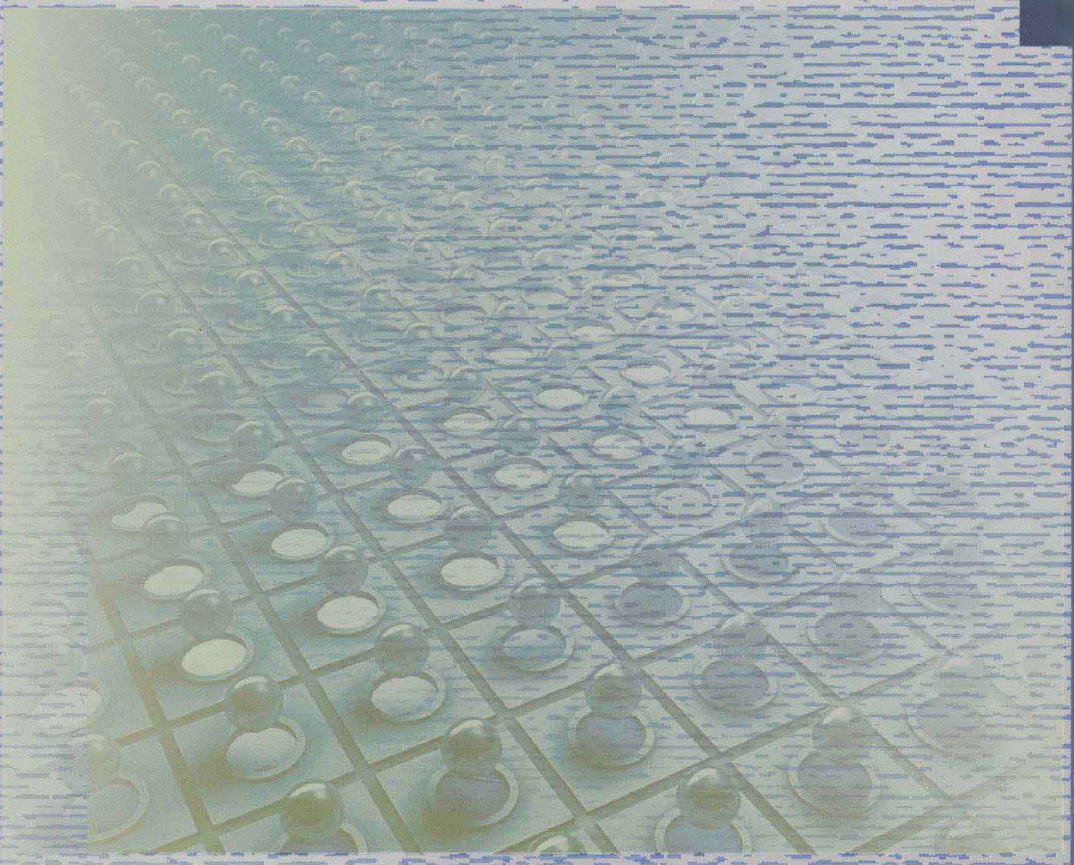
计算智能与组合软件测试优化

王曙燕 潘晓英 孙家泽 等 著



科学出版社

计算智能与组合软件测试优化

王曙燕　潘晓英　孙家泽　等　著

科学出版社

内 容 简 介

本书面向智能系统学科的前沿领域，针对软件测试中的热门关键技术，系统地讨论了组合软件测试中的关键技术以及计算智能方法在该领域中的应用，比较全面地反映了国内外智能组合软件测试的最新研究进展。内容包括软件测试基础、计算智能理论基础、软件可靠性、软件可靠性建模、基于群体智能的软件可靠性分配、面向故障的软件测试、灰盒测试、组合覆盖准则、基于粒子群算法的测试用例自动生成与约简、基于模拟退火的测试用例自动生成与约简、基于协同进化的测试用例自动生成与约简和基于族群进化算法构造最有组合测试用例集。

本书取材新颖、内容深入浅出，可作为软件工程专业研究生及高年级本科生的教材，也可作为软件测试人员的参考用书，还可供从事相关领域研究的教师和科技工作者参考使用。

图书在版编目(CIP)数据

计算智能与组合软件测试优化 / 王曙燕等著. —北京：科学出版社，2013.6
ISBN 978-7-03-037863-7

I. ①计… II. ①王… III. ①计算智能－应用－组合软件－测试－最优化 IV. ①TP311.5

中国版本图书馆 CIP 数据核字(2013)第 132443 号

责任编辑：潘斯斯 张丽花 / 责任校对：刘 洋
责任印制：闫 磊 / 封面设计：迷底书装

科 学 出 版 社 出版
北京东黄城根北街 16 号
邮政编码：100717
http://www.sciencep.com

北京市文林印务有限公司 印刷

科学出版社发行 各地新华书店经销

*

2013 年 6 月第 一 版 开本：720×1000 B5
2013 年 6 月第一次印刷 印张：14
字数：270 000

定价：58.00 元

(如有印装质量问题，我社负责调换)

前　言

随着软件产业的更新换代和软件企业规模的扩大，软件测试的重要性日益凸显，软件测试行业在国内逐渐兴起。软件测试(Software Testing)作为软件工程学科的一个重要分支，随着软件的发展而发展，已经成为软件质量保证的关键技术之一，也是软件开发过程中的一个重要环节。

智能计算与智能优化是近年来计算机及信息学科的热点研究方向，并且在近10年来迅速发展，其应用领域也越来越广，从工业控制、模式识别、知识自动获取、经济管理、生物医学到网络智能自动化等领域都取得了激动人心的研究成果和广泛的应用。

为培养新世纪的高素质人才，并且在软件工程中的软件测试领域开辟新的解决思路，本书将软件测试基础及智能计算这两个独特的视角相结合，系统地介绍软件测试的基础知识及智能计算在组合软件测试领域的作用。全书共四部分：第一部分介绍软件测试和计算智能的相关基础知识；第二部分介绍软件可靠性的基础理论、软件可靠性模型，以及基于群体智能的软件可靠性分配；第三部分为灰盒测试与组合覆盖准则，包括面向故障的软件测试、灰盒测试和常见的组合覆盖准则；第四部分为计算智能在组合测试中的应用，包括粒子群算法、模拟退火算法、协同进化算法、族群进化算法等在组合测试用例生成和约简中的应用。

在国家自然科学基金计划、陕西省工业攻关计划的资助下，项目组从2005年开始展开相关课题的研究，在软件测试、计算智能等领域取得了一定的研究成果，特别是在智能组合软件测试方面。本书试图将这两个前沿领域的相关研究展现给广大读者，希望通过整理和总结，引起更多研究者对这些新兴领域的关注。由于其中的一些理论还处于发展阶段，书中的诸多看法只是科学研究中的一己之见，难免有失偏颇，欢迎广大读者批评指正。

本书受国家自然科学基金项目(No.61050003)的资助，是西安邮电大学软件测试项目组近8年集体智慧的结晶。本书由王曙燕主编，第1章由王曙燕编写；第2、11章由潘晓英编写；第5、9章由孙家泽编写；第4、6章由王小银编写；第7章由王春梅编写；第8章由曹小鹏编写；第10章由王博编写；第12章由陈皓编写；第3章由贾冀婷编写。王曙燕和潘晓英对全书进行了统稿。感谢科学出版社、西安邮电大学各级领导及西安邮电大学计算机学院的老师给予的大力支持和帮助。

书中存在不足之处，敬请有关专家和读者提出宝贵意见。

编　者

2013年4月

目　录

第 1 章　软件测试基础

1.1　软件测试概述

1.1.1　软件测试的起源与发展[1]

软件测试是伴随着软件的产生而产生的。在早期的软件开发过程中，软件规模很小，复杂程度低，软件开发的过程混乱无序、相当随意，测试的含义比较狭窄。开发人员将测试等同于“调试”，目的是纠正软件中已知的缺陷，常由开发人员自行完成这部分工作。对测试的投入极少，测试介入也晚，常等到形成代码，产品已经基本完成时才进行测试。

直到 1957 年，软件测试才开始与调试区别开来，成为一种发现软件缺陷的活动。由于一直存在“为了让我们看到产品在工作，就得将测试工作往后推一点”的思想，人们的潜意识里对测试的目的就理解为“使自己确信产品能工作”。测试活动始终滞后于开发的活动，测试通常被作为软件生命周期中最后一项活动而进行。当时人们也缺乏有效的测试方法，主要依靠“错误推测(Error Guessing)”来寻找软件中的缺陷。因此，大量软件交付后仍存在很多问题，软件产品的质量无法保证。

到 20 世纪 70 年代，这个阶段开发的软件仍然不复杂，但人们已开始思考软件开发流程的问题，尽管对“软件测试”的真正含义还缺乏共识，但这一词条已经频繁出现，一些软件测试的探索者建议在软件生命周期的开始阶段就根据需求制订测试计划。这时也涌现出一批软件测试的专家，Bill Hetzel 博士就是其中的领导者。1972 年，软件测试领域的先驱 Bill Hetzel 博士在美国的北卡罗莱纳大学组织了历史上第一次正式的关于软件测试的会议。

到 20 世纪 80 年代初期，软件和 IT 行业开始大发展，软件趋向大型化、高复杂度，软件的质量越来越重要。这时，软件测试的一些基础理论和实用技术开始形成，并且人们开始为软件开发设计各种流程和管理方法，软件开发的方式也逐渐由混乱无序的开发过程过渡到结构化的开发过程，以结构化分析与设计、结构化评审、结构化程序设计，以及结构化测试为特征。人们还将“质量”的概念融入其中，软件测试定义发生变化，测试不单纯是一个发现错误的过程，而且将测试作为软件质量保证(SQA，Software Quality Assurance)的主要职能，包含软件质量评价的内容。Bill Hetzel 在《软件测试完全指南》(Complete Guide of Software Testing)一书中指出：“测试是以评价一个程序或者系统属性为目标的任何一种活

动。测试是对软件质量的度量。”这个定义至今仍被引用。软件开发人员和测试人员开始共同探讨软件工程和测试问题。软件测试已有行业标准(IEEE/ANSI)，IEEE 于 1983 年提出的软件工程术语中给软件测试下的定义是：“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，其目的在于检验它是否满足规定的需求或确定预期结果与实际结果之间的差别”。这个定义明确指出：软件测试的目的是为了检验软件系统是否满足需求。它再也不是一个一次性的，而且只是开发后期的活动，而是与整个开发流程融合成一体。软件测试已成为一个专业，需运用专门的方法和手段，需有专门人才和专家来承担。

20 世纪 90 年代，软件行业迅猛发展，软件的规模变得非常大。在一些大型软件开发过程中，测试活动需花费大量的时间和成本，而当时几乎完全是手工测试，测试的效率非常低，同时，由于软件复杂度不断提高，从而出现了很多通过手工方式无法完成测试的情况。为了解决这样的问题，测试工具开始出现并逐渐盛行，测试工具的选择和推广也越来越被重视。在测试过程中，通过使用工具进行部分的测试设计、实现、执行、比较等工作，从而提高软件测试的自动化程度及测试效率。

近几年来，随着计算机和软件技术的飞速发展，软件测试技术研究也取得了很大的成果：测试专家总结了测试模型，如 V 模型、W 模型等；在测试过程改进方面提出了 TMM(Testing Maturity Model)的概念；在单元测试、自动化测试、负载压力测试，以及测试管理等方面涌现了大量优秀的软件测试工具。软件测试虽然有了如此大的发展，但仍落后于软件开发的发展水平，使得软件测试面临很大的挑战。

(1) 测试人才缺乏。我国软件产业已经获得长足的进步，但测试人员缺乏，这在很大程度上制约了软件产业的发展，因此加紧建立和健全软件测试人才培养成为当务之急。

(2) 软件测试理论不成熟。软件测试行业的兴起很大程度上取决于测试理论的成熟度。目前，软件测试过程还存在一些问题没有定论或没有明确定论，如软件测试的终止标准、如何评价测试价值等。

(3) 测试技术有待提高。目前，国内软件测试技术比较落后，手工测试比重较大，自动化的性能测试、白盒测试、代码测试、安全测试等都处于初级阶段，软件测试的质量、进度、成本和风险都无法有效保证和控制。

1.1.2 软件测试的基本概念

(1) 软件测试。软件测试是指为了发现错误而执行程序，根据软件开发各阶段的规格说明和程序的内部结构而精心设计一批测试用例，并利用这些测试用例运行程序，以发现程序错误的过程。广义指软件生存周期中所有的检查、评审和确认工作，其中包括对软件需求分析、设计阶段，以及完成开发后维护阶段的各类文档、代码的审查和确认。狭义指识别软件缺陷的过程，即实际结果与预期结果不一致。

软件测试 = 验证(Verification)+确认(Validation)：“验证”指保证软件正确实现某一特定功能的一系列活动。“确认”指的是保证软件实现满足用户需求的一系列活动。

(2) 软件测试的目的。软件测试的目的是使软件缺陷降低到一定程度，以较少的用例、时间和人力找出软件中的各种错误和缺陷，以确保软件的质量，最终确保软件的功能符合用户的需求，把尽可能多的问题在发布或交付前发现并改正。

(3) 软件测试的原则。软件测试的原则包括权衡投入/产出比(Good-enough)；保证测试覆盖程度非穷举测试；测试应追溯到用户需求；尽早测试，测试过程与开发过程相结合；测试规模由小及大，由单元到系统；由独立的第三方参与测试；不能为了便于测试而修改程序；明确测试软件该怎么办，不该怎么办；软件测试是证伪而非证真；重视无效数据和非预期使用习惯的测试；充分注意测试中的群集现象；用例定期评审，适时补充修改用例；应当全面检查每一个测试结果；测试现场保护和资料归档。

(4) 软件测试的规律。① 木桶原理。软件质量的关键因素是分析、设计和实现，测试应该是融于其中的补充检查手段，其他管理、支持，甚至文化因素也会影响软件最终的质量。测试是提高软件质量的必要条件，最直接、最快捷的手段，但绝不是一种根本手段。② 80–20 原则。分析、设计、实现阶段的复审和测试工作能够发现和避免 80%的 Bug，而系统测试又能找出其余 Bug 中的 80%，最后约 5%的 Bug 可能只有在用户的大范围、长时间使用后才会暴露出来。

(5) 测试用例。测试用例是指为某个特殊目标而编制的一组测试输入、执行条件，以及预期结果，以便测试某个程序路径或核实是否满足某个特定需求。

(6) 软件测试的重点。测试用例的设计是整个软件测试工作的核心，测试用例反映对被测对象的质量要求，决定对测试对象的质量评估。测试工作管理，尤其对于对包含多个子系统的大型软件系统，其测试工作涉及大量人力和物力，有效的测试工作管理是保证有效测试工作的必要前提。测试环境建立，测试环境应该与实际测试环境一致。

(7) 软件测试度量。软件测试度量指测试覆盖率，有多少需求、代码已经被测试。缺陷发现率，缺陷何时被发现，并且有多少缺陷已经被发现；缺陷可以根据严重程度来分类；需记录的值有缺陷数目和缺陷的严重程度。测试成功率，有多少测试已经通过，并且有多少是运行正常的，需记录的值有已通过的测试用例数目和可利用的测试用例数目。

(8) 软件测试的分类。软件测试分为功能测试、可靠性测试、容错测试、恢复测试、易用性测试、性能测试、可维护性测试、可移植性测试、安全测试、用户文档测试。

(9) 软件的可测试性。软件的可测试性包括指标可确认，可以明确确认软件是否符合要求，如有明确的要求和指标；可观察，用于确认的结果可以进行有效的观察；可控制，相对应的测试环境可以进行控制，从而保证测试有效；可分解，软件可以进行分解，对分解的结构进行测试。

软件测试作为一种有效提高软件质量的手段，不是可有可无，也不是随心所欲的，而应尽早规划规范化的软件开发的需求。

软件测试贯穿于软件定义和开发的整个期间，包括需求分析、概要设计、详细设计，以及程序编码等各个阶段所得到的文档，包括需求规范说明、概要设计规范说明、详细设计规范说明，以及源程序。

1.1.3　软件测试的分类与方法[2]

软件测试是一项复杂的系统工程，从不同的角度考虑可以有不同的划分方法。对测试进行分类是为了更好地明确测试的过程，了解测试究竟完成哪些工作，尽量全面测试。

按是否执行被测软件的角度，可分为静态测试和动态测试：前者不利用计算机运行待测程序，而应用其他手段实现测试目的，如代码审核，主要是让测试人员对编译器发现不了的潜在错误进行分析，如无效的死循环、多余的变量等；动态测试则通过运行被测试软件来达到目的。

1）按阶段划分

(1) 单元测试是对软件中的基本组成单位进行的测试，如一个模块、一个过程等。它是软件动态测试最基本的部分，也是最重要的部分之一，其目的是检验软件基本组成单位的正确性。因为单元测试需知道内部程序设计和编码的细节知识，所以一般应由程序员而非测试员来完成，往往需开发测试驱动模块和桩模块来辅助完成单元测试。因此，应用系统有一个设计很好的体系结构就显得尤为重要。

一个软件单元的正确性是相对于该单元的规约而言的。因此，单元测试以被测试单位的规约为基准。单元测试的主要方法有控制流测试、数据流测试、排错测试、分域测试等。

(2) 集成测试是在软件系统集成过程中所进行的测试，其主要目的是检查软件单位之间的接口是否正确。它根据集成测试计划，一边将模块或其他软件单位组合成越来越大的系统，一边运行该系统，以分析所组成的系统是否正确，各组成部分是否合拍。集成测试的策略主要有自顶向下和自底向上两种。

(3) 系统测试是对已经集成好的软件系统进行彻底的测试，以验证软件系统的正确性和性能等满足其规约所指定的要求，检查软件的行为和输出是否正确并非一项简单的任务，它被称为测试的“先知者问题”。因此，系统测试应该按照测试计划进行，其输入、输出和其他动态运行行为应该与软件规约进行对比。软件系统测试方法很多，主要有功能测试、性能测试、随机测试等。

(4) 验收测试旨在向软件的购买者展示该软件系统满足其用户的需求。它的测试数据通常是系统测试的测试数据的子集。所不同的是，验收测试常有软件系统的

购买者代表在现场，甚至是在软件安装使用的现场。这是软件在投入使用之前的最后测试。

(5) 回归测试是在软件维护阶段对软件进行修改之后进行的测试，其目的是检验对软件进行的修改是否正确。“修改是否正确”有两重含义：一是所作的修改达到预定目的，如改正错误，能够适应新的运行环境等；二是不影响软件其他的正确功能。

(6) Alpha 测试。Alpha 测试是指在系统开发接近完成时对应用系统的测试。测试后，仍然会有少量的设计变更。这种测试一般由最终用户或其他人员完成，不能由程序员或测试员完成。

(7) Beta 测试。Beta 测试是指开发和测试完成时所执行的测试，而最终的错误和问题需在最终发行前找到。这种测试一般由最终用户或其他人员完成，不能由程序员或测试员完成。

2) 按测试方法划分

(1) 白盒测试也称为结构测试或逻辑驱动测试，是指基于一个应用代码的内部逻辑知识，即基于覆盖全部代码、分支、路径、条件的测试。它是指导产品内部工作过程的测试，可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行；按照程序内部的结构测试程序；检验程序中的每条通路是否都有能按预定要求正确工作，而不顾它的功能。白盒测试的主要方法有逻辑驱动、基路测试等，主要用于软件验证。

“白盒”法需要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”法是穷举路径测试。在使用这一方案时，测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。贯穿程序的独立路径数可能是天文数字。即使每条路径都经过测试，仍然可能有错误：第一，穷举路径测试不能查出程序违反了设计规范，即程序本身是个错误的程序；第二，穷举路径测试不可能查出程序中因遗漏路径而产生的错误；第三，穷举路径测试可能发现不了一些与数据相关的错误。

白盒测试可以借助一些工具来完成，如 Junit Framework，Jtest 等。白盒测试的内容有如下四方面：

① 对程序模块所有的独立执行路径至少测试一次。

② 对所有的逻辑判定，取“真”与取“假”的两种情况都至少测试一次。

③ 在循环的边界和运行的边界限内执行循环体。

④ 测试内部数据结构的有效性。

(2) 黑盒测试是指不基于内部设计和代码的任何知识，而基于需求和功能的测试。黑盒测试也称为功能测试或数据驱动测试。它是已知产品所应具有的功能，通过测试来检测每个功能是否都能正常使用。在测试时，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，测试者在程序接口进行

测试。它只检查程序功能是否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收输入数据而产生正确的输出信息，并且保持外部信息(如数据库或文件)完整。黑盒测试方法主要有等价类划分、边值分析、因果图、错误推测等，主要用于软件确认测试。

“黑盒”法着眼于程序外部结构，不考虑内部逻辑结构，针对软件界面和软件功能进行测试。“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用时，才能以这种方法查出程序中所有的错误。实际上，测试情况有无穷多个，人们不仅测试所有合法的输入，而且还对那些不合法，但可能的输入进行测试。

黑盒测试也可以借助一些工具，如 WinRunner、QuickTestPro、Rational Robot 等。黑盒测试的内容有如下六方面：

① 如何测试功能的有效性。

② 如何测试系统行为和性能。

③ 何种类型的输入会产生好的测试用例。

④ 系统是否对特定的输入值特别敏感。

⑤ 如何分隔数据类的边界。

⑥ 系统能够承受何种数据率和数据量。

(3) 灰盒测试方法是指综合白盒测试和黑盒测试的测试方法。白盒测试又称为结构测试，在测试过程中测试者可以看到被测的程序，通过分析程序的内部结构，根据其内部结构设计测试用例。理想的白盒测试应该使选取的测试用例覆盖所有的路径，然而，这是不可能的，而且白盒测试不关注测试程序的外部功能。黑盒测试又称为功能测试，在测试过程中被测程序视为黑盒，测试者在完全不考虑程序内部结构和内部特征(或对上述信息无从获知)的情况下，根据需求规格说明书设计测试用例，推断测试结果是否正确。黑盒测试的不足在于测试用例选择时，只考虑程序的输入，以及在该情况下的输出，并没有考虑程序的内部结构。因此，程序内部结构是否规范、结构化程度的好坏、系统的性能如何等都无法测试。

白盒和黑盒测试各有其自身的特点，但也都存在明显的不足，主要表现在只考虑程序某一方面的属性和特征，没有综合考虑。为进行较全面的程序测试，必须把测试工作分两次进行，分别用白盒和黑盒各测试一次。这样不但浪费时间，而且测试的效果不一定好。灰盒正是基于这一点提出的。

灰盒测试是介于白盒和黑盒之间的一种综合测试法，灰盒测试以程序的主要性能和主要功能为测试依据，测试方法主要根据程序的程序图、功能说明书，以及测试者的实践经验来设计。这里所说的主要性能和主要功能凭借测试者的经验来确定，即可以把那些不容易发生错误的变量输入和流程图中的不影响或不改变内部逻辑的细节忽略。事实上，许多测试工作是在不完全了解程序内部逻辑的情况下进行的，这也就是“灰盒”的由来。

同时，灰盒测试涉及输入和输出，但通常使用关于代码和程序操作等在测试人

员视野之外的信息设计测试。在现在的测试工程中，最常见的灰盒测试是集成测试。但是灰盒测试已经由原来单一的白盒测试和黑盒测试的一些测试方法的简单叠加，衍生出许多新颖的分析方法。与白盒测试和黑盒测试相比，灰盒测试具有以下特性：

① 灰盒测试通常是在集成测试前期进行的。灰盒测试通常在程序员完成白盒测试之后，在功能测试人员进行大规模集成测试之前进行的。

② 灰盒测试需要了解代码工程的实现方法。

③ 灰盒测试是通过类似于白盒测试的方法进行的，是通过编写代码，调用函数或者封装好的接口进行的。

④ 灰盒测试是由测试人员进行的。

(4) ALAC(Act-like-a-customer)测试是一种基于客户使用产品的知识开发出来的测试方法。ALAC 测试是基于复杂的软件产品有许多错误的原则，最大的受益者是用户，缺陷查找和改正那些客户最容易遇到的错误。

它的出发点也是著名的 Pareto 80-20 规律。一个软件产品或系统中全部功能的20%是常用的功能，用户的 80%时间都在使用这 20%功能。测试发现的所有错误中的 80%很可能集中在 20%的程序模块中。由于测试的时间有限，像客户那样做，对常用的功能进行测试。ALAC 测试方法适合一些特别的场合，如产品只是一个演示版，开发预算很低，没有足够时间进行测试，可降低测试成本，缩短测试时间。

1.1.4　软件测试模型及演变

所谓测试模型(Test Model)，是测试和测试对象的基本特征、基本关系的抽象。它是测试理论家们根据大量的实际测试应用总结出来的，能够代表某一类应用的内在规律，并对应于适合此类应用的一组测试框架。目前，主流的软件生命周期模型或软件开发过程模型有：瀑布模型、快速原型模型、螺旋模型、增量模型、渐进模型、快速软件开发，以及 Rational 统一过程等，这些模型对于软件开发过程具有很好的指导作用，但是在这些过程方法中，软件测试的地位和价值并没有体现出来，也没有给软件测试足够的重视，利用这些模型无法更好地指导测试实践。软件测试是与软件开发紧密相关的一系列有计划、系统的活动，显然软件测试也需要测试模型指导实践。

1) V 模型

软件测试模型与软件测试标准的研究随着软件工程的发展而越来越深入。在 20 世纪 80 年代后期，Paul Rook 提出了著名的软件测试 V 模型，旨在改进软件开发的效率和效果。V 模型反映测试活动与分析设计活动的关系。图 1-1 从左到右描述基本的开发过程和测试行为，非常明确地标注了测试过程中存在的不同类型的测试，并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系。

V 模型图中箭头代表时间方向，左边下降的是开发过程阶段，与此相对应的是右边上升的部分，即测试过程的各个阶段。