西北工業大學
NORTHWESTERN POLYTECHNICAL UNIVERSITY

第二章　软件测试策略

授课教师：　郑炜

第二章 软件测试策略

西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

- 2.1 软件开发过程及模型
  - 2.1.1 软件开发过程
  - 2.1.2 软件开发模型
- 2.2 软件测试过程
  - 2.2.1 测试计划和控制
  - 2.2.2 测试分析和设计
  - 2.2.3 测试实现和执行
  - 2.2.4 测试出口准则的评估和报告
  - 2.2.5 测试活动结束行

- 2.3 软件测试与软件开发的关系
- 2.3.1 软件测试在软件开发中的作用

2.3.2 软件测试与软件开发各阶段的关系

2.3.3 常见软件测试模型

- 2.4 黑盒测试和白盒测试

2.4.1 黑盒测试

2.4.2 白盒测试

2.4.3 黑盒测试与白盒测试的比较

2.1.1 软件开发过程

软件开发过程对于整个软件工程来说是十分重要的，软件测试也是基于软件开发完成的。

正规的软件开发过程可以分为8个部分：

可行性研究、需求分析、概要设计、详细设计、实现、集成测试、确认测试，以及使用与维护。

2.1.2 软件开发模型

（1）瀑布模型

1970年，温斯顿·罗伊斯（Winston Royce）提出了著名的“瀑布模型”，直到20世纪80年代早期，它一直是唯一被广泛采用的软件开发模型。瀑布模型将软件生命周期划分为制订计划、需求分析、软件设计、程序编写、软件测试和运行维护6个基本活动，并且规定了它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落，如图2-1所示。

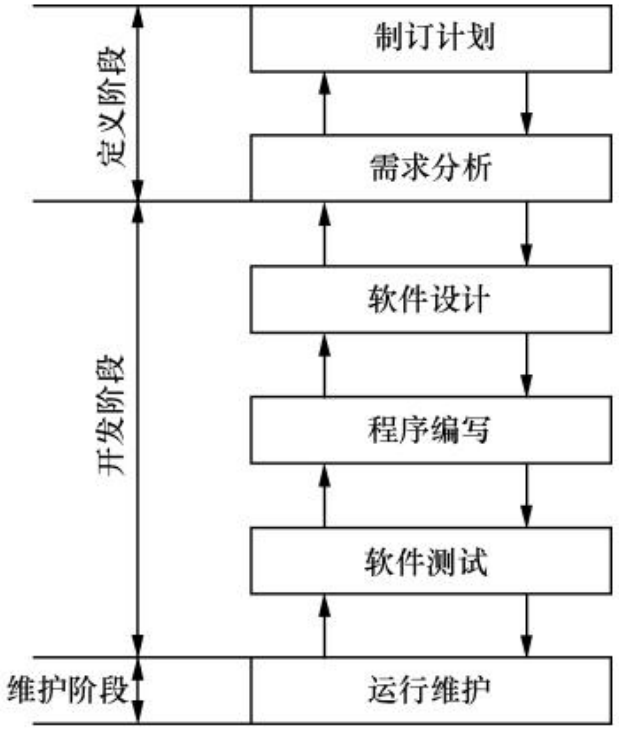


图2-1 瀑布模型

（1）瀑布模型

瀑布模型强调文档的作用，并要求每个阶段都要仔细验证。但是，这种模型的线性过程太理想化，已不再适合现代的软件开发模式，几乎被业界抛弃。其主要问题有以下3个方面。

① 各个阶段的划分完全固定，阶段之间产生大量的文档，极大地增加了工作量。

② 由于开发模型是线性的，用户只有等到整个过程的末期才能见到开发成果，从而增加了开发的风险。

③ 早期的错误可能要等到开发后期的测试阶段才能发现，进而带来严重的后果。

（2）快速原型模型

快速原型模型首先根据需求分析进行原型开发，即建造一个快速原型，实现客户或未来的用户与系统的交互，然后用户或客户对原型进行评价，进一步细化待开发软件的需求，如图2-2所示。通过逐步调整原型使其满足客户的要求，开发人员最终确定客户的真正需求是什么；最后在快速原型的基础上开发客户满意的软件产品。

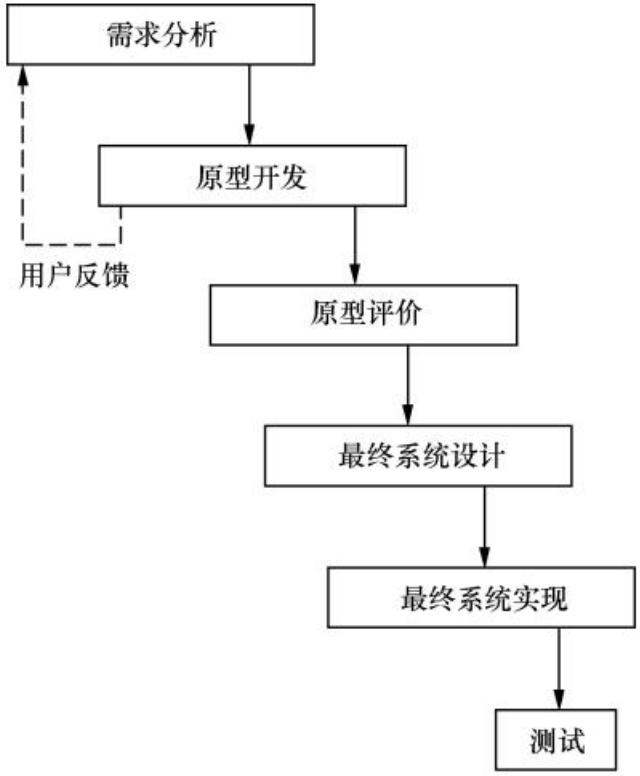


图2-2 快速原型模型

（2）快速原型模型

显然，快速原型模型可以克服瀑布模型的缺点，减少由于软件需求不明确带来的开发风险，具有显著的效果。快速原型模型的关键在于尽可能快速地建造出软件原型，一旦确定了客户的真正需求，所建造的原型将被丢弃。因此，原型系统的内部结构并不重要，重要的是必须迅速建立原型，随之迅速修改原型，以反映客户的需求。

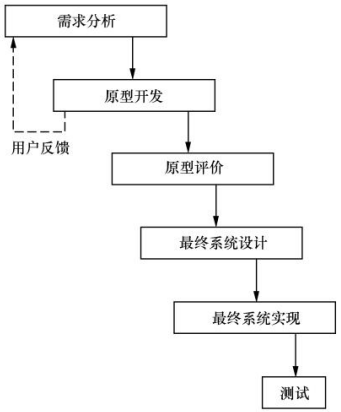


图2-2 快速原型模型

（3）增量模型

增量模型又称演化模型，如图2-3所示。

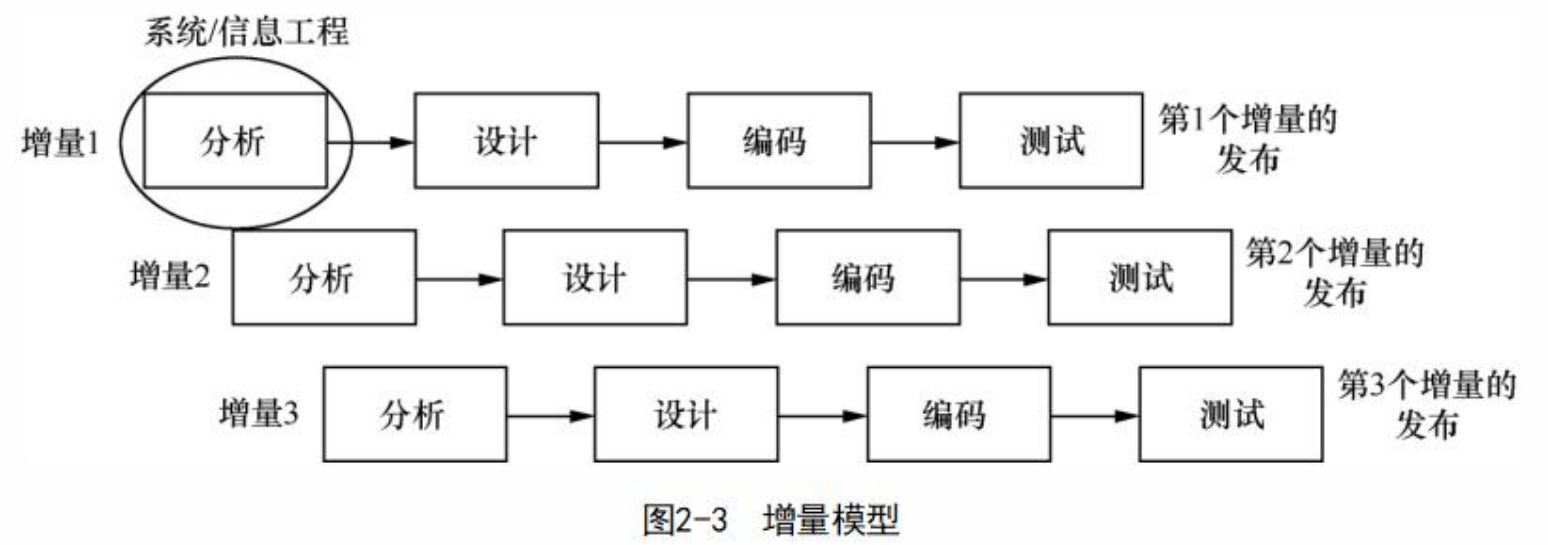


图2-3 增量模型

（3）增量模型

在增量模型中，软件被作为一系列的增量构件来设计、实现、集成和测试，每一个构件是由多种相互作用的模块所形成的提供特定功能的代码片段构成的。增量模型在各个阶段并不交付一个可运行的完整产品，而是交付满足客户需求的一个子集的可运行产品。整个产品被分解成若干个构件，开发人员逐个构件地交付产品，这样做的好处是软件开发可以较好地适应变化，客户可以不断地看到所开发的软件，从而降低开发风险。

（3）增量模型

但是，增量模型也存在以下缺陷。

① 由于各个构件是逐渐并入已有的软件体系结构中的，所以加入的构件必须不破坏已构造好的系统部分，这需要软件具备开放式的体系结构。

② 在开发过程中，需求的变化是不可避免的。增量模型适应变化的能力大大优于瀑布模型和快速原型模型，但也很容易退化为边做边改模型，从而使软件过程的控制失去整体性。

在使用增量模型时，第1个增量往往是实现基本需求的核心产品。核心产品交付用户使用后，经过评价形成下一个增量的开发计划，它包括对核心产品的修改和一些新功能的发布。这个过程在每个增量发布后不断重复，直到产生最终的完善产品。

（4）螺旋模型

1988年，巴利·玻姆（Barry Boehm）正式发表了软件系统开发的“螺旋模型”，它将瀑布模型和快速原型模型结合起来，强调了其他模型所忽视的风险分析，特别适合于大型复杂的系统。螺旋模型沿着螺旋线进行若干次迭代，图2-4所示的螺旋模型的4个象限分别代表了制订计划、风险分析、实施工程和客户评估4个活动。

（4）螺旋模型

螺旋模型也有一定的限制条件，具体如下。

① 螺旋模型强调风险分析，但要求许多客户接受和相信这种分析，并做出相关反应是不容易的，因此，这种模型往往适应于内部的大规模软件开发。

② 如果执行风险分析会大大影响项目的利润，那么进行风险分析将毫无意义，因此，螺旋模型只适合于大规模软件项目。

③ 软件开发人员应该擅长寻找可能的风险，准确地分析风险，否则将会带来更大的风险。

2.1.2 软件开发模型

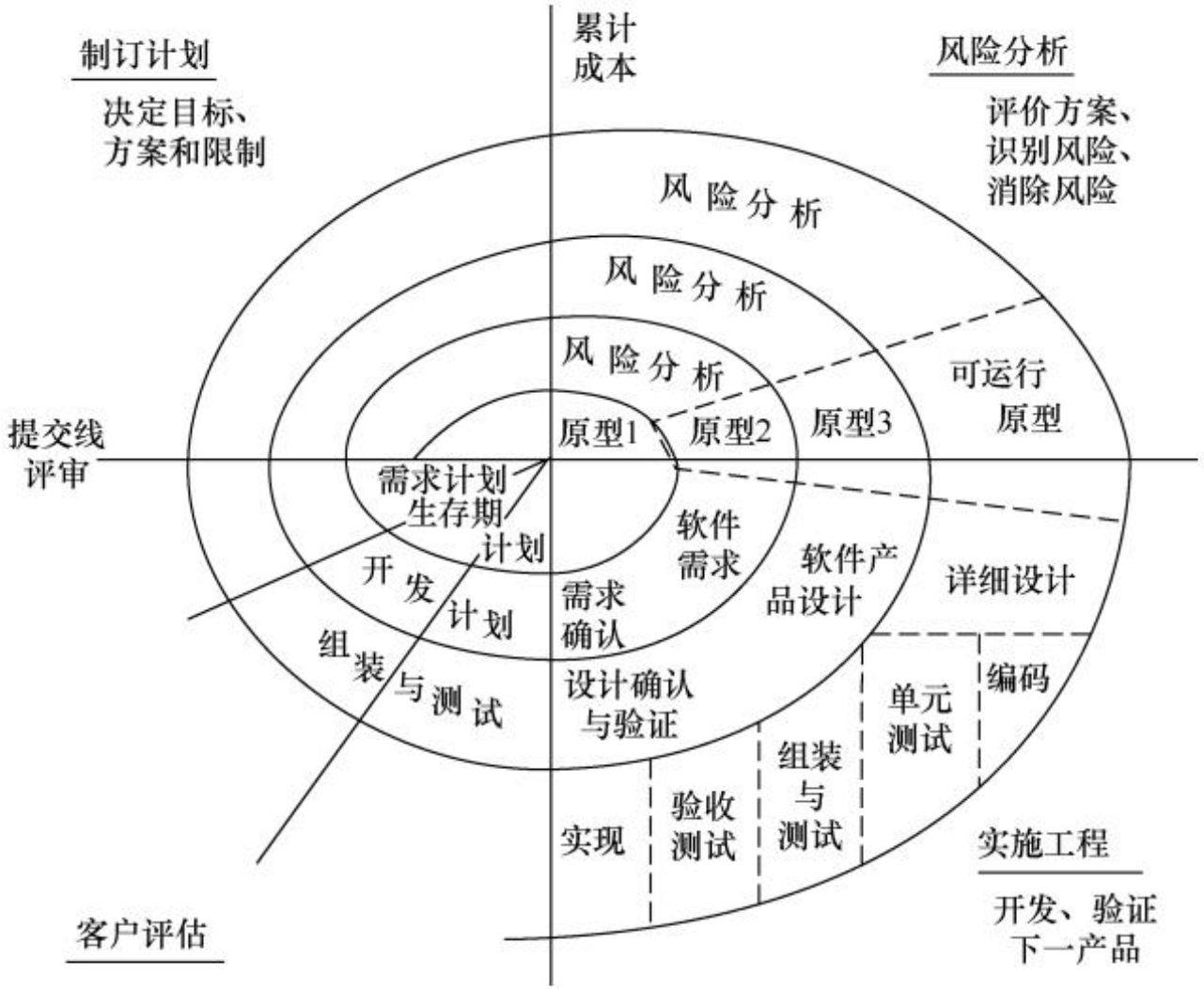


图2-4 螺旋模型

- 2.1 软件开发过程及模型
- 2.2 软件测试过程
  - 2.2.1 测试计划和控制
  - 2.2.2 测试分析和设计
  - 2.2.3 测试实现和执行
  - 2.2.4 测试出口准则的评估和报告
  - 2.2.5 测试活动结束行

2.2.1 测试计划和控制

测试作为贯穿整个软件开发过程的活动，需要有一份完善且周详的测试计划作为指导。测试计划是整个测试过程的路由图，在需求活动一开始时，相关人员就需要着手进行测试计划的编写了。形成一份完整、详细的测试文档需经过计划、准备、检查、修改和继续5个步骤。

测试计划主要的任务就是确定测试策略，它定义了项目的测试目标和实现方法。

测试强度很大程度上取决于使用哪种测试工具，需要达到哪一个级别的测试覆盖率，涉及源代码的测试覆盖率常用作测试出口准则之一。因为软件项目经常是在苛刻的时间压力下完成的，所以需要在计划过程中考虑这个问题，合理分配时间，保证软件的最重要部分优先级最高，最先被测试，以免由于时间上的限制而无法执行已经计划好的测试。

2.2.2 测试分析和设计

测试计划完成后，测试过程就进入了测试用例的设计和测试脚本的开发阶段。测试用例的规格说明分为两步进行：首先要定义逻辑测试用例，然后选择实际输入，将逻辑测试用例转换成具体测试用例。

测试用例设计的方法和管理

每个测试用例都必须描述其初始状况，即前置条件：测试用例要清楚定义需要什么样的环境条件，以及必须满足的其他条件，此外，还需要提前定义期望得到哪些结果和行为。结果包括输出、全局化数据和状态的变更，以及执行测试用例后的其他任何结果。而常见的编写测试用例的方法有等价类划分、边界值分析、因果图、错误推测法、状态迁移图、流程分析法、正交验证法等。

测试脚本的开发

要进行测试脚本的开发，首先需要设立测试脚本的开发环境，安装测试工具，设置管理服务器和具有代理的客户端，建立项目的共享路径和目录，并能连接到脚本存储库和被测软件等。然后执行录制测试的初始化过程、独立模块过程、导航过程和其他操作过程，结合已经建立的测试用例，将录制的测试脚本进行组织、调试和修改，构造成一个有效的测试脚本体系，并建立外部数据集合。

但测试脚本的开发也存在一些常见的问题，如测试脚本很乱、测试脚本与测试需求或测试策略没有对应性、测试脚本不可重用、测试过程被作为一个编程任务来执行导致脚本可移植性差等。这些问题应该尽量避免，设计好脚本的结构、模块化、参数传递、基础函数等方面。

2.2.3 测试实现和执行

测试用例的设计和测试脚本的开发完成之后，就可以开始执行测试了。测试的执行有手工测试和自动化测试两种。

手工测试是在合适的环境下，按照测试用例的条件、步骤要求，准备测试数据，对系统进行操作，比较实际结果和测试用例所描述的期望结果，以确定系统是否正常运行；而自动化测试是使用测试工具运行测试脚本，从而得到测试结果。自动化测试的管理相对比较容易，测试工具会完整执行测试脚本，而且可以自动记录测试结果。

判断测试终结

测试出口准则的评估是检验测试对象是否符合预先定义的一组测试目标和出口准则的活动。测试出口准则的评估可能产生下列结果：

测试结果满足所有出口准则，可以正常结束测试；

要求执行一些附加测试用例；

测试出口准则要求过高，需要对测试出口准则进行修改。

执行完所有测试用例后再对比测试出口准则，如果发现还有一个，甚至多个条件没有被满足，那么就需要执行进一步的测试，或是思考测试出口准则是否合理、是否需要修改。此时如果测试人员要增加新的测试用例，需要考虑新加的测试用例是否符合测试用例准则；否则，额外的测试用例只会增加工作量，对测试没有起到任何帮助。

测试过程中，有时可能会出现测试对象本身问题导致定义的测试出口准则无法满足的情况。

测试终结的其他标准

除测试覆盖率以外，测试出口准则还可以包括其他条目，例如，失效发现率，也叫软件缺陷发现百分比。它是由单位时间内新发现的失效个数的平均值计算来的。如果失效发现率降低到设定的阈值以下，就表明不再需要更多的测试，测试工作可以结束了。根据失效发现率评估测试出口准则时还应考虑失效的严重程度，对失效进行分类并区别对待。

测试开销

在实际测试过程中，能够结束测试还要考虑时间与成本方面的因素。如果测试人员由于这些因素不得不停止测试工作，很可能是在资源分配阶段中没有做好相应的工作，或者对测试某个活动的工作量做出了一个误判，导致后期的时间或成本短缺。

2.2.5 测试活动结束

测试的全部完成，并不意味着测试过程的结束。在测试过程的最后，有可能会遗漏一些应当被完成的活动。例如，测试人员在测试的最后应该分析并且整理在测试工作中积累的经验，以便在以后的项目中使用。测试人员测试时还应注意在不同活动中观察到的计划和执行的背离以及可能引起这些背离的原因。在测试的最后阶段，测试人员需要编写测试或质量报告，还需要记录项目的一些重要信息。例如，一些应该记录的信息有：软件系统是何时发布的、测试工作是何时完成或者结束的、软件何时抵达一个里程碑或者完成一个版本的维护更新等。

此外，除测试报告或质量报告的写作之外，还要对测试计划、测试设计和测试执行等进行检查分析，完成项目总结并编写项目总结报告。

- 2.1 软件开发过程及模型
- 2.2 软件测试过程
- 2.3 软件测试与软件开发的关系
- 2.3.1 软件测试在软件开发中的作用

  2.3.2 软件测试与软件开发各阶段的关系

  2.3.3 常见软件测试模型

测试在开发各阶段的作用如下：

- **项目规划阶段：**负责监控整个测试过程。
- **需求分析阶段：**确定测试需求分析，即确定在项目中需要测试什么，同时制订系统测试计划。
- **概要设计和详细设计阶段：**制订集成测试计划和单元测试计划。
- **程序编写阶段：**开发相应的测试代码和测试脚本。
- **测试阶段：**实施测试，并提交相应的测试报告。

2.3.2 软件测试与软件开发各阶段的关系

软件测试与软件开发各阶段的关系如图2-6所示。

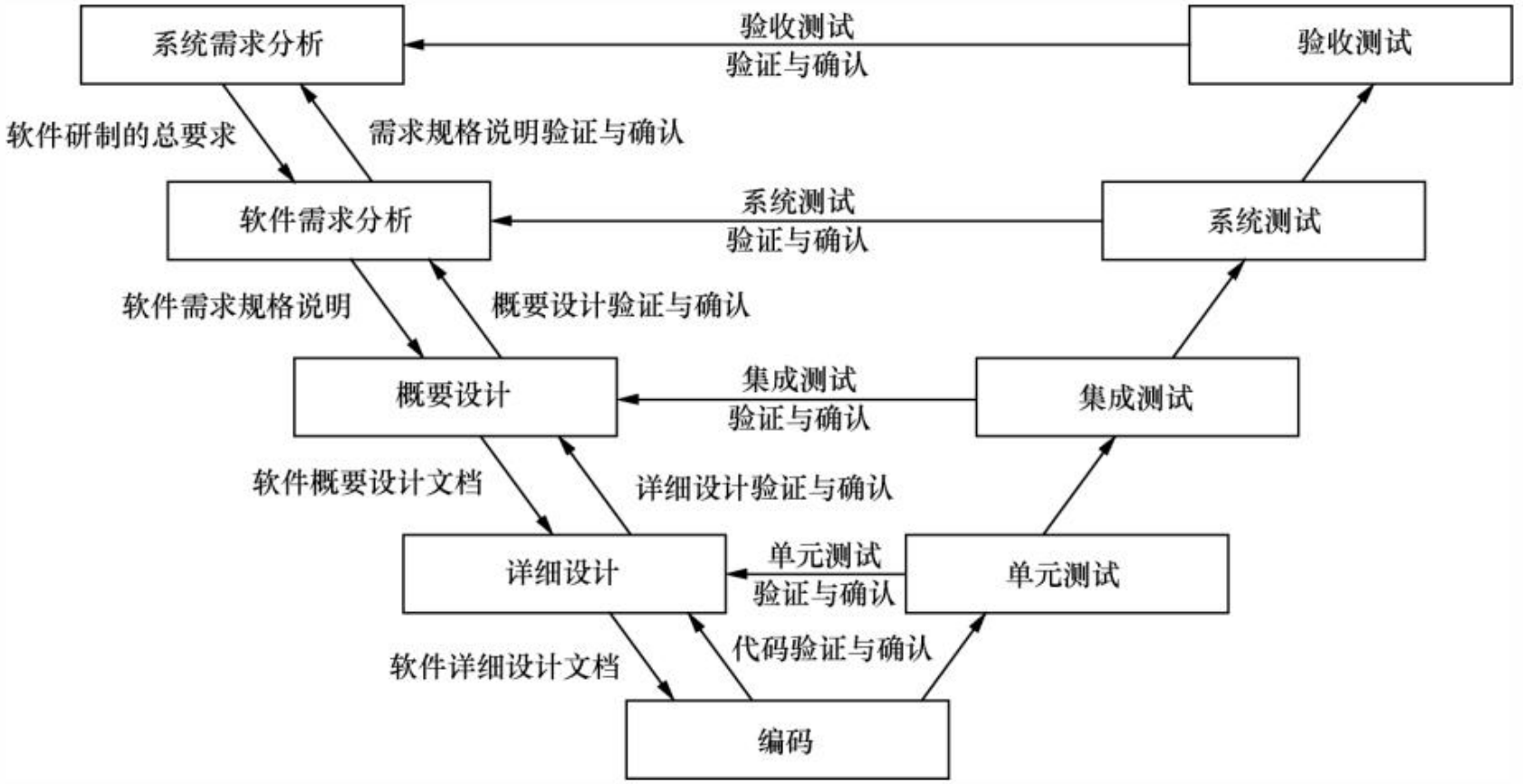


图2-6 软件测试与软件开发各阶段的关系

2.3.3 常见软件测试模型

- V模型

V模型是软件开发瀑布模型的变种，主要反映了测试活动分析与设计的关系，如图2-7所示。

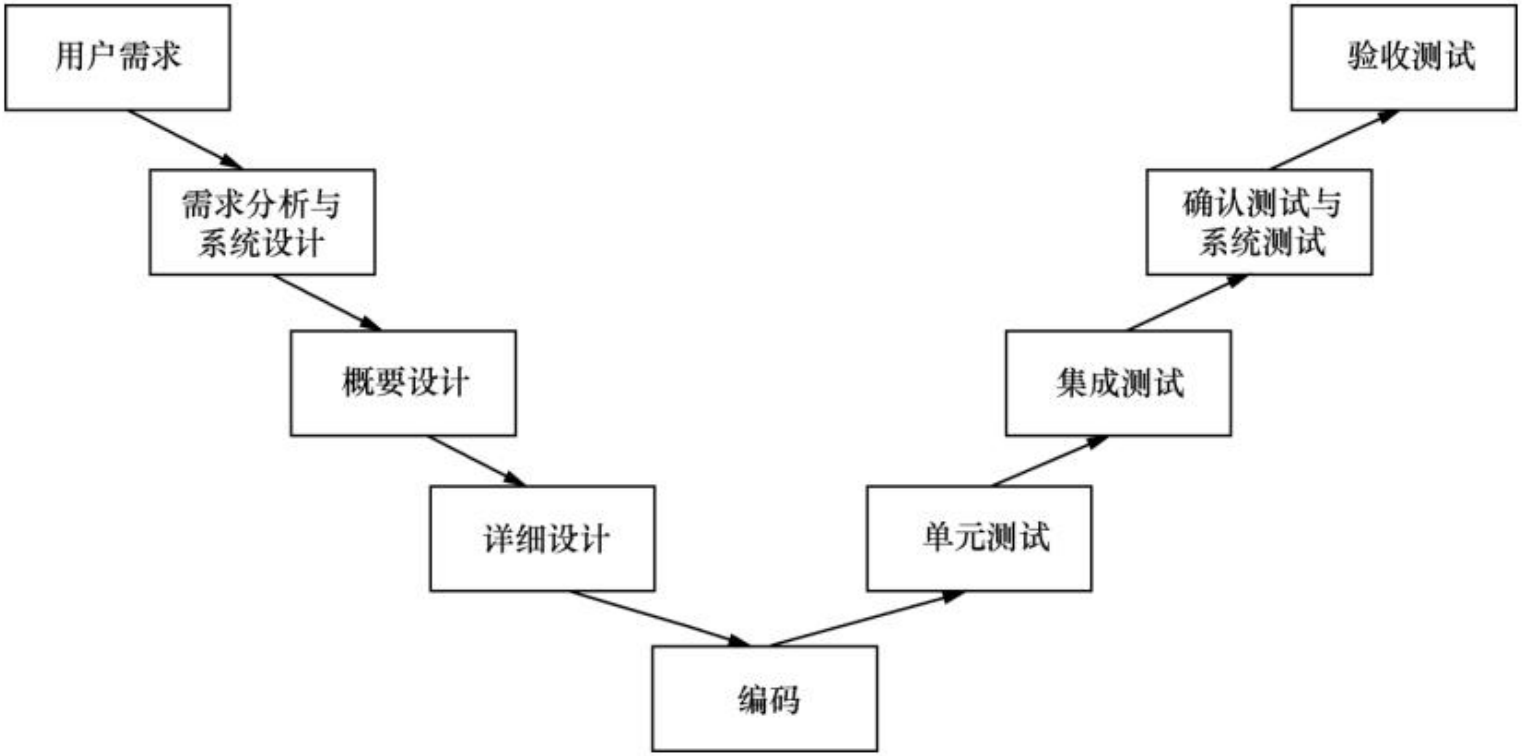


图2-7 V模型

2.3.3 常见软件测试模型

- V模型

V模型描述了基本的开发过程和测试行为。它不仅包含保证源代码正确性的低层测试，而且包含满足用户系统要求的高层测试。图2-7箭头方向为时间方向，从左至右分别是开发的各阶段与测试的各阶段。

V 模型非常明确地标注了测试过程中存在的不同级别，并使测试的每个阶段都与开发的阶段相对应。但 V 模型也存在着一定的局限，它不能发现需求分析等前期阶段产生的错误，直到编码完成之后才进行测试，因此早期出现的错误不能及时暴露。此外，V 模型只是对程序进行测试，早期的需求、设计环节并没有涵盖其中，也为后来的测试埋下了隐患。

2.3.3 常见软件测试模型

- W模型

W模型如图2-8所示

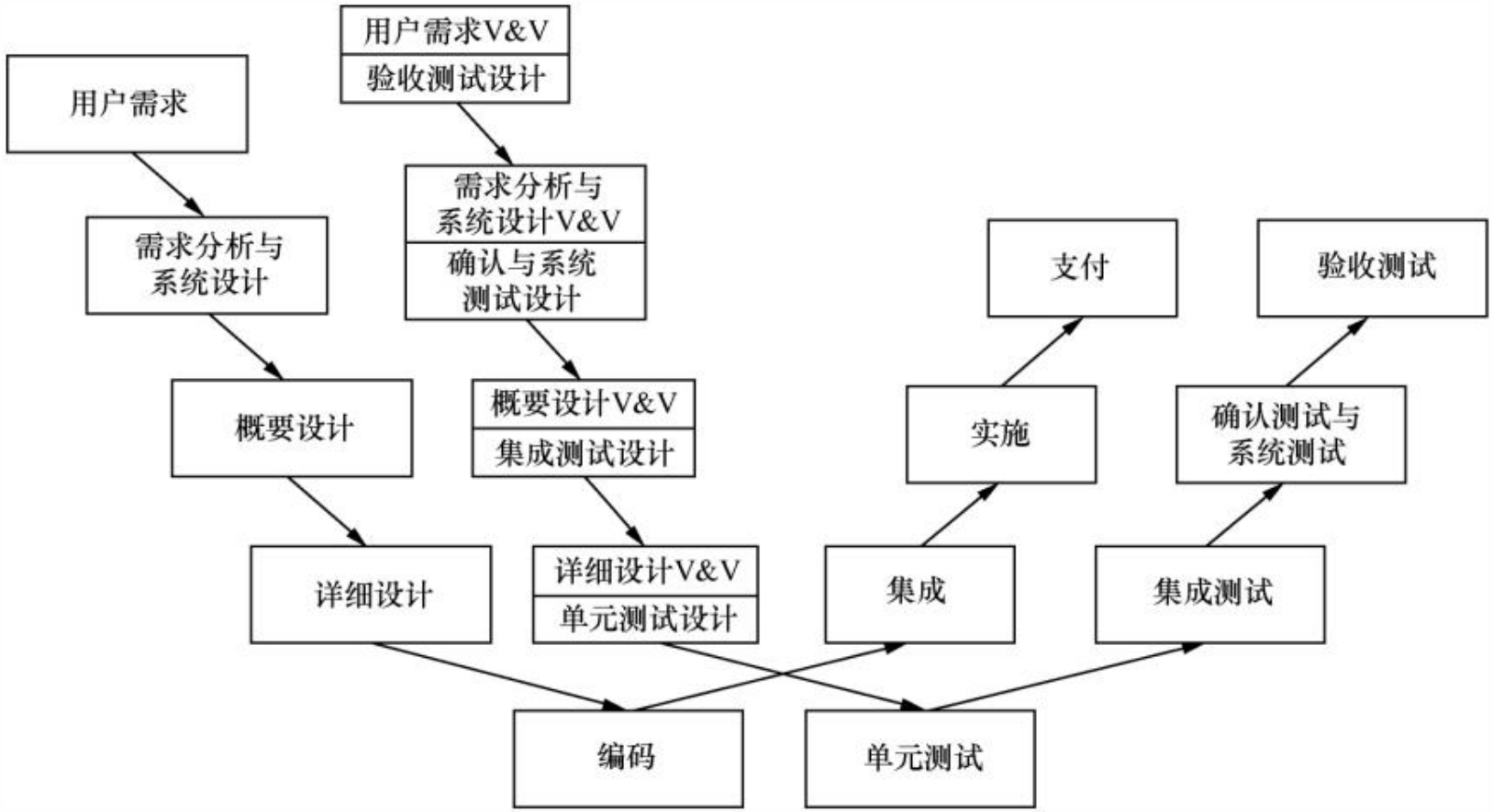


图2-8 W模型

2.3.3 常见软件测试模型

- W模型

W模型是由V模型自然演化发展而来的，它强调了测试应贯通于整个开发过程，而且测试的对象还包括了需求、功能与设计等。在W模型中，可以认为测试与开发是同步进行的，这样可以使开发过程中的问题及早地暴露出来。同时W模型强调了测试计划等工作的先行和对系统需求、系统设计的测试。

但W模型仍存在着较为明显的问题，因为它将开发活动认定为从需求开始到编码结束的串行活动，只有在上一活动结束后才能开始下一步的行动，无法支持需要迭代、自发性以及变更调整的项目。

2.3.3 常见软件测试模型

H模型

H 模型将测试活动从开发流程中完全独立出来了，清晰地表现了测试准备活动与测试执行活动，如图2-9所示；测试流程仅演示了整个生产周期中某个层次上的一次测试“微循环”，而图2-9中的其他流程可以是任意开发流程。一旦测试条件成熟，准备活动完成后，就可以执行测试活动了。

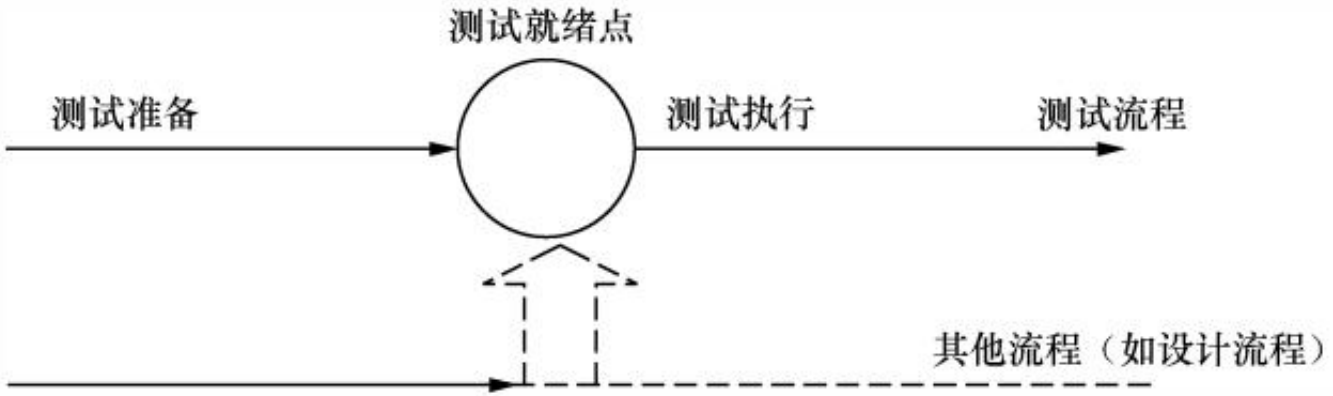


图2-9 H模型

2.3.3 常见软件测试模型

- H模型

H模型强调软件测试是一个独立的流程，它需要“尽早准备，尽早执行”。软件测试贯穿于产品的整个生命周期，可以与其他流程并发进行。但是H模型的缺点在于它太过于抽象化，测试人员应该重点理解其中的意义，并以此来指导实际测试工作，而模型本身并无太多可执行的意义。

2.3.3 常见软件测试模型

- X模型

X模型如图2-10所示

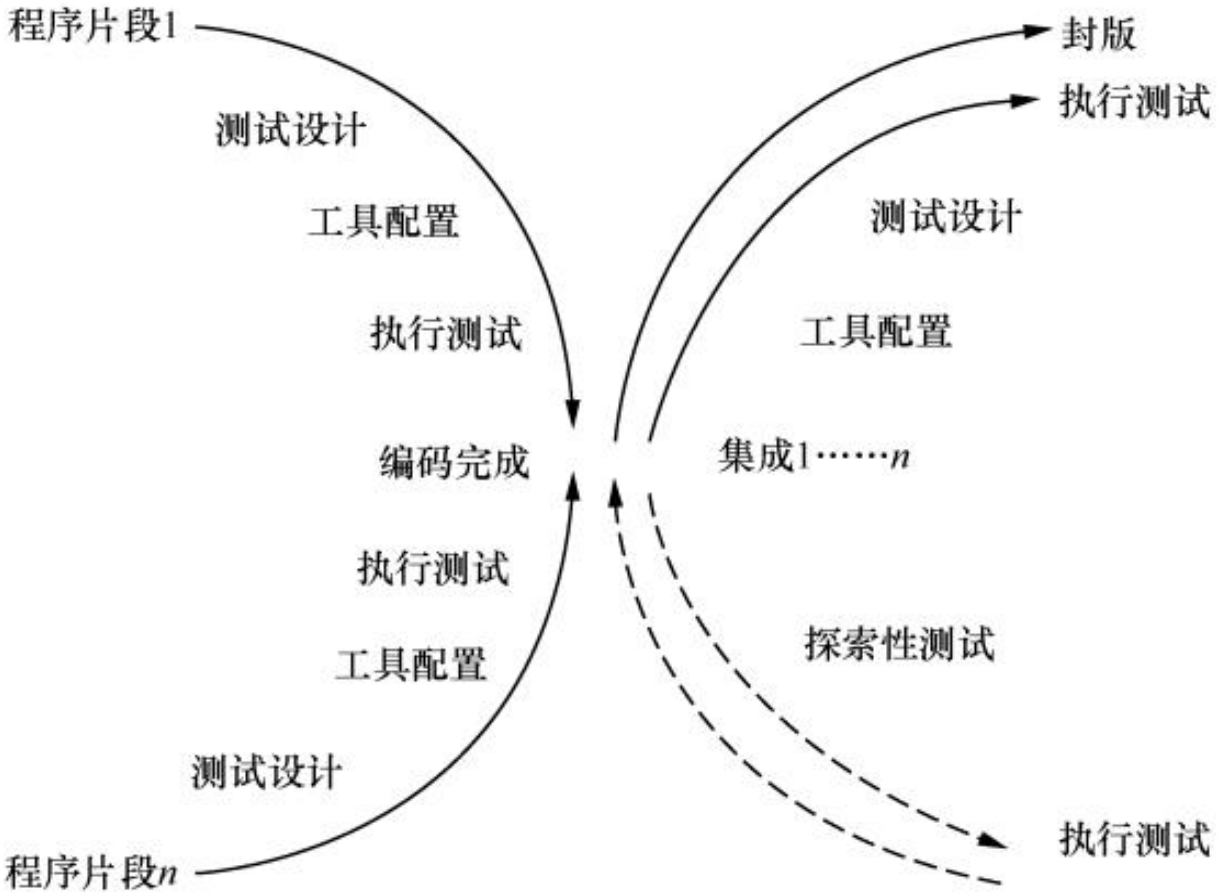


图2-10 X模型

2.3.3 常见软件测试模型

- X模型

X模型左边描述的是针对单独程序片段所进行的相互分离的编码和测试，如图2-10左边所示，然后频繁地交接，再通过集成最终合成为可执行的程序，如图2-10右上方所示，而且可执行程序还需要进行测试，已经通过集成测试的成品可以进行封版并提交给用户，也可以作为更大规模和范围内集成的一部分。同时，X模型还定位了探索性测试，如图2-10右下方所示，这是不进行实现计划的特殊类型测试，例如，“我就这么测一下，结果会怎么样”。

作为探索性测试，X模型能够帮助有经验的测试人员在测试计划之外发现更多的软件错误，但也存在一定的弊端：它有可能对测试造成人力、物力和财力的浪费，同时也对测试人员的熟练程度要求比较高。

2.3.3 常见软件测试模型

- 前置测试模型

前置测试模型将开发和测试的声明周期整合在一起，如图2-11所示。

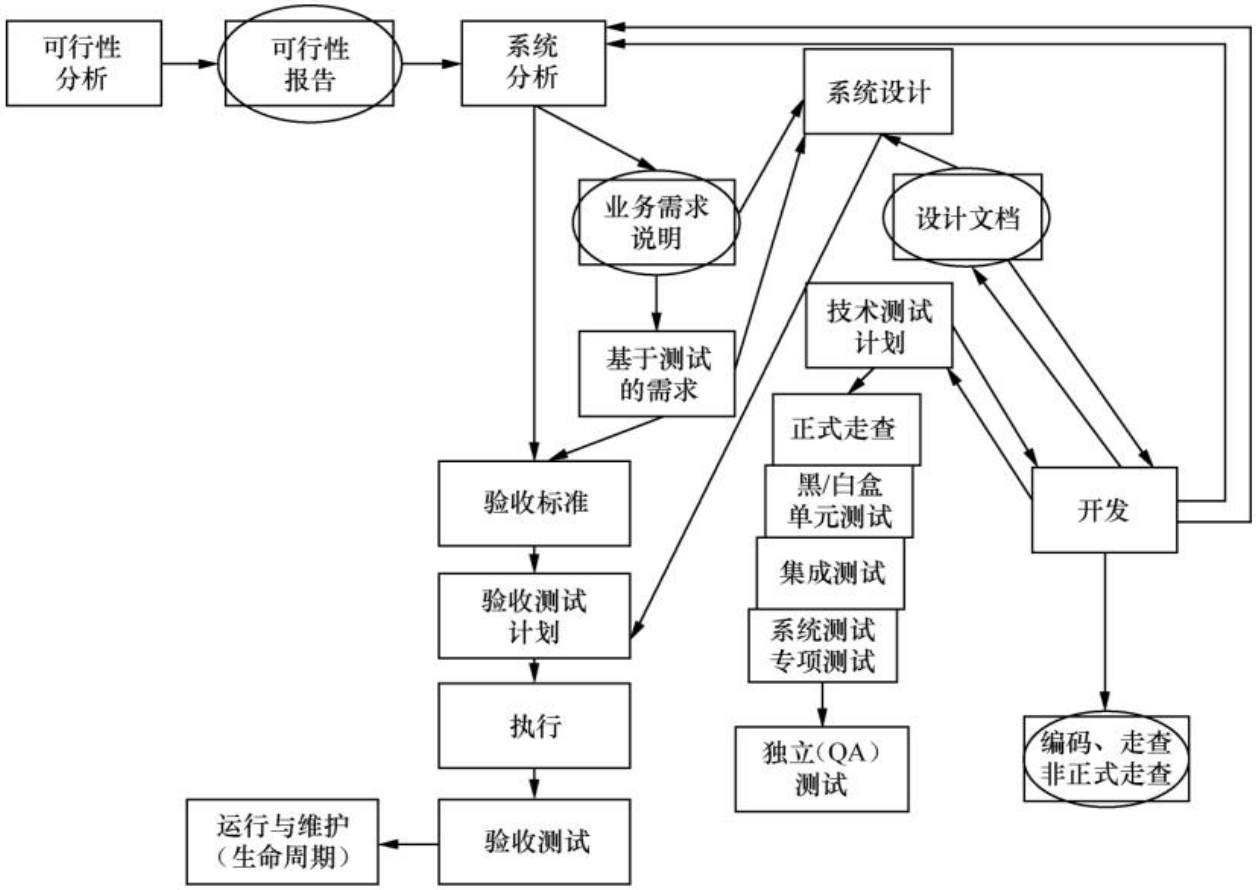


图2-11 前置测试模型

2.3.3 常见软件测试模型

- 前置测试模型

前置测试模型表示了项目声明周期从开始到结束之间的关键行为。它对每个交付内容进行测试（图2-11中的椭圆框表示了其他一些要测试的对象），在设计阶段制订测试计划和进行测试设计，让验收测试和技术测试保持相互独立。总之，它是一个将测试和开发紧密结合的模型。

前置测试模型能给开发人员、测试人员、项目经理和用户等带来很多不同于传统方法的内在的价值。与以前的测试模型中很少划分优先级所不同的是，前置测试模型用较低的成本及早发现错误，并且充分强调了测试对确保系统的高质量的重要意义。它不仅能节省时间，而且能减少那些令开发人员十分厌恶的重复工作。

2.3.3 常见软件测试模型

这些测试模型中，V 模型强调了在整个软件项目开发中需要经历的若干个测试级别，但是它没有明确指出应该对软件的需求、设计进行测试。在这一点上，W模型给出了补充，但是与V模型一样没有专门针对测试进行流程说明。随着软件测试的不断发展，第三方测试已经独立出来的时候，H 模型就得到了相应的体现，它表现为测试独立。X 模型和前置测试模型又在此基础上增加了许多不确定因素的处理情况，这就对应了在实际的项目中经常发生变更的情况。

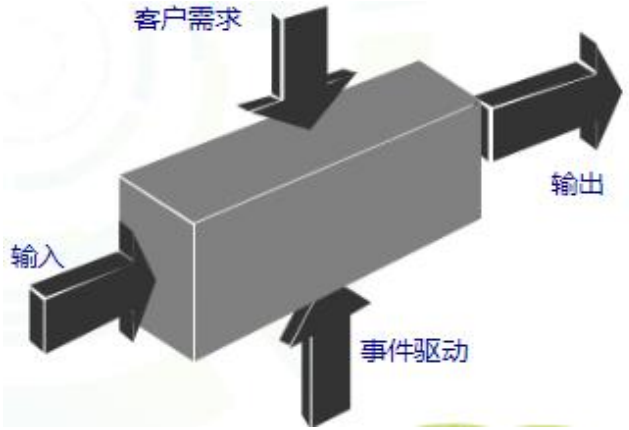
在实际的项目中，要合理应用这些测试模型的优点。例如，在W模型下，合理运用H模型的思想进行独立的测试，或者在前置测试模型中，参考X模型的一个程序片段也需要相关的集成测试的理论等，将测试与开发紧密结合，寻找最适合的测试方案。

- 2.1 软件开发过程及模型
- 2.2 软件测试过程
- 2.3 软件测试策略
- 2.4 黑盒测试和白盒测试
  - 2.4.1 黑盒测试
  - 2.4.2 白盒测试
  - 2.4.3 黑盒测试与白盒测试的比较

2.4.1 黑盒测试

- 黑盒测试

黑盒测试也称功能测试，通过测试来检测每个功能是否能够正常使用。在测试中，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下对程序进行测试。它只检查程序功能是否能按照需求规格说明书的规定正常使用，程序是否能适当地接受输入数据而产生正确的输出信息。



2.4.1 黑盒测试

黑盒测试有两个重要的优点

- 黑盒测试与软件的具体实现方式无关，因此软件实现方式如果发生了变更、修改但功能测试不变，仍可以使用原来的测试用例。
- 在进行软件开发的同时，也可以进行软件黑盒测试用例的设计，这样可以节省一部分时间成本，减少总开发时间。

常见的黑盒测试方法有等价类划分、边界值设计、因果图分析和正交试验法等

黑盒测试的常用工具

- **QACenter**

  QACenter主要包括功能测试工具QARun、性能测试工具QALoad、应用可用性管理工具EcoTools和应用性能优化工具EcoScope。

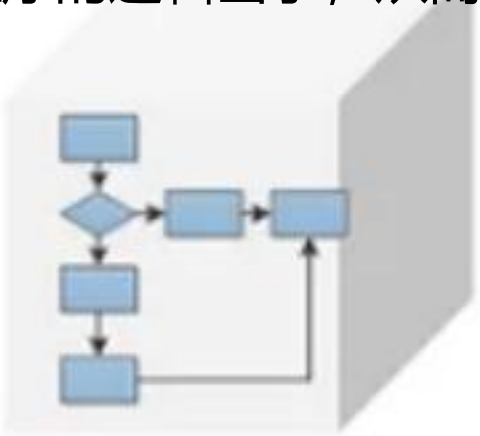
- **WinRunner**

  WinRunner 是一种企业级的功能测试工具能够有效地帮助测试人员对复杂的企业级应用的不同发布版本进行测试，提高测试人员的工作效率和质量，确保跨平台的、复杂的企业级应用无故障发布及长期稳定运行。WinRunner 具有以下几个显著的功能：创建测试、插入检查点、检验数据、增强测试、运行测试、分析结果与维护测试。

2.4.2 白盒测试

- 白盒测试

白盒测试又称结构测试、透明盒测试、逻辑驱动测试或基于代码的测试。白盒测试是一种测试用例设计方法，盒子指的是被测试的软件。对于白盒测试来说，“盒子”是可视的，测试人员可以看到盒子内部的东西并且了解程序的运作过程。白盒测试需全面了解程序内部逻辑结构，对所有逻辑路径进行测试。白盒测试是穷举路径测试，测试人员必须了解程序的内部结构，从检查程序的逻辑出手，从而得出测试数据。



2.4.2 白盒测试

在白盒测试的使用流程中，必须遵从以下规则

- 一个模块中的所有独立路径都需至少得到一次测试。
- 所有逻辑值的真与假情况都需要被测试到。
- 为了保证程序结构的有效性，需要检查程序的内部逻辑结构。
- 在程序的上、下边界与可操作范围内都能保证循环的顺利运行。

白盒测试的工具

- **Jtest**

  Jtest是一个代码分析和动态类、组件测试工具，是一个集成的、易于使用和自动化的Java单元测试工具。它可以增强代码的稳定性，防止软件错误。

- **Jcontract**

  Jcontract用于系统级验证或测试部件是否正确工作并被正确使用。Jcontract是一个独立工具，在功能上是Jtest 的补充。可以用Jcontract插装按DbC（Design by Contract）注解的Java代码。当将类或部件组装成系统时，Jcontract在运行时监视并报告错用和功能性问题。Jcontract可帮助开发人员有效地考核类或部件的系统级行为。

2.4.2 白盒测试

白盒测试的工具

- **C++Test**

  C++Test 可以帮助开发人员防止软件错误，保证代码的健全性、可靠性、可维护性和可移植性。

- **CodeWizard**

  CodeWizard是代码静态分析工具、先进的C/C++源代码分析工具，使用超过500个编码规范自动化地标明有危险的，但是编译器不能检查到的代码结构。

- **Insure++**

  Insure++是一个基于C/C++的自动化内存错误、内存泄漏精确检测工具。

黑盒测试与白盒测试的主要区别在以下几个方面。

- **已知产品的因素**

黑盒测试	白盒测试
黑盒测试已知程序的功能需求、设计规格，可以通过测试验证程序需要的功能是否已被实现、是否符合要求。	白盒测试已知程序的内部工作结构，可以通过测试验证程序的内部结构是否符合要求、是否含有软件缺陷。

2.4.3 黑盒测试与白盒测试的比较

黑盒测试与白盒测试的主要区别在以下几个方面。

- **检查测试的主要内容**

黑盒测试

黑盒测试主要检查的内容包括但不限于以下几条：

（1）功能是否都满足需求、是否有功能出现软件缺陷。
（2）接口上是否能正确接受输入、输出结果是否正确。
（3）是否有数据结构信息或者外部信息访问错误。
（4）是否有初始化或终止性错误。

白盒测试

白盒测试主要检查的内容包括但不限于以下几条：

（1）所有程序模块的独立路径都需要至少被测试一遍。
（2）所有的逻辑判定的真值与假值都需要至少被测试一遍。
（3）在运行的界限内和循环的边界上执行循环体。
（4）测试内部的数据结构是否有效。

2.4.3 黑盒测试与白盒测试的比较

黑盒测试与白盒测试的主要区别在以下几个方面。

- **静态测试方法**

黑盒测试	白盒测试
静态黑盒测试检查产品需求文档、用户手册、帮助文件等。	静态白盒测试走查、复审、评审程序源代码、数据字典、系统设计文档、环境设置、软件配置项等。

黑盒测试与白盒测试的主要区别在以下几个方面。

- **动态测试方法**

黑盒测试

动态黑盒测试通过数据输入并运行程序来检验输出结果，如功能测试、验收测试和一些性能测试（或兼容性、安全性）等

白盒测试

动态白盒测试通过驱动程序来调用，如进行单元测试、集成测试和部分性能测试（或可靠性、恢复性）等。