一、软件测试基本概念

1. Crash

> *A **crash** is when your computer dies. It is also called an **idiosyncrasy**(行为异常).*

2. Software Bug

- *软件缺陷，程序错误*

3. A Formal Defination

1. ***功能缺失*** Doesn't do something it should do.
2. ***错误、缺陷*** Does something it shouldn't do.
3. ***功能多余*** Does something it doesn't mention.
4. ***对隐性需求的把握，同时发现需求的遗漏*** Doesn't do something it doesn't mention but should.
5. ***用户体验的角度*** Difficult to understand, hard to use, or will be viewed by the end user as just plain not right.

4. Other Defination

- ***Software Fault*** : A static defect in the software
- ***Software Error*** : A incorrect internal state that is the manifestation of some fault.
- ***Software Failure*** : External, incorrect behavior with respect to the requirements or other description of the expected behavior.

5. 软件生产中出现问题的不确定性

- *需求调研*
- *需求分析*
- *产品立项*
- *总体设计*
- *详细设计*
- *程序编码*

6. Bug在软件生产各环节中发生概率

Specification	Design	Code	Other
56%	27%	7%	10%

7. The Cost of Bugs

The cost to fix bugs can increase dramatically over time.

Specification	Design	Code	Test	Release
$1	$10	$100	$1000	$1000+

8. What Exactly Does a Tester do ?

- **Find bugs** as early as possible
- make sure they **get fixed**

9. What makes a Good Software Tester ?

1. **探索精神** *Explorers*
2. **故障排除** *Troubleshooters*
3. **严格的，苛刻的，无情的** *relentless*
4. **创造力** *creative*
5. **完美主义者** *perfectionists*
6. **判断力** *good judgment*
7. **行为得体** *tactful and diplomatic*
8. **有说服力** *perspective*
9. **编程经验** *software programming*
10. **计算机领域外相关知识** *expert in non-computer field*

二、软件开发过程

1. The Software Development Process[1]

> *The **specifics** of **what these people do**, how they interact, and **how they make decisions** are all part of the software development process.*

2. The Software Development Process[2]

- What ***major components*** go into a software product
- What ***different people*** and ***skills*** contribute to a software poduct
- How software progresses from an idea to ***a final product***

3. Product Components

- ***What Efforts*** Goes Into a Software Porduct ?
- ***What Parts*** Make Up a Software Product ?

4. What Efforts ?

> *A lot of **hidden effort** goes into a software product.*

1. **产品规格** *Product Specification*
2. **产品评估** *Product Reviews*
3. **设计文档** *Design Documents*
4. **日程计划** *Schedules*
5. **旧版反馈** *Feedback from Previous Versions*
6. **竞争信息** *Competitive Information*
7. **测试计划** *Test Plans*
8. **用户调研** *Customer Surveys*
9. **可用性数据** *Usability Data*
10. **产品外观** *Look and Feel Specificaitons*
11. **软件架构** *Software Architecture*
12. **软件代码** *Software Code*

5. Customer Requirements

> *To properly fill that need, the product development team must **find out what the customer wants**.*

- Competitive product information
- Magazine reviews
- Focus groups
- Numerous other methods, some formal, some not

6. Specifications

- The specifications take all this information plus any unstated but mandatory requirements and ***truly define what the product will be***, what it will do, and how it will look.

7. Scheduling

The Goals of Scheduling are to know:

- which work has been ***completed***
- how much work is ***still left to do***
- when it will all be ***finished***

8. Software Design Documents

1. **架构** *Architecture*
2. **数据流图** *Data Flow Diagram*
3. **状态转换图** *State Transition Diagram*
4. **流程图** *Flowchart*
5. **注释代码** *Commented Code*

9. Test Documents

1. **测试计划** *Test Plan*
2. **测试用例** *Test Cases*
3. **Bug报告** *Bug Reports*
4. **测试工具及其自动化** *Test Tools and Automation*
5. **度量，统计，总结** *Metrics，Statics，and Summaries*

10. What Parts？

> *The software CD-ROM is just **one of the many pieces** that make up a software product.*

1. **帮助文件** *Help Files*
2. **用户手册** *User's Manual*
3. **样本和示例** *Samples and Examples*
4. **产品支持信息** *Product Support Info*
5. **错误信息** *Error Messages*
6. **安装** *Setup and Installation*
7. **标签和不干胶** *Labels and Stickers*
8. **图标和标志** *Icons and Art*
9. **广告和宣传材料** *Ads and Marketing Material*
10. **说明文件** *Readme File*

另外别忘了错误提示信息（Error Messages）！

11. 软件项目成员 Software Project Staffs

1. **项目经理、程序经理**或者**监制人员** *Project Managers*，*Program Managers* or *Producers*
2. **架构师**或者**系统工程师** *Architects* or *System Engineering*
3. **程序员**、**开发人员**或者**代码制作者** *Programmers*，*Developers* or *Coders*
4. **测试员**或者**质量保证（QA）员** *Tester* or *QA（Quality Assurance）Staff*
5. **技术作者**、**用户协助专员**、**用户培训专员**、**手册编写者**或者**文案专员** *Technical Writers*，*User Assistance*，*User Education*，*Manual Writers* or *Illustrators*
6. **配置管理员**或者**构建员** *Configuration Management* or *Builder*

12. 软件开发生命周期模式 Software Development Lifecycle Models

1. **大爆炸模式** *Big-Bang*
2. **边写边改模式** *Code-and-Fix*
3. **瀑布模式** *Waterfall*
4. **螺旋模式** *Sprial*
5. **敏捷软件开发** *Agile Software Development*

13. 测试驱动开发 Test-Driven Development，TDD

- **基本思想**：*在开发功能代码之前，**先编写测试代码**，然后**只编写使测试通过的功能代码**，从而以测试来驱动整个开发过程的进行*
- **优点**：*有助于**编写简洁可用和高质量的代码**，有很高的**灵活性和健壮性**，能**快速响应变化**，并**加速开发过程***
- **口号**：***不可运行** -> **可运行** -> **重构***

14. “V”模型

- 测试过程中**存在不同级别**
- 各测试阶段与开发过程中存在 **各阶段的对应关系**

---

1. 程序是否满足***软件设计的要求***
2. 质量特性是否达到***系统要求的指标***
3. 软件的实现是否满足***用户需要***

- **缺陷**：忽视了测试对***需求分析、系统设计***的验证

15. “W”模型

- 增加了软件各开发阶段中**应同步进行的验证和确认活动**
- 明确了**测试与开发的并行性**

---

1. 伴随***整个软件开发周期***
2. ***需求、设计和功能***同样要测试
3. 一旦有文档提供，就要***及时确定测试的条件、编写测试用例***

- **缺陷1**：需求、设计、编码等活动被视为***串行的***，同时，测试和开发活动也保持着一种***线性的前后关系***，上一阶段完全结束，才可正式开始下一个阶段工作。
- **缺陷2**：无法支持***迭代、自发性***以及***变更调整***

16. "H"模型

- 软件测试是一个**独立的进程**，贯穿产品整个生命周期，与其他流程**并发的进行**
- 尽早准备，尽早执行

17. “X”模型

- 针对**单独的程序片段**进行**相互分离的编码和测试**，此后通过频繁的交接，通过集成**最终合成为可执行的程序**

三、软件测试的实质

- The **Realities** of Software Testing

1. 完全测试程序是不可能的 Impossible to Test a Program Completely

- **输入**量太大
- **输出**结果太多
- 软件**执行路径**太多
- 软件**说明书太主观**（a **Bug** in the eye of the beholder）

2. 软件测试是有风险的行为 Risk-Based Exercise

- **1. 减少测试规模到可控范围**
- how to ***reduce*** the huge domain of possible tests ***into a manageable set***

- **2.** 针对**风险**与**测试重要性**作出**明智选择**
- how to ***make wise risk-based decisions*** on what's ***important*** to test and what's no

3. 测试无法显示潜伏的软件缺陷 Testing Can't Show Bugs that Don't Exist

- 无法保证缺陷不存在
- You can ***only continue*** your testing and ***possibly find more***

4. 找到的缺陷越多，存在的缺陷也就越多

- ***Programmers*** have ***bad days***
- ***Programmers*** often make the ***same mistake***
- Some bugs are really just the ***tip of the iceberg***

- 缺陷的**“传递”**和**“放大”**
- 找到软件**缺陷越多**的模块，**遗留缺陷**也就**越多**
- 如果某软件无论如何也找不出软件缺陷，也可能是软件经过精心编制，确实存在极少的软件缺陷

5. 杀虫剂怪事 The Pesticide Paradox

- Software ***undergoing the same repetitive tests*** eventually ***builds up resistance*** to them !
- 必须不断编写不同的新的测试程序

6. 并非所有缺陷都要修复

- **没有足够时间** *Not enough time*
- **不算真正的软件缺陷** *Really not a bug*
- **风险太大** *Too risky to fix*
- **不值得修复** *Just not worth it*

---

e.g. 相互关系

1. **不可能进行完全的测试**
2. **由于1，测试是有风险的**，即测试会遗漏软件缺陷
3. **由于2，测试不能证明程序无错**，而仅能证明程序有错
4. **由于1、2、3，测试发现的错误越多，则说明软件的错误越多。**错误密度大，遗漏的也多。

---

7. 缺陷何时成为缺陷难以定义 When a Bug's a Bug Is Difficult to Say

1. ***功能缺失*** Doesn't do something it should do.
2. ***错误、缺陷*** Does something it shouldn't do.
3. ***功能多余*** Does something it doesn't mention.
4. ***对隐性需求的把握，同时发现需求的遗漏*** Doesn't do something it doesn't mention but should.
5. ***用户体验的角度*** Difficult to understand, hard to use, or will be viewed by the end user as just plain not right.

8. 产品说明书从没有最终版本 Product Specifications Are Never Final

- As a software tester, you must assume that the spec will change. Features will be added that you didn't plan to test. ***Features will be changed or even deleted that you had already tested and reported bugs on.*** It will happen.

9. 软件测试员在产品小组中不受欢迎

- **早点找出缺陷** *Find bugs early*
- **控制情绪** *Temper your enthusiasm*

- **不要总是报告坏消息** *Don't just report bad news*

10. 软件测试是一项讲究条理的技术专业

- Software Testing is a ***Disciplined Technical Profession***

11. 所有的测试都应追溯到用户需求

- 事实上，**需求是驱动整个研发过程的源头**，不仅仅设计、开发要需求驱动，测试更是要需求驱动

12. 所有测试活动都应该是有计划的，并且计划能够得到保障

- 严格执行测试计划,**排除**测试的**随意性**

13. Good-Enough原则

- 测试要**权衡"投入－产出"比**，即要充分也不要过分。不充分的测试是不负责任的；过分的测试是一种资源的浪费，同样也是一种不负责任的表现
- **过分测试**指**无意义的重复测试**和需要消耗过大投入的**非关键性测试**

14. 制定最低测试通过标准

- 对于相对复杂的产品或系统来说，***"zero-bug"***是一种理想，***"good-enough"***是我们的原则
- 寻求合适的测试策略，在质量和成本之间寻求合适的平衡点

15. 软件测试应遵循Parito法则（二八法则）

- 80%的故障存在于20%的代码中
- 80%的故障归因于20%的故障原因
- 关注测试中的群集现象
- 关注发现缺陷较多的代码

16. 尽早地和不断地进行软件测试

17. 测试应由小到大，从小规模到大规模

18. 程序员避免测试自己的程序

- 为了达到最佳效果，应由**独立于开发的专门测试人员**来构造测试

19. 谁来完成测试？

1. 开发者测试
2. 对等测试
3. 独立测试小组
4. 独立测试机构

20. 独立测试的好处

1. 客观性
2. 专业性
3. 权威性
4. 资源有保证

21. 验证（Verification）和确认（Validation）

- **验证** *Verification* -> **评审**
- 构造的产品正确吗？保证软件符合产品说明书的过程

- **确认** *Validation* -> **测试**
- 保证软件满足用户要求的过程

22. QC和QA

- **质量控制** *Quality Control*
- 验证产品的正确性，当发现与设计不一致的时候进行纠正

- **质量保证** *Quality Assurance*
- 充当支持执行全面质量管理的角色

23. 测试和QA

- **软件测试员** *QC*
- 检测软件产品
- 找出缺陷
- 评价软件质量

- **质量保证人员** *QA*
- 检测软件过程
- 创建和加强促进软件开发并防止软件缺陷的标准、方法和过程

24. SQA与软件测试的关系

- ***Software Quality Assurance***

- SQA是**管理工作**、审查对象是**流程**、强调以**预防为主**
- 测试是**技术工作**、测试对象是**产品**、主要是**事后检查**

- SQA指导测试、监控测试
- 测试为SQA提供依据

四、黑盒测试

1. 黑盒测试定义

- **功能**测试
- **数据驱动**测试
- **基于规格说明书**的测试

- 一种**从用户观点出发的测试**

2. 主要测试的错误类型

1. 不正确或遗漏的**功能**
2. **接口、界面**错误
3. **性能**错误
4. **数据结构**或**外部数据访问**错误
5. **初始化**或**终止条件**错误

3. 对程序的功能性测试要求

1. 每个 ***软件特性*** 必须被一个测试用例或被认可的异常所 ***覆盖***
2. 利用数据类型和数据值的 ***最小集测试***
3. 测试超负荷和其他 ***最坏情况*** 的结果
4. 测试排斥 ***不规则输入*** 的能力
5. 测试影响 ***性能*** 的关键模块

- 优点：可以证明产品是否达到用户要求的功能，**符合用户的工作要求**

- 缺点：需要**充分了解**测试软件产品所用到的**各项技术**，测试用例**数量较大/冗余**，覆盖范围**不可能达到100%**，手工测试操作、负责大量文档

4. 黑盒测试的实施过程

- 测试 ***计划*** 阶段
- 测试 ***设计*** 阶段

---

1. **软件功能划分**

2. **设计测试用例**

- 测试 ***执行*** 阶段
- 测试 ***总结*** 阶段

5. 测试用例

- 测试用例就是一个 ***文档***，描述 ***输入、动作、或者时间和一个期望的结果***
- 目的：***确定*** 应用程序的某个特性 ***是否正常工作***

6. 黑盒测试用例设计技术

- ***等价类划分*** 方法

---

1. 把 ***所有可能的输入数据***（即程序的输入域）划分成 ***若干子集***，然后从每一个子集中 ***选取少数具有代表性的数据*** 作为测试用例
2. 划分： ***有效等价类*** 和 ***无效等价类***
3. 标准：***完备测试，避免冗余***
4. 划分有效等价类的六条原则：
5. 设计测试用例

```
1. 0~100
2. =9 !=9
3. Yes or No
4. 枚举
5. 一个符合规则、若干个从不同角度违反规则
6. 进一步划分为更小的等价类
```

```
1. 每一个等价类 -> 唯一的编号
2. 尽可能多地覆盖尚未被覆盖的有效等价类
3. 仅覆盖一个尚未被覆盖的无效等价类
```

- ***边界值分析*** 方法

---

1. 是对 ***等价类划分*** 方法的 ***补充***
2. 这个等价类的 ***每个边界*** 都要作为测试条件
3. 使输出值达到 ***边界值*** 及其 ***左右值***

- ***错误推测*** 方法

---

1. 基于 ***经验和直觉推测*** 程序中所有可能存在的各种错误
2. ***列举出所有可能*** 有的错误和容易发生错误的特殊情况，根据他们选择测试用例

> 例如：测试一个对线性表（比如数组）进行排序的程序，可推测列出以下几项需要特别测试的情况：
>
> 1）输入的线性表为空表
> 2）表中只含有一个元素
> 3）输入表中所有元素已排好序
> 4）输入表已按逆序排好
> 5）输入表中部分或全部元素相同

- ***因果图*** 方法

---

1. **Cause-Effect Graphics**
2. 关系：恒等、非、或、与
3. **ci表示原因**，置于图的左部；**ei表示结果**，置于图的右部
4. 0表示状态不出现，1表示某状态出现
5. 约束：**异** *E*，**或** *I*，**唯一** *O*，**要求** *R*，**强制** *M*

- ***判定表*** 方法

五、白盒测试

1. 相关概念

- 又称 ***结构测试*** 或 ***逻辑驱动测试***
- 一种 ***测试用例设计方法***
- 从 ***程序的控制结构*** 导出测试用例

2. 白盒测试要求

1. 所有 ***独立路径*** 都至少被执行一次
2. 对所有逻辑值需要测试 ***真、假两个分支***
3. 在 ***上下边界*** 以及 ***可操作范围内*** 运行所有循环
4. 检查内部数据结构以 ***确保其有效性***

3. 测试覆盖标准

1. 必须有程序的 ***规格说明*** 以及 ***程序清单***
2. 考虑的是测试用例对程序内部逻辑的 ***覆盖程度***
3. 只能希望 ***覆盖的程度尽可能高些***

- ***语句*** 覆盖
- ***判定（分支）*** 覆盖
- ***条件*** 覆盖
- ***判定/条件*** 覆盖
- ***条件组合*** 覆盖
- 基本路径测试

4. 逻辑驱动测试方法[1]：语句覆盖

- 选择足够的测试用例，使得程序中 ***每个语句至少都能被执行一次***
- ***最弱*** 的逻辑覆盖，必须与其他方法 ***交互使用***

5. 逻辑驱动测试方法[2]：判定覆盖（分支覆盖）

- 执行足够的测试用例，使得程序中的 ***每个分支至少都通过一次***
- 可能不满足判定覆盖的要求

6. 逻辑驱动测试方法[3]：条件覆盖

- 程序中 ***每个判断的每个条件的每个可能取值*** 至少执行一次

7. 逻辑驱动测试方法[4]：判定/条件覆盖

- 判定中 ***每个条件*** 取到各种可能的值
- ***每个判定*** 取到各种可能的结果
- 有缺陷，往往某些条件掩盖了另一些条件

8. 逻辑驱动测试方法[5]：条件组合覆盖

- ***每个判定中条件的各种可能组合*** 都至少出现一次

9. 基本路径测试

- 设计足够多的测试用例，运行所测程序，要 ***覆盖程序中所有的可能路径***
- 最强的覆盖准则，但在路径数目很大时完全覆盖是很困难的，必须 ***把覆盖路径数目压缩到一定程度***

- 在 ***程序控制图*** 的基础上，通过分析控制构造的 ***环行（圈，loop）复杂性*** ，导出基本可执行路径集合
- 保证在测试中程序的 ***每一个可执行语句至少被执行一次***

10. 综合策略：黑盒法补充测试用例

1. 在任何情况下都需使用 ***边界值分析***
2. 必要的话，再用 ***等价类划分法*** 补充一些测试用例
3. 再用 ***错误推测法*** 附加测试用例
4. 检查上述例子的 ***逻辑覆盖程度*** ，如果未能满足某些覆盖标准，则再增加足够的测试用例
5. 如果功能说明中含有输入条件的组合情况，则一开始就可先用因果图（判定表）法

11. 基本路径测试：4个步骤

1. 程序的 ***控制流图***
2. 程序的 ***圈复杂度***
3. ***导出*** 测试用例
4. ***准备*** 测试用例

“

计算圈复杂度方法

1. 流图中 ***区域的数量*** 对应于环型的复杂性
2. 给定流图G的圈复杂度V(G)，定义为 ***V(G)=E-N+2*** ，E是流图中边的数量，N是流图中结点的数量
3. 给定流图G的圈复杂度V(G)，定义为 ***V(G)=P+1*** ，P是流图G中判定结点的数量

12. 图形矩阵

- 连接权为“1”表示存在一个连接，在图中如果一行有两个或更多的元素“1”，则这行所代表的结点一定是一个判定结点，通过连接矩阵中有两个以上（包括两个）元素为“1”的个数，就可以得到确定该图圈复杂度的另一种算法

六、静态测试

1. 静态测试方法

1. ***代码审查***： 第三方测试
2. ***代码走查***： ***发现错误*** 而不是纠正错误
3. ***桌面检查***： 程序员自己阅读自己所编的程序
4. 主要由软件工具自动进行的 ***静态分析***
5. 广义的理解，包括 ***技术评审***

2. 静态测试内容

1. ***需求定义***
2. ***设计文档***
3. ***源代码***

3. 静态测试定义

1. 通过 ***检查*** 和 ***评审*** 软件而 ***不是运行软件*** 对软件进行测试的方法
2. 可以 ***手工进行***，也可以 ***借助测试工具自动进行***
3. 最多识别软件所有缺陷中 ***50%~70%的缺陷***

4. 代码审查

- ***代码审查组***

---

1. 资深程序员
2. 程序编写者
3. 专职测试人员

- 代码审查的 ***步骤***

---

1. 准备
2. 程序阅读
3. 审查会
4. 跟踪及报告

- 至少要读程序4次

1. 印刷错误
2. 数据结构
3. 控制流
4. 处理

5. 代码走查

1. 不是读程序和使用代码审查单
2. 对每个测试用例 ***用头脑执行***，也就是 ***用测试用例沿程序逻辑走一遍***

6. 需求定义的软件质量因素

1. 完备性： 是否包含 ***所有内容***？
2. 一致性： 是否有 ***冲突矛盾***？
3. 正确性： 是否满足 ***标准的要求***？
4. 可行性： 是否 ***可行***？
5. 易修改性： 是否 ***易于修改***？
6. 健壮性： 是否有 ***容错的需求***？
7. 易追溯性： 是否可从 ***上一阶段*** 查找相应内容？
8. 易理解性： 是否每个需求只有 ***一种解释***？
9. 易测试性和可验证性： 是否 ***可以验证***？
10. 兼容性： 是否使软硬件系统具有 ***兼容性***？

7. 设计文档的静态测试

- 分析设计是否 ***与需求定义一致***

---

1. 完备性
2. 一致性
3. 正确性
4. 可行性
5. 易修改性
6. 模块性
7. 可预测性
8. 健壮性
9. 结构化
10. 易追溯性
11. 易理解性
12. 可验证性/易测试性

8. 源代码的静态测试

1. 完备性
2. 一致性
3. 正确性
4. 易修改性
5. 可预测性
6. 健壮性
7. 结构化
8. 易追溯性
9. 易理解性
10. 可验证性

七、单元测试

1. 定义

1. ***针对*** 最小的可测试软件元素——***模块进行的正确性测试工作***
2. 又称 ***模块测试***

2. 范畴

1. 测试模块功能 ***符合期望***
2. 确认 ***任何情况下*** 都符合期望
3. 验证 ***代码可靠性***
4. 了解 ***代码的用法***

3. 何时进行单元测试?

- 在编码阶段进行
- 在后续软件生命周期中，单元测试仍将持续

4. 单元测试环境

1. 辅助模块

- 驱动模块： 所测模块的 ***主程序***
- 桩模块： 代替所测模块 ***调用的子模块***

5. 单元测试内容

1. 模块接口测试： ***I/O参数、文件属性等***
2. 重要路径测试： ***控制结构***
3. 边界条件测试
4. 错误处理测试
5. 局部数据结构测试

八、集成测试

1. 定义

- 在单元测试的基础上，***将所有模块按照设计要求组装成子系统或系统*** 而进行的测试活动
- 又称 ***组装测试***

2. 集成测试的内容

1. ***单元间的接口***
2. ***集成后的功能***

3. 集成测试的层次

1. ***模块内*** 集成测试
2. ***子系统内*** 集成测试
3. ***子系统间*** 集成测试

4. 集成测试方法

1. 静态测试技术： ***针对概要设计*** 的测试
2. 动态测试技术： ***灰盒测试***
3. 灰盒测试优点

---

1. 能够进行 ***基于需求的测试*** 和 ***基于路径的覆盖测试***
2. 可 ***深入被测对象的内部***
3. 保证黑盒测试 ***用例的完整性***
4. 能够 ***减小需求或设计对测试有效性造成影响***

5. 集成策略

1. ***非增量式*** 集成策略： ***一步到位***
2. ***增量式*** 集成策略： ***逐步实现***

6. 非增量式集成策略

- 又称大爆炸式集成 ***Big Bang***
- 优点

1. ***方法简单***
2. 允许 ***并行工作***，资源 ***利用率高***

- 缺点

---

1. 测试 ***成本较高***
2. 一旦集成后包含多种错误，***难以纠正***

7. 增量式集成策略

- 是 ***逐步实现的***
- 三种不同方法

---

1. ***自顶向下***
2. ***自底向上***
3. ***三明治增量式***（ ***混合增量式*** 测试 ）

8. 自顶向下式增量测试

- 优点

---

1. 较早验证 ***主要控制*** 和 ***判断点***
2. 按深度优先可以 ***首先实现和验证一个完整的软件功能***
3. 功能较早实现，带来信心
4. 只需一个驱动，减少驱动器开发费用
5. 支持 ***故障隔离***

- 缺点

---

1. 桩模块的开发量大
2. 底层验证被推迟
3. 底层组件测试不充分

9. 自底向上式增量测试

- ***逐层向上集成***
- ***最常用*** 的集成策略

- 优点

---

1. 对 ***底层组件行为*** 较早验证
2. 可以 ***并行集成***，效率高
3. 减少了桩的工作量
4. 能较好的 ***锁定软件故障所在位置***

- 缺点

---

1. 驱动的开发工作量大
2. 高层验证被推迟，设计上的错误不能及时发现

10. 三明治方法（混合增量式测试）

- ***目标层*** 之上采用自顶向下集成，之下采用自底向上集成
- 优点： ***集合了*** 两种策略的 ***优点***
- 缺点： ***中间层*** 测试不充分
- 适用范围： 大部分软件开发项目

九、单元测试框架

1. XUnit家族

1. JUnit： ***Java***
2. NUnit： ***.NET***
3. CppUnit： ***C++***
4. PHPUnit： ***PHP***
5. SQLUnit： ***SQL***
6. PythonUnit： ***Python***
7. DUnit： ***Dephi***

2. JUnit单元测试框架

- ***testXXX()测试方法*** 必须满足以下几个条件

---

1. ***public***
2. ***void***
3. ***无方法参数***
4. 方法必须 ***以test开头，后面的部分自定义***

```
public class FirstTestExample{

  int add(int a,int b){

    return a+b;

  }

  public static void main(String[] args){

    FisrtTestExample fta = new FirstTestExample();

    System.out.println("Add value is:"+fta.add(2,3));

  }

}
```

```
import junit.framework.TestCase;

public class FirstTestExampleTest extends TestCase{

  public void testAdd(){

    assertEquals(5,new FirstTestExample().add(2,3));

  }
}
```

- Junit框架让我们 ***继承TestCase类*** ，用Java来编写自动执行、自动验证的测试。这些测试在JUnit中称作 ***测试用例***
- JUnit ***提供一个机制能够把相关测试用例组合*** 到一起，称之为 ***测试套件（test suite）***
- JUnit还提供了一个 ***运行器*** 来执行一个测试套件

3. JUnit的益处

1. 提高 ***开发效率*** 与测试代码的 ***执行效率***
2. 实用小版本发布至整个系统集成， ***便于除错***
3. 引入 ***重构概念***，让代码更加 ***干净、富有弹性***
4. 提升系统的 ***可信赖度*** ： 它是回归测试的一种，支持修复或更正后的 ***再测试***，可确保代码的正确性

4. 测试置具Test Fixture

- 把这样的代码分离出来单独写，让所有的测试都可以利用这些对象代码
- 在JUnit框架里，把这些对象称为 ***测试置具***

十、测试驱动开发

1. 测试驱动的概念

- 测试驱动是一种 ***开发形式***

---

1. 首先要 ***编写测试代码***
2. 除非存在相关测试，否则 ***不编写任何产品代码***
3. ***由测试来决定*** 需要编写什么样的代码
4. 要求 ***维护*** 一套详尽的 ***测试集***

2. 测试驱动的目标

- ***Clean Code that Work***

---

1. 只有测试失败，我们才写代码
2. 消除重复设计，优化设计结构

3. 测试驱动的过程

- ***Start***

---

1. ***Write a test for new capability***
2. ***Compile***
3. ***Fix compile errors***
4. ***Run the test and see it fails***
5. ***Write the code***
6. ***Run the test and see it pass***
7. ***Refactor as needed***
8. [Back to ***Point 1***]

十一、软件系统测试

1. 测试生命周期 Life Cycle Testing

1. ***用户需求***
2. ***软件需求***
3. ***体系结构设计***
4. ***详细设计***
5. ***编码实现***
6. ***单元测试***
7. ***集成测试***
8. ***系统测试***
9. ***验收测试***

2. 系统测试定义

- 将 ***经过集成测试的软件***，作为计算机系统的一个部分，***与系统中其他部分结合起来***

3. 性能测试

1. 评估系统 ***能力***
2. 识别系统中的 ***弱点***
3. 系统 ***调优***

4. 压力测试

1. ***模拟工作负荷*** 以 ***检验系统*** 在峰值使用情况下 ***是否可以正常运行***
2. 通过逐步增加系统负载来测试系统性能的变化，并最终确定在什么负载条件下系统性能处于失效状态，以此来 ***获得系统性能提供的最大服务级别*** 的测试

5. 容量测试

- 首要任务是 ***确定被测系统的容量极限***
- 检测系统能够 ***承载处理任务的极限值***
- 使系统 ***承受超额的数据容量*** 来检测它是否能够正确处理

6. 容量测试与压力测试的区别

1. 都是检测 ***特定情况下系统能够承担的极限值***
2. 压力测试侧重 ***让系统承受速度方面的超额负载***，例如短时间内的吞吐量

3. 容量测试侧重 ***数据方面的承受能力***，目的是 ***显示系统可以处理的数据容量***

7. 健壮性测试

- 用于测试系统 ***抵御错误的能力***

---

1. ***高可靠性***
2. 从错误中 ***恢复的能力***

8. 安全性测试

1. 检查系统 ***对非法侵入的防范能力***
2. 目的是为了 ***发现软件系统中是否存在安全漏洞***

9. 恢复性测试

1. 检查系统的 ***容错能力***
2. 用各种办法 ***强迫系统失败***，然后 ***验证系统是否能尽快恢复***

10. 备份测试

1. 恢复性测试的 ***补充***
2. ***验证*** 系统发生软件或者硬件失败时 ***备份数据的能力***

11. 兼容性测试

1. 检查软件之间是否能够正确地交互和共享信息
2. 方法： 确定兼容性测试标准

12. 安装性测试

1. 验证系统成功安装的能力
2. 保证程序安装后能正常运行

13. Alpha测试

1. 是由 ***一个用户在开发环境下进行的测试***，也可以是 ***公司内部的用户在模拟实际操作环境下进行的测试***
2. 目的： 评价软件产品的FLURPS
3. ***非正式验收测试***

14. Beta测试

1. 最终用户们在 ***一个或多个客户场所*** 进行
2. 开发者通常不在Beta测试的现场

15. 回归测试

1. 在 ***软件发生变动时*** 保证 ***原有功能正常运作*** 的一种测试策略和方法
2. 不需要进行全面的测试，而是 ***根据修改的情况进行有选择的测试***

十三、软件性能测试

1. 性能测试内容

1. ***评估系统能力***
2. ***识别体系弱点***
3. ***系统调优***

2. 性能测试目标

1. ***评价系统当前性能***
2. ***寻找瓶颈，优化性能***
3. 预测 ***未来性能*** 及 ***可扩展性***

3. 性能测试方法

1. ***性能测试*** ： *Performance Testing*
2. ***负载测试*** ： *Load Testing*
3. ***压力测试*** ： *Stress Testing*
4. ***并发测试*** ： *Concurrency Testing*

4. 性能测试流程

1. 测试 ***前期准备***
2. 测试 ***需求 & 计划***
3. 测试 ***设计 & 开发***
4. 测试 ***执行***
5. 结果 ***分析 & 报告***

十四、软件测试环境

1. 软件测试环境定义

1. 包括 ***设计环境*** ，***实施环境*** 和 ***管理环境***三部分，是指为了完成软件测试工作所必需的硬件、软件、设备、数据的总称
2. 测试环境适合与否严重影响测试结果的 ***真实性*** 和 ***正确性***

2. 测试环境的要素

1. 硬件、软件、网络环境、数据准备、测试工具
2. ***硬件、软件*** 是测试环境中最基本的两个要素，并 ***派生出后三者***

3. 数据生成器DataFactory

1. DataFactory 是一种快速的、易于产生测试数据工具，它能建模复杂数据关系，且带有GUI界面
2. DataFactory是一个功能强大的数据产生器，它允许开发人员或测试人员 ***毫不费力地产生百万行有意义的测试数据***

十五、软件测试计划

1. 软件测试计划定义

- 是 ***软件测试员*** 与 ***产品开发小组*** 交流意图的主要方式

2. 软件测试计划的目标

1. 规定测试活动的 ***范围*** 、***方法*** 、***资源*** 和 ***进度***
2. 明确 ***正在测试的项目*** 、***要测试的特性*** 、***要执行的测试任务*** 、***每个任务的负责人*** ，以及 ***与计划相关的风险***

3. 测试计划制订过程

1. 分析和测试 ***软件需求***
2. 定义 ***测试策略***
3. 定义 ***测试环境***
4. 定义 ***测试管理***
5. ***编写和审核*** 测试计划

4. 定义工作进度过程

1. 确认工作任务
2. 估算工作量
3. 编写进度计划

十六、软件缺陷管理

1. 软件缺陷的定义

1. 软件缺陷（***Defect***），又称 ***Bug***
2. 某种 ***破坏正常运行能力的问题、错误***，或者 ***隐藏的功能缺陷***

2. 如何面对缺陷？

- 确保发现的软件缺陷全部被关闭，但 ***不一定被修复***

---

1. 没有足够的时间
2. 不算真正的软件缺陷
3. 修复的风险太大
4. 不值得修复

- 尽快报告软件缺陷
- 有效描述软件缺陷

十七、软件配置管理

1. 软件配置管理定义 Software Configuration Management，SCM

1. 通过执行 ***版本控制、变更控制*** 等规程
2. 使用合适的 ***配置管理软件***
3. 保证所有配置项的 ***完整性*** 和 ***可跟踪性***
4. 一种 ***对工作成果的有效保护***

2. 配置项 Configuration Item

- 定义： 软件配置管理的对象，配置管理的 ***基本单位***
- 配置项包括：

---

1. 与合同、过程、计划、产品有关的 ***文档和数据***
2. ***源代码、目标代码、可执行代码***
3. ***相关产品***，包括软件工具、库内的可利用软件、外购软件以及用户提供的软件

3. 版本 Version

1. 定义： 某一 ***配置项*** 的 ***已标识了的实例***；或不可变的源对象经质量检查合格后所形成的 ***新的相对稳定*** 的格局（配置）
2. ***每个软件对象可具有一个版本组***，它们彼此间具有特定的关系，这种关系用以描述其演变情况，通常软件对象的版本组呈树形结构

4. 版本控制 Version Control

- 定义： 版本控制就是 ***管理*** 在整个软件生存周期中建立起来的 ***某一配置项的不同版本***

5. 基线 Baseline

1. 基线指一个配置项在其生存周期的某一特定时间，被正式标明、固定并 ***经正式批准的阶段性版本***
2. 基线是软件生存周期中 ***各开发阶段末尾的特定点***，又称 ***里程碑***
3. 只有 ***由正式技术评审*** 而得到的 ***软件配置项协议*** 和 ***软件配置的正式文本*** 才能成为 ***基线***
4. 基线的作用： 使各阶段 ***工作的划分更加明确化***；便于 ***检验和肯定阶段成果***

6. 配置控制组/委员会 Configuration Control Board

- 一组负责 ***评估和审批配置项变更*** 的人员，以确保所有的变更都是经过审核的

7. 变更管理 Change Management

1. 控制和协调不同责任的软件开发人员进行有效的交流
2. 使开发人员不会在无序环境下各自为战

8. 常用配置管理工具

1. ***Visual SourceSafe***
2. ***Subverison*** ： ***CVS*** 的替代产品
3. ***IBM Rational ClearCase***

9. 实施配置管理的好处

1. ***版本*** 得到很好的 ***控制***
2. ***变更*** 的处理 ***更规范***
3. 可以保证 ***产品*** 的 ***一致性*** 和 ***完整性***
4. 项目更 ***易于管理***

十八、软件维护

1. 软件维护的定义

- 软件系统 ***交付使用以后***，为了 ***改正错误或满足新的要求*** 而 ***修改软件*** 的过程
- 维护的原因有以下三点：

---

1. ***改正错误和缺陷***
2. ***改进设计*** 以 ***适应*** 新的软件、硬件 ***环境***
3. 增加新的应用范围

2. 软件维护的类型

1. ***改正性维护***
2. ***适应性维护***
3. ***完善性维护***
4. ***预防性维护***

3. 改正性维护

- ***诊断*** 和 ***改正错误*** 的过程

---

1. ***识别和纠正软件错误***
2. ***改正性能缺陷***
3. ***排除*** 使用中的 ***误使用***

4. 适应性维护

- 为使软件 ***适应变化而去修改软件*** 的过程
- 变化包括 ***外部环境*** 以及 ***数据环境***

5. 完善性维护

- 用户提出 ***新的功能与性能要求***
- 扩充软件 ***功能***
- 增强软件 ***性能***
- 改进加工 ***效率***

6. 预防性维护

- 又称 ***软件再工程***
- 对软件中的某一部分 ***重新进行设计、编制和测试***

7. 衡量程序可维护性的七个特性

1. 可理解性
2. 可使用性
3. 可测试性
4. 可移植性
5. 可修改性
6. 可靠性
7. 效率