第五讲：单元测试与集成测试

LIAN YU
THE SCHOOL OF SOFTWARE AND MICROELECTRONICS
PEKING UNIVERSITY
NO.24 JINYUAN RD, BEIJING 102600

提　纲

1. 引言
2. 单元测试
   - -单元测试考虑事项
   - -单元测试规程
   - -单元测试局限性
3. 集成测试
   - -自顶向下/自底向上集成
   - -混合式集成
   - -端到端集成测试
4. 总结

导言(1/2)

- 按阶段进行测试是一种基本的测试策略。
  - 最开始，测试着重于每一个单独的模块，以确保每个模块都能正确执行，所以，我们把它叫做单元测试。
    - 单元测试大量地使用白盒测试技术，检查每一个控制结构的分支以确保完全覆盖和最大可能的错误检查；
  - 接下来，模块必须装配或集成在一起形成完整的软件包，集成测试解决的是功能验证与程序构造的**双重问题**，在集成过程中使用最多的是黑盒测试用例设计技术。当然，为了保证覆盖一些大的分支，也会用一定数量的白盒测试技术；

导言(2/2)

- 在软件集成(构造)完成之后，一系列高级测试就开始了。最后的高级测试步骤已经跳出了软件工程的边界，而属于范围更广的计算机系统工程的一部分，软件一旦经过验证之后，就必须和其他的系统元素(比如硬件、人员、数据库)结合在一起。
- 系统测试要验证所有的元素能正常地啮合在一起，从而完成整个系统的功能/性能。
  - 确认标准(在需求分析阶段就已经确定了的)必须进行测试，确认测试提供了对软件符合所有功能的、行为的和性能的需求的最后保证，在确认过程中，只使用黑盒测试技术。

提纲

- 导言
- 单元测试
  - 单元测试考虑事项
  - 单元测试规程
  - 单元测试局限性
- 集成测试
  - 自顶向下集成
  - 自底向上集成
  - 混合式集成
  - 端到端集成测试

单元测试

- 单元测试（Unit testing）是对最小的软件设计单元（模块或源程序单元）的验证工作。
  - 用更学术的方式讲，我们应该把一个单元理解成一个应用程序中最小可测部分。
- 在面向过程的设计（Procedural Design）里，一个单元可能是单独的程序、函数、过程、网页以及菜单等。但在面向对象的设计（Object Oriented Design）里，最小单元永远是类，可能是基/父类、抽象类或派生/子类。

单元测试

- 单元测试使用构件级别的设计规格说明书作为指南，对重要的控制路径进行测试以发现模块内的错误。
- 单元测试把重点放到内部处理逻辑和构件边界内的数据结构。这种测试可以对多个构件并行进行。
- 通常情况下，单元测试是由开发者执行测试而不是由最终用户执行测试，主要使用白盒测试技术，并辅助使用黑盒测试技术，如边界值分析法。

单元测试考虑事项

- 单元测试对构件的五方面进行测试：
  - (1)模块或构件接口；
  - (2)局部数据结构；
  - (3)边界条件；
  - (4)独立路径；和
  - (5)处理错误的路径。

模块或构件接口

- 对模块接口的测试保证在测试时进出程序单元的数据流是正确的，
  - 包括接口名称，传入参数的个数、类型、顺序等是否与模块接口匹配；
  - 模块输出或返回值或类型是否正确。
  - 对穿越模块接口的数据流的测试需要在任何其他测试开始之前进行，如果数据不能正确地输入和输出的话，所有的其他测试都是没有实际意义的。

局部数据结构

- 对局部数据结构的检查保证临时存储的数据在算法执行的整个过程中都能维持其完整性。
- 另外，应该测试局部数据结构，并在单元测试时确认对于全局数据的局部影响。

边界条件

- 对边界条件的测试以保证模块在所限定或约束处理的条件边界上能够正确执行。
- 边界测试是单元测试任务的一项重要步骤。软件通常是在边界情况下出现故障的，这就是说，错误往往出现在一个n元数组的第n个元素被处理的时候，或者当一个i次循环的第i次调用，或者当允许的最大或最小数值出现的时候。
  - 使用刚好小于、等于和刚好大于最大值和最小值的数据结构、控制流、数值来作为测试用例就很有可能发现错误。
- 边界条件的测试是利用黑盒测试技术中的边界值分析法。

独立路径 (1/3)

- 在控制结构中的所有独立路径(基本路径)都要走遍，以保证在一个模块中的所有语句都能执行至少一次。
  - 在单元测试过程中，对执行路径的选择性测试是最主要的任务。
  - 测试用例应当能够发现由于错误计算、不正确的比较、或者不正常的控制流而产生的错误。
- 基本路径测试和循环测试是发现更多的路径错误的一种有效技术。

独立路径(2/3)

- 计算中常见的错误有：
  - (1)误解的或者不正确的算术优先级；
  - (2)混合模式的操作；
  - (3)不正确的初始化；
  - (4)精度不够精确；
  - (5)表达式的不正确符号表示。

独立路径(3/3)

- 比较和控制流是紧密地耦合在一起的(也就是说，控制流的转移是在比较之后发生的)，测试用例应当能够发现下列错误：
  - (1)不同数据类型的比较；
  - (2)不正确的逻辑操作或优先级；
  - (3)应该相等的地方由于精度的错误而不能相等；
  - (4)不正确的比较或者变量；
  - (5)不正常的或者不存在的循环中止；
  - (6)当遇到分支循环的时候不能退出；
  - (7)不适当地修改循环变量。

处理错误的路径 (1/2)

- 要对所有处理错误的路径进行测试。好的设计要求错误条件是可以预料的，而且当错误真的发生的时候，错误处理路径被建立，以重定向或者干脆终止处理。
  - Yourdon[YOU75]把这种方法叫做反调试(antidebugging)。
  - 不幸的是，存在一种倾向，就是把错误处理过程加到软件中去，但从不进行测试。

处理错误的路径 (2/2)

- 在错误处理部分应当考虑的潜在错误有这几种情况：
  - （1）对错误描述费解。
  - （2）所报的错误与真正遇到的错误不一致。
  - （3）在错误处理之前错误条件先引起系统干涉造成系统异常。
  - （4）例外条件处理不正确。
  - （5）错误描述没有提供足够的信息来帮助确定错误发生的位置。

单元测试规程 （1/5）

- 单元测试通常看成为是附属于编码步骤。在源代码级的代码被开发、复审、和语法正确性验证之后，单元测试用例设计就开始了。
- 对设计信息的复审可能能够为建立前面讨论过的每一类错误的测试用例提供指导，每一个测试用例都应当和一系列的预期结果联系在一起。

单元测试规程 （2/5）

- 因为一个模块本身不是一个独立的（stand-alone）程序，所以必须为每个单元测试开发驱动器（driver）或/和程序桩(stub)。
- 在绝大多数应用中，一个驱动器只是一个“主程序”，负责接收测试数据，并把数据传送给(要测试的)模块，然后打印相关结果。
- 子程序桩的功能是替代那些隶属于被测模块(被调用)的模块。一个子程序桩或“空子程序”使用被调子模块的接口，可能要做一些最少量的数据操作，并打印入口处验证的信息，然后把控制返回给被测模块。
  - 在面向对象的程序里，模仿对象（mock objects）技术取代程序桩(stub) 。模仿对象是以一种可控方式来模拟真实对象行为的仿真对象。

单元测试规程 （3/5）

- 驱动器和程序桩都是额外的开销，这就是说，两种都属于必须开发但又不和最终软件一起交付的软件。
- 如果驱动器和程序桩很简单的话，那么额外开销相对来说是很低的。当一个模块被设计为高内聚时，单元测试是很简单的。

单元测试规程 （4/5）

- 单元测试通常是被自动执行，但可能仍由手工进行的。有关软件单元测试的IEEE标准[ISS87]并没有规定是用自动化的还是用手工的方法进行单元测试。
- 手工方法可以按照指示文档一步一步地进行。
- 然而，单元测试的目标是要隔离一个单元并验证其正确性，自动化方法能有效地达到这一目标，使得单元测试取得应有效果。
  - 使用自动化方法，为完全实现隔离效果，进行单元测试的代码体是在其自然环境以外的框架内运行的，也就是说，在产品以外或被原始创建时调用环境以外。

单元测试规程 （5/5）

- 因而，单元测试习惯上是作为程序员创建具有松耦合、高内聚代码体的一个动力。这种实践促进软件开发的健康习惯。
- 设计模式、单元测试和重构常结合使用以便形成最理想解决方案。

单元测试局限性

- 单元测试不能捕获程序中的每一个错误。根据定义，单元测试只测试单元自身的功能。
  - 因此它不捕获集成错误、性能问题或其它任何系统范围的问题。
  - 另外，要预料现实中被测程序可能接受到的输入的所有特殊情况是一项不易之事。
  - 对于任何非平凡的软件块要测试所用的输入组合是不现实的。

提纲

- 导言
- 单元测试
  - 单元测试考虑事项
  - 单元测试规程
  - 单元测试局限性
- 集成测试
  - 自顶向下集成
  - 自底向上集成
  - 混合式集成
  - 端到端集成测试

集成测试

- 集成测试（Integration testing），有时也称作集成与测试（I&T）是软件测试的一个阶段，在这个阶段单独的软件模块被结合在一起，作为一个群接受测试。
- 什么时候进行集成测试? 在三种情况下进行需要进行集成测试:
  - （1）由若干单元或模块要组成一个构件;
  - （2）由若干构件组成为一个工件;
  - （3）由若干工件组成为一个系统。
- 集成测试被定义为在单元测试与系统测试之间级别的测试。

接口连接问题

- 在所有的模块都已经完成单元测试之后，有人或许会问这样一个似乎很合理的问题：“如果它们每一个都能单独工作得很好，那么你为什么要怀疑把它们放在一起就不能正常工作呢？”当然，这个问题就在于“把它们放在一起”— 即接口连接问题。
  - 数据可能在通过接口的时候丢失；
  - 在连接时一个模块可能对另外一个模块产生无法预料的副作用；
  - 当子函数被联到一起的时候，可能不能达到期望的功能；
  - 在单个模块中可以接受的不精确性在联起来之后可能会扩大到无法接受的程度；
  - 全局数据结构可能也会存在问题。

两种集成测试策略

- 集成测试被看作是一种系统化技术，来构造程序并实施测试以发现与接口连接有关的错误，
  - 它的目标是把通过了单元测试的模块拿来，构造一个在设计中所描述的程序结构。
- 有两种集成测试策略：
  - 瞬时集成测试，和
  - 增量集成测试。

瞬时集成测试

- 有时候被称为“创世大爆炸（Big Bang）”的方法。由Myers在1979年定义的一种方法，当所有的被隔离的构件通过了测试，就把它们组合成一个最终系统，并观察它是否运转正常。
  - 这种方法的结果通常是混乱不堪！会遇到许许多多的错误，错误的修正也是非常困难的，因为在整个程序的庞大区域中想要分离出一个错误是很复杂的。
  - 一旦这些错误被修正之后，就马上会有新的错误出现，这个过程会继续下去，而且看上去似乎是个无限循环的。

瞬时集成方法缺点

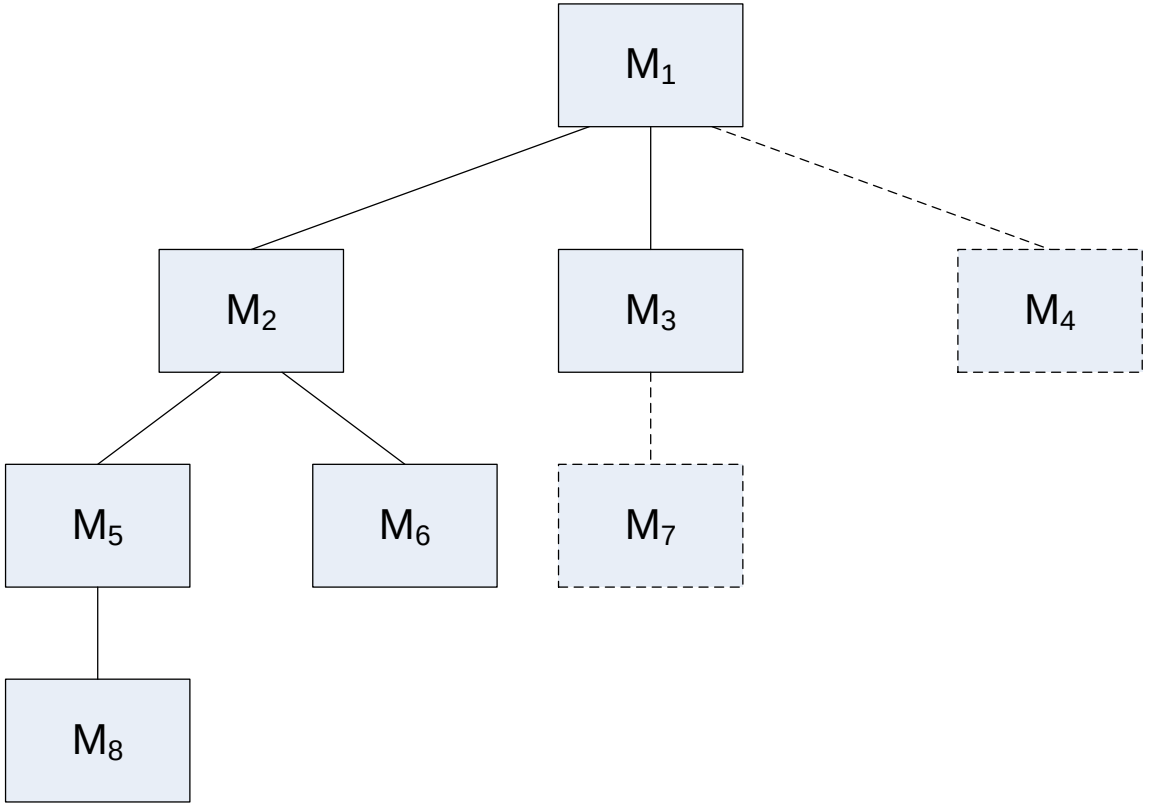
- 许多程序员在开发小程序的时候都会用到瞬时集成测试技术，但对大型程序不太适用。
- 事实上，瞬时集成方法有这样几个缺点：
  - （1）对独立组件测试需要驱动程序和树桩程序的支持；
  - （2）由于所有组件都是一次性的结合在一起，所以很难找出错误的原因；
  - （3）不容易辨别接口错误和其他类型的错误。
- 一般情况下，不推荐瞬时集成用于任意系统，而是推荐使用增量集成策略。

增量集成

- 是创世大爆炸的方法的对立面。
- 程序先分成小的部分进行构造和测试，这个时候错误比较容易分离和修正；接口也更容易进行彻底地测试；而且也可以应用一种系统化的测试方法。
  - 增量集成测试会有格外的开销，但会大大减少发现和改正错误的时间，最佳的增量方法本质上取决于各个项目和不同利弊选择的考虑。

自顶向下集成

- 自顶向下的集成是一种构造程序结构的增量实现方法。模块集成的顺序是首先集成主控模块(主程序)，然后按照控制层次结构向下进行集成。
- 隶属于(和间接隶属于)主控模块的模块按照深度优先或者广度优先的方式集成到整个结构中去。



自顶向下的集成—示例

深度优先集成及广度优先集成

- 深度优先集成是集成结构中的某一个主控路径上的所有模块。
  - 主控路径的选择是有些任意的，它依赖于应用程序的特性，例如，选择图中最左边的路径，模块M1，M2，和M5，将会首先进行集成，然后是M8; 或者是(如果对M2的适当的功能是必要的) M6，然后开始构造中间的和右边的控制路径。
- 广度优先的集成是沿着水平的方向，把每一层中所有直接隶属于上一层模块的模块集成起来，从图中来说，模块M2，M3和M4首先进行集成，然后是下一层的M5，M6，然后继续。

自顶向下及自顶向下

- 自顶向下测试采用深度优先方法时，每一模块在测试中层层地由真实代码替代程序桩，使该模块得到不断详尽的测试。
- 自顶向下测试采用广度优先方法时，应用程序中处于同一控制层上的模块在进行测试时得到精化。
- 在现实中一般是结合使用这两种技术进行测试。
  - 在初始阶段所有的模块可能只是提供部分功能，这可以用宽度优先技术进行测试，过了一段时间后，模块被越来越精化，模块的功能也越来越全，这时候就可以对一个模块进行深度优先测试而同时所有的模块进行宽度优先测试。

五个步骤

- 集成的整个过程由下列五个步骤来完成：
  1. 主控模块作为测试驱动器，所有的程序桩由直接隶属于主控模块的各模块替换。
  2. 根据集成的实现方法(如深度或广度优先)，子模块的程序桩依次地被替换为真正的模块。
  3. 在每一个模块集成的时候都要进行测试。
  4. 在完成了每一次测试之后，又一个程序桩被真正的模块替换。
  5. 可以用回归测试来保证没有引进新的错误。
- 整个过程回到第2步循环继续进行，直至这个系统结构被构造完成。

自顶向下

- 在自顶向下的测试开始的时候，程序桩代替了低层的模块，因此，在程序结构中就不会有重要的数据向上传递，测试者只有下面的三种选择：
  - (1)把测试推迟到程序桩被换成实际的模块之后再进行，
  - (2)开发能够实现有限功能的程序桩，用来模拟实际模块，或者
  - (3)从层次结构的最底部向上来对软件进行集成。

自顶向下的优缺点

- 简单总结一下自顶向下的优点：
  - （1）对高层行为进行早期确认；
  - （2）至多只需一个驱动程序；
  - （3）每步可以只加一个模块；
  - （4）支持深度优先和宽度优先。
- 自顶向下的缺点：
  - （1）对低层行为确认比较晚；
  - （2）对缺少的元素需要编写树桩程序；
  - （3）测试用例的输入和输出可能很难明确表示。

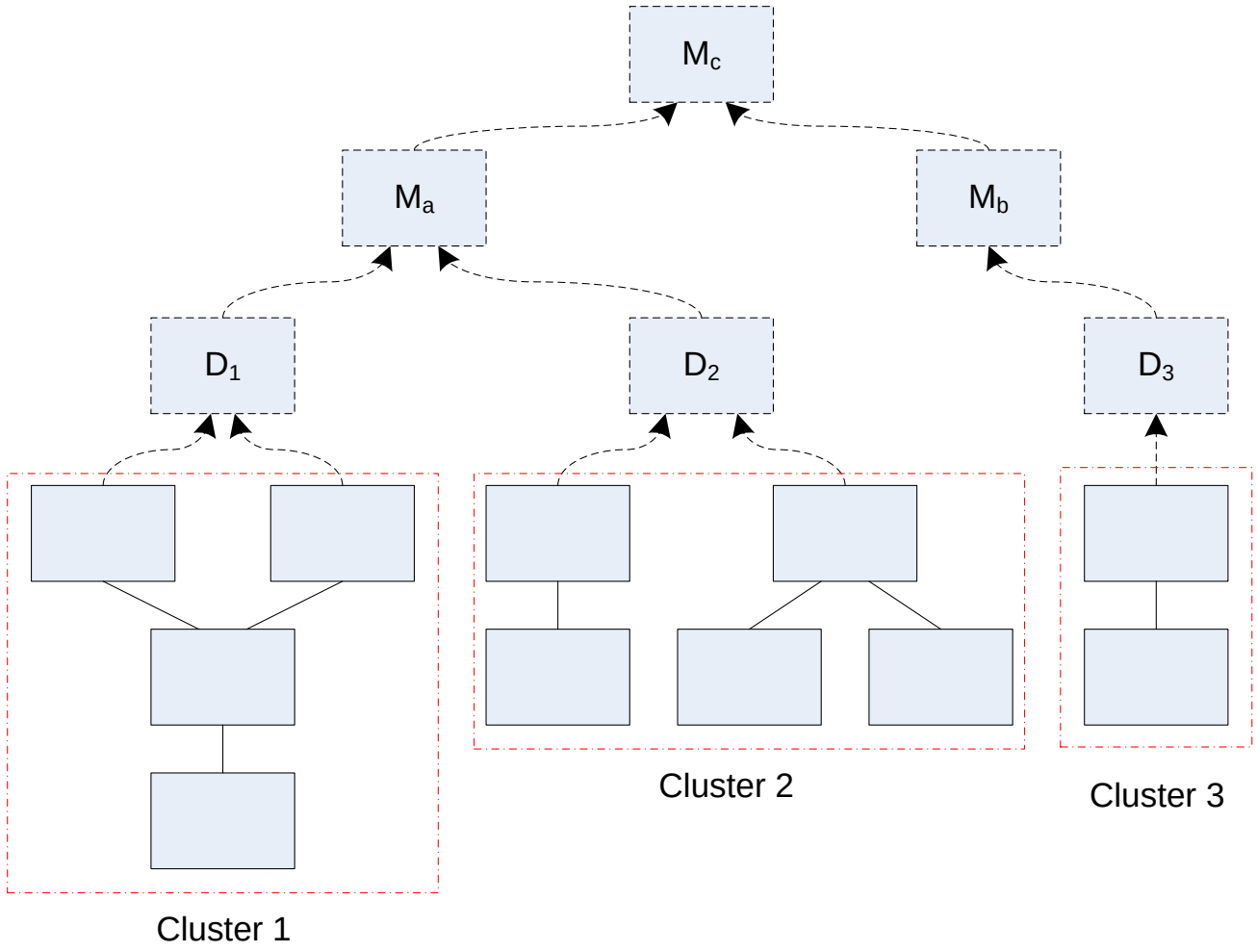
自底向上集成

- 自底向上的测试，就象它的名字中所暗示的一样，是从原子模块(也就是在程序结构的最低层的模块)开始来进行构造和测试的。每个模块由测试装置（test harness）进行测试。
- 一旦各个独立的模块测试完毕，把它们组合起来形成一组模块，称为造件（build）。一组造件再由第二个测试装置进行测试。这个过程将继续直到造件中包括整个应用系统。
- 因为模块是自底向上集成的，在进行时要求所有隶属于某个给定层次的模块总是存在的，而且也不再有使用程序桩的必要。

自底向上集成的策略步骤

- 自底向上的集成策略可以使用下列步骤来实现：

1. 低层模块组合成能够实现软件特定子功能的造件（builds），有时也称为簇（clusters）。
2. 写一个测试装置(一个供测试用的控制程序)来协调测试用例的输入输出。
3. 对簇进行测试。
4. 撤去测试装置，沿着程序结构的层次向上对造件进行组合。



自底向上的集成—示例

自底向上的优缺点

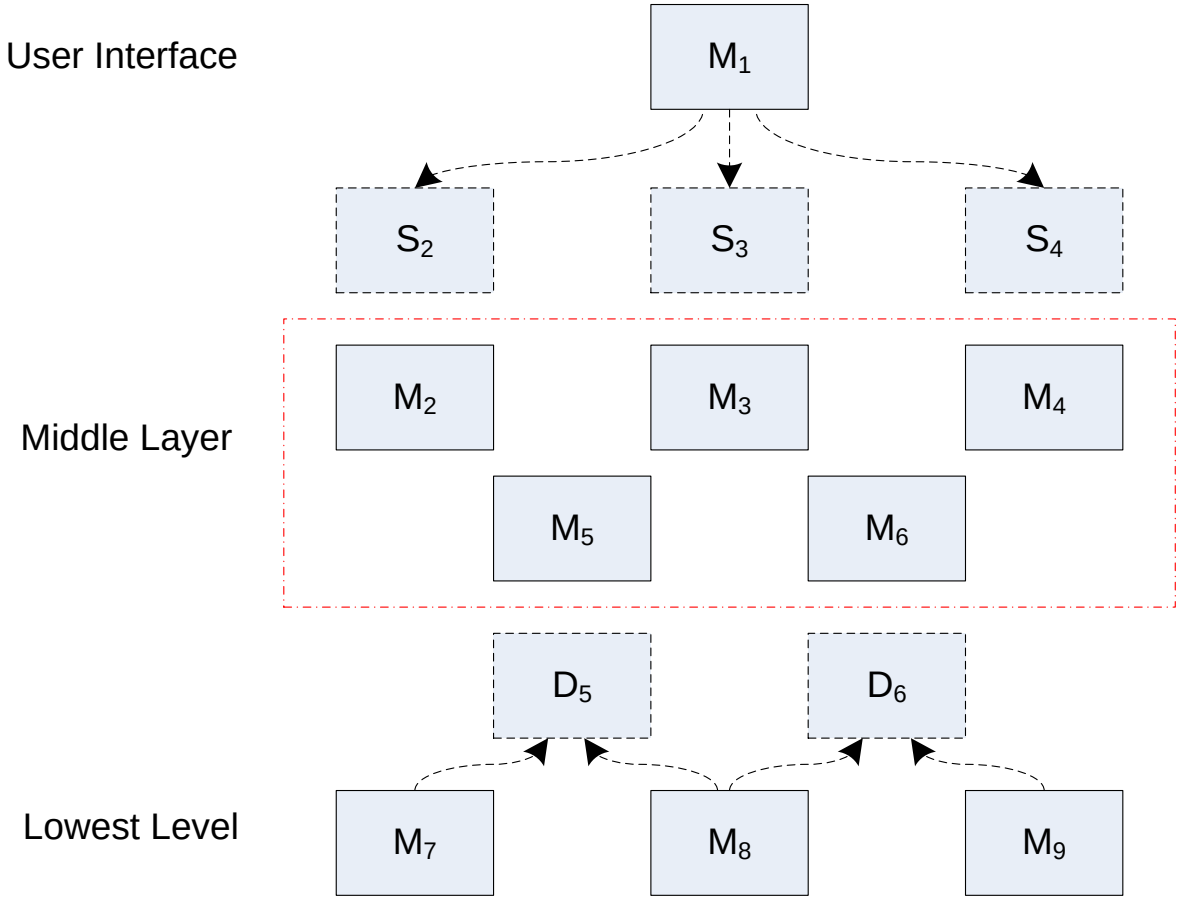
- 总结一下自底向上的优点：
  - （1）对底层行为早期进行确认；
  - （2）不需要写程序桩；
  - （3）对一些子树而言比较容易明确表示输入，比较容易解释对其他的输出。
- 自底向上的缺点：
  - （1）推迟对高层行为的确认；
  - （2）需要驱动程序；
  - （3）当组合子树的时候，一大堆元素要进行集成。

混合式集成

- 在实际中测试通常是结合了自顶向下和自底向上这两种方法，称作混合式集成测试（mixed testing），也称作三明治式集成测试（sandwich testing）.
- 在由几个小组一起开发的大的软件项目中，或者一个小项目但不同模块是由不同的人进行构建的情况下，小组或个人可以对自己开发的模块采用自底向上测试，然后再由集成小组进行自顶向下测试。

混合式集成策略步骤

- 混合式集成策略可以使用下列步骤来实现：
  1. 用程序桩独立地测试用户界面。
  2. 用驱动程序测试最低层功能模块。
  3. 当集成整个系统时，只有中间层是要进行测试的对象集。


User Interface
Middle Layer
Lowest Level
M1
S2
S3
S4
M2
M3
M4
M5
M6
D5
D6
M7
M8
M9

混合式集成－示例

三种增量测试策略的比较

	自顶向下	自底向上	混合式
形成基本可工作程序所需时间	早	晚	早
是否需要构件驱动器	否	是	是
是否需要程序桩	是	否	是
集成开始时可否平行工作	低	中等	中等
测试特殊路径的能力	难	易	中等
计划和控制顺序的能力	难	易	难

混和增量集成方法

- 风险驱动：从最关键或最复杂的模块开始进行集成，逐步加入它们调用或被调用的模块。
- 进度驱动：一旦模块就绪，比如，以某种方式可以获得或编码完成，就马上进行集成。
- 功能或线程驱动：选择跟某一个功能或线程有关的模块进行集成，逐步加入其他功能或线程。

端到端集成测试

- 传统的集成测试方法注重测试接口连接的。与传统的集成测试不同，端到端集成测试完全从最终用户的角度出发，强调对系统或应用程序进行端到端的功能测试（functional testing）。
  - 这一测试过程用于验证由一组相互连接的系统形成的集成系统是否正常运行，其中每个被连接的系统现在是集成系统的一个子系统。
- 端到端集成测试（E2E integration testing）一般是面向大型系统。端到端集成测试假设子系统的模块（或单元）测试和集成测试都已被执行并得到认可，但可能依然存在未被观察到的错误，其中集成测试可能包括多层次的集成测试。

端到端集成测试的功能

- 端到端集成测试提供以下功能：
  - 辅助生成测试用例，改进软件项目的生产率
  - 支持风险分析，通过确定风险领域从而进行全面的测试
  - 支持变更管理，从而使回归测试和波漪效应测试可以被正确和有效的执行
  - 支持数据质量评估，从而决策者可以从数量上客观的评估测试成果
  - 支持远程项目管理和分布式协作，使工程师和项目经理可以通过网络一起工作

端到端集成测试的过程

- 终端到终端（E2E）的测试过程：
  - 测试计划：确定主要任务及与其相关的进度安排和资源
  - 测试设计：开发测试规范、测试场景、测试用例和测试进度表
  - 测试执行：执行测试用例和记录结果
  - 测试结果分析：分析测试结果覆盖率，评估测试并确定过失
  - 重新测试和回归测试：在改进后的系统上进行附加测试

E2E集成测试计划

- E2E集成测试计划的核心是确定集成系统的范围，包括系统构架和功能；确定处理方法和工具以便于完成集成测试，并在实际测试之前制定结果评估标准。
- 在一个E2E测试计划中需要考虑一些重要的因素包括：
  - （1）主要的目标，（2）测试范围，（3）系统功能和非功能需求，（4）测试环境，（5）测试自动控制，（6）测试结果分析计划，（7）测试重用计划，（8）系统支持计划，（9）活动时间表，（10）退出标准。
- 建立测试工作小组，用来最好的确定E2E测试所需信息的方法是通过形成一个测试工作组来建立有效的沟通。

E2E集成测试设计

- E2E测试设计包括在测试环境下定义集成系统的任务和定义测试程序的任务。
- 可以通过如下两种观点来定义被测的集成系统：
  - （I）E2E功能视图；
  - （2）结构视图，包括物理结构和逻辑结构。
- 通过一个细线程树及所附条件来给出被测系统规范。

细线程 (THIN THREAD)

- 定义：一个完整的数据或消息的踪迹,使用最低限度的具有代表性质的外部输入数据样本,通过系统的内部的相互连接部分的转换,产生最低限度的具有代表性质的外部输出数据样本.使用细线主要是用来阐释某一方法具有指定的功能的.
- 细线程是集成的系统中最小的场景，从最终用户角度看,它是一个完整的场景，系统接受输入数据,经过计算,并输出处理的结果。它描述了整个场景,而且只描述一个功能。

细线程组

- 细线程间的关系，组/子组，一些共享某些公共数据的细线程可以组成一个细线程组。
  - 这些组是具有层次结构的。细线组中的所有细线程和子细线程组可以构成一棵细线程树。
  - 树的根是整个被测的集成系统，它的分支节点代表相关的细线程集合,它的叶子代表一个具体的细线程。

E2E集成测试规格书

- E2E测试规格书是E2E测试的核心，表述系统需求，从最终用户的角度出发的使用情景，测试观点出发的详细系统行为描述：正常的输入，非正常输入，常规用例和特殊处理。
- E2E测试规格书是一种半规范化，等级的构架，用例生成的相关数据，追溯到其他软件工件上，主要有两部分信息：细线程和条件。

细线程的定义模版

- 细线程的定义模版：ID、名字、描述、输入/输出、前提条件/后置条件、隐藏的成分、状态、代理以及风险。
- 细线程组：一组具有特定公共部分的细线程的集合，递归分组，重组为一个树结构。细线程树：自顶向下构建，按功能性进行分解；自底向上构建，利用提取和合成方法。

条件分析

- 条件是断言激活这个功能条件必须是真。
- 条件的例子有数据条件、信息条件、环境条件以及系统状态。
- 条件分析是完全性分析和一致性分析的一部分，可以发现新的细线程，通过发现不完全/不一致的条件，发现附属于矛盾条件的细线程。

条件定义模版

- 条件定义模版：ID、名字、描述、受影响的细线程等。条件组是一组具有特定公共属性的条件的集合。递归分组，重组为一个树结构。条件树，自顶向下分解，自底向上提取和合成。
- 条件间的关系：
  - 独立条件，互斥条件，引发/被引发的条件。相关条件。

E2E集成测试风险分析

- 风险分析是系统和软件开发中的一项重要活动。
- 基于风险分析，系统的临界条件可以被彻底的测试。
- 当资源有限的时候，测试应该把那些重要的线程放在首位。
- 一种排列细线程的方法是至少基于以下两种因素给每一个细线程分配一个风险。
- 系统可能舍弃一个已给定的细线程。

失败可能性

- 下面几个构件常有很高的失败可能性：
  - 在先前的模块或综合测试中就显现出不可靠性的成分
  - 具有复杂的实现过程或者具有合并的复杂功能的成分
  - 与许多其他成分相连的成分
  - 由于错误或功能的变更而最近刚刚修改过的成分

RISK CALCULATIONS

- Risk thin-thread = F (probability$_{\text{thin-thread}}$, Consequence$_{\text{thin-thread}}$)
- Risk condition = F (probability$_{\text{condition}}$, Consequence$_{\text{condition}}$)
- Risk test-case = F (probability$_{\text{test-case}}$, Consequence$_{\text{test-case}}$)

E2E测试执行

- E2E测试执行准备。在测试之前，测试工程师要识别出以下组件：要测的子系统；支持系统：包括硬件，固件，数据库，第3方组件，确保子系统合适地执行。
- 备份系统和程序，防止万一测试损坏了系统，系统还可以恢复；测试数据，包括测试输入数据，数据库，执行测试用例所需要的文件；测试工具，包括自动输入数据生成工具，测试驱动，测试结果记录工具；测试组；

模仿环境下进行E2E测试

- 在模仿环境下进行E2E测试包括以下步骤：
  - 开发或取得模拟程序，把模拟程序的参数调用跟要测试的系统一样；
  - 执行应用系统；
  - 选择测试用例，生成输入数据，记录执行结果；
  - 重复选择和执行测试用例的过程，在必要的时候恢复系统和模拟状态，直到所有安排的测试用例执行了为止。

操作环境中进行E2E测试

- 在操作环境中进行E2E测试包括以下步骤：
  - 建立环境，调用应用系统，选择测试案例，根据系统外部接口生成输入数据；
  - 重复选择和执行测试案例的过程，在必要的时候恢复系统和模拟状态，直到所有安排的测试案例执行了为止。

- 退出标准，所有的测试用例都被执行，测试覆盖率要求被满足

测试结果分析

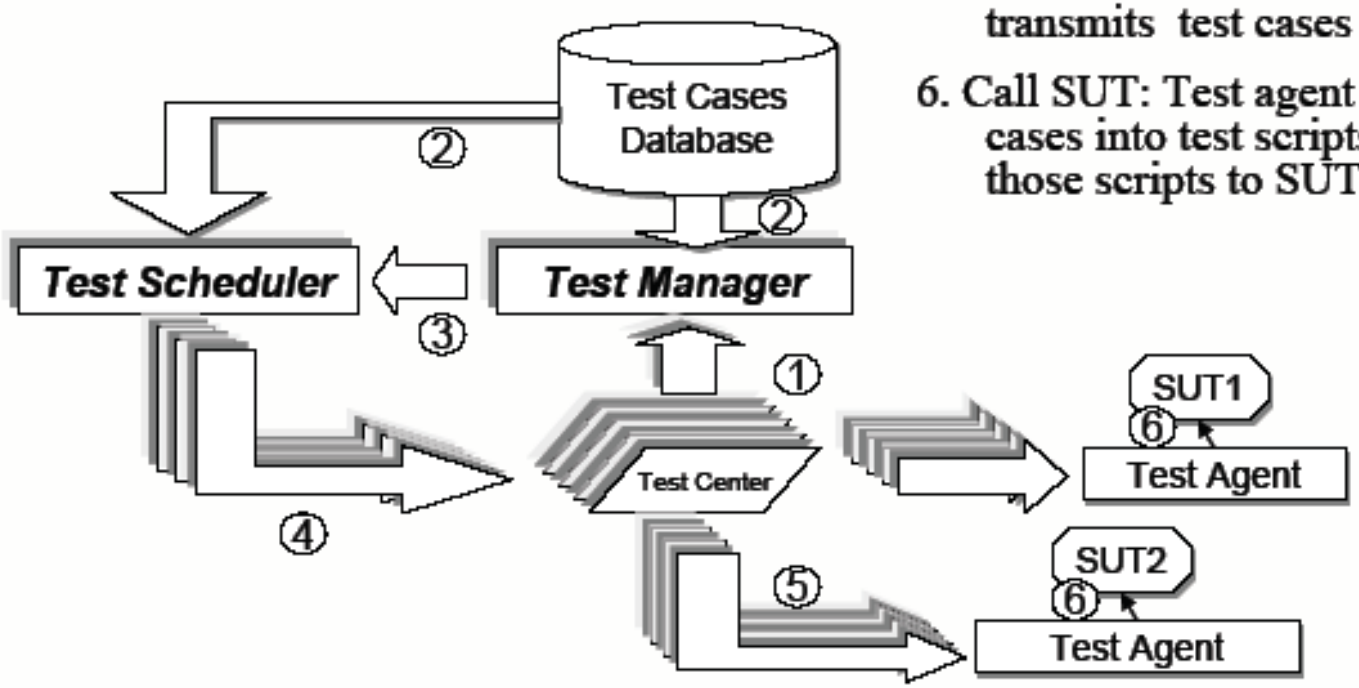
- 缺陷识别和改正，缺陷是当执行的时候可能会产生不正确结果的代码错误。要找出缺陷，测试输出要跟预期输出进行比较。在测试过程中缺陷识别可以被优先并改正。
- 评估波漪效应，软件工件的一部分改动会影响其他相关部分的现象叫做波漪效应，而迭代的分析而且去除改变的副作用的过程就叫做波漪效应分析。

波漪效应分析

- 波漪效应分析过程包括以下步骤：
  - 提出软件修改请求
  - 识别依赖于被改变部分的其他模块
  - 决定是否要改变依赖部分来保持一致性
  - 如果需要改变，从第一步开始循环进行REA
  - 如果不需要改变，停止并等待修改软件

远程测试(REMOTE TESTING)

1. Registration: Test center registration process
2. Retrieve test cases: Test manager starts a test process/test plan that includes a set of test cases, which are retrieved from the database
3. Testing Schedule: Test manager sends current test plan's test cases and available test center information to the test scheduler
4. Transfer test cases to test center: Test scheduler sends testing cases to their corresponding test centers
5. Transfer test case: Test center transmits test cases to test agents
6. Call SUT: Test agent translates test cases into test scripts and transmits those scripts to SUT for execution



回归测试

- 回归测试是在修改过的软件上重新运行测试用例，普遍用于软件程序被修改的情况 一个要点就是回归测试只能保证这些应该保留的部分保持不变。
- 需要在不同的层面上进行回归测试，首先是模块测试层面，之后是在集成测试层面，最后是终端到终端测试层面。
- 在回归分析的每一个层面，测试人员有不同的任务。

总 结

单元测试

- 单元测试考虑事项
- 单元测试规程
- 单元测试局限性

集成测试

- 自顶向下集成
- 自底向上集成
- 混合式集成
- 端到端集成测试