Ch 1 **概述**(专业化软件开发、软件工程职业道德、案例\*4)

1. **软件失效**的因素：

①不断增长的系统复杂性②未有效采用软件工程方法

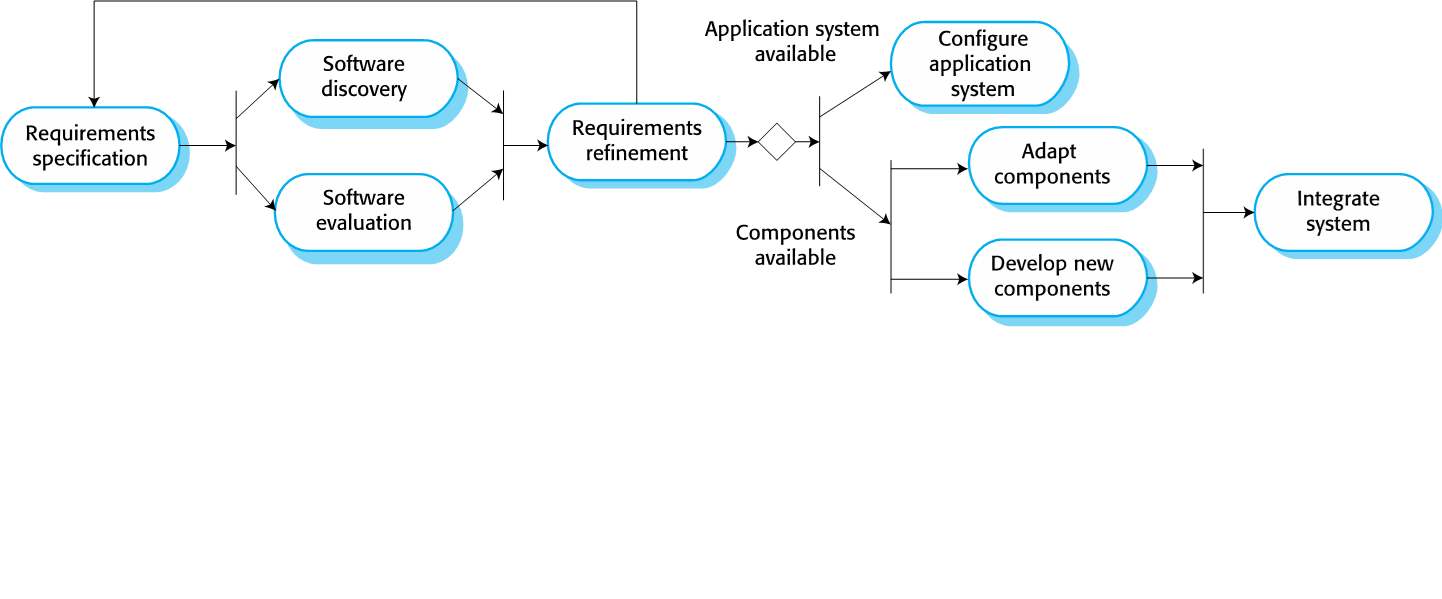
1. **软件**不仅包括程序本身，还包括所有是程序能够正常使用的相关文档、库、支持网站、配置数据等。**影响软件的一般问题**：①异构性②企业和社会的变革③信息安全与信任④规模
2. **好的软件的基本属性**：①向用户提供所需的功能与性能(可接受性)②可维护性、可依赖性、可用性、效率
3. **通用软件产品**和**定制化产品**的关键区别在于软件规格说明和软件变更决策由开发者自己确定还是客户给出。
4. **软件工程**是一个工程学科，涵盖了软件生产的各个方面，从最初的系统规格说明到运行和维护。**面临的挑战**：不断增长的多样性、缩短交付时间、开发可信软件要求。**重要性**：个人和社会more依赖先进的软件系统，就要求以经济且快速的方式开发出可靠可信的系统；长远看运用软件工程方法和技术开发专业化的软件系统比单纯作为个人编程项目编写程序cheaper，无法有效应用软件工程方法将导致更高的测试、质量保障和长期维护的成本。**应用的类型**：独立的应用；基于事务的交互式应用；嵌入式控制系统；批处理~；娱乐~；建模和仿真~；建模和仿真~；数据收集和分析~；系统之系统
5. **软件工程基础适用于所有类型的软件系统**：软件系统开发过程应当是受管理的并被开发人员所理解；可靠性和性能对于所有类型的系统来说都很重要；理解和管理系统规格说明和需求是很重要的；应该尽可能高效地使用已有地资源，在一些适当的地方复用软件。
6. 软件复用已成为构建基于Web的系统的主流方法；~总是增量开发和交付的；~可使用面向服务的软件工程来实现，软件构件是独立的Web服务；创建富客户端界面
7. 保密；诚实陈述工作能力；知识产权；计算机滥用

Ch 2 **软件过程**(Software Processes)

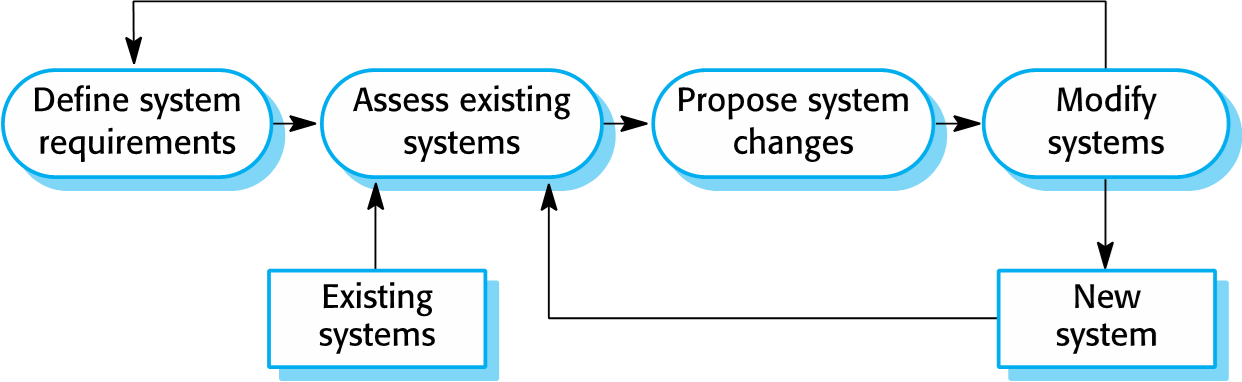
1. Process descriptions:产品交付物；角色；前置后置条件
2. **瀑布模型**：包含规格说明、设计、实现、测试及维护这几个离散的开发阶段。原则上一个阶段必须在进入下一个阶段之前完成。在实践中，阶段之间存在显著迭代。**阶段**:需求分析和定义;系统和软件设计;实现和单元测试;集成和系统测试;运行和维护. **缺点**:很难容纳变化

**适用**:嵌入式系统(硬件)\关键性系统(安全)\大型软件系统

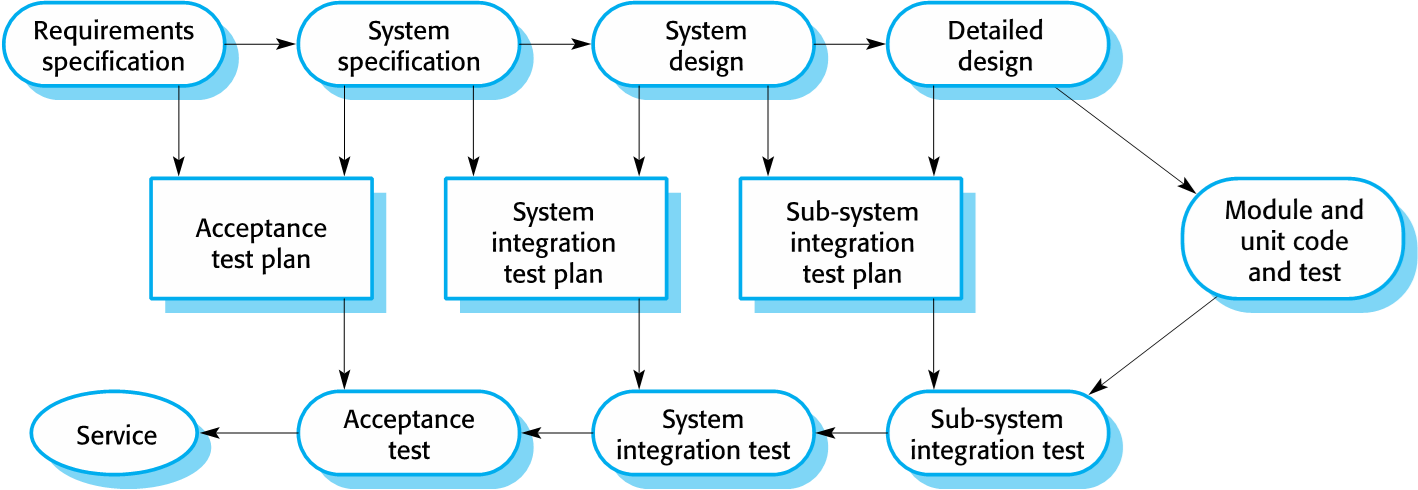
1. **增量式开发**使得规格说明、开发和确认活动交错进行，系统开发体现为一系列版本，每一版本在前一版基础上加一些功能，软件以增量的方式进行交付和部署. **优势**:降低了实现需求变更的成本,重新分析和修改文档的工作量要少很多;在开发过程中更容易得到客户对于已完成的开发工作的反馈意见;及时并未将所有的功能包含其中,也使得在早期向客户交付和部署有用的软件成为可能,客户可以更早地使用软件并从中获得价值. **问题**:过程不可见,管理人员需要常规的交付物来掌握进度,系统快速开发,产生系统每个版本的文档就很不划算;伴随着新的增量的增加,系统结构会逐渐退化,代码凌乱,向系统添加新特性困难昂贵.
2. **集成和配置**依赖于可复用的构件或系统.关注在新的使用环境中配置这些构件并将它们集成为一个系统.
3. 3类软件构件常被复用:经过配置后可以在特定环境中使用的独立应用系统;作为一个构件或一个包开发的并且将与一个构建框架相集成的一组对象;按照服务标准并且可以通过互联网进行远程调用的Web服务
4. **面向复用的软件工程**:



1. **优劣势**:降低软件开发量,降低成本和风险,实现更快软件交付,但需求权衡是不可避免的,这导致系统不完全满足用户的真实需求,还可能失去一些对系统演化的控制.
2. **软件规格说明**:理解定义系统需要提供哪些服务并识别对于系统开发和运行的约束.**需求工程活动**需求抽取与分析(可能会开发系统模型和原型),需求规格说明(用户与系统需求),需求确认(纠错修改需求文档)
3. **软件设计和实现**:对将要实现的软件的结构\系统所使用的数据模型和结构\算法\系统构件间的接口的描述;将需求规格说明开发为一个可执行的系统以交付给客户. **设计输入**:平台(OS\DB\中间件\其他应用系统:软件将会在其中运行的环境)信息,软件需求,数据描述;**设计活动**:体系结构设计,数据库~,接口~,构件选取和~;**设计输出**:系统体系结构,数据库设计,接口规格说明,构件描述
4. **软件确认**:验证和确认目的是确定系统是否符合它的规格说明,同时是否符合系统客户的期望.**程序测试**用模拟的测试数据运行系统,是最基本的确认技术. Testing stages:构件测试(函数/对象类/实体构成的内聚的分组单独测试),系统~(构件间交互/接口/需求),客户~
5. **软件演化(维护)**:修改已有的软件系统满足新的需求

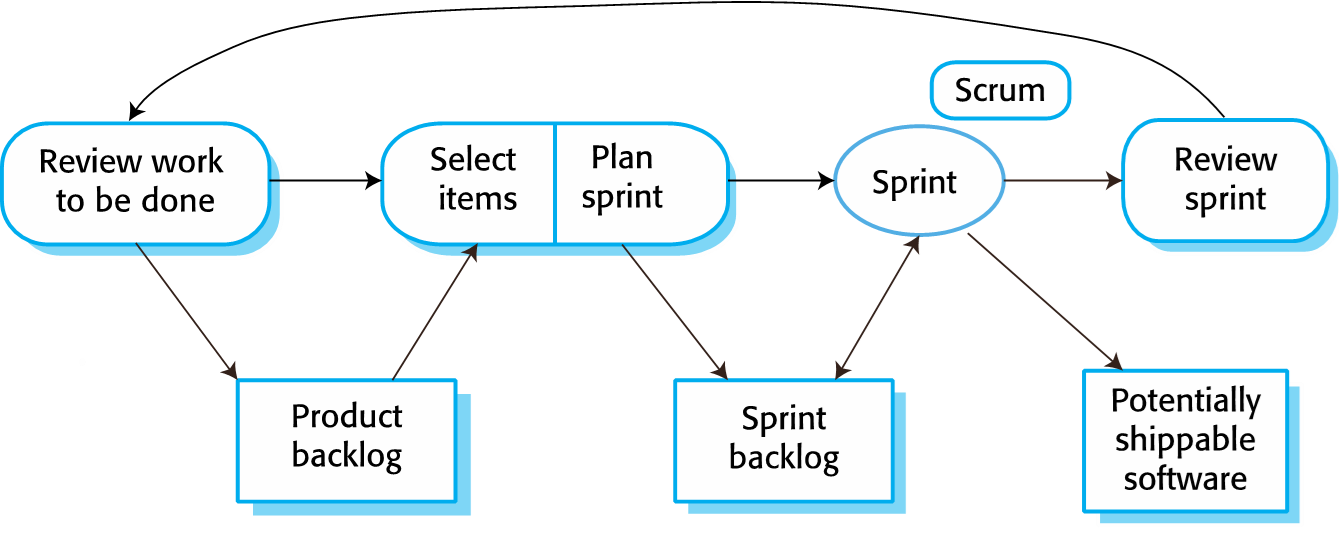


1. **计划驱动的软件过程中的测试阶段**(V-model):



1. 过程应包含应对变化的活动.系统需求随着业务应对外部牙利\竞争和管理优先级的需要等变化而发生改变;新技术出现使新的设计和实现方法成为可能;平台改变APP改变
2. 变化意味着已经完成的工作必须重做(**返工**),需求重新分析,实现新功能,都是成本.**降低成本**:变化预测;变化容忍
3. **原型**prototyping:系统或系统的一部分的早期版本被快速开发以检验客户需求以及设计决策的可行性,用于演示概念\尝试候选设计方案\更好地理解问题以及可能的解决方案. 在需求工程过程中,帮助对系统需求进行抽取和确认;在系统设计过程中,探索软件解决方案和系统用户界面的开发;测试过程中,run back-to-back tests. **优点**:改进系统的可用性;更贴合用户真实需求;改善设计质量;提高可维护性;减少开发工作.**原型开发的过程**:定义原型目标(原型计划),定义原型功能(概要设计),开发原型(可运行原型),评估原型(评估报告)
4. **增量式交付**:根据服务优先级将服务分配到各个增量,优先级最高的首先被实现和交付并部署到客户的工作环境供试验,还可以改进后续增量的需求. 开发过程需求不能改. **优点**:客户可以将早期的增量作为原型使用,从中获得关于后续系统增量需求的经验;客户不用等到整个系统交付就能从系统中获得价值;变更容易加进来;整个项目失败风险降低;最重要的系统服务测试最充分,这不太可能失效. **问题**:当**新系统**准备**替换**一个已有系统时,迭代化交付会有问题;大多数系统需要一组由系统的不同部分使用的基础设施;迭代化过程的本质是规格说明与软件一起开发,与组织的采购方式冲突,开发**合约**要求完整系统规格说明.
5. **过程改进**:意味着理解当前的过程,并对其进行改进以提升软件质量、降低成本或加快开发过程的一种方式.2种方法
6. **过程成熟度方法**:关注改进过程和项目管理,并将好的软件工程实践引入组织中. **过程度量**(对软件过程或产品的属性);**过程分析**(识别过程中的弱点和瓶颈),**过程改变**(提出改变方法,引入变更,改进后再度量看是否改善,由此循环过程)
7. **敏捷方法**:软件按照增量进行开发和交付,过程文档和管理开销被最小化,开发关注代码本身,而不是支持文档. **主要特点**是快速交付功能以及对客户需求变更的快速响应. **The SEI capability maturity model**:初始级(uncontroll),受管理级,已定义级(关注组织的标准化),量化管理级,优化级

Ch 3 Agile SD**敏捷软件开发**(敏捷方法\技术\项目管理\伸缩)

1. **敏捷方法的共性**:规格说明\设计和实现过程交织在一起;系统按照一些列增量进行开发,利益相关者参与每个增量的规格说明和评估;新版本快速交付以评估;使用广泛的工具来支持开发过程;最小化文档:关注代码
2. **敏捷宣言manifesto**:概括了敏捷软件开发方法背后的基本思想的一组原则. 个体和交互胜过过程和工具;可工作的软件>全面的文档;客户协作>合同谈判;响应变化>遵循计划. **敏捷方法的原则principles**:客户参与\拥抱变化\增量交付\保持简洁\人而不是过程, **适用于**软件企业开发的用于市场销售的中小规模产品\组织内的定制化系统开发
3. **极限编程**:**极限**在于多个新版本在一天之中开发;增量每两周交付给客户;每个build后必测试，测试成功接受该生成。**XP发布周期**:环(选择本次发布的**用户故事**,将用户故事分解为任务;规划发布;开发\集成\**测试**;发布软件;评估系统). **XP实践**:共同拥有权\**持续集成**\现场客户\**结对编程**\简单设计\重构\小的发布\可持续的步调\测试先行的开发; **用户故事**:解释一个软件或系统如何被使用的情形以及关于系统之间可能发生的交互的一种自然语言描述. **重构**:修改一个程序以改进它的结构和可读性,同时不改变其功能.(去除重复代码\对属性和方法进行整理和重命名\用对定义的方法的调用来代替相似的代码段).**增量开发和计划驱动重要区别:系统测试方式.**
4. **XP testing features**:测试先行的开发;基于场景的增量测试开发,用户参与测试开发和确认,使用自动化测试框架
5. **测试先行的开发**:可执行的测试是在程序代码之前编写的,每对程序进行修改后都会自动执行这组测试.**问题**:程序员更喜欢编程而不是测试,有时会在编写测试走捷径;有些测试很难增量地编写;判断一组测试的完备性很难.
6. **结对编程**:程序员两两配对工作而不是独立工作来开发代码的一种开发情形.**好处**:支持对于系统的共同所有权和共同责任的思想;扮演了非正式的评审过程的角色,∵每一行代码都由至少两个人看着;鼓励通过重构改进软件的结构;知识共享很重要,∵降低由团队成员离开而带来的项目风险.
7. **敏捷项目管理**:Scrum是基于冲刺sprint(短开发迭代周期)的一种敏捷开发方法.可以和其他敏捷方法一起用作敏捷项目管理的基础. **3阶段**:初始阶段:大纲规划,确立总目标并设计软件体系结构;一系列冲刺周期;结束项目,完成所需文档.
8. **Scrum benefits**:产品被分解为一组可管理\可理解\利益相关者可以对应上的条块;不稳定的需求不会影响进度;整个团队都对所有的一切保持可见,提升团队交流沟通和士气;客户按时看到增量的交付并获得产品如何工作的反馈;客户和开发者建立了信任与积极的文化,每个人都期望项目成功
9. Scrum sprint cycle
10. Scaling agile methods**敏捷方法的伸缩**:**规模化**以处理大系统开发,应用范围从专业的开发团队**扩展到**在一个有着多年软件开发经验的大企业内更加广泛地使用.**问题**:非正式性与大型企业基于法律的合同定义不相符;最适合于新的软件开发而不是软件维护(缺少产品文档\保持客户参与\开发团队的延续性),大型企业软件成本大多在维护;适用于小的\同处一地的团队.
11. 规模化和扩展敏捷方法的根本要求是与计划驱动的方法集成, Agile&plan-based因素:系统:规模\类型(复杂性\实时)\生命周期\外部监管;团队:人\开发技术\组织分布\竞争力;组织:合同\交付\文化
12. 系统之系统;棕地系统(关注与现有系统的交互,很难灵活);集成创建新系统关注配置而不是代码开发;受外部规则和监管制约;采购和开发时间长;各样利益相关者→IBM敏捷规模化模型ASM
13. **规模化的敏捷方法共性**:完全增量的需求工程方法是不可能的;没有哪个人能单独作为产品负责人或客户代表;不能只关注系统代码;必须设计和实施整个团队的沟通机制;集成时持续集成无法实现,保持频繁系统构建和常规的系统发布是重要的.多团队Scrum:角色重复(产品负责人和Scrum主管),产品架构师,发布同步;**面向组织扩展**:难∵项目经理没有敏捷方法经验有风险,官僚主义有质量规程和标准,团队人员技术参差不齐,文化上的抵制

**Ch 4 需求工程RE**

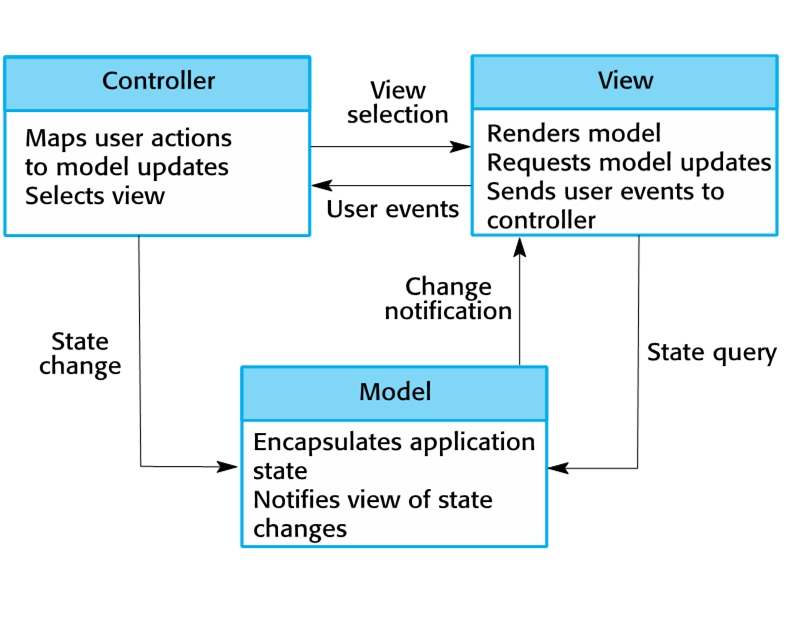
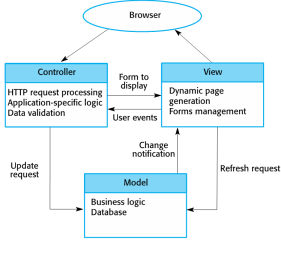
1. 对于一个系统的**需求**是关于该系统应当提供的服务以及对其运行的约束的描述.**用户需求**是高层的抽象陈述(自然语言和图形),而**系统需求**是关于系统功能的详细和正式的定义(结构化文档). 找出\分析\文档化并且检查这些服务和约束的过程被称为RE.
2. **功能性需求**(取决于软件类型\期望用户\书写需求方法):对系统应该提供的服务或功能的陈述(一致\完整).**非功能性需求**:对系统的约束或期望行为的陈述.约束可针对开发软件的涌现特性(可靠性\响应时间\存储使用)或针对开发过程(IDE\编程语言).分类:产品~,组织~,外部~. 实现跨越整个系统∵影响系统的体系结构;会产生多个相互联系的功能性需求. 应该可测试
3. **需求抽取**:通过和利益相关者交互发现需求(需求发现和理解,~分类和组织,~优先级排序和协商,~文档化,循环).**问题**:利益相关者不知道想从系统得到什么,用他们工作的知识表达需求,需求冲突,组织和政策因素影响,需求(可变)所处经济和业务环境是动态的.
4. **访谈**:开发者与系统利益者谈论他们做的事(封闭式\开放式),虚心倾听,避免预设,用跳板性的问题\需求建议\原型等提示受访人;**ethnography**(**人种学调查**):观察人们做自己的工作来了解他们使用哪些制品&如何使用.从实际工作方式\与他人的合作\对他人活动的了解中得出的需求.不助于创新.
5. **故事和场景**:关于系统使用的高层描述;场景是对于系统的典型使用方式或者用户执行某些活动的典型方式一种描述.(结构化故事,最初假设,常规事件流,可能问题,其他活动,结束时的系统状态)
6. **需求规格说明**:在需求文档中正式撰写用户和系统需求的过程.
7. **书写系统需求的方法**:自然语言,结构化自然语言,图形化表示,数学规格说明;可以作为系统实现的合约的一部分. **需求和设计无法分离**:可能要设计一个初始的系统体系结构来帮助组织需求规格说明/满足非功能性需求;系统须与现有系统互操作,对设计构成约束并提出新需求.
8. **使用自然语言进行需求规格说明的问题**:表述**不清楚**,功能和非功能**需求混合**在一起,多个需求混合到一条语句中表述
9. Use case用况(UML的一种):是一种使用图形化模型和结构化文本描述用户和系统间交互的方式.
10. 软件需求文档:是关于系统开发者应当实现的所有东西的正式陈述. **用户**:系统客户\管理人员\系统工程师\测试~\维护~
11. **需求确认**:检查需求是否定义了客户真正想要的系统的过程.**检查**:正确性\一致性\完整性\现实性(预算)\可验证性(可测试),**技术**:需求评审(-\来源\变更不影响其他需求),原型化,测试用例生成
12. **需求变更**:由于业务和技术环境发生改变/为系统付钱的人和系统用户不是同一批,利益相关者需求冲突,优先级矛盾,要折中.
13. **需求管理**:管理对需求的变更以保证所做的变更是经过适当的分析并在系统中保持追踪的过程. 维护需求间依赖关系连接,这样就可以评估需求变更的影响. **decisions**:需求标识,变更管理过程(评估变更影响),追踪关系策略,工具支持(需求存储\-\-管理).
14. **需求变更管理**:问题分析和变更规格说明(变更请求合理或撤回),变更分析和成本考虑(评估成本),变更实施(模块化,可修改性).

**Ch 5 系统建模**

1. **系统建模**modeling:建立系统**抽象**模型的过程,其中每一个模型表示系统的一个不同的**视角**或观点.基于UML的图形化表示系统
2. **系统视角**:外部视角(系统的上下文或环境),交互~(系统与环境或系统构件之间的交互),结构化~(组织或处理的数据结构),行为~(动态行为以及如何响应事件)
3. **活动图**描述一个过程或数据处理中所包含的活动,**用况图**描述系统与其环境之间的交互,**顺序图**描述参与者与系统间以及系统构件间的交互,**类图**描述系统中对象类以及这些类之间的关系,**状态图**描述系统如何对内部和外部的实践做出响应.
4. **图形化模型的使用方式**:推动关于现有或所设想的系统的讨论,使讨论聚焦;文档化现有系统;用于生成系统实现的详细系统描述
5. **上下文模型**:描述一个所建模的系统是如何被置于包含其他系统和过程的环境之中,帮助定义待开发系统的**边界**(哪些属于/不属于系统).类图.可以与**业务过程模型**(UML活动图显示)一起使用．
6. **交互模型**:用户交互建模帮助识别用户需求,建模系统间交互突出可能出现的通信问题,建模构建交互帮助理解提出的系统结构是否能实现所要求的系统性能和可依赖性. **用况建模**:每个用况表示一个包含与系统的外部交互的离线任务.**顺序图**:显示在一个特定的用况或用况实例执行过程中发生的交互序列.
7. **结构模型**:按照构成系统的构件以及它们之间的关系显示系统的组织,是描述系统设计组织的静态模型或系统执行时的组织的动态模型.**类图**,**泛化**(△类继承),**聚集**Aggregation(⋄一个对象(整体)由其他对象(部分)组成)
8. **行为模型**:关于系统在运行时的动态行为的模型.**数据驱动的模型**:业务系统.描述处理输入数据以及生成相关的输出过程中所涉及的动作序列. **数据流图**(用活动图表示),**顺序图**. **事件驱动的模型**:实时系统具有有限的状态.描述系统如何对外部和内部事件做出响应. **状态图**.
9. **模型驱动的体系结构**MDA:在构建一组系统模型的基础上开发软件的方法,这些模型可以被自动或半自动化地处理以生成一个可执行的系统.**计算无关模型**CIM对系统中使用的重要领域抽象进行建模,**平台无关模型**PIM在不涉及实现的情况下对系统的运转进行建模;**平台相关模型**对PIM转换得到,每个应用平台都有一个单独PSM,存在多层次,每个层次增加一些平台相关的细节. **Cons**:对讨论有用的抽象对实现而言并不总是正确的抽象;对于复杂系统,实现不是主要问题,RE\信息安全可依赖性\遗留系统集成\测试更重要;追求平台无关性只对大型\长生命周期的系统有意义;平台的模型转换器成本可能超过自动生成代码的节省成本.

**Ch 6 体系结构设计**

1. **体系结构设计**是设计和RE之间的关键性衔接环节,∵它会确定组成一个系统的主要结构构件以及它们之间的关系.输出是一个**体系结构模型**,描述系统如何被组织.基础是**分离和独立性思想**,∵这可以使变更被局部化.
2. 两个抽象层次:**小体系结构**关注单个程序的体系结构;**大体系结构**关注包括其他系统\程序和程序构件的复杂企业系统体系结构. explicit architecture**优点**:聚焦利益相关者的讨论,系统分析:影响非功能性特性;大范围复用(相似需求的系统有相似的体系结构).用简单**框图**表示系统结构高层样貌. Use of体系结构模型:鼓励对系统设计进行讨论的方式;文档化已经设计好的体系结构的方式.
3. **体系结构设计决策**:问题:是否可复用?系统将如何分布到各个硬件核或者处理器上?体系结构模式或风格?什么策略来控制系统中构件的运行?系统的体系结构如何文档化?什么样体系结构组织最佳实现系统非功能性需求?系统结构构件如何分解为子构件?什么方法组织系统?
4. 体系结构风格&结构由非功能性需求来决定:性能(关键操作集中部署到一台PC,用大构件减少通信)\信息安全(层次化结构)\安全性(集中在单个构件)\可用性(包含冗余构件)\可维护性(容易改变的细粒度\自包含的构件)
5. **体系结构视图**Views:”4+1”软件体系结构视图模型:共同的用况或场景+逻辑(对象类)\进程(运行时,交互)\开发(软件如何面向开发任务分解为构件)\物理视图(硬件和构件如何分布在处理器上)
6. **体系结构模式**Patterns:已经在很多不同的软件系统中尝试和验证过的一种软件体系结构的抽象描述.模式中所描述的信息包括该模式适合在何地使用与体系结构中构件的组织结构. **将模式作为一种表示\分享\复用软件系统相关知识的思想.**
7. **MVC**:将呈现和交互从系统数据中分离出来.系统被组织为3个相互交互的逻辑构件.模型构件管理系统数据以及相关的对这些数据的操作.视图构件定义并管理数据呈现给用户的方式.控制器构件管理用户界面(如按键\鼠标点击)并将这些交互传递给视图和模型.**优点**:允许数据独立于呈现方式进行变更,反之亦然.支持以不同的方式呈现同样的数据,在某个呈现方式中进行的修改可以在所有呈现方式中显示.**缺点**:当数据模型和交互比较简单时,可能包含额外的代码以及代码复杂性.**适用于多种**交互呈现/它们**未知**.

1. **分层体系结构**:将系统组织为多个层次,每个层级与一些相关的功能相联系.每个层次向其上的一层提供服务,最底层:核心功能.**优点**:只要接口不变,允许对整层进行替换.可以在每层提供冗余设施以增加系统可依赖性.**缺点**:实践中各层之间分离很难做到,较高层次可能不得不与较低层次直接交互,而不是紧邻着的一层.性能也是问题,∵当服务请求在各个层次上进行处理时需要多层解析.**适用于**在已有系统之上构件新的设施时使用;开发涉及多个团队,每个团队负责一个层次上功能;存在多个层次上的信息安全需求
2. **知识库体系结构**:描述了一组相互交互的构件如何共享数据.系统的所有数据在所有系统构件都能访问的中心知识库中进行管理.构件不能直接交互,仅通过知识库交互. **优点**:构件保持独立,不需要知道其他构件的存在;一个构件进行的修改被传播到所有构件;所有数据一致进行管理(同时备份).**缺点**:单点失效,知识库的问题会影响整个系统;通过知识库通信效率不高;将知识库分布到多台计算机上比较困难. **适用于**系统生成大量需要长时间保存的信息时;数据驱动的系统:知识库中添加新数据时会触发一动作或工具;数据由一构件生成同时由另一构件使用的应用.
3. C/S**模式**:分布式系统.一组向其他构件提供服务的服务器.一组使用这些服务的客户端.一个允许客户端访问这些服务的网络.**优点**:服务器分布在网络上,通用的功能可以向所有客户端开放使用,不需要由所有服务实现.**缺点**:每个服务都可能单点失效,因此易受DoS攻击或服务器失效;性能(取决于网络和系统)不可预测,管理问题如果服务器属于不同的组织.**适用于**当一个共享数据库中的数据必须从一系列不同的位置进行访问;系统的负载多变的情况.
4. **管道和过滤器体系结构**:系统中的数据处理通过组织,每个处理构件filer可以分离开来并执行一种数据转换.功能性的变换处理输入并产生输出.**适用于**数据处理应用,批处理,**不适于**交互系统.
5. **应用体系结构**:对其进行配置和调整以创建特定需求的系统. uses of ~:系统结构设计的起点,设计检查表,组织开发团队工作的方式,评价构件复用的方式,谈论应用时的词汇表. **事务处理系统**:处理用户获取数据库中信息的请求或更新数据库的请求.数据库事务为操作序列.交互式.pipe and filer; **信息系统**:基于事务的系统∵和共享数据库交互,layered结构; **语言处理系统**:将一种语言翻译为另一种语言的系统.例编译器.repository符号表,管道过滤器(词法\语法\语义分析顺序化)

**Ch 7设计和实现(使用UML的面向对象软件设计)**

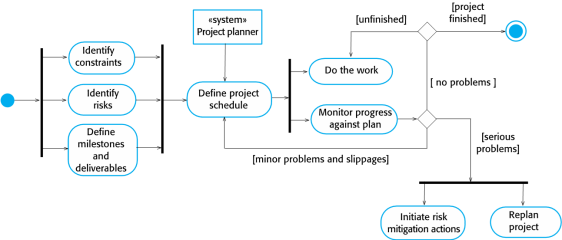
1. 在一个软件项目的早期阶段必须做出的决策是**确定自己构建还是购买应用软件**.进行适应性调整和裁剪的成品应用系统commercial-off-the-shelf systems (COTS)，指可以采购到的具有开放式标准定义的接口的软件或硬件产品，可以节省成本和时间。
2. 为开发一个系统设计(从概念到详细的**面向对象设计**),你需要:
3. 理解并定义**上下文**以及与系统的**外部交互**:**系统上下文模型**(框图)是一种结构化模型,展示了所开发的系统的环境中的其他系统,**交互模型**(用况模型)是动态模型,显示系统在使用时如何与环境进行交互,两者呈现的关于系统与其环境之间关系的视图是互补的.
4. **体系结构设计**:识别构成系统的主要构件以及它们之间的交互.
5. **对象类**(系统中主要对象)**识别**:方法:对所要构建的系统的自然语言描述进行**语法分析**;使用**应用领域**中有形的实体或事物;使用一种**基于场景**的分析方法,识别所需的对象\属性\操作.
6. **开发设计模型**models:设计模型所需要的详细程度取决于开发系统的类型与开发人员所使是用的设计过程.当使用UML来开发一个设计时,应当开发:**结构模型**使用静态类及其关系描述系统的静态结构.**动态模型**描述系统的动态结构并展示所期望的系统对象之间的运行时的交互.3种UML模型:**子系统模型**展示如何对对象进行逻辑分组以构成内聚的子系统,**顺序模型**展示对象的交互序列,**状态机模型**展示对象如何在事件响应中改变自己的状态.
7. **接口**规格说明:与系统构件相关的属性和操作的规格说明.接口时作为访问构件功能的一种手段使用的.对象与接口不是一对一.
8. OCL(**对象约束语言**):用于定义适用于UML模型中的对象类和交互的断言.定义构件.描述接口的语义等.
9. **设计模式**patterns:对于**通用问题**的一个经过成功验证的**解决方案**,以一种**可复用**的方式描述了其中的经验和好的实践.是一种可以以多种不同方式进行实例化的**抽象表示**.依赖**继承和多态**等对象特性来提供通用性.**模式4元素**:名字\问题描述\解决方案\效果.
10. **观察者模式**:将对象的状态的呈现与对象本身分离开.用于为状态信息提供多种呈现方式时.抽象对象subject/observer,具体对象继承相关抽象对象属性.提高呈现性能的优化无法实现.
11. **设计问题**:(observer)向多个对象告知一些其他对象的状态发生了变化,(Façade门面)整理面向一些相关的且经常是增量地开发出来的对象的接口,(迭代器)为一个合集中的元素提供一种标准的访问方式,不用考虑合集如何实现,(装饰者)允许在运行时扩展一个已有类的功能. (使用模式是对思想进行复用)
12. **实现问题**:**复用**可以在多级别(抽象级(设计\体系结构模式)\对象级(编程语言库)\构件级(框架)\系统级(COTS))上发生.**成本**:寻找可复用的软件和评估是否满足花费的时间成本,购买软件成本,适配和配置的成本,集成代价.**配置管理**:目的是支持系统集成过程以使所有开发者都可以以一种受控的方式访问项目的代码和文档,找出代码和文档做了哪些修改,以及对构件进行编译和链接来创建一个系统.**宿主机**-**目标机**host-target**开发**:软件在host上开发,在target上运行.软件开发平台工具:集成编译器,语法制导的编辑系统,语言调试系统,图形化编辑工具,测试工具,支持重构和程序可视化的工具,配置管理工具. IDE:集成开发环境是一组位于一些公共框架和用户界面中的支持软件开发的不同方面的软件工具.一个通用的IDE是一个为所开发的软件提供数据管理设施以及允许各种工具一起工作的集成机制的框架,可以容纳各种软件工具.
13. Component/system deployment factors:一个构件的硬件和软件需求,系统的可用性需求(高可用性要求构件部署到多个平台),构件通信.
14. **开源开发**:一种软件开发方法,系统的源代码公开,鼓励外部用户参与到系统的开发当中去.Open source issues:正在开发的产品一个利用开源构件嘛?使用开源的方法开发软件嘛?开源许可证:GPL(须开源),LGPL(修改了要开源),BSD(不可修改)License management:建立系统维护开源构件信息,了解不同类型许可证,了解构件的演化路径,进行关于开源软件的教育,建立审计系统,参与开源社区.

**Ch 22 项目管理(风险管理\人员管理\团队协作)**

1. **软件项目管理者的任务**是确保软件项目满足和服从预算和工程进度的约束,并确保交付高质量的软件产品.(满足客户期望,团队凝聚力). **不同之处**/**有难度**∵软件产品无形的,大型软件项目常是一次性的项目,无以往经验,软件开发过程是可变的并且与特定组织相关(不能准确预见何时会出问题).**影响因素**:公司规模,软件客户(内部/外部客户/政府组织),软件规模,软件类型(消费性/安全性),组织文化(官僚主义),软件开发过程(敏捷/正式过程).**基础项目管理活动**:项目计划,风险管理,人员管理,工作报告,编写建议书(项目目的,如何实施,成本和进度估计,以获得合同)
2. **风险管理**:识别风险,评估风险的严重性,规划风险应对措施,监控软件和软件过程中的风险的过程.项目风险(影响进度,资源),产品风险(影响质量,性能),业务风险(影响软件开发组织,软件产品购买组织). **风险识别**:集体讨论或管理者经验识别,风险检查表(估算/组织/人员/需求/技术/工具风险)→风险实例,削减以便于管理; **风险分析**:对风险的可能性和严重程度给出一个等级的判断,根据严重程度排序,所有灾难性风险与可能性超过中等&影响严重的风险. **风险计划**:通过制订策略来对威胁项目的关键风险进行管理.3类策略:规避(降低可能性)/最小化策略(减小影响)/应急计划(发生后应对); **风险监控**:检查之前对产品,过程以及业务风险的假设是否改变的过程.对每一个识别的风险定期评估,确定出现可能性是否改变,影响后果是否改变.风险监测指标
3. **人员管理**:影响管理人员和他所管理的团队成员之间关系的4要素:一致性(同等对待),尊重(尊重技能差异),包容(不一样的见解),诚实(对待自己的技术水平).**激励人员**:协调工作和提供良好工作环境方面能使员工尽可能更有效率地工作.确保社会需求(提供与同事交谈的时间和场所,团队认同感),受尊重需求(成绩给予认同,支付报酬),自我实现需求(培训与发展机会).将团队作为一个整体去激发他们的积极性.3种职业人士(面向任务型,自我型,交互型)
4. **团队协作**:小组不宜太大,a cohesive group**有凝聚力的小组的好处**:能够建立起小组自己的质量标准,成员互相学习,互相帮助,知识能够共享,鼓励重构和持续改进.**影响团队合作**:小组中的人(需要不同类型人员),小组的组织(成员尽其所能按时完成任务),技术上和管理上的交流.互补的成员,缓解工程师竞争问题.小型项目**组织结构**非正式的.敏捷开发小组也是.需要成员都是有经验有能力,集体决策.而大型项目则层次化组织,变更会引起多部分变化,从下层到上层沟通少,自上而下的交流意味少机会使用新技术.**小组的沟通影响因素**:小组规模,小组结构(非正式更有效),小组构成(混合性别个性),物理工作环境(私人工作+小组工作区域),可用的沟通渠道

**Ch 23 项目计划project planning**

1. **包含**:把工作分解开来并分配给团队成员,预见可能出现的问题,准备试探性解决方案去应对.**发生在3阶段**:投标建议书阶段,项目开始阶段,贯穿于项目过程中.
2. **软件报价**:**成本**:工作量成本,软硬件费用,差旅费,培训费.**影响因素**:合同条款(开发者保留源代码所有权),成本估算的不确定性(增加应急开支),财务状况(降低报价),市场机遇(为进入一个新软件市场),需求易变性.Pricing to win(**赢得合同的报价**):软件公司对于客户期望的报价有所了解,继而根据客户期望的价格投标.项目中不变的是成本不是需求.
3. **计划驱动的开发**:给开发过程制订详细的计划的软件工程方法.**pros**:早期计划能够很好地考虑组织问题(组织员工等)能够在项目开始前发现潜在问题和依赖性,而不用等到项目开发时.**cons**:由于软件开发和使用环境的变化,必须修改许多早期的决策.**项目计划书包含**:引言(项目目标,约束),项目组织,风险分析,硬件和软件资源需求,工作分解,项目进度安排,监控和报告机制.**里程碑**是进度安排中能够估计项目进展的那些位置点,**可交付物**是交付给客户的项目成果.



适当保持悲观

1. **项目进度安排**scheduling:决定如何组织项目工作,将其分割成单独的一个个任务,并且何时以何种方式完成各项任务的过程.估算任务时间,成本,人员,软硬件资源.协调并行任务,充分利用工作量资源.尽量避免各个任务之间不必要的依赖性.**进度安排表示方式**:基于日历时间的条状图,活动网络.项目活动是基本的计划单元,每个活动有:以天或月计算的工期(持续时间),工作量估计,活动完成的ddl,定义好的终点(文档/举办评审会/成功执行了所有测试).



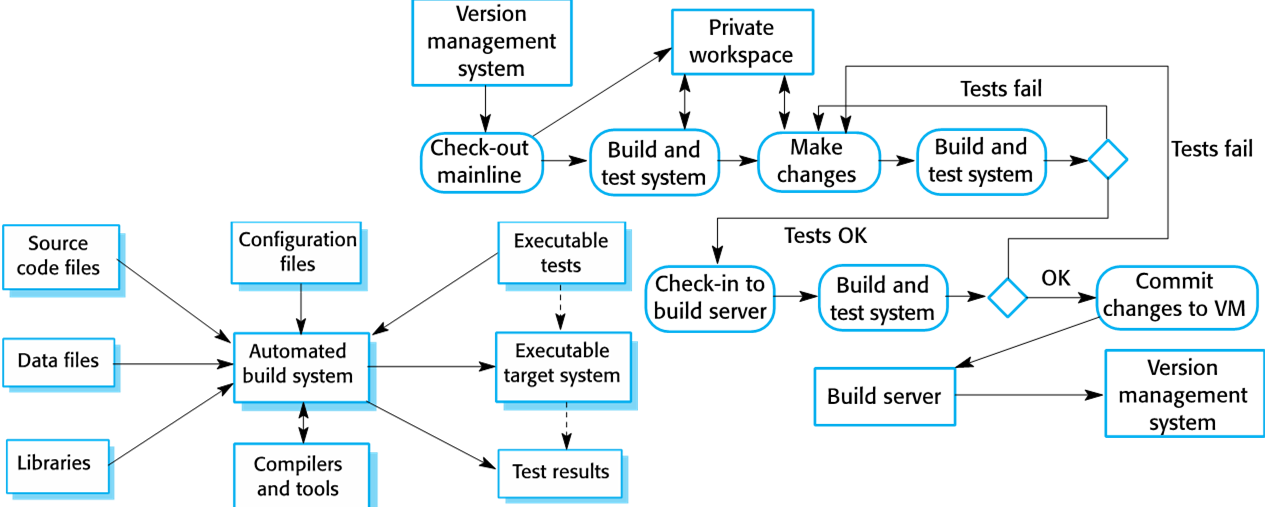
1. **敏捷计划**:两阶段的计划方法:**发布计划**(对未来几月的展望并决定系统的发布版本中应该包含的功能),**迭代计划**(计划系统下一个增量)
2. **计划游戏**planning game:基于**用户故事**,按时间**相对**排序,分配理论**工作量点**,用”速度”估算整个系统工作量.发布计划:选择和完善上述故事.选择发布日期,检查故事的工作量是否满足发布日期.迭代计划是开发增量交付系统的第一步.故事拆分为多个开发任务,然后任务分配(pros:整个项目组对迭代过程中要完成的任务有一个整体认识,开发者可自主选择要完成的任务).中间有进度评审,可减少工作量,但交付进度任何时候都不会延长.
3. **估算技术**:**基于经验的技术**:管理者使用之前项目和应用领域的经验估算要求的未来工作量(主观,新软件可能没有可用经验),**算法成本建模**:使用一种公式化的方法计算项目的工作量,基于对产品属性(规模,过程特点和参与员工的经验)的估计.
4. **工作量估算**:effort=A×SizeB×M,A常量因子依赖于组织的实践经验和所开发的软件类型,Size表示代码规模,B:软件复杂度,M乘数因子反映过程,产品,开发属性. **算法成本模型很复杂而且很难使用,模型参数需要校准.** SLOC**源代码行数**作为基本规模度量标准,基于已开发的相似系统代码规模估算/估算功能点和应用点数量再公式化转换为代码行数/根据构件的相对大小排序,通过参考构件转化为代码规模.
5. COCOMO**成本建模**:**应用组合模型**,基于应用点数量对所需工作量建模PM=(应用程序点总数NAP×(1-%reuse/100))/应用点生产率PROD;**早期设计模型**,在得到需求后用功能点估算PM=2.94×Size(1.1-1.24)×M,M=个人能力PERS\*个人经验PREX\*产品可靠性和复杂性RCPX\*所要求的复用RUSE\*平台困难程度PDIF\*进度SCED\*支持设施FCIL(6档);**复用模型**:用于计算集成可复用构件和自动生成程序代码所需的工作量,A,B,M同上,新源代码的等价行数ESLOC=(必须修改的复用构件代码行数ASLOC×(1-可以自动修改的服用代码所占百分比AT/100)×改写调整因子AAM),AAM=评估因子AA+理解部分SU+改写部分AAF;**后体系结构模型**:当设计出系统体系结构后可以对软件规模做更精确的估算Size=要开发的新代码行数估算SLOC+ESLOC+由于需求变更可能要修改的代码行数的估算,B通过估算(体系结构/风险解决方案(风险分析程度)/开发灵活性/有先例可循/团队凝聚力/过程成熟度)0-5打分相加/100+1.01得到. **项目的工期和人员配备**:项目理论进度TDEV=3×(PM)(0.33+0.2\*(B-1.01)),增加人员不一定提高生产率,∵要多花时间去交流,看别人写的代码

**Ch 24 质量管理quality management**

1. QM:一组过程,关注定义如何实现高质量的软件以及开发软件的组织如何知道软件满足所需要的质量要求.既是**组织层面**的重要问题,也是**单个项目层面**的重要问题.**组织层面**与建立能生产高质量软件的组织过程框架和标准有关.**项目层面**包括专门的质量过程的应用,对所规划的过程执行情况的检查,以及确保项目的输出符合项目所适用的标准,为项目制订一个质量计划(给出质量目标,定义用什么样的过程和标准).
2. 质量管理对软件开发过程提供独立的检查,质量管理团队检验项目可交付结果,从而确保它们能够符合组织的标准和目标.还检验软件过程文档,检验是否遗漏重要的任务.QM团队一个独立于开发小组的团队.
3. **质量规划**是为项目制定一个简短的质量计划的过程,列出要达到最重要的软件质量,描述怎样评估这些质量.**质量规划的概要结构**:产品介绍,产品计划,过程描述,质量目标,风险和风险管理.
4. **软件质量**:无法对一个软件系统是否满足规格说明得出客观结论∵写出一个完整的和无歧义的软件需求是相当困难的,软件开发人员和客户可能对于需求有不同的解读，并且可能对软件是否符合规格说明没法达成共识;规格说明通常整合了各类利益相关者的需求,这些需求不可避免地存在着取舍;对某些质量特性(如可维护性)的直接度量是不可能做到的,它们无法以一种无歧义的方式刻画.∴评估软件质量是一个主观的过程.QM存在一个通用的假设:按照需求测试系统.QM团队应该评审所设计的测试并检查测试记录,以核实测试是否被正确地执行.
5. **一个软件系统的主观质量很大部分依赖于非功能性特性**.软件开发过程的质量直接影响软件的质量.软件开发过程中过程质量和产品质量之间的关系更加复杂.评估软件质量的属性非常困难. “质量文化”,产品质量意识
6. **软件标准**:**重要性原因**:智慧的结晶,对组织有重要意义,软件标准封装了对于组织来说最成功的或是最适合的软件开发实践,帮助企业复用以往的经验并避免重犯过去的错误;标准为定义特定环境中的”质量”提供了一个框架,软件质量是主观的,标准的使用为判断软件是否达到要求的质量水平提供了基础;还有助于工作的延续性,确保一个组织中所有的工程人员采用相同的做法,这样开始一项新工作时就节省了学习时间.**两类**:**产品标准**(文档标准;文档编写标准;编码标准),**过程标准**(对规格说明的定义,设计,确认过程,过程支持工具以及对在这些过程中产生的文档的描述).**为了让工程师认可标准的价值**:让软件工程师参与产品标准的选择;定期评审和修改标准以反映技术的变化;尽可能提供工具来支持基于标准的开发,开发人员付出少精力. ISO 9000是在所有行业建立质量管理系统的国际标准集;ISO 9001最具普遍性,适用于设计、开发和产品(包括软件)维护等组织内的质量过程。ISO 9001标准不是软件开发标准而是开发软件标准的框架.
7. **评审与审查**:是检查项目可交付物质量的QA活动;这涉及检查软件,文档以及审查过程的记录,以发现错误和遗漏,还要检查是否遵循质量标准. **评审过程**3**阶段**:评审前活动(评审计划与准备工作),评审会议,评审后活动(解决评审期间提出的问题).分布式评审(共享文档注释评论).**程序审查**:团队成员合作发现已开发程序中的漏洞,不要求执行程序.常使用一份常见编程错误的检查表(数据/控制/输入/输出/接口/存储管理/异常管理故障)
8. **质量管理与敏捷开发**:建立质量文化,团队成员良好实践:提交之前进行检查,绝不破坏构建,发现问题就修复它.Scrum在每次软件迭代完成后有一个评审会议(冲刺评审),关注重构和质量改进.代码评审可能是个人责任也可能依赖于结对编程的使用.结对编程的潜在问题:共同误解,结对声誉,工作关系.当客户是大公司或分布式开发团队或者开发长生命周期的系统(开发团队会换人),采用少使用文档的敏捷方法进行质量管理是不切实际的.
9. **软件度量**:对软件系统的某些属性进行量化. **软件量度**metric:软件系统或过程的属性,可以用数字化的方式表达并且被测量.**过程量度(控制量度**)是过程的属性,例如完成任务的时间,特定过程所需的资源,特定事件的发生次数;**产品量度(预测性量度**)是软件自身的属性,例如规模或复杂度. **use of measurements**:给系统质量属性赋值,通过度量系统构件的特性,并将这些度量综合起来,就能评估系统质量属性;识别质量低于标准的系统构件. 软件的**外部属性**和与其有关的一些**内部属性**存在某些关系,但能够做出有益预测,取决于:内部属性必须被精确度量;在能够度量的属性与我们感兴趣的外部质量属性之间必须有关系,而且这关系还必须是可理解的,可确认的,能用公式或模型表达出来.
10. **系统性的度量比较困难的原因**:引入一个组织量度或软件分析程序的投资回报是无法量化计算的;现在还没有软件量度的标准，也没有标准化的度量和分析过程;度量可能需要开发和维护专门的软件工具,当度量的回报未知时,很难证明工具开发的成本是合理的;许多公司软件过程是非标准化的,没有很好定义而且也很难控制;许多关于软件度量和量度的研究主要侧重于基于代码的量度和计划驱动开发过程,而越来越多的软件开发是复用和配置现有应用系统或使用敏捷方法∴开发者并不知道先前对量度的研究是否适用于这些软件开发技术;引入度量增加了额外开销(不敏捷方法)
11. **产品量度**:量化一个软件系统内部属性的预测性量度.分为:**动态量度**(对执行中的程序进行度量,评估一个程序的效率,可靠性),**静态量度**(对系统各种表现形式如文档,设计,程序等进行度量,评估一个软件系统或系统构件的复杂度,可理解性和可维护性)两类.**静态软件产品量度**:扇入/扇出(调用其他函数的函数数/被调用的函数数),代码长度,环路复杂度,标识符长度,条件嵌套深度,雾指数(文档字和句子平均长度).CK**面向对象量度套件:**每个类的加权方法数(WMC),继承树的深度(DIT),孩子数(NOC),对象类间耦合度(CBO),对类的响应(RFC),方法内聚缺乏度(LCOM).
12. **软件构件分析**:选择要做的度量,选择要评估的构件,度量构件特性,识别异常度量,分析异常构件. **度量歧义**:在收集有关软件和软件过程的量化数据时,必须分析这些数据以了解它们的真正含义,曲解了这些数据就很容易得出错误的结论,不能只了解数据本身还须考虑收集数据的上下文环境.
13. **软件解析**:对于与软件系统相关的动态数据和静态数据进行自动分析以发现这些数据间的关系,这些关系可以提供关于改进软件质量的可能途径的深入洞悉.为项目管理者和开发人员提供规律.**两个因素促使**:软件产品公司的产品在使用时能够自动收集用户数据,如果软件失效,关于失效和系统状态的信息可以通过因特网从用户的计算机发送到由产品开发者运行的服务器上∴可以获得单个产品的大量数据并用于分析;开源软件工程数据的开源存储库,开源软件的源代码可用于自动分析且有时可以链接到开源存储库中的数据. **有效分析工具**:应该易于使用,因为管理者可能不太有分析经验;应该运行快速并产生简洁的输出,而不是产生大量的信息;应该使用尽可能多的参数进行多方面的度量,不可能提前预测出什么规律可能出现;应该是交互式的并允许管理者和开发人员探索分析.不应是可预测的,而应当基于对过去和当前数据的分析为决策提供支持。

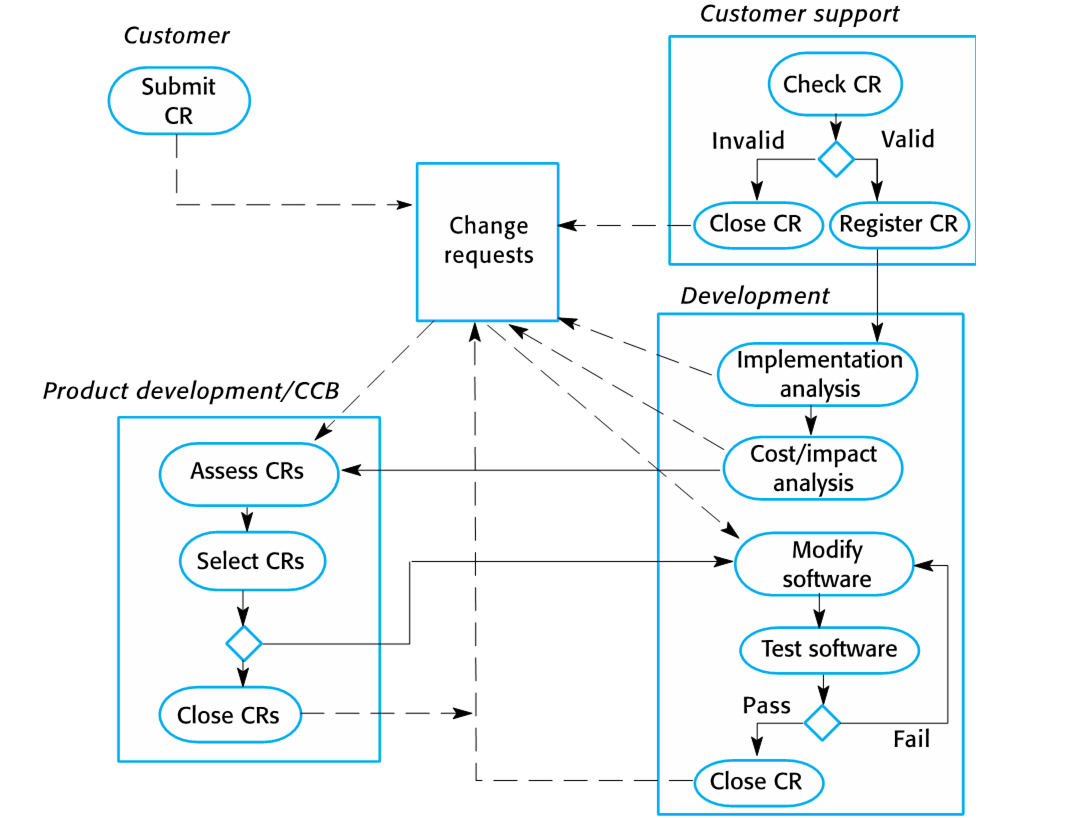
**Ch 25 配置管理configuration management**(版本管理,系统构建,变更管理,发布~)

1. **CM**:管理对演化中的软件产品的变更的过程.与管理变更管理的软件系统的政策,过程和工具有关. 演化中的系统之所以需要管理,是因为系统在演化时会产生许多不同的版本,这些版本包含了变更提议,故障修正以及对不同硬件和操作系统的适应等诸多内容.可能有几个版本同时开发和使用.这样就需要跟踪已经实现的变更以及这些变更是怎样包含在软件产品中的. 配置管理对个人项目管理来说非常有用, 因为开发者容易忘记已做过的变更. 对于多个开发者同时完成一个软件系统的团队项目来说,团队的成员分散在世界的各个角落,配置管理系统的使用确保了团队能够访问正在开发的系统,并且能够管理正在执行的变更.
2. **敏捷开发**中构建和系统每天都要变更多次,构件最终版本存在共享项目存储库.开发人员将它们复制到自己的工作空间,进行修改,使用系统构建工具在自己计算机上创建一个新系统进行测试,如果变更满意,就提交到项目存储库,供大家使用.
3. **软件产品**/**客户软件系统的开发分为**3**阶段**:开发阶段,系统测试阶段,发布阶段.
4. **版本管理**:是持续跟踪软件构件和使用这些构件的系统的不同版本的过程.确保有不同开发者做出的变更不会相互影响.是管理代码线和基线的过程.**代码线**:是软件构件以及构件所依赖的其他配置项的集合.**基线**:组成系统的构件版本加上所使用的库的描述,配置文件和其他系统信息等的集合.**版本控制**:管理对于一个软件系统及其构件变更的过程,其目的是了解构件/系统的每一个版本中进行了哪些修改以及恢复/重建该构件/系统此前的版本.**版本控制**(VC)**系统**:用于支持版本控制过程的软件工具.识别,存储和控制对不同版本构件的访问,分为**集中式和分布式系统**.**主要特征**:版本和发布版本识别(分配唯一标识符和属性);变更历史记录;独立开发;项目支持;存储管理(存差异列表).从项目存储库拷贝(检出,check-out)到私人工作空间处理,完成变更后再返回(检入,check-in)到存储库中.**集中式系统**中,检出的是需要修改的构件文件,检入是不同版本分开存储.分布式系统中,检出是克隆这个项目存储库,**优点**:为存储库提供备份机制,允许离线工作,可以在本机编译和测试整个系统.对开源开发至关重要.**存储管理**:保存增量列表,保存变更行的列表,增量定义了如何重建更早的系统版本.而Git对存储的文件及其相关的元信息采用了**标准压缩算法**,检索文件时只需要解压缩文件,还使用packfiles概念,多个小文件索引为单个文件,减少与许多小文件相关联的开销,packfiles中使用增量进一步减小规模.
5. **系统构建**:编译组成一个系统的构件或单元(外部库,配置文件)并将这些与其他构件相链接以构建一个可执行程序的过程.必须**集成系统构建工具和版本控制工具**,因为构建过程从版本控制系统管理的存储库中获取构件版本.**系统集成和构建工具特征**:构建脚本生成,版本控制系统集成,最小化再编译,可执行系统创建,测试自动化,报告,文档生成.**构建脚本**是对要构建的系统的定义包含构件的信息,构件之间的依赖关系,用于编译和链接系统的工具的版本信息.**构建有**3**种系统平台**:开发系统,构建服务器(构建最终的,可执行的系统版本),目标环境(系统运行的平台).敏捷方法建议使用自动测试执行那些非常频繁的系统构建即**持续集成**.



**持续集成的优缺点**:能够发现并且尽快修复由不同开发人员的交互引起的问题.主线中最近的系统是最终的工作系统.如果系统很庞大,构建和测试可能要花费大量时间,尤其是要与其他应用系统集成时,因此不太可能每天构建系统几次;如果开发平台和目标平台不相同,就不大可能在开发人员的私有工作空间中运行系统测试,硬件,操作系统或者安装的软件都可能存在差异,因此需要更多的时间来测试系统.**解决方案**:每日构建系统daily build:开发机构设定系统构件的交付时间(比如下午2点),如果开发人员有正在编写的构件的新版本,他们必须在下午2点前提交;构件可能并不完整,但是应该提供一些能够测试的基本功能;通过编译和链接这些构件构建系统的新版本,形成完整的系统;将系统交付给测试小组,执行一系列预定义的系统测试;记录系统测试中发现的故障,并交还给系统开发人员,他们在随后的版本中修复这些故障.

1. **编译量最小化**:检测是否存在可用的已编译版本,若存在就不需要重新编译.**源代码构件文件和相对应的目标代码文件联系起来,赋予唯一签名**:两类(修改时间戳,源代码校验和).时间戳通过名字和源代码文件联系起来,只有最新的编译版本可用.校验和方法优点是用文件中数据计算的校验和作为签名,可同时保留构件目标代码的多个不同版本.
2. **变更管理**:对提出的软件系统变更进行记录,检查,分析,估算和实现的过程.**目的**是确保系统演化是一个可控的过程,并且最紧急和成本效益最优的变更拥有最高的优先权.变更管理是对所提议的变更进行成本效益分析,根据成本效益审批变更请求,并且跟踪系统的哪些构件发生变化的过程.Factors in change analysis:**不做变更会引起的后果**;**变更的收益**;**变更影响的用户数**(如果多数系统用户都将重新适应所做的修改,那么实现变更可能是不明智的);**变更的成本**(费时/变更影响许多系统构件(因此增加了引入新错误的机会));**产品发布周期**(如果软件的一个新版本刚刚发布完,那么推迟变更直到计划发布下一个版本,这是有意义的).

变更管理过程

1. **发布管理**:release是分发给客户的系统版本.**两类**:**主要发布版本**(交付重要新功能),**小型发布版本**(修补漏洞和修复用户报告问题).**系统的发布版本包括**:可执行代码,配置文件,数据文件,安装程序,电子和书面文档,包装和相关的宣传.**何时发布的影响因素**:竞争,市场需求,平台变更,系统的技术质量.**发布版本的创建**是创建包含系统发布版本所有构件的文件和文档集合的过程.这个过程包含6阶段:1.程序的可执行代码及所有相关数据文件都必须在版本控制系统中能够被识别,并标记发布标识符;2.为不同的硬件和操作系统编写配置描述;3.为需要配置自己的系统的客户编写更新指令;4.为安装程序编写脚本;5.创建网页来描述发布版本,并链接至系统文档;6.最后,当所有的信息都可用,必须准备该软件的可执行的主映像并移交给客户或销售网点。
2. 对于定制软件或者软件产品线,系统发布管理过程的复杂度与系统用户的数量相关.因此,**一个系统的发布版本生成时,必须编制文档以保证将来可以准确地重新构建它**. 这一点对定制的、生命周期长的嵌入式系统尤为重要. 为发布版本编制文档,必须**记录用来产生可执行代码的源代码构件的特定版本**. 也要**保存源代码文件、相应可执行代码以及所有的数据和配置文件的拷贝.** 还应该**记录用来构建这个软件的操作系统的版本、库的版本、编译器的版本和其他工具的版本**。这些工具在日后构建完全相同的系统时用得上。因此，平台软件和用来创建系统的工具以及目标系统的源代码都要在版本控制系统中留有备份。
3. **提供软件即服务**(SaaS):通过Web浏览器远程访问而不是在本地计算机上安装的软件应用, 越来越多地被用于向最终用户提供应用服务.