CHAPTER 1 第1章

软件测试项目管理概述

1.1 概述

随着科学技术的迅速发展，人类已经进入信息社会。信息的获取、处理、交流和决策都需要大量的软件，软件成为人们工作和生活中不可或缺的工具。软件的应用日益广泛，软件的质量受到人们越来越多的关注。特别是在航空航天、金融保险、交通通信、工业控制等重要领域，软件一旦失效将造成重大损失，因此这些领域对软件质量也提出了更高的要求。

1996 年 6 月，阿丽亚娜 5(ARIANE 5)型运载火箭在历时 10 年研制后的首次发射中，软件故障造成火箭升空 40 s 后即发生爆炸，直接经济损失达到 5 亿美元，还使得耗资 80 亿美元的开发计划推迟了近 3 年。2003 年 5 月，俄罗斯 TMA1 号宇宙飞船，由于软件错误导致导航系统故障，造成飞船从国际空间站返回地面时与飞行控制中心失去联系长达 11 min，飞船最终降落在与预定降落点偏差超 460 km 的地方。2003 年 8 月，First Energy 公司电力监测与控制管理系统的预警服务出现软件错误，导致多个重要设备出现故障，而操作员在一个小时后才得到控制站的指示，造成美国及加拿大部分地区发生史上最大停电事故。

软件测试是保障软件质量的重要手段，是构建高可信软件的关键环节。随着人们对软件测试重要性的认识越来越深刻，软件测试阶段在整个软件开发周期中所占的比例日益增大。统计数据表明，软件测试占软件开发总成本的比例一般达到 50%以上[1]。现在有些软件开发机构将 40%以上的研制力量投入软件测试之中；对于某些性命攸关的软件，其测试费用甚至高达所有其他软件工程阶段费用总和的 3～5 倍。尽管人们在软件开发过程中也采用形式化方法描述和证明软件规约，并采用程序正确性证明、模型检验等方法保证软件质量，但是这些方法都存在一定的局限性，尚未达到广泛实用阶段。软件测试在今后较长时间内仍将是保证软件质量的重要手段。

1.1.1 软件测试简史

人们对软件测试的认识是逐步发展起来的，如图 1-1 所示。在 20 世纪 50 年代，计算机技术发展初期，软件规模都很小，复杂度相对较低，大部分软件错误在开发人员的调试阶段就被发现并解决了，软件测试被定义为“程序员为了在他们的程序中找到缺陷(bug)所做的事情”。在这个阶段，测试和调试是等同的，都由开发人员自己完成。

20 世纪 60 年代早期，软件测试与调试区分开来，测试主要在程序编写后进行。人们开

始考虑以遍历代码的可能路径或可能的输入变量的方式，对软件进行“彻底”的测试。现在我们知道，由于程序的输入域太大、有太多可能的输入路径及设计和规约等问题很难测试，完全、彻底地测试一个程序是不可能的。

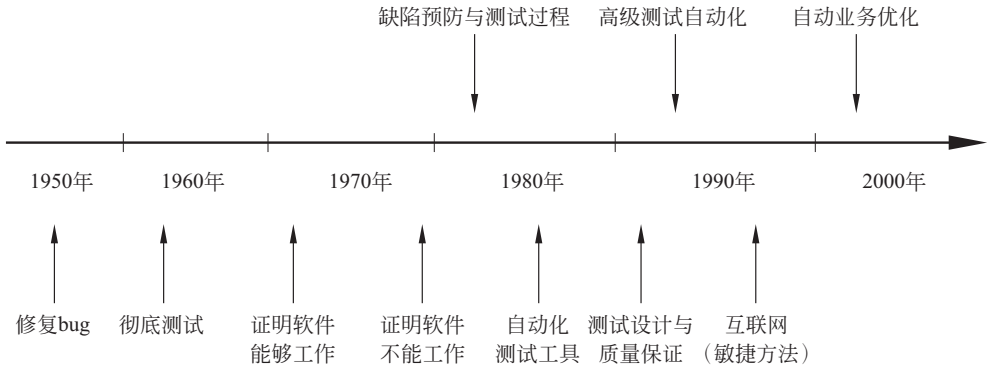

图 1-1 软件测试的发展

20 世纪 70 年代早期，随着软件开发的成熟，人们开始提出软件开发的工程化思想，软件测试得到发展，软件测试的地位得到确认。软件测试被定义为“证明一个程序的正确性而要做的事情”，即证明软件能够工作。这期间提出的“正确性证明技术”，建议在软件分析、设计和实现过程中通过证明的方式进行软件正确性验证，在理论上很有前途，但在实践中过于耗时、效率极低。

20 世纪 70 年代后期，测试被定义为带着找到错误的意图执行程序的过程，而不是证明程序能够运行的过程。与早期观点相反，这种观点强调了好的测试用例能够有更大概率去发现尚未发现的错误，成功的测试是发现了尚未发现的错误的测试。

20 世纪 80 年代，随着软件行业的飞速发展，软件测试的理论和技术得到了快速的发展，人们开始把软件测试作为软件质量保证的重要手段，认识到软件测试不是一次性在开发后期完成的，软件测试也不再仅限于程序本身，软件测试的定义被扩展到缺陷预防，软件测试活动被扩充到对需求、设计、编码、测试等整个软件开发生存周期的过程控制中。IEEE 对软件测试进行了定义：“使用人工或自动的手段来运行或测定某个软件系统的过程，其目的在于检验它是否满足规定的需求或弄清楚预期结果与实际结果之间的差别”，并发布了软件测试文档的国际标准。

20 世纪 80 年代中期，自动化测试工具出现了。相比于手动测试，自动化测试工具的引入大幅提高了测试的效率和质量。测试工具起初还是相当原始的，不具有高级脚本语言工具。

20 世纪 90 年代早期，从质量保证的观点，测试的含义被定义为“策划、设计、建造、维护和执行测试及测试环境”，软件测试的过程应是受管理的，其自身也存在一个生存周期。更多的录制/回放测试工具提供了丰富的脚本语言和报告功能，测试管理工具可帮助管理从测试需求分析和设计到测试脚本及缺陷的所有内容，也有一些商用的性能工具来测试系统的性能，能够进行压力和负载测试等。

20 世纪 90 年代中期，人们仍认为测试应是一个贯穿整个软件开发生存周期的过程，但是随着互联网的流行，软件生存周期模型和开发模式发生了很大的变化，使得软件测试变得

越来越困难。测试有时在没有明确预先定义所有测试方向的情况下进行，这种测试方法被认为是敏捷测试，其他测试技术还有探索测试、快速测试和基于风险的测试等。

随着信息科学和软件开发技术的发展，人们对软件测试的概念及其作用的认识已趋于成熟和稳定，进入 21 世纪以来，对软件测试的研究主要集中于软件测试的技术方法和最佳实践上，目的是提升软件测试的效率和效果。自动业务优化(BTO)的基本思想是衡量软件在其整个生存周期中的价值并使其最大化，以确保软件达到可用性、性能、质量和经济上相协调的目标。软件测试作为软件整个生存周期的一项重要活动被内置其中[2]。

由于人们对软件质量越来越关注，对软件测试越来越重视，可以预测，在未来很长的时间内，软件测试将会得到快速发展，软件测试理论将更加完备，软件测试工具将更加丰富，软件测试行业将蓬勃发展，测试人员的素质和能力将不断提升。测试有效性和效率最大化是软件测试的共性问题，测试自动化和智能化将成为软件测试的重要发展方向之一。

1.1.2　软件测试定义

在 GB/T 11457—2006《信息技术软件工程术语》中，测试被定义为："①在规定的条件下操作系统或部件、观察或记录结果并对系统或部件的某些方面作评价的过程；②分析软件项以检测在存在的和要求的条件之间的区别(隐错)以评价软件项的特征"。也就是说，软件测试可分为两部分：确认和验证。

"确认"指的是检查所开发的软件是否满足用户真正的需求。从用户的真正需要出发，对软件需求和设计说明存疑，以发现软件需求定义、产品设计和程序实现中的问题。

"验证"指的是检验软件是否正确实现了软件需求规格说明、设计说明等文档中所定义的功能和特性的活动。验证过程应提供证据以表明软件产品与研制要求相一致，或表明两者之间的差异。

人们对软件测试的认识是不断深入的。随着软件测试的持续发展，曾出现了许多对软件测试很重要的观点或定义。这些观点一度非常流行，有些至今仍很有价值，表现了人们从特定角度对软件测试的认识。

"(软件测试)就是建立一种信心，表明程序能够按照预期的设想运行"，这是软件测试先驱 Bill Hetzel 在 1973 年给出的软件测试的定义。后来，Bill Hetzel 在其 1983 年出版的经典著作 *The Complete Guide to Software Testing* 中，对上述定义进行了修订："测试是评价一个程序或者系统的属性和能力的各种活动，并确定它是否达到预期的结果。"Bill Hetzel 的核心观点是：测试的目的是验证软件是"工作的"，即以正向思维方式，针对软件系统的所有功能点，逐个验证其正确性。这种观点比较符合用户的角度。对于用户来说，适当的测试是能够接受软件或系统的依据。用户通过亲自的或委托第三方的测试，获得测试结果，对软件系统的质量进行判断并做出最后的抉择。

"测试是为了发现错误而执行程序的过程"，这是另一位软件测试先驱 Glenford Myers 1979 年在其著名的著作 *The Art of Software Testing* 中给出的定义。后来，Glenford Myers 对该定义进行了完善，进一步提出了他对软件测试的重要观点：测试是为了证明程序有错，而不是证明程序无错误；一个好的测试用例在于它能发现至今未发现的错误；一个成功的测试是发现了至今未发现的错误的测试。这种观点的核心是证明软件是"不工作的"，

即以反向思维，不断思考软件或系统的弱点，发现软件或系统中存在的问题。这种观点比较符合软件技术人员的角度。对开发方来讲，软件测试可尽早地发现和改正软件中的错误，避免给用户造成损失和给开发方的信誉造成不良影响。

“测试是使用人工或自动的手段来运行或测定某个软件系统的过程，其目的是检验软件系统是否满足规定的需求或弄清预期结果与实际结果之间的差别”，这是 1983 年 IEEE 软件工程术语标准中对软件测试给出的定义。这个定义明确指出软件测试的目的是检验软件系统是否满足需求。软件测试不再是一个一次性的、软件开发后期的活动，而是与整个开发流程融为一体。现在我们认为，软件测试是贯穿整个软件开发生存周期、对软件产品（包括阶段性产品）进行验证和确认的活动过程，其目的是尽早、尽快地发现软件产品中存在的各种问题及与用户需求、预先定义不一致的地方。

软件测试还存在如下重要特征：软件测试是不完全的或无法穷尽的；软件测试是证错而不是证对的，即不完全的软件测试无法证明软件的正确性，只能证明软件的不正确性。

软件测试是不完全的或无法穷尽的。这主要是因为如下 3 点。

（1）测试是几乎不可能 100％覆盖的。一般来说，由于软件的复杂性和大规模，软件的输入空间非常庞大，程序中可以执行的路径无法穷尽。如果有充足的条件不断地进行测试，总是可以找到更多的缺陷。

（2）测试环境难以和实际运行或用户环境完全吻合。某些缺陷只有在用户环境下才存在，有些缺陷只有在软件运行一段时间后，随着过多的过量数据或无效数据的积累才会发生。

（3）测试人员对产品的理解不能完全代表实际用户的理解。这两者之间的差异意味着可能会存在某些对测试人员来说不是缺陷但对于用户来说是缺陷的缺陷。

软件测试是证错而不是证对的，即不完全的软件测试无法证明软件的正确性，只能证明软件的不正确性。这主要是因为如下两点。

（1）发现的错误越多并不能说明软件中剩余的错误就越少，相反地，发现错误越多的地方往往漏掉错误的可能性越大。软件错误存在“群集现象”，根据“二八原理”，20％的代码可能包含了软件中 80％的错误。

（2）修正过去的缺陷往往会导致新缺陷的产生。需求总是变化的，软件系统不是一成不变的，变是永恒的，“变”可能会带来新的缺陷。

1.2 软件测试项目阶段要求

1.2.1 测试需求分析与测试策划

1.2.1.1 测试需求分析的管理要点

1）测试需求分析

测试需求分析应根据软件测评任务书、合同或其他等效文件，以及被测软件的软件需求规格说明或设计文档，对测评任务进行测试需求分析，分析中应包括：

(1) 确定需要的测试类型及其测试要求并进行标识(编号),标识应清晰、便于识别。测试类型包括功能测试、性能测试等类型,测试要求包括状态、接口、数据结构、设计约束等要求。确定的测试类型和测试要求均应与合同中提出的测试级别(单元测试、部件测试、配置项测试、系统测试)和测试类型相匹配。

(2) 确定测试类型中的各个测试项及其优先级。

(3) 确定每个测试项的测试充分性要求。根据被测软件的重要性、测试目标和约束条件,确定每个测试项应覆盖的范围及范围所要求的覆盖程度。

(4) 确定每个测试项测试终止的要求,包括测试过程正常终止的条件(如测试充分性是否达到要求)和导致测试过程异常终止的可能情况。

应建立测试类型中的测试项与软件测评任务书、合同或其他等效文件,以及被测软件的需求规格说明或设计文档的追踪关系;应将测试需求分析结果按所确定的文档要求形成测试需求规格说明;测试需求规格说明应经过评审,并应受到变更控制和版本控制。

2) 测试策划

应根据软件测评任务书、合同或其他等效文件,以及软件需求规格说明和设计文档进行测试策划,策划一般包括:

(1) 确定测试策略,如部件测试策略;

(2) 确定测试需要的技术或方法,如测试数据生成与验证技术、测试数据输入技术、测试结果获取技术等;

(3) 确定要受控制的测试工作产品,列出清单;

(4) 确定用于测试的资源要求,包括软硬件设备、环境条件、人员数量和技能等要求;

(5) 进行测试风险分析,如技术风险、人员风险、资源风险和进度风险等;

(6) 确定测试任务的结束条件,根据软件测评任务书、合同或其他等效文件的要求和被测软件的特点确定结束条件;

(7) 确定被测软件的评价准则和方法;

(8) 确定测试活动的进度,应根据测试资源和测试项,确定进度;

(9) 确定需采集的度量及采集要求,应根据测试的要求,确定要采集的度量,特别是测试需求度量、用例度量、风险度量、缺陷度量等,并应明确相应的数据库。

可建立测试计划与测试需求规格说明的追踪关系;应将测试策划的结果,按所确定的文档要求形成测试计划。

1.2.1.2　测试需求分析阶段评审

针对测试需求规格说明的评审:

(1) 测试级别和测试对象所确定的测试类型及其测试要求是否恰当;

(2) 每个测试项是否进行了标识,并逐条覆盖了测试需求和潜在需求;

(3) 测试类型和测试项是否充分;

(4) 测试项是否包括了测试终止要求;

(5) 文档是否符合规定的要求。

测试计划也应经过评审,并应受到变更控制和版本控制。

1.2.2 测试设计和实现

1.2.2.1 测试设计与实现的管理要点

测试设计和实现阶段的主要工作包括以下几部分：

(1) 设计测试用例；

(2) 编写测试程序；

(3) 生成测试数据；

(4) 验证测试环境。

测试设计与实现的工作要点包括以下 3 点。

1) 应根据测试需求规格说明和测试计划进行测试设计和实现，应完成如下工作。

(1) 按需要分解测试项。将需测试的测试项进行层次化的分解并进行标识，若有接口测试，还应有高层次的接口图说明所有接口和要测试的接口。

(2) 说明最终分解后的每个测试项，说明测试用例设计方法的具体应用、测试数据的选择依据等。

(3) 设计测试用例。

(4) 确定测试用例的执行顺序。

(5) 准备和验证所有的测试用数据。针对测试输入要求，设计测试用的数据，如数据类型、输入方法等。

(6) 准备并获取测试资源，如测试环境所必须的软、硬件资源等。

(7) 必要时，编写测试执行需要的程序，如开发部件测试的驱动模块、桩模块及测试支持软件等。

(8) 建立和校核测试环境，记录校核结果，说明测试环境的偏差。

2) 应将测试设计与实现的工作结果，按照所确定的文档要求编写测试说明，测试说明一般应包括以下几点。

(1) 测试名称和项目标识。

(2) 测试用例的追踪。说明测试所依据的内容来源，并跟踪到相应的测试项的标识(编号)。

(3) 测试用例说明。简要描述测试的对象、目的和所采用的测试方法。

(4) 测试用例的初始化要求，包括硬件配置、软件配置(包括测试的初始条件)、测试配置(如用于测试的模拟系统和测试工具)、参数设置(如测试开始前对断点、指针、控制参数和初始化数据的设置)等的初始化要求。

(5) 测试用例的输入。每个测试用例输入的描述中包括：

① 每个测试输入的名称、用途和具体内容(如确定的数值、状态或信号等)及其性质(如有效值、无效值、边界值等)；

② 测试输入的来源(如测试程序产生、磁盘文件、通过网络接收、人工键盘输入等)，以及选择输入数据所使用的方法(如等价类划分、边界值分析、猜错法、因果图以及功能图等)；

③ 测试输入是真实的还是模拟的；

④ 测试输入的时间顺序或事件顺序。

(6) 测试用例的期望测试结果。期望测试结果应有具体内容(如确定的数值、状态或信号等),不应是不确切的概念或笼统的描述。必要时,应提供中间的期望结果。

(7) 测试用例的测试结果评估准则。评估准则用以判断测试用例执行中产生的中间或最后结果是否正确。评估准则应根据不同情况提供相关信息,如:

① 实际测试结果所需的精确度;

② 允许的实际测试结果与期望结果之间差异的上、下限;

③ 时间的最大间隔或最小间隔;

④ 事件数目的最大值或最小值;

⑤ 实际测试结果不确定时,重新测试的条件;

⑥ 与产生测试结果有关的出错处理;

⑦ 其他有关准则。

(8) 实施测试用例的执行步骤。编写按照执行顺序排列的一系列相对独立的步骤,执行步骤应包括:

① 每一步所需的测试操作动作、测试程序输入或设备操作等;

② 每一步期望的测试结果;

③ 每一步的评估准则;

④ 导致被测程序执行终止伴随的动作或指示信息;

⑤ 需要时,获取和分析中间结果的办法。

(9) 测试用例的前提和约束。在测试用例中还应说明实施测试用例的前提条件和约束条件,如特别限制、参数偏差或异常处理等,并要说明它们对测试用例的影响。

(10) 测试终止条件。说明测试用例的测试正常终止和异常终止的条件。

3) 确定测试说明与测试计划或测试需求规格说明的追踪关系,给出清晰、明确的追踪表。

1.2.2.2　测试设计与实现阶段评审

测试说明应经过评审,得到相关人员的认同,受到变更控制和版本控制。根据测试实际情况,修订测试说明。

测试说明评审应关注:

(1) 测试说明是否完整、正确和规范;

(2) 测试设计是否完整和合理;

(3) 测试用例是否可行和充分。

在测试计划评审和测试说明评审后,还必须进行测试就绪评审,以确定能否开始执行测试。测试就绪评审应关注:

(1) 通过比较测试环境与软件真实运行的软件、硬件环境的差异,审查测试环境要求是否正确合理、满足测试要求;

(2) 审查测试活动的独立性和公正性;

(3) 审查测试需求规格说明、测试计划和测试说明评审中的遗留问题是否得到了解决;

(4) 审查是否存在影响测试执行的其他问题。

1.2.3 测试执行

应按照测试计划和测试说明的内容与要求执行测试。

应如实填写测试原始记录，当结果有量值要求时，应准确记录实际的量值。原始记录应：

(1) 受到严格管理；

(2) 规范格式；

(3) 至少包括测试用例标识、测试结果和发现的缺陷。

应根据每个测试用例的期望测试结果、实际测试结果和评估准则，判定测试用例是否通过。

当测试用例不通过时，应根据不同的缺陷类型，采取相应的措施：

(1) 将测试工作中的缺陷，如测试说明的缺陷、测试数据的缺陷、执行测试步骤时的缺陷、测试环境中的缺陷等，记录到相应的表格中(如“问题及变更报告”)，并实施相应的变更；

(2) 将被测软件的缺陷记录到软件问题报告中，软件问题报告的格式应规范。

当所有的测试用例都执行完毕后，应根据测试的充分性要求和有关原始记录，分析测试工作是否充分，是否需要进行补充测试：

(1) 当测试过程正常终止时，如果发现测试工作不足，或测试未达到预期要求，应进行补充测试；

(2) 当测试过程异常终止时，应记录导致终止的条件、未完成的测试或未被修正的错误。

在执行测试的过程中，可根据测试的进展情况补充测试用例，但应留下用例记录，并在执行测试后，变更测试说明。

1.2.4 测试总结

1.2.4.1 测试总结的管理要点

应根据软件测评任务书、合同(或其他等效文件)、被测软件文档、测试需求规格说明、测试计划、测试说明、测试记录、测试问题及变更报告单和被测软件问题报告等，对测试工作和被测软件进行分析和评价。

对测试工作的分析和评价应包括：

(1) 总结测试需求规格说明、测试计划和测试说明的变化情况及其原因；

(2) 在测试异常终止时，说明未能被测试活动充分覆盖的范围及其理由；

(3) 确定无法解决的软件测试事件并说明不能解决的理由。

对被测软件的分析和评价应包括：

(1) 总结测试中所反映的被测软件与软件需求(或软件设计)之间的差异；

(2) 可能时，根据差异评价被测软件的设计与实现，提出改进的建议；

(3) 当进行配置项测试或系统测试时，如有需要，测试总结中应对配置项或系统的性能

作出评估，指明片查、缺陷和约束条件等对于配置项或系统运行的影响。

应根据软件测评任务书、合同（或其他等效文件）、被测软件文档、测试需求规格说明、测试计划、测试说明、测试记录和软件问题报告等有关文档，对测试结果和问题进行分类和总结，按所确定的文档要求编写测试报告或测评报告。测评报告除了应包括对测试结果的分析外，还应包括对被测软件的评价和建议，测评报告和测试报告有时可合并。

宜分析测评项目中的数据和文档，以供以后的测试使用。数据如缺陷数据（包括缺陷描述、类型、严重性等）、用例数据、管理数据（如生产率、工作量、进度等）；文档如好的用例设计、好的需求规格说明等。

1.2.4.2　测试总结评审

测试总结评审应在前述各项工作完成后进行，以确定是否达到测试目的，给出评审结论。评审的具体内容和要求是：

（1）审查测试文档与记录内容的完整性、正确性和规范性；

（2）审查测试活动的独立性和有效性；

（3）审查测试环境是否符合测试要求；

（4）审查软件测试报告与软件测试原始记录和问题报告的一致性；

（5）审查实际测试过程与测试计划和测试说明的一致性；

（6）审查测试说明评审的有效性，如是否评审了测试项选择的完整性和合理性、测试用例的可行性和充分性；

（7）审查测试结果的真实性和准确性。

1.3　软件测试项目管理模型

软件开发活动是一个过程，遵循特定的生存周期模型。无论什么样的软件开发生存周期模型，软件测试都是其中必不可少的环节，是贯穿整个生存周期的重要活动。本节通过对 V 模型、W 模型、H 模型等常见模型的分析，介绍软件测试在不同软件生存周期中的位置和起到的作用。

1.3.1　V 模型

V 模型最早于 20 世纪 80 年代后期由 Paul Rook 在软件开发瀑布模型的基础上提出，是最经典和具有代表意义的测试模型。V 模型反映了软件测试活动与软件需求分析和设计的关系，明确指出软件测试不仅仅是软件开发中的一个独立阶段，而应贯穿整个软件开发生存周期。

V 模型分为左右两部分，如图 1-2 所示，左半部分描述基本的软件开发过程，按照箭头的方向从上到下分为不同的开发阶段；右半部分描述与开发相对应的测试过程，按照箭头的方向自下而上分为不同的测试类别。右半部分中的软件测试类别与左半部分中的软件开发阶段有着同等的重要性，在形状上呈现 V 字形，故称为 V 模型。V 模型中的左右两部分存

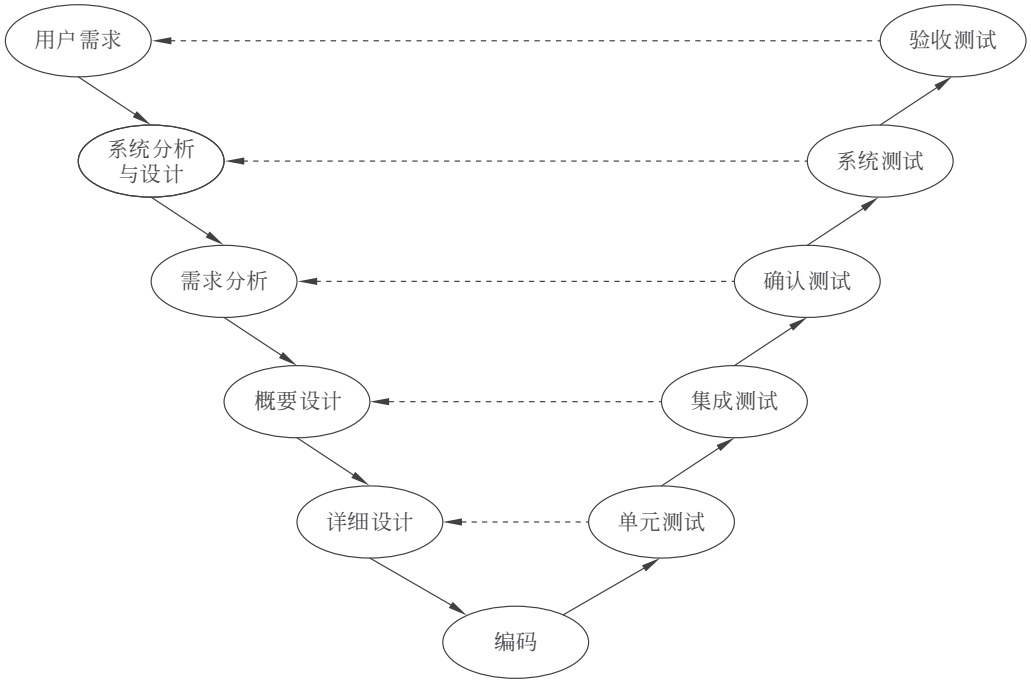

图 1-2 V 模型示意图

在连通关系，使得不同的测试类别与相应的开发阶段对应，不仅说明了要做什么事，还说明了在什么时间及如何来实施这些任务。

V 模型的主要不足在于：它把测试置于软件开发活动之后，对“尽早测试和不断测试”的原则体现不充分；它把不同类别的测试与软件开发阶段一一对应，但在实际项目中，各级测试与不同开发阶段很难有严格的对应关系；它容易使人理解为测试就是针对程序进行的，缺少需求评审、设计评审、代码审查等验证确认与静态测试内容，有可能导致需求分析、概要设计等早期开发阶段的问题直至系统测试和验收测试才能被发现，从而错过最佳的缺陷发现和修复时机。

1.3.2 W 模型

W 模型是在 V 模型的基础上，由 Paul Herzlich 在 1999 年提出的。相对于 V 模型，W 模型增加了各软件开发阶段中同步进行的验证和确认活动。

W 模型由两个 V 模型组成，如图 1-3 所示，开发过程是一个 V 字形，伴随的测试过程是另一个 V 字形，两者是并行关系。W 模型强调测试是伴随着整个软件开发周期的，而且测试对象不仅仅是程序，还应包括需求、设计等阶段的工作产品，也就是说，测试和开发是同步进行的。W 模型有助于尽早、全面地发现问题。例如，在需求分析完成后，测试人员就应该参与到对需求的验证和确认活动中，以及时地找出存在的错误；同时，对需求的测试也有利于了解掌握项目情况和测试风险，尽早制定应对措施。

W 模型的不足主要在于：把软件开发和测试都看作串行的活动，呈现出一种线性关系，