什么是软件

一个软件由以下部分组成：

1. 在执行功能时提供一些所需功能的指令
2. 使软件运行的数据结构
3. 配置文件
4. 系统文档
5. 用户文档

什么是bug

1. Bug最早是因为一只小虫子飞进mark二号计算机，导致计算机无法正常工作，随后这个叫法就沿用下来了，用来表示系统中的错误缺陷漏洞

Fault Error Failure的区别

1. Fault 软件中的静态缺陷
2. Error 错误的内部状态，某些故障的表现
3. Failure error传播到外面，错误的输出

Pie模型三个必要条件 针对failure

1. Execution 执行必须通过错误的代码
2. Infection 触发错误的中间状态
3. Propagation 错误的中间状态必须传播到最后输出

Validation和Verification的区别

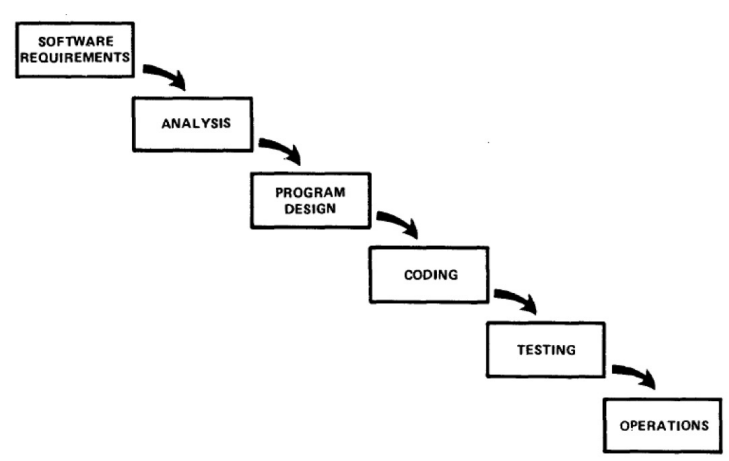
1. Verification是软件验证，从技术人员角度测试当前开发成果，保证按照流程开发
2. Validation是软件确认 从用户角度，测试当前的开发是否符合用户需求

软件测试原则 software testing axioms

1. 完全测试一个程序是不可能的
2. 软件测试是基于风险的练习
3. 测试不能展示没有的错误
4. 发现的bug越多，bug就越多
5. 并非所有的bug都能完全修复
6. 一个bug是否是bug很难说清楚
7. 测试对技术的要求很高
8. 软件测试人员不受欢迎

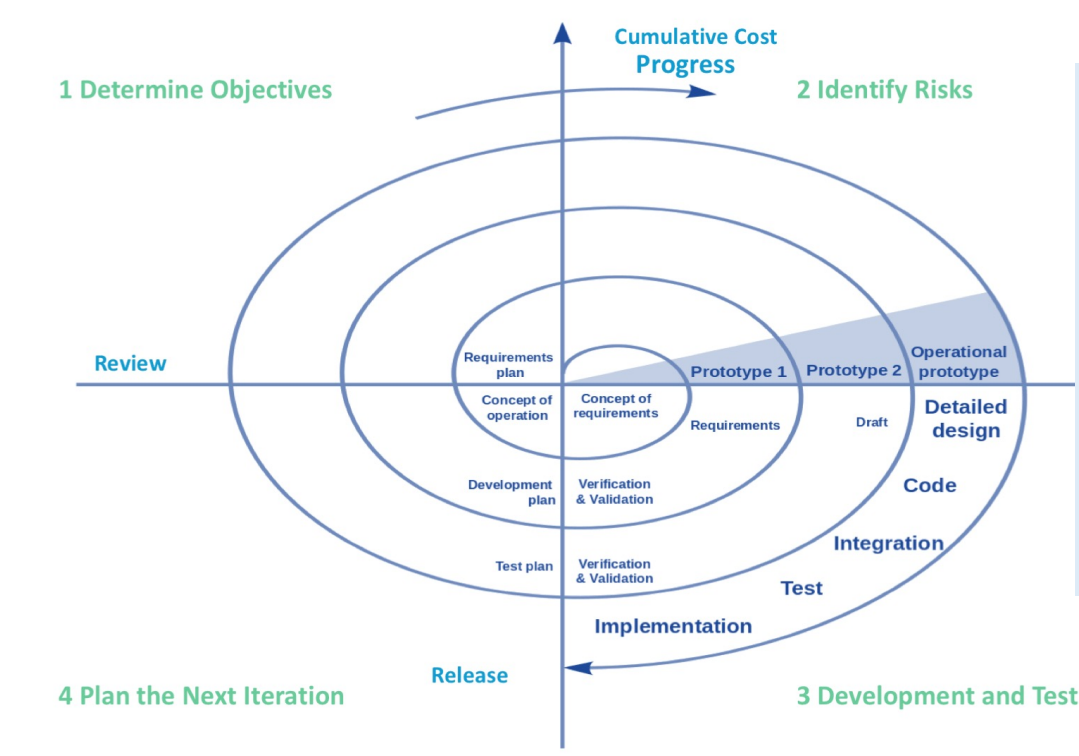
软件测试者的目标

1. 找到bugs
2. 尽快找到bugs
3. 确保bug能被修复

瀑布模型 waterfall model

1. 所有计划开始后就做好，一旦创建就不能修改
2. 任何后续阶段之间不存在重叠
3. 任何人看到程序的第一次机会在测试之后

螺旋模型 spiral model



1. 风险驱动开发
2. 瀑布模型与快速原型迭代模型的组合
3. 从设计目标开始到客户审查结束

Vmodel

用户需求 URS 验收测试

需求分析 SRS 系统测试

概要设计 SDS 集成测试

详细设计 SS 单元测试

编码实现

1. 瀑布模型的扩展 通过标记生命周期的每个阶段与测试活动之间的关系来强调验证和确认
2. 一旦代码完成，测试就开始了
3. 从单元测试开始，每次提升一个测试级别，直到 验收测试阶段完成

Wmodel

需求分析 需求测试 系统安装 验收测试

概要设计 概要设计测试 系统构建 系统测试

详细设计 详细设计测试 模块设计 集成测试

编码实现 单元测试

1. v的扩展 双v 测试不在代码实现后 与开发过程并行 开发与测试的合作

敏捷模型 Agile Model

1. 强调短时间构建可发布的软件
2. 他的时间周期是以周而不是以月来衡量的，并且工作是以高度协作的方式进行的
3. 开发人员与客户协商时，测试人员要一起加入
4. 测试人员需要将对话达成一致的要求转化为测试用例
5. 需求改变时，测试人员必须相应的更改测试用例

极限编程 extreme programing value xp

Communication 与客户交流

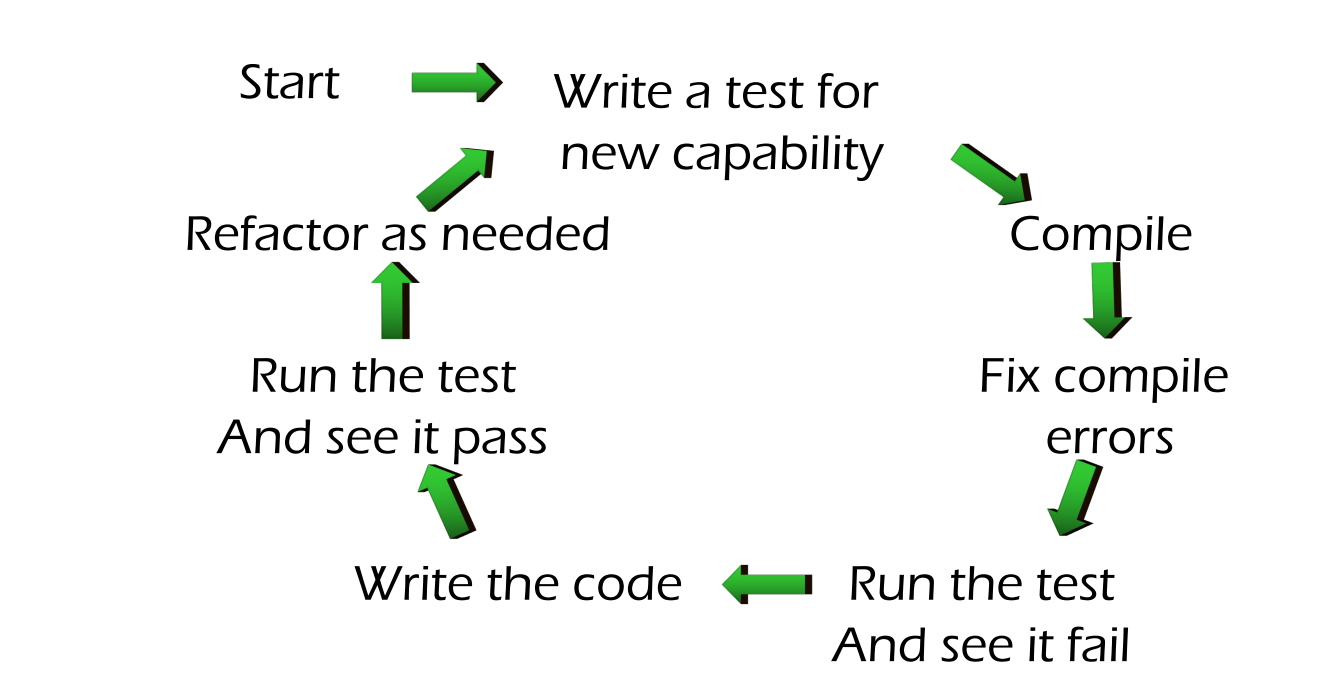
Simplicity 设计简单

Feedback 反馈

Coverage尽早交付 勇于面对需求

Respect

TDD 测试驱动开发 test driven development

1. 是敏捷开发的一项核心实践与技术
2. 在开发功能代码前，先编写单元测试用例
3. 测试代码确定需要编写什么代码产品，之后再开始真正的业务需求开发
4. 尽可能频繁的进行测试，每次更改代码都测试

静态动态验证 static dynamic verification

1. 静态验证 不需要执行软件代码 动态验证需要
2. 静态验证采用数学方法，规范和源代码之间的转换的符号验证

黑盒白盒测试

1. 黑盒 基于程序规范 旨在验证是否满足需求功能测试
2. 白盒 结构测试 检测软件编码时的错误

黑盒测试的方法

1.等价类划分 equivalence partitioning

程序输入域划分为若干部分，然后从各部分中选取少量代表性数据作为测试用例

每一类的代表性数据在测试中等价于这一类中的其他值，如果有一类发现错误，那么这一等价类中的其他例子也能发现同样的错误

三个原则：1.为每个等价类规定一个唯一的编号

2.设计一个新的测试用例，使其尽可能多的覆盖尚未被覆盖的有效等价类 重复 直到所有有效等价类都被覆盖为止

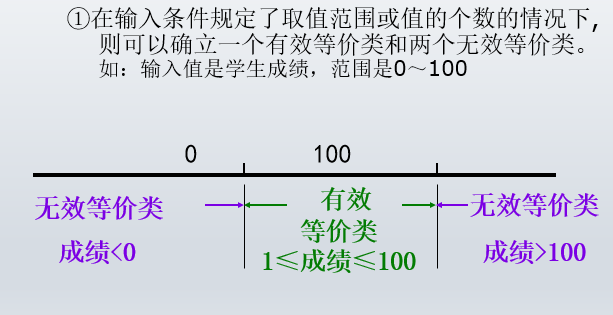
3.设计一个新的测试用例，使其反覆盖一个尚未被覆盖的无效等价类。重复这一步，知道所有的无效等价类都被覆盖为止。

有效等价类 valid equivalence class 符合程序要求，合理且有意义

无效等价类 invalid equivalence class 不符合要求 不合理且无意义

六个划分原则：

1. 输入条件为取值范围时，应该设1个有效等价类 2个无效等价类



1. 输入条件为必须如何时，需要设置1个有效，1个无效



1. 输入条件为bool值时，1个有效，1个无效
2. N组值选一，选择框，应该n个有效，1个无效

5.必须遵守规则下，1个有效，n个无效（从不同角度违反规则）

6.在得知已划分的等价类中各元素在程序处理中的方式不同，需要继续细分。

2.边界值分析法 boundary value analysis

是对等价类划分的补充 将等价类的边界值作为测试条件

10-50 需要测9.99 10.01 49.99 50.01

3.错误推测法 error guessing

基于经验和直觉推测程序中所有可能出现的错误

4.场景测试 scenario test

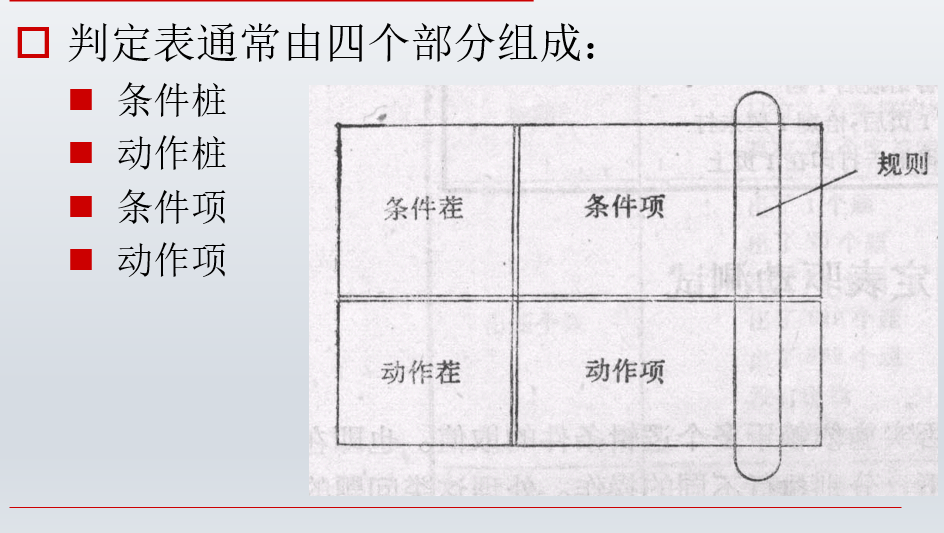
测试人员假设自己是最终用户，并找到用户可以在软件上运行的真实场景或用例

需要模拟用户正常的操作 也需要模拟用户操作中的错误

分为基本流程和备选流程

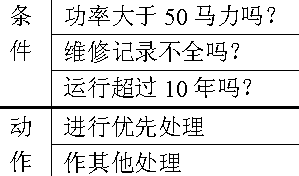
适合场景测试的情况：只有一个场景包含基本流程

5.判定表测试 decision table

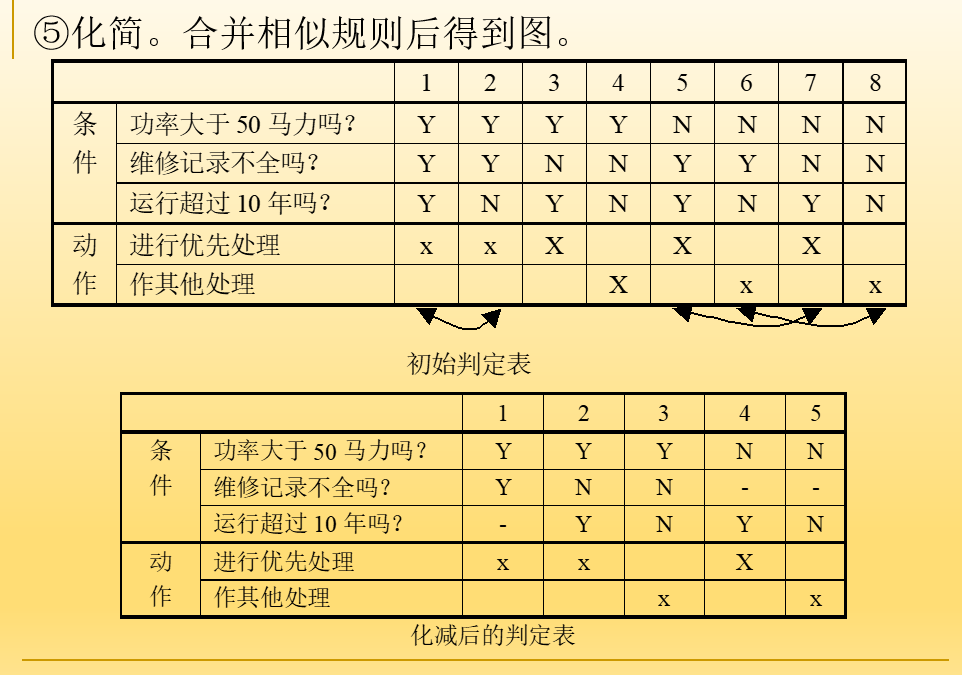
一个表分为四块

条件桩 condition stub 动作桩 action stub 条件项 condition entry 动作项 action entry

要会做化简







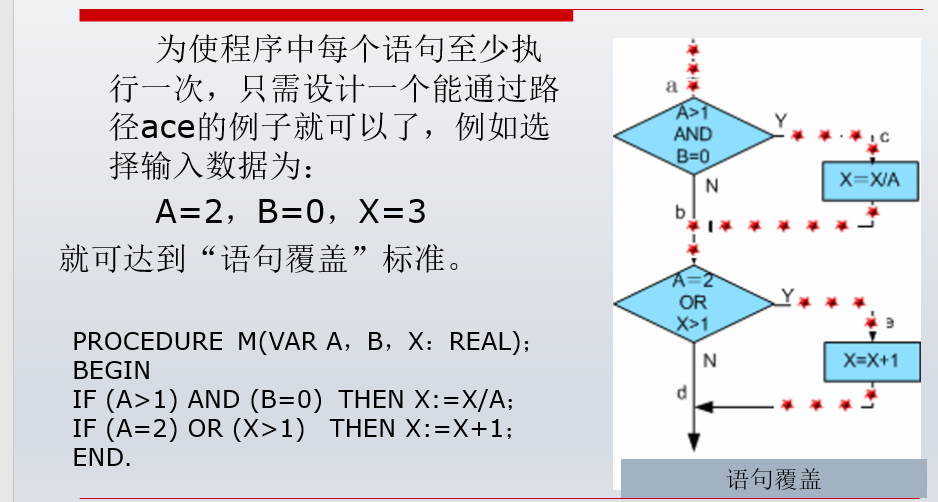
白盒测试

是结构测试，盒子是可视的，清除代码是如何运作的 白盒测试期望覆盖程度高

覆盖标准

1. 语句覆盖 最弱的逻辑覆盖 选择足够的测试用例使得每一个语句都被执行至少一次

Statement coverage



1. 判定覆盖 执行足够的测试用例 使得程序中的每一个分支都通过一次

Decision coverage 每个判断的每种情况执行一次就够

1. 条件覆盖 使得程序中的每个判断的每个条件的每个可能的取值都执行一次

Condition coverage 每个条件都要有执行

1. 判断/条件覆盖 使得判定中每个条件取到各种可能的值，并使每个判定取到各种可能的结果。
2. 条件组合覆盖 使得每个判定中条件的各种可能组合都至少出现一次。

Condition combination coverage 每个条件互相组合

1. 基本路径测试 设计足够的测试用例 覆盖程序中所有可能的路径

碰到环只执行一次 根据程序流图画出控制流图Control flow graph 化简 算出圈复杂度

算复杂度方法：区域 边-节点+2 判定节点+1

导出测试用例 复杂度=测试路径

准备测试用例

静态测试

不通过运行软件，而是通过检查和评审软件 容易开展

代码审查四要素： identify problem follow rules prepare write a report

1. 桌面检查 程序员自查 效果不好
2. 代码审查 code review
3. 同行审查 peer review 一个非正式的由程序员组成的小组充当评审员
4. Walkthrough 代码走查 更加正式 开发者将代码正式的呈现给一小群开发者或测试者 开发者一行一行的解释代码 检查者仔细倾听

测试计划 Test Plan 描述软件测试范围和活动的文档 他是正式测试活动的基础

分为：主测试计划 单元测试计划 集成测试计划 系统测试计划 验收测试计划 安全测试计划

测试计划的步骤 1.定义测试策略2.定义测试系统3.估计测试工作量4.编制和审查测试计划

测试用例 test case 测试用例是为某个特殊目标而编制的一组测试输入、执行条件以及预期结果，用于核实是否满足某个特定软件需求 一个测试用例可以拥有多个测试脚本

测试用例的集合称为测试套件

单元测试 unit test

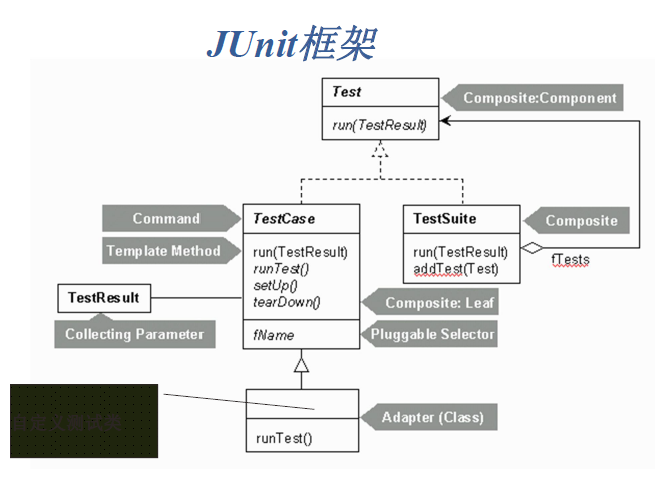
1. 对软件中的最小可测试单元在与其他部分隔离的情况下进行检查和验证。

内容：模块接口测试 重要路径测试 边界条件测试 错误处理 局部数据结构

自动单元测试 automated unit test

自动 可重复 独立

Junit



集成测试 integration test

为了确保各单元组合后可以正常运行 driver stub

一个登录页跳转到主页 登录页是driver 主页是stub

1. bigbang 大爆炸 将整个系统视为一个子系统 在单个测试会话中测试所有模块 只有一个集成测试会话 故障隔离十分困难
2. top-down自上而下 主控模块作为测试驱动器（driver），使用集成的方式（深度优先广度优先）将桩模块一个个替换，每个模块被集成时都需要进行单元测试。

优点：1尽早验证主要控制和判断点 2.支持故障隔离 3.只需要一个驱动器，减少驱动开发

缺点：1 桩的开发量大 2.底层测试不充分

1. bottom-up 自下而上 起始于模块依赖树的底部叶子节点 使用驱动模块对选定模块测试 使用实际模块替换驱动模块

优点：1.可以并行集成，效率高2.减少桩的工作量3.对底层组件较早验证

缺点：1驱动开发工作量大2.高层的验证推迟，设计上的错误不能及时发现

经过集成测试后的程序，作为计算机系统的一部分，与系统中的其他部分结合起来，在实际运行环境下进行测试

1. 功能测试 functional test

验证软件的行为是否符合预期

输入坏的用例以测试隐藏的假设 即无意义的输入应该有合理的输出

贯穿所有级别的测试 但是对单元测试要求高

用于测试的接口是用户接口，网络接口，专用硬件接口

基于黑盒测试

1. 非功能测试 nonfunctional test

性能测试 高强度

压力测试 高并发

容量测试 处理任务的极限值

健壮性测试 错误回复能力与健壮性

安全性测试 防范能力

恢复性测试 容错

备份测试 失败时能成功备份数据

兼容性测试 与操作系统浏览器等的兼容程度

安装性测试 安装是否成功

1. 回归测试 regression test

软件变动需要确保原有功能正常

验收测试 accepting test

是所有测试的最后一关 用于测试系统是否符合业务需求 在系统测试后 正式交付前进行 在黑盒测试方法下进行

Alpha test

在开发环境下的测试 在模拟实际环境下的测试 非正式的验收测试

Beta test

无开发者，只有用户 软件已经上线 用户在测试时需要把问题提交给开发者 开发者不能控制beta测试的测试环境

软件测试文档

包括：测试计划书 测试设计 测试用例规范 测试策略 周报 用户验收报告 风险评估 测试日志 错误报告 总结报告

Bug report:

1. 包含需要修复的错误信息
2. 是bug发现者和bug修复者的有效沟通形式
3. 发送给负责人
4. 可以尽快解决bug
5. 建立合作的共同基础
6. 以规定的方式提交

测试总结报告

测试完成后的交付成果 向各方解释有关项目测试的细节

自动化测试 automated test

支持重复测试用例和执行 有助于测试大型项目 支持并行测试 提高准确性 节省时间

缺点： 设备昂贵 需要具备丰富的编码经验

软件维护 software maintenance

在程序投入后修改程序 通常不进行大的变动

影响软件维护的因素：团队稳定性 合同责任 员工技术 项目的结构以及年龄

软件迭代 software revolution

小、差、简易的系统的持续性的变为更好的系统