摘要

随着软件产品在整个社会中的作用越来越重要，公司在生产和交付软件产品及服务上的竞争进一步加剧，使得在短时间内有效地开发出高质量的软件成为必须。其中，测试是保证软件质量最重要和最常用的手段。对测试工作量进行合理准确的估算是制定测试计划并顺利完成测试工作的重要前提。一个好的测试工作量估算模型可以帮助测试经理更加合理地计划和安排测试进度与资源，以此来进一步提高机构软件产品的竞争力。而事实上，到目前为止对软件测试执行工作量估算的研究非常匮乏，并且这些模型通常具有时间开销较大，主观性较强等缺陷。

基于此，本文首先对现有常用的软件估算模型和方法进行了全面而详细的研究，并对当前仅有的一些软件测试执行工作量估算方法进行了重点讨论。然后，在此基础上，本文提出了一个基于经验的软件测试执行工作量估算模型 STEEM。本模型将测试套件特征化为一个包含了测试用例数量，测试执行复杂度和测试人员熟练程度的三维矢量，即测试套件执行矢量，并定义了关于这三个矢量的度量标准，然后建立经验数据库。基于回归分析的思想，模型通过收集机构项目的经验数据来分析和研究测试套间矢量中的三个参数与执行完该测试套件所需要的工作量时间之间的关系，并使用机器学习方法对该关系进行建模，以此来实现对执行给定的测试套件所需要工作量的估算。本文在某金融软件公司内部使用该模型进行了实例分析，以此来评估该模型，根据公司的实际情况配置测试套件执行矢量的三个元素，并依此来收集公司内部项目的历史数据，然后利用支持向量机方法进行训练得到最终的模型，取得了较好的估算效果。并在该公司中某具体项目的实际测试周期中使用该模型，从而帮助了测试经理更好地完成制定测试计划和资源分配等测试管理工作。

关键词： 软件测试，工作量估算，机器学习，测试套件执行矢量

Abstract

As the role of software products becomes more and more important in the whole society, jointly with the competition on producing and delivering software products and services increases, it is very necessary to produce low cost high quality software in a short period. Software testing is one of the most important ways in the quality assurance. Making the reasonable and accurate estimation on software test execution effort is the crucial precondition of designing software plan and finishing testing successfully. A good software test execution effort estimation model can help test managers plan their schedules and resources more properly and much easier, and then improve the quality of their software products. But actually, the research on software test execution effort estimation is very short up to now, and also the existing models have some severe disadvantages, e.g. time consuming, strong subjectivity and so on.

Based on this, first, this paper fully introduces the existing software estimation methods, and also discusses the existing software test execution effort estimation models in detail. Then this paper proposes an experience-based model for software test execution estimation effort. In this model, we characterize a test suite as a 3-dimensions vector called test suite execution vector which includes test case number, test execution complexity and its tester rank. Based on the definition and measurements on test suite execution vector, we set up an experience database. According to Regression Analysis concept, we use machine learning algorithm to train the experience data sets to find the relationship between test suite execution vector and test execution effort, and then estimate efforts for a given test suite execution vector. We evaluate this model through an empirical study in a financial software company, and get a good estimation result. Then we use this model in a real QA cycle for a project in this company, which helps test managers finish software test management work much better, such as designing test plan, allocating test resources and so on.

Keywords: software testing, effort estimation, machine learning, test suite

图目录

图 1.1 Standish 组织对软件项目完成情况的统计数据 2
图 2.1 软件测试工作流程图 8
图 2.2 功能点复杂度权重对应表 10
图 2.3 “订单处理系统”系统级用例图 15
图 2.4 “订单处理系统”涵盖的一部分事件流 16
图 2.5 系统的技术复杂度因子及权重分布 18
图 2.6 项目组的环境复杂度因子及权重分布 18
图 2.7 神经网络示例图 21
图 2.8 V-模型 23
图 2.9 估算执行一组测试套件的工作量 26
图 2.10 为某个测试用例计算 EP 数 27
图 2.11 使用 EP 和 CF 来计算测试执行工作量 28
图 3.1 测试套件结构图 30
图 3.2 STEEM 模型整体结构图 32
图 3.3 一个典型的测试组成结构图 34
图 3.4 度量测试套件测试执行复杂度的影响因素及衡量标准 35
图 3.5 测试人员熟练程度度量图 37
图 3.6 STEEM 模型估算流程图 39
图 3.7 STEEM 模型建立过程图 41
图 4.1 影响测试效率因素分类图 43
图 4.2 测试执行复杂度问卷 46
图 4.3 WEKA 输入文件 50
图 4.4 WEKA Explorer - Preprocess 51
图 4.5 WEKA 测试方式选择 52
图 4.6 WEKA 训练结果 53
图 5.1 FIXRouting 系统功能图 56
图 5.2 FIXRouting 系统结构图 57
图 5.3 TT Upgrade QA Cycle 各测试套件执行复杂度 60
图 5.4 TT Upgrade QA Cycle WBS 图 63

表目录

表 2.1 功能点分析法中的功能类型定义及示例 10
表 2.2 基本模型中不同软件项目类型对应的系数表 12
表 2.3 中等模型所涵盖的四种开销驱动及子属性 13
表 2.4 中等模型中不同项目类型对应的系数表 14
表 2.5 Actor 各类型权重分布 16
表 2.6 基于事务的用例权重分布 17
表 2.7 UCP 在测试工作量估算中的 Actor 类型权重分布 24
表 2.8 UCP 方法中对于影响测试的技术和环境因素列表 24
表 4.1 测试输入各影响因素 44
表 4.2 系统处理过程各影响因素 44
表 4.3 输出检查各影响因素 0 45
表 4.4 测试环境各影响因素 45
表 4.5 测试执行复杂度问卷调查统计结果 47
表 4.6 测试执行复杂度度量 Guideline 48
表 4.7 STEEM 模型实验结果分析表 55
表 5.1 TT GUI Regression 测试工作量估算值 60
表 5.2 RTDM 测试工作量估算值 61
表 5.3 OM 测试工作量估算值 61
表 5.4 BV 测试工作量估算值 61
表 5.5 AIM 测试工作量估算值 61
表 5.6 FX over FIX NG 测试工作量估算值 62
表 5.7 FX over FIX Legacy 测试工作量估算值 62
表 5.8 测试工作分配结果 62

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得浙江大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：徐新　　签字日期：2008年6月6日

学位论文版权使用授权书

本学位论文作者完全了解浙江大学有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权浙江大学可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：徐新　　导师签名：

签字日期：2008年6月6日　　签字日期：2008年6月6日

第1章 绪论

1.1 研究意义

计算机软件和硬件持续不断地发展，加上软件产品在整个社会中的作用越来越重要使得公司在生产和交付软件产品及服务上的竞争进一步加剧，对短时间内开发出低成本高质量的软件的需求变得越来越迫切。在这些竞争性的市场，例如金融软件系统市场，出品低质量产品的公司会很快地失去他们的客户。因此，公司应该确保产品同客户的期望一致。其中，测试是保证软件质量最重要和最常用的手段。软件测试涵盖了大量的活动来确保最终的产品达到一定的质量标准。

软件测试是人们利用自己所能想到的方法，来尝试发现软件产品中可能存在的错误和不足的过程，其目标是要保证被测试的软件产品的功能与需求相符，达到客户的需求。软件测试的重要性使得公司一般都会安排专门的组和人员来进行测试活动。有报道显示，在所调查的 65 家软件机构中，44 家都有独立的软件测试组[1]。对于这些测试组来说，如何能够对测试工作进行更好的计划和安排，对测试的顺利完成起到了举足轻重的作用。而对测试工作量进行合理和准确的估算是安排测试进度和分配测试资源的重要前提。

1.1.1 工作量估算

制造业和软件业最大的不同可能在于，前者大部分时间都能符合时间表的进度安排，而后者则不然。在 1995 年，Standish 组织对 8000 多个软件项目进行了调查，发现这些项目中只有 16.2%是成功的，即没有超过预算和最后完成期限；有 31.1%是失败的，即项目被取消或者未完成；有 52.7%称为被质疑的，即虽然完成但是超出了计划和预算，而其中平均超过计划预算的 90%，超过时间表的 222%，所有完成的项目当中有 50%多都只实现了最初需求的不到 50%[2]。之后在 2004 年，该组织的统计调查项目扩充到了 50，000 多，结果表明有 29%的项目是成功的，而仍有 53%的项目属于被质疑的范围[3]。具体如图 1.1 所示。

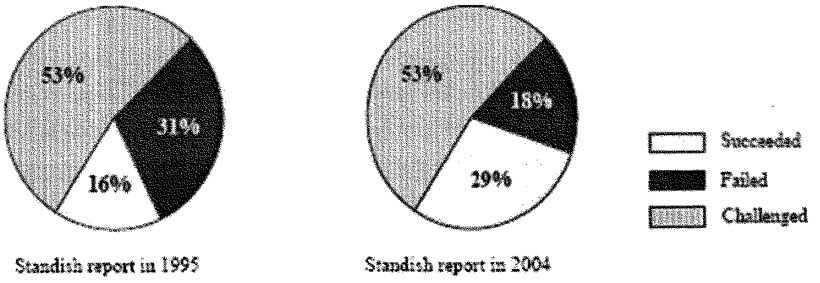

图 1.1 Standish 组织对软件项目完成情况的统计数据[2,3]

软件工作量估算是对开发软件产品所需的人力的估算，是用来确定项目开发时间和开发成本的过程[4]。这是任何软件项目所共有的主要成本。通常以人小时、人天、人月、人年的形式来衡量，并且有转换系数在不同单位之间进行转换。工作量估算是由软件规模和与项目有关的因素所驱动的，如团队的技术和能力、团队的稳定性、所使用的语言和平台、平台的可用性与适用性、项目中的自动化程度等。

对软件项目进行工作量估算历来是比较复杂的事情，因为软件本身的复杂性、历史经验的缺乏、估算模型和方法的缺乏以及一些人为错误等，导致软件项目的估算往往和实际情况相差甚远。可以说，对软件工作量估算的不足是造成这种糟糕局面，即 53%以上的软件项目远远超出计划的重要原因之一。

1.1.2 测试工作量估算重要性

众所周知，软件测试占了软件开发总工作量中的相当大的一部分比例。测试执行工作量估算是对完成测试软件产品的人力和时间开销的估算，通常以人小时为单位。

在大型的软件公司中，测试组的工作总是超负荷的。在这种情况下，测试经理就必须能够合理地计划和安排测试进度和资源，必须能够估算出执行一定量的测试所需要的工作量，从而能够据此来申请更多的资源或者延长最终的完成期限等。同时对于已处在维护期的软件项目，每一次版本升级都需要进行一定量的回归测试，而在实际中测试所能有的开销都是有限的，因此对所要执行的测试工作

量的估算也是回归测试范围选择的一个额外标准。

而在大部分的公司中，测试执行工作量很少被好好地计划，使得测试执行时间经常超出之前安排的进度，有时候甚至到了交付的最终期限还没有完成测试，从而导致没有被充分测试过的产品进入市场，产生了大量的客户负面反应，破坏了公司形象等。为了避免这些，测试经理必须能够较精确的估算出软件测试执行的工作量，并以此来制定测试计划和安排测试资源。但是，对于之前从没有执行过的测试用例来说，如何能够较精确的估算其工作量，这对于测试经理来说仍然是一个很大的挑战。

针对软件测试工作量估算的重要性，我们非常有必要寻找一种更好的方法或模型来解决当前在估算中存在的种种问题，以此来进一步地提高机构软件产品的竞争力。

1.1.3 估算方法研究现状

软件项目需要进行详细的计划，这包括进行一系列的活动，而其中对工作量进行准确估算是最基本和最重要的，因为估算为其他的活动提供了方向。而根据2001年的一份最新研究表明，只有45.7%的项目完成了软件工作量估算[5]。虽然并没有一个具体的研究来分析进行软件估算率低的原因，但是缺少可靠的模型来进行估算应该是其中一个重要的影响因素。

在早期，瀑布模型多应用于软件开发中，在该模型中，软件测试仅被当作整个开发生命周期中的一个非常小的阶段。在过去的几十年中，已经产生了几种模型和方法用来估算软件开发的工作量，复杂度和开销，但是由于软件开发工作量估算模型是基于软件的规模和开发复杂度的，而非是测试规模和测试执行复杂度，所以并不适合来估算执行一定量测试用例的工作量开销。事实上，对于占软件项目总工作量 40%以上的软件测试工作[6]来说，其估算方法和模型却经常被忽略掉，到现在为止对软件测试执行工作量估算的研究也只有很少的几个，并且这些模型通常具有时间开销较大，主观性较强等缺陷。

1.2 研究工作

基于软件测试对于出品高质量软件产品的重要性及软件测试执行工作量估算模型匮乏等现状，本文首先对现有常用的软件估算模型和方法进行了详细而全面的研究，介绍了功能点分析法，COCOMO 模型，用例点分析法，回归分析和机器学习等估算方法的基本概念和原理，并对当前已有的一些专门针对软件测试执行工作量的估算方法进行了重点讨论。

在此基础上，本文提出了一个基于经验的软件功能性测试执行工作量估算模型，该模型引入了测试套件执行矢量的概念，涵盖了测试用例数量，测试执行复杂度及测试人员熟练程度等三个对测试执行效率影响巨大的元素，并定义了关于这三个元素的度量标准。基于回归分析的思想，模型通过收集机构项目的经验数据来分析和研究测试套间执行矢量中的三个参数与执行完该测试套件所需要的工作量时间（以 man-hour 为单位）之间的关系，并使用机器学习中的支持向量机方法对该关系进行建模，以期达到较好的估算效果。

本文在某金融软件公司内部使用该模型进行了实例分析，根据公司实际的情况配置测试套件执行矢量的三个元素，并以此来收集公司内部多个项目的历史数据，然后利用支持向量机方法进行训练得到最终的模型。并在该公司中某具体项目的实际测试周期中使用该模型，帮助了测试经理更好地完成制定测试计划和资源分配等测试管理工作，从而提高软件产品的质量，达到了较好的效果。

估算其实就是根据过去预测未来的过程，只有以可靠准确的历史经验数据作为基础，推出来的估算模型才有较好的说服力，该模型的其他使用者可以采用更符合本机构或者本项目的历史经验数据集，或者重新校准该模型的某些参数、某些定义及具体的度量标准，从而得到更符合本机构的模型公式。对可知的信息把握得越充分，了解得越透彻，对未知的估算才有可能更加准确和可靠。

1.3 本文组织结构

根据以上介绍的研究意义及研究工作，本文其他部分内容组织如下：

第2章主要分三部分，首先对软件测试的相关理论进行了简单介绍，然后详细阐述了现有常用的一些估算方法的基本概念和原理，最后对现有的专门针对软件测试执行工作量的估算模型进行了研究和分析，为之后建立估算模型做铺垫。

第3章是基于经验的软件测试执行工作量估算模型建立部分，分析了模型建立所遵循的设计原则和基本原理，并详细阐述了测试套件执行矢量中三个元素的定义及度量标准，最后介绍了模型建立的过程和步骤。

第4章以某金融软件公司为背景对本模型做了实例研究，结合该公司特点介绍了如何配置测试套件执行矢量，以及如何收集数据和进行训练，并对最终得到的模型进行了估算精确度分析。

第5章主要介绍了如何在某项目的实际测试周期中使用该模型的估算结果来完成测试计划和资源分配等工作。

第6章对全文内容进行了总结，回顾了本文的主要研究内容，对文中涉及到的重要问题做了进一步的讨论，并指出需要在未来进行研究的内容，作为下个阶段研究的重点。

第2章 软件测试相关理论和估算常用方法研究

2.1 软件测试知识概述

在实际的软件开发过程当中，软件测试对于整个项目的成功与否非常的重要，并且对软件的可靠性也起到了关键性的影响作用。在开发大型软件系统的漫长过程中，人们面对着极端错综复杂的问题和状况，主观认识不可能完全符合客观现实，与软件工程各方面工作密切相关的人员之间的沟通和配合也不可能完美无缺。因此，在软件生命周期的各个阶段都不可避免地会产生一些差错。即使我们尽量在每个阶段结束之前都进行严格的技术审查，尽可能早地发现并纠正差错，但是，经验表明审查并不能发现所有差错，此外在编码过程中还会不可避免地引入新的错误[7]。如果在软件投入生产运行之前，我们没有及时发现并纠正软件中的大部分差错，则这些差错早晚会在产品运行过程中暴露出来，那时不仅改正这些差错的代价更高，而且往往会造成很恶劣的后果和影响。

大量统计资料表明，软件测试的工作量往往占软件开发总工作量的40%以上，在某些极端情况下，测试那种关系人的生命安全的软件，测试所花的成本可能相当于软件工程其他步骤总成本的三倍到五倍[8]。因此，我们必须高度重视软件测试工作，绝不能以为写出程序之后软件开发工作就算差不多完成了，实际上，大约还有同样多的开发工作量需要完成，也就是软件测试工作。

2.1.1 软件测试

首先要清楚什么是测试？软件测试是为了寻找错误而执行一个程序或系统的过程[9]，通过评价一个程序或系统的属性和性能来确定其是否满足所期望的结果[10]。

软件测试的目标既不是要找到软件中所有的错误，或者找出某一代码段里存在具体错误的数量，也不是要证明某个软件的设计是否正确，而是要保证被测试的软件产品的功能与需求相符，达到客户的需求。测试包括检查（Verification）

和验证（Validation）两个方面。检查用于评估软件产品的计划、文档、代码、说明书等内容中是否存在错误或者矛盾，验证用于检验软件是否满足软件说明书的要求[11]。

从传统意义上来说，软件测试可以从测试工程师设计测试用例时的角度来划分为：黑盒测试和白盒测试。黑盒测试不考虑软件的内部行为，目标是根据需求来测试软件的功能[12]；而白盒测试则要求测试人员必须理解软件内部的数据结构，代码和算法等，必须包含满足某些代码覆盖率要求的测试。

此外，软件测试还可以按照层次关系划分成三种不同的测试：测试软件功能模块内部代码的单元测试；测试功能模块之间联系的集成测试；针对整个软件产品整体功能的系统测试。

还可以按照不同的要求划分成许多具体的测试类型。例如可以根据软件具体的功能要求，进行功能测试；也可以根据软件需要满足的性能需求，来进行性能测试等。

2.1.2 软件功能测试

软件功能测试（Software Functional Testing）是系统测试中最基本的测试，与性能测试和压力测试等其他测试截然不同，功能测试与软件内部的实现逻辑无关，主要根据产品的需求规格说明书和测试需求列表，验证产品的功能实现是否符合产品的需求规格，属于黑盒测试。

功能测试主要是为了发现以下几类错误[13]：

- 是否有不正确或者是遗漏了的功能；
- 功能实现是否满足用户需求和系统设计的隐藏需求；
- 能否正确地接受输入？能否正确地输出结果？

软件功能测试要求测试用例设计者非常熟悉产品的需求文档，需求规格说明书，产品业务逻辑等，同时也要对测试用例的设计方法有一定的了解和掌握，才能设计出好的测试方案、测试套件和测试用例等，高效地进行功能测试。

2.1.3 软件测试流程和管理

尽管软件测试在不同的机构中都会有所不同，但是一般来讲，一个完整的软件测试工作流程大都包括以下六个阶段，如图 2.1 所示：

- 需求分析阶段：在软件开发生命周期的需求分析阶段就应该开始测试相关活动，在设计阶段，测试人员应该同开发人员一起来决定哪些设计是可测试的；
- 测试计划阶段：制定测试计划的目的是依据需求分析的结果来确定测试活动的范围、方法、资源和进度，明确正在测试的项目要测试的特性，要执行的测试任务，任务的责任人以及与计划相关的风险[14]；
- 设计测试用例阶段：测试需求收集完成之后，开始测试用例设计；
- 执行测试阶段：测试人员基于测试计划和测试用例来执行软件，将发现的任何错误都告知开发组；
- 出具测试分析报告阶段：一旦测试完成，测试人员随即开始编写最终的测试报告，分析所测软件是否可以投入使用；
- 缺陷再测试阶段：测试人员将对开发组已修复的缺陷进行再测试以保证其满足客户的需求。并不是发现的所有错误或缺陷都必须被开发组修复。一些可能由于错误的配置测试软件，一些可能在真实产品环境下可以被处理，其他的可能会放在未来的软件版本中再进行修复。

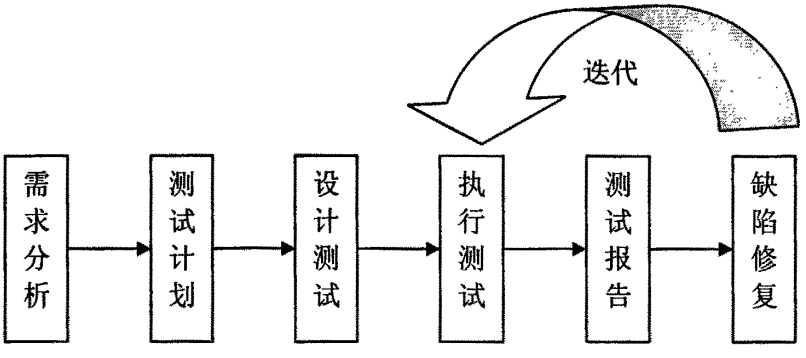

图 2.1 软件测试工作流程图

2.2 工作量估算常用方法研究

估算，即确定项目开发时间和开发成本的过程。在过去的几十年中，产生了几种模型和方法用来估算软件的规模，复杂度，工作量和开销。在一些研究的调查实验[15] [16] [17]中总结了迄今为止的软件估算方法，本文详细研究了其中一些比较著名的软件估算方法的基本概念和原理，包括：功能点分析法，COCOMO 模型，用例点分析法，回归分析和机器学习等方法，以下五节将分别对其进行介绍和分析。

2.2.1 功能点分析法

上个世纪 70 年代，IBM 公司试图开发一种独立于语言的对软件开发工作量进行估算的方法，即功能点分析法（Function Point Analysis，简称为 FPA）[18]。

功能点分析法从功能性的或者用户的角度进行度量，并独立于计算机语言，开发方法，技术平台和项目组的能力等因素。与代码行数（Line of Code，简称为 LOC）方法从纯开发的角度出发进行估算截然不同。该方法最初被用来度量计算机应用软件及开发该软件的项目的规模，连同关于功能点计数的规则，其已被多次修改并优化过。

功能点分析法定义了五种基本的功能类型来估算软件的规模，两种数据功能类型：内部逻辑文件（ILF）和外部接口文件（EIF），三种事务功能类型：外部输入（EI），外部输出（EO）和外部查询（EQ）[18]。表 2.1 详细介绍了各个功能类型的定义。功能点分析法现在被国际功能点用户组（International Function Point Users Group，简称 IFPUG）进行维护[19]，该组织提出了这 5 种基本类型功能点的统计规则，提供对功能点计算人员的培训，并且根据引入的新开发语言，不断更新转换表。

一般来说，计算功能点数目的步骤如下：

1、计算输入，输出，查询，内部逻辑文件，和外部接口文件等五种基本类型的数目；

2、将这些数据进行加权乘。图 2.2 为一个典型的功能点复杂度权值表；

3、估算者根据对复杂度的判断，总数可以进行一定的调整。

表 2.1 功能点分析法中的功能类型定义及示例

功能类型	定义	示例
内部逻辑文件(ILF)	重要的数据结构或内部文件	一个系统中出最终用户来维护的数据逻辑分组
外部接口文件(EIF)	重要的配置数据或文件	其他系统中的仅被用来做引用的数据分组
外部输入（EI）	针对内部逻辑文件（ILF 的界面，对话框等	使用户可以通过添加，修改和删除内容来维护 ILF 中的数据
外部输出（EO）	输出	表报或某些视图等
外部接口文件(EIF)	用户输入选择信息来获取满足具体标准的数据	字数多少，查询结果或关于软件的状态问题等

Measured parameter	simple	moderate	hard	total
Number of user inputs	3×□+	4×□+	6×□	=
Number of user outputs	4×□+	5×□+	7×□	=
Number of user queries	3×□+	4×□+	6×□	=
Number of files	7×□+	10×□+	15×□	=
Number of outer interfaces	5×□+	7×□+	10×□	=
			totalling	

图 2.2 功能点复杂度权重对应表

到现在为止，不断有基于 Albrecht 的功能点分析模型的新的 FPA 版本发表，包括 IFPUG 方法[20]，MarkII[21]，COSMIC-FFP[22]等。

对于一个软件产品的开发，功能点对项目早期的规模估算比较有帮助，但是

对于软件测试工作量的估算来说，并非像开发工作量那样直观。此外，在软件工程界，功能点分析模型也一直备受争议，计算功能点数目需要繁重的工作量，相比其对于开销估算所带来的精确性并没有太多的优势，也没有考虑到影响软件项目复杂度的其他很多重要因素，比如需求变化，资源限制和优先级重新定义等[23]。

2.2.2 COCOMO 模型

1981 年 Barry J. Boehm 在《Software engineering economics》一书中首次发表了构造性成本模型（Constructive Cost Model，简称为 COCOMO）作为用来估算软件项目工作量，开销和进度的模型。该模型基于 63 个项目的研究，项目大小从 2000 到 100,000 代码行不等。这些项目的软件开发过程都是以当时非常流行的瀑布模型为主的[24]。该模型之后被称为 COCOMO81。随着 Barry J. Boehm 等人进一步的研究，1997 年开发出了 COCOMO II，并于 2001 年在《Software Cost Estimation with COCOMO II》一书中发表了该模型。COCOMO II 对 COCOMO81 做了彻底的更新，更加适合于估算现代软件开发项目。在原先的项目数据库上进行了扩充，对现代软件开发过程也支持的更好[25]。软件开发技术从大型机和批处理发展为桌面开发，代码的可重用性和使用非定制的软件模块等都使得这个新模型出现成为必然。

无论是 COCOMO81 还是 COCOMO II，都是将功能点 FP 和源代码行数 SLOC 等转换为所开发系统的工作量估计。公式利用了工作量乘以比例因数，该比例因数根据开发环境的特征，项目组和项目开发流程等来定义[26]。两者的基本原理都是：将软件开发所需工作量表示为软件规模和一系列成本因子的函数。

通用的 COCOMO 算法模型[24]如下：

$$PM = A * S^{E} * \prod_{i=1}^{n} EM_i \tag{2.1}$$

其中，

PM：Person Months，以人月衡量的工作量；

A：从历史项目数据推导出的常量；

S：源代码千行数（KSLOC），也可从功能点或者对象点转换得到；

E：对预期工作量存在指数或者非线性影响的比例因子的总和；

EM：成本驱动因子，涵盖产品、人员、平台、项目等方面。

接下来分别来对 COCOMO81 和 COCOMO II 进行简单的介绍。

2.2.2.1 COCOMO81

COCOMO81 包含了三个层次，其详细度和精确度呈递增形式发展。该模型中，软件项目的特征由开发模式和成本驱动因子来表现，开发模式确定了工作量估算公式中的系数 A 和指数 E，成本驱动因子的影响表现为工作量乘数 EM[27]。

1. 基本模型（Basic COCOMO）

第一个层次，基本模型（Basic COCOMO）对软件开销可以进行快速早期粗略的数量级估算，但是由于缺少对不同的项目属性进行考虑，像硬件限制，个人能力和经验，现代工具和技术的使用等对软件开销影响巨大的因子（成本驱动），其精确度是非常有限的。基本模型可以应用于三种类型的软件项目，分别为：组织型，半组织型和嵌入型。

基本模型的公式如下：

$$E = a_b * (KLOC)^{b_b} \tag{2.2}$$

其中，

E：以人月为单位的工作量；

KLOC：估算的项目需要交付的千行代码数；

系数 a_b, b_b 见下表：

表 2.2 基本模型中不同软件项目类型对应的系数表

软件项目类型	a_b	b_b
组织型	2.4	1.05
半组织型	3.0	1.12
嵌入型	3.6	1.20

2. 中等模型（Intermediate COCOMO）

中等模型（Intermediate COCOMO）将软件开发工作量表示为软件规模与一组开销驱动因子的函数，包括产品的主观评估，硬件，人员和项目属性等四种开销驱动，每种都涵盖了一些子属性。具体见下表：

表 2.3 中等模型所涵盖的四种开销驱动及子属性

产品属性	要求的软件可靠性
	数据库规模
	产品复杂性
硬件属性	执行时间约束
	主存储器约束
	虚拟机的易变形
	计算机周转时间
人员属性	分析员能力
	程序员能力
	应用经验
	虚拟机经验
	编程语言经验
项目属性	现代编程规范
	软件工具的使用
	要求的开发进度

上述的 15 个属性，每个都分为了 6 个等级，每个等级都被赋予了一个工作量乘数，用以体现其对项目工作量的影响程度。所有工作量乘数的乘积即为工作量调整因子（Effort Adjustment Factor，简称为 EAF），值一般在 0.9-1.4 之间。

中等模型的公式如下：

$$E = a_i * (KLOC)^{b_i} * EAF \tag{2.3}$$

其中，

E：以人月为单位的工作量；

KLOC：估算的项目需要交付的千行代码数；

EAF 为所有工作量乘数的乘积；

系数 a_i, b_i 见下表：

表 2.4 中等模型中不同项目类型对应的系数表

软件项目	a_i	b_i
组织型	3.2	1.05
半组织型	3.0	1.12
嵌入型	2.8	1.20

3. 详细模型（Detailed COCOMO）

详细模型（Detailed COCOMO）在中等模型的基础上还考虑了这些因子对单个项目阶段的影响，比如分析，设计等阶段。

2.2.2.2 COCOMOII

COCOMO II 对 COCOMO81 做了较大的更新，可以支持更多的开发过程模型，比如螺旋模型，RUP 等。针对不同的软件过程和软件项目开发阶段提供了三个模型：应用组装模型，早期设计模型和后体系结构模型[27]。可以根据所选择的软件过程，项目目前所处的阶段来决定选用哪个模型来进行估算。在软件规模估算上，COCOMO II 除了可以以源代码行为单位外，还可以通过功能点估算方法，模型可以自动将未调整的功能点数转化为源代码行数来进行估算。此外，COCOMO II 对成本驱动因子做了进一步的更新，使得更加符合现代的软件开发模式。

COCOMO 模型在众多估算方法中得到了广泛的应用和认可，这与其具有开放，完全文档化，数据驱动及面向公众的优点是分不开的。但是，该模型是将软

件生命周期中的产品设计，编码，集成与测试当成一个整体来进行工作量估算的，并没有独立对某个阶段的工作量进行估算，此外，COCOMO 起初是为大型项目来设计和构想的估算模型，不太适合于中小型项目估算，并且对于小型项目来说，估算过程也过于繁琐和复杂，估算结果也往往不太理想。

2.2.3 用例点分析法

用例点分析法（Use Case Point Analysis，简称为 UCPA）是功能点分析法的一个扩展，是基于用例规格说明书（Use Case）来估算系统的规模，并进而估算项目开发的工作量等。

精确计算功能点 FP 需要关于目标软件的详细信息，而通常这些信息都定义在软件详细设计书中。而为了在项目的更早阶段就能估算软件的工作量，提出了用例点 Use Case Point（UCP）方法[28]。UCP 基于定义了所开发软件系统功能领域的用例模型。

用例点 UCP 的数量从 Use Case 模型中计算得出。用例模型主要包括两种文档，系统/子系统文档和用例文档，涵盖了：系统名称，风险系数，系统级用例图，架构图，子系统描述，用例名称，简要介绍，上下文关系图，前提，事件流，用户接口等。

其中用来计算 UCP 的主要是系统级用例图和事件流。系统级用例图包含了一个或多个用例图来表示系统中的所有用例及 Actor。事件流包含了每个用例中的一组正常路径和每个可能的路径。图 2.3 和图 2.4 分别表示了某“订单处理系统”的系统级用例图和该系统所包含的一部分事件流。

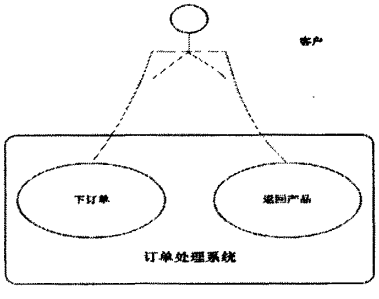

图 2.3 “订单处理系统”系统级用例图

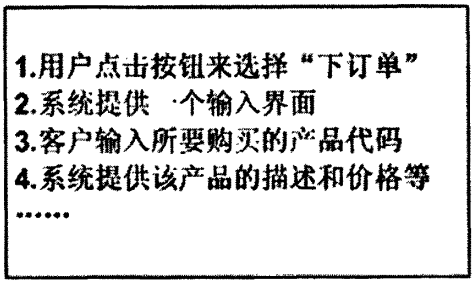

图 2.4 “订单处理系统”涵盖的一部分事件流

可以通过以一定权重来计算 Actors 和事件流中包含的事务的数量来得到用例点。事务是在 Actor 和目标系统间发生的事件，完全完成或者完全不发生。

基于用例点分析法进行工作量估算的方法步骤[29]如下：

步骤一：计算 Actor 权重

用例模型中的 Actor 被分为了三类：简单，中等和复杂。一个简单的 Actor 表示另一个通过已定义 API 进行交互的系统。一个中等 Actor 表示通过某种协议如 TCP/IP 进行交互的系统或者通过基于文本的接口进行交互的用户等。一个复杂的 Actor 指通过界面与系统进行交互的用户。

计算目标软件包含的同一个 Actor 类型的数目，并与表 2.5 中的权重因子相乘，最后将这些值相加即可得到总的 Actor 权重。

表 2.5 Actor 各类型权重分布

类型	描述	权重
简单	程序接口	1
中等	交互式的或基于某种协议	2
复杂	图形界面	3

步骤二：计算用例权重

将所有的用例都分为三种：简单，中等和复杂。通过每个用例中事务的数量来进行区分，每个事务都被定义为一组行为的原子集合。

计算目标软件包含的同一个用例类型的数目，并与表 2.6 中的权重因子相乘，

最后将这些值相加得到总的用例权重。

表 2.6 基于事务的用例权重分布

类型	描述	权重
简单	不超过 3 个事务	5
中等	4 到 7 个事务	10
复杂	超过 7 个事务	15

步骤三：计算未调整的用例点

将总的 Actor 权重和总的用例权重相加得到未调整的用例点（Unadjusted Use Case Point，即 UUCP）。

步骤四：技术因子和环境因子

定义了一组技术因子和一组环境因子，根据其对项目的假定的影响，每个因子都被赋予了一个 0 到 5 之间的值，0 表示该因子跟项目不相关，5 表示该因子非常重要。

技术复杂度因子（Technical Factor，即TCF）：图2.5中每个因子（T1-T13）的值乘以其权重，然后所有这些值相加可得TFactor。最后，计算公式为：

$$TCF = 0.6 + (0.01 * TFactor) \tag{2.4}$$

环境复杂度因子（environmental Factor，即ECF）：图2.6中每个因子（E1-E13）的值乘以其权重，然后所有这些值相加可得EFactor。最后，计算公式为：

$$ECF = 1.4 + (-0.03 * EFactor) \tag{2.5}$$

Factor	Description	Weight
T1	Distributed system	2
T2	Response or throughput performance objectives	1
T3	End-user efficiency (online)	1
T4	Complex internal processing	1
T5	Code must be reusable	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent	1
T11	Includes special security features	1
T12	Provides direct access for third parties	1
T13	Special user training facilities are required	1

图 2.5 系统的技术复杂度因子及权重分布[29]

Factor	Description	Weight
F1	Familiar with the Rational Unified Process	1.5
F2	Application experience	0.5
F3	Object-Oriented Experience	1
F4	Lead analyst capability	0.5
F5	Motivation	1
F6	Stable requirements	2
F7	Part-time workers	-1
F8	Difficult programming language	-1

图 2.6 项目组的环境复杂度因子及权重分布[29]

步骤五：计算 UCP

调整用例点公式为：

$$UCP = UUCP * TCF * ECF \tag{2.6}$$

步骤六：估算工作量

将每个用例点所需的人小时（man-hours）与 UCP 相乘可得最终的估算工作量。最后的计算公式为：

$$Effort = n(man-hours / perUCP) * UCP \tag{2.7}$$

用例点分析法从 1993 年提出至今，有许多研究者在此基础上作了进一步的应用研究。2005 年，Anda 对 UCP 方法步骤进行的修改结合了 COCOMO II 中针对改编软件的计算公式，使得它更适于增量开发的大型软件项目[30]。2005 年，Carroll 在其研究中对 UCP 等式加入了一个风险系数，最后估算结果是，95%的项

目估算误差在 9%以内[31]。

本文所要提出的模型是关于测试用例的规模和测试执行复杂度的，而功能点分析法和用例点分析法都是关于系统开发复杂性的。因此，我们很难直接将它们应用到对测试执行工作量的估算上面，但是我们仍然可以借鉴其关于系统的技术复杂度因素和项目组的环境复杂度因素的考虑和权重分布。

2.2.4 回归分析

回归分析（Regression Analysis）是研究变量之间相互作用关系的一种统计推断方法，由一个（或多个）因变量及一个（或多个）自变量组成。回归分析研究的目的就是通过收集到的样本数据用一定的统计学方法探讨自变量对因变量的影响关系[32]。根据研究的因果关系所涉及到的自变量个数，可以分为一元回归分析和多元回归分析。此外，依据描述自变量与因变量之间因果关系的函数表达式是线性的还是非线性的，分为线性回归分析和非线性回归分析。

在已有的回归分析工作量估算方法中，多使用线性回归方法，即试图寻找到关于一个或多个预测参数与一个因变量之间的线性关系，使得在数据集观察范围内的平方误差的平均达到最小[5]。其中，普通最小二乘回归（ordinary least squares regression，简称为OLS）是一种最传统的回归方法，其假定了将一个因变量与一个（或多个）自变量相关联的函数形式，通常，使用OLS的回归函数可以表示为：

$$y_i = \beta_1 + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + u_i \text{ [33]} \tag{2.8}$$

其中，

$x_{i2} \cdots x_{ik}$ 为对第 i 次观测值的回归变量，例如项目规模（SLOC）等对工作量估算影响巨大的因素，$\beta_{i2} \cdots \beta_{ik}$ 为响应系数，β_1 为截距参数，y_i 为第 i 次观测值的响应变量，u_i 为随机误差。

线性回归分析估算方法非常的易用，同时也具有一些局限性，包括：1.所需的历史数据的数量依赖于回归模型的自变量个数，当自变量变得非常多时，需要较多的经验数据来支撑；2.为了使回归分析估算系统更加稳定，需要使得自变量之

间不能具有很强的相关性，而这对复杂的软件工程数据提出了很严格的要求。

2.2.5 机器学习

机器学习(Machine Learning)是研究计算机如何模拟或实现人类的学习行为，以此来获取新的知识或技能，重新组织已有的知识结构使之不断改善自身的性能。它是人工智能的核心，是使计算机具有智能的根本途径，其应用遍及人工智能的各个领域。其中，像神经网络，规则归纳，遗传算法和基于案例的推理等都应用于非常广泛的领域，包括计算机视觉，经济和医学，在这些领域人类的能力都被证明要优于计算机。机器学习的这些技术可以通过不同类型的输入来产生一个输出。已经有一些研究将人工神经网络引入到了软件估算模型当中，并且表明其与传统的线性回归分析方法相比，可以更好地提高估算的精确性[5]。

人工神经网络（Artificial Neural Networks，即 ANNs）的灵感来源于生物神经网络结构，它可以看作由节点和节点间的连接（权重）组成的有向图。神经网络由一组矢量集来表示，即训练集，一次一个矢量。每个矢量都包含输入值和输出值。如图 2.7 中所示，输入为 x_0 到 x_{N-1}，输出为 y。神经网络的目标是为整组矢量集来特征化输入和输出之间的一个关系。在神经网络的训练期间，训练矢量中的输入在网络中进行传播。输入跟相应的权重相乘，然后乘积相加。在图 2.7 中，即为 $w_i * x_i$。如果和超过了某个节点特定的极限，那么该节点的输出将作为另一个节点的输入。不断重复该过程直到神经网络生成了对该矢量的相应输入部分的输出值。计算出的输出值跟期望得到的输出进行比较，得到一个误差值。依据神经网络算法，或者在每个矢量通过之后重新校正权重，或者在通过所有训练矢量之后进行校正。这两种情况的目标都是减少误差总数。不断的处理直到得到 0 误差。训练完成之后，特征化所有矢量的输入与输出之间的关系的神经网络模型被嵌入到神经网络的架构当中（节点和连接），然后用一组独立的矢量集，即测试集来测试该架构。如果训练适当，神经网络将会针对测试集产生合理的结果。

神经网络的输入包括一些对软件的度量，通常为软件的规模，开发复杂度等。最简单的情况就是我们以最基本的源代码行数即 SLOC 为输入，输出为开发的工

作量，即表示完成开发所需要的小时数。

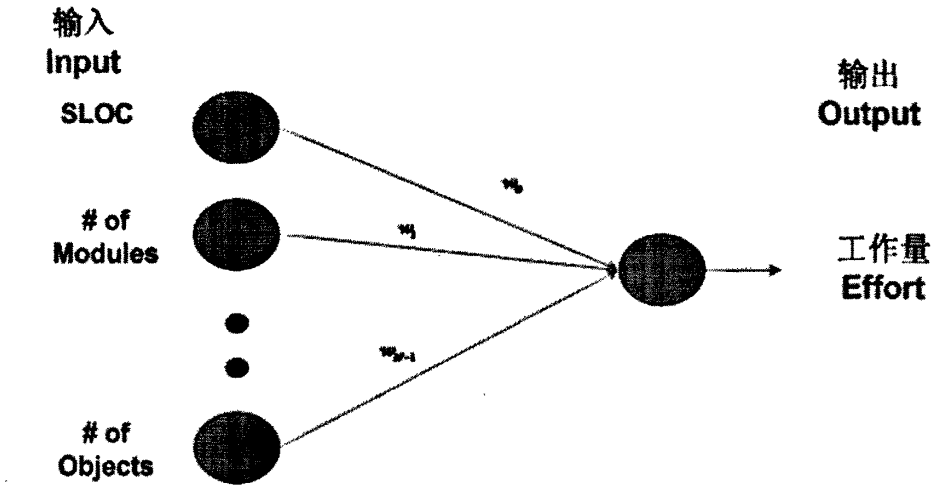

图 2.7 神经网络示例图

虽然人工神经网络在对工作量估算上比一般的线性回归方法要好，但是其仍然存在着一定的问题。神经网络的预测能力与训练能力存在着矛盾。一般情况下，训练能力差时，预测能力也会相应的比较差，并且在一定程度上，随着训练能力的提高，预测能力也会提高。但这种趋势有一个极限，当达到此极限时，即使训练能力在不断提高，预测能力反而会下降，即出现所谓“过拟合”现象。此时，神经网络学习了过多的样本细节，而并不能反映样本内含的规律。

2.3 软件测试执行工作量估算模型研究

测试经理会使用很多不同的方法来估算他们的测试工作量，并以此来安排测试工作。根据项目的类型，不同的组织会使用不同的方法，比如项目的内在风险，使用的技术等。而在实际工作中的大部分时间，测试工作量估算都捆绑在开发工作量的估算上，并没有单独的数字。但是随着软件模型估算方法的发展，也逐步有了一些方法和模型用来估算软件测试执行工作量。

2.3.1 传统方法

2.3.1.1 Ad hoc 方法

Ad hoc 方法指测试工作并不基于任何确定的期限，可以一直进行，直到一些管理或者市场方面的工作人员预先设定的时间为止，或者，直到用完财政预算为止。

在极端不成熟的企业中，这种方法非常的普遍，误差范围经常超过 100%。

2.3.1.2 开发时间的百分比

使用该方法的基本前提是测试工程开销依赖于开发时间和工作量。首先，开发工作量由一些技术，比如 LOC 或者功能点来进行估算，下一步是利用一些启发式的方法来确定一个值，该值通常浮动很大，多依赖于之前的经验。

该方法并不基于任何科学的原理或技术，因此易受攻击。真正的进度甚至超过估算时间的 50%到 75%，但是该方法也是目前使用最多的一种[34]。

2.3.2 功能点分析估算

基于对开发工作量进行估算的功能点方法，Capers Jones 提出针对相应的工作量，测试用例的数目可以由对功能点的估算来决定。具体的计算公式[35]如下：

$$NumberofTestCases = (FunctionPoints)^{1.2} \tag{2.9}$$

通过之前项目的历史数据可以得到一个关于测试用例与工作量之间的转换因数，并由此来计算实际的人小时（man-hour）工作量。

使用功能点来估算测试工作量的缺点在于我们提前需要非常详细的需求，并且该方法的主观因素非常大，估算精确度不高。

2.3.3 用例点分析估算

Suresh Nageswaran 提出了可以通过 Use Case 到 Test Case 的映射来将估算开发工作量中的用例点分析法 UCP 直接换算得到测试用例的数量，前提是必须使用如图 2.8 所示的 V-模型，并且在需求收集阶段即开始生成用例 Use Case[35]。

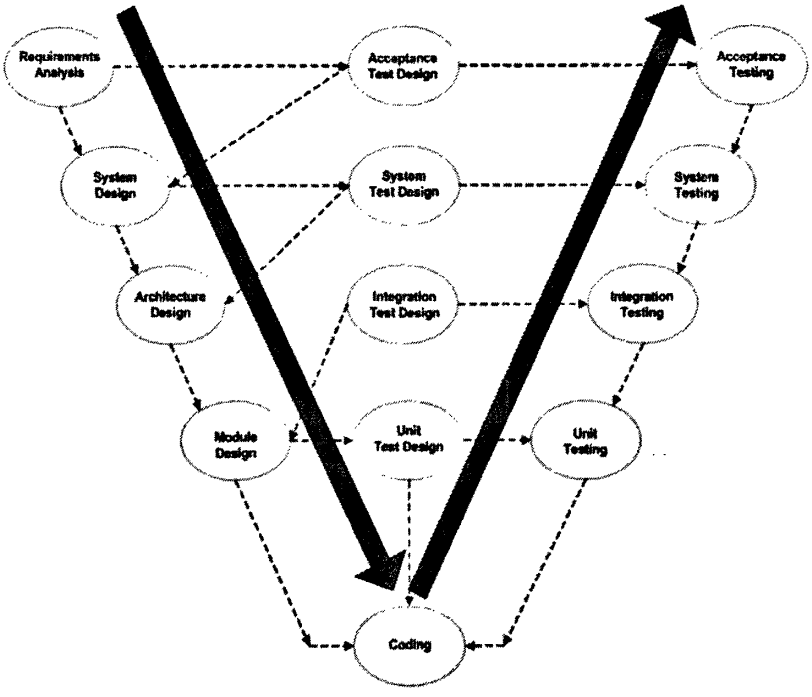

图 2.8 V-模型[35]

V-模型中的每一个测试活动都有一个相应的开发活动与之关联，如果在获得需求之后将估算过程放在图 2.8 的第二个阶段，那么很明显用例 Use Case 将作为估算过程的输入，可以直接得到测试用例的数目。用例 Use Case 的每个 scenario 和异常流（exception flow）是一个测试用例的输入。

利用用例点分析法来估算开发工作量的方法只需轻微的改动就可以用来估算测试工作量。现简单介绍该方法的具体步骤：

步骤一：确定系统中 Actor 的数量，可以得到 Unadjusted Actor Weights，即 UAW

Actor 处于系统之外并与系统进行交互，包括终端用户（End-User），其他程序，数据存储等。有三个等级，分别为：简单，中等，复杂。测试工作量估算中的 Actor 划分与开发估算中有所不同。

终端用户属于简单的 Actor，在测试当中，可以使用自动化的测试脚本来简

单模拟终端用户的行为；中等的 Actor 通过一些协议来与系统进行交互，他们也可以是一些数据存储，之所以将他们归为中等是因为通常需要手工运行 SQL 语句来完成这些测试用例；复杂的 Actor 是那些通过 API 与系统交互的独立的系统。表 2.7 中列出了 UCP 在测试工作量估算中各种类型 Actor 的描述和权重分布。

每类 Actor 的数量与相应权重因子的乘积，最后相加即可得到 UAW。

表 2.7 UCP 在测试工作量估算中的 Actor 类型权重分布

类型	描述	权重
简单	界面 GUI	1
中等	交互式的或基于某种协议	2
复杂	API/低端相互	3

步骤二：确定系统中用例 Use Case 的数量，得到 UUCW

步骤三：UUCP = UAW + UUCW

步骤四：计算技术和环境因素，如表 2.8 中所示

表 2.8 UCP 方法中对于影响测试的技术和环境因素列表

影响因素	描述	权重
T1	测试工具	5
T2	文档输入	5
T3	开发环境	2
T4	测试环境	3
T5	测试组件重用	3
T6	分布式系统	4
T7	性能目标	2
T8	安全功能	4
T9	接口复杂度	5

为每个技术和环境因素根据一定的标准赋予一定的值，然后与相应的权重相乘，所有乘积最后相加即可得到 Technical Environmental Factor，即 TEF 值。

步骤五：计算调整后的 UCP，计算公式为：

$$AUCP = UUCP * [0.65 + (0.01 * TEF)] \tag{2.10}$$

步骤六：得到最终的工作量估算值。

将 UCP 与某个转换因子（Conversion Factor）相乘得到最终的估算值，该转换因子表示测试工作量中的 man-hour。一般来说，机构需要根据很多各种不同的因素来确定这个值。

$$Effort = AUCP * CF \tag{2.11}$$

2.3.4 基于测试规格说明书的测试工作量估算模型

Eduardo Aranha提出了一种基于测试规格说明书的测试工作量估算模型，并定义了一种关于测试规模和执行复杂度的度量标准，该标准基于用一种受约束的自然语言写的测试规格说明书[26]。该模型是现在仅有的几个测试工作量估算方法和模型中描述较完整和详细的，本节将对该模型进行一定的介绍。

该模型的核心为使用一种测试规范语言。测试常用前提（Precondition），过程（step）（包括输入和期望输出的一系列测试步骤）和后期条件（Post condition）等来指定[36]。这些测试用例一般用自然语言来书写，经常会带来一些像含糊，冗余和缺乏书写标准等问题，使得理解测试和估算执行复杂度变得更加困难。然而，他们可以通过使用受约束的自然语言来避免。

受约束的自然语言（Controlled Nature Language，即CNL）是自然语言的子集，语法和词典是有限制的，从而用更简明和标准的方式来书写。这种限制减少了描述一个事件可能的方式。

该模型所涉及的测试规格说明书是用CNL来描述的。使用一种简化的方式，规格说明书中的每句（测试步骤）与下述结构一致：一个主动词和0或多个名词。动词描述了所要完成的测试步骤中的行为。名词提供了关于该行为的额外的信息。CNL可以针对不同的应用领域有自己的词典和语法。

该估算模型的输入是一组测试套件，输出是估算执行完测试套件内的所有用例的开销，以man-hour为单位。工作方式大致如图2.9所示：

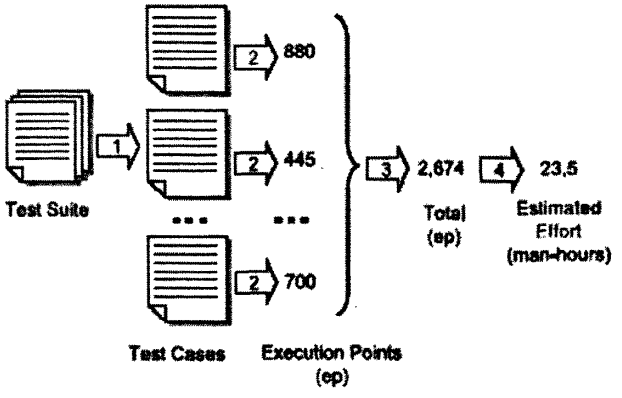

图 2.9 估算执行一组测试套件的工作量[26]

该测试执行工作量估算模型按如下步骤进行工作：

步骤一：分析测试套件中的所有测试用例；

步骤二：为每个测试用例分配一定量的执行点（Execution Point）。在该模型中是用来描述测试用例规模和执行复杂度的一个度量标准；

步骤三：将分析后的测试用例的所有EP累加。该值描述了整个测试套件的规模和执行复杂度；

步骤四：估算执行完整个测试套件中所有用例所需要的工作量开销。

图2.10阐明了模型如何度量测试用例规模和执行复杂度。首先，（a）单个分析测试规格说明书中每一个测试步骤。根据一系列的特征（C_1到C_n）来分析每一个测试步骤。

这些特征代表了该步骤执行时，涉及的一些常规功能性和非功能性的需求。一些可能的关于特征的例子如在屏幕间的导航数，按键数目和网络应用等。特征列表对于不同的应用领域可能会有所不同。

（b）该模型中的每个特征对测试的规模和执行复杂度都有影响，并且用一

种顺序的度量（低，中，高）来表征。

之后，(c) 根据其影响级别为每个特征分配执行点EP。目的是将定性的等级（影响级别）转化为一个定量的值。例如，级别为“低”的特征C_1可以被赋予30个执行点EP。

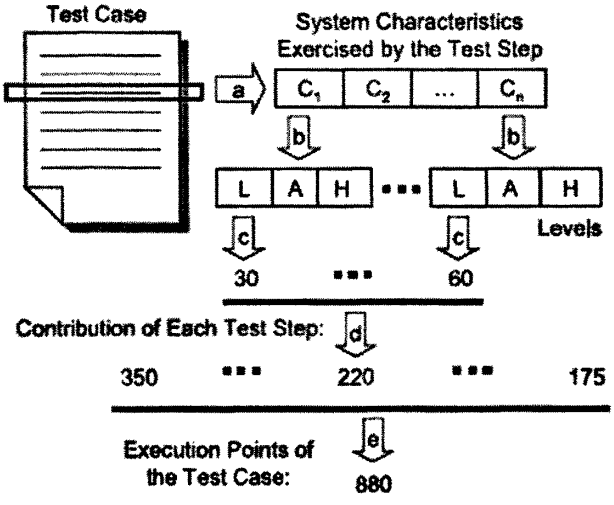

图 2.10 为某个测试用例计算 EP 数[26]

为了计算一个测试步骤的EP数目总和，(d)将每个特征的执行点EP数目相加，(e) 将测试用例中每个步骤的执行点数目相加，可以度量该用例的规模和执行复杂度。

一个测试套件总的执行点EP数提供了关于其规模和执行复杂度的参考。工作量的估算基于这个信息及转换参数（CF）进行计算。该转换参数表现了测试执行工作量和执行点之间的关系，视测试组的生产力而有所不同。

转换因数为每EP的秒数，表示执行一个测试用例中的每一个执行点所需的秒数。为了计算转换因数，测试人员可以度量几个测试用例的测试规模和执行复杂度。测试的执行时间可以通过查看历史数据库或者执行来收集。

由图 2.11 可见，(f) 转换因数为总的工作量开销除以总的执行点 EP 数。这条信息可以用来估算新的测试套件所需要的执行开销，只需要 (g) 用 EP 数乘以

转换因数即可得到。

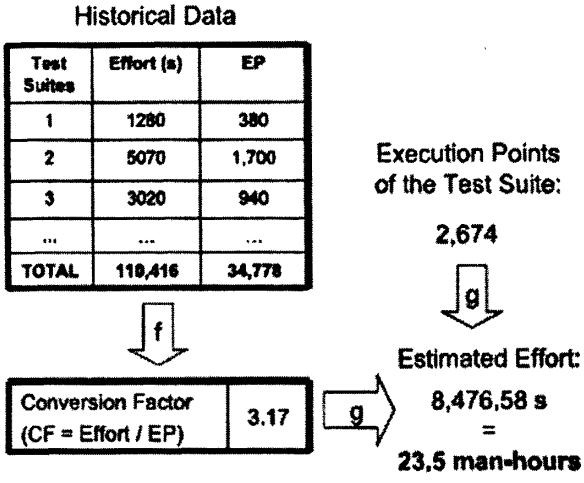

图 2.11 使用 EP 和 CF 来计算测试执行工作量[26]

该方法较具体的介绍了一种对软件测试执行工作量进行估算的方法，但是其对软件测试规格说明书有着严格的要求，并且在进行一个测试套件的总的执行点 EP 数的计算时，需要较大的人力物力并花费较长的时间来进行统计，此外，转换因数 CF 的确定也有着很大的主观性。

2.4 本章小结

本章主要分三部分，首先对软件测试的相关理论进行了简单介绍，包括软件测试的定义、分类，软件功能测试的定义及软件测试的基本流程等。然后详细阐述了现有较著名的软件工作量估算模型和方法，分析和研究了功能点分析法，COCOMO 模型，用例点分析法，回归分析和机器学习等估算方法的基本概念和原理，并做了一定的对比，最后对现有的专门针对软件测试执行工作量的估算模型进行了研究和分析，为之后建立估算模型做铺垫。

第3章 基于经验的软件测试执行工作量估算模型

软件测试对于保证软件质量来说至关重要，而如何能够合理安排和计划软件测试的资源和进度对于顺利完成测试工作来说，则是重中之重。本章将基于之前对已有相关模型的认识基础上，借鉴其中精华提出一种新的专门用于估算软件测试执行工作量的模型，称之为 Software Test Effort Estimation Model，简称为STEEM。该模型基于机构中已有项目的经验数据，利用机器学习的方法进行训练，来对机构未来项目中软件测试的工作量估算提供参考。

3.1 STEEM 模型设计原则

➢ 估算具有不确定性

软件的测试活动受到很多因素的影响，比如，测试周期所涵盖的测试范围，测试环境，测试人员的技术水平和熟练程度等都会对测试执行工作量的多少产生很大的影响。而在一个新的测试周期中，测试范围可能会发生变化，测试环境可能会出现问题，部分测试人员可能会休假或者离开等等，这些提前预见不到的情况都使得对软件测试执行工作量的估算具有很大的不确定性。

➢ 估算的结果是一个承诺

估算其实是根据过去预测未来的过程，我们不可能使未来的测试情形同过去完全相同，因此估算并没有正确不正确之说，只能用是否合理来评价[37]。合理的估算倾向于变成一种自我实现的预言，有了提前设定的工作量和未来的进度目标，人们的天性和成功的愿望就会保证其达到估算的要求。所以说估算是一种承诺，当估算结果出来后，人们会以此为目标，尽其所能的完成任务。

➢ 完成设计测试用例之后进行估算

对已有信息了解的越充分，对所要测试的功能和范围越清楚，得出的估算结果将会越合理，因此，在现阶段，本文的工作量估算模型同其他已有的软件测试估算模型一样，也是在完成设计测试用例之后进行的。虽然软件组织希望能在项

目早期就把估算误差控制在10%以内，但理论上这是不可能实现的[38]。此外，在大部分时间中，对于已处于维护期的软件项目，每一次版本升级都需要进行一定量的回归测试，因此可以很轻松地利用本模型对这些已有测试用例的回归测试执行工作量进行估算。

➢ 估算手工功能测试执行工作量

本文的估算模型是专门针对于手工功能测试执行工作量的。功能测试是系统测试中最基本和最重要的测试类型，任何软件项目都不能缺少功能测试。而对于测试人员来说，通常最花费时间和精力的就是那些繁重，枯燥的手工功能测试。对于功能测试范围中已经实现自动化测试的那部分，所需的时间基本上已经确定，因此并不在本文估算模型考虑范围之内。

➢ 以测试套件为估算级别

通常将用来测系统某大类功能的一组测试用例（test case）进行整理归并形成一份测试套件（test suite），比如对于C/S或B/S架构的软件项目，一般会分为专门针对于GUI测试的测试套件和专门针对Server功能的测试套件。在真实的测试工作周期中，测试套件作为一组相近测试用例的集合，通常会分配给同一位测试人员来负责，完整的测试范围则一般涵盖多个测试套件。因此，本文的估算模型是以测试套件为考察对象级别来进行工作量的估算，而并非像Eduardo Aranha提出的基于测试规格说明书的测试工作量估算模型[26]那样细化到每个测试用例，从而使得本模型使用起来更加高效，在真实的测试情形下也更有现实意义。

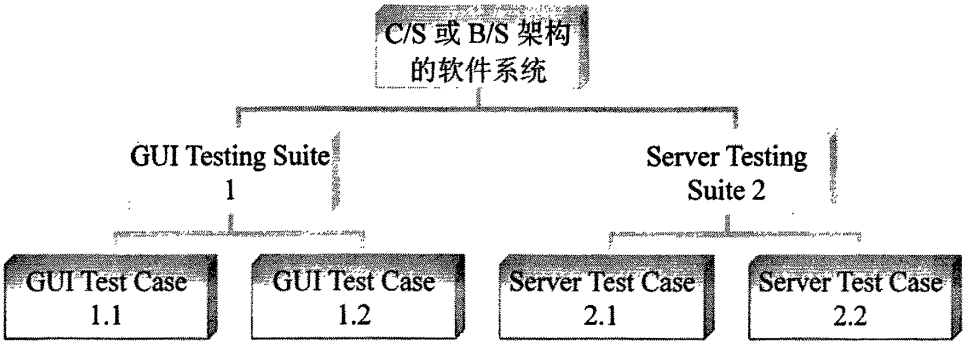

图3.1 测试套件结构图

3.2 STEEM模型概要

就像其他基于经验的估算模型和方法那样，我们的 STEEM 模型也是基于以下断言：相近的测试具有相近的工作量开销。模型借鉴了基于案例的推理（Case-Based Reasoning，即 CBR）方法中提到的四个步骤[39]：

步骤 1：获得最相近的案例；

步骤 2：重用该案例中的方法和知识来解决问题；

步骤 3：重新修订提出的方法；

步骤 4：记录这部分经验，为将来使用做准备。

首先，本文经过对测试套件及真实的测试情况进行分析和研究，提出了测试套件执行矢量（Test Suite Execution Vector，即 TSEV）这一概念，该矢量包含了对测试执行工作量开销影响巨大的三个元素，分别为测试用例数量，测试执行复杂度及测试人员熟练程度。测试用例数量（Test Case Number）指测试套件中的需要被执行的测试用例数目；测试执行复杂度（Test Execution Complexity）是对所执行测试套件的复杂度分级，我们用一种定量的方法对其度量；测试人员熟练程度（Tester Rank）即根据测试经验技能和对所测系统的了解来对测试人员进行级别划分。其中，测试用例数量和测试执行复杂度的定义来源于软件产品的规模和开发复杂度[40][41][42]，并且借鉴了[26]中提到的相关概念。

然后，判断当前需要进行工作量估算的测试套件执行矢量与经验数据库中的每个测试套件执行矢量之间的相似度。然后利用经验数据库中的已知工作量开销值来得到对该测试套件执行矢量的工作量估算值。本文中使用机器学习的方法来进行训练和估算。

STEEM 模型以测试套件为考察级别，模型的输入是这一组测试套件，输出是估算执行完测试套件内的所有测试用例的工作量开销，以人小时（man-hour）为单位。基于以上说明，模型整体结构图如下所示。对于测试套件执行矢量中的三个元素，本文将在以下几节重点介绍。

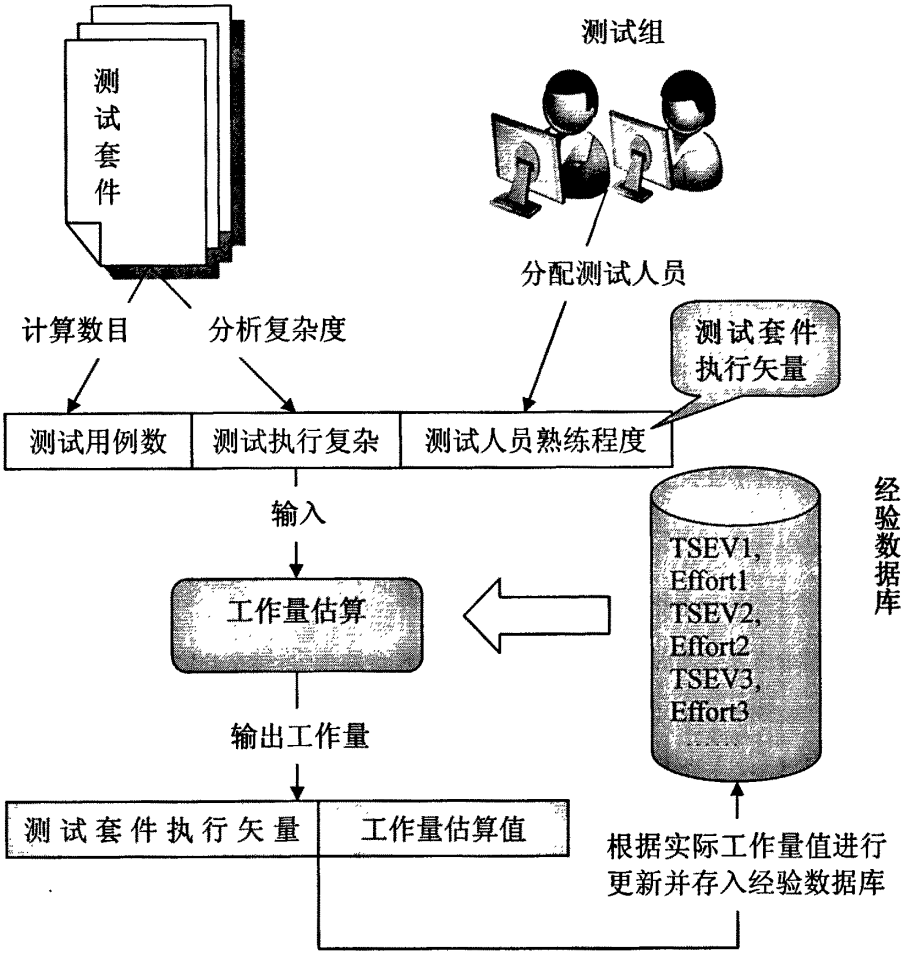

图 3.2 STEEM 模型整体结构图

3.3 测试套件执行矢量

3.3.1 测试用例数量

STEEM 模型以一个测试套件为考察对象，如前面设计原则中提到的，测试套件是由一组测试相近功能的测试用例组成的。测试用例数量指一个测试套件当中计划要执行的测试用例数目。一般情况下，一个测试套件中只需要执行部分测试用例。在测试人员编写测试套件文档时，通常都会对测试用例进行编号，因此我们可以很容易就得到每个测试套件中所要执行的测试用例数量。

测试用例的数量是对测试套件一个非常直观的度量因素，很明显，对于两个不同的测试套件，测试用例数量的多少将会直接影响最终执行测试的工作量开销。所以在此将测试用例数量作为测试套件执行矢量中的第一个元素，也是模型的输入参数之一。然而，对于测试用例的设计来说并没有一个非常统一的格式规范限制，因此仅仅考虑测试用例数量的话，在估算工作量方面并不全面。作为补充，本文引入了测试执行复杂度的概念。

3.3.2 测试执行复杂度

考虑这样一个问题：

Q：两个包含相同数量测试用例的测试套件让同一个人去执行，执行其中一个测试套件需要花费2天，而执行另一个则需要花费5天，是什么因素造成这样的差距？

在测试过程当中，测试人员会明显感觉到包含某些测试用例的测试套件测起来比较容易，也比较快，而有些测试用例则相当地费时费力。但是在平时的工作中，并没有去仔细思考具体是什么样的因素影响了执行测试的效率。

本文中的测试执行复杂度完全从测试执行的人工消耗的角度来考虑，从测试套件（Test Suite）这个级别来分析，对测试进行复杂度分级。

首先，我们需要统计和抽象出影响测试执行效率的一系列因素。根据测试套件的定义，我们知道其包含的所有测试用例都是用来测某类功能的，并且会具有相似的形式和结构。因此我们可以从整体上分析和研究测试用例和测试套件的具体组成部分来抽象影响测试效率的各种因素。一个测试用例（Test Case）包含了事件，行为，输入，输出，期望结果和实际结果等。一般来说，输入和期望结果决定了这个测试用例[43]。通常测试用例会包括一系列详细的测试步骤(Test Step)。基于测试用例的这些组成部分及实际的测试经验，可以从以下几个大的方面来考虑这些因素：测试输入（Test Input），输出检查（Output Validation）和测试环境（Test Environment）等。一个典型的测试结构模型如图3.3所示：

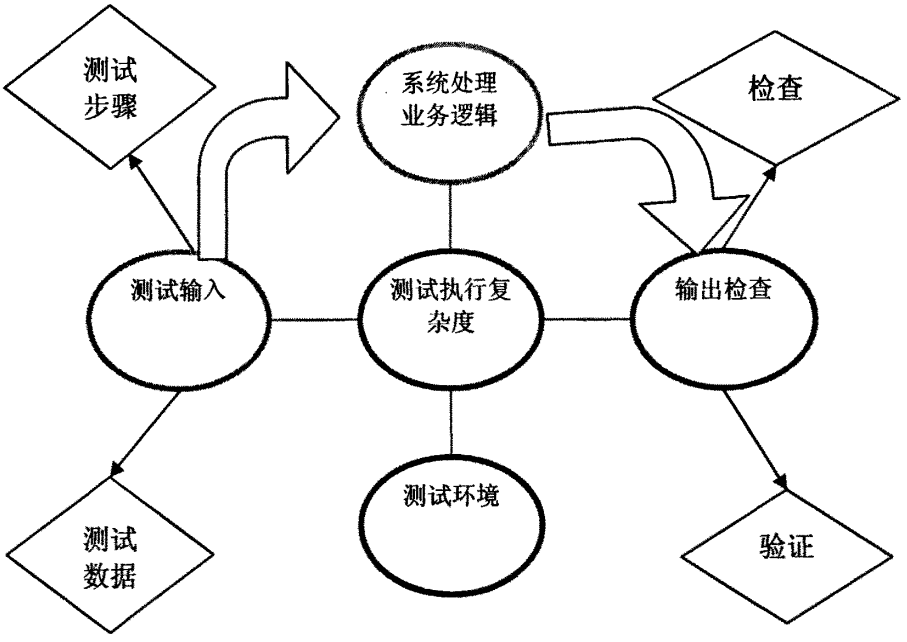

图 3.3 一个典型的测试组成结构图

然后，制定关于测试复杂度的度量标准。根据每个因素影响程度不同分配给相应的权重（Weight），并针对每个因素都有一个衡量的标准，根据该标准给测试套件中各个相应复杂度因素进行评分（Scale，可以分为 N/A，Low，Average，High 四种情况）。

最后，将各个因素的分值与相应权重的乘积加起来，总和即该测试套件的测试执行复杂度。计算公式如下：

$$\text{测试执行复杂度} = \sum_{i}^{n} [Scale(f_i) * Weight(f_i)] \tag{3.1}$$

其中，

f_i为复杂度因素列表$(f_1, f_2, f_3, ..., f_n)$中的一个；

$Scale(f_i)$为针对该测试套件，对因素 f_i所打的分数，值（0，1，2，3）分别代表了等级（N/A, Low, Average, High）；

$Weight(f_i)$为因素 f_i的权重值。

图 3.4 示例了一组用来度量一个测试套件的测试执行复杂度的影响因素及对各因素评分的简单衡量标准。该表以某金融软件公司为背景，列出了在该机构中影响测试执行效率的主要因素，下一章中我们将针对该金融软件公司应用 STEEM 模型，并介绍各个因素的具体含义和权重分布等。

Category	Factors		Measure Basis			
Test Input	*Test Steps*	Number (per Test Case)	A Few	Quite a Few	Large Number	
		Dependency Between Steps	N/A	Low	High	
	Test Data	Number (per Test Case)	A Few	Quite a Few	Large Volume/Batch File	
		Logic dependency between Data	N/A	Low	High	
Output Validation	*UI Validation*	Static Item Display	N/A	Easy	Hard	
		UI Actions/Feedback	N/A	Easy	Hard	
	Business Logic	Business Process/Calculation Rules	N/A	Easy	Normal	Complex
		Data Volume	N/A	Single Item	Multiple Items	Large Volume
Test Environment	*Software Environment*	Integration with external applications	N/A	Loose Coupling	High Dependency	
		Require simulators for execution	N/A	Required		
		Configuration Change: DB/Configurati	N/A	Required		
	Hardware Environment	NetworkCondition	Bad	Good		
		Server Equipments Performance	Bad	Good		
	Management Environment	Require Support Team due to Access Permission	N/A	Required		
		Specifcial Procedure for some Test due to Company Policy	N/A	Required		

图 3.4 度量测试套件测试执行复杂度的影响因素及衡量标准

一般来讲，不同类型的软件系统或不同类型的软件公司和机构有着千差万别的特点，互联网公司开发的 Web 应用同普通商用软件就有着显著的不同，影响测试效率的因素就会有很大的差异。此外，国内软件公司的产品同需要不同国家协同开发的软件系统也有着很大的差异，比如：测试环境（Test Environment）中的网络条件（Network Condition）在不同的公司可能有着不同的定义，对于国内软件公司，一般就是指外界的网络条件等，而对于跨国的软件公司来讲，大多数测试可能都需要在另一个国家的测试环境下进行，这个时候就涉及到了通过 VPN（Virtual Private Network）或直接远程登录到对方的机器上测试，在此情况下网络对测试效率的影响就会变得特别突出，因此给的权重也会有所不同。针对这种

差异，不同机构应该从实际的软件系统特点出发进行分析总结，调整图 3.4 中的因素以适合自身测试情况，并根据具体的影响程度来赋予相应的权重（Weight）。

抽象这些影响因素并制定度量标准的方法主要有两种：

1. 专家分析

安排机构中了解整体软件产品特点并具有丰富测试经验的专家一起讨论来总结出这些影响因素，权重及评分衡量标准；

2. 针对整个机构中的测试人员进行调查，统计

这也是比较好的一种方法，可以为所有测试人员提供这样一份问卷，涵盖了一些基本的影响因素，比如 *Test Steps*，*UI Validation* 等作为基准，然后让他们根据自己的实际情况进行添加和打分，最后统计问卷得出本机构中影响效率的各大因素及权重。

3.3.3 测试人员熟练程度

在已有的软件测试工作量估算模型中，很少有把人的因素独立出来考虑的[26][35]。或者是根本没有计算在内，或者是仅仅作为影响成本驱动因子所有因素中的一个而已[24]。而在实际测试当中，人的因素影响巨大。

跟之前的那个问题类似，我们有：

Q：同样一个测试套件让两个测试人员去执行，其中一个人完成测试需要花费 2 天，而另一个则需要花费 5 天，是什么因素造成这样的差距？

显而易见，在这里人是造成这个巨大差距的最主要因素。因此，STEEM 模型将测试人员的熟练程度作为了一个独立的因素来考虑。

在软件开发工作量估算中很难度量人的因素的影响，比较幸运的是，在测试执行中，我们对测试人员行为的度量相对比较容易。因为测试人员在执行测试过程中仅仅是按照测试用例一步一步的在做，比起进行系统设计和编码来说，是相对比较机械和固定的。因此，我们可以从执行测试的角度，通过对两个不同维度的组合对测试人员的熟练程度进行级别划分，分别是：

- 维度 X：测试人员的技术背景和测试经验

对于一个有着丰富经验的测试员和一个刚刚进入这个行业的新手来说，执行一定量测试所需要的时间差别非常的大，通常，新手对测试中一些异常情况的把握，对测试方法和测试过程的理解都有一定欠缺，在进行工作量估算时，必须将这一点考虑在内才能保证估算的合理性。本文中，模型以一年的测试工作经验为分界线来将该维度分为初级和高级两个层次；

- 维度 Y：测试人员对所测软件系统的熟悉度

两个具有同样丰富测试经验的测试员，一个在目标项目从事测试工作多年，对系统功能和特点非常熟悉，而另一个是刚刚从其他项目或公司过来的测试员，对所要测的目标项目完全不了解。让这样的两个人去进行测试，所需要的时间差距可想而知。我们用过去已测试这一测试套件的次数来区分该维度，已测过多于两次则算高级，否则为初级。

基于以上两个维度，我们可以对测试人员的熟练程度进行度量并分为 1，2，3，4 四个级别（Rank），如下图所示：

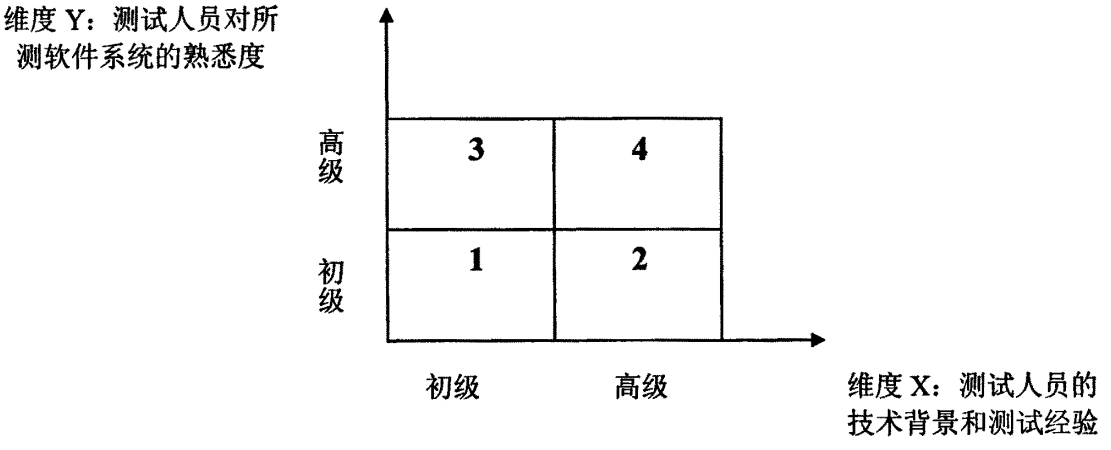

图 3.5 测试人员熟练程度度量图

级别 1：两个维度都为初级；

级别 2：维度 X 为高级，维度 Y 为初级；

级别3：维度Y为高级，维度X为初级；

级别4：两个维度都为高级。

在实际对测试做计划的过程中，可以很容易就判断出测试人员是否满足这两个维度，从而可以对测试人员熟练程度这个元素进行评级，并作为模型的输入之一。

3.4 STEEM模型训练和预测

STEEM模型基于回归分析的思想，通过收集机构项目的经验数据来分析和研究测试套间矢量中的三个参数与执行完该测试套件所需要的工作量时间（man-hour）之间的关系，属于多元回归分析。

$$(TestCaseNo., TestExecComplexity, TesterRank) \xrightarrow{f} Effort \qquad (3.2)$$

过去已有一些基于线性回归分析方法的估算模型[5][33]，而仅仅将软件工程中有着复杂相互关系的参数用普通的线性函数表达式来进行拟合，存在着一定的局限性。随着计算能力的不断提高，出现了较多新的技术和方法来解决非线性回归分析问题。本文尝试使用机器学习中的支持向量机方法来对历史数据进行训练和学习，以期达到比较好的估算效果。

通过模型，我们可以得到执行某个测试套件所需要的工作量估算值。在完成执行该测试套件后，将测试套件执行矢量与实际的执行时间一并放到经验数据库中，以备未来的使用。STEEM模型进行估算的过程如图所示。

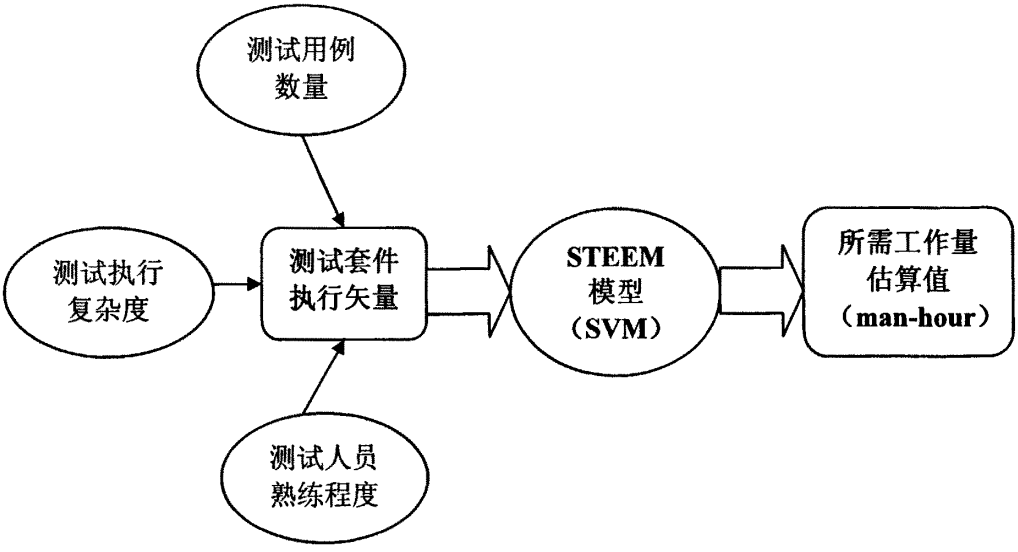

图 3.6 STEEM 模型估算流程图

3.4.1 支持向量机的优势

支持向量机（Support Vector Machine，简称为SVM）最初是由V. Vapnik等首先提出的方法，建立在VC维(Vapnik-Chervonenkis dimension)理论和结构风险最小原理基础之上，根据有限的样本信息在模型的复杂性和学习能力之间寻求最佳，以期获得最佳的推广能力[44]。作为建立在统计学习理论基础之上的新一代机器学习算法，支持向量机的优势主要体现在解决线性不可分问题，它通过引入核函数，巧妙地解决了在高维空间中的内积运算，从而很好地解决了非线性分类和回归问题[45]。

近年来 SVM 在理论研究和算法实现方面都取得了突破性进展，开始成为克服“维数灾”和“过学习”等传统困难的有力手段，并且 SVM 在非线性回归分析中表现出了极好的性能。

用 SVM 估计回归函数的思想就是通过一个非线性映射，把输入空间的数据映射到一个高维特征空间。然后在此空间做线性回归。SVM 在解决非线性回归分析中具有以下典型的优势：

第一，SVM 在有限样本的情况下，目标是得到现有信息和知识下的最优解而不仅仅是样本趋于无穷大时的最优值，减少了出现“过学习”情况的可能性；

第二，在神经网络方法中存在局部极值问题，SVM 很好地解决了这一点，最终得到的是全局最优点；

第三，SVM 通过非线性映射，把输入空间的数据映射到一个高维特征空间，从而使其有更好的推广能力。

3.4.2 STEEM 模型建立

初次在一个机构中建立 STEEM 模型进行工作的大致流程如图 3.7 所示，现介绍其包含的步骤：

1. 配置模型参数

模型的输入——测试套件执行矢量中的测试用例数量和测试人员熟练程度基本不需要再调整，关于测试执行复杂度，需要根据机构的项目特点进行一定的调整使之更符合实际情况，具体涉及到：影响测试效率的因素及其权重，对这些因素进行评分的衡量标准等；

2. 收集历史数据

在该机构中，根据确定的测试套件执行矢量的定义对历史项目的经验数据进行收集。对于测试套件的测试执行复杂度的计算，为了使历史数据更加可靠，在这里使用专家数据，即让那些测试人员熟练程度为级别 4 的该项目的测试专家对影响测试效率的各个因素评分并计算最终的值。可以从该测试周期的测试报告中提取出执行完某一测试套件所需要的实际工作量时间；

3. 数据预处理

从所收集的结果中剔除掉一些无效的数据，并使剩下的数据尽可能的覆盖各种情况下模型参数的可能取值，保证模型训练结果的精度和合理性。然后按一定的方法和比例将数据集分为训练集（Training Set）和测试集（Test Set）；

4. 数据训练

使用一些已实现的开源的机器学习工具，如 WEKA 和 LIBSVM 等对数据进

行训练；

5. 精确度分析

基于测试集对模型结果进行精确度分析，使用 MMRE，PRED(k)等误差分析方法，将模型估算的结果和实际值进行比较，对模型的合理程度做出一定的判断。

STEEM 模型建立过程图

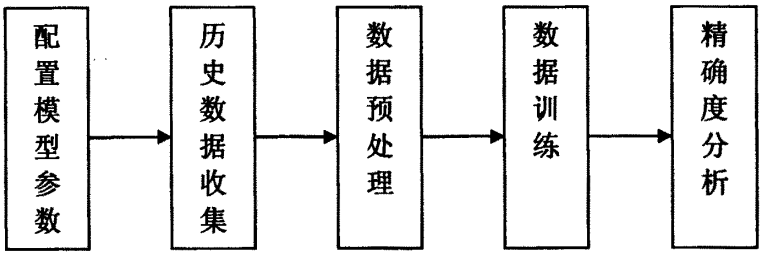

图 3.7 STEEM 模型建立过程图

3.5 本章小结

本章重点介绍了基于经验的软件测试执行工作量估算模型的建立，分析了模型建立所遵循的设计原则和基本原理，并详细阐述了测试套件执行矢量的概念，将测试套件特征化为一个包含了测试用例数量，测试执行复杂度和测试人员熟练程度的三维矢量，并定义了关于这三个元素的度量标准，最后介绍了模型建立的过程和步骤，包括配置模型参数、收集历史经验数据、数据预处理、数据训练和精确度分析等五部分。

第4章 STEEM在某金融软件公司的应用实例

本章介绍了在某金融软件公司中首次使用 STEEM 模型的过程，针对金融软件系统的特点配置测试套件执行矢量中的三个元素，然后基于该公司中历史测试周期的数据库对模型进行训练，并评估模型估算的精确度。

4.1 实验公司背景介绍

该金融软件公司为某国外银行在中国的全资子公司，致力于研究和开发全球化金融服务中的信息技术，为该银行的全球金融业务提供 IT 研发和支持，并已建立起全球化协同软件开发模式下的应用软件工程和项目管理标准。负责的项目已涵盖了固定收益交易，证券交易，外汇交易及金融衍生产品等多个方向。

为了更好的在本机构中应用 STEEM 模型，本文简单总结了金融软件系统测试方面的一些特点：

- 复杂的业务逻辑

通常，金融软件系统的业务逻辑都比较复杂。以股票交易为例，系统需要提供给客户尽可能优化的价格，帮助客户路由到不同的股票交易所等服务。这个时候就需要测试人员对相关金融知识有一定了解，才能更快更好地完成测试工作；

- 对功能正确性的高要求

在某些类金融软件系统中，一个小数点的错误就可能会影响到数额巨大的交易结果，从而造成严重并恶劣的后果；

- 庞大的数据量

多数金融软件系统每天都要处理数量庞大的订单（Order），并且需要将这些重要的订单和交易信息等保存到数据库。而辅助进行 post-trade 工作的软件系统，也要处理包含数据数量巨大的报表等。因此，在很多情况下，金融软件系统的测试都需要大量的数据，包括进行输入，输出检查等；

- 复杂的网络和硬件环境

处于性能的考虑，金融软件系统通常对硬件和网络的要求都非常的高。系统往往依赖于复杂的硬件环境，在测试过程中，关于硬件环境和网络等因素的影响很多时候是比较大的。

4.2 配置测试套件执行矢量

由第三章的介绍可知，测试套件执行矢量作为 STEEM 模型的输入，包含了测试用例数量，测试执行复杂度及测试人员熟练程度等三个元素。其中，对于测试用例数量和测试人员熟练程度这两项，不同机构间并没有太大的区别，只要按照模型中该元素的定义进行度量即可。以下将主要介绍如何在该金融软件公司中配置测试套件执行矢量中的测试执行复杂度这一元素来使其更加符合本机构的特点，从而保证模型估算的合理程度和准确性。

测试执行复杂度完全从测试执行的人工消耗的角度来考虑，从测试套件这个级别来分析，对测试进行复杂度分级。根据测试套件的定义，我们知道其包含的所有测试用例都是用来测某类功能的，并且会具有相似的形式和结构。因此我们可以从一个测试套件中平均每个测试用例的复杂度如何来分析具体的影响因素。

4.2.1 专家讨论

首先，我们在该机构内部进行了一个小范围的专家讨论，即邀请一些有丰富测试经验并且相对比较熟悉公司项目背景和特点的 QA 来参加，共同讨论并确定符合本机构软件项目特点，影响测试执行复杂度的各个因素。基于上一节中对金融软件项目测试特点的总结，确定了测试输入（Test Input），系统处理过程（System Process），输出检查（Output Validation）和测试环境（Test Environment）等四个大的考虑方向，然后每一大项下面又分为一些具体的影响因素。如图 4.1 所示：

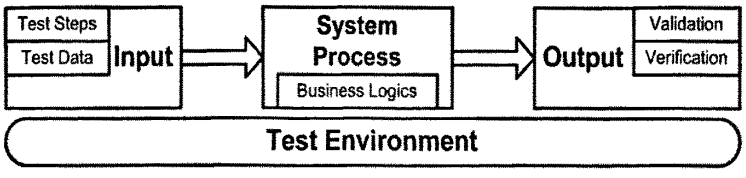

图 4.1 影响测试效率因素分类图

现分别从这四个方向来具体介绍各个因素的含义。

➢ 测试输入（Test Input）

测试输入包含了测试步骤（Test Steps）和测试数据（Test Data）两个方面，如表 4.1 所示；

表 4.1 测试输入各影响因素

Category	Factors		Description
Test Input	*Test Steps*	Number (per Test Case)	测试套件中测试用例平均包含的测试步骤的数目
		Dependency Between Steps	测试套件中测试用例步骤之间的平均关联程度
	Test Data	Number (per Test Case)	测试套件中测试用例平均所需的测试数据数量
		Logic dependency between Data	测试套件中测试用例所需的测试数据之间的平均关联程度

➢ 系统处理过程（System Process）

系统处理过程主要从软件系统的业务逻辑（Business Logic）方面来考虑，如表 4.2 所示；

表 4.2 系统处理过程各影响因素

Category	Factors		Description
System Process	*Business Logic*	Business Process / Calculation Rules	测试套件中涉及到的业务逻辑处理过程及计算规则等
		Data Volume	验证业务逻辑时涉及到的数据量

➢ 输出检查（Output Validation）

输出检查主要涉及到静态显示项（Static Item Display）及界面行为/反馈（UI Actions/Feedback）两方面的验证和检查，如表 4.3 所示；

表 4.3 输出检查各影响因素

Category	Factors		Description
Output Validation	*UI Validation*	Static Item Display	测试套件中对静态显示项的平均检查次数
		UI Actions/Feedback	测试套件中对界面行为/反馈的平均检查次数

- 测试环境（Test Environment）

测试环境包括了软件，硬件及管理等三个方面为测试提供的大的环境，如表 4.4 所示。

表 4.4 测试环境各影响因素

Category	Factors		Description
Test Environment	*Software Environment*	Integration with external applications	测试套件中的测试用例需要与外部应用程序集成进行测试
		Require simulators for execution	执行测试套件中的测试用例需要特殊的脚本或软件辅助模拟
		Configuration Change (DB/Configuration file)	执行测试套件中的测试用例需要修改配置并重启（如数据库的表或系统某些配置文件等）
	Hardware Environment	Network Condition	执行测试套件受网络状况的影响
		Server Equipments Performance	执行测试套件受服务器性能的影响
	Management Environment	Require Support Team due to Access Permission	执行测试套件过程中的访问系统或服务器权限的申请/解锁需要支持部门的协助
		Specific Procedure for some Test due to Company Policy	公司制度规范等对执行测试规定的一些具体流程

4.2.2 内部问卷调查

基于专家讨论的结果，我们设计了一份关于测试执行复杂度因素的调查问卷。问卷主要考察以下三个方面：

- 基于问卷中已经列出的复杂度因素示例，添加个人认为会影响测试人工花销但问卷中未列出的其他因素；

- 补充完复杂度因素后，针对问卷中每个因素选择问题“是否影响测试执行开销”(“Does it impact test execution efforts?”)的答案，包括“Yes/No”。为了让测试人员能够更准确地回答该问题，我们还提供了一组衡量标准（Measure Basis），以测试步骤（Test Steps）下面的 Number (per Test Case)为例，根据 Measure Basis 作判断，如果认为测试步骤多的（large number）比测试步骤少的(A few)会明显费时，那就选择 Yes；如果觉得步骤多和步骤少的耗时没有多少区别，那就选择 No；
- 对于选择 Yes 的项，填写每个因素相对的权重“Weight for efforts”，即每个因素相对而言对测试执行工作量花销的影响程度，用“Low, Middle, High”三个等级进行区别。权重（Weight）是用来对影响因素之间相互做比较的，即针对测试步骤（Test Steps），Large Number(per Test Case)和 High Dependency between Steps 哪个对测试执行时间影响更大，更大的填写高的权值（High），少的填写低的权值（Low）。

面向该金融软件机构内部所有测试人员，我们通过邮件发放了这一调查问卷。问卷格式如图 4.2 所示。

File Edit View Insert Format Tools Data Window Help

Test Execution Complexity Survey

Notes:
1. It focus on manual functional test
2. It is from test efforts cost perspective
3. It is on TestSuite level

Category	Factors		Measure Basis				Does it impact test execution efforts?	Weight for efforts
Test Input	*Test Steps*	Number (per Test Case)	A Few	Quite a Few	Large Number			
		Dependency Between Steps	N/A	Low	High			
	Test Data	Number (per Test Case)	A Few	Quite a Few	Large Volume/Batch File			
		Logic dependency between Data	N/A	Low	High			
Output Validation	*UI Validation*	Static Item Display	N/A	Easy	Hard			
		UI Actions/Feedback	N/A	Easy	Hard			
	Business Logic	Business Process/Calculation Rules	N/A	Easy	Normal	Complex		
		Data Volume	N/A	Single Item	Multiple Items	Large Volume		
Test Environment	*Software Environment*	Integration with external applications	N/A	Loose Coupling	High Dependency			
		Require simulators for execution	N/A	Required				
		Configuration Change: DB/Configuratic	N/A	Required				
	Hardware Environment	NetworkCondition	Bad	Good				
		Server Equipments Performance	Bad	Good				
	Management Environment	Require Support Team due to Access Permission	N/A	Required				
		Specifcial Procedure for some Test due to Company Policy	N/A	Required				

图 4.2 测试执行复杂度问卷

最后，我们共收到了 30 份回复，其中有效问卷为 27 份。具体统计结果如下

表所示，其中列出了对于每个因素是否影响测试效率选择 Yes 和 No 的人数，以及针对权重，选择 Low，Middle 和 High 的人数。

表 4.5 测试执行复杂度问卷调查统计结果

Category	Factors		Does it impact test execution efforts?		Weight for efforts		
			Yes	No	Low	Middle	High
Test Input	*Test Steps*	Number (per Test Case)	26/27	1/27	4	19	3
		Dependency Between Steps	26/27	1/27	7	8	11
	Test Data	Number (per Test Case)	26/27	1/27	4	20	2
		Logic dependency between Data	22/27	5/27	3	5	14
Output Validation	*UI Validation*	Static Item Display	22/27	5/27	10	12	0
		UI Actions/Feedback	24/27	3/27	3	10	11
System Process	*Business Logic*	Business Process/Calculation Rules	27/27	0/27	4	6	17
		Data Volume	27/27	0/27	6	7	14
Test Environment	*Software Environment*	Integration with external applications	27/27	0/27	7	8	12
		Require simulators for execution	24/27	3/27	5	17	2
		Configuration Change: DB/Configuration file	25/27	2/27	6	11	8
	Hardware Environment	Network Condition	27/27	0/27	4	10	13
		Server Equipments Performance	27/27	0/27	6	9	12
	Management Environment	Require Support Team due to Access Permission	27/27	0/27	12	7	8
		Specific Procedure for some Test due to Company Policy	22/27	5/27	15	5	2

4.2.3 确定 Guideline

从调查问卷统计结果可以看出，对于每个因素（Factor）都有超过 80%的测试人员认为其会影响测试执行效率，基于该结果，我们可以确定出测试执行复杂度所包含的影响因素列表。

对于每个影响因素的权重（Weight），可以将 Low，Middle 和 High 转换为 1，3，9 三个值，并根据选择的人数确定出最终的权值。具体公式为：

$$\text{权重 Weight} = \frac{N(L)*1 + N(M)*3 + N(H)*9}{N(L) + N(M) + N(H)} \tag{4.1}$$

其中，N(L), N(M)和 N(H)分别为选择 Low，Middle 和 High 的人数。

在通过问卷调查最终确定的影响因素列表基础上，我们再次邀请了机构内部一些经验丰富的 QA 共同来讨论并量化对每个因素打分的衡量标准（Scale，分为 N/A，Low，Average，High 四种情况，分别对应于值 0，1，2，3），从而保证在数据采集时更加客观和统一。

在该金融软件公司应用 STEEM 模型的度量测试执行复杂度 Guideline 如表 4.6 所示：

表 4.6 测试执行复杂度度量 Guideline

Category	Factors		Scale Measurement Guideline			
			N/A	Low	Average	High
Test Input	*Test Steps*	Number (per Test Case)	0	Up to 5	5-10	More than 10
		Dependency Between Steps	0	Up to 20%	20%-60%	More than 60%
	Test Data	Number (per Test Case)	0	Up to 4	4-8	More than 8
		Logic dependency between Data	0	Up to 30%	30%-70%	More than 70%
Output Validation	*UI Validation*	Static Item Display	0	Up to 10	10-20	More than 20
		UI Actions/Feedback	0	Up to 4	4-8	More than 8
System Process	*Business Logic*	Business Process/Calculation Rules	0		Need to recalculate	Need to recall
		Data Volume	0	Up to 20	20-100	More than 100
Test Environment	*Software Environment*	Integration with external applications	0	Stable	Not very stable	Extremely instable
		Require simulators for execution	0	Simple operation	Average operation	Complex operation
		Configuration Change: DB/Configuration file	0	Need to reload		Need to restart
	Hardware Environment	Network Condition	0	Local Network	VPN	Remote
		Server Equipments Performance	0	Wait time <2mins	2-10	Wait time>10mins
	Management Environment	Require Support Team due to Access Permission	0	Wait time <1day		Wait time >=1day
		Specific Procedure for some Test due to Company Policy	0	Simple	Average	Fussy

4.3 数据收集及预处理

根据上节中对测试套件执行矢量的配置，我们开始在该金融软件公司内部的项目中收集历史测试周期数据，包括：测试用例数量，测试执行复杂度，测试人员熟练程度，以及完成该部分测试套件的真实的工作量（以man-hour为单位）。

在正式开始收集数据前，我们将公司内部的某12个项目作为收集数据的目标项目，并对这些项目的测试人员进行关于STEEM模型的培训，介绍模型中测试套件执行矢量三个元素的定义和度量标准，以期收集到较统一和合理的历史数据。

到目前为止，我们面向这些项目共收集到了130组有效数据，这些数据都来自于过去一年当中的历史测试周期，并最终经过公司内部熟悉该模型的测试专家组的审核，去掉了一些不完整或有歧义的无效数据。

我们用Excel表记录这些最终确定下来作为模型训练的数据，并将对各个影响测试执行效率的复杂度因素的打分与相应的权重相乘，最后相加得到每组数据的测试执行复杂度值。

4.4 数据训练

鉴于支持向量机在进行非线性回归估算问题上的优势，我们对收集并预处理过的历史数据利用机器学习中的支持向量机方法进行训练。

通常直接从训练数据集上得到的关于模型估算精确度的分析总是很乐观的。为了得到STEEM模型的更真实的估算精确度，我们采用了与[46]中相近的过程。基于该过程，首先，我们将这130组数据分成两部分：训练数据集（Train set）和测试数据集（Test set），然后基于训练数据集进行训练并建立模型，最后在测试数据集上来分析和评估估算的精确度。

将这130组历史数据按照实际工作量时间的多少进行排序，然后从中选出所有编号是4的倍数的数据，即第4组，第8组……第128组等32组数据组成测试集，而剩下的98组数据作为模型的训练集。

本文使用了WEKA工具进行数据训练。WEKA的全名是怀卡托智能分析环境

（Waikato Environment for Knowledge Analysis），WEKA 作为一个开放的数据挖掘工作平台，集合了大量能承担数据挖掘任务的机器学习算法，包括对数据进行预处理，分类、回归、聚类、关联规则以及在新的交互式界面上的可视化[47]。

WEKA 支持的输入数据格式包括 ARFF 和 CSV 等。本文的历史数据存在 Excel 表格内，可直接另存为 CSV 格式，从而可以直接作为 WEKA 工具的输入。由于篇幅关系，本文仅列出其中的一小部分，具体如图 4.3 所示：

	A	B	C	D
1	Complexit	Case Num.	Rank	Effort
2	103	1	2	1
3	49	10	4	2
4	65	14	3	2
5	80	14	4	2
6	65	15	3	2
7	65	19	3	2
8	65	20	3	2
9	72	6	2	2.4
10	65	9	2	2.4
11	71	10	2	2.4
12	65	14	2	2.4

图 4.3 WEKA 输入文件

在 Explorer 操作环境下，提供了六种大的功能，分别是：Preprocess、Classify、Cluster、Associate、Select attributes、Visualize。具体操作过程为：

- 使用 Preprocess 下的 Open file 控件打开包含有原始的 98 组历史数据的训练文件，之后窗口上会显示该数据集的一些基本特征，如属性的个数及名称，所选属性的一些简单统计量，右下方还给出了一些可视化效果。界面可参考下图 4.4 所示；

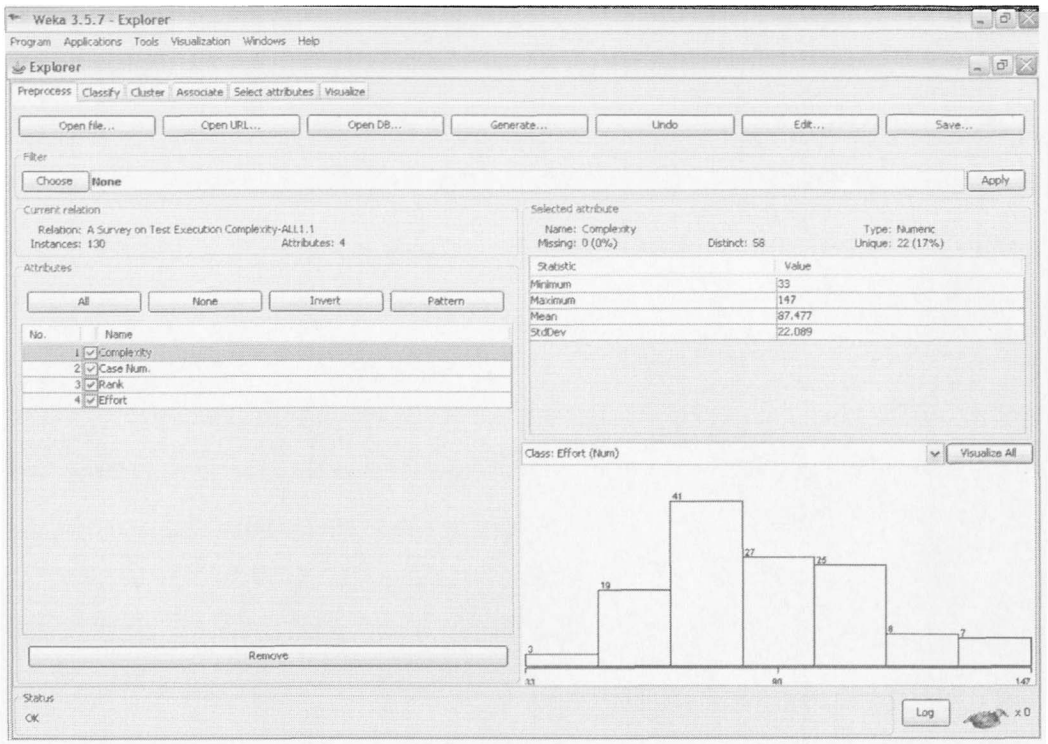

图 4.4 WEKA Explorer - Preprocess

- WEKA 把分类(Classification)和回归(Regression)都放在“Classify”选项中。在 Classify 下选择测试数据集（Test Set)。WEKA 提供了四种测试方式，分别为：直接使用训练集（Use training set)，提供专门的测试集（Supplied test set)，交叉验证（Cross-validation)，将原始训练集分割出一定比例作为测试集（Percentage split)。为了保证生成的模型的准确性而不至于出现过拟合现象，本文采用了十折交叉验证（10-fold Cross-validation）来选择和评估模型，如图 4.5 所示；

图 4.5 WEKA 测试方式选择

- 在 Classify 下选择具体的算法进行数据训练。在该实例分析中，支持向量机的算法采用 SMOreg，具体参数和核函数通过反复实验调整至最优，具体为：

```
weka.classifiers.functions.SMOreg -S 0.0020 -C 20.0 -T 0.0010 -P
1.0E-12 -N 0 -K "weka.classifiers.functions.supportVector.Puk -C
250007 -O 2.5 -S 2.5"
```

- 点击“Start”即开始数据训练，结果将在窗口中显示，可参考图 4.6。实验结果为具体为：

Correlation coefficient	0.9234
Mean absolute error	2.1635
Root mean squared error	3.2465
Relative absolute error	33.1025 %
Root relative squared error	38.7605 %
Total Number of Instances	98

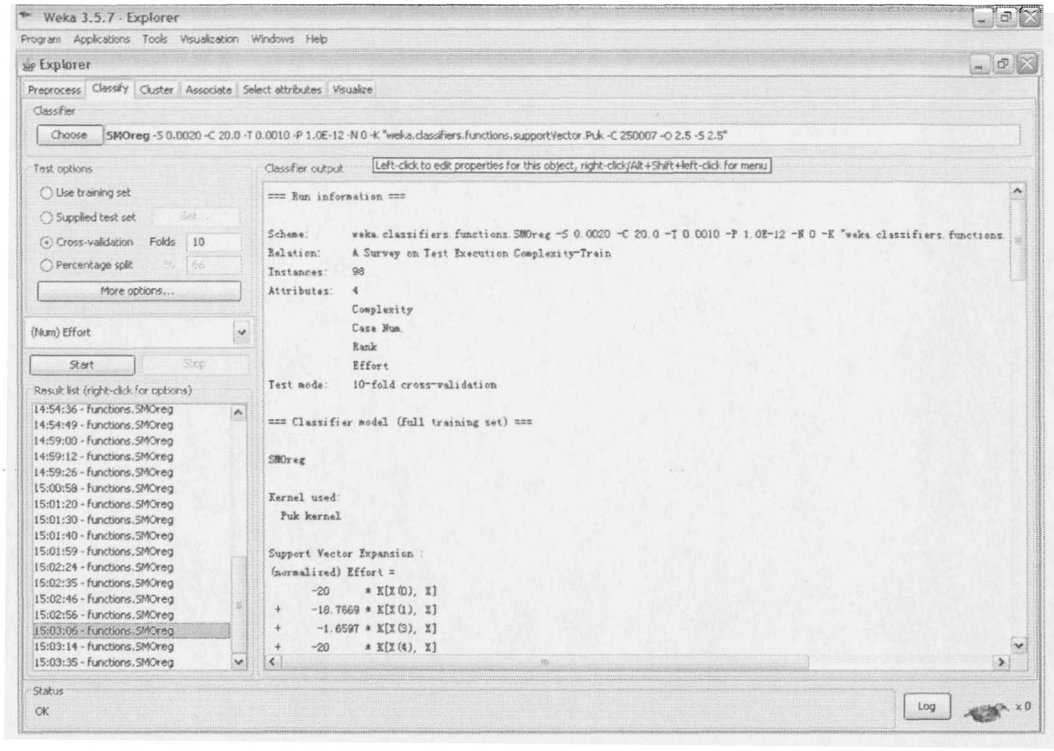

图 4.6 WEKA 训练结果

4.5 STEEM 模型估算精确度分析

4.5.1 评价标准

在实际中，有很多的方法来评价软件工作量估算模型的估算精确度。本文使用了 MAE，MMRE 和 PRED(k)等三种方法来分析 STEEM 模型估算的合理程度[48] [49] [50]。

- 平均绝对估算误差 MAE

平均绝对估算误差（Mean absolute estimation error，简称 MAE），计算公式为：

$$MAE = \frac{\sum_{i}^{T} abs\left(Estimated_i - Actual_i\right)}{T} \tag{4.2}$$

其中，

$T = NumberOfTests$

- 平均相对估算误差 MMRE

平均相对估算误差(Mean magnitude of the relative estimation error，简称 MMRE)，是评估软件估算模型性能最常用的方法之一。

$$MMRE = \frac{\sum_{t=1}^{T} MRE_t}{T} \tag{4.3}$$

其中，

$$MRE_t = abs(\frac{Estimated_t - Actual_t}{Actual_t})$$

$T = NumberOfTests$

- MRE 值在某个范围 k 以内的估算平均百分比

Average percentage of estimates that were within k of the actual values，表示为 PRED(k)。

$$PRED(k) = \frac{\sum_{t=1}^{T} (1, ifMRE_t \le k, 0, otherwise)}{T} \tag{4.4}$$

其中，变量含义同 MMRE。

基于这些估算准确度评价标准，我们可以从两个方面来考察 STEEM 模型的估算性能。

- 使用我们的 STEEM 模型，是否能从整体上得到一个较小误差
- 当使用 STEEM 模型时，误差在真实值 25%/40%范围内的估算数量是否有所提高

4.5.2 实验结果分析

为了对模型估算精确度进行更加全面的分析，本文将 32 组测试数据集平均分成了两组，然后分别利用 STEEM 模型作了两次实验，包括：

Experiment1：以 32 组数据中的奇数组作测试集；

Experiment2：以32组数据中的偶数组作测试集；

表4.7列出了用这三种精确度评价标准对实验结果的分析，可以看出，我们的模型并不像很多模型那样强烈依赖于测试集的数据，STEEM模型对于不同的测试数据集都得到了较好的一致结果，并且MAE维持在2 man-hours，这个误差对于实际工作中估算测试执行工作量来说，是完全可以接受的。超过75%的估算结果都满足了PRED(40)，这在工作量估算模型上，也是一个较大的突破。

表4.7 STEEM模型实验结果分析表

实验名称	MAE	MMRE	PRED(25)	PRED(40)
10折交叉验证	2.16	33.10%	N/A	N/A
Experiment1	2.05	35.58%	75.0%	87.5%
Experiment2	2.45	37.85%	56.3%	75.0%

4.6 本章小结

本章以某金融软件公司为背景对本模型做了实例研究，以此来评估模型的性能。我们结合该公司的实际情况和特点介绍了如何配置测试套件执行矢量的三个元素，以及如何在公司和机构内部收集历史经验数据，如何利用支持向量机方法进行数据训练等，并对最终得到的模型进行了估算精确度分析，取得了较好的估算效果。

第5章 STEEM在具体项目测试管理中的应用实例

在某金融软件公司内部搭建好 STEEM 模型之后，我们即可使用该模型来对公司内部的软件项目的测试工作提供一定的参考和帮助。本章介绍了在该软件公司内部的一个实际项目 QA Cycle 中应用 STEEM 模型进行测试执行工作量估算，并根据估算结果来制定测试计划，分配测试资源的过程。

5.1 实验项目背景介绍

FIXRouting 基于第三方软件实现，是关于 FIX(Financial Information eXchange) 消息协议的路由系统，为该金融机构的客户（Customer）和经纪人（Broker）之间提供交易消息的转化和路由。系统功能如图 5.1 所示：

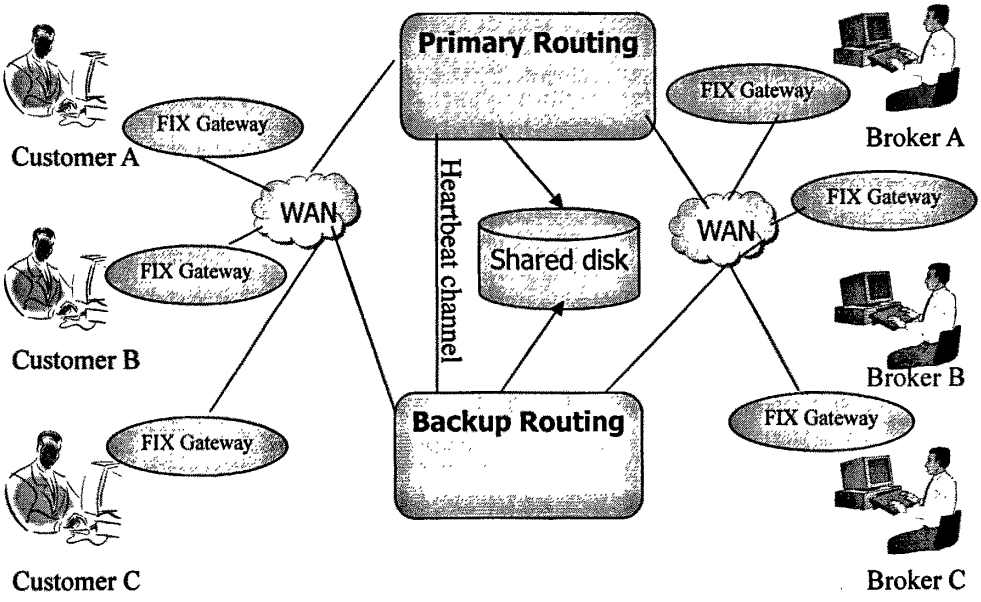

图 5.1 FIXRouting 系统功能图

FIXRouting 包含了三个实例（instance）：EQ-FIXRouting，FX-FIXRouting 和 FO-FIXRouting，分别为该机构的股票（Equity），外汇（Foreign eXchange）及期

货期权（Future and Option）交易提供消息路由（Routing）和转换（Translation）服务。其中，第三方软件 TT 由 ttConnect，ttMonitor，ttAlert 三个模块组成，而 EQ-FIXRouting，FX-FIXRouting 和 FO-FIXRouting 的转换和路由功能均是基于 ttConnect 来实现的。此外，在 EQ-FIXRouting 上，基于 ttAlert 实现了 RTDM，BV，AIM 和 OM 等四个模块，用来辅助实现系统的一些特殊需求。详见系统结构图 5.2。

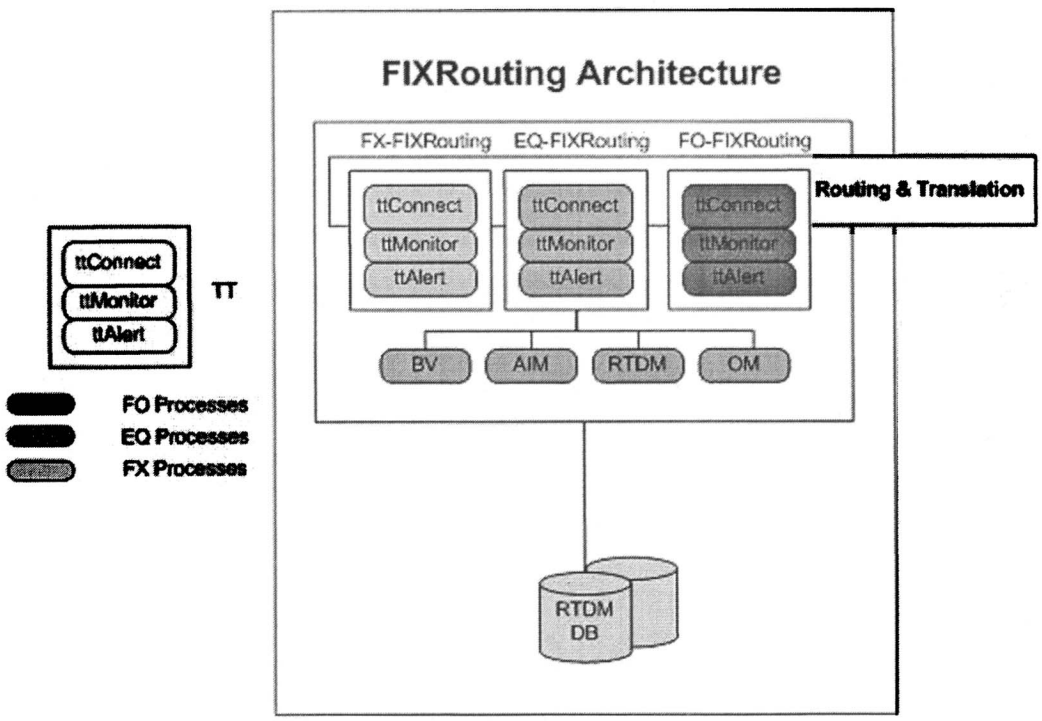

图 5.2 FIXRouting 系统结构图

5.2 模型应用

由上节介绍可知，整个 FIXRouting 项目都是基于第三方软件 TT 来实现的。所以，如果 TT 升级到新的版本，并且我们的项目要采用这个新的版本的话，就需要做一个完整的回归测试（Regression Testing）来验证系统的功能是否正常，这个测试周期通常称为 TT Upgrade QA Cycle。它将涵盖系统所有的 instance，周

边模块等，测试范围和涉及到的测试套件数量都非常大，这对测试经理准确估算这个QA Cycle所需要的时间开销提出了较大的挑战。

本文以FIXRouting项目的实际TT Upgrade QA Cycle为背景，讨论如何使用STEEM模型来实现测试计划和资源分配等测试管理工作。

5.2.1 确定测试范围

确定测试范围（Test Scope），主要依据需求文档以及该版本具体做了哪些变动，一般可以根据release notes来判断。FIXRouting的TT Upgrade QA Cycle的测试范围包括：

- TT GUI自身的回归测试（TT GUI Regression Test）

  测试是否能够通过TT的web页面正确配置某些属性，重启系统后是否能够生效，能否正确创建新的Connection，Notification Rule，用户/角色权限是否正确等。涉及到的测试套件为：TT_GUI_Regression_Test_Case.xls，共包含252个测试用例；

- EQ-FIXRouting
  - EQ-FIX Routing：已利用simulator实现自动化测试
  - EQ-FIX Translation：同上
  - RTDM Testing：涉及到的测试套件为：RTDM_Test_Case.xls，共包含122个测试用例
  - OM Testing：涉及到的测试套件为：OM_Test_Case.xls，共包含56个测试用例
  - AIM Testing：涉及到的测试套件为：AIM_Test_Case.xls，共包含32个测试用例
  - BV Testing：涉及到的测试套件为：BV_Test_Case.xls，共包含19个测试用例
- FO-FIXRouting
  - FO-FIX Routing：已利用simulator实现自动化测试
  - FO-FIX Translation：同上

- FX- FIXRouting
  - FX-FIX Routing&Translation for NG：涉及到的测试套件为：FX over FIX_NG_Test_Case.xls，共包含 102 个测试用例
  - FX-FIX Routing&Translation for Legacy：涉及到的测试套件为：FX over FIX_Legacy_Test_Case.xls，共包含 92 个测试用例

5.2.2 明确测试人员

在大多数软件机构中，测试组及测试人员可能被多个项目共享。因此，在开始 QA Cycle 之前，需要先明确哪些测试人员可以参与此次测试。针对 TT Upgrade QA Cycle，有三位测试人员，我们用测试员 A，B 和 C 来指代。根据测试人员熟练程度的定义，我们可以针对不同的测试套件对 A，B，C 的级别进行划分。

5.2.3 工作量估算

确定好测试范围及测试人员之后，我们可以开始利用 STEEM 模型进行工作量估算。以下将以 TT Upgrade QA Cycle 为例来具体介绍使用模型估算的过程。

➢ 分析测试范围中涉及到的各个测试套件的测试复杂度

主要包括功能性回归测试，对于已经实现自动化测试的部分，其工作量开销已固定，并不在本文讨论范围之内。测试复杂度分析结果如图 5.3 所示：

➢ 针对每个测试套件使用模型进行工作量估算

我们的原则是，以每个测试套件为考察对象，然后估算三个测试人员 A，B，C 分别单独执行该套件需要多少 man-hour。估算结果如表 5.1-5.7 所示：

File Edit View Insert Format Tools Data Window Help

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1														
2	Factors		Scale				Weight	TT GUI Regression	AIM	BV	OM	RTDM	FX-FIXRouting Legacy	FX-FIXRouting NG
3	*Test Steps*	Number (per Test Case)	N/A	L	M	H	3	3	1	1	3	2	1	1
4		Dependency Between Steps	N/A	L	M	H	5	3	3	2	3	2	3	3
5														
6	*Test Data*	Number (per Test Case)	N/A	L	M	H	3	1	2	1	2	2	2	2
7		Logic dependency between Data	N/A	L	M	H	7	0	1	1	1	1	2	2
8														
9	*UI Validation*	Static Item Display	N/A	L	M	H	2	2	1	2	2	1	2	2
10		UI Actions/Feedback	N/A	L	M	H	6	3	2	2	3	2	1	1
11														
12	*Business Logic*	Business Process/Calculation	N/A	L	M	H	6	3	1	2	3	1	2	3
13		Data Volume	N/A	L	M	H	6	0	1	1	3	2	1	1
14														
15	*Software Environment*	Integration with external applications	N/A	L	M	H	5	3	2	0	0	0	3	3
16		Require simulators for execution	N/A	L	M	H	3	1	2	2	2	2	3	3
17		Configuration Change: DB/Configuration file	N/A	L	M	H	4	0	0	0	0	2	0	0
18														
19	*Hardware Environment*	Network Condition	N/A	L	M	H	6	3	3	2	1	1	3	3
20		Server Equipments Performance	N/A	L	M	H	5	0	2	0	0	0	2	2
21														
22	*Management Environment*	Require Support Team due to Access Permission	N/A	L	M	H	4	0	0	0	0	3	3	3
23		Specifcial Procedure for some Test due to	N/A	L	M	H	2	0	0	0	0	0	0	0
24														
25	Test Execution Complexity							103	101	75	107	93	130	136

图 5.3 TT Upgrade QA Cycle 各测试套件执行复杂度

表 5.1 TT GUI Regression 测试工作量估算值

测试套件名称	测试人员	测试套件执行矢量			工作量估算结果（man-hour）
		测试用例数量	测试执行复杂度	测试人员熟练程度	
TT_GUI_Regression_Test_Case	A	252	103	4	29
	B	252	103	2	32
	C	252	103	1	44

表 5.2 RTDM 测试工作量估算值

测试套件名称	测试人员	测试套件执行矢量			工作量估算结果（man-hour）
		测试用例数量	测试执行复杂度	测试人员熟练程度	
RTDM_Test_Case	A	122	93	4	23
	B	122	93	4	23
	C	122	93	1	24

表 5.3 OM 测试工作量估算值

测试套件名称	测试人员	测试套件执行矢量			工作量估算结果（man-hour）
		测试用例数量	测试执行复杂度	测试人员熟练程度	
OM_Test_Case	A	56	107	2	7
	B	56	107	2	7
	C	56	107	1	8

表 5.4 BV 测试工作量估算值

测试套件名称	测试人员	测试套件执行矢量			工作量估算结果（man-hour）
		测试用例数量	测试执行复杂度	测试人员熟练程度	
BV_Test_Case	A	19	75	2	6
	B	19	75	4	3
	C	19	75	1	6

表 5.5 AIM 测试工作量估算值

测试套件名称	测试人员	测试套件执行矢量			工作量估算结果（man-hour）
		测试用例数量	测试执行复杂度	测试人员熟练程度	
AIM_Test_Case	A	32	101	2	8
	B	32	101	4	4
	C	32	101	1	8

表 5.6 FX over FIX NG 测试工作量估算值

测试套件名称	测试人员	测试套件执行矢量			工作量估算结果（man-hour）
		测试用例数量	测试执行复杂度	测试人员熟练程度	
FXoverFIX_NG_ Test_Case	A	102	136	4	18
	B	102	136	2	21
	C	102	136	1	22

表 5.7 FX over FIX Legacy 测试工作量估算值

测试套件名称	测试人员	测试套件执行矢量			工作量估算结果（man-hour）
		测试用例数量	测试执行复杂度	测试人员熟练程度	
FXoverFIX_ Legacy_ Test_Case	A	92	130	4	14
	B	92	130	2	17
	C	92	130	1	18

➢ 分析估算结果并分配测试资源

根据上述估算结果，我们可以看出，对于 RTDM 和 OM 测试部分，不同测试人员所需的工作量时间大致相同，而对于其他测试，不同熟练程度的测试人员所需时间产生了较大差别，因此，本着以尽量减少测试周期时间跨度和平均分配测试人员工作量的原则，来对测试人员进行分工，结果如表 5.8 所示：

表 5.8 测试工作分配结果

测试人员	测试任务	工作量估算结果总和（man-hour）
A	FXoverFIX_Legacy_Test_Case	32
	FXoverFIX_NG_Test_Case	
B	TT_GUI_Regression_Test_Case	39
	BV_Test_Case	
	AIM_Test_Case	
C	RTDM_Test_Case	32
	OM_Test_Case	

最终，并行执行完所有测试，B 所需的时间总和比其他两个人多 7 个小时，因此，我们可以安排 C 在完成自己的测试任务后来辅助完成 B 剩下的工作，而对于对整个项目和测试工作都比较熟悉的测试人员 A 来说，可以用多出来的 7 个小时来完成测试结果汇总，测试报告及更新测试用例等工作。

5.2.4 制定测试计划

根据工作量估算的最终结果，我们可以制定测试计划。在本实例应用中，完成整个 TT 升级测试周期需要 39 个小时。WBS 可参考图 5.4。

	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	TT Upgrade QA Cycle	######	Mon 08-5-5	Fri 08-5-9		
2	FXoverFIX_Legacy_Test_Case	18 hrs?	Mon 08-5-5	Wed 08-5-7		A
3	FXoverFIX_NG_Test_Case	14 hrs?	Wed 08-5-7	Thu 08-5-8	2	A
4	TT_GUI_Regression_Test_Case	32 hrs?	Mon 08-5-5	Thu 08-5-8		B
5	BV_Test_Case	3 hrs?	Fri 08-5-9	Fri 08-5-9	4	B
6	AIM_Test_Case	4 hrs	Fri 08-5-9	Fri 08-5-9	5	B,C
7	RTDM_Test_Case	24 hrs?	Mon 08-5-5	Wed 08-5-7		C
8	OM_Test_Case	8 hrs?	Thu 08-5-8	Thu 08-5-8	7	C
9	Update Test Case and Design T	7 hrs?	Fri 08-5-9	Fri 08-5-9	3	A

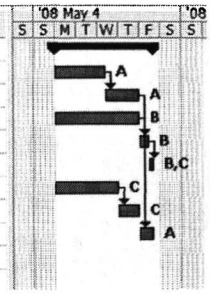

图 5.4 TT Upgrade QA Cycle WBS 图

5.3 对软件测试管理的帮助和意义

通过使用 STEEM 模型可以让测试经理更加合理地对测试执行工作量进行估算，尤其是在比较复杂的项目中，需要测试的功能模块很多，并且拥有的测试人员熟练程度也有一定的差异，这个时候使用该模型，就可以更好地解决资源分配和计划的问题，使得完成一个测试周期的时间跨度尽可能的短。

在收集历史数据的过程中，我们计算过收集每组数据所需的平均时间，大约为 7 分钟左右，以 FIXRouting 项目的 TT Upgrade QA Cycle 为例，需要收集关于这 7 个测试套件的数据，差不多需要为 50 分钟，然后再利用 STEEM 模型进行估算和计划，最后总共需要 1 个小时。能够在 1 个小时以内完成这样一个较复杂的测试周期的测试计划和资源分配工作，还是相当有效的。

5.4 本章小结

本章通过在该公司中某具体项目的实际测试周期中使用该模型，来阐述如何利用该模型来估算各个测试套件的执行工作量，并以此来合理制定测试计划和安排测试资源等，从而提高测试质量。介绍了一个完整的软件测试管理过程，包括确定测试范围、明确测试人员、工作量估算和制定测试计划等。

第6章 总结及未来工作

6.1 总结和讨论

测试作为保证软件质量的重要手段，对于软件机构来说至关重要。而在实际情况中，测试工作执行时间经常超出之前安排的进度，有时候甚至到了交付的最终期限还没有完成测试从而被迫投入市场，产品的质量低下为机构造成了恶劣的影响。其中，对测试执行工作量的估算不足是造成这种局面的重要原因之一。

本文对已有的工作量估算模型进行了全面仔细的研究，发现单纯对测试执行工作量进行估算的方法和模型非常贫乏，因此为了更好地解决测试估算问题，本文提出了一种基于经验的软件测试执行工作量估算模型 STEEM。到目前为止，我们对 STEEM 模型进行了详细的介绍，并在某金融软件公司内部使用该模型进行了实例分析和研究，取得了较好的估算效果。下面对本文中涉及到的一些重要问题进行讨论和分析。

为什么针对测试执行工作量估算？如第二章中介绍的，测试作为一个系统的工作包括测试计划，测试管理，测试设计，测试执行和测试报告等步骤。测试计划和测试设计可以与开发计划，设计和编码并行进行，测试报告可以在 bug 修复阶段完成。而只有测试执行不能与任何的软件开发工作并行进行，这使得测试工作成为了软件生命周期关键路径的一个重要部分。因此，合理准确地估算测试执行工作量可以更好地对整个项目时间表进行管理。

如何得到了测试套件执行矢量的定义？在分析已有的测试执行工作量估算模型时，我们发现基于单独的测试用例和测试步骤来估算的话，有着较大的时间开销。如果使用测试套件作为考察单元，会极大地降低使用模型进行估算的时间开销。在真实的项目测试管理当中，测试经理通常会把测试套件作为分配测试工作的单元。因此我们将测试套件作为了 STEEM 模型估算的考察对象。关于测试用例数量，依据当前的研究[35]和测试经验，我们相信测试套件中的测试用例数量与需求的规模紧密相关。为了得到更加合理的估算结果，我们使用了测试执行复杂

度作为对测试用例数量的补充。对测试人员熟练程度的考虑可以帮助进行测试工作的分配。在理想情况下，一旦我们知道了每个测试人员在每个测试套件上的执行时间，就可以得到关于测试执行工作安排的最优解。我们甚至可以在软件需求分析阶段就使用测试套件执行矢量进行测试执行工作量估算。现有的另一个挑战就是我们用来度量测试执行复杂度的方法仍然比较主观，有待改进。

使用STEEM模型估算的时间开销如何？我们的方法所需的时间开销主要分为两部分：一个是建立模型时间，包括准备测试执行复杂度评估标准，建立历史数据库，选择SVM算法和参数等；另一个则是在真实的测试周期中为了估算执行工作量而准备测试套件执行矢量的各个值。这两部分的开销都不是很大。我们花了大约3个星期的时间来准备和设计调查问卷，然后分析问卷，并设计出最后的测试执行复杂度评估标准。主要的时间花在了评估测试复杂度，并为建立历史数据库收集测试套件执行矢量值上，大约花了2个星期的时间收集到了130组历史数据。我们记录了收集每组数据所需的时间，平均算下来准备每组数据大约需要7分钟左右。这样的话，在每个测试周期中，准备测试套件执行矢量的时间是完全可以接受的。

总的来说，我们的模型时间开销较小，引入了合理的测试套件执行矢量模型，并应用了支持向量机方法得到了较好的估算效果，可以说在测试执行工作量估算上是非常有效的。对测试人员熟练程度的度量，使得模型估算结果更加精确合理，并有助于测试经理进行测试计划和资源分配工作。

6.2 不足与展望

如前面所讨论的，现在的STEEM模型中也存在了一些需在未来改进的部分：

- 对模型进行进一步的研究，使其可以在项目早期如需求分析阶段就可以度量出测试套件执行矢量中的三个元素，并进行估算；
- 重新修订测试执行复杂度评估标准，使其更加客观和通用；
- 分析研究估算的算法，以提高估算精确度；
- 扩展历史数据库，并在跨机构/公司范围中进行验证。

参考文献

[1] S.P. Ng, T. Murnane, K. Reed, D. Grant, T.Y. Chen. A preliminary survey on software testing practices in Australia. Software Engineering Conference, 2004. Proceedings. 2004 Australian, 2004:116–125

[2] The Standish Group. 1995. CHAOS Chronicles, Standish Group Internal Report, 1995. http://www.standishgroup.com

[3] The Standish Group. 2004. 2004 the 3rd Quarter Research Report, 2004. http://www.standishgroup.com

[4] 李明树, 何梅, 杨达, 舒风笛, 王青. 软件成本估算方法及应用. 软件学报, 2007, 18(4):775–795

[5] I.F. Barcelos Tronto, J.D. Silva, N. Sant'Anna. 2006. The artificial neural networks model for software effort estimation.
http://mtc-m18.sid.inpe.br/col/sid.inpe.br/ePrint%4080/2006/12.20.23.38/doc/v1.pdf

[6] 朱芳, 李曦, 赵振西. 一种多平台自动化测试工具的设计和实现. 计算机工程, 2004, 30(24):186–188

[7] 曹小鹏. 嵌入式软件的测试方法研究. 西安邮电学院学报, 2007, 12(5):92–94

[8] 曹文静, 宫云战. 软件测试性计算方法研究. 计算机工程与设计, 2003, 24(10):67–70

[9] Myers, Glenford J. The art of software testing. New York: Wiley, 1979:177

[10] Hetzel, William C. The Complete Guide to Software Testing, 2nd ed. Wellesley, Mass.: QED Information Sciences, 1988:280

[11] Tran, Eushiuan. Verification/Validation/Certification. Topics in Dependable Embedded Systems, 1999

[12] Laycock, G. T. The Theory and Practice of Specification Based Software Testing. Dept of Computer Science, Sheffield University, 1992:2

[13] 古乐, 史九林. 软件测试技术概论. 北京: 清华大学出版社, 2004:105

[14] Rex Black(天宏工作室). 测试流程管理. 北京: 北京大学出版, 2001:50–55

[15] B. Boehm, C. Abts, S. Chulani. Software development cost estimation approaches - a survey. Ann. Software Eng., 2000, 10:177–205

[16] M. Jorgensen, M. Shepperd. A systematic review of software development cost estimation studies. IEEE Transactions on Software Engineering, 2007, 33(1):33–53

[17] K. Molokken, M. Jorgensen. A review of surveys on software effort estimation. Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003:223–230

[18] Y Ahn, J Suh, S Kim, H Kim. The software maintenance project effort estimation model based on function points. Journal of Software Maintenance and Evolution: Research and Practice, 2003, 15(2):71–85

[19] Function Point FAQ
http://ourworld.compuserve.com/homepages/softcomp/fpfaq.htm, 2007

[20] International Function Point Users Group (IFPUG). Function Point Counting Practices Manual, Release 4.1.1, 2002.

[21] C. Symons. Software Sizing and Estimating. New York: John Wiley &Sons, 1991: 207–209

[22] Common Software Measurement International Consortium. COSMIC-FFP Version 2.0 2000. http://www.cosmicon.com/.

[23] Vernon V. Chatman III. Change points: A proposal for software productivity measurement. Journal of Systems Software, 1995, 31(1):71–91

[24] Barry J. Boehm. Software engineering economics. Englewood Cliffs, NJ: Prentice-Hall, 1981

[25] Barry J. Boehm. Software Cost Estimation with COCOMO II (with CD-ROM). Englewood Cliffs, NJ: Prentice-Hall, 2000

[26] E. Aranha, P. Borba. An Estimation Model for Test Execution Effort. Empirical Software Engineering and Measurement, 2007:107–116.

[27] 杜云梅, 李师贤, 孙恒. 构造性成本模型 COCOMO. 计算机科学, 2005, 32(12):106-109

[28] G. Schneider, J. P. Winters. Applying Use Cases: A Practical Guide, 2nd ed., New York: Addison Wesley, 2001

[29] Shinji Kusumoto, Fumikazu Matukawa, Katsuro Inoue. Estimating Effort by Use

Case Points: Method, Tool and Case Study. Proceedings of the 10th International Symposium on Software Metrics (METRICS'04), 2004:292 – 299

[30] Mohagheghi P, Anda B, Conradi R. Effort estimation of use cases for incremental large-scale software development. In: Proc of the 27th Int'l Conf. on Software Engineering. St Louis, 2005:303–311

[31] Carroll ER. Estimating software based on use case points. In: Proc. of the Conf. on Object Oriented Programming Systems Languages and Applications. New York: ACM Press, 2005:257–265

[32] 谢益辉. 回归分析：起源、理论以及应用概述. http://www.cos.name/, 2006

[33] Gujarati DN. Basic Econometrics. 3rd ed.. New York: McGraw Hill, 1995

[34] Suresh Nageswaran. Test Effort Estimation Using Use Case Points. Quality Week 2001, San Francisco, California, USA, 2001

[35] Capers, Jones. Applied software measurement. New York: McGraw-Hill, 1996

[36] P. Jorgensen. Software Testing, A Craftsmans Approach. 2nd ed.. Boca Raton: CRC Press, 2002

[37] S. Kishore and R. Naik. 软件需求与估算. 北京: 机械工业出版社, 2004

[38] Steve McConnell. 快速软件开发. 北京: 机械工业出版社, 2003

[39] A. Aamodt, E. Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. AI Communications, IOS Press, 1994, 7(1):39-59

[40] L. Briand, K. E. Emam, S. Morasca. On the application of measurement theory in software engineering. Empirical Software Engineering: An International Journal, 1996, 1(1):61–88

[41] N. Fenton. Software measurement: A necessary scientific basis. IEEE Transactions on Software Engineering, 1994, 20(3):199–206

[42] C. Pandian. Software Metrics: A Guide to Planning, Analysis, and Application. Boca Raton: CRC Press, Inc., 2003

[43] IEEE standard for software test documentation. New York: IEEE. ISBN 0-7381-1443-X

[44] 姜德峰, 杜子平. 基于支持向量机的利率期限结构研究. 现代管理科学, 2007,

1(1):106-108

[45] 郭丽娟, 孙世宇, 段修生. 支持向量机及核函数研究. 科学技术与工程, 2008, 8(2):487-490

[46] P. Sentas, L. Angelis, I. Stamelos, G. Bleris. Software productivity and effort prediction with ordinal regression. Journal Information and Software Technology, 2005, 47(1):17-29

[47] Weka入门教程. 2007. http://www.ieee.org.cn/

[48] Colin J. Burgess, Martin Lefley. Can genetic programming improve software effort-A comparative evaluation. Information and Software Technology, 2001, 43(14): 863-873

[49] Emilia Mendes, Nile Mosley. Further Investigation into the Use of CBR Regression to Predict Development Effort for Web Hypermedia Applications. Proceedings 2002 International Symposium on Empirical Software Engineering (ISESE'02), 2002:79-90

[50] Lioel C. Briand, Khaled EL Emam, Dagmar Surmann, Isabella Wieczorek, Katrina Maxwell. An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques. International Software Engineering Research Network technical Report ISBRN-98-27, 1999:313-322

致谢

时光飞逝，日月如梭，转眼两年的研究生学习生涯马上就要过去了。在此论文完成之际，我首先要衷心感谢我的导师杨小虎老师。杨老师严谨的治学态度、渊博的知识理论、和蔼可亲的态度以及诲人不倦的谆谆教诲都让我铭记难忘。此外，我还要感谢浙大道富技术中心的孙建伶老师、李善平老师、蔡亮老师、周波老师和黄忠东老师，谢谢老师们为我们精心营造的开放自由的研究环境，在这里，大家相互交流、相互帮助、相互鼓励，共同进步。

我要感谢祝小春博士。谢谢师兄对我论文研究工作的指导和帮助，在实验室期间与师兄一起研究学习教给了我许多思考问题和做研究的方法。

同时我也要感谢道富技术中心质量保证组的所有成员和我的朋友们，谢谢你们对我研究和学习的支持。大家在我遇到困难的时候帮我排忧解难，鼓励我度过难关，我仅在此表示衷心的感谢！

感谢我的父母，在我多年的求学过程中，无微不至地关心、照顾着我，谢谢你们给我的爱。

感谢我的男朋友杨超，无论我碰到多大的阻挠和挫折，你都一直支持、鼓励、帮助着我，你的理解和信任让我更加坦然的面对一切困难。

最后，感谢评阅、评议论文和答辩委员会的各位专家学者在百忙的工作中能给予指导。再次谢谢你们！

侯 莉

2008年5月于浙大求是园

作者简历

本人侯莉，于 2002.10-2006.06 就读于浙江大学计算机学院计算机科学与技术专业，2006 年获得学士学位，并被保送为浙江大学计算机学院计算机应用技术专业硕士研究生。于 2006.09-2008.06 在浙江大学就读硕士研究生期间，学习勤奋努力，认真钻研计算机专业知识，学习成绩名列前茅。在浙江大学道富技术中心研究学习期间，积极参与所在项目的开发和研究，注重知识与实践的有效结合，不断地提高自己的科研创新能力。并于 07 年暑假前往美国道富公司总部接受了为期三个月的培训，涉及软件工程方法学，软件质量保证方法学，软件开发过程控制及证券交易等各方面知识，承担了金融信息路由系统及外汇交易系统等多个项目的质量保证及软件开发过程控制工作。