第7章 软件测试的资源分配、进度管理与最优发行

BACK

7.1 软件测试与可靠性增长

作为本章的基础，本节介绍软件测试的基本流程及其相关测试方法，软件测试的人力投入数量描述和软件可靠性增长模型。

7.1.1 软件测试概述

软件测试的目标是希望以最少的人力费用和时间发现潜在的各种差错和缺陷，以期进行改正。为此需要一定的测试方法、测试策略和测试流程。

1. 软件测试方法和测试流程

软件的测试方法有很多，不同的测试方法往往针对不同的测试目标和测试对象、不同的思路以及采用不同的手段。例如，按照被测对象的不同，可分为面向功能、结构为主的测试和面向对象的测试。而面向功能、结构为主的测试又依据是否在计算机上执行被测软件这一准则来划分，可分为静态测试与动态测试。动态测试中又可以是否关心被测软件内部程序细节为依据，划分为黑盒测试和白盒测试。此外，面向功能、结构的测试亦可依阶段目标的不同，划分为单元测试、集成测试、系统测试和运行测试，而面向对象的测试可分为方法测试、类测试、类簇测试、系统测试和运行测试；还可以软件系统目标为依据来划分，分为功能性测试、性能测试、可靠性测试、安全性测试、强度测试、恢复性测试(可看成可修性的一个方面)。

NIS软件的测试过程通常包括拟定测试计划和编制测试大纲，设计和生成测试用例，按序完成单元测试、集成测试、系统测试和运行测试，生成相应的测试报告等基本活动，其测试流程见图7.1。需要说明的是，系统测试是需在相关硬件(计算机硬件与网络设备)配置好的情况下所进行的软/硬件系统联试，经系统测试通过后即可交付用户运行，而运行测试则是在用户的作用下为提高软件可靠性所做的相关测试。此外，为使软件测试能省时高效，应采用测试与开发同步进行和逐步推进的渐近策略，并将测试贯穿于软件的整个生命周期的始终。

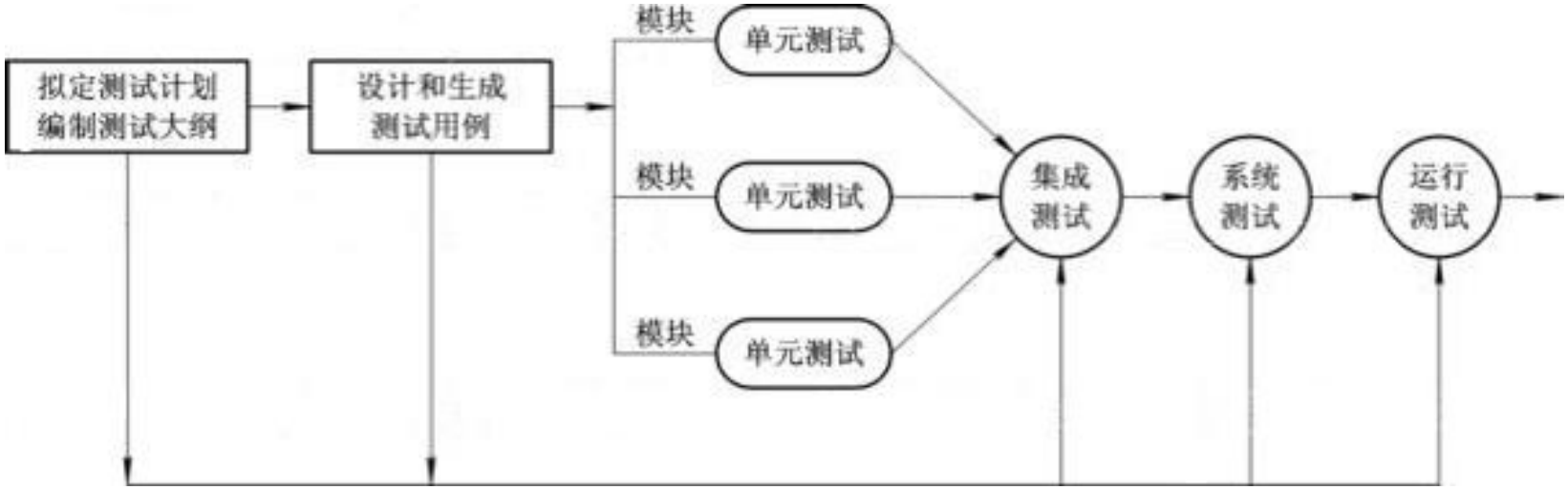
拟定测试计划
编制测试大纲
设计和生成
测试用例
模块
单元测试
模块
单元测试
模块
单元测试
集成
测试
系统
测试
运行
测试

图7.1　软件测试流程图

2. 静态测试

静态测试是通过阅读程序来查找软件的差错与问题的一种方法，其检查的重点为代码与设计要求是否一致，代码的逻辑表示是否正确与完整，代码结构是否合理，是否有未定义或用错的局部变量或全局变量，是否有不适当的循环嵌套和分支嵌套，是否有潜在的死循环等。

3. 黑盒测试

黑盒测试又称功能测试或数据驱动测试。它将软件视为一个看不到内部状况的黑盒子，在完全不考虑软件内部程序结构和特性情况下考察软件的外部功能与性能特性。由于黑盒测试不能进入程序内部，因而只能作用于程序的接口处，并通过一些典型数据的输入/输出发现被测软件是否有不正确或遗漏的功能，在接口上，输入信息是否能被正确地接收，能否输出正确的结果，是否有数据结构错误或外部信息(例如数据文件)访问错误，是否有初始化或终止性错误，性能上是否满足设计要求等。

4. 白盒测试

白盒测试又称为结构测试或逻辑驱动测试，它将软件视为一个内部结构透明的白盒子，因而被测软件内部程序的逻辑结构和其他有关信息已被测试人员所了解，于是测试人员可根据所了解的上述信息，根据覆盖准则来设计或选择测试用例，对程序中的每个语句、每个条件分支、每个控制路径进行模块测试，以确定实际状态与设计是否一致。白盒测试通常要求对程序中的所有独立路径至少执行一次；在所有的逻辑判断中，取“真”和取“假”的两种情况至少都能执行一次；每个循环都应在边界条件下和一般条件下各执行一次，以此来考察内部数据结构的有效性。

5. 单元测试

由于软件开发是一个由单元(模块)到整体(系统)的过程，因此软件测试的首次活动应为单元测试，以确定每个单元能否正常工作。单元测试除进行功能测试外，主要测试单元与单元间的接口、局部数据结构、重要执行路径、故障处理通路四项特征以及各项特征的边界条件。单元测试一般采用白盒测试，在通过对单元的结构分析后设计出一些测试用例来考察上述各类特征，但也可在白盒测试基础上再采用黑盒测试的思想，对原有的测试用例加以补充，并继续进行考察。

由于模块本身无法独立运行，且模块与模块之间存在着有机的联系，如调用与被调用的关系、数据传递关系等，因此在对每个模块进行测试时需要开发称为驱动模块(driver)和桩模块(stub)的两种辅助模块，用以模拟模块间的调用关系和数据传递关系。其中驱动模块相当于一个主程序，用来接收测试用例的数据，并将这些数据送到被测模块，同时输出测试结果；桩模块也称为存根模块，它被用来代替被测模块中需要调用的其他模块，但内部结构要简单得多，桩模块内部可进行少量的数据处理，目的是为了检验入口和返回、输出调用等被测模块与其下级模块的接口关系。图7.2给出了单元测试的示意图。

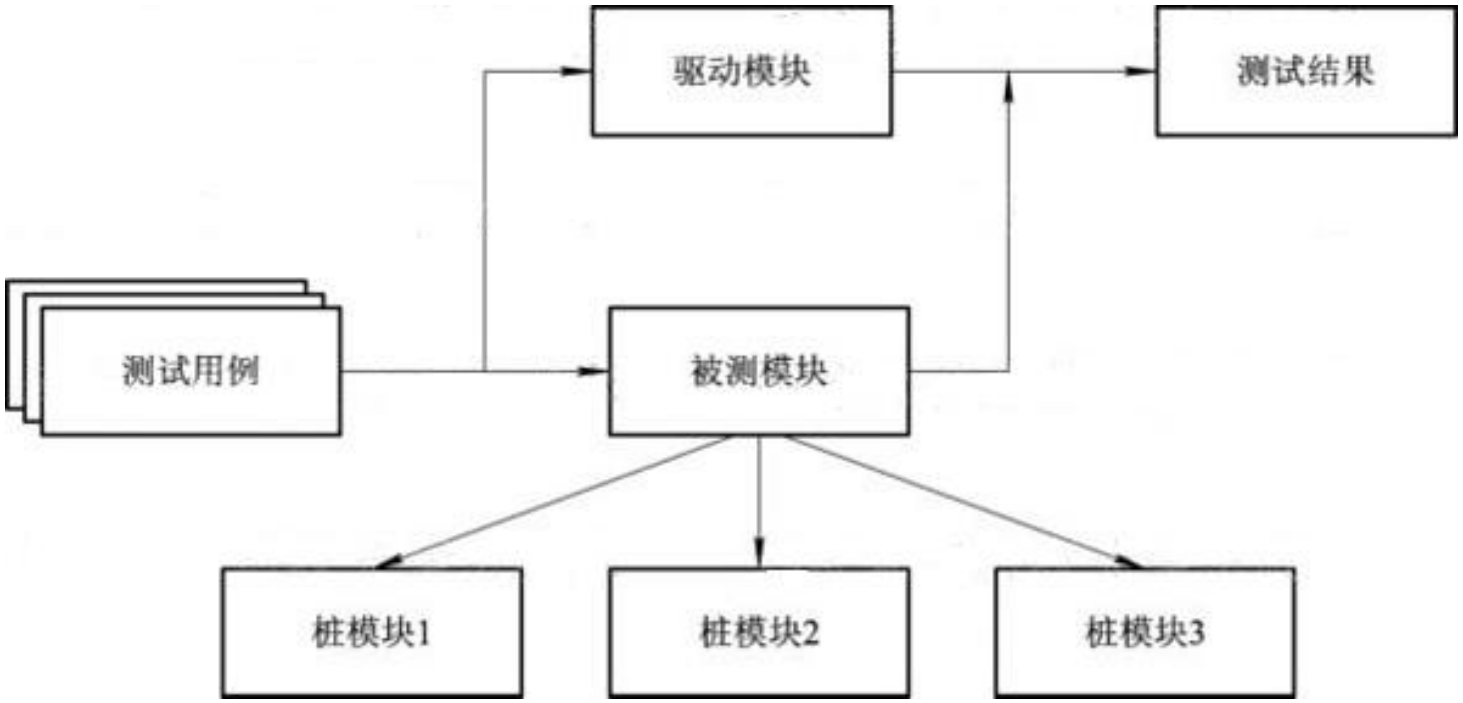

图7.2　单元测试示意图

6. 集成测试

集成测试是在对被测软件所有单元(模块)分别独立测试完后，按照系统设计的模块结构进行逐步组装的一种有序测试。其原因在于尽管各单元单独工作得很好，但并不能保证它们组装后就能正常地工作。例如数据可能在通过接口时丢失；一个模块可能会对另一模块产生无法预料的副作用；当一些子函数被连接在一起时，可能无法实现设计中所预期的功能；在单个模块中可以接受的不精确性在各模块联通后可能会变得无法接受，合理的局部数据结构组装后所构成的全局数据结构可能也会存在问题……因此集成测试具有组装和检验的双重意义：一方面在软件完成集成测试的同时，逐步将各个模块组装起来，形成一个完整的运行系统；另一方面又检验了每一个组装步骤是否正确，即检验每加入一个模块是否能使功能和性能可以得到增长，以及能否与已存在的模块正确结合、协调工作。

集成测试包括功能集成测试、操作剖面建立和有效性测试三部分，其中功能测试通常采用非增量式集成方法或增量式集成方法。非增量式集成方法是首先分别测试各个模块，然后再把这些已被测试并确认为功能与性能符合设计要求的模块组合起来进行整体测试；增量式集成测试方法则是采用测试一个模块组装一个模块，然后再测试再组装，直到所有模块均被组装完毕，并被整体测试合格为止的一种逐步组装的方式。显然，非增量式集成测试可以对所有模块并行进行单元测试，能充分利用人力，加快工程进度；但这种一步到位的方法容易形成混乱，出现错误后不容易查找和定位，故一般适用于规模较小的软件。增量式集成测试虽然采用逐步到位的方法，要多费人力和工时，但由于每个已被测试过的模块还可以在以后组装过程中的每一步骤(组装一个新模块)进行新的测试，从而使得程序测试更为彻底。因而从测试有效性角度来看，增量式集成测试将比非增量式集成测试更为有效。在增量式集成测试中，要求每增加一个新的单元，必须验证新加入的单元和原已被测试通过单元之间接口的正确性，并且必须在经组装(一个单元)后所得到的构件的新层次上再次进行扩充后的单元测试。集成测试通过的准则为：

(1) 达到规定的测试覆盖类和覆盖率要求；

(2) 对测试中的异常要有合理的解释；

(3) 各模块无错误链接；

(4) 满足各项功能增长和性能增长的要求；

(5) 对错误输入有正确的处理能力。

(6) 在组合模块进行测试时，为了避免引入新的软件差错和产生新的问题，往往采用回归测试法。

自顶向下的增量集成方式能较早地验证控制和判断点，出现问题能及时修正，这种早期的功能性验证有利于增加开发人员和客户的信心，且由于在测试时不需要重新编写驱动模块，故测试效率较高。但当出现高层模块对下层模块依赖性很大时，由于需要返回大量信息，故在用桩模块替代时，桩模块的编写就较为复杂，从而会增加开销，因而此时可采用自底向上的增量集成方式。有关自顶向下增量集成测试之流程图详见图7.3。

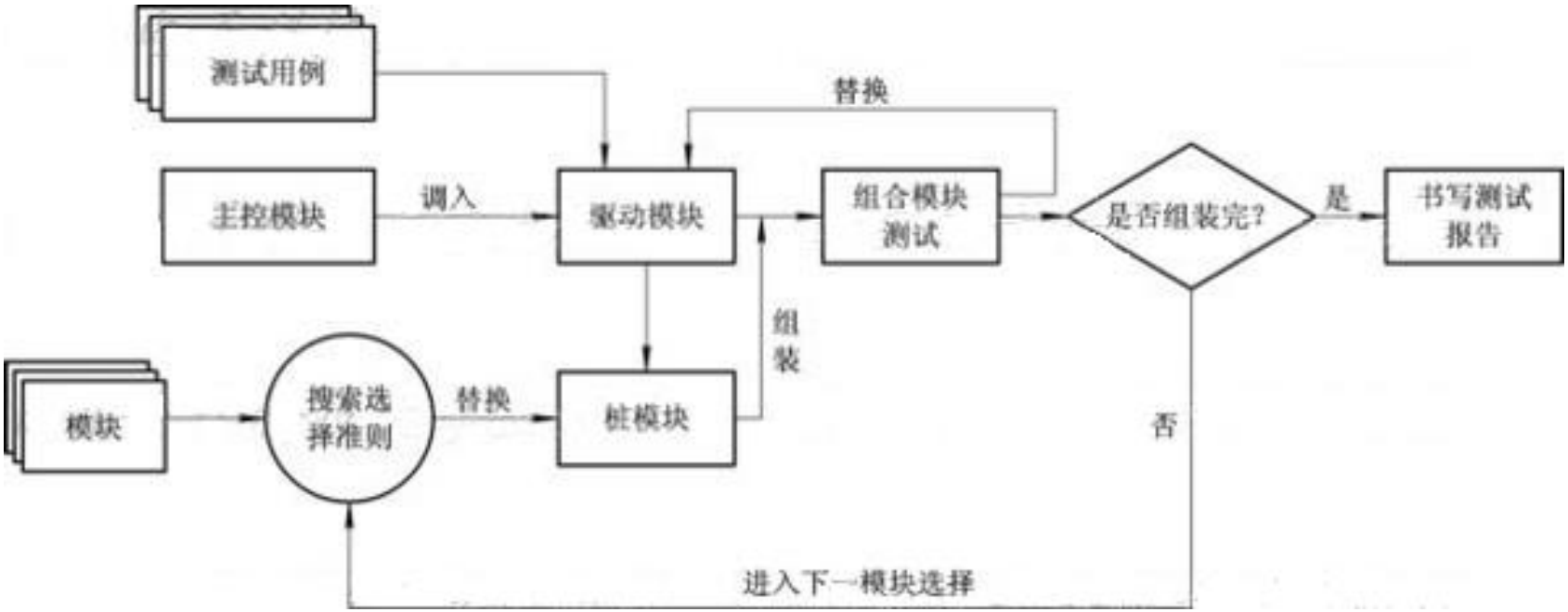

图7.3 自顶向下增至集成测试流程图

当采用各种增量集成方式作集成测试时常运用回归测试方法。所谓回归测试，是对某些已经测试过的程序子集进行部分或全部的重复测试的一种测试方法。由于在采用各种集成测试方法时，每当一个新的单元模块加入到组合模块中，就会给原有的组合模块带来一些新的变化，例如会出现一些新的数据流路径，产生一些新的I/O操作，还可能激活一些新的控制逻辑，因此要对某些已被测试过的程序子集做回归测试，以检查由于上述的变化是否会引入新问题或带来无法预料的副作用。此外，在集成测试的过程中，如果发现错误则均须进行排错；每当软件在排错时，通常会修改某些程序子集或某些数据、文档，而回归测试的实施也可用来保证软件在进行排错时不至于带来新的差错或造成新的不可预料的后果。

回归测试可以通过重新执行全部或部分测试用例人工实现，也可以使用自动化的捕获回放工具来进行。考虑到在集成测试过程中需要实施回归测试的数量可能会很庞大，因此在回归测试的选择上应优先考虑那些涉及在主要的软件功能中出现过一个或多个错误类时的测试，如果在每做一次改错时均实施一次回归测试，则显然是不切实际的也是低效率的。

集成测试的第二个重要部分是建立操作剖面。所谓操作剖面(Operation Profiles)，是对软件用户运行环境与系统使用方式的一种概念描述，它被定义为一组功能方案的概率分布。显然，为在开发方地点来模拟用户的运行环境与实际操作方式，建立操作剖面是必要的。操作剖面一般可以根据软件的以前版本或现有类似系统的实际应用过程中的操作状况来建立。建立操作剖面的过程实际上是一个逐步细化软件运行条件和使用范围的过程，通常将其划分为四个主要阶段，即分析顾客剖面或用户剖面、确定系统方式剖面、定义功能剖面和构造操作剖面。其中分析顾客剖面或用户剖面、确定系统方式剖面、定义功能剖面是在软件开发的系统分析阶段之前建立的，有时可以延伸到详细设计阶段，它们描述了运行条件和用户使用条件的基本特征，是建造操作剖面的基础。而构造操作剖面是在软件实现和软件测试阶段建立的，它描述了各种类型的运行类型出现的概率。有关操作剖面建立的进一步内容可参见作者文献[15]。

集成测试的第三个重要部分是有效性测试。由于软件经组装测试并排错后，接口方面的问题已经解决，故以后集成测试的主要问题是解决软件的有效性问题，所谓软件的有效性问题，是指软件的功能、性能、可靠性、安全性及保障性等方面软件的实际水平是否达到用户的需求。有效性测试是在开发方地点在模拟用户运行环境的条件下所进行的一种用户需求测试，一般采用黑盒测试来检验所开发并经单元测验、组装集成测试及排错后的软件是否与描述用户需求的需求分析说明书相一致。测试人员一般由开发方的测试人员及软件设计人员组成。以下简述各类测试的基本内涵。

1) 恢复测试

恢复测试是检测系统的容错能力。检测方法是采用各种方法让系统出现故障后，检验系统是否按照要求能从故障中恢复过来，并在预定的时间内开始事务处理，而且不对系统造成任何损害。如果系统的恢复是自动的(由系统自动完成)，需要检验重新初始化、检查点、数据恢复等是否正确。如果恢复需要人工干预，就要对恢复的平均时间进行评估，并判断它是否在允许的范围内。由于系统的容错能力常作为系统可靠性的表征之一，故恢复测试亦可作为可靠性测试中的一个组成部分。

2) 安全性测试

系统的安全性测试是检测系统的安全防范机制、保密措施是否完美且没有漏洞。测试的方法是测试人员模拟非法入侵者，采用各种方法冲破防线。例如，以系统的输入作为突破口，利用输入的容错性进行正面攻击；故意使系统出错，利用系统恢复的过程，窃取口令或其他有用的信息；想方设法截取或破译口令；利用浏览非保密数据，获得所需信息等。从理论上说，只要时间和资源允许，没有进不了的系统。所以，系统安全性设计准则是使非法入侵者所花费的代价比进入系统后所得到的好处要大，此时非法入侵已无利可图。

3) 强度测试

强度测试是对系统在异常情况下的承受能力的测试，是检查系统在极限状态下运行，性能下降的幅度是否在允许的范围内。因此，强度测试要求系统在非正常数量、频率或容量的情况下运行。例如，运行使系统处理超过设计能力的最大允许值的测试用例；设计测试用例，使系统传输超过设计最大能力的数据，包括内存的写入和读出、外部设备等；对磁盘保留的数据，设计产生过度搜索的测试用例等。强度测试主要是为了发现在有效的输入数据中可能引起不稳定或不正确的数据组合。

4) 性能测试

性能测试是检查系统是否满足系统分析说明书对性能的要求。特别是实时系统或嵌入式系统，即使软件的功能满足需求，但性能达不到要求也是不行的。性能测试覆盖了软件测试的各阶段，而不是等到系统的各部分所有都组装之后，才确定系统的真正性能。通常与强度测试结合起来进行，并同时对软、硬件进行测试。软件方面主要从响应时间、处理速度、吞吐量、处理精度等方面来检测。

5) 可靠性测试

对于在系统分析说明书中提出了可靠性要求的软件或对一些要求高可靠性的软件，可靠性测试是必要的。由于衡量软件保持应有功能与性能持久能力的可靠性要求对于任何的网络管理软件和网络应用软件来说均是需要的，因此应该通过软件的可靠性测试来判断被测软件是否满足用户的可靠性要求。这种对软件可靠性目标的验证测试，在软件集成中的有效性测试和软件验收中的系统测试均需要实施。由于软件集成的有效性测试是交由开发方实施的，因此已发现软件差错就将进行排错，从而使软件呈现可靠性增长的趋势，故这种现象又称为软件可靠性增长。然而在验收阶段的可靠性测试则是一种依据软件验收规范所做的测试，以期对软件是否验收做出决策，故在这一过程中一般暂不进行软件排错，而留待测试后再考虑排错问题，因此这种测试是一种影响软件可靠性水平的测试。有关软件可靠性增长测试的基本流程见图7.4，相关内容可参见作者文献[15]。

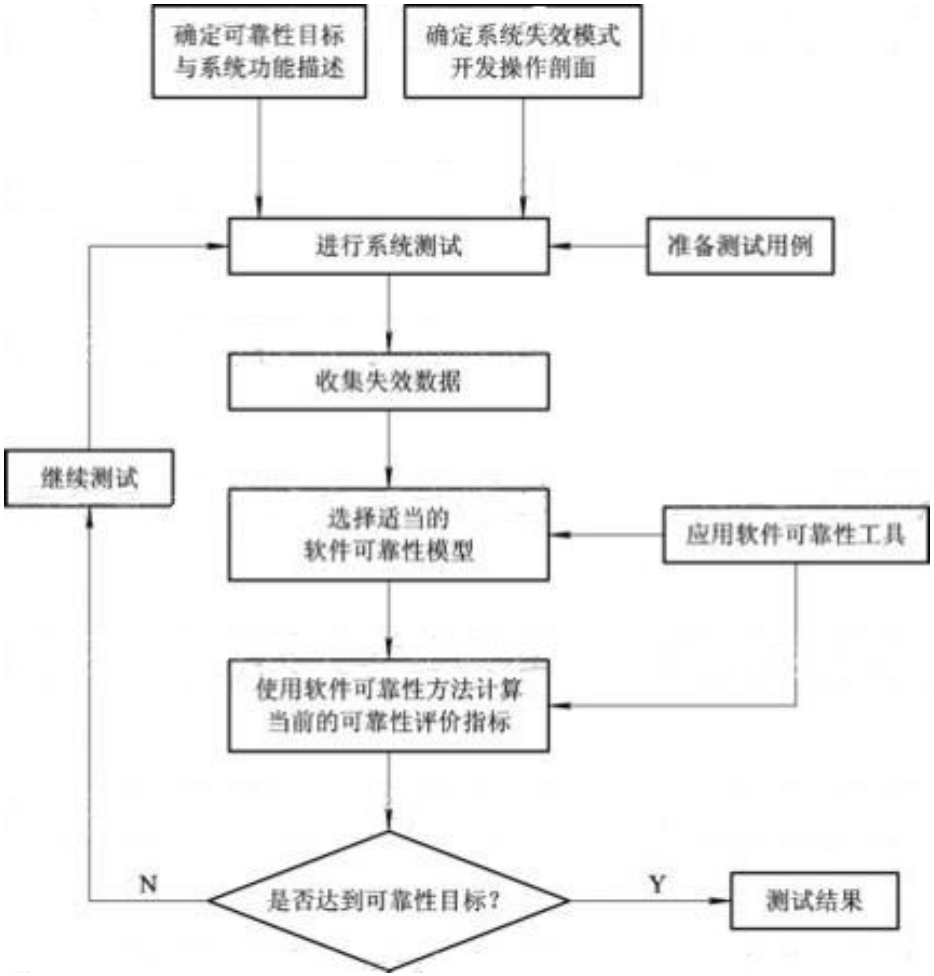
确定可靠性目标
与系统功能描述
确定系统失效模式
开发操作剖面
进行系统测试
准备测试用例
收集失效数据
继续测试
选择适当的
软件可靠性模型
应用软件可靠性工具
使用软件可靠性方法计算
当前的可靠性评价指标
N
是否达到可靠性目标?
Y
测试结果

图7.4 测试过程流程图

集成测试的特点是边查错、边排除。差错的排除不能采用试探的方法，因为这种做法很难达到预期的排错要求，也是排错人员无能的表现。

由于排错通常要局部修改原来的程序，并破坏原有程序的完整性，从而对程序的结构和可读性产生消极的影响，而且排错又是在开发进度约束下进行的，时间的紧迫性又使问题变得更加尖锐，因而新的错误很容易在毫无觉察的情况下引入。经验表明，在改正错误的过程中引入新的错误的概率高达20%~50%。这种情况表明排错工作不能分配给缺乏经验的新手，而应由有经验的程序员去担任。必须确保排错的质量，对排错的要求应像对设计的要求一样严格。而且在排错过程中若更改与排错有关的文档，则改正后的程序要经过代码检查和走查。如果错误是在单元测试后发现的，应追加一些单元测试用例或重新进行单元测试。要防止速成的做法，即先改动目标程序，事后再修改源程序，这种做法容易造成目标程序和源程序不一致；正确的做法是先改正源程序，然后进行编译，这种做法表面上看来似乎费时，但是出错的可能性较小。常用的排错技术包括诊断输出语句、抽点打印分析、跟踪分析、指令断点分析、断言控制排错和执行历史分析六种，详情在此从略。

7. 验收测试与运行测试

软件验收测试是检验被测软件所具有的功能和性能水平是否满足用户需求的一种验证测试，经过软件验收测试并通过后，软件作为产品将交付用户使用。考虑到软件如网络管理软件和网络应用软件等均有一定的规模和较高的复杂性，即使是在单元测试、集成测试与系统测试后，仍然会残留一些差错，因此在软件交付用户使用后仍须进行经常性测验以保证软件的可靠性增长，这就是软件的运行测验。表7.1列出了将软件验收测试和运行测验与软件单元测试和集成测试比较，在测试地点与运行环境、测试参与人员、测试关注重点、排错及测试基本内容等方面的不同之处。需要说明的是，软件测试的基本内容除有效性(性能、可靠性、安全性、恢复性、强度)测试外，尚须进行软件配置的检查如检查软件(源程序、目标程序)和文档(包括面向开发方和用户方两方面)是否齐全以及分类是否有序等，以确保文档、资料的正确和完善，以便于软件维护；软件运行测试则主要是软件可靠性增长测试，以利于软件在今后的工作中发挥更大的作用。

表 7.1 四种软件测试的比较

序号	测试类别	测试地点与运行环境	测试参与人员	测试关注重点	排错	测试基本内容
(1)	单元测试	开发方，各单元独立测试	开发人员自身测试	编码方面的差错	查出差错，立即排除	功能性测试
(2)	集成测试	开发方，模拟用户环境	开发人员，系统分析人员	单元间接口方面的差错及系统设计差错	查出差错，主动排除	集成增量(组装)测试，建立操作剖面，有效性(性能、可靠性、安全性、恢复性、强度)测试
(3)	验收测试	用户方，用户实际运作环境	第三方测试人员，用户测试人员	需求分析方面的差错与问题	查出差错，暂不排除	有效性(性能、可靠性、安全性、恢复性、强度)测试，软件配置检查
(4)	运行测试	用户方，用户实际运行环境	用户，测试人员	各方面差错	查出差错，立即排除	可靠性增长测试

7.1.2 软件可靠性增长模型

1. 软件测试、差错排除与可靠性增长

由于在集成测试与运行测试(包括验收测试后的差错排除)中差错的检出和排除，软件系统的残存差错数将不断减少，并呈现如图7.5所示的阶梯型下降曲线，从而实现了软件系统的可靠性增长现象。由于这样的可靠性增长现象是由开发机构所投入的开发人员数量与业务素质(测试经验、测试技术)及测试工具所决定的，因而必然会呈现出一定的数量规律性。于是，人们就用随机过程这一数学工具来描述测试过程中故障(或差错)减少与可靠性增长过程的概率规律性，并在此基础上提出了各种反映软件可靠性度量指标之间及其与人力投入依赖关系的可靠性增长模型。利用各种可靠性增长模型，人们可以对正在开发或已开发完成的软件可靠性属性作出定量评估，预计开发过程的可靠性增长状况，判断已开发完成的软件是否符合验收标准，进而为软件开发过程中的人力资源组织、分配与控制以及软件发行等问题的决策提供依据。

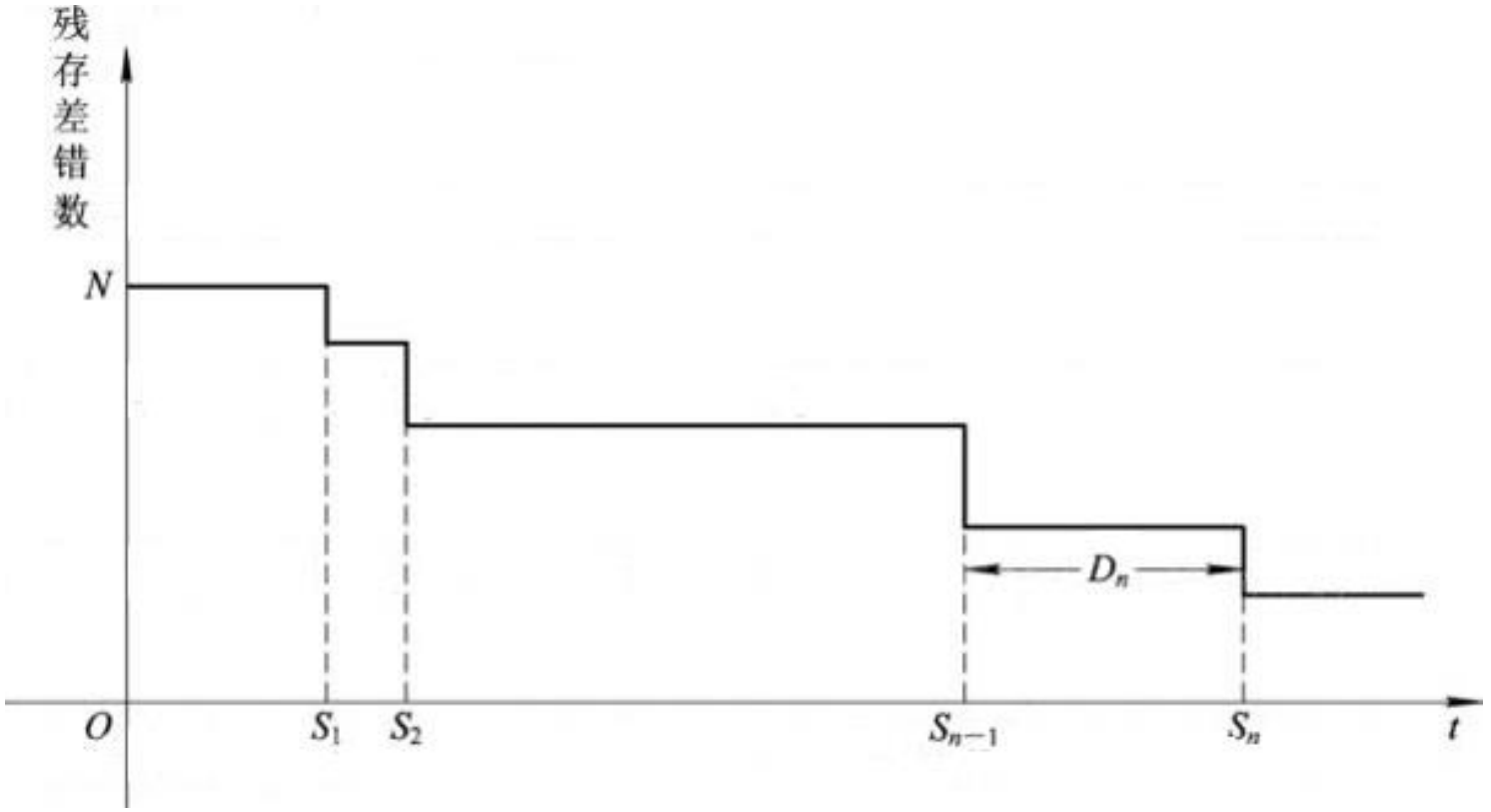

图7.5 差错排除过程

目前已发表的可靠性增长模型数量很多，并在其模型假设条件的合理性、指标的系统性与关联性、参数估计的复杂性、对其他软件工程经济问题决策支持的深度以及数据要求等方面各有所长。限于篇幅，本节仅介绍与软件工程经济学相关较为紧密的三个模型：G-O 模型，SPQL模型和基于测试人力的可靠性增长模型，需要进一步了解其他可靠性有关内容的读者可参阅作者撰写的文献[15]、[16]。

G-O模型是由A.L.Goel和K.Okumoto于1979年提出的。该模型的基本出发点是对软件的量测与排错将无限制地延续下去，并认为差错的查出与排错累计过程是一个非时齐的泊松过程(NHPP)。该模型对随后的软件可靠性测试评价模型研究也有着重大的影响。

G-O模型族包括有基本G-O模型、扩展G-O模型、延迟S型G-O模型和普通S型G-O模型等多种变种，以下仅介绍前两种。

2. 基本G-O模型

基本G-O模型的基本假设为：① 差错随机地存在于程序中，在对软件的量测与排错过程中，差错的出现是程序运行的函数，在任何时间区间内出现的期望差错数与时间区间的长度Δt成正比，与剩余差错数成正比，比例系数设为b；② 在量测与排错过程中，差错的累计(计数)过程是一个非时齐的泊松过程；③ 纠错时不发生新的错误；④ 前后出现的差错无关联。

设$N(t)$表示$(0，t)$内查出的累计差错数，则由上述基本假设②知$N_T=\{N(t)，t\geq 0\}$为NHPP；若设$\lambda(t)$为N_T的强度函数，$m(t)=\int_0^t \lambda(u)\mathrm{d}u$ 为累计强度函数，则由非齐次泊松过程理论可以证明，对 $\forall t\geq 0$ 有$E[N(t)]=m(t)$，从而表明$m(t)$即为在$(0，t)$内查出的期望累计差错数。此外，又由于 $\frac{\mathrm{d}m(t)}{\mathrm{d}t}=\lambda(t)$ ，故$\lambda(t)$可理解为在 t 时的差错查出率(单位时间内查出的平均差错数)。另外，注意到差错与排错将无限制地延续下去，故$N(t，\infty)=N(\infty)-N(t)$表示$t$时的剩余差错数，$a=\lim_{t\to\infty} m(t)$ 表示最终查出的期望差错数。利用非齐次泊松过程理论可以证明如下结论。

定理7.1 在上述基本假设①、②、③、④条件下，软件的差错查出与排错计数过程$\{N(t)，t\geq 0\}$(NHPP)有：

(1) 对$\forall\ t\geq 0$，$(0，t)$内查出的期望累计差错数$m(t)$及 t 时的差错查出率 $\lambda(t)$ 有

$$m(t)=a(1-\mathrm{e}^{-bt}),\lambda(t)=ab\mathrm{e}^{-bt} \qquad a>0,b>0 \tag{7.1}$$

(2) 当软件的量测与排错过程无限制延续下去时，其最终查出的累计差错数具有均值为a的泊松分布，即有

$$\lim_{t\to\infty} p(N(t)=k)=\frac{a^k}{k!}\mathrm{e}^{-a} \qquad k=0,1,2,\cdots \tag{7.2}$$

t 时的剩余差错数的期望与方差为

$$E[N(t,\infty)]=\mathrm{var}(N(t,\infty))=a\,\mathrm{e}^{-bt} \tag{7.3}$$

(3) 在 t 时已查出 d 个差错的条件下，t 时的剩余差错数有如下的条件分布与条件数学期望:

$$P(N(t,\infty)=k \mid N(t)=d)=\frac{(a-m(t))^k}{k!}\mathrm{e}^{-(a-m(t))}=\frac{a^k\,\mathrm{e}^{-bkt}}{k!}\mathrm{e}^{-(a\mathrm{e}^{-bt})} \quad k=0，1，2，\cdots$$

$$E(N(t,\infty)\mid N(t)=d)=a-m(t)=a\,\mathrm{e}^{-bt}$$

(4) 设S_j表示第j个差错的出现时刻，$X_j=S_j-S_{j-1}$为相邻差错出现的间隔时间，$j=1，2，\ldots$，则在 $S_{n-1}=T$ 的条件下，$X_n>t$ 的条件概率为

$$R(t|T)=P(X_n>t|S_{n-1}=T)=\exp\{-m(t+T)+m(T)\}$$

$$=\exp\{-m(t)\mathrm{e}^{-bT}\}=\exp\{-a(1-\mathrm{e}^{-bt})\mathrm{e}^{-bT}\} \qquad (7.4)$$

此$R(t|T)$即为软件可靠度。

在将上述定理的有关结论应用时，需对a与b作出估计，下面用最大似然估计法原理来对a、b作参数估计。设在量测与排错过程中，已得到了在时间区间$\Delta_k=(0，t_k)$查出的累计软件差错数d_k，事实上当t_k即为样本值时显然有$d_k=k$，$k=1，2，\ldots，n$，其中$0<t_1<t_2<\ldots<t_n$，则可建立如下关于参数a与b的似然函数：

$$L(t_1，t_2，\cdots，t_n；d_1，d_2，\cdots，d_n；a，b)$$

$$=P(N(t_1)=d_1，N(t_2)=d_2，\cdots，N(t_n)=d_n)$$

$$=P(N(t_1)=d，N(t_1，t_2)=d_2-d_1，\cdots，N(t_{n-1}，t_n)=d_n-d_{n-1})$$

将上述似然函数$L(t_1, t_2, \cdots, t_n; d_1, d_2, \cdots, d_n; a, b)$分别对未知参量$a$与$b$求偏导并令其等于0。然后通过非齐次泊松过程(NHPP)的过程统计推断理论(可详见作者文献[16])可以推得如下方程组成立：

$$\begin{cases} \dfrac{\partial \ln L}{\partial a} = \dfrac{1}{a}\sum_{j=1}^{n}(d_j - d_{j-1}) - (1-\mathrm{e}^{-bt_n}) = \dfrac{d_n}{a} - (1-\mathrm{e}^{-bt_n}) = 0 & (7.5) \\ \dfrac{\partial \ln L}{\partial b} = \sum_{j=1}^{n}\dfrac{(d_j - d_{j-1})(t_j \mathrm{e}^{-bt_j} - t_{j-1}\mathrm{e}^{-bt_{j-1}})}{\mathrm{e}^{-bt_{j-1}} - \mathrm{e}^{-bt_j}} - ab\mathrm{e}^{-bt_n} = 0 & (7.6) \end{cases}$$

由式(7.5)可得

$$a = \frac{d_n}{1-\mathrm{e}^{-bt_n}} \qquad (7.7)$$

将式(7.7)代入式(7.6)，可得非线性方程

$$\sum_{j=1}^{n}\frac{(d_j-d_{j-1})(t_j\,\mathrm{e}^{-bt_j}-t_{j-1}\,\mathrm{e}^{-bt_{j-1}})}{\mathrm{e}^{-bt_{j-1}}-\mathrm{e}^{-bt_j}}=\frac{bd_n\,\mathrm{e}^{-bt_n}}{1-\mathrm{e}^{-bt_n}} \tag{7.8}$$

上式为仅含未知参数b的非线性代数方程，求解式(7.8)可得 $\hat{b}$ ，代入式(7.7)可得 $\hat{a}$ 。

3. 扩展G-O模型

在基本G-O模型的假设中规定软件差错一旦被发现将立即被修正，并不会引入新的差错，而扩展G-O模型则将上述假设进一步放宽，即假设⑤软件在t时刻发现的差错并非一定会修正(或排除)，并设 p 表示在 t 时刻被发现的软件差错完全修正的概率。对这种推广的G-O模型有定理7.2结论。

定理7.2 在上述基本假设①、②、③、④、⑤下，软件的查错与排错过程$\{N(t)，t\geq 0\}$(NHPP)有：

(1) 对 $\forall\, t\geq 0$，$(0，t)$内查出的期望累计差错数 $m(t)$ 及 t 时的差错查出率 $\lambda(t)$ 有

$$m(t)=\frac{a}{p}(1-\mathrm{e}^{-bpt}),\quad \lambda(t)=ab\mathrm{e}^{-bpt} \tag{7.9}$$

(2) 当量测与排错过程无限制地延续下去时，其最终查出的累计差错数具有均值为a/p 的泊松分布，即有

$$\lim_{t\to\infty} P(N(t)=k)=\lim_{t\to\infty}\frac{\left[\frac{a}{p}(1-\mathrm{e}^{-bpt})\right]^k}{k!}\mathrm{e}^{-\frac{a}{p}\mathrm{e}^{-bpt}}=\frac{\left(\frac{a}{p}\right)^k}{k!}\mathrm{e}^{-\frac{a}{p}} \quad k=0,1,2,\cdots \tag{7.10}$$

T 时的剩余差错数的期望与方差有

$$E(N(t,\infty))=\mathrm{var}(N(t,\infty))=\frac{a}{p}\mathrm{e}^{-bpt}$$

(3) 在 t 时已查出 d 个差错的条件下，t 时的剩余差错数有如下的条件分布与条件数学期望：

$$P(N(t,\infty)=k\mid N(t)=d)=\frac{\left(\frac{a}{p}\mathrm{e}^{-bpt}\right)^k}{k!}\mathrm{e}^{-\frac{a}{p}\mathrm{e}^{-bpt}} \quad k=0,1,2,\cdots \tag{7.11}$$

$$E(N(t,\infty)\mid N(t)=d)=\frac{a}{p}\mathrm{e}^{-bpt}$$

(4) 软件可靠度为

$$R(t\,|\,T)=P(X_n>t\,|\,S_{n-1}=T)=\exp\{-m(t)\,\mathrm{e}^{-bpt}\}$$

$$=\exp\left\{-\frac{a}{p}(1-\mathrm{e}^{-bpt})\,\mathrm{e}^{-bpT}\right\} \qquad (7.12)$$

该定理的证明与定理7.1类同，故从略。容易验证定理7.1各结论为定理7.2结论当p=1时的特殊情形。

上述结论中的参数a与b可利用与前述类同的方法来计算，即将$P(N(t_1)=d_1，N(t_2)=d_2，\cdots，N(t_n)=d_n)$，$0\leq t_1<t_2<\cdots<t_n$，作为似然函数分别对$a$、$b$求偏导，并令其为零组成联立方程来求解。不同的是$m(t)$由$a(1-e^{-bt})$变为$\frac{a}{p}(1-e^{-bpt})$。下面我们来介绍另一种似然函数的参数估计方法。

在软件量测与改错过程中，设 s_j 表示第 j 个差错的查出时刻(j=1，2，…)，若已得到了S=(s_1，s_2，…，s_n)的观察值(t_1，t_2，…，t_n)，显然0≤t_1<t_2<…<t_n，则由作者文献[16]知S有联合密度函数

$$f_s(t_1,t_2,\cdots,t_n)=\mathrm{e}^{-m(t_n)}\prod_{j=1}^{n}\lambda(t_j)=\mathrm{e}^{-\frac{a}{p}(1-\mathrm{e}^{-bpt_n})}\prod_{j=1}^{n}(ab\mathrm{e}^{-bpt_j}) \quad 0<t_1<t_2<\cdots<t_n$$

根据最大似然估计原理，亦可将上述f_s(t_1，t_2，…，t_n)作为似然函数L，并将ln$L$达最大时对应的$\hat{a}$、$\hat{b}$作为a、b的估计值。因此有

$$\ln L=\ln f_S(t_1,t_2\cdots,t_n)=\sum_{j=1}^{n}[\ln(ab)-bpt_j]-\frac{a}{p}(1-\mathrm{e}^{-bpt_n})$$

并将ln$L$分别求对a、b的偏导，并令其为零，可得如下联立方程:

$$\frac{n}{a}=\frac{1}{p}(1-\mathrm{e}^{-bpt_n}) \tag{7.13}$$

$$\frac{n}{b}=p\sum_{j=1}^{n}t_j+at_n\,\mathrm{e}^{-bpt_n} \tag{7.14}$$

由式(7.13)可得

$$a=\frac{np}{1-\mathrm{e}^{-bpt_n}} \tag{7.15}$$

将(7.15)式代入(7.14)式，可得

$$\frac{n}{b}=p\sum_{j=1}^{n}t_j+\frac{npt_n}{1-\mathrm{e}^{-bpt_n}}\mathrm{e}^{-bpt_n}=p\sum_{j=1}^{n}t_j+\frac{npt_n}{\mathrm{e}^{bpt_n}-1} \tag{7.16}$$

上式为仅含b的非线性代数方程，利用有关数值计算方法，容易求得b的估计值 $\hat{b}$ ，将其代入(7.15)式，即可得a的估计值 $\hat{a}$ 。

[例7.1] 某数据分析系统在软件测试过程中获得了如表7.2所示的差错查出并纠正的时间序列，试运用扩展G-O模型来估计该数据分析系统的期望累计差错函数$m(t)$与软件可靠度$R(t|T)$，并计算为使该软件可靠性达到R_0=0.98时还需要测试与排错的时间及讨论结果的有效性。

表 7.2 差错查出时间表 单位：天

差错序号 j	差错间隔 $t_j - t_{j-1}$	差错出现时刻 t_j	差错序号 j	差错间隔 $t_j - t_{j-1}$	差错出现时刻 t_j
1	9	9	14	9	87
2	12	21	15	4	91
3	11	32	16	1	92
4	4	36	17	3	95
5	7	43	18	3	98
6	2	45	19	6	104
7	5	50	20	1	105
8	8	58	21	11	116
9	5	63	22	33	149
10	7	70	23	7	156
11	1	71	24	91	247
12	6	77	25	2	249
13	1	78	26	1	250

解 (1) 作参数估计。注意到n=26，t_1~t_{26}见表7.2，修正概率 p 在表7.3中分别取 $p_i = \frac{i}{10}(i = 1,2,\cdots,10)$ ，将这些数据代入(7.16)式求解，可解得 $\hat{b}_i$ 及其对应 $\hat{a}_i$ ，i=1，2，…，10，并列于表7.3中。

表 7.3 参 数 估 计 表

序号 i	p_i	$\hat{a}_i$	$\hat{b}_i$
1	0.1	3.399 351	0.057 090
2	0.2	6.798 702	0.028 950
3	0.3	10.198 060	0.019 300
4	0.4	13.597 400	0.014 475
5	0.5	16.996 790	0.011 580
6	0.6	20.396 120	0.009 650
7	0.7	23.795 460	0.008 271
8	0.8	27.194 820	0.007 237
9	0.9	30.594 170	0.006 433
10	1.0	33.993 510	0.005 709

(2) 求解期望累计差错函数，若取p=0.1，则有

$$\hat{m}(t)=\frac{\hat{a}}{\hat{p}}(1-\mathrm{e}^{-\hat{b}\hat{p}t})=33.99(1-\mathrm{e}^{-0.005\,709\,t})$$

(3) 计算软件可靠度。若仍取p=0.1，则有

$$R(t\,|\,T)=\exp\left\{-m(t)^{-bpT}\right\}=\exp\{-33.99(1-\mathrm{e}^{-0.005\,709t})\mathrm{e}^{-0.005\,709T}\}$$

(4) 计算软件可靠度达到R_0时所需的测试与排错时间。注意到此时由(7.12)式应有

$$R(t\,|\,T)=\mathrm{e}^{-m(t)\mathrm{e}^{-bpt}}=R_0$$

对上式变形，并取对数可得

$$1\mathrm{n}\frac{1}{R_0}=m(t)\mathrm{e}^{-bpT}$$

对上式再取对数有

$$1\mathrm{n}\ln\frac{1}{R_0}=\ln m(t)-bpT$$

从而可以解出

$$T=\frac{1}{bp}[1\mathrm{n}m(t)-1\mathrm{n}\ 1\mathrm{n}\frac{1}{R_0}] \qquad (7.17)$$

由(7.12)式知 t 为测试与排错终止时刻T以后的时间增量，故可取t=1及 p_1=0.1，$\hat{a}_1=3.399\ 351$，$\hat{b}_1=0.057\ 090$，R_0=0.98代入(7.17)式，有

$$T_0=\frac{1}{0.005\ 709}[1\mathrm{n}(33.99(1-\mathrm{e}^{-0.005\ 709}))-1\mathrm{n}\left(1\mathrm{n}\frac{1}{0.98}\right)]=393\text{天}$$

上式说明为使软件可靠度达到0.98，共需测试与排错393天。由表7.2知已测试与排错共250天，则还需要的测试与排错时间为393－250=143天。

上述结论是在排错率为p=0.1情况下得到的，如果提高排错率p(如增加测试工具、投入较高水平的测试人员)，则相应的T_0将会减少。

(5) 为评判上述方法特别是$m(t)$的有效性，可采用χ^2检验或Kolmogorov-Simirnov检验法来作假设检验。由式(7.9)知$pm(t)/a$具有参数为bp的负指数分布形式，故可利用表7.2所示的样本数据及上述检验方法作假设检验，有关检验过程从略。图7.6列出了$\hat{m}(t)$ 曲线与表7.2所示 t_j (j=1，2，…，26)的对照结果，从图可知拟合程度较好。

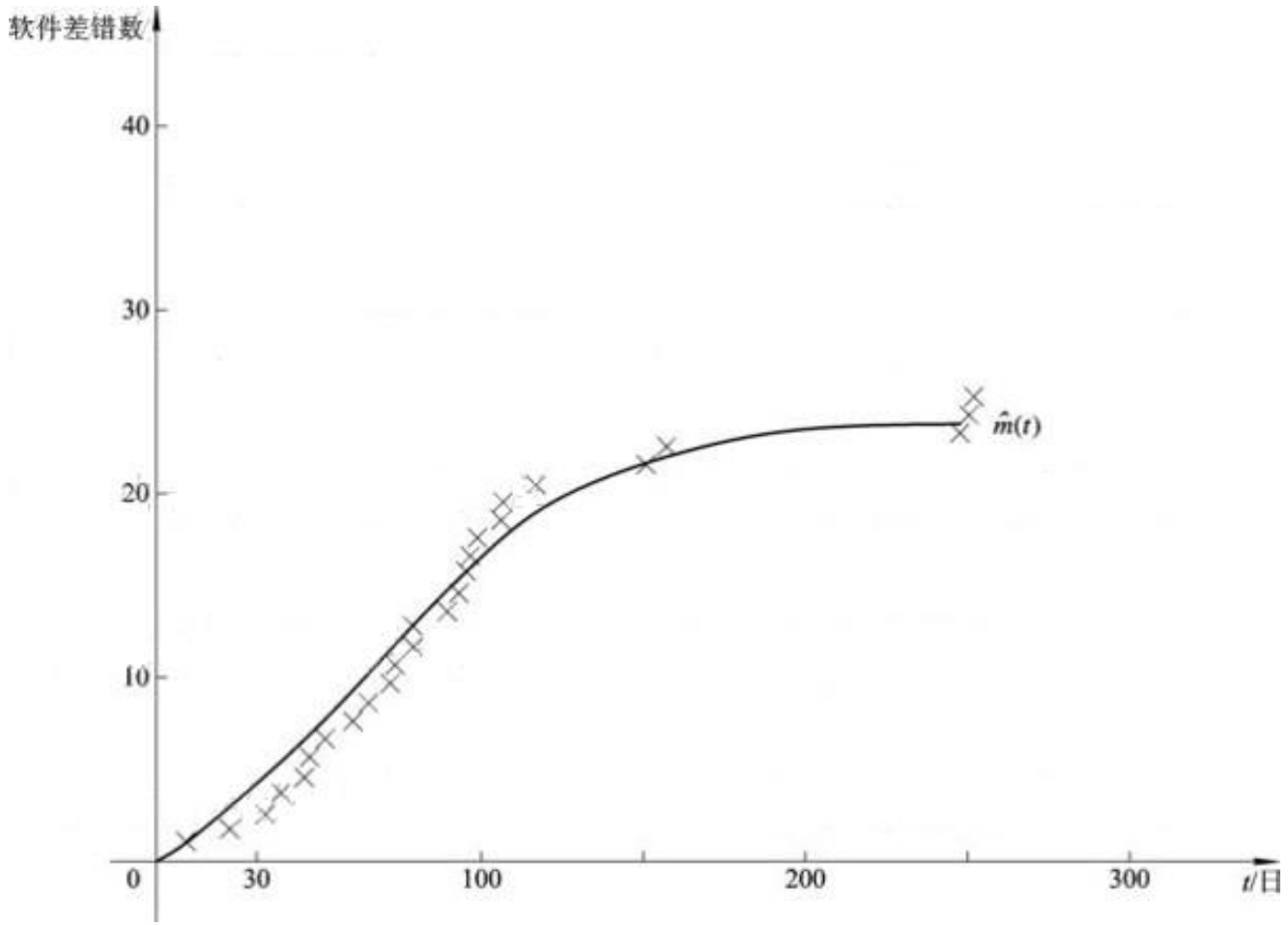

图7.6 样本数据序列与拟合曲线对照

7.1.3 软件产品质量水平评价模型

在采用G-O模型或其他可靠性增长模型估计被测软件的潜在初始固有差错数时，其估计精度应为人们所关注。显然，此软件的潜在初始固有差错数的估计值将大大依赖于软件测试的覆盖率和测试质量，而软件测试的覆盖率又将依赖于投入的测试工作量或测试人力数、确认的程序路径数等因素，而测试质量则依赖于被测软件的差错属性(如差错能否易于正确分离)、差错捕捉率和遗漏率等，因此在对被测试软件评价时兼顾考虑测试覆盖率与测试质量的方法显然是人们所欢迎的，于是一种将NHPP模型与差错植入模型结合起来的组合模型——软件产品质量水平评价模型(Evaluation Model of the Software Product of Quality Level，简称SPQL模型)出现了。以下介绍其基本变量含义和模型算法。

1. 变量注释

A_c：测试精确度，$0 \le A_c \le 1$；

C_v：测试覆盖率，或在测试期间发现软件的潜在初始固有差错数的可能性，$0 \le C_v \le 1$；

N_c：经测试有可能发现的埋设伪差错数;

N_v：经测试有可能发现的潜在初始固有差错数;

M_c：测试前埋设的伪差错数，$M_c \ge N_c \ge 0$;

N_0：软件的潜在初始固有差错数，$N_0 \ge N_v \ge 0$;

n_c：经测试已发现的伪差错数，$N_c \ge n_c \ge 0$;

n_v：经测试已发现的潜在初始固有差错，Nv≥nv≥0;

m_c(t)：当测试排错过程为NHPP时，在(0，t)内查出并排除的累计伪差错数;

$m_v(t)$：当测试与排错过程为NHPP时，在(0，t)内查出并排除的累计潜在初始固有差错数;

a_c：$\lim_{t\to\infty} m_c(t) = m_c(\infty) = a_c$;

a_v：$\lim_{t\to\infty} m_v(t) = m_v(\infty) = a_v$;

s_j：在测试过程中查出并排除的第j个伪差错数的时刻，j=1，2，…，n_c;

$\tilde{s}_j$：在测试过程中查出并排除的第j个潜在初始固有差错数的时刻，j=1，2，…，n_v;

t_j：在测试过程中查出并排除伪差错的时间间隔，$t_j=s_j-s_{j-1}$，s_0=0，j=1，2，…，n_c;

$\tilde{t}_j$：在测试过程中查出并排除潜在初始固有差错的时间间隔，$\tilde{t}_j = \tilde{s}_j - \tilde{s}_{j-1}, \tilde{s}_0 = 0$，$j$=1，2，…，$n_v$;

SPQL：被测软件质量水平，0≤SPQL≤1。

2. 模型算法与求解步骤

在满足差错植入模型(见作者文献[15])的四条模型假设条件下，可通过如下步骤求解被测软件的质量水平。

(1) 在完成编译的被测软件中，随机埋设人为制造的M_c个差错，然后将软件交由专门测试小组进行测试。

(2) 对于已埋设差错一无所知的测试小组对软件进行例行测试，共发现并排除M个差错，对这M个差错分析得知其中有伪差错n_c个，潜在初始固有差错n_v个($n_c+n_v=M$)。并在测试过程中记录下伪差错发现时间序列$\{s_j，\ j=1，2，\ldots，n_c\}$或伪差错发现时间间隔序列$\{t_j，\ j=1,2,\ldots,n_c\}$，同时也记录下潜在初始固有差错发现时间序列$\{\tilde{s}_j, j=1,2,\cdots n_v\}$或潜在初始固有差错发现时间间隔序列$\{\tilde{t}_j, j=1,2,\cdots n_v\}$。

(3) 对已获得的时间序列$\{s_j，\ j=1，\ 2，\ \cdots，\ n_c\}$或$\{t_j，\ j=1，\ 2，\ \cdots，\ n_c\}$运用G-O模型族中任何一种NHPP模型，如指数型可靠性增长模型进行拟合和参数估计，可得参数估计值$\hat{a}_c$与$\hat{b}_c$，从而可得指数可靠性增长模型:

$$m_c(t)=a_c(1-\mathrm{e}^{-b_c t}),a_c=m_c(\infty)=\lim_{t\to\infty}m_c(t)$$

(4) 对已获得的另一时间序列$\{\tilde{s}_j, j=1,2,\cdots n_v\}$或$\{\tilde{t}_j, j=1,2,\cdots n_v\}$运用与上同样的NHHP模型，如指数可靠性增长模型进行拟合和参数估计，可得参数估计值$\hat{a}_v$与$\hat{b}_v$，从而可得另一个指数可靠性增长模型:

$$m_v(t)=a_v(1-\mathrm{e}^{-b_v t}),a_v=m_v(\infty)=\lim_{t\to\infty}m_v(t)$$

(5) 注意到测试精度$A_c = N_c/M_c \approx m_c(\infty)/M_c = a_c/M_c$，而测试覆盖率有

$$C_v = \frac{n_v}{N_v} \approx \frac{n_v}{m_v(\infty)} = \frac{n_v}{a_v}$$

定义SPQL=$A_c \cdot C_v$，从而有

$$\mathrm{SPQL} = A_r \cdot C_v \approx \frac{a_c}{M_c} \cdot \frac{n_v}{a_v}$$

(6) 利用(7.4)式求解条件可靠度函数$R(t|T)$。

[例7.2] 某软件在测试前已埋设伪差错M_c=21个，经测试后查出埋设伪差错n_c=20个，查出潜在初始固有差错数n_v=23个，并同时获得查出的埋设伪差错时间序列{s_j，j=1，2，…，20}和查出的潜在初始固有差错时间序列{ $\tilde{s}_j$ ，j=1，2，…，23}(具体数据在此从略)。试求解：

(1) 测试精确度A_c，测试覆盖率C_v，软件产品质量水平SPQL，并进而判断该软件测试的有效性；

(2) 该软件的潜在初始固有差错数N_0及经测试排错后的残存差错数，以及经测试后的软件质量水平；

(3) 该软件在经历 $\tilde{s}_{23}$ =T=15周测试后的软件可靠度函数$R(t|T)$。

解 (1) 对已获得的伪差错查出时间序列{s_j，j=1，2，…，20}采用指数型可靠性增长模型拟合，并利用(7.7)式和(7.8)式可获得参数估计值 $\hat{a}_c = 20.5, \hat{b}_c = 0.372$ 。

同理，对已获得的潜在初始固有差错查出时间序列{ $\tilde{s}_j$，j=1，2，…，23}采用指数型可靠性增长模型拟合，并利用(7.7)式和(7.8)式求得参数估计值 $\hat{a}_v = 24.6, \hat{b}_v = 0.338$ 。从而有如下两个拟合模型：

$$m_c(t) = a_c(1-\mathrm{e}^{-b_c t}) \approx \hat{a}_c(1-\mathrm{e}^{-\hat{b}_c t}) = 20.5(1-\mathrm{e}^{-0.372\,t})$$

$$m_c(\infty) = \lim_{t\to\infty} m_c(t) \approx \hat{a}_c = 20.5$$

$$m_v(t) = a_v(1-\mathrm{e}^{-b_v t}) \approx \hat{a}_v(1-\mathrm{e}^{-\hat{b}_v t}) = 24.6(1-\mathrm{e}^{-0.338\,t})$$

$$m_v(\infty) = \lim_{t\to\infty} m_v(t) \approx \hat{a}_v = 24.6$$

由此可求得

$$\hat{A}_c = \frac{\hat{a}_c}{M_c} = \frac{20.5}{21} = 0.976, \hat{C}_v = \frac{n_v}{\hat{a}_v} = \frac{23}{24.6} = 0.935,$$

$$\text{SPQL} = \hat{A}_c \cdot \hat{C}_v = 0.976 \cdot 0.935 = 0.913$$

由上述结果可知软件测试的精确度为97.6%，测试覆盖率为93.5%，因此该测试可认为是有效的。

(2) 软件测试前的潜在初始固有差错数N_0和经测试并排错后的残存差错数x_v分别为

$$\hat{N}_0 = \frac{M_c \cdot n_v}{n_c} = \frac{21 \times 23}{20} = 24$$

$$\hat{x}_v = \hat{N}_0 - n_v = 24 - 23 = 1$$

上述计算结果表明该软件在测试前质量属中等水平(SPQL=0.913，$\hat{N}_0 = 24$)，但经测试与排错后，该软件已有较高的质量水平$(\hat{x}_v = 1)$，因此一般来说，可交付用户并作运行使用。

(3) 注意到在经历了 $\tilde{s}_{23} = T = 15$ 周后，由(7.4)式可得条件可靠度函数为

$$R(t \mid T)\big|_{T=15} = \exp\left\{-a_v(1-\mathrm{e}^{-b_v t})\mathrm{e}^{-b_v T}\right\} \approx \exp\left\{-\hat{a}_v(1-\mathrm{e}^{-\hat{b}_v t})\mathrm{e}^{-\hat{b}_v T}\right\}$$

$$= \exp\left\{-24.6(1-\mathrm{e}^{-0.338t})\mathrm{e}^{-0.338\times 15}\right\} = \exp\left\{-0.1547(1-\mathrm{e}^{-0.338t})\right\}$$

7.2 软件测试的资源分配与进度管理

前述的可靠性增长模型均是通过测试过程中所耗费的测试时间、所发现的潜在初始固有差错数以及差错发现时间序列等的数量关系来研究软件的可靠性增长规律，并进而来获得被测软件的可靠性预计模型的。在这些模型中，测试人力的投入并未直接进入模型，而是通过测试期间所发现的差错数多少及发现差错的时间先后间接地来影响软件的可靠性增长。事实上，如果将测试人力投入的时间分布进行数量描述并直接进入模型，然后通过对测试人力投入的时间分布和所发现的差错数、所耗费的测试时间等的数量关系来研究软件的可靠性增长规律，这对于合理安排测试人力，提高软件测试效率，加强测试进度管理以及确定被测软件的最优交付期限将是有益的。为此，S.Yamada等人利用负指数函数和瑞利函数来描述测试人力投入时间分布，并进而建立了软件的最优交付期限(又称最优发行时间)模型。

作者在S.Yamada等人的研究基础上采用威布尔曲线来描述测试人力投入时间分布，并进而建立了一个在更广泛意义下的可靠增长模型，最后讨论了软件测试的资源分配、进度管理与其最优发行问题。以下介绍作者所建立的基于威布尔分布的可靠性增长模型。

7.2.1 考虑测试人力投入的可靠性增长模型

1. 符号注释

$w(t)$：当前测试时刻t投入的测试人力密度，其积分形式为 $W(t)=\int_0^t w(x)\mathrm{d}x$， 表示(0，t)内投入的累计人力数；

α、β、m：测试人力函数中的参数，$\alpha>0$，$\beta>0$，$m>0$；

$t_{w\max}$: w(i)达到最大值的时刻；

$m(t)$: (0，t)时间内NHPP模型中的差错发现数期望值函数，$m(0)=0$；

a：测试开始前潜在固有差错总数期望值；

r：相应于软件内残存一个差错及单位测试人力投入量的差错发现率；

$r(t)$: t时刻的残存差错数期望值函数；

$R(x|t)$：条件软件可靠度，即假定对软件的测试已实施到时刻t，在$[t，t+x)$时间段内软件的无故障概率；

C_1、C_2：在测试或运行阶段分别修改一个差错所需的成本；

C_3：单位测试人力成本；

T_{LC}：软件生命周期(从测试开始时算起)；

T：总测试时间或软件释放时间；

$C(T)$：对应于$(0，T)$时间区间内总平均软件成本；

T^*：最佳软件释放时间；

$A(T)$：表示对应于当前测试人力下，单位人力在T时刻后每单位时间平均发现差错数，$A(0)=ar$，$A(\infty)=ar\cdot\exp(-ar)$；

C_4：超过工期改造软件的罚金费用；

T_s：软件工期；

W_G：测试过程中投入累计人力资源目标值；

m_G：测试剩余差错期望目标值。

2. 基于威布尔曲线测试人力函数的可靠性增长模型

本模型有如下假设：

(1) 软件系统由于受到残留在系统中的差错引发的故障的影响，其失效是随机的；

(2) 软件差错一经发现立即被修正和排除，修改差错的作业不会造成新差错；

(3) 测试人力采用威布尔曲线描述；

(4) 单位时间内差错发现数与软件内残存差错数成正比，也与投入的测试人力成正比；

(5) 软件测试中差错发现过程采用NHPP模型描述。

注意到在许多软件测试情况下，由于实际测试结果数据变化多样，很难用指数或瑞利曲线去描绘，因此我们采用如下的威布尔(Weibull)曲线作为测试人力函数去描述在测试时刻 t 的测试人力情况：

$$\omega(t)=\alpha\beta m t^{m-1}\cdot\exp[-\beta t^{m}] \quad , \quad t\geq 0,\alpha>0,\beta>0,m>0 \qquad (7.18)$$

其中，α表示测试所需总测试人力；β表示测试人力投入率；m表示测试人力投入模型的形状参数。事实上威布尔曲线是指数曲线和瑞利曲线更广泛的一种形式，因为(7.18)式当m=1时，测试人力函数表现为指数曲线，即有函数

$$\omega(t)=\alpha\beta\exp[-\beta t] \qquad t\geq 0，\ \alpha>0，\ \beta>0$$

当m=2时，测试人力函数则为瑞利(Rayleigh)曲线，即有

$$\omega(t)=2\alpha\beta t\exp[-\beta t^2] \qquad t\geq 0，\ \alpha>0，\ \beta>0$$

(7.18)式的积分形式为

$$W(t)=\int_0^t \omega(t)\,\mathrm{d}t=\alpha[1-\exp(-\beta t^m)] \qquad t\geq 0 \tag{7.19}$$

由于$W(\infty)=\alpha$，故$W(t)$与一般的概率分布函数略有区别，但更符合实际情况。

由假设(5)，设$N(t)$表示(0，t)内发现的差错总数，则$N_T=\{N(t)，t\geq 0\}$服从NHPP，利用非齐次泊松过程有关特性可得到条件可靠度

$$R(x|t)=\exp\{-m(t+x)+m(t)\} \qquad t\geq 0，\ x\geq 0 \tag{7.20}$$

利用非齐次泊松过程(NHPP)的有关特性还可以建立在(0，t)内经测试发现的差错数期望$m(t)$的微分方程，再运用初始条件$m(0)=0$，可解得

$$\begin{cases} m(t)=a[1-\exp[-rW(t)] \\ \lambda(t)=\dfrac{\mathrm{d}\,m(t)}{\mathrm{d}\,t}=ar\cdot\omega(t)\cdot\exp(-rW(t)) \end{cases} \tag{7.21}$$

注意到$W(\infty)=\alpha$，故有

$$m(\infty)=a(1-\exp(-r\cdot\alpha)) \tag{7.22}$$

(7.22)式隐含意义为：即使软件系统的测试时间无限长，系统中的所有差错也不能被全部发现，测试未发现的差错数的均值为$a\exp(-r\alpha)$。

利用(7.20)式与(7.21)式可以得到两个定量评价软件可靠性的指标：NHPP模型t时刻残存差错数的期望值和条件可靠度，分别为

$$\begin{cases} r(t) = m(\infty) - m(t) = a\{\exp[-rW(t)] - \exp[-rW(\infty)]\} \\ R(x \mid t) = \exp\{-a[\exp(-rW(t)) - \exp(-rW(t+x))]\} \end{cases} \tag{7.23}$$

注意到在上述可靠性指标$R(x|t)$，$r(t)$中均含有Weibull累计人力资源投入函数$W(t)$及参数a和r，为此应对$W(t)$与a和r作出估计。首先我们来考虑$W(t)$的估计。Weibull人力资源投入密度函数$\omega(t)$为

$$\omega(t) = \alpha \cdot \beta \cdot m \cdot t^{m-1} \cdot \exp(-\beta t^m) \qquad t \geq 0，\ \alpha > 0，\ \beta > 0，\ m > 0$$

$$\ln\omega(t) = \ln\alpha + \ln\beta + \ln m + (m-1)\ln t - \beta t^m$$

若经测试已得到了观测数据序列(t_k, ω_k)，$k=1, 2, \ldots, n$，其中ω_k表示在时刻t_k投入的测试资源数，则采用最小二乘法，可建立如下的$S(\alpha, \beta, m)$来估计参数α、β及m:

$$S(\alpha,\beta,m)=\sum_{k=1}^{n}[\ln\omega_k-\ln\alpha-\ln\beta-\ln m-(m-1)\ln t_k+\beta t_k^m]^2 \qquad (7.24)$$

$S(\alpha, \beta, m)$分别对α、β、m求偏导并令其等于零，然后运用有关的数值解法求解如下方程组，从而可得到满足关系式 $S(\hat{\alpha},\hat{\beta},\hat{m})=\min S(\alpha,\beta,m)$ 时的 $\hat{\alpha},\hat{\beta}$及$\hat{m}$ ，并用其作为α、β及m的估计值：

$$\begin{cases}\dfrac{\partial S}{\partial\alpha}=0\\ \dfrac{\partial S}{\partial\beta}=0\\ \dfrac{\partial S}{\partial m}=0\end{cases} \qquad (7.25)$$

由(7.18)式与(7.19)式知$\omega(t)$与$W(t)$是α、β与m的函数，通过(7.25)式方程组可求得α、β、m的估计值，从而也就得到了$\omega(t)$与$W(t)$的具体估计式。

于是，在得到了$W(t)$(或$\omega(t)$)的估计式后，即可采用最大似然估计法来对可靠性增长模型中$m(t)$的参数a和r作出估计。因为由模型假设(5)知有$\{N(t)，t\geq 0\}$为NHPP，其中$N(t)$为$(0，t)$时间区间内检测到的差错累计数，当通过测试获得了观测数据序列$(t_k，y_k)$，$k=1，2，\ldots，m$时，其中y_k表示在时间区间$(0，t_k)$内观察到的差错累计数，则由NHPP性质，建立似然函数L为

$$L(a,r)=\prod_{k=1}^{m}P(N(t_{k-1},t_k)=y_k-y_{k-1})$$

$$=\prod_{k=1}^{m}\frac{[m(t_k)-m(t_{k-1})]^{y_k-y_{k-1}}}{(y_k-y_{k-1})!}\cdot\exp[-m(t_k)+m(t_{k-1})]$$

$$=\prod_{k=1}^{m}\frac{\{a(\mathrm{e}^{-rW(t_{k-1})}-\mathrm{e}^{-rW(t_k)})\}^{y_k-y_{k-1}}}{(y_k-y_{k-1})!}\cdot\exp\{-a[\mathrm{e}^{-rW(t_{k-1})}-\mathrm{e}^{-rW(t_k)}]\}$$

对上式两边取对数有

$$\ln L(a,r)=\sum_{k=1}^{m}(y_k-y_{k-1})\ln a+\sum_{k=1}^{m}(y_k-y_{k-1})\ln(\mathrm{e}^{-rW(t_{k-1})}-\mathrm{e}^{-rW(t_k)})$$

$$-\sum_{k=1}^{m}\ln[(y_k-y_{k-1})!]-\sum_{k=1}^{m}a(\mathrm{e}^{-rW(t_{k-1})}-\mathrm{e}^{-rW(t_k)})$$

为求$L(a，r)$极小值，对$\ln L(a，r)$分别对a与r求偏导并令其为零，有

$$\frac{\partial \ln L(a,r)}{\partial a}=\frac{1}{a}\sum_{k=1}^{m}(y_k-y_{k-1})-\sum_{k=1}^{m}[\mathrm{e}^{-rW(t_{k-1})}-\mathrm{e}^{-rW(t_k)}]=0 \tag{7.26}$$

$$\frac{\partial \ln L(a,r)}{\partial r}=\sum_{k=1}^{m}(y_k-y_{k-1})\cdot\frac{W(t_k)\mathrm{e}^{-rW(t_k)}-W(t_{k-1})\mathrm{e}^{-rW(t_{k-1})}}{\mathrm{e}^{-rW(t_{k-1})}-\mathrm{e}^{-rW(t_k)}}$$

$$+\sum_{k=1}^{m}a[W(t_{k-1})\mathrm{e}^{-rW(t_{k-1})}-W(t_k)\mathrm{e}^{-rW(t_k)}]=0 \tag{7.27}$$

由(7.26)式解出a并代入(7.27)式，即可得到仅含r的一个非线性代数方程组，运用适当的数值计算方法(如迭代法)即可获得a与r的估计值 $\hat{a}$ 与 $\hat{r}$ 。注意到残存差错数期望值$r(t)$是a与$W(t)$的函数，而条件可靠度$R(x|t)$也是r与$W(t)$的函数，因而也就使$r(t)$与$R(x|t)$的具体表达式得到了求解。

3. 案例

[例7.3] 某软件开发机构采用PL/1编写了规模为1317kLOC的数据库应用程序，现对其进行耗时19周的可靠性测试，共获得19组数据$\{(t_k, w_k, y_k), k=1, 2, \ldots, 19\}$如表7.4所示。其中，$t_k$为第$k$个测试时段(单位：周)，$w_k$为在$t_k$时段内投入的测试人力(CPU小时)，$y_k$为在$(0, t_k)$时间区间内发现且排除的累计差错数。试求：

(1) 测试人力投入密度$w(t)$和人力投入累计函数$W(t)$;

(2) 期望累计差错数$m(t)$和软件可靠度函数$R(x|t)$;

(3) 残存期望差错数$n(t)$，自t后经无限测试能发现的期望差错数$r(t)$。

表 7.4 测 试 数 据 表

t_k/周	1	2	3	4	5	6	7	8	9	10
w_k/时	2.45	2.45	1.98	0.98	1.68	3.37	4.21	3.37	0.96	1.92
y_k	15	44	66	103	105	110	146	175	179	206
t_k/周	11	12	13	14	15	16	17	18	19	
w_k/时	2.88	1.44	3.26	3.84	3.84	2.30	1.76	1.99	2.99	
y_k	233	255	276	298	304	311	320	325	328	

解 (1) 由人力投入密度函数

$$\omega(t)=\alpha\beta m t^{m-1}\mathrm{e}^{-\beta t^m} \qquad t\geq 0$$

可得平方误差函数(见(7.24)式)

$$S(\alpha,\beta,m)=\sum_{k=1}^{n}[\ln\omega_k-\ln\alpha-\ln\beta-\ln m-(m-1)\ln t_k+\beta t_k^m]^2$$

观察表7.4中$\omega_k=\omega(t_k)$之几何图形知，可用$m=2$时的瑞利分布来作曲线拟合。即有

$$\omega(t)=2\alpha\beta t\,\mathrm{e}^{-\beta t^2} \qquad t\geq 0$$

令 $\dfrac{\partial S}{\partial\alpha}=0,\dfrac{\partial S}{\partial\beta}=0$,求得左述代数方程组得 $\hat{\alpha}=55.21,\hat{\beta}=5.1702\times10^{-3}$ ，从而可得

$$\hat{\omega}(t) = 2\alpha\beta t\,\mathrm{e}^{-\beta t^2} = 0.5709t\,\mathrm{e}^{-5.1702*10^{-3}t^2} \qquad t \geq 0$$

$$\hat{W}(t) = \alpha\left(1-\mathrm{e}^{-\beta t^m}\right) = 55.21\left(1-\mathrm{e}^{-5.1702\times10^{-3}t^2}\right) \qquad t \geq 0$$

(2) 为求$m(t)$之参数α和γ，可将上述求得的 $\hat{\omega}$ 及 $\hat{W}(t)$ 以及表7.4中的y_k数据代入(7.26)、(7.27)式，可解得

$$\hat{a} = 395.8, \hat{\gamma} = 3.7806\times10^{-2}$$

从而得到

$$\hat{m}(t) = a\left[1-\mathrm{e}^{-\gamma\hat{W}(t)}\right] = 395.8\left[1-\mathrm{e}^{-3.7806\times10^{-2}\bullet\hat{W}(t)}\right]$$

其中 $\hat{W}(t)$ 由(7.28)式确定。

$$R(\hat{x}/t)=\exp\left\{-a\left(\mathrm{e}^{-\gamma\hat{W}(t)}-\mathrm{e}^{-\gamma\hat{W}(t+x)}\right)\right\}$$

$$=\exp\left\{395.8\left(\mathrm{e}^{-3.7806\times10^{-2}\hat{W}(t+x)}-\mathrm{e}^{-3.7806\times10^{-2}\times\hat{W}(t)}\right)\right\}$$

(3) 经测试后残存期望差错数

$$n(t)=a-y_{19}=395.8-328\approx68$$

经无限测试后仍无法发现的期望差错数由(7.22)式知为

$$a-m(\infty)=ae^{-\gamma\alpha}=395.8\times\exp\{-3.7806\times10^{-2}\times55.21\}\approx47$$

故自y_{19}后至无限测试后能发现的期望差错数为

$$\hat{\gamma}(t)=68-47=21$$

7.2.2 软件测试中的静态资源分配与进度管理

在软件测试中，测试人力资源分配与进度管理是企业高层管理与项目经理所关注的重要问题之一。上述问题实际上是测试人力投入量、测试工期(进度)及测试可靠目标三者的适度均衡问题。这样的资源分配与进度管理一般需要解决如下几个问题：

(1) 在软件开发机构人力资源充分的情况下，给出测试人力分配与进度计划表，并据此实施跟踪与控制。

(2) 在软件开发机构人力资源充分的情况下，对于给定的工期T，求解在(0，T)内能查出(排除)的期望累计差错数a_T及在(t_n，T)内尚需要投入的人力资源数ΔW_T。此中t_n为预先所作的初期测试过程的耗费时间。

(3) 当软件开发机构人力资源为有限数W_0时，若按照初期测试过程的人力投入规律，求解将W_0全部开销完时所需的时间T_0以及在(0，T_0)内能查出并排除的期望差错数a_0与可靠性比例G_0。

(4) 对于给定的计划目标值G^*，当有限资源量w_0只能完成的可靠性比例G_0小于G^*时，求解为完成计划目标值G^*的人力资源缺口ΔW^*，以便为人力资源部门引进或借调测试人员提供信息。

利用上节介绍的基于Weibull人力资源投入的可靠性增长模型，容易解决上述问题。其求解的步骤如下：

(1) 软件开发机构做初期测试过程，从而获得数据序列$\{(t_k,w_k,y_k)，k=1,2,\ldots,n\}$。其中，$t_k$为初期测试过程中第$k$个测试时段；$w_k$为在$t_k$时段内软件机构投入的测试人力数；$y_k$为在$(0，t_k)$内发现且排除的累计差错数。利用(7.25)式可求解 $\hat{\alpha},\hat{\beta},\hat{m}$ ，利用(7.26)、(7.27)式可求解 $\hat{a},\hat{r}$ ，从而使$m(t)$与$W(t)$成为已知，即有

$$\hat{m}(t)=\hat{a}\left(1-\mathrm{e}^{-r\hat{W}(t)}\right),\quad \hat{W}(t)=\hat{\alpha}\left(1-\mathrm{e}^{-\hat{\beta}t^{\hat{m}}}\right)$$

在软件机构人力资源充分的情况下，若设a_j表示(0，T_j)内能查出并排除的差错目标量(计划量)，注意到a表示测试开始前软件的初始潜在固有差错数，故a_j/a可作为软件测试可靠性目标(比例)。不妨设$a_j/a=G_j$，G_j可取20%，40%，60%，80%，90%，…。由于a估计值 在期初测试过程已求得，故 $a_j=\hat{a}\bullet G_j$ 亦可求得，另一方面，考虑到$m(t)$表示在(0，t)内查出并排除的期望差错数，为求解能达到可靠性目标a_j时应测试的时间T_j，可令

$$m(t)=a(1-\mathrm{e}^{-rW(t)})=a_j$$

或有

$$W(t)=-\frac{1}{r}\ln\left(1-\frac{a_j}{a}\right)=-\frac{1}{r}\ln\left(1-G_j\right) \tag{7.29}$$

另一方面，由(7.19)式有

$$W(t)=\alpha\left(1-\mathrm{e}^{-\beta t^{m}}\right)$$

综合上式与(7.29)式有

$$-\frac{1}{r}\ln\left(1-G_j\right)=\alpha\left(1-\mathrm{e}^{-\beta t^{m}}\right)$$

从而可解得达到测试可靠性目标a_j(或G_j)时需测试的时间T_j为

$$T_j=\left[-\frac{1}{\beta}\ln\left(1+\frac{1}{\alpha r}\ln\left(1-G_j\right)\right)\right]^{\frac{1}{m}} \tag{7.30}$$

经初期测试(耗时t_n)后为达可靠性目标a_j(或G_j)尚需投入的人力资源量为

$$\Delta W_j=W\left(T_j\right)-W\left(t_n\right)=-\frac{1}{r}\ln\left(1-G_j\right)-\sum_{k=1}^{n}w_k \tag{7.31}$$

对于给定的可靠性目标，利用(7.30)式与(7.31)式可得到测试人力分配与进度计划表如表7.5所示，利用此表可画出相应的测试进度横道图，并据此可作测试进度跟踪与控制。有关初期测试过程及目标测试过程的时间顺序可详见图7.7。

表 7.5　测试人力分配与进度计划表

可靠性目标	$a_1(G_1)$	$a_2(G_2)$	…	$a_K(G_K)$
时间段	T_1	T_2	…	T_K
人力资源分配	ΔW_1	ΔW_2	…	ΔW_K

图7.7　测试时间过程

(2) 在软件机构人力资源充分的情况下，对于给定的工期T，可以求得$(0，T)$内能查出与排除的期望累计差错数a_T为

$$a_T = m(T) = \hat{a}\left(1 - \mathrm{e}^{-\hat{r}W(T)}\right)$$

其中

$$W(T) = \hat{\alpha}\left(1 - \mathrm{e}^{-\hat{\beta}T^m}\right)$$

在$(t_n，T)$内，尚需投入的人力资源数ΔW_T为

$$\Delta W_T = W(T) - W(t_n) = \hat{\alpha}\left(1 - \mathrm{e}^{-\hat{\beta}T^m}\right) - \sum_{k=1}^{n} w_k$$

(3) 软件开发机构的测试人力为有限数W_0时，按照初期测试过程的人力投入规律，将W_0全部开销完所需要的时间T_0可由下式求解：

$$W_0 = W(T_0) = \hat{\alpha}\left(1 - \mathrm{e}^{-\hat{\beta}T_0^m}\right) \quad \text{或} \quad T_0 = \left[-\frac{1}{\beta}\ln\left(1 - \frac{W_0}{\alpha}\right)\right]^{\frac{1}{m}}$$

在(0，T_0)内能完成的可靠性指标为

$$a_0 = m(T_0) = \hat{a}\left(1 - \mathrm{e}^{-\hat{r}W(T_0)}\right) = \hat{a}\left(1 - \mathrm{e}^{-\hat{r}W_0}\right)$$

$$G_0 = \frac{a_0}{a} = 1 - \mathrm{e}^{-\hat{r}W_0}$$

(4) 软件开发机构的测试人力为有限数W_0时，将W_0全部开销完后能达到的可靠性指标G_0仍然不能满足可靠性目标G^*时($G_0<G^*$)，可以计算出该软件开发机构为达到可靠性目标G^*时的测试人力缺口ΔW^*为

$$\Delta W^* = W\left(T^*\right) - W_0 = -\frac{1}{\hat{r}}\ln\left(1-G^*\right) - W_0$$

7.2.3 软件测试中的动态资源分配

观察7.2.2节的测试人力分配模型，发现有如下两个特点：① 测试人力资源分配只分配到系统级而未涉及到软件系统的各模块。② 测试人力资源分配是在软件可靠性测试前完成的(在此之前需作初期测试过程)，它与测试后不同时间的残存差错数大小无关。我们将这样的测试人力分配称为静态资源分配。本节将介绍作者提出的一种新的软件测试人力资源动态分配模型(文献[21])，该模型将测试人力分配深入到模块级，并且测试后的人力资源分配并非一次性完成，而是依据测试各阶段各模块的残存差错数的变动过程来动态分配测试人力。显然，这样的测试人力资源分配将是高效率的。

以下介绍作者给出的两种动态资源分配模型：① 当所投入的测试人力资源总数一定时，以软件模块中剩余差错数平均值最小为目标的模型；② 当给定测试结束，软件模块中剩余差错数平均值达到预定的指标时，以所用的测试人力资源最少为目标的模型。下面分别介绍两种模型的建立方法和求解过程。

1. 符号注释

Q：测试投入人力资源总量；

M：软件中模块数；

K：测试阶段数；

Q_j：分配于测试阶段j的测试投入人力资源量，$Q=\sum_{j=1}^{K}Q_j$；

T_j: 第j个测试阶段开始时刻，j=1，2，…，K;

$N_{ij}(t)$: 第i个模块在第j个测试阶段时，从T_j开始直到时刻t累计检测到的差错总数，$T_j \le t < T_{j+1}$，i=1，2，…，M；j=1，2，…，K;

$m_{ij}(t)$: $N_{ij}(t)$的期望函数，$m_{ij}(T_j)$=0，i=1，2，…，M；j=1，2，…，K;

v_i: 加权因子, 由模块i的相对重要性和操作频度确定, 如果所有模块同等重要且操作频度近乎相等, 则

$v_1=v_2=\ldots=\ldots v_M=1$;

a_{ij}: 第i个模块在第j个测试阶段开始时的剩余差错数;

ω_{ij}: 第i个模块在第j个测试阶段中投入的测试人力资源密度;

z_{ij}: 第i个模块在第j个测试阶段中结束时的剩余差错数;

$t^{(ij)}_k$: 第i个模块在第j个测试阶段中检测到第k个差错的时刻;

X_{ij}: 第i个模块在第j个测试阶段中检测到的差错总数;

z_0: 给定软件模块中剩余差错数达到的预定指标;

R_{ij}: 第i个模块在第j个累计测试阶段中分配的人力资源。

2. 模型假设

(1) 在每一个测试阶段中的任一时间区间[t，$t+\Delta t$)内，平均检测到的差错数目和Δt成正比，和人力资源密度(单位时间内所消耗的资源)成反比，和软件中剩余差错数成正比；

(2) 在每一测试阶段中，所有模块中检测到的累计差错数服从NHPP(非齐次泊松过程)；

(3) 错误一经检测，立即排除，并不引入新的错误。

3. 模型建立及参数估计

假设第j个测试阶段是从T_j时刻到T_{j+1}时刻，则定义第j个累计测试阶段是从时刻T_j开始到时刻T_{K+1}结束的时间段。显然，第j个累计测试阶段的时间长度为第j个测试阶段、第j+1个测试阶段、……、第K个测试阶段时间长度的累加，即有$T_{K+1}-T_j$，j=1，2，…，K，如图7.8所示。

(1) 当所投入的人力资源总数一定时，以软件模块中剩余差错平均值最小为目标的模型：

首先考虑软件可靠性模型的建立。根据假设(1)，第i个模块在第j个测试阶段有

$$\frac{\mathrm{d}m_{ij}(t)}{\mathrm{d}t}=r_{ij}\cdot\left\{\omega_{ij}\left[a_{ij}-m_{ij}(t)\right]\right\}\qquad T_j<t<T_{j+1} \tag{7.32}$$

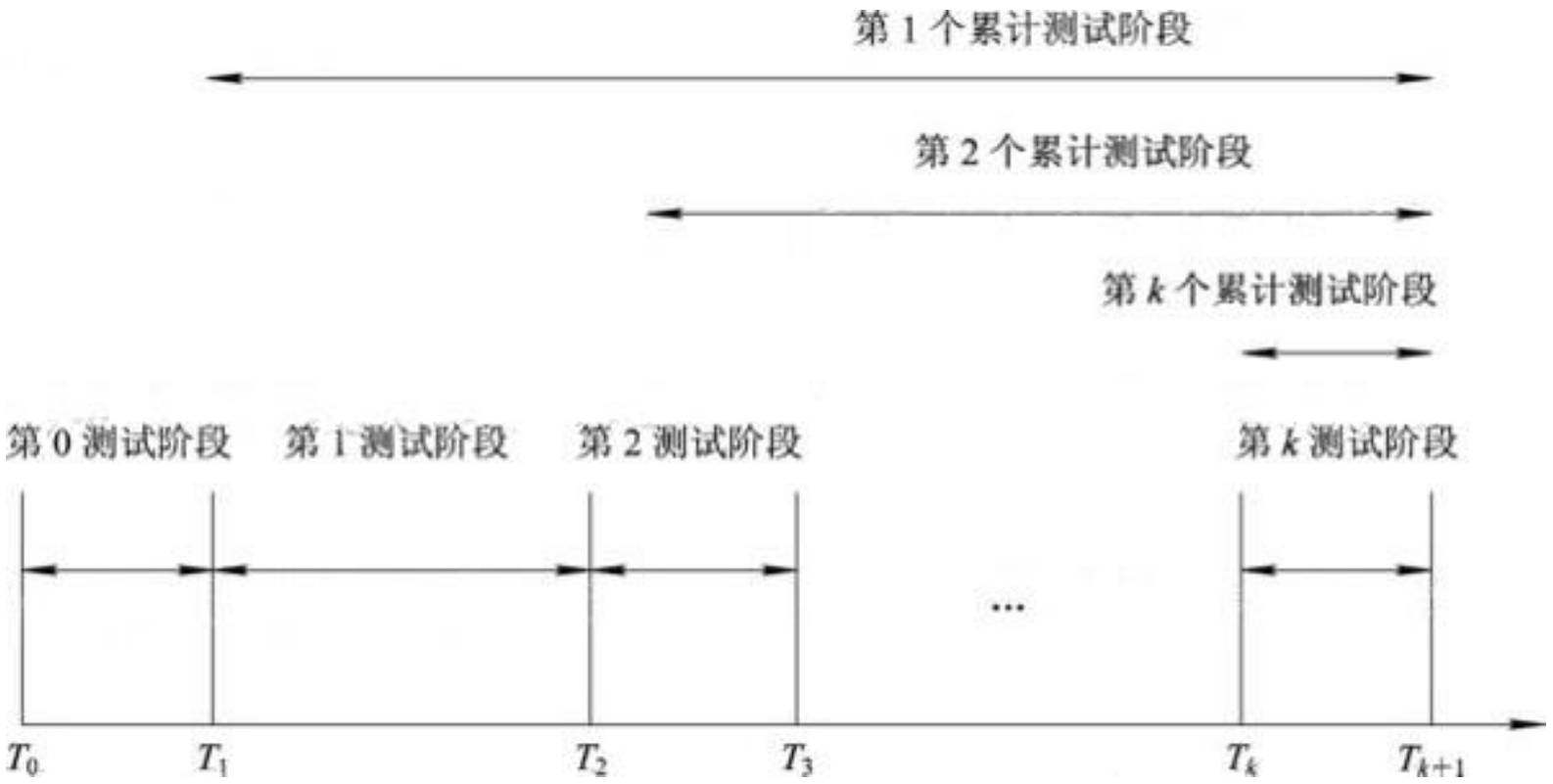
第 1 个累计测试阶段
第 2 个累计测试阶段
第 k 个累计测试阶段
第 0 测试阶段
第 1 测试阶段
第 2 测试阶段
第 k 测试阶段
...
T_0
T_1
T_2
T_3
T_k
T_{k+1}

图7.8 测试阶段示意图

其中，r_{ij}是比例系数。求解上述微分方程并代入初始条件，可得

$$m_{ij}(t)=a_{ij}\left[1-\mathrm{e}^{-r_{ij}\omega_{ij}\left(t-T_j\right)}\right] \qquad T_j<t<T_{j+1} \tag{7.33}$$

$$z_{ij}=a_{ij}-m_{ij}\left(T_{j+1}\right)=a_{ij}\cdot\mathrm{e}^{-r_{ij}\omega_{ij}\left(T_{j+1}-T_j\right)}=a_{ij}\cdot\mathrm{e}^{-r_{ij}q_{ij}} \tag{7.34}$$

由于r_{ij}和$r_{i,\ j+1}$表示的是相同的比例系数，只不过$r_{i,\ j+1}$是用更多的错误数据估计而已，因此在后面的推导中用符号r_i取代r_{ij}，即根据假设(1)认为r_{ij}与阶段序号j无关。

根据前述的动态分配测试资源的基本思想，采用递推估计的方法估计模型中的参数，即首先估计r_i(或r_{i1})和a_{i1}(i=1，2，…，M)。假设在时刻T_0为软件模块i分配的测试资源为Q_{i0}，然后记录下在时间区间[T_0，T_1)内检测到的失效发生次数，则可根据前述的方法及记录的失效数据估计r_{i1}和a_{i1}(i=1，2，…，M)，于是根据估计的结果为下一测试阶段分配测试资源并进行下一阶段的测试。依次类推，通过第j－1测试阶段收集到的失效数据估计参数a_{ij}(i=1, 2, …, M)，再根据估计的参数为第j阶段分配测试资源。若第j个测试阶段获取的失效数据序列为 $T_j < t_1^{(ij)} < t_2^{(ij)} < \cdots < t_{x_{ij}}^{(ij)} \le T_{j+1}$ ，则根据假设(2)及NHPP性质可建立如下似然函数：

$$L_{ij}=\prod_{k=1}^{x_{ij}}\lambda_{ij}\left(t_k^{(i,j)}\right)\exp\left[-m_{ij}\left(T_{j+1}\right)\right]=\left[\prod_{k=1}^{x_{ij}}\frac{\mathrm{d}\,m_{ij}(t)}{\mathrm{d}\,t}\Big|_{t=t_k^{(ij)}}\right]\exp\left[-m_{ij}\left(T_{j+1}\right)\right]$$

利用(7.33)式可得

$$L_{ij}=\left(a_{ij}r_i\omega_{ij}\right)^{x_{ij}}\left\{\prod_{k=1}^{x_{ij}}\exp\left[-r_i\omega_{ij}\left(t_k^{(ij)}-T_j\right)\right]\right\}\cdot\exp\left[-a_{ij}\left(1-\mathrm{e}^{-r_i\omega_{ij}\left(T_{j+1}-T_j\right)}\right)\right]$$

则第i个模块从第0个到第j个测试阶段中所有检测到的错误数据的似然函数为$S_{ij}=L_{i0}L_{i1}\cdots L_{ij}$，利用关系式

$$\ln S_{ij}=\sum_{n=0}^{j}x_{in}\ln\left[\left(a_{i(j+1)}+\sum_{k=n}^{j}x_{jk}\right)r_i\omega_{in}\right]-r_i\sum_{n=0}^{j}\omega_{in}\sum_{k=1}^{x_{in}}\left(t_k^{(in)}-T_n\right)$$
$$-\sum_{n=0}^{j}\left(a_{i(j+1)}+\sum_{k=n}^{j}x_{jk}\right)\left(1-\mathrm{e}^{-r_i\omega_{in}\left(T_{n+1}-T_n\right)}\right)\qquad i=1,2\cdots,M\qquad(7.35)$$

由于r_i已由第0测试阶段获得的失效数据估计得到，在此用最大似然估计法估计$a_{i(j+1)}$。将上式两边对$a_{i(j+1)}$求导，并令导数等于零，然后通过迭代法求解该非线性代数方程组即可得到$a_{i(j+1)}$，i=1，2，…，M。

其次来考察测试资源分配模型。将测试资源总量Q按测试时间长度均匀地分配到每一个测试阶段，则有

$$Q_j = \left(\frac{T_{j+1} - T_j}{T_{k+1} - T_1} \right) Q \tag{7.36}$$

在第j个测试阶段(j=1，2，…，K)为使各模块中剩余错误总数值为最小，则该测试阶段为各模块分配的资源应满足下式:

$$\begin{cases} \min \sum_{i=1}^{M} \nu_i a_{ij} \mathrm{e}^{-r_i q_{ij}} \\ \quad \mathrm{s.t.} q_{ij} \geq 0 \qquad i=1，2，\ldots，M \\ \quad \sum_{i=1}^{M} q_{ij} = \mathrm{Q_j} \end{cases} \tag{7.37}$$

为了求解这一最优化问题，可构造如下拉格朗日函数：

$$\Psi_j = \sum_{i=1}^{M} \nu_i a_{ij} \mathrm{e}^{-r_i q_{ij}} + \lambda_j \left(\sum_{i=1}^{M} q_{ij} - Q_j \right)$$

其中，λ_j为拉格朗日乘子。为了使Ψ_j最小，应满足

$$\frac{\partial \Psi_j}{\partial q_{ij}} = -\nu_i a_{ij} r_i \mathrm{e}^{-r_i a_{ij}} + \lambda_j = 0 \qquad i = 1, 2, \cdots M$$

从而可得第j个测试阶段的最优资源分配方案$\{q^*_{ij}$，$i=1$，2，…，$M\}$如下式所示：

$$q_{ij}^{*}=\begin{cases}\dfrac{1}{r_i}\left(\ln A_{ij}-\ln\lambda_j\right) & A_{ij}\geq\lambda_j\geq 0\\ 0 & \text{其他}\end{cases}\qquad i=1，2，\ldots，M \tag{7.38}$$

其中，$A_{ij}=v_i a_{ij} r_i$，将(7.38)式代入(7.37)式并经整理可得

$$\ln\lambda_j=\frac{\displaystyle\sum_{i:A_{ij}\geq\lambda_j}\frac{\ln A_{ij}}{r_i}-Q_j}{\displaystyle\sum_{i:A_{ij}\geq\lambda_j}\frac{1}{r_i}} \tag{7.39}$$

若将M个模块顺序重排，要求满足

$$A_{1j}\geq A_{2j}\geq\cdots\geq A_{(p-1)j}\geq\lambda_j\geq A_{pj}\geq\cdots\geq A_{Mj} \tag{7.40}$$

则(7.39)式可重写为

$$\ln \lambda_j = \frac{\sum_{i=1}^{p-1} \frac{\ln A_{ij}}{r_i} - Q_j}{\sum_{i=1}^{p-1} \frac{1}{r_i}}$$

为寻求p值，注意到λ_j应满足(7.40)式，即有

$$A_{(p-1)j} \geq \lambda_j = \exp\left[\frac{\sum_{i=1}^{p-1} \frac{\ln A_{ij}}{r_i} - Q_j}{\sum_{i=1}^{p-1} \frac{1}{r_i}}\right] \geq A_{pj}$$

依次取p=2，…，M，寻求满足上式的p值。得到p后，代入式(7.39)即可得到λ_j的值(j=1，2，…，K)，再将λ_k代入式(7.38)中可得q^*_{ij}(i=1，2，…，M；j=1，2，…，K)。

(2) 当给定测试结束，软件模块中剩余差错平均数要达到预定目标时，以所用的测试人力资源最少为目标的模型：

首先考虑软件可靠性模型。当给定软件模块中剩余差错数达到预定目标z_0时，动态资源分配的原则是使用资源最少，优化的方法是首先在T_1时刻确定第1个累计测试阶段的最优资源分配，并将这些资源均匀地分配到该累计测试阶段中的各个测试阶段，这样就很容易得到第1个测试阶段的资源分配。然后在T_2时刻根据第1个测试阶段的差错检测数据为第2个累计测试阶段分配资源，从而得到第2个测试阶段的资源分配情况。依此类推，可以得到第j个累计测试阶段的资源分配和第j个测试阶段的资源分配(j=1, 2, …, K)。根据上述求解思路，我们可根据假设建立软件可靠性模型，并用每一测试阶段的结果估计软件可靠性模型的参数。有关建立软件可靠性模型与求解参数的过程与第一种模型类似，此从略。

其次来考虑测试资源分配模型。第i个模块在第j个测试阶段(j=1，2，…，K)分配的资源应满足下述数学规划：

$$\begin{cases} \min \sum_{i=1}^{M} R_{ij} \\ \text{s.t.} \quad R_{ij} \geq 0 \quad i = 1,2,\cdots,M \\ \sum_{i=1}^{M} \nu_i a_{ij} \, \mathrm{e}^{-r_i R_{ij}} = z_0 \end{cases} \tag{7.41}$$

用拉格朗日方程求解$\{R^*_{ij}$，i=1，2，…，$M\}$，容易得到上述最优化问题的如下结果：

$$R_{ij}^{\ *} = \begin{cases} \dfrac{1}{r_i} \ln\left(\lambda_j A_{ij}\right) & A_{ij} \geq \dfrac{1}{\lambda_j} > 0 \\ 0 & \text{其他} \end{cases} \qquad i=1，2，…，M \tag{7.42}$$

其中，$A_{ij}=\nu_i a_{ij} r_j$。

用与第一种模型相似的方法可得到拉格朗日乘子λ_j的值，从而最终得到R^*_{ij}。于是第i个模块在第j个测试阶段的最优资源分配方案为

$$q_{ij}^{*}=\left(\frac{T_{j+1}-T_j}{T_{k+1}-T_1}\right)R_{ij}^{*} \qquad i=1,2,\cdots,M;\ j=1,2,\cdots,K \tag{7.43}$$

注意到对(7.41)式，一般实际问题给出 $\sum_{i=1}^{M}\nu_i a_{ij}\,\mathrm{e}^{-r_i R_{ij}}\le z_0$ ，但只要引进松弛变量即可将其化成标准型来求解，故(7.41)式不失一般性。

4. 算法步骤

总结上述求解思路可得第一种模型算法步骤如下：

(1) 给出测试阶段数k、各测试阶段起始时刻$0\le T_0<T_1<\ldots<T_k<T_{k+1}$、软件中模块个数$M$、各模块重要性因子$v_i(i=1，2，\ldots，M)$以及测试资源总量$Q+Q_0$，其中$Q_0$为第0个测试阶段投入的测试资源量；

(2) 将第0个测试阶段资源量Q_0按模块重要性分配到各模块，则第i个模块的初始资源量

$$Q_{0i}=\frac{v_i}{v_1+v_2+\cdots+v_M}\cdot Q_0 \qquad i=1,2,\cdots,M;$$

(3) 开始第0个测试阶段，经测试获得失效数据序列 $T_0 \le t_1^{(i,0)} < t_2^{(i,0)} < \cdots < t_{X_{ij}}^{(i,0)} \le T_{j+1}$ ，运用此失效数据序列采用极大似然估计法求解r_i(即r_{i1})和a_{i1}，i=1，2，…，M，其中似然函数见(7.35)式(取j=0)；

(4) 求解(7.37)式的数学规划，求得第j个测试阶段第i个模块分配的测试资源量q^*_{ij}(i=1，2，…，M)；

(5) 开始第j个测试阶段，获得该测试阶段失效数据序列 $T_j \le t_1^{(ij)} < t_2^{(ij)} < \cdots < t_{X_{ij}}^{(ij)} \le T_{j+1}$ ，并由(7.35)式中的似然函数通过极大似然估计法求解$a_{i(j+1)}$，i=1，2，…，M；

(6) 赋值，即$j \leftarrow j+1$，如果$j<K$，则转第(4)步，否则输出有关参数，结束。

第二种模型的算法步骤可依据前述的求解思路得到，在此不再详述。

5. 结论

考虑到即使用于软件模块的资源数量为一定数，测试中发现的错误数目仍然是不确定的，因此尽可能地提高测试效率，减少模块测试中检测错误数目的方差是很关键的。通过资源的动态分配，可以充分利用测试资源，达到较好的测试效果，同时有效地减少软件模块中剩余错误数的方差，可提高剩余错误数的预测精度。特别当整个测试阶段很长时，该模型的效果很明显。

7.3 软件最优发行问题

7.3.1 基本概念

软件测试工程结束，转入用户运用阶段的过程，称为软件产品上市，或软件产品发行(release)。尽管测试工程实施的时间越长，可靠性越高，但是费用也会因此而增加，况且用户要求的交付期是既定的，测试必须在交付期之前结束；另一方面，倘若测试实施的不够充分，发行以后的维护费用异常庞大，这也是软件开发管理的一大忌讳。这就是说，何时结束软件测试，或者说软件何时发行，是一个如何综合考虑软件可靠性、相关费用以及合同交付期诸因素的多目标问题。这个问题也称为软件最优发行问题(Optinal Sofeware Release Problem)。 由运筹学或系统工程的理论与方法可知，软件最优发行问题本质是一个单目标或多目标的最优化问题, 其研究的主线是目标确定,根据此最优化问题的工程经济需要，其研究目标有测试成本目标、测试工期目标和测试可靠性目标等。限于篇幅,以下仅介绍基于可靠性目标或基于测试成本目标的单目标软件最优发行问题和基于成本——工期的双目标最优发行问题。

7.3.2 基于可靠性目标的最优发行问题

注意到对于给定的软件可靠性目标，软件的发行时间取决于软件测试的人力投入，同时软件的可靠性目标又可采用不同的度量指标，如残存差错数或条件可靠度等。故以下讨论将残存差错数作为可靠性目标的最优发行时间求解和以条件可靠度为可靠性目标的最优发行时间求解问题。

1. 以残存差错数为可靠性目标的最优发行问题

一个带有初始潜在固有差错数a的软件，随着测试人力的不断投入和差错的不断被查出并排除，其t时段的残存差错数$r(t)$显然是时间t的单调减函数或t时段查出并排除的累计差错数$m(t)$是时间t的单调增函数(见图7.9)。显然，为使软件可靠性目标(查出并排除的累计差错数)达到a_0(或G_0)时，其最优发行时间(即测试停止时间)由(7.21)式知，由下式决定:

$$m(t)=a(t-\mathrm{e}^{-rW(t)})=a_0$$

由(7.30)式可知最优发行时间T_0为

$$T_0=\left[\frac{1}{\beta}\ln\left(1+\frac{1}{\alpha\gamma}\ln\left(1-\frac{a_0}{a}\right)\right)\right]^{\frac{1}{m}} \tag{7.44}$$

其中，参数a、γ、α、β、m的估计与7.2节相同。

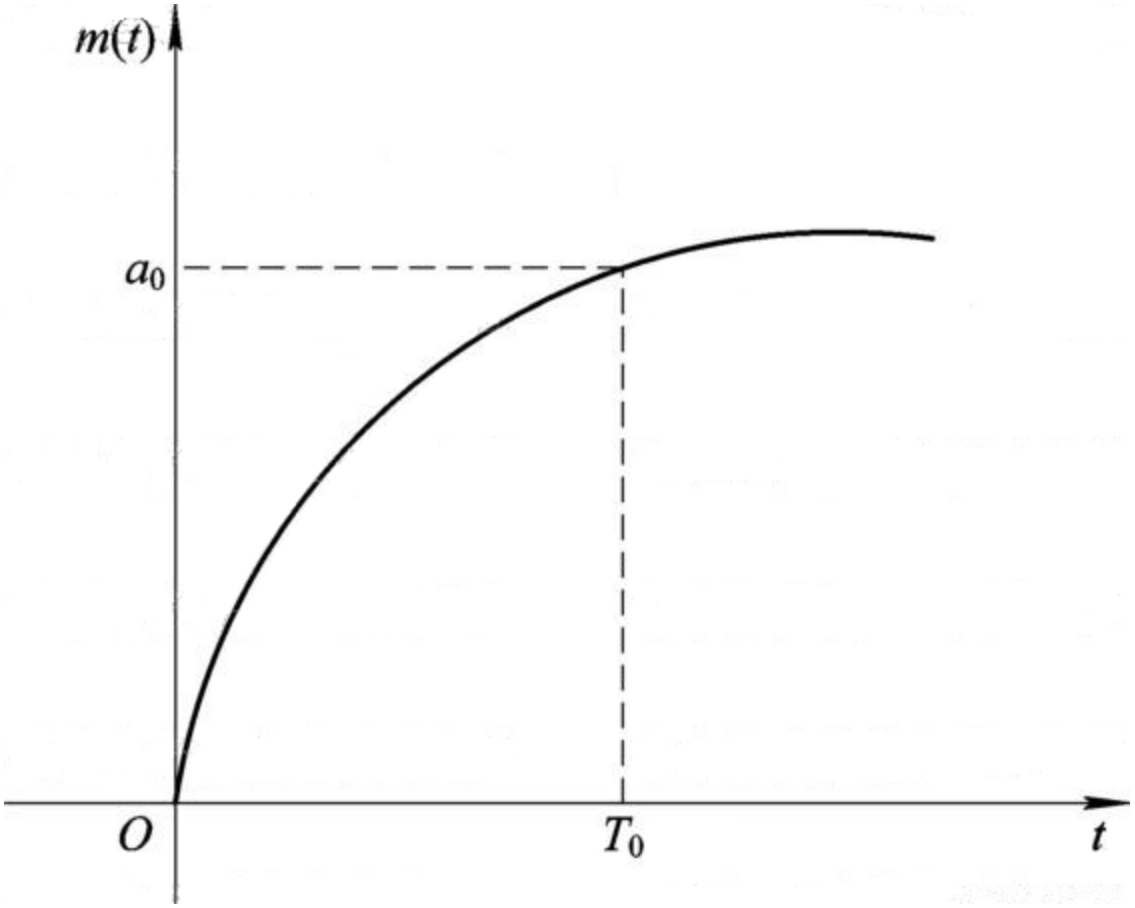

图7.9 软件测试可靠性增长图

2. 以条件可靠度为目标的最优发行问题

注意到随着软件测试人力的不断投入和差错的不断查出与排除，软件的条件可靠度$R(t/T)$同样是时间t的单调增函数，因此，与上同理可知为使软件经测试后达到可靠度目标R_0时其软件的最优发行时间T_1由下式决定:

$$R(t/T)=\exp\{-m(t)\mathrm{e}^{-bpt}\}=R_0$$

由(7.17)式知最优发行时间T_1为

$$T_1=\frac{1}{bp}\left[\ln m(t)-\ln\left(\ln\frac{1}{R_0}\right)\right]$$

其中

$$m(t)=\frac{a}{p}\left(1-\mathrm{e}^{-bpt}\right)$$

参数b、p、a的估计见扩展模型G-O模型一节。

7.3.3 基于费用目标的最优发行问题

通过考虑已达到的可靠性和已花费测试资源两者之间的关系，利用总期望成本这一评价标准，最优软件发行问题可归纳为得到一个最优发行时间以使总期望成本最小的数学规划问题minC(T)(详可见作者文献[22])。其中，把软件测试阶段的测试人力成本和软件在释放前后的改错成本都考虑在成本因素之内，则软件总成本为

$$C(T)=C_1 m(T)+C_2\{m(T_{LC})-m(T)\}+C_3\int_0^T w(t)\,\mathrm{d}t$$

最优释放时间T_*可以通过$C(T)$对总测试时间T求导而算出：

$$\frac{\mathrm{d}C(T)}{\mathrm{d}T}=(C_2-C_1)w(T)\{\frac{C_3}{C_2-C_1}-A(T)\}=0$$

其中利用了关系式

$$\frac{\mathrm{d}\,m(t)}{\mathrm{d}\,t} = w(t)\bullet r\bullet(a - m(t)) = w(t)A(t)$$

且$A(T)$由下式给出：

$$\begin{cases} A(T) \equiv r[a - m(T)] = a\bullet r\bullet\exp[-rW(t)] \\ A(0) = ar, A(\infty) = a\bullet r\bullet\exp[-\alpha r] \end{cases}$$

由上式可解得

$$A(T^*) = \frac{C_3}{C_2 - C_1} \tag{7.45}$$

上式中的$A(T)$是一个关于时间T的严格单调减函数。以下分三种情况来分别讨论。

(1) 如果 $A(0) \le \dfrac{C_3}{C_2 - C_1}$，那么对$0<T<T_{LC}$，有 $A(T) < \dfrac{C_3}{C_2 - C_1}$，从而有 $\dfrac{\mathrm{d}C(T)}{\mathrm{d}T} >$ ，所以$T^*=0$；

(2) 如果 $A(0) > \dfrac{C_3}{C_2 - C_1} > A(T_{LC})$，则对$0<T<T_{LC}$，$A(T)$存在一个有限且唯一的解并满足(7.45)式，且

$$T_0 = \left\{ -\frac{1}{\beta} \ln \left[1 - \frac{\ln(\mathrm{ar}(C_2 - C_1)/C_3)}{\mathrm{ar}} \right] \right\}^{1/m} \tag{7.46}$$

截止时刻T^*为止所投入总测试人力$W(T^*)$就可由下式给出：

$$W(T_0) = -\frac{1}{r} \ln \frac{C_3}{ar(C_2 - C_1)} \tag{7.47}$$

(3) 如果 $A(T_{LC}) > \dfrac{C_3}{C_2 - C_1}$，则$T^*=T_{LC}$。

综上所述，可得决定软件最优发行时间的方案如下：设$C_2>C_1>0$，$C_3>0$，则

(1) 如果 $A(0) \le \dfrac{C_3}{C_2 - C_1}$，则最优发行时间为$T^*=0$；

(2) 如果 $A(0) > \dfrac{C_3}{C_2 - C_1} > A(T_{LC})$，则存在满足(7.46)式的有限唯一的解$T=T_0$，最优发行时间$T^*=T_0$。

(3) 如果 $A(T_{LC}) \ge \dfrac{C_3}{C_2 - C_1}$，则最优发行时间为$T^*=T_{LC}$。

[例7.4] 求解最优发行时间。设C_1=2千元/周，C_2=10千元/周，C_3=0.5千元/周，T_{LC}=100周，NHPP模型参数a=1392.2，r=1.4755×10^{-3}，α=2423.7，β=3.996×10^{-4}，m=2.3026。

解 因为 $A(0) > \dfrac{C_3}{C_2 - C_1} > A(T_{LC})$，可以根据(7.46)式求得最优发行时间

$$T^* = \sqrt[2.3026]{-2502.1 \bullet \ln[0.27963 \bullet \ln 0.030426 + 1]} = 53.1 \text{ 周}$$

7.3.4 基于成本—工期目标的最优发行问题

上面讨论的问题并没有考虑到用户希望的交付期限因素。在软件开发的测试阶段，若追求软件可靠性目标，长时间实施测试，就会造成工期延误；若交付期限超过，还要缴纳逾期罚金。但是若一味只顾交付期限而牺牲软件可靠性，又会造成测试不充分，运行阶段故障多发，软件维护费用激增的恶果。

考虑到软件工期T_s的最优发行问题也可以用NHPP模型中的指数可靠性增长模型来描述，软件工期T_s从软件测试开始时刻算起，测试工程及运行阶段所需要费用总和期望值，即软件总费用期望值$C(T)$可由下式决定：

$$C(T)=C_1m(T)+C_2\left\{m(T_{LC})-m(t)\right\}+C_3W(T)+U_{TS}(T)C_4(T-T_s) \tag{7.48}$$

其中，$U_{Ts}(T)C_4(T-T_s)$为超过交付期限时交付的罚金费用；$UT_s(T)$为如下阶跃函数：

$$U_{TS}(T)=\begin{cases}0 & T\le T_s\\ 1 & T>T_s\end{cases}$$

通过求解(7.48)式为最小值的测试时间$T=T^*$，即可求出最优发行时间T^*。在求解(7.48)式时，当$T\le T_s$时，结论与上一节一致；当$T>T_s$时，对(7.48)式的$C(T)$关于时间求导，可得到是$C(T)$为最小值的必要条件：

$$\frac{\mathrm{d}C(T)}{\mathrm{d}t}=-(C_2-C_1)\omega(T)A(T)+C_3\omega(T)+C_4=0$$

由此可求得

$$A(T)=\frac{C_3\omega(T)+C_4}{\omega(T)(C_2-C_1)}=\frac{C_3}{C_2-C_1}\left[1+\frac{C_4\exp(\beta T^m)}{C_3\alpha\beta T^{m-1}}\right] \tag{7.49}$$

根据$A(T)$的定义易知$A(T)$是关于时间T的严格的单调减函数, 另一方面，又容易证明由(7.49)式右端表述的函数是在 $T_p=\left(\frac{m-1}{m\beta}\right)^{\frac{1}{m}}$ 处有最小值 $\frac{C_3+C_4/\omega(T_p)}{C_2-C_1}$ 的凹函数，故当 $A(0)\le\frac{C_3+C_4/\omega(T_p)}{C_2-C_1}$ 时，在(0，T_{LC}]内，(7.48)式意味着不存在关于T的解，这意味着不经测试就把软件交付给用户。

所以，要想使(0，T_{LC}]区间内存在有限且唯一解$T=T_0$，至少必须满足 $A(0)>\dfrac{C_3+C_4/\omega(T_p)}{C_2-C_1}$ 。一般可以假设时间T_p小于软件工期T_s，当$T_p<T_s$, 并且$\dfrac{C_3\omega(T_s)+C_4}{\omega(T_s)(C_2-C_1)}\geq A(T_s)$，$\dfrac{C_3\omega(T_p)+C_4}{\omega(T_p)(C_2-C_1)}\leq A(T_p)$时，在($T_p$，$T_s$]时间内存在(7.49)式的有限且唯一解$T=T_1$，这时的最优发行时间$T^*$可以根据$T_p$、$T_s$和$T_1$的大小关系而由下式给出：

$$T^*=\begin{cases}T_1 & T_s<T_1\leq T_{LC}\\ T_s & T_p<T_1\leq T_s\end{cases} \tag{7.50}$$

而截止测试时间T投入总测试人力$W(T^*)$可由下式求出：

$$W\left(T^*\right)=a\left\{1-\exp\left[-\beta\left(T^*\right)^m\right]\right\} \tag{7.51}$$

综上所述，软件最优发行时间可归纳为：

(1) 如果 $A(0) \le \dfrac{C_3}{C_2 - C_1}$，则最优发行时间为$T^*=0$；

(2) 如果 $A(0) > \dfrac{C_3}{C_2 - C_1} > A(T_s)$，则最优发行时间为由(7.46)式决定的$T^*=T_0$；

(3) 如果 $A(T_s) > \dfrac{C_3}{C_2 - C_1} \ge A(T_{LC}), \dfrac{C_3\omega(T_s)+C_4}{\omega(T_s)(C_2 - C_1)} \ge A(T_s), \dfrac{C_3\omega(T_p)+C_4}{\omega(T_p)(C_2 - C_1)} \le A(T_p)$，则存在满足(7.49)式的有限且唯一解$T=T_1$，最优发行时间由(7.50)式决定，该式中$T_1$可由解(7.49)式求出。

(4) 如果 $A(T_{\mathrm{LC}}) \geq \dfrac{C_3}{C_2 - C_1} + \dfrac{C_4}{(C_2 - C_1)\omega(T_{\mathrm{LC}})}$，则最优发行时间 $T^{*}=T_{LC}$。

[例7.5] 试求解最优发行时间。其中，设C_1=1, C_2=50, C_3=3，C_4=5，T_{s}=45，T_{LC}=100，求解NHPP模型参数a、r、α、β、m同前例7.4，C_j单位(j=1~4)与T_{s}、T_{LC} 单位同例7.4。

解 经计算有$A(T_s)>\frac{C_3}{C_2-C_1}\geq A(T_{LC})$,并算得$T_p=\left(\frac{m-1}{m\beta}\right)^{\frac{1}{m}}=23.4$，同时验证满足条件$\frac{C_3\omega(T_s)+C_4}{\omega(T_s)(C_2-C_1)}\geq A(T_s),\frac{C_3\omega(T_p)+C_4}{\omega(T_p)(C_2-C_1)}\leq A(T_p)$，则由软件最优发行时间结论(3)知最优发行时间$T^*$可通过由(7.49)式解得的$T_1$与软件工期$T_s$相比较而得到。由于有$T_1$=48.9>$T_s$=45，故有$T^*=T_1$=48.9。其中，解(7.49)式可采用迭代法等数值计算方法。

7.4 软件系统信息库建设

7.4.1 支持信息及其分类

总结前述各章的有关内容，我们看到软件项目的支持信息大致包括如下7类： (1) 软件企业的外部环境信息。它包括在系统规划与投资可行性分析阶段所需要的国内外软件市场需求、用户特征、产品价格、技术发展动向；竞争对手的产品质量、功能、价格、营销策略；国家政策、国家宏观经济发展信息、世界经济发展信息等。

(2) 软件企业的内部条件信息。它包括企业的发展战略、筹资渠道、标书信息、企业生产计划信息、财务信息、人力资源信息、产品的营销渠道与营销策略等。

(3) 软件工程的领域知识和技术支持信息。它包括软件的应用领域如通信、雷达、计算机、航空、航天、机械、微电子、电力、水利、交通等工程领域知识；软件的网络环境所需要的计算机网络、通信设备、传感器、传输设备等硬件的运行、测试与维护知识；软件生产本身所需要的开发工具、系统软件、测试用例；系统分析所需要的应用统计分析知识等。

(4) 软件开发工作中的重要工程经济参数确定所需要的经验时间序列与历史资料的支持信息。它包括软件开发过程中的环境因子E、规模因子a、人力增长率D_0、人员的生产费用率F_c、劳动生产率F_d、软件生产函数$S(K，t_d)$等重要工程经济参数的经验计算公式(或经验值)确定时所需要的各种经验时间序列和历史资料信息，如已完成软件项目的程序规模、工作量、成本、工期、累计投入的人力费用、各时段投入的人力密度、峰值时刻、开发难度、项目效益、效果等的经验时间序列和历史资料信息等。

(5) 软件测试尤其是可靠性测试过程中为确定可靠性增长模型的几个待定参数而需要的测试排错时间序列、测试人力投入时间序列等信息。

(6) 为建立适合于各软件机构的最佳工作分解结构WBS和进度计划所需要的已完成项目的BWS和各阶段/活动的工作量、成本、进度的分配比例以及项目团队成员、组织结构、开发平台、工具、过程等属性信息。

(7) 各种工程经济参数之间的交叉影响分析和风险分析与控制的分析及报告信息。这样的交叉影响分析有可靠性、复杂性等软件属性对软件成本、工期估计的影响分析，人力资源属性对软件成本、进度估计的影响分析等；软件项目风险分析与控制的分析及报告信息有该分析中需要的项目风险树、风险因子鱼骨图、风险控制策略等有关信息。

7.4.2 软件信息库建设

综观上述软件支持信息及其分类可知：软件的支持信息范围广泛，内容众多。它既包括了工程技术知识的有关信息，又包括了工程经济信息的有关内容；在工程经济信息中既包括了国家乃至世界宏观经济、产业经济、软件市场的有关信息，又包括了企业经济的有关信息；在工程技术信息中，既包括了各种软件应用领域的知识与信息，又包括了软件开发环境所涉及的网络环境等硬件知识与信息；在企业经济信息中既包括了软件项目开发与使用的有关工程经济信息，又包括了软件投标、融资、产品营销等有关工程经济信息；在软件项目开发与使用信息中既包括了软件规模、成本、工作量、效益、效果等物质属性有关信息，又包括人力资源投入、团队组织、进度控制等人员属性有关信息。因此软件信息库的建设是一项长期而艰巨的系统工程。为保障上述任务的完成，作者认为需要做好下述工作。

(1) 建立软件信息库的专门管理机构和行之有效的信息采集机制。这样便可通过信息库的专门管理机构来组织和调动整个软件企业的生产、销售、财务、人力资源管理等部门的管理和技术人员的积极参与来完成有关上述各类信息的采集任务。并由专门管理机构的人员来完成信息的存储、维护与更新工作。

(2) 从技术设计的角度来看，软件信息库应成为企业信息系统中的一个组成部分，或称为企业信息系统下属数据库系统的一个重要子库，通过网络通信及有关的信息采集机制来完成企业各信息子系统(如财务管理子系统、营销管理子系统等)中相关信息的采集任务。

(3) 开展经常性的信息处理与研究是信息库建设过程中最重要的工作。应有专人或专门小组负责研究各类重要的工程技术参数、工程经济参数的数学建模，假设检验，参数估计，相关分析，回归分析等统计分析工作以及与国外相关参数的比较工作。这样便可经过多年的信息采集、存储与分析研究，建立一套适合于我国国情或开发机构实际环境的工程技术参数体系和工程经济参数体系，以便为我国今后开展软件项目的工程经济分析奠定基础，同时也为企业的重大决策(如投资决策、设计方案决策、营销决策、成本决策、人力资源选择与团队组织决策、供应链决策等)提供强有力的信息支持和辅助决策支持。

(4) 注意到软件信息库的建设是一项需要企业内部人人参与的人—机工程，尤其是企业高层主管的大力支持与组织协调。因此，加强软件信息库建设重要性的宣传，积极争取企业领导的大力支持和广大管理与技术人员的积极参与是十分重要的。此外，考虑到信息处理是一项需要多学科领域知识(如系统工程、运筹学、应用统计学、宏观经济学、管理经济学、技术经济学、软件可靠性、数据挖掘与联机分析等)支持的较为复杂的研究工作，因此重视信息处理骨干人才的培养和开展经常性的知识与技能培训也是重要的管理工作。

(5) 软件工程或软件企业的行业协会应有专人负责国内各软件开发机构信息库建设的组织、协调与技术指导，以及国内外的学术交流与人才培养工作。

作 业

5

习 题 七

1. 简述软件测试的类别与测试过程(步骤)，并说明软件测试对软件可靠性的影响及其时间特性。

2. 软件可靠性评价指标有哪些? 说明它们之间的数量关系。

3. 什么是软件可靠性增长模型? 以G-O模型为例说明可靠性增长模型的功能与作用。

4. 如何求解扩展G-O模型的待定参数a、b、p? 写出其求解步骤与相关算式。

5. 某证券投资分析软件经250小时测试获得26个差错，同时获得了在排错过程中的差错查出(同时修正)时间t_j和对应的差错查出(同时修正)累计数d_j的时间序列{(t_j，d_j)，j=1，2，…，26}，通过上述数据及相关代数方程的迭代计算，获得参数估计值 $\hat{b}$ =0.005 79。试利用G-O模型求解此证券投资分析软件的如下可靠性指标：

(1) 期望累计差错函数$m(t)$和差错查出率函数$\lambda(t)$；

(2) 该软件经250小时测试后的期望差错数$a-m(t)$；

(3) 欲使该软件目标可靠度R_0=0.98，问尚需继续测试与排错多少时间。

6. 某软件在可靠性测试前已埋设播种差错数M_c=28，经测试查出播种差错数x_c=24，潜在固有差错数n_v=30，并在测试过程中记录下播种差错查出时间序列$\{S_1, S_2, \ldots, S_{24}\}$和潜在固有差错查出时间序列，并利用上述两个时间序列数据和相关方程求解得到能查出的期望累计播种差错数$\{\tilde{S}_1, \tilde{S}_2, \cdots, \tilde{S}_{30}\}$和期望累计潜在固有差错数$\hat{a}_v = 31.6$。试求解该软件的如下测试与可靠性指标：

(1) 测试精度A_c，测试覆盖率C_v及软件质量水平SPOL，并据此评价该软件测试的有效性。

(2) 软件的潜在固有差错数N_0和该软件经可靠性测试后的残存固有差错数。

7. 基于威布尔(Weibull)分布曲线的测试人力投入曲线描述了什么样的变量依赖关系? 威布尔分布曲线与负指数分布曲线、瑞利分布曲线有何联系? 为什么说威布尔分布曲线比后两个分布曲线所反映的变量依赖关系更为广泛? 对于一个软件的测试人力投入过程，如何来求解其基于威布尔分布曲线的测试人力投入曲线的待定参数α、β、m?

8. 某软件经测试一段时间后，经统计分析与计算，已获得了其基于威布尔分布曲线的测试人力投入待定参数α、β、m或$W(t)$，下面如何来求解其可靠性增长模型$m(t)=a[1-\exp(r\cdot W(t)]$中的待定参数$a$与$r$?

9. 某软件经过一段可靠性测试后，通过统计分析与计算，获知在软件测试过程中，已发现的软件差错能完全修正的概率p=0.1，经无限测试最终能查出的期望差错数a=42，查出率参数b=0.06。试求为使该软件可靠度达到R_0=0.985以上时的软件最优发行时间。

10. 某软件经过一段可靠性测试后，通过统计分析与计算，获知其测试人力投入函数有$W(t)=\alpha[1-\exp(-\beta t^m)]$, 其中, α=2800, $\beta=4\times10^{-4}$。此外，还获得该软件的潜在固有差错总数a=1500，投入单位测试人力的残存差错发现率$r=1.5\times10^{-3}$。试求在测试阶段差错修改单位成本C_1=2.5千元，运行阶段差错修改单位成本C_2=10千元条件下的软件最优发行时间。

11. 在软件测试中，静态资源分配与动态资源分配有何区别? 动态资源分配计算的设计思想与计算步骤如何?

12. 在对软件，特别是中、大型软件作工程经济分析时，需要哪些类别的支持信息? 这样类别的支持信息能支持求解软件工程经济分析中的哪些问题? 为建设一个符合我国国情并科学、有效的软件企业信息库与软件行业信息库，你认为应如何来进行组织? 应采取哪些对策与措施?