情况下，瀑布模型的计划驱动性质有助于协调工作。

3.1.2 增量开发

增量开发收益

- 降低了适应不断变化的客户需求的成本。
    - 必须重做的分析和文档量比瀑布模型所需的要少得多。
- 更容易获得客户对已完成开发工作的反馈。
    - 客户可以评论该软件的演示并查看已实施的程度。
- 更快速地向客户交付和部署有用的软件是可能的。
    - 与瀑布流程相比，客户能够更早地使用该软件并从中获得价值。

增量开发问题

- 该过程不可见。
    - 管理人员需要定期交付成果来衡量进度。如果系统开发得很快，那么生成反映系统每个版本的文档并不划算。
- 随着新的增量的增加，系统结构趋于退化。
    - 除非将时间和金钱花在重构以改进软件上，否则定期更改往往会破坏其结构。合并进一步的软件更改变得越来越困难且成本高昂。

3.1.3 面向复用的软件工程

- 基于系统重用，其中系统从现有组件或 COTS（商业现货）系统集成。
- 流程阶段
    - 成分分析；
    - 需求修改；
    - 重用系统设计；
    - 开发与集成。
- 重用现在是构建多种业务系统的标准方法
    - 重用在第 16 章中有更深入的介绍。

面向重用的软件工程

软件组件的类型

- 根据服务标准开发的可用于远程调用的 Web 服务。
- 作为要与组件框架（如 .NET 或 J2EE）集成的包而开发的对象集合。
- 配置为在特定环境中使用的独立软件系统 (COTS)。

3.2 过程活动

- 真正的软件过程是技术、协作和管理活动的交错序列，其总体目标是指定、设计、实施和测试软件系统。
- 规范、开发、验证和演化这四个基本过程活动在不同的开发过程中以不同的方式组织。在瀑布模型中，它们按顺序组织，而在增量开发中，它们是交错的。

3.2.1 软件规范

- 确定需要哪些服务以及对系统运行和开发的约束的过程。
- 需求工程过程
    - 可行性研究
        - 构建该系统在技术和财务上是否可行?

    - 需求获取和分析
        - 系统利益相关者对系统有什么要求或期望?
    - 要求规范
        - 详细定义需求
    - 需求验证
        - 检查要求的有效性

需求工程过程

3.2.2 软件设计与实现

- 将系统规范转换为可执行系统的过程。
- 软件设计
    - 设计实现规范的软件结构；
- 执行
    - 将此结构翻译成可执行程序；
- 设计和实现的活动密切相关，可能相互交错。

设计过程的一般模型

设计活动

- *架构设计*，您可以在其中确定系统的整体结构、主要组件（有时称为子系统或模块）、它们的关系以及它们的分布方式。
- *接口设计*，您可以在其中定义系统组件之间的接口。
- *组件设计*，您可以在其中获取每个系统组件并设计其运行方式。
- *数据库设计*，在其中设计系统数据结构以及如何在数据库中表示这些结构。

3.2.3 软件验证

- 验证和确认 (V & V) 旨在表明系统符合其规范并满足系统客户的要求。
- 涉及检查和审查过程以及系统测试。
- 系统测试涉及使用测试用例来执行系统，这些测试用例源自系统要处理的真实数据的规范。
- 测试是最常用的 V & V 活动。

测试阶段

测试阶段

- 开发或组件测试

    - 单个组件独立测试；
    - 组件可以是这些实体的功能或对象或连贯的分组。
- 系统测试
    - 对整个系统进行测试。紧急特性的测试尤其重要。
- 验收测试
    - 使用客户数据进行测试以检查系统是否满足客户的需求。

计划驱动的软件过程中的测试阶段

3.2.4 软件演进

- 软件本质上是灵活的，可以改变。
- 随着业务环境的变化导致需求发生变化，支持业务的软件也必须发展和变化。
- 尽管在开发和演化（维护）之间存在分界线，但随着越来越少的系统是全新的，这越来越无关紧要。

系统进化

关键点

- 软件过程是涉及生产软件系统的活动。软件过程模型是这些过程的抽象表示。
- 通用过程模型描述了软件过程的组织。这些通用模型的示例包括“瀑布”模型、增量开发和面向重用的开发。

关键点

- 需求工程是开发软件规范的过程。
- 设计和实现过程涉及将需求规范转换为可执行的软件系统。
- 软件验证是检查系统是否符合其规范以及是否满足系统用户的实际需求的过程。
- 当您更改现有软件系统以满足新需求时，就会发生软件进化。软件必须不断发展才能保持有用。

第 2 章 – 软件过程

第二讲

3.3 应对变化

- 在所有大型软件项目中，变化都是不可避免的。
    - 业务变化导致新的和变化的系统需求
    - 新技术为改进实施开辟了新的可能性
    - 改变平台需要改变应用程序
- 变更导致返工，因此变更成本包括返工（例如重新分析需求）以及实现新功能的成本

降低返工成本

- 变更避免，其中软件过程包括可以在需要大量返工之前预测可能的变更的活动。
    - 例如，可以开发一个原型系统来向客户展示系统的一些关键特性，并重新定义需求。
- 变更容忍度，其中的流程被设计为可以以相对较低的成本适应变更。
    - 这通常涉及某种形式的增量开发。提议的更改可能会以尚未开发的增量实施。如果这是不可能的，那么可能只需要更改单个增量（系统的一小部分）以包含更改。

3.3.1 软件原型设计

- 原型是系统的初始版本，用于演示概念和尝试设计选项。
- 原型可用于：
    - 帮助需求获取和验证的需求工程过程；
    - 在设计过程中探索选项和开发 UI 设计；
    - 在测试过程中运行背靠背测试。

原型设计的好处

- 提高了系统的可用性。
- 更贴近用户的实际需求。
- 提高设计质量。
- 提高了可维护性。
- 减少开发工作量。

原型开发过程

原型开发

- 可能基于快速原型语言或工具
- 可能涉及遗漏功能
    - 原型应该专注于产品中没有被很好理解的领域；
    - 错误检查和恢复可能不包含在原型中；
    - 专注于功能性而非非功能性需求，例如可靠性和安全性

一次性原型

- 原型应该在开发后丢弃，因为它们不是生产系统的良好基础：
    - 可能无法调整系统以满足非功能性需求；
    - 原型通常没有文档记录；
    - 原型结构通常会因快速变化而退化；
    - 原型可能不符合正常的组织质量标准。

3.3.2 增量交付

- 不是将系统作为单一交付交付，而是将开发和交付分解为增量，每个增量交付所需功能的一部分。
- 用户需求被优先考虑，最高优先级的需求包含在早期的增量中。
- 一旦增量的开发开始，需求就会被冻结，尽管后续增量的需求可以继续发展。

增量开发和交付

- 增量开发
    - 以增量方式开发系统并评估每个增量，然后再进行下一个增量的开发；
    - 敏捷方法中使用的正常方法；
    - 由用户/客户代理完成的评估。
- 增量交付
    - 部署一个增量供最终用户使用；
    - 对软件的实际使用进行更现实的评估；
    - 替换系统难以实现，因为增量的功能比被替换的系统少。

增量交付

增量交付优势

- 每个增量都可以交付客户价值，因此可以更早地使用系统功能。
- 早期增量充当原型，以帮助引出对后期增量的需求。
- 降低整个项目失败的风险。
- 最高优先级的系统服务往往接受最多的测试。

增量交付问题

- 大多数系统需要一组由系统不同部分使用的基本设施。
    - 由于在实现增量之前不会详细定义需求，因此很难确定所有增量所需的公共设施。
- 迭代过程的本质是规范是与软件本身一起开发的。
    - 然而，这与许多组织的采购模式相冲突，其中完整的系统规范是系统开发合同的一部分。

3.3.3 Boehm 螺旋模型

- 过程被表示为一个螺旋，而不是一个带有回溯的活动序列。
- 螺旋中的每个循环代表过程中的一个阶段。
- 没有固定的阶段，例如规范或设计 - 根据需要选择螺旋中的环。
- 在整个过程中明确评估和解决风险。

软件过程的 Boehm 螺旋模型

螺旋模型扇区

- 目标设定

    - 确定了该阶段的具体目标。
- 风险评估和降低
    - 评估风险并开展活动以降低主要风险。
- 开发和验证
    - 选择系统的开发模型，该模型可以是任何通用模型。
- 规划
    - 对项目进行审查，并计划螺旋的下一阶段。

螺旋模型使用

- 螺旋模型在帮助人们思考软件过程中的迭代和引入风险驱动的开发方法方面非常有影响力。
- 然而，在实践中，该模型很少用于实际软件开发。

3.4 Rational统一流程

- 现代通用过程源自 UML 和相关过程的工作。
- 汇集了前面讨论的 3 个通用流程模型的各个方面。
- 通常从3个角度描述
    - 一个动态的视角，随着时间的推移显示阶段；
    - 显示流程活动的静态透视图；
    - 建议良好实践的实践观点。

Rational Unified Process 中的阶段

RUP 阶段

- 成立
    - 为系统建立业务案例。
- 细化
    - 了解问题域和系统架构。
- 建造
    - 系统设计、编程和测试。
- 过渡
    - 在其操作环境中部署系统。

RUP迭代

- 同相迭代

    - 每个阶段都是迭代的，结果逐渐发展。
- 跨阶段迭代
    - 如 RUP 模型中的循环所示，整个阶段集可以递增地执行。

Rational Unified Process 中的静态工作流

工作流程	描述
商业建模	业务流程是使用业务用例建模的。
要求	识别与系统交互的参与者并开发用例以对系统需求建模。
分析设计	使用架构模型、组件模型、对象模型和序列模型创建和记录设计模型。
执行	系统中的组件被实现并构建为实现子系统。从设计模型自动生成代码有助于加速这一过程。

Rational Unified Process 中的静态工作流

工作流程	描述
测试	测试是一个迭代过程，它与实现一起进行。系统测试在实施完成后进行。
部署	产品发布被创建，分发给用户并安装在他们的工作场所。
配置和变更管理	此支持工作流管理对系统的更改（参见第 25 章）。
项目管理	这个支持工作流管理系统开发（见第 22 章和第 23 章）。
环境	此工作流涉及向软件开发团队提供适当的软件工具。

RUP 良好实践

- 迭代开发软件
    - 根据客户优先级计划增量并首先交付最高优先级的增量。
- 管理需求
    - 明确记录客户要求并跟踪这些要求的变化。
- 使用基于组件的架构
    - 将系统架构组织为一组可重用的组件。

RUP 良好实践

- 可视化建模软件
    - 使用图形 UML 模型来呈现软件的静态和动态视图。

- 验证软件质量
    - 确保软件符合组织的质量标准。
- 控制对软件的更改
    - 使用变更管理系统和配置管理工具管理软件变更。

关键点

- 流程应该包括应对变化的活动。这可能涉及原型设计阶段，有助于避免对需求和设计做出错误的决定。
- 可以为迭代开发和交付构建流程，以便在不破坏整个系统的情况下进行更改。
- Rational Unified Process 是一个现代通用流程模型，它被组织成阶段（启动、细化、构建和转换），但将活动（需求、分析和设计等）与这些阶段分开。