Part I Big picture

测试：在软件结构层次范围内或多种环境下评价软件功能点和行为的过程；根据规格说明书和用户需求验证&有效性确认软件产品；发现软件bugs。

测试空间：获取性、性能特征、细节层次

测试循环：（死循环）

执行用例->结果->评价 猜想原因->附加测试->测试用例 ->执行测试

识别原因->修改->回归测试->测试用例 ->执行测试

Ch.1 软件测试背景

软件缺陷：软件错误；引起软件失败的因素；官方定义。

高质量的软件是合理的bug-free，及时交付并在预算之内，满足了需求或期望并可维护。

不同的视角有不同的观点

高质量软件类型：产品质量、过程质量、商业环境下的质量。

软件失败的因素：确定、故障、问题、错误、时间、异常、偏差、失败、矛盾、特性、bug

故障、错误、失败统称bug

软件缺陷形式上的定义：

产品规格说明书：定义了产品应该有的的所有特性/功能。

Bug定义：（有一个就是）

1. 软件没有做规格说明里的东西
2. 软件做了，但和规格说明里不一样
3. 软件做了规格说明里没有提到的（修改规格说明）
4. 软件没有做规格说明没有提到但是应该做的
5. 软件很难理解，很难使用，很慢或（在测试人员角度）最终用户抱怨

软件失效机理可以描述为：软件错误→软件缺陷→软件故障→软件失效

（了解）

* 软件错误（**software error**）：是指在软件生存期内的不希望或不可接受的人为错误，其结果是导致软件缺陷的产生。
* 软件缺陷（**software defect**）：是存在于软件（文档、数据、程序）之中的那些不希望或不可接受的偏差，其结果是软件运行于某一特定条件时出现软件故障，此时称软件缺陷被激活。当软件意指程序时，软件缺陷（**defect**）与软件**/**程序污点（**bug**）同义。软件缺陷是存在于软件内部的、静态的一种形式。
* 软件故障**（software fault）**：是指软件运行过程中出现的一种不希望或不可接受的内部状态。出现软件故障时若无适当措施（容错）加以及时处理便产生软件失效软件故障是一种动态行为
* 软件失效（**software failure**）：是指软件运行时产生的一种不希望或不可接受的外部行为结果。

ISO 8402质量：

产品的所有功能和特征、过程或提供的服务都具有它的能力-----满足指定或必然包含的需要。

Bug：故障，错误（内部存在的状态，开发和维护时存在）、

失败（外部状态，与系统本来要求的行为和本身的不一致、背离）

故障、错误VS失败：

内部VS外部

代码/设计/数据VS功能/特性

白盒测试VS黑盒测试 检测出来

（需求）错误->故障->失败 （设计）故障->失败 （测试数据）故障->失败

引起bug a. 规格说明50% b. 设计 c. 其他

为什么规格说明是第一位？没有写；没有被详细写；一致改变；没有和队员及时沟通。

If you can’t say it, you can’t do it.

设计：和规格说明原因一样；仓促，改变，沟通不好

为什么软件有bug？沟通不良；不沟通；软件变化；自我

Bug成本、损失：指数级增长，从规格说明->设计->编码->测试->发布

测试员的目标：找到bugs；尽可能早的发现它们；确保它们最终都被修正；学习被业界讨论的技术。

测试员：开发人员、测试组、QA组、最终用户

调试员：程序分析员、开发人员

测试：以找错误的目的，在控制条件下对系统或应用的操作，评价最终结果。尝试使出错，意图是“检测”

调试：开始于一个识别的错误，是定位错误，并修正的过程。不关注怎样发现bug，关注怎样修正bug

Some exercises:

What’s wrong with just testing that a program works as expected?

Give some reasons why the product specification is usually the largest source of bugs in a software product.

Ch.2 软件开发过程

软件产品：

不仅仅是一个程序，是由正常交付的软件组件组成；

典型地被组织到和生命周期中一些典型阶段一致的种类中

所有测试员对这些产品组成都有所了解且去检测他们

产品组成：用户需求；MRD（任务/特征描述，可用数据）；规格说明；进度表；设计文件；在线帮助；read me；

发布包裹。帮助文件；用户手册；错误信息；安装说明；产品支持信息

需求：使用者所需要的能力条件，去解决一个问题或获得某个事物（IEEE）

好的需求：可见；一致；完整；运行可靠；可度量；可取得；可跟踪；可修改

规格：和需求不太一样，把需求最终正式文档化，最终定义产品是什么、外观怎样、有什么功能

写MRD和SPEC，注意可确认性，符合原来规格说明定义的

Some ways to quantify：速度；规模；易使用；可靠性；健壮性；易携带性

进度表：详细制定；有软件工具生成，也有很多估算时间和花费的方法；测试人员也要考虑测试计划和进度表。

设计文档：详细层次。结构设计文档，数据流图，状态转换图，实体关系图，决策表

测试文档：说明软件产品测试的文档。测试计划，测试用例，bug报告，度量统计数据及报告总结，发现了多少bug

软件工程人员：PM；高级的工程师；系统测试员；系统分析员；构架师；编程人员；测试员；QA；技术文档的书写者；用户协作的书写者（用户手册，安装说明）；用户训练员；手册编写者；配置管理员

PM：和设计人员协作决定什么功能可以在什么时间实现；在测试时和设计人员、测试设计工程师沟通

测试组织中的角色

测试管理者：协调，决定问题优先解决

测试领导：领导设计测试计划等

高级测试员：设计测试模块和测试用例，测试用例执行，提供技术支持

低级测试员：执行测试用例，记录测试结果，提交bug报告和测试中的问题

为什么要再SDLC建立模型？

SDLC软件开发生命周期：起于应用第一次被构想，止于它不再被使用

计划->分析->设计->实现->测试->roll-out

SDLC过程：需求->产品设计->技术设计->编码&单元测试->功能&系统&验收测试->发布

big-bang大爆炸模型：

集中很多人和金钱；花费很多精力和时间；极少的计划、安排和正式开发过程；没有测试；软件or废品

code-and-fix边写边修改模型：

开始于一个不正式产品规格说明，编码、修改，重复直到满意；工作随意；

“There’s never time to do it right, but there’s always time to do it over.”

随着项目一直在变，要不断改变测试方法；在测完这个版本前，接到了下一个；

waterfall瀑布模型：相邻阶段间可回溯；可跳跃回溯

每件事都被仔细地严格地说明；安排好了准确的计划和进度表；测试直到后期才开始；

会有bug潜伏在早期程序中，最后影响产品；修复代价会很高

Rapid快速模型：快速应用，把产品定义过程缩短

没有特别提供测试，但经常在重复阶段时做；缺少正式定义测试阶段；整个模型易退化成边写边修改模型

V模型--------快速应用开发

Spiral螺旋模型

程序员可以较早介入到软件开发中；帮助测试员选择合适的测试计划和安排；开发需求和说明在随时间变化；

每个螺旋都是瀑布模型；加入风险分析；太复杂，周期太长

XP极限编程：

小团队，高风险，快速变化的需求，需要可测性

主要关注交流、简单、回馈、面对高风险的勇气

TDD测试驱动开发：测试先于开发，TF+重构

在编之前就写测试代码，优化代码等要求粗要重构 功能点+性能点

阶段开发模型：bulid release1 -> use release1

Bulid release2 -> use release2

Bulid release3 -> use release3

增量模型VS迭代模型：一步一步完成VS现有抽象概念、定义，然后再填充

Some exercise:

* What disadvantage is there to having a formal, locked‐down specification?
* Why can the waterfall method be difficult to use?

Ch.3 软件测试的现实

不存在bug-free的产品；测试永远也无法穷尽；适合的停止时间；

测试有可能的话，尽早介入；bug严重等级；测试和fix没有界限；测试人员数字要足够大

测试公理：

1. 不可能完全测试一个程序：输入/输出太多；路径太多；观看者自己的评判标准
2. 测试不能够证明bug不存在
3. 软件测试是非常有风险的：寻找最佳点（成本和缺陷）
4. 发现的bug越多，这里就有更过的bug：bad days；犯同样的错；发现的知识冰山一角
5. 杀虫剂悖论：有抵抗，杀虫剂不再有效；新的，各种不同的测试用例
6. 不是所有发现的bug都会被修改：时间；根本不是bug；风险；不值得；不能被修改；第三方因素
7. 很难说清什么样的bug算是一个bug
8. 产品规格说明永远不会有最终版本：随着用户需求变化
9. 软件测试人员不是开发团队中受欢迎的：早点发现bug；控制情绪；不要总是报告坏消息
10. 软件测试是一个比较严格的技术

精确度（precision）：所有的答案都差不多，但不是正确答案 eg. 0.123±0.03 可接受

准确度（accuracy）：准确的答案 0.125

产品规格是错的，有精确度，无准确度

确定（verification）：确认系统满足规格说明要求

验证（validation）：是否能够满足用户需求

质量（quality）：是否是质量中的一个要素，决定权完全在使用者手里，主观

可靠性（reliability）：是否很稳定的类型，看用户的需求

测试（testing）：参与质量的控制方面，只关心错误，保证被修正，找bug

质量保证QA（quality accuracy）：防止软件发生缺陷，创造比较标准的流程，可控，预防

QC -> QA -> QM -> TQM 手工操作者->专职检验员->过程统计技术->全面质量管理->以顾客为中心

黑盒测试：不知道结构，只关心输入和输出 功能测试、数据驱动测试

白盒测试：知道内部结构、路径是怎么实现的 路径、分支、循环、结构性测试、逻辑驱动测试

灰盒测试：综合以上两种，知道部分结构

静态测试：检查并review代码及其他支持材料，不需要运行程序

动态测试：代码进行运行

测试分类：黑盒；白盒；单元（在函数或模块内的测试，不关注接口）；集成（更注重接口，新增的功能）；

回归（难，原来的bug已修复，再次提交时，对这个bug的测试；原来已经通过的测试用例要不要测试，争议较大，如何选择测试的用例集合）

测试阶段：

比较测试：和竞争产品比较弱点和优点

Alpha（α）测试：系统测试

开发接近完成时测试；可能需要少量设计修改；应用到真实的潜在用户；小范围，可控；把人集中起来

Beta（β）测试：验收测试

开发和测试都可以看作完成，在最终发布前寻找最后的bugs和问题；范围更广，不可控。Eg. 游戏公测

测试分类：

功能性测试：黑盒测试的同义词

系统测试：类黑盒测试、灰盒测试，基于总体的需求和规格说明

压力测试：经常和负荷测试，性能测试一起

性能测试：经常和负荷测试，压力测试一起

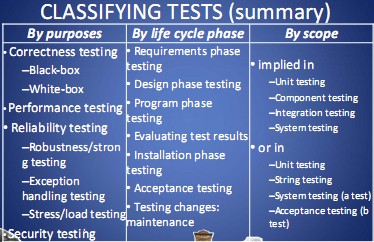
负荷测试：在重负荷下测试，得到系统响应时间在哪一点下降了或者失败了

End-to-end：测试模仿现实世界使用

验收测试：基于最终用户或客户的规格说明的最终测试

可用测试：测试用户友好度，主观的，决定于最终用户或客户

Recovery复活测试：测试系统从碰撞、硬件损坏或者其他问题中的恢复情况

安装/卸载、安全、兼容性

Some exercises：

* If you were testing a simulation game such as a flight simulator or a city simulator, what do you think would be more  important to test –its accuracy or its precision?
* Is it possible to have a high‐quality and low‐reliability product? What might an example be? 汽车，手机
* If you were testing a feature of your software on Monday and finding a new bug every hour, at what rate would you expect to find bugs on Tuesday?

Part II Testing Fundamentals

SDLC中的测试阶段：

单元测试->集成测试------->功能测试--------->系统测试----------->验收测试--------->安装测试

组件 设计规格说明 系统功能需求 其他软件需求 用户需求规格说明 用户环境 使用系统

Ch.4 Examining the Specification---------静态、黑盒测试

为什么要审查规格说明？

能发现无法通过测试发现的bugs，特别是逻辑设计问题；在生产之前就发现缺陷；降低成本；缩短SDLC

审查规格说明是静态测试（它还不是一个程序）、黑盒测试（不考虑规格说明是如何确定的）。

黑盒测试：测试人员无需了解被测试项的内部结构，直接根据程序输入和输出之间的关系或程序的需求规范确定测试数据，推断测试结果的正确性 主要测试功能需求。同义词：行为、功能、不透明、封闭的盒子

需求

功能测试、系统测试、验收测试-------------是黑盒测试

输入A

输出B

黑盒测试优点：

测试员不需要任何程序语言的知识，测试员和程序员是互相独立的；

事件

测试员从用户的角度测试，而不是设计者的角度；

规格说明一完成，测试用例就可以被设计；

在大型代码单元里，比白盒测试更有效；

帮助暴露规格说明中的歧义和不一致

缺点：测试每种可能的输入是不可能的；测试可能是多余的；没有明确的规格说明，测试用例很难设计；

无法指明错误位置；

文档测试：

确保书写了需要的文档；

确定规格说明文档的信息是一致的、准确的、易读的

有的需求说明指明了文档的形式和阅读者

高层review：

检查是否有逻辑错误；

阅读MRD，更多地想用户需要什么

现有的标准和指导方针（工厂需求；政府标准；硬件和网络标准）

Review and test 类似的软件

低层规格说明测试技术：

规格说明属性检查列表/清单：

完整性、准确性、精确性（无二义）、一致性、相关性、合理性（可行性）、code-free不涉及代码、可测性

规格说明术语检查列表/清单：

always，every，all，none，never；无例外？

certainly，therefore，clearly，obviously，evidently；很有说服力，同意否？

some，sometimes，often，usually，ordinarily，customarily，most，mostly；太模糊，无法测试

软件评审过程（SIP----software inspection process）

Planning：定义要评审的工作产品，建立评审日程表； 主持人；作者

Overview：让不熟悉要评审工作产品的队员熟悉它； 主持人；作者；检查者

Preparation：队员独立评审产品，寻找缺陷； 检查者；主持人；读者

Inspection：评审队员开会，讨论产品中合适的缺陷； 作者；检查者；主持人；读者；记录员

Rework：工作产品被修改，直到符合需求和规格说明； 解决缺陷；未解决的报告项目经理；告知主持人

follow up：rework被验证，收集和总结最后评审数据，评审结束 作者；主持人

选择评审团队->安排地点和时间->分发文档->主持评审->记录结果和所做之事

SIP角色：主持人；作者；读者；记录员；检查者inspector

Exercises：

* Explains what a tester should worry about with this line from a spec: The software will allow up to 100 million simultaneous connections, although no more that 1 million will normally be used.

Ch.5 Testing the software with blinders on------------动态、黑盒测试

动态、黑盒测试：运行软件

不需要知道如何得到结果的；不可以看代码，但可以问程序员数字限制，内部事件

Test-to-pass：通过性测试，关注最小能够完成的功能，不测性能，按正常情况，不测边界值

Test-to-fail or error-forcing：失效性测试，目的是想破坏它，选择软件的弱点

在失效性测试前使用通过性测试暴露bugs

定义等价类（自反、传递、对称）：帮助我们把测试用例进行划分而不会带来高风险。数据、逻辑流

数据测试：定义边界条件---数据；分成三类：有效值、边界值、无效值；边界条件在需求或规格说明里定义好

等价类划分：把所有的输入分成一系列等价类（互不相交，但∪是整个全集），减少测试用例个数

分为：有效等价类、无效等价类

有限个数的输入等价类：

行为类似的放入同一个类里；每个类只要测是个代表；

如果在一个等价类里发现缺陷，那这个等价类里的其他也会有缺陷。

每个类定义一个/多个测试用例：覆盖有效类的测试用例；覆盖最多一个无效类的测试用例

Eg. If input is a 5‐digit integer between 10,000 and 99,999, Equivalence partitions are:

< 10,000           10,000 ‐ 99,999         > 99,999

1. 如果输入是一个范围或一个特定的值，一个有效等价类，两个无效等价类
2. 如果输入是一系列有关系的值的集合或一个布尔值，一个有效类，一个无效类

Further 分类可能：检查默认值、空值、零值；检查无效值、错误值

黑盒测试----边界值分析（BVA）：

基于定义、产生探索边界条件的测试用例的技术；边界条件是很多错误的源头

对输入输出的边界都要考虑；白盒测试应用也要测边界

每个等价类中选择一个/多个随意值；选择等价类里准确的（上、下）边界；选择趋于但不等于边界值的值

Eg. – classes x < 0,  x >= 0,  on boundary : x = 0

– classes  x < 0,  x >= 0,  below and above:        x  =  ‐1,  x = 1

Test a function which limit user input to 6‐digit positive integer

– Test cases :

Class arbitrary value:                   X1 = 123123

Class boundary value:                 X2 = 12345；X3 = 1234567；X4 = 1； X5 = 0

Class invalid value:                      X6 = ‐123123；X7 = asdasd；X8 =  000123；X9 = asd123；X10 = Empty

可能的边界条件：

输入输出：大小、类型；

寻找限制：First‐1/Last+1；Start‐1/Finish+1；Min‐1/max+1；Less than empty/ more than full；Just Over/Just Under

ASCII Encoding Always Produce Sub‐Boundary Conditions

• The ASCII code is 8 bits

– If numeric digits are to be input, check just out of range with the ‘/’ and ‘:’

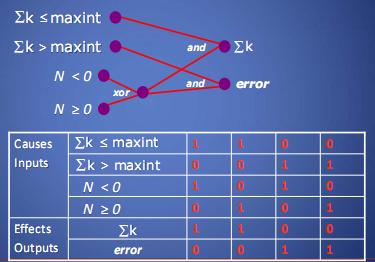
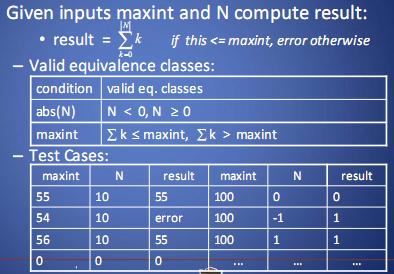
– If capital letters are to be input, check just out of range with the ‘@’ and ‘[’

– If lower case letters are to be input, check just out of range with the forward quote(‘`’) and the ‘{’

因果关系：

因/果：一个输入/输出条件或一个等价类输入/输出条件；

因果用布尔值表示，之间的关系可以用布尔图表示



黑盒技术分析输入条件的组合

定义规格说明中的因（输入当前状态）和果（输出新的状态）

case矩阵

布尔图连接因果

去掉不可能的因果组合

开发决策表，从图中每一列输入和输出的组合

因果分析过程步骤：

列出并标记因和果

画一个因果图描述原因、中等原因和结果影响的逻辑组合

看图开发决策表（因VS果）

把每一行转变成一个测试用例，或把多行转变成一个测试场景

错误猜测：

测试员利用直觉和经验去定义潜在的错误并设计测试用例来揭发它们

开发者可能做的合理但不正确的假设；特殊状态；不期望的、不正常的程序应用环境

 常见的条件：重复实例；空白或字符串空值；负值；数字填空里没有数字值；输入太长或太短

只是猜想，为可能的错误或错误状态列清单；基于清单写测试用例

更老练的错误猜测：风险分析 高风险的代码更要被彻底的测试

黑盒测试技术：等价类划分、边界值分析、因果图、错误猜测。。。。

没有一个是完整的；它们是互补的

状态测试：

所有软件都送一个状态转换到另一个状态；对于有些软件，状态转换是明显的；

为了决定状态，需要建立状态转换图STM

STM显示从一个状态到另一个状态的逻辑流；定义状态和转换（从一个状态到另一状态的触发）

通过性测试：

分类的可能选择：

每个状态至少访问一次；测试最常见的状态-状态的转换；

测试状态间最不常见的路径；测试错误状态的所有进口和出口；

检查所有定义状态的状态变量

和规格说明的书写者和程序员定义可能的状态 ，但是DO NOT GO TO THE CODE LEVEL!

失败性测试：竞态条件下

repetition（发现内存泄露）&stress（bad conditions，内存少、CPU慢）&负载测试（大数据文件、长时间运行）

Exercise：

* If you are testing a program’s ability to print to printers, what generic test-to-fail test cases might be appropriate?
* Assume that you have a 10-character-wide ZIP code text box. What equivalence partitions would you create for this text box?

Ch.6 Examining the code--------静态、白盒测试

三种方式：

桌面检查 走查 审查

Peer review walkthrough inspection

非正式-----------------------------------------------🡪正式

正式review：结构性过程

4个元素：定义问题，直接关注代码，而不是谁写的；建立并遵循审查的规则；准备并分配责任；写报告

并不只只是 ”get together and go over code”

正式inspection：更严格

定义好的角色和责任；定义好的步骤（overview；preparation；inspection；meeting；rework；follow-up）

桌面检查：最不正式；涉及到一个编码人员和少量同伙；但仍要保证4个元素

走查：向正式走近了一步；程序员和一个小组（<=5）的程序员和测试员一起审查；每个人事先都有代码拷贝；

 line‐by‐line, function by function地读代码，说明做了什么，为什么做了

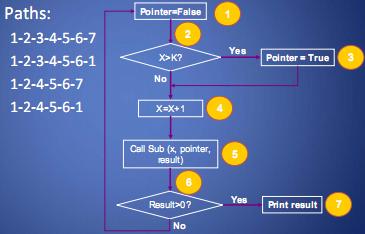
彻底地测试代码，有几种方法：

语句测试：每个语句至少被执行一次

分支测试：每个分支产生2个路径。n个分支，则有2n条路径

路经测试：一般无法达到所有路径都被测试。折中，有选择的测试

定义使用路经测试



评审：

最正式的review；高度结构化；在会议之前日程和代码都被考虑了；展示者或阅读者不是程序员；

所有其他人是扮演着不同角色的评审者：用户、测试员、产品支持者、主持人、记录员

所有正式review的目标：

寻找代码的问题和遗漏；代码是否遵循标准或指标（代码风格eg. 不适用goto；用while代替do-while）

规范（代码风格）的好处：增加了可靠性；可读性；一致性；易读性

Organizations Producing Various Standards and Guidelines

• ANSI ‐ American National Standards Institute

• IEC ‐ International Engineering Consortium

• ISO ‐ International Organization for Standardization

• NCITS ‐ National Committee for Information Technology Standards

• Plus various professional organizations

– ACM ‐ Association for Computing Machinery

– IEEE ‐ Institute of Electrical and Electronic Engineering

• Plus some of the testing organizations which we will consider later produce some standards for testing processes

Exercises：

– What key element makes formal reviews work?

– Besides being more formal, what’s the big different between inspections and other types of reviews?

Ch.7 Testing the software with X-ray glasses----------动态、白盒测试=结构性测试

不仅仅测试代码，也直接涉及了代码结构（路径；数据流）

动态、白盒测试：

片：较低层次的函数，过程，子路径，库函数；

高层测试；

对于变量和状态，访问控制命令，强制执行

度量有多少代码已经被测试了

调试角色：bug之间会掩盖；

白盒测试：

基路径测试（目标、最基本）：来测试每一条独立路径，至少每条一次；以路径为主

条件测试：所有逻辑条件（T/F）都进行测试；

循环测试：都有控制变量，在边界或边界内部进行测试；

数据流测试：所有的数据结构都测，保证其有效性

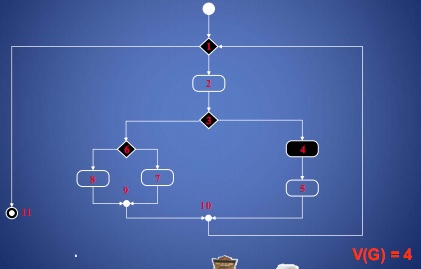
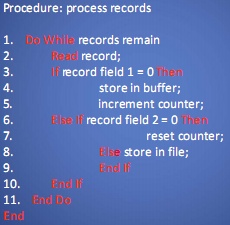
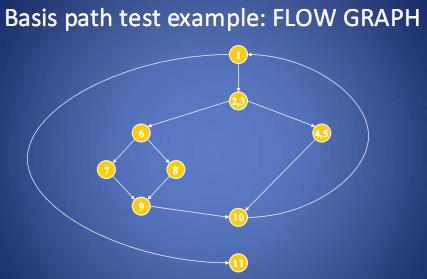
基路径测试：

使用代码，画出相应的程序流程图（控制流图，有编号和节点），有向图要有结束节点；

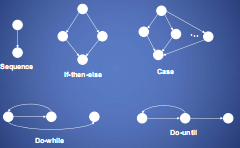
把圈复杂度确定下来，有向图中形成的圈的个数代表了圈的复杂度

线性独立路径条数（基路径条数）=圈复杂度

为这些路径设计测试用例 有向图



Flow graph-----控制流图？

圈复杂度：度量代码逻辑复杂度，定量

V(G) = 边和节点共同围成的区域

V(G) = 边的个数—结点个数 + 2

V(G) = 谓词节点数（入度>=2的节点）+ 1

外部有一个圈：和外部屏幕围成一个区域

Sequence有1个；if-else、do-while、do-until有2个；

圈复杂度的上升会提高程序模块化程度，但一旦超过

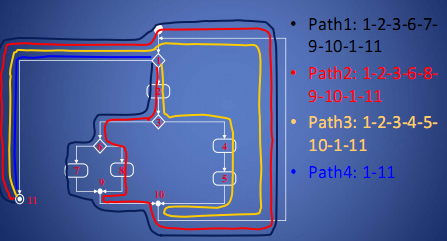
某个值，会是模块化程度降低，且出错率增加。

决定线性无关（独立）路径的一个基本集合：

线性无关（独立）路径：引入至少一个新的条件或过程；

基本集合：通过代码的独立路径；不是唯一的，原始路径不一样

测试用例：测试基本集合中的每个路径



条件测试：

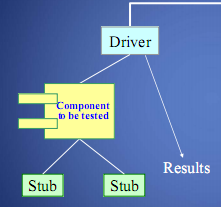
逻辑条件：简单条件（a≤b; NOT(a≤b)）、复杂条件（ (a>b) AND (c<d)）、关联条件（ ((a\*b+c)>(a+b+c))）

测试错误包括：布尔操作；布尔值；布尔圆括号

分支测试：复杂条件C，测试C的分支是T/F；e.g., for C = (a>b) AND (c<d)

域测试：表达式E1关联E2，测试E1> < = E2；n个变量，有2n个测试

分支测试：运行足够的测试用例，尝试去覆盖所有路径

循环测试：简单循环测试、鸟巢循环测试、关联循环测试（独立->简单；依赖->鸟巢）、无结构循环测试

数据流测试：根据变量的声明位置和使用来选择测试路径

DEF(S)----定义X的位置；USE(S)----使用X的位置

数据使用链：[X, S, S’]的集合，X是没有被重定义

每个数据使用链都被覆盖一次

单元测试用例：接口、本地数据结构、边界条件、独立路径、错误处理路径

Test driver：测试驱动，不是产品的一部分，简单程序，调用被测试模块

Test stub：测试桩，下级模块，被测试模块需要调用的单元，为单元提供输入数据

每个单元测试都要开发驱动或桩

集成测试：big bang VS 迭代、回归测试

Button-up 测试：单元测试、集成部分单元并测试。。。。；驱动代码一定要写

Top-down测试：

先写顶层函数，如main函数、写桩去处理调用其他函数

测试并调试顶层函数、用及代码代替桩、测试并调试桩和它调用的函数、直到。。。

– What's the difference between dynamic white‐box testing and debugging?

– What's the difference between a test stub and a test driver?

– Of the three code coverage measures described, which one is the best? Why?