CH1

软件的历史

最早：用户与电脑的直接交互，software即instruction together to do something useful

1950：有了programer的概念，high level language出现

1960：产出了一些大型的软件

软件：本身是一个产品（information transformer）

也是用于交付产品的工具

**软件定义**：一组items 和 objects的集合，其中包括instructions（电脑程序，提供所需的功能与表现）、data structures（使程序可以更好的利用操作信息） 和 documents（说明程序的使用）

软件是一种设计与开发，并不是传统意义的创造

软件行业朝着component based（组件构成）方向进展，但现在大多数软件都是定制的

软件应用类型：系统、应用、工程/科学、嵌入式、产品线软件（基于同一套核心技术/平台开发得到的一组产品，存在一定的定制化），网页app，人工智能软件

软件应有的特性：

1. 适应新环境与新技术
2. 可更新以满足新需求
3. 支持可拓展以与新数据库及新系统交互
4. 可重构使其在网络环境中可运行

Webapp：

1. 不只是超文本语言与图像的结合
2. 有XML，java等工具辅助交互
3. 对终端用户或具备独立的处理能力，或是数据库与业务应用的集合
4. 美学性质依旧是webapp的决胜因素
5. 语义网技术：

语义网（Semantic Web）是一种描述性的、基于互联网的相关标准和技术，旨在为人类和机器之间的互操作性提供支持。

核心技术：RDF、OWL、SPARQL（图形查询语言，可用于关系型数据查询，也可以通过查询语句对RDF数据进行检索）

RDF采用了一种三元组（Subject，Predicate，Object）的数据模型来描述资源之间的关系，其中Subject表示资源本身，Predicate表示资源属性或关系，Object则表示资源的值或其他资源。

如 john的年龄是30岁可以表示为：

<http://www.example.org/John><http://www.example.org/age>"30"^^[http://www.w3.org/2001/XMLSchema#integer](http://www.w3.org/2001/XMLSchema" \l "integer)

Mobile application：

1. 基于移动平台
2. 同时具有设备特性与位置属性
3. 在平台内提供持久存储功能
4. 移动应用设备可以直接访问设备上的硬件，已提供本地处理与存储功能

云计算（cloud computing）

1. 提供分布式的数据与算力资源
2. 云外计算资源可以访问云内资源
3. 云计算需要前段与后端的服务机构；前端包括可访问（云）的客户端设备与应用程序；后端包括服务器，数据存储和服务器上相应的应用程序
4. 需要注意对云上私有数据的访问限制

生产线软件：

1. 使用相同的应用程序和数据体系结构进行开发（可重用软件作为核心）
2. 还共享需求、设计模式、测试用例等其他产品
3. 借助产品线中所有产品的通用性进行设计

CH2

软件工程：

软件工程是在软件的开发（development）操作（operation）与维护（maintenane）中应用的系统性的（systematic），有规范的（disciplined）可量化的（quantifiable）的方法。

层次化结构：

Quality focus（最后的实践部分）、process model、methods、tools

Software process（过程框架 --- 框架活动）：

Communication：用户间的协作与需求的整合

Planning：建立工程计划，描述技术风险 列出资源需求，生产何种产品，详定工作日程

Modeling：使用model来便于理解需求与具体设计

Construction（构建）：编写与测试

Deployment（部署）：发布software让用户评价与反馈

软件过程：活动（activity）+动作（action）+任务（task）

过程框架：框架活动（framework activity）+普世性活动（umbrella activity）

普世性活动：工程管理、质量保证、技术回顾/审查、configuration（配置）管理、风险管理、可重用管理

Process model（一般模型）：

总体流程及依赖关系

每一个框架活动中任务动作的细化程度

工作产品的定义和要求

质量保证活动的应用方式

目的跟踪和控制活动

过程描述的详细与严谨程度

客户利益相关者对项目参与的程度

软件团队的自主权

队伍组织角色的明确程度

Software engineering practice（实践的精髓）：

理解问题（利益、问题划分）

计划一个解决方案

实施这个计划

检验最后结果的正确性

Practice相关的通用原则：

为用户提供价值，保持简单（KISS），关注使用者，面向未来，复用（plan ahead of reuse），认真思考

谬误：

软件工程不是机械工作，人数增加未必带来正向收益

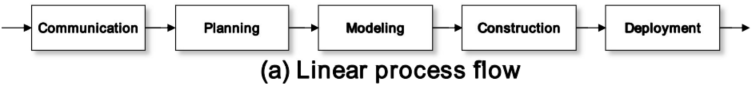
详细说明在一开始就是必须的

CH3

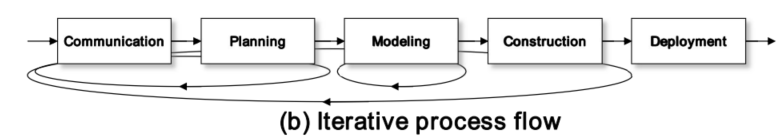
软件工程过程结构：

通用过程模型：

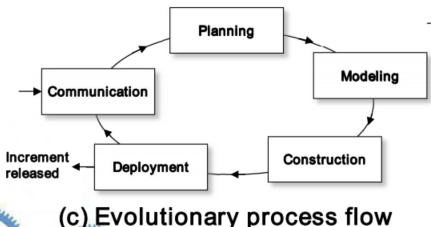
Linear process flow



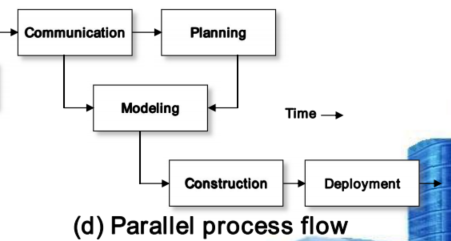
Iterative process flow



Evolution process flow



Parallel process flow



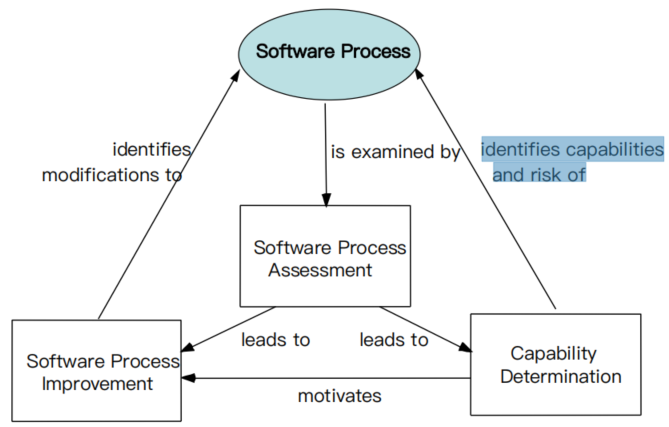
Process pattern（过程模式）

过程模式定义了一系列活动，动作，任务，产品以及相关表现

通用过程模式的元素有：

类型：task pattern（工程动作与工作任务），stage pattern（阶段模式，定义了过程的框架性活动 [有什么框架活动]），phase pattern（框架活动的时序关系）

Process assessment（过程评估）



CMM评估：

Capability Maturity Model for Software，即“软件能力成熟度模型”，是对组织软件过程的描述，核心内容是将软件开发视为一个过程，并且根据相应的原则对于软件开发进行相应的监控和研究。

软件过程评估：SPA

软件过程改进：SPI

软件能力评价：SCE

软件过程成熟度是指一个软件过程被明确定义、管理、度量和控制的有效程度。成熟意味着软件过程能力持续改善的过程，成熟度代表软件过程能力改善的潜力。

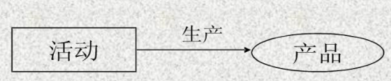
CMMI：能力成熟度模型集成

SCAMPI：用于过程改进的CMMI标椎评估方法

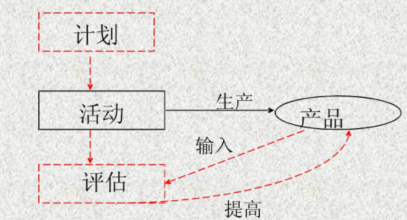
CBA IPI：用于组织内部过程改进的CMM评估

CMM等级结构

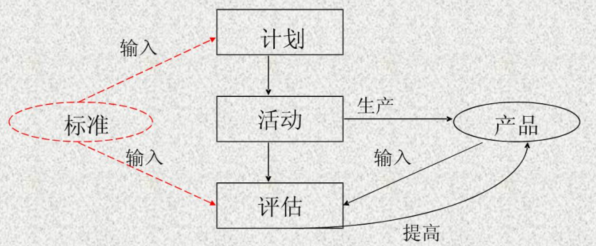
初始级（不可预测与控制）



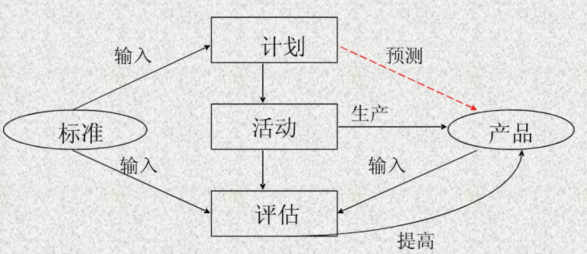
受管理级（重复以前的成功，项目管理体系健全，可重复）



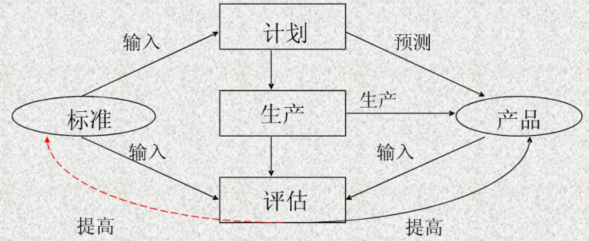
已定义级（过程清晰明确，统一的软件工程和管理过程定义）



定量管理级（过程可度量控制）



持续优化级（过程改进制度化）



CMMI：Capability Maturity Model Integration，能力成熟度模型集成

0级，incomplete，未完成所有确定目标

1级，performed，完成了所需求的工作

2级，managed，利益相关方加入项目，工作被有效监督、审查与评估，人员有权限

3级，defined，管理与工程过程文档化、标准化

4级，quantitatively managed，定量管理，详细衡量

5级，optimizing，优化，根据定量反馈信息进行整体优化

CH4

Process model过程模型

Prescriptive model（约定俗成的model）

Water model

Communication、planning、modeling、construction、deployment顺序进行

真是的project很少使用此模型，因为需求并不是一层不变的

特点：

以上一项活动的产品作为当前活动的输入

给出当前活动的产品，作为下一项活动的输入

对当前活动进行评审，若工作得到确认则继续下一项活动

优点：

强调开发的阶段性

强调早期计划和需求检查

强调产品测试

缺点：

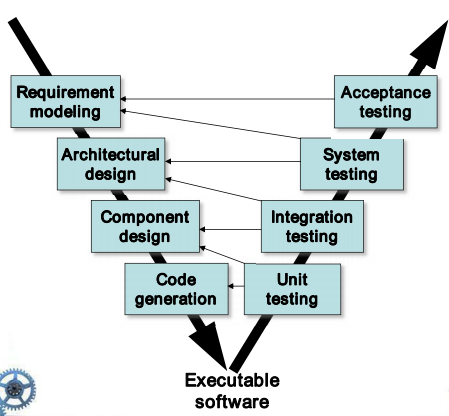
过度依赖早期进行的唯一需求检查，不能适应需求的变化

软件开发是知识密集型开发活动，但water模型是单一模型，开发中的经验不能反馈与当前开发产品中

可能存在错误的积累与放大效应

V-model（Water model延伸版本）

在test过程中对需求确定、模型设计以及code阶段进行反馈



Increment process model（增量过程模型，又称铲平改进模型）

一般增量模型

在第一轮process中先构建核心产品，再后续轮次中再不断扩展特性与功能 --- 迭代的使用water模型

每一代增量均发布一个可操作的产品，后续迭代等价于功能的不断扩展与完善

RAD模型（rapid application development model）

依赖于大量的人力资源

如果系统不能正确地模块化，RAD可能无法工作

强调较短的开发周期（基于可复用构件的方法），不同迭代的water模型均有时间的交集

不适用于较高技术风险的项目，且调试接口欠缺

不适用于要求高性能指标的软件

Evolutionary process model

应用环境：

竞争压力下产品交付时间短暂

需求可能经常性变化

产品核心已定义但扩展细节没有定义

种类：

原型开发：五阶段循环，重点在于快速构建原型以便获取用户反馈和验证需求。

螺旋模型：

在原型基础上，进行多次原型反复并增加风险评估（区别点），形成螺旋模型，具有快速开发越来越完善软件版本的潜力

协同开发模型（Concurrent process model）：

多任务并发，不再是顺序事件序列，而是一个活动网络

由用户需求，管理决策，结果复审共同驱动

优点：

并行化软件开发全过程

强调开发团队的协作性

可用于所有类型的软件开发

增量模型与螺旋模型的区别

增量模型要求软件有明确的需求定义，且增量模型未明确把风险问题考虑进入过程模型。

螺旋模型是过程级迭代，每次迭代均有完整的软件版本；增量模型是活动级迭代

专业的软件开发模型

Component based development 基于组件的开发（本质是演化模型）

Formal method强调需求的数学规范（形式化方法模型）

Aspect-oriented software development（AOSD）面向方法的软件开发（面向方面编程）

横切关注点（cross-cutting concern）指在应用程序中涉及多个模块或组件的功能或关注点（不仅仅属于单一模块）

The unified process（UP）

Unified modeling language（UML）

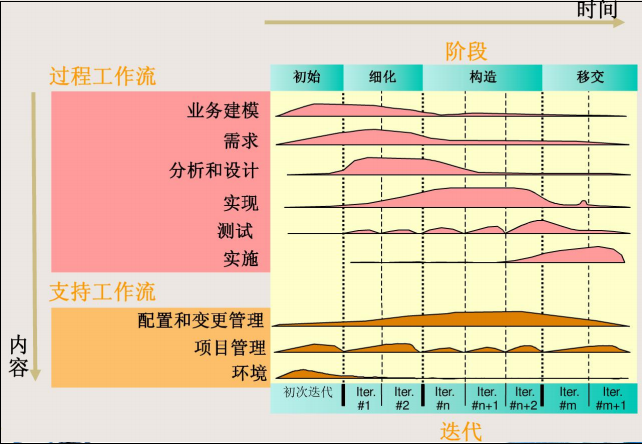
Inception（初始阶段）：communication + planning确定项目的边界

Elaboration（细化阶段）：planning + modeling建立健全体系结构基础，编制项目计划

Construction（构建阶段）：construction 构件、功能的开发并集成为产品（含有test过程）

Transition（移交阶段）：construction + deployment 构建与部署活动（含用户反馈）

Production（生产阶段）：部署活动



UP由两大类核心工作流：核心过程工作流和核心支持工作流

Personal software process（PSP）

Planning

High-level design

High-level design review

Development

Postmortem（后期工作）

Team software process（TSP）

团队需要自主性

激励的测量机制

对测量进行一定的分析以改进团队过程

CH5

Agile development（敏捷开发）

敏捷的表现：

对改变的有效响应

在利益相关方之间有有效的交流

快速的增量式迭代软件

组织一个团队，以控制所执行的工作

敏捷开发过程

以用户对需求的描述为导向

认识到计划是短暂的，在构建软件活动上有重心偏移

适应变化的产生

敏捷开发的原则

最高优先级为满足用户需求并迭代产生有价值的软件

时刻可以相应变化的产生

经常更新迭代软件

利益相关方有良好的交流沟通

Human（team） factors：

决策能力；共同的关注点；协作；困难问题的解决能力；相互信任与尊重

Extreme Programming（XP）XP编程

从用户需求出发，根据需求预期成本；需求实现被划分至不同的迭代时期；

XP design

遵循简单的原则

鼓励使用CRC

对于不同的问题，倡导spike solutions（针对特定技术或问题的短期研究和探索，旨在获取关键信息或验证假设，以帮助团队更好地制定后续的开发计划或决策）

鼓励重构（refactoring）

XP coding

鼓励pair 编程（一人编写一人看）

在编码前构建单元测试库

XP testing

每日完成单元测试

接受性测试由用户定义

Industrial XP（IXP工业级XP）：

IXP有更好的包容性管理，扩展了用户角色，并更新了技术实践

涵盖六个新的实践：

准备阶段评估

项目社区

项目章程（project chartering）

测试导向管理

持续学习

回顾（retrospectives）

敏捷软件开发框架：Scrum

区别特征：

开发工作被划分为packets

测试与文档的在产品构建的过程中的持续改进

Dynamic systems development method（DSDM）

旨在提供一种框架，帮助团队快速交付高质量的软件系统。DSDM 着重于在软件开发项目中实现时间、成本和质量的平衡，同时强调与业务需求的紧密对齐。

与XP类似，有九条指导性准则：

必须授权DSDM team来决策

关注与频繁的推出产品

验收标准是适合商业目的

开发过程中的所有变化都是可逆的

测试贯穿整个生命周期

敏捷model 原则

带有目标的进行模式化

使用多种模式相结合的方式

内容本身比表现形式重要

敏捷unified process（统一流程，AUP）

Modeling

Implementation

Testing

Deployment

Configuration and project management

Environment management

CH6

Human aspects of software engineering

软件工程师的特性

Individual responsibility责任心

理解用户

勇于承认错误并提供建设性意见（constructive criticism）

注重细节

Resilient under pressure（较好的抗压能力）

务实（pragmatic）

The psychology of software engineering（软件工程心理学）

软件实现团队

高效团队的特性：

目的性、不断改进、自我进化、相互信任、团队技能多样性

团队结构的影响因素：

问题的难易程度

需要实现程序的大小

团队的组建时间

问题可被模块化的程度

产品交付日期

质量与可依赖度需求

组织范式：

Closed paradigm传统的团队，有权威等级观念

Random paradigm相对松散的团队，取决于个人主动性

Open paradigm既有管理又倡导鼓励活力创造力

Synchronous paradigm（同步范式）依赖于问题的自然划分，分为不同的小组解决问题的不同模块；不同小组之间仅有较少的积极沟通

PM项目经理，QA质量保障师，TTL开发经理，SE软件工程师，TE测试工程师

敏捷软件生产团队

强调个人能力和团队协作能力

需要有较好的激励政策

多种团队结构的融合，有较强的自主性

敏捷方法的基本观点：

强调适应性而不是可预测性

强调人在项目中的关键作用

XP team value

Communication 非正式深度交流

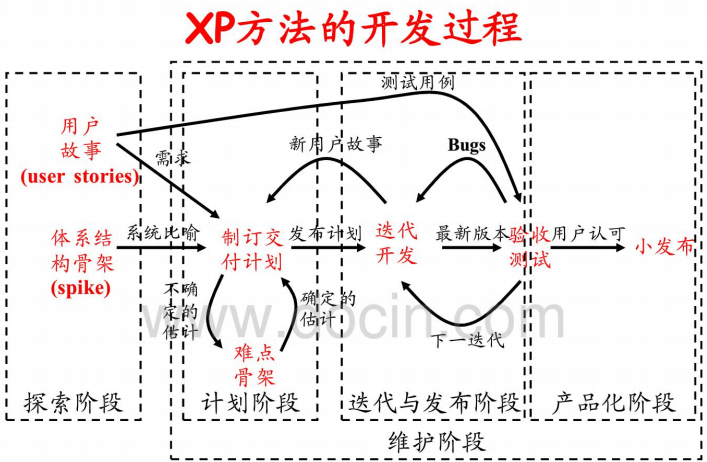
Simplicity（简单）仅面对当下的需求

Feedback

Courage 主要是抗压

Respect相互尊重

XP开发过程



探索阶段：开发初始的用户需求和体系结构骨架；根据用户需求编制相应的测试用例

计划阶段：制定迭代计划和发布计划

迭代到发布阶段：开发软件模块并完成集成，进行验收测试，在提出新需求时，接受变化并回到计划阶段

产品化阶段：进行更多的测试并做好进入产品化的准备；整理必要的文档

维护阶段：系统的运行与支持

Social media的影响

Blogs分享team成员与用户间的信息

Microblogs 向关注的人群发布实时信息

Targeted on-line forums（座谈会）提问与解答相关

Social networking sites 允许软件开发人员之间的连接，以共享信息

Social book marking 允许开发人员跟踪和共享基于web的资源

软件开发过程对“云”的使用

便于所有产品访问权限的控制

克服设备依赖性

一人更改，全员可见

可能带来可靠性与安全性的风险

可能带来冲突性的操作

“云”的可用性与性能往往需要与安全性、隐私与依赖性相平衡

协作开发环境服务Services of collaborative development environments(CDEs)

Namespace（安全性、隐私、工作空间）、calendar（工程协作日历）、量化质保（metrics）支持、communication analysis

Global team团队决策的复杂性

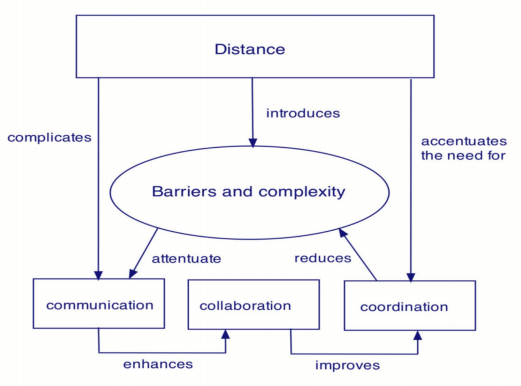
问题复杂

决策时不确定的因素与风险

当前决策可能会影响另一工程

不同的视角导向不同的决策

全球软件团队面临着与协作、协调和协调困难相关的其他挑战



CH7

Principle that guide practice指导实践的原则

软件工程知识可能有半衰期，但软件工程原则可以受益终身

指导process的原则

保持敏捷

每一步都关注质量

做好适应的准备

组件高效的团队

建立交流与协调机制

项目的变更管理

评估风险

创建有价值的工作成果

指导practice的原则

分模块解决问题，关注点分离（separation of concerns，SoC）

使用抽象

保持一致性

关注信息的传递（对界面的分析、设计、构建、测试给予特殊关注）

使用模块化的方法

寻找已有的模式

从不同角度表达解决问题

软件需要有人维护

Communication 原则

倾听、prepare for communication、需要有人主导、面对面交流是最好的、做记录写文档、合作高效、保持讨论问题的集中性、解释不清时请画图、

Planning 原则

划定工程的范围

将用户邀请进入计划活动中（定义优先级与实现限制）

构建迭代性的计划

在计划时即考虑风险

计划需要现实（劳逸结合的考虑）

制定计划时调整粒度（granularity）

计划如何保证产品质量

计划如何适应变化

Modeling原则

建立需求模型（分析模型）

Information 角度、functional 角度、behavioral 角度

建立设计模型

敏捷模型原则：

目标是实现软件而不是建立model

Model建立简单化

建立方便修改的model

明确每个建立model的目的

Model追求有用而不是完美

尽快得到反馈

需求模型原则：

Information层面的问题必须被表示与理解

软件的function必须被定义

软件的behavior必须被表达

以层次化的方式展现细节

设计模型原则：

设计应跟随需求模型

设计时需关照系统的结构

设计数据类与设计功能同样重要

界面的设计一定要满足需求

构件级设计需保证功能独立性

模型的设计需要利于理解

设计也需要不断开发迭代

Living modeling原则（活模型，持续更新和演进的建模方法）

需关注特定利益相关者及任务

模型与代码需要紧密耦合

模型与代码之间应建立双向数据流

模型信息应该持久化且允许系统修改

每个模型元素均融入利益相关方的权利与责任

Construction原则

Construction过程包括了code与test，使得软件可以交付给用户使用

编码原则涉及编程分割、编程语言与编程方法

测试原则是以最小的时间和代价揭示系统不同种类的错误

Preparation 原则（在编写代码前）

理解需要解决的问题

了解基本的设计原则与概念

设定编程的语言与环境

建立一组用于测试目标编写组件的用例

Code 原则

考虑使用结队编程

构建贴合数据的数据结构

理解软件结构并建立合适的界面

保持逻辑判断尽可能简单

创建利于测试的嵌套循环

Validation 原则（检验原则）code结束之后

在恰当的时候走读回顾代码

继续修改直至test全部通过且error被cover

重构代码

Testing原则

所有测试必须满足用户提出的可追溯性需求

测试应在test阶段前被规划、编写

软件测试遵循pareto 原则（二八原则）

测试应从小入口切入，逐渐扩展至全局

全面的测试是不可能的

测试投入需要与预期default密度相称

Static测试也是一种高效的测试方法（不执行程序，对软件和文档进行审查）

Deployment 原则

必须解决客户对软件的期望

完整的交付包应被组装与测试

必须在软件交付前建立支撑制度

需要给最终用户提供适当的指导性资料

有bug的软件应先修复，再交付

CH8

理解需求

需求工程

Inception起始

了解问题的基本

所期望的解决方案的性质

开发者和用户之间初步沟通的有效性

清楚各利益相关方

组织对软件的不同视角与观点

标识公共区域和矛盾区域（以及非功能性需求）

Non-functional requirements（非功能性需求NFR）

质量属性、性能属性、安全属性、一般系统约束

判断NFR是否兼容的两个阶段：

建立一个矩阵，每一个NFR作为列标题，系统SE指南作为行

使用一组决策规则来定义每个NFR的优先级，通过将每个NFR和指南对分类为互补、重叠、冲突或独立来判断实现兼容性

Elicitation 获取

所有利益相关方的需求

组织软件工程师和用户一起参加会议

建立准备与参与软件实现相关的规则

需要引导者和相关议程的准备

最重要的部分：

框定问题的范围

理解问题

问题的不断变化

需要获取一组初步的解决方案

Quality function deployment（质量功能部署）

功能部署决定了系统所需的每一个功能的价值

信息部署标识了数据对象和事件

任务部署测试了系统的表现

价值分析定义了相关任务的需求优先级

由此定义三种类别的需求

Normal 需求

Expected 需求

Exciting 需求

该过程的产出：

对需求的可行性（feasibility）描述

对最终系统和产品的范围性框定

对用户、各方利益者的统计

描述系统的技术环境

需求列表及主要约束

一些必要的原型设计

在不同操作环境下对系统产品使用的洞察

Elaboration 细化（主要是用例）

建立分析模型，细化数据、功能、行为需求

收集描述系统可用性的用例，该用例需要站在用户或设备的视角上

每个用例都要辨清主要与次要角色，以及他们对应的目标；这些角色的主要任务与功能优势什么；角色间互动是否可以增添一些变化；角色与哪些系统信息会产生获取、生产或修改等操作；角色是否期望被告知一些意外的变化

用例图的使用（角色、事件、工具间的交互关系）、UML活动图

建立分析模型

Scenario-based 基于场景的元素

用例与用例图

在特定语境下的活动序列

Class-based 基于类的元素

类图（类似于E-R图，类名 + 属性 + 函数名的形式）

Behavioral 基于行为的元素

状态元素 （类似于类图，由状态名 + 状态变量 + 状态活动构成）

状态图，状态元素的集合（箭头线条注明状态间的变化趋势）

分析pattern步骤：pattern name、intent（大体对pattern进行描述）、motivation（解决什么问题）、可能影响pattern的上下文环境、pattern提出的解决方案/结构与行为、pattern带来的影响以及平衡、pattern应该如何通过已设计pattern进行设计、在系统中应该如何应用这个pattern、与当前pattern相关的信息

Negotiation 协商

商定一个切实可行的交付系统

标志关键的利益相关方

这些利益相关方需要参与谈判

确定每个利益相关方的期望条件

期望条件总不相同

协商

得到双赢的统一需求方案

Requirements monitoring（需求监控）

在增量模型中特别需要

Distributed debugging分布式调试，发现错误并确定原因

Run-time verification 运行时验证，看软件是否符合规格

Run-time validation 运行时估计，判断软件是否满足了用户需求

Business activity monitoring业务活动监控，评估系统是否满足业务需求

Evolution and co-design 共同设计与进化，向利益相关方提供信息以追求不断改进

Specification 规格说明

文档、一组模型、原型设计、收集的场景用例、正式的数学表达

Validation 确认

寻找内容与解释中的错误

标记可能需要更清楚表述的区域

填补缺失的信息

冲突或无法实现的需求说明

不一致性内容解决

Requirements management需求管理

未完待续

CH13

Architectural design体系结构设计

可分析设计在满足需求方面的有效性

降低软件构建的风险

在设计改变相对简单时考虑架构的备选项

体系结构是什么：沟通载体（不同利益方之间）、（前期决策对产品的成功）影响重大、整体表示（组件的工作方式）

体系结构是：architectural description（AD）是一组体现系统不同视图的工作产品

描述规范：

建立概念性框架和词汇表

提供体系架构描述准则

鼓励良好的设计实践

体系结构类型

体系结构风格：

考虑组件（数据库，计算模型）、连接器（辅助组件间交流协作）、约束条件（组件间交互状态）、语义模型（分析系统组成的已知属性来了解系统的整体属性）

数据中心架构，组件以数据库为核心

数据流架构（数据流定向流动）

Pipes and filters：管道和过滤器，过滤器仅需独立考虑数据流的转换 --- 高并发高吞吐量，但不适合动态交互（一旦数据到达即开始操作）

Batch sequential：顺序批处理

Call and return：类树结构，主程序调用其他程序进行系统的运行；有主程序/子程序体系结构，远程过程调用体系结构

Depth调用的深度

Width调用的广度（每一层的最大个数）

Fan-out 一个程序调用其他程序的最大个数

Fan-in 调用同一程序的最大其他程序个数

Object-oriented面向对象的架构

Layered架构：层级架构，从core逐渐向外扩展，tool层，应用层，用户接口层

Service-oriented架构：服务导向架构，定义多个服务接口供软件组件调用

Web service三要素：SOAP描述传递信息的格式、WSDL基于XML描述Web service、UDDI管理分发查询web service

体系结构模式：

并发性（concurrency）

操作系统流程管理模式

任务调度管理模式

数据持久性（persistence）

数据库的数据保留持久性及较强的检索能力

应用级数据持久性

分布式（distribution）

体系结构设计：

软件必须关注上下文

上级系统、下级系统、同级系统、参与者（用户，提供数据）

需要定义一组架构相关的原类型（系统行为的类抽象）

通过定义实现每一个架构原型来整合得到系统架构

组件间结构

体系结构需要考虑

Economy（简洁、经济），visibility（系统结构的设计及原因应该较为明显），spacing（分离问题，不存留隐藏关联），symmetry（对称性，功能设计一致而平衡），emergence（可解决出现的紧急问题）

体系结构复杂性（组件依赖关系的评估）：

共享依赖性，使用相同资源的消费者或生产者之间

流依赖性，消费者和生产者之间

受约束依赖性，一组活动之间对相对流的约束

ADL（architectural description language）：体系结构描述语言

便于分解架构组件，将组件组合成块，定义组件间接口

系统架构评估（review）：

满足质量要求并发现潜在风险的能力

早期解决潜在问题降低成本的能力

通常使用基于经验的评审、原型评估、场景评审和检查清单

基于模式的结构评估：

用样例判断质量属性

讨论系统架构图与需求间关系

使用现有的文档和用例来确定每个模式对质量属性的影响

识别由在设计中使用的体系结构模式所引起的所有质量问题

CH14

构建级设计

组件的定义：系统的一个模块化（modular）、可部署（deployable）和可替换（replaceable）的部分，它封装实现并公开一组接口

OO（面向对象）组件：组件包括了一组协作的class

Conventional观点：组件含有一定的处理逻辑，且附带了这段处理逻辑需要的数据结构，并为外部提供了调用的接口

Basic design原则

Open-closed 原则（OCP）：对扩展保持开放，对修改保持关闭

Liskov substitution 原则（LSP）：子类可以被替换为基类

Dependency inversion原则（DIP）：依赖于抽象

Interface segregation原则（ISP）：多个特定的功能接口优于一个通用功能接口

Release reuse equivalency原则（REP）：重用的粒度应与发布的粒度相匹配

Common closure原则（CCP）：一起变化的类应该组合（组建）在一起

Common reuse原则（CRP）：不会一起重用的类尽量不组建在一起

Design指导

组件：命名需要符合体系结构的规范

接口：需要提供有关沟通和协作的重要信息

依赖关系与集成：从左至右建立依赖关系，从底部至顶部建立继承关系

Cohesion（内聚性）

传统view：一个模块的专注性

面向对象view：组件或类只封装彼此之间与类或组件本身密切相关的属性和操作

内聚的level：

Functional 功能内聚

Layer 元素按照层次结构排列，高层元素调用底层元素

Communicational（通讯内聚）元素共享相同的输入数据或输出数据

Sequential（顺序内聚）前一个元素的输出作为后一个元素的输入

Procedural（过程内聚） 模块中的元素按照一定的顺序组成一个完整的过程或算法

Temporal（时间内聚） 元素在同一个时间段内执行

Utility（工具内聚）逻辑上没有明显关联，仅仅因为它们在同一个模块中被组合在一起

Coupling耦合性

传统view：一个组件与其他组件/外部环境的联系程度

面向对象view：类之间关联的定性度量

Level：

Content（内容耦合）一个模块直接访问、读取或修改另一个模块的内部数据/状态

Common（公共耦合）多个模块共享某个全局变量、数据结构或资源

Control（控制耦合）一个模块通过传递控制信息（如条件、标志等）来影响另一个模块的执行流程

Stamp（标记耦合）一个模块向另一个模块传递标记、标志或参数来影响其行为

Data（数据耦合）一个模块向另一个模块传递数据

Routine call（调用耦合）一个模块调用另一个模块的函数或方法

Type use（类型使用耦合）一个模块使用另一个模块定义的数据类型或接口

Inclusion or import（包含、导入耦合）一个模块包含、导入另一个模块的代码、文件

External（外部耦合）一个模块依赖于外部环境或其他系统的输入或输出

组件level design过程

确定与问题域对应的所有设计类

确定与基础结构域对应的所有设计类

详细阐述非reuse的组件class设计

设定class与组件协作时的消息细节

为每一组件设计合适的接口

设计详细的属性、数据类型及数据结构

描述处理流程

描述持久性数据源（数据库和文件），并确定管理它们所需的类

为一个类或组件开发和详细阐述其行为表征

Collaboration diagram协作图，描述对象之间的协作关系与消息流

Activity diagram 活动图，用于描述系统中活动流程和操作顺序的图形化工具

Statechart状态图，描述对象在其生命周期中状态变化和转移的图形化工具

Web app中的组件设计

Web app组件是被较好定义的操控内容，进行数据计算或操作的函数

内容与功能的内聚工具包，为终端用户解决需求

Content design：

对于特定内容对象，进行打包并提供给终端用户

Functional design：

本地处理以生成内容和导航能力

提供数据处理与计算的能力

提供复杂的数据库访问能力

建立与外界环境的数据交互接口

较为简练的web-based客户端：

仅在设备商有接口

数据实现借助与云服务

较为强大的客户端：

Data、business、interface三层均实现了接口

往往会受到移动设备的限制

Design传统组件

处理逻辑设计遵循算法设计和结构化编程原则。

Data structure由data model定义实现

Interfaces 由协作控制

基于组件的实现（CBSE）

考虑已有商用组件、内部开发组件是否可满足需求

考虑可用组件与当前架构是否兼容

复用的限制：

没有复用计划

很少使用复用组件

应用复用组件的培训较少

分为domain engineering和software engineering

Domain engineer：

定义要调查实现的范围

分析该领域中具有代表性的程序

建立一个对映的分析模型

需要建立组件库

组件间需要有一致的结构

CBSE activities

组件确认（qualification）

使用提供的API

组件所需的开发和集成工具

运行时间需求（包括资源利用及网络协议）

服务需求（操作系统接口及其他组件的支持）

安全性（身份验证等）

异常处理

组件适应（adaptation）

一致的资源管理方法

相同的数据管理等活动

体系结构内部与外部环境接口保持一致

组件组成（composition）

基础（infrastructure）组件的建立

往往需要包含的组件：

数据转化model

自动化

结构存储

基础对象模型

组件更新（update）

CBSE

1.OMG/CORBA

对象管理组（object management group）已发布了一个公共对象请求代理体系结构（common object request broker architecture）（OMG/CORBA）

对象请求代理（ORB）提供的服务使可重用组件（对象）能够与其他组件通信，而不考虑其他组件在系统中的位置

如果为每个组件创建一个接口定义语言（IDL）接口，则可以确保系统中CORBA组件的集成（无需修改）

客户机应用程序中的对象从ORB服务器请求一个或多个服务。请求是通过IDL或在运行时动态发出的

接口存储库包含有关服务的请求和响应格式的所有必要信息

1. Microsoft COM

组件对象模型（COM）提供了一个规范，在Windows操作系统下运行的单个应用程序中使用不同供应商生成的组件。

COM需要COM特定的接口以及在COM接口之间注册和传递消息的机制

1. Sun JavaBeans

JavaBeans组件系统是一个可移植的、独立于平台的CBSE基础设施

含有Bean工具开发包（Bean Development Kit BDK）

分析现有的Beans work

自定义他们的行为和外观

为交流与协调建立机制

测试和评估Bean的行为

分类（classification）

Enumerated classification 枚举分类 组件层级定义设计，注明了软件组件的类和不同级别的子类

Faceted 分类（逐面分类）分析一个领域区域，并确定了一组基本的描述性特征

Attribute-value分类 （属性值分类）为域区域中的所有组件定义了一组属性

复用环境

可以存储与检索组件的组件数据库

Library管理系统

软件组件检索系统（retrieval system）

CBSE tools 支持将可重用的组件集成到新的设计或实现中

CH15

用户界面设计

Golden rules

让用户自主控制

交互模式减少强制性

提供灵活的互动

允许交互可以中断与撤回

对用户隐藏技术细节

简化交互并允许定制交互

减少用户的内存负荷

减少对短期内存的需求

建立有意义的默认值

使用直观的快捷方式

界面的视觉布局应该基于一个真实世界的隐喻

以渐进的方式展示信息

使界面保持一致性

允许用户将当前的任务放到一个有意义的上下文中

在一个应用程序家族之间保持一致性

保持原有满足用户期待的功能直到不得已进行修改

用户界面设计 model

User model 该系统的所有最终用户的配置文件

Design model 一种user model的设计实现

Mental model（构思模型，系统感知）用户期待界面模样

Implementation model 界面的外观和感知，以及相应的描述语义与语法

用户界面设计过程四步

Interface design

Interface construction

Interface validation

Interface analysis and modeling

了解使用交互界面的用户

了解用户依赖界面实现的任务

了解界面需要呈现的信息

了解任务进行运算依赖的环境

用户分析

任务分析及模型

Use-cases define basic interaction 用例定义基本交互方式

Task elaboration refines interactive tasks 任务细化细化了交互任务

Object elaboration identifies interface objects (classes) 对象细化设计了接口对象

Workflow analysis defines how a work process is completed when several people (and roles) are involved工作流分析定义了多个角色参与时工作的完成流程

泳道图（swimlane diagram） 展示多个参与者或角色在完成某个流程或任务时的交互和责任分配。泳道图将流程中的各个步骤或活动分配到不同的泳道（swimlane）中，每个泳道代表一个参与者或责任方，从而清晰地展示各个参与者在流程中的角色和职责。

Display content分析（显示内容分析）

用户是否支持自定义屏幕内容

是否为屏幕上所有内容分配了合理的空间

Error如何显示

Color显示如何规划

图形输出是否会因为设备不同而被压缩

界面设计步骤

使用界面分析中的信息，进行界面对象与动作的设计

设计用户可触发的引导界面变化的事件

描述每一个会呈现在用户面前的界面状态

指示用户如何通过界面提供的信息来解释系统的状态

界面设计时需要考虑的因素

响应时间

帮助工具

错误处理

菜单与命令标签

程序的可访问性

Internationalization（国际化）

Web app和 mobile app的界面设计

通过界面提示用户当前所处的内容层级位置

并直白的提供用户可以进行的操作（功能、链接等）

界面需要便于导航

高效的界面设计

视觉上是明显且宽容的，直截的向用户提供控制感（如何操作，如何跳转等等）

界面实现与用户的系统的内部工作无关，用户工作被仔细和不断地保存，用户可以在任何时候撤销任何活动

在用户提供最少的信息的前提下进行最大限度的工作与服务

界面设计原则（详见CH17）

界面设计workflow

Review分析阶段的信息并细化需求

粗略设计app的界面草图

将用户的需求动作映射到界面的操作中

定义一组与每个操作相关联的用户任务

Storyboard（类似于UI布局的屏幕图像或屏幕布局图）

在层级中加入美学设计

开发用户与界面进行交互的过程表示

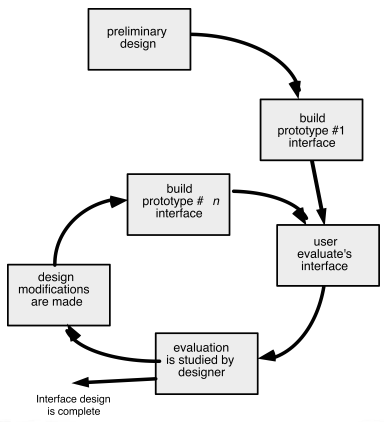
开发界面的行为表示

描述界面的层级状态

完善并审核界面设计模型

美学设计（详见CH17）

设计evaluation cycle



CH16

设计模式

Design pattern：展现特定背景（上下文）、问题与解答关系的规则

Context：问题产生的环境；在此环境下何种解决方法可能有效

A system of forces：一组包含限制、约束和影响的需求

如何在上下问中解释这个问题、解答如何被高效的实施

高效的pattern（要求）

真实解决了问题

经过验证

问题的解决方案并不显然

不要仅仅描述模块，还要描述更深层次的系统结构和机制

模式有重要的人为组成因素

**生成**模式种类

Architectural pattern 架构模式，使用结构性方法解决基于广泛的设计问题

Data pattern

Component pattern（design pattern）解决子系统与组件的开发问题；组件间的相互交互以及在更大架构中的相对位置

Interface design pattern描述常见的界面问题与解决方案

Web app pattern建造web app时会遇到的问题，并且经常包含刚才提到的许多其他模式类别

模式种类

Creational pattern，关注于 creation、composition（组合）和representation（表现）

Abstract factory pattern集中决定实现何种factory

Factory method pattern 集中创建一个特定类型的对象，并选择几种实现中的一种

Structural pattern 关注类和对象如何组织和集成以构建更大的结构的问题和解决方案

Adapter pattern 将类的一个接口调整为客户端期望的接口

Aggregate pattern具有子元素聚合方法的复合模式的一个版本

Behavioral pattern关注对象之间的责任分配和在对象之间进行沟通的方式

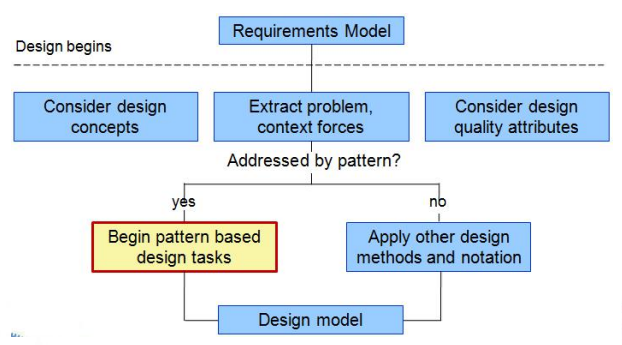
Chain of responsibility pattern 命令对象通过包含逻辑的处理对象来处理或传递给其他对象

Command pattern 命令对象封装了一个操作及其参数

Frameworks框架

框架是可以复用的“微型体系结构，可以作为应用其他设计模式的基础”；不是完全的建筑模式，而是一个带有“插头点”集合的骨架

Pattern-based 软件设计



Thinking in pattern

确保了解蓝图，软件建立的环境

建立抽象层次上的模式

建立背景或未来设计工作的骨架

由外部转向内部，寻找有助于设计解决方案的较低抽象层次的模式

具体细节优化设计

Design tasks

common design mistakes

没有花费足够的时间探寻问题，导致选择的pattern不是最恰当的

一旦pattern选中，只会强迫去适应而不加改变

Architectural patterns

体系结构模式解决了诸如并发性、持久性和分发性等问题

Component-Level Patterns

组件级设计模式提供了一个经过验证的解决方案，可以解决从需求模型中提取的一个或多个子问题

User interface（UI）pattern

Design focus：

Whole UI 顶部结构和导航设计

Page layout 页面的一般组织显示以及不同屏幕的交互显示

Forms and input 考虑表单级输入

Tables 为建立和控制表格数据提供设计指导

Direct data and manipulation 关注数据的直接编辑与调整

Navigation 使用多级菜单协助用户导航

Searching 提供基于特定信息检索持久性存储数据库的能力

Page elements 实现web app中特定的元素

e-commerce 电子商务应用

web app pattern

信息结构（information architecture）关于信息空间的结构以及用于与信息进行交互的途径

导航pattern 关于导航链接结构

Interaction pattern（交互模式）关于用户可进行的动作序列集，如何借助界面为用户展示必要的信息（链接的目的地，当前活动的状态，界面相关提示等）

Presentation pattern（表现模式）内容借助界面向用户表现的模式，如何组织用户界面控制来提供更好的利用率；如何表现动作与内容的关系，如何建立高效的内容层次结构

Functional pattern（功能模式）定义系统工作流，表现，处理过程，逻辑交互等信息

Mobile app pattern

签到、地图、弹窗、注册流程、导航、邀请等

活跃对象、应用控制、数据传递对象、实体翻译、可靠会话、同步问题

CH17

两种基本的设计思路：

表达自己的艺术理想

为客户解决问题的工程理想

Web app 质量

Security 安全性

抵御外部攻击

拒绝非认证访问

确保使用者隐私

Availability 可用性

App 可供使用的时间占比

Scalability 可扩展性

是否兼容接口的较大变化

Time to Market 投入市场的时间

对于终端用户而言的webapp质量维度

Time 网页升级的改变信息量；如何突出显示网页发生的变化

Structural （结构上的）网页各部分如何组织在一起；链接跳转是否工作；图像是否显示；web的所有网页是否均可连接

Content（内容）关键内容是否在合适的位置；关键内容是否具有时效性；动态生成网页相关；

Accuracy and Consistency （准确度和一致性）

Response Time and Latency（响应时间和延迟）

Performance（性能）网页与浏览器连接是否迅速；性能随（一天中）时间、负载、使用情况的变化；

Web app的设计目标

Consistency 一致性

内容保持一致

美学设计保持一致

体系结构设计模板化（超媒体系统保持一致性，超媒体系统是一种基于超文本和多媒体技术的信息系统，它允许用户通过非线性的方式浏览和检索信息。其扩展了传统的文本链接概念，通过在文本中嵌入超链接和多媒体内容，使用户可以按照自己的兴趣和需求跳转到相关的信息资源。）

接口设计，统一的交互、导航、内容显示模式

导航模式一致性

Identity 身份

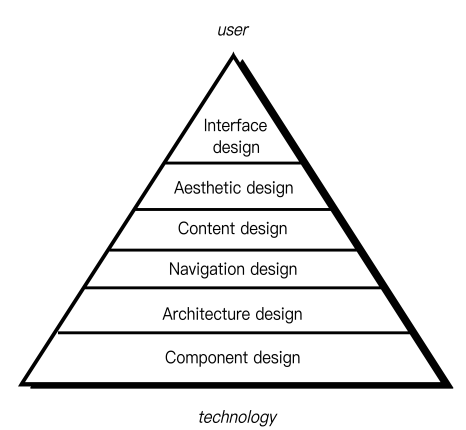
Robustness 鲁棒性，用户基本需求功能在面临异常、攻击等时的服务提供质量保持稳定、高效的能力。

Navigability 导航的设计需要保持intuitive（直观，不用额外学习即可使用）与predictable（可预测，预测系统运行结果）

Visual appeal 视觉吸引力

Compatibility 兼容性，尽可能满足各类环境与配置

Web app设计金字塔



Interface design principles

Anticipation预期，web app应该可以预期用户的下一步操作

Communication展现用户进行活动的状态

Consistency控件、菜单、美学等设计需保持一致

Controlled autonomy控制自主权，界面移动跳转自由，但需要遵守web app设定的规范

Efficiency优化用户使用功能的效率，而不是关注设计效率和CS环境执行效率

Focus专注于用户的功能需求，而不是具体的技术实现

Fitt’s law 获取目标的时间是到目标的距离和其大小的函数

Human interface objects人机界面对象，可以建立相应的对象库

Latency reduction通过多任务的方法，使交互感觉像已完成了一些先前发布的指令（即使正在运行中）

Learnability界面设计需要减少使用界面的学习时间

Maintain work product integrity维护产品完整性，error产生时的驻留不丢失

Readability页面展现的信息需要为尽可能多的群体所阅读

Track state记录用户交互的历史，以便可以追溯

Visible navigation可见的导航，让用户一直以为它在同一个地方

Aesthetic design美学设计

White space很重要

标注重点内容

从左上至右下布局

页面内应该包含分组导航、内容以及功能

避免过度滚动条来扩展可展示的内容

设计时考虑分辨率与窗口大小

Content design（内容设计）

对内容对象设计一个表示的方法

实例化表示不同组件/类之间的关系

不同内容对象应有其特定的展示内容和用于实现的特定属性

Architecture design

Content architecture关注于内容、网页的呈现以及导航方面的结构

Web app architecture Web app程序架构涉及应用程序如何设计和构建，以管理用户交互、处理内部任务、实现导航，并呈现内容

体系结构design往往与界面design，美学design和内容design并行

MVC architecture（Model-View-Controller架构）

Model 包含了所有特定内容以及处理逻辑（数据流）

所有的内容对象

对外部数据/信息的资源访问

所有特定应用程序的处理功能

View包含界面特定的功能

内容表示和处理逻辑

终端用户所需的处理函数

Controller

控制器管理对模型和视图的访问，并协调它们之间的数据流

Navigation design

考虑用户的层次结构和相关用例（不同的常用功能需求可能对映不同的导航需求）

Navigation semantic units（NSUs 导航语义单元）：控制器管理对模型和视图的访问，并协调它们之间的数据流。

WoN（ways of navigation）使用户实现特定需求目标的最好的导航方式

Navigation node（NN）组成，由navigation link连接

Navigation syntax

Individual navigation link 个人导航链接组件

Horizontal navigation bar水平导航栏

Vertical navigation column 垂直导航栏

Tabs在需要链接时选择内容或功能类别

Site maps（站点地图）提供全包式的内容标签，以便导航至web app中包含的所有内容对象与功能

Component-level design组件设计

通过本地处理，以动态的方式生成内容和导航

提供满足需求的数据计算能力

提供复杂的数据库访问与获取数据能力

建立与外部系统的数据接口

CH18

Mobile app design

需要考虑的点

更多维的硬件与软件平台

更多的框架与编程语言

更多的app store，各有自己的技术需求与设计准则

更短的开发周期

用户界面大小有限制

复杂的传感器、电量管理、安全与隐私、设备限制

移动应用开发模型

Formulation

Planning

Analysis

Engineering

Implementation and testing

User evaluation

Mobile app 质量检测

内容功能的用户喜好设置、基于通信速率定制（以及app在低通信速率下的表现形式）

根据用户偏好生成上下文

是否完备考虑了power可用性以及目标设备

是否恰当的使用了云、媒体、图片等服务

App总页数是否便于阅读与导航

是否考虑设备间屏幕大小不同、用户界面与设备交互标椎是否具有一致性

App是否可依赖、够安全并且保护个人隐私

是否在所有目标环境下测试mobileapp

Mobile app 用户界面设计considerations

用户界面的品牌特征

关注各类产品的结合

优化UI流和UI元素

定义缩放规则

创建用户性能仪表板

Mobile app的设计谬误

不一致性

过度设计

响应速度匮乏

大量冗余

非标准的交互

杂乱无章（信息过多）kitchen sink

Mobile app的最佳实践

确定受众（audience）

设计可用的上下文

充分利用平台优势

使用户可以较为简便的学习发现高级功能

不应该以牺牲用户理解为代价来开发切割器图标（图标样式更多的遵循通用性）

滚动条的使用胜过多屏幕

Mobile app middleware（移动app中间件）

促进分布式组件的交流与协作

让开发者更多依赖于抽象环境而隐藏mobile的环境细节

帮助mobile app在需要时更好的理解上下文

Mobile app的一些特征

应用界面简洁美观，易于上手（不用花费大量的学习时间）

满足iOS、Android多平台使用需求

应用功能贴近用户实际需求

数据上传成功或错误均有明确提示

功能选项可以充分考虑用户喜好，进而进行个性化定制

后台数据库需要定时更新，保证数据准确性

CH20

评审技术

评审是技术人员为技术人员召开的会议

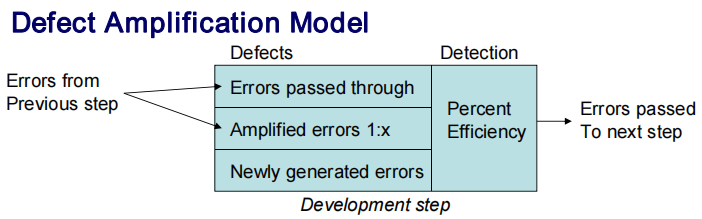
产品创造过程的技术评估

软件质量保证机制

Error 在产品交付于用户前发现的错误

Defect 在产品交付给用户后发现的错误

Defect amplification 缺陷放大



遗留的错误也有直接后传与放大后后传之分，percent efficiency 表示解决当前块内错误的比率

我们一般既假定design环节没有克服的问题会产生1.5 monetary unit的错误；到before testing阶段后，该错误会产生6.5 monetary unit的错误；during test 则达到 15 monetary unit的错误；如果 after release 则为 67 - 100 monetary unit的错误

Defect density（缺陷密度） = Err tot（在评审阶段找到的所有错误） / WPS（工作产品的单位之和）

WPS：work product size，已审查的工作产品大小的度量，往往是UML模型的数量，或文档页面数

Total review effect = preparation effort（Ep 在产品评审meeting之前审查产品的effort [以人·时为单位 ]）+ assessment effort（Ea 评审meeting期间花费的努力）+ rework effort（Er 纠正在审查过程中发现的错误的努力）

Total number of errors = minor errors found（次要错误的数量，不需要预先规定的努力[pre-specified effort]来纠正错误） + major errors found（需要一定预先规定的努力方法来克服）

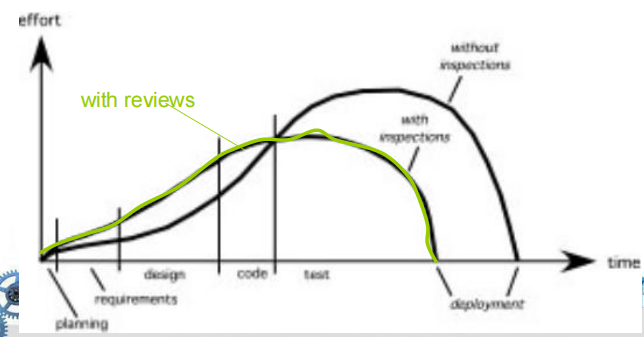
评审的优势：

发现并修复错误上时间的节省

预测工作表现

评审：

更少的努力，更早的结束项目



参考模型：

评审复杂性的影响因素（评审过程包含的角度）：为评审员明确地定义了不同的角色；对审查有足够多的计划和准备工作；定义了评审的独特结构；审稿人对进行的任何更正进行跟进

非正式评审：

临时会议，配对编程，简单的软件desk check

正式技术评审（FTR）：常常包括walkthroughs and inspections（演练和检查）

目标：

解决逻辑、功能、实现上的错误

确保软件满足需求

确保软件遵守了先前定义的规范

使工程更利于管理

确保软件可以以统一的形式进行开发

评审会议：review leader、standards bearer（SQA，标椎执行者）、producer、reviewer、recorder、maintenance oracle（数据库维护）

Sample-Driven Reviews (SDRs) 样本驱动的审查

量化产品

抽样统计错误个数并推广到全局

将所有产品按错误总数倒序排序，将有限的评审资源放到产品错误总数更高的产品上