第3章 软件项目管理

软件工程研究的主要内容包括："软件工程技术"和"软件项目管理"两部分。经验表明，许多软件项目的失败并非由于软件工程师技术水平差，而主要是因为管理水平低。在经历了失败之后，人们逐渐认识到了软件项目管理的重要性和特殊性。本章将集中介绍与软件项目管理有关的内容。

学习目标：了解软件项目管理的特殊性和重要性，了解软件项目管理的常用技术活动内容。重点理解定量估算、进度控制、质量保证和配置管理活动内容。

3.1 软件项目管理概述

软件项目管理的基础内容大部分来自于工程管理学科的相关技术，而软件生产的特殊性，也使得对软件项目的管理有别于其他工程项目，所以在引用相关学科的技术或方法的同时，也要重视软件生产的特殊性。

3.1.1 软件生产的特殊性

软件开发活动是知识密集型脑力劳动，具有以下特性：

(1) 软件是逻辑产品，具有高度的抽象性；

(2) 需求相同的软件，可以有多种设计和实现方案，属创造性劳动；

(3) 软件生产过程复杂，无形的劳动过程难以管理，生产过程和产品质量不易控制；

(4) 软件开发和维护依据用户需求进行，其过程具有复杂性和易变性；

(5) 软件开发和运行受计算机系统环境的限制，存在安全性和移植性问题；

(6) 软件开发技术更新快，对软件工程管理也是一种挑战。

3.1.2 软件项目管理的重要性

软件工程本质上是一个复杂的系统工程，所以软件过程综合了多学科的相关技术和方法。经验与教训表明，软件项目的成败很大程度上并非技术问题，而是管理水平造成的。

软件工程诞生之前，软件开发是"个人英雄主义"时代，一个人包揽全部任务与活动，工程管理的因素没有显现出来。但是随着软件规模的不断扩大，复杂性不断提高，软件开发成为团队集体创造性的脑力劳动过程。集体劳动就离不开计划、组织、协调与控制等管理活动，复杂的软件工程活动更是有赖于工程管理的成功。软件危机促使软件工程学科的诞生充分说明了这一点。解决软件危机问题，有赖于软件工程技术和软件项目管理技术的不断发展与创新。

3.1.3 软件项目管理的主要内容

软件项目管理是对与软件过程相关的技术活动和资源要素进行组织、协调、监督与控制的一系列活动，包括成本估算、人员组织安排、项目计划与进度控制、软件配置管理、软件质量保证等一系列活动。项目管理先于任何技术活动，而且贯穿整个生命周期。通过项目管理，合理地配置和使用各种资源，以保证软件质量和开发过程的效率。

软件成本控制主要基于对项目规模的估算，从而做出对开发人员、项目开发周期和经费的估算。软件过程控制包括进度控制、需求变更控制、经费控制和质量保证。由于软件产品的特殊性和软件工程的不成熟，制订准确的软件进度计划比较困难，对于比较复杂和规模稍大的项目，通常执行“整体规划，分期实施”的策略。例如，分为一期工程、二期工程等，然后再制订各期工程的具体计划，这样才能保证计划切实可行，便于控制。

软件工程管理很大程度上是通过对文档管理来实现的。因此，要为开发过程设计一套完整的文档资料。文档标准化是文档管理的一个重要方面。

软件配置管理和软件质量保证是贯穿整个生命周期的重要软件工程活动，是保证各项软件工程活动协调一致，最终获得满足用户需求的高质量的软件产品的基础。

3.2 软件规模估算

软件项目管理从项目计划开始，制订计划的基础是工作量估算，工作量的估算又是基于对软件规模的估算。为了估算项目的工作量和工期，首先要估算软件的规模。

早期，软件规模很小，估算误差影响不大。随着软件规模的不断扩大和软件需求越来越复杂，软件开发的费用在计算机应用系统中所占比例越来越大，软件成本估算的准确性成了影响软件项目成败的关键因素。软件规模估算成为软件工程管理技术的一项重要内容。

以下介绍两种软件规模估算技术和估算方法。

3.2.1 代码行技术

代码行是用以表示软件规模的一个单位。代码行技术依据以往开发类似产品的经验和历史数据，估计实现一个功能所需要的源程序行数。当有以往开发类似产品的历史数据可供参考时，用这种方法估计出的数值是比较准确的。把实现每个功能所需要的源程序行数累加起来，就可得到实现整个软件所需要的源程序行数。

可以组织多名有经验的软件工程师分别做出估计。每个人都估计程序的最小规模(a)、最大规模(b)和最可能的规模(m)，分别算出这 3 种规模的平均值和之后，再用下式计算程序规模的估计值：

$$L = \frac{\bar{a} + 4\,\bar{m} + \bar{b}}{6} \tag{3.1}$$

用代码行技术估算软件规模时，当程序较小时用代码行数(LOC)表示规模，当程序较大时用千行代码数(KLOC)表示。

代码行技术的主要优点是代码是所有软件开发项目都有的构成，容易计算代码行数。

缺点主要有以下三方面：

(1) 源程序仅是软件配置的一个成分，用它代表整个软件的规模似乎不太合理；

(2) 用不同语言实现同一个软件所需要的代码行数并不相同；

(3) 这种方法不适用于非过程语言。

表 3.1 是用代码行技术估算软件规模的一个例子。对软件各项功能需要的代码行数和开发成本分别估算。每行成本与开发活动的复杂性及工资水平有关，估算后填入表中。然后计算出估计的工作量和软件成本。每项功能的工作量(人·天)等于代码行数除以每人每天可完成的行数；每项功能的成本等于代码行数乘以每行成本。最后，分别计算工作量的合计和成本合计。

表 3.1　代码行估算技术的例子

功　能	估计需要代码行数	按经验(行/人·天)	每行成本(元)	该项成本(元)	工作量(人·天)
获取实时数据	840	92			9.1
更新数据库	1212	122			9.9
脱机分析	600	134			4.5
生成报告	450	145			3.1
实时控制	1120	80			14
合计					40.6

3.2.2　功能点技术

为了克服代码行技术的缺点，提出了功能点技术。功能点技术依据对软件信息域特性和软件复杂性的评估结果估算软件规模。这种方法用功能点(FP)为单位度量软件规模。

1. 信息域特性

功能点技术定义了信息域的 5 个特性：输入项数(Inp)、输出项数(Out)、查询数(Inq)、主文件数(Maf)和外部接口数(Inf)。

(1) 输入项数。用户向软件输入的项数，这些输入给软件提供面向应用的数据。输入不同于查询，后者单独计数，不计入输入项数中。

(2) 输出项数。软件向用户的输出项目，它们向用户提供面向应用的信息，例如报表和出错信息等，报表内的数据项不单独计数。

(3) 查询数。查询是一次联机输入，它导致软件以联机输出方式产生某种即时响应。每一个不同的查询都要计数。

(4) 主文件数。逻辑主文件是数据的一个逻辑组合，它可能是大型数据库的一部分或是一个独立的文件。

(5) 外部接口数。对所有用来将信息传送到另一个系统中的接口(即磁带、磁盘和可读写光盘上的数据文件)均应计数。

2. 估算功能点的步骤

用以下 3 个步骤，可估算出一个软件的功能点数，即软件规模。

(1) 计算未调整的功能点数

首先，把产品信息域的每个特性(即 Inp、Out、Inq、Maf 和 Inf)都分类为简单级、平均级

或复杂级三个级别，并根据其等级为每个特性分配一个功能点数。例如，一个简单级的输入项分配 3 个功能点，一个平均级的输入项分配 4 个功能点，而一个复杂级的输入项分配 6 个功能点。然后，用下式计算未调整的功能点数：

$$\mathrm{UFP} = a_1 \times \mathrm{Inp} + a_2 \times \mathrm{Out} + a_3 \times \mathrm{Inq} + a_4 \times \mathrm{Maf} + a_5 \times \mathrm{Inf} \tag{3.2}$$

其中，$a_i(1 \leqslant i \leqslant 5)$是信息域特性系数，其值由相应特性的复杂级别决定。

(2) 计算技术复杂性因子 TCF

度量 14 种技术因素对软件规模的影响程度。这些因素包括高处理率、性能标准(如响应时间)、联机更新等。根据软件特点，为每个因素分配一个从 0～5 的值(0 表示不存在或对软件规模无影响，5 表示有很大影响)，然后用下式计算技术因素对软件规模的综合影响：

$$\mathrm{DI} = \sum_{i=1}^{14} F_i \tag{3.3}$$

技术复杂性因子 TCF 由下式计算：

$$\mathrm{TCF} = 0.65 + 0.01 \times \mathrm{DI} \tag{3.4}$$

因为 DI 的值在 0～70 之间，所以 TCF 的值在 0.65～1.35 之间。

(3) 计算功能点数 FP

用下式计算功能点数：

$$\mathrm{FP} = \mathrm{UFP} \times \mathrm{TCF} \tag{3.5}$$

注：功能点数与所用的编程语言无关，看起来功能点技术比代码行技术更合理一些。但是，在判断信息域特性复杂级别和技术因素的影响程度时，存在较大的主观因素。

3.2.3 估算方法

估算是凭经验做出的推算，要求技术人员有项目经验做基础。以往成功的软件工程项目中的相关数据，是软件估算的参考，可以凭借经验公式推算出当前软件项目的规模和费用。

常用的估算方法有自顶向下的方法、自底向上的方法、差别估算法、专家估算法、类推估算法、算式估算法等。

1. 自顶向下的估算方法

估算人员参照以前完成的项目所耗费的总成本或总工作量，来推算当前项目的总成本或总工作量，然后按阶段、步骤和工作单元进行分解，这种方法称为自顶向下估算法。

自顶向下估算方法的主要优点是对系统级工作重视，不会遗漏系统级活动，如集成、测试、配置管理等，估算任务的工作量小，速度快。

该方法的缺点是往往不清楚较低层工作的技术性细节，这会增加估算的难度和估算结果的准确性。

2. 自底向上的估算方法

将每一部分的估算工作交给负责该部分的人员来做，然后汇总。它的优点是估算较为准确，缺点是系统级的工作量估算不够。实践中，常以自顶向下和自底向上两个方法结合使用。

3. 差别估算方法

差别估算是将开发项目与已成功完成的同类项目进行比较，找出若干不同之处，并估算每个不同之处对成本的影响，从而推导出开发项目的总成本。该方法的优点是可以提高估算的准确度，缺点是不容易明确差别的界限。

4. 专家估算法

依靠一个或多个专家对项目进行估算，其精确性取决于专家对估算项目的定性参数的了解和他们的经验。

5. 类推估算法

自顶向下的方法中，将估算项目与类似项目进行直接比较；自底向上的方法中，在具有相似条件的相同活动之间进行比较。

6. 算式估算法

专家估算法和类推估算法的缺点是具有盲目性和主观猜测性，不同的人估算的结果可能会有很大的偏差。算式估算法企图避免主观因素的影响，用于估算的方法有“由理论导出”和“由经验导出”两种类型。

3.3 软件项目工作量估算

软件估算模型使用由经验导出的公式来预测软件开发工作量。工作量是软件规模(KLOC 或 FP)的函数，工作量的单位通常是人月(pm)。支持大多数估算模型的经验数据都是从有限个项目的样本集中总结出来的，因此，没有一个估算模型可以适用于所有类型的软件和开发环境。

一般都是用经验公式来预测工作量，分为三类：静态单变量模型、动态多变量模型和构造性成本模型。

3.3.1 静态单变量模型

静态单变量模型分为面向 KLOC 的估算模型和面向 FP 的估算模型，这类模型的总体结构形式如下：

$$E = A + B \times (ev)^C \tag{3.6}$$

其中：A、B 和 C 是由经验数据导出的常数，E 是以人月为单位的工作量，ev 是估算变量(KLOC 或 FP)。

下面给出几个典型的静态单变量模型。

1. 面向 KLOC 的估算模型

(1) Walston_Felix 模型

$$E = 5.2 \times (\mathrm{KLOC})^{0.91} \tag{3.7}$$

(2) Bailey_Basili 模型

$$E = 5.5 + 0.73 \times (\mathrm{KLOC})^{1.16} \tag{3.8}$$

(3) Boehm 简单模型

$$E = 3.2 \times (\mathrm{KLOC})^{1.05} \tag{3.9}$$

(4) Doty 模型(在 KLOC>9 时适用)

$$E = 5.288 \times (\mathrm{KLOC})^{1.047} \tag{3.10}$$

2. 面向 FP 的估算模型

(1) Albrecht & Gaffney 模型

$$E = -13.39 + 0.0545\mathrm{FP} \tag{3.11}$$

(2) Maston、Barnett 和 Mellichamp 模型

$$E = 585.7 + 15.12\mathrm{FP} \tag{3.12}$$

从上面列出的模型可以看出,对于相同的 KLOC 或 FP 值,用不同模型估算将得出不同的结果。主要原因是,这些模型多数都是仅根据若干应用领域中有限个项目的经验数据推导出来的,适用范围有限。因此,必须根据当前项目的特点选择适用的估算模型,并且根据需要适当地调整估算模型(如修改模型常数)。

3.3.2 动态多变量模型

动态多变量模型也称为软件方程式,它是根据从 4000 多个当代软件项目中收集的生产率数据推导出来的。该模型把工作量看做是软件规模和开发时间这两个变量的函数。

动态多变量估算模型的形式如下:

$$E = (\mathrm{LOC} \times B^{0.333}/P)^3 \times (1/T)^4 \tag{3.13}$$

其中,E 是以人月或人年为单位的工作量;

T 是以月或年为单位的项目持续时间;

B 是特殊技术因子,它随着对测试、质量保证、文档及管理技术的需求的增加而缓慢增加,对于较小的程序(KLOC=5～15),B=0.16,对于超过 70 KLOC 的程序,B=0.39;

P 是生产率参数,它反映了下述因素对工作量的影响:

- 总体过程成熟度及管理水平;
- 使用良好的软件工程实践的程度;
- 使用的程序设计语言的级别;
- 软件环境的状态;
- 软件项目组的技术及经验;
- 应用系统的复杂程度。

开发实时嵌入式软件时,P 的典型值为 2000;开发电信系统和系统软件时,P=10 000;对于商业应用系统来说,P=28 000。可以从历史数据导出适用于当前项目的生产率参数值。从式(3.13)可以看出,开发同一个软件(即 *LOC* 固定)的时候,如果把项目持续时间延长一些,则可降低完成项目所需的工作量。

3.3.3 COCOMO2 模型

COCOMO 是构造性成本模型(Constructive Cost Model)的英文缩写。1981 年 B. Boehm 在《软件工程经济学》中首次提出了 COCOMO 模型。1997 年 B. Boehm 等人提出的 COCOMO2 模型,是原始的 COCOMO 模型的修订版,它反映了十多年来在成本估计方面所积累的经验。

COCOMO2 给出了 3 个层次的工作量估算模型,对软件细节考虑的详尽程度逐级增

加。这些模型既可以用于不同类型的项目，也可以用于同一个项目的不同开发阶段。这3个层次的估算模型分别是：

(1) 应用系统组成模型。主要用于估算构建原型的工作量，模型名字暗示在构建原型时大量使用已有的构件。

(2) 早期设计模型。适用于体系结构设计阶段。

(3) 后体系结构模型。适用于完成体系结构设计之后的软件开发阶段。

下面以后体系结构模型为例介绍COCOMO2模型。该模型把软件开发工作量表示成代码行数(KLOC)的非线性函数：

$$E = a \times \mathrm{KLOC}^b \times \prod_{i=1}^{17} f_i \tag{3.14}$$

其中，E是开发工作量(以人月为单位)，a是模型系数，KLOC是估计的源代码行数(以千行为单位)，b是模型指数，f_i(i=1～17)是成本因素。

每个成本因素都根据它的重要程度和对工作量影响大小被赋予一定数值(称为工作量系数)。这些成本因素对任何一个项目的开发工作量都有影响，即使不使用COCOMO2模型估算工作量，也应该重视这些因素。

为了确定模型指数b的值，原始COCOMO模型把软件开发项目划分成组织式、半独立式和嵌入式3种类型，并指定每种类型对应的b值(分别是1.05、1.12和1.20)。COCOMO2采用了更加精细得多的b分级模型，这个模型使用5个分级因素W_i($1 \leqslant i \leqslant 5$)，其中每个因素都划分成从甚低($W_i$=5)到特高($W_i$=0)的6个级别，然后用下式计算$b$的数值：

$$b = 1.01 + 0.01 \times \sum_{i=1}^{5} W_i \tag{3.15}$$

因此，b的取值范围为1.01～1.26。显然，这种分级模式比原始COCOMO模型的分级模式更精细、更灵活。

COCOMO2使用的5个分级因素描述如下：

(1) 项目先例性。这个分级因素指出，对于开发组织来说该项目的新奇程度，诸如开发类似系统的经验，需要创新体系结构和算法，以及需要并行开发硬件和软件等因素的影响，都体现在这个分级因素中。

(2) 开发灵活性。这个分级因素反映出，为了实现预先确定的外部接口需求及为了及早开发出产品而需要增加的工作量。

(3) 风险排除度。这个分级因素反映了重大风险已被消除的比例。在多数情况下，这个比例和指定了重要模块接口(即选定了体系结构)的比例密切相关。

(4) 项目组凝聚力。这个分级因素表明了开发人员相互协作时可能存在的困难。这个因素反映了开发人员在目标和文化背景等方面相一致的程度，以及开发人员组成一个小组工作的经验。

(5) 过程成熟度。这个分级因素反映了按照能力成熟度模型度量的过程成熟度。

在原始的COCOMO模型中，仅粗略地考虑了前两个分级因素对指数b之值的影响。

工作量方程中模型系数a的典型值为3.0，在实际工作中应该根据历史经验数据确定一个适合本组织当前开发的项目类型的数值。

3.4 软件项目进度控制

一个大的项目往往可以分解为若干工序，根据施工要求，各工序前后衔接有序。项目的总工期由各工序衔接中最长的一条路径决定，这个路径被称为“关键路径”。非关键路径上的工序如果没有严重拖后，就不会影响整个项目的完成时间，而处于关键路径之中的工序如果进度拖后，则整个项目的完成日期就会拖后，管理人员应该高度关注关键任务的进展情况。

项目管理者的目标是定义全部项目工序，识别出关键工序，跟踪关键工序的进展状况，以保证能及时发现拖延进度的情况。为达到上述目标，管理者必须制定一个足够详细的进度表，以便监督项目进度并控制整个项目。

3.4.1 估算开发时间

估算出完成给定项目所需的总工作量之后，接下来估算总工期。对于一个估计工作量为 20 人月的项目，可能想出下列几种进度表：1 个人用 20 个月完成该项目；4 个人用 5 个月完成该项目；20 个人用 1 个月完成该项目。但是，这些进度表并不现实，实际上软件开发时间与从事开发工作的人数之间并不是简单的反比关系。

通常，成本估算模型也同时提供了估算开发时间 T 的方程。与工作量方程不同，各种模型估算开发时间的方程很相似，例如：

(1) Walston_Felix 模型

$$T = 2.5E^{0.35} \tag{3.16}$$

(2) 原始的 COCOMO 模型

$$T = 2.5E^{0.38} \tag{3.17}$$

(3) COCOMO2 模型

$$T = 3.0E^{0.33+0.2\times(b-1.01)} \tag{3.18}$$

(4) Putnam 模型

$$T = 2.4E^{1/3} \tag{3.19}$$

其中，E 是开发工作量(以人月为单位)，T 是开发时间(以月为单位)。

用上列方程计算出的 T 值，代表正常情况下的开发时间。客户往往希望缩短软件开发时间，显然，为了缩短开发时间应该增加从事开发工作的人数。但是，经验告诉我们，随着开发小组规模扩大，个人生产率将下降，证明开发时间与人数并非反比例关系。

3.4.2 Gantt 图

Gantt(甘特)图是历史悠久、应用广泛的制定进度计划的工具，下面引用一个非常简单的例子介绍这种工具。假设有一个旧木板房，屋内有四面墙需要油漆。这项工作必须分 3 步完成：①首先刮掉旧漆，②然后刷上新漆，③最后清除溅在窗户上的油漆。假设一共分配了 15 名工人，然而工具却有限：只有 5 把刮旧漆用的刮板，5 把刷漆用的刷子，5 把清除溅在窗户上的油漆用的小刮刀。怎样安排才能使工作进行得更有效呢？

一种做法是：①首先刮掉四面墙壁上的旧漆；②然后给每面墙壁都刷上新漆；③最后清除溅在每个窗户上的油漆。显然，这是效率最低的做法，因为总共有 15 名工人，然而每种

工具却只有 5 件，这样安排工作，在任何时候都有 10 名工人闲着没活干。

采用“流水作业法”是比较高效的，相信大部分人都会这么做：①先由 5 名工人用刮板刮掉第 1 面墙上的旧漆，这时其余 10 名工人休息。②当第 1 面墙刮净后，前 5 名工人转去刮第 2 面墙上的旧漆；与此同时，另外 5 名工人立即开始用刷子给第 1 面墙刷新漆。③一旦刮旧漆的工人转到第 3 面墙而且刷新漆的工人转到第 2 面墙以后，余下的 5 名工人立即拿起刮刀去清除溅在第 1 面墙窗户上的残漆，……

各工序所需工作量不同。假设木板房的第 2、4 两面墙的长度比第 1、3 两面墙的长度长一倍，此外，不同工作需要用的时间长短也不同，刷新漆最费时间，其次是刮旧漆，清理残漆需要的时间最少。表 3.2 列出了估计每道工序需要用的时间。

表 3.2　各道工序估计需用的时间（小时）

工序 / 墙壁	刮旧漆	刷新漆	清理
1 或 3	2	3	1
2 或 4	4	6	2

可以使用图 3.1 中的 Gantt 图描绘上述流水作业过程：在时间为零时开始刮第 1 面墙上的旧漆；两小时后刮旧漆的工人转去刮第 2 面墙，同时另 5 名工人开始给第 1 面墙刷新漆；每当给一面墙刷完新漆之后，第 3 组的 5 名工人立即清除溅在这面墙窗户上的漆。

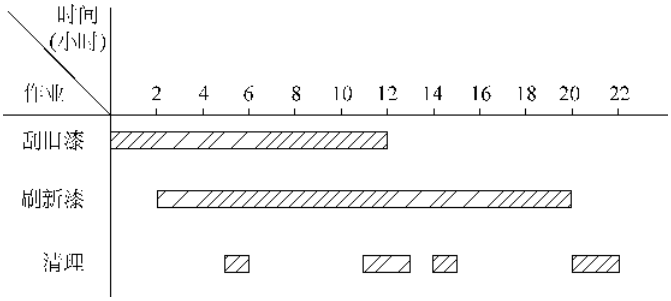

图 3.1　旧木板房刷漆工程的 Gantt 图

从图 3.1 可以看出 12 小时后刮完旧漆，20 小时后完成刷漆工作，再过 2 小时后清理工作结束。全部工程在 22 小时后结束，如果用前述的第一种做法，则需要 36 小时。

Gantt 图能很形象地描绘任务分解情况，以及每个工序的开始时间和结束时间，因此是进度计划和进度管理的有力工具。它具有直观简明、容易掌握、容易绘制的优点，但是 Gantt 图也有明显不足：

（1）不能显式地描绘各项作业彼此间的依赖关系；

（2）进度计划的关键部分不明确，难于判定哪些部分应当是主攻和主控的对象；

（3）计划中有潜力的部分及潜力的大小不明确，往往造成潜力的浪费。

3.4.3　工程网络

工程网络是制订进度计划时另一种常用的图形工具，它同样能描绘任务分解情况以及每项工序的开始时间和结束时间，此外，它还显式地描绘各个作业彼此间的依赖关系。

在工程网络中用箭头表示工序，例如刮旧漆、刷新漆、清理残漆等，用圆圈表示事件，即

一项工序开始或结束。事件仅仅是可以明确定义的时刻，它并不消耗时间和资源。工序通常既消耗资源又需要持续一段时间。图 3.2 是旧木板房刷漆工程的工程网络。

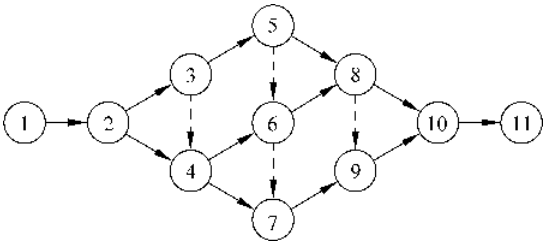

图 3.2　旧木板房刷漆工程的工程网络

图中表示刮第 1 面墙上旧漆的工序开始于事件“1”，结束于事件“2”。用开始事件和结束事件的编号标识一个工序，因此“刮第 1 面墙上旧漆”的工序是“1—2”。

在工程网络中的一个事件，如果既有箭头进入又有箭头离开，则它既是某些作业的结束又是另一些作业的开始。例如，图 3.2 中事件“2”既是工序“1—2”的结束，又是工序“2—3”（刮第 2 面墙上旧漆）和工序“2—4”（给第 1 面墙刷新漆）的开始。也就是说，只有第 1 面墙上的旧漆刮完之后，才能开始刮第 2 面墙上旧漆和给第 1 面墙刷新漆这两个工序。因此，工程网络显式地表示了工序之间的衔接顺序关系。

在图 3.2 中还有一些虚线箭头，它们表示虚拟工序，即事实上并不存在的作业。引入虚工序是为了强调工序之间的依赖关系。例如，事件“4”既是给第 1 面墙刷新漆结束，又是给第 2 面墙刷新漆开始（工序“4—6”），但是，在开始给第 2 面墙刷新漆之前，不仅必须已经给第 1 面墙刷完了新漆，而且第 2 面墙上的旧漆也必须已经刮净，即事件“3”结束。也就是说，在事件“3”和事件“4”之间有依赖关系，或者说在工序“2—3”（刮第 2 面墙上旧漆）和工序“4—6”（给第 2 面墙刷新漆）之间有依赖关系。虚工序“3—4”明确地表示了这种依赖关系。

3.4.4　工程进度估算

画出工程网络之后，需要在工程网络上增加一些必要的信息，用于估算工程进度。首先，把每个工序估计需要的时间写在该项工序的箭头上方；其次，为每个事件计算下述两个统计数字：最早时刻 EET 和最迟时刻 LET。

事件的最早时刻 EET，也是后一工序的最早开工时间，事件的最迟时刻 LET，也是前一工序的最迟结束时间。

计算最早时刻 EET 使用下述 3 条简单规则：

（1）考虑进入该事件的所有作业；

（2）对于每个作业都计算它的持续时间与起始事件的 EET 之和；

（3）选取上述和数中的最大值作为该事件的最早时刻 EET。

按照这种方法，不难沿着工程网络从左至右顺序算出每个事件的最早时刻，计算结果标在图 3.3 的工程网络中每个圆圈内右上角的数字。

计算最迟时刻 LET 使用下述 3 条规则：

事件的最迟时刻是在不影响工程竣工时间的前提下，该事件最晚可以发生的时刻。按惯例，最后一个事件的最迟时刻就是它的最早时刻。其他事件的最迟时刻在工程网络上从右至左按逆工序流的方向计算。

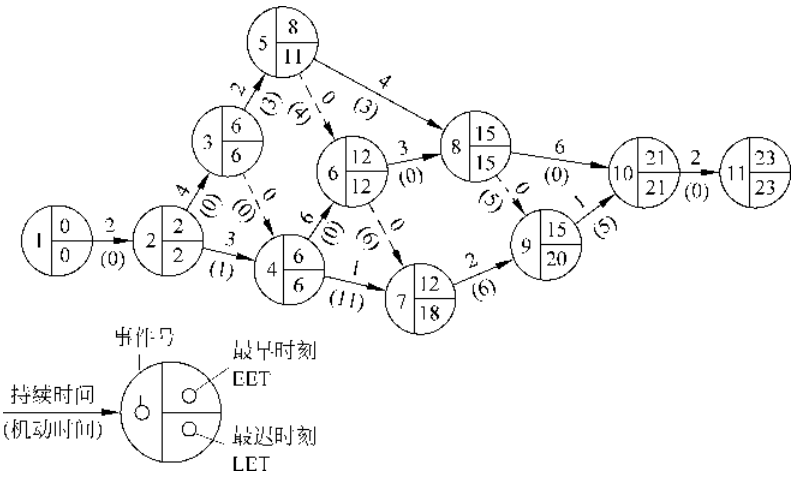

图 3.3　旧木板房刷漆工程的完整的工程网络

(1) 考虑离开该事件的所有作业；

(2) 从每个作业的结束事件的最迟时刻中减去该作业的持续时间；

(3) 选取上述差数中的最小值作为该事件的最迟时刻 LET。

图 3.3 中每个圆圈内右下角的数字就是该事件的最迟时刻。

3.4.5　关键路径

图 3.3 中有几个事件的最早时刻和最迟时刻相同，这些事件定义了关键路径。关键路径上的事件必须准时发生，否则工程就不能按时完工。

工程项目的管理人员应该密切注视关键作业的进展情况，如果关键事件出现的时间比预计的时间晚，则会使最终完成项目的时间拖后。如果希望缩短工期，只有往关键工序增加资源才会有效果。

3.4.6　机动时间

不在关键路径上的工序有一些机动余地，即实际开始时间可以比预定时间晚一些，或者实际持续时间可以比预定的持续时间长一些，而并不影响整个工程的工期。

一个工序可以有的全部机动时间等于它的“结束事件”的最迟时刻减去它的“开始事件”的最早时刻，再减去这个工序的持续时间：机动时间＝$(LET)_{结束}$－$(EET)_{开始}$－持续时间。对于油漆旧木板房的例子，非关键工序的机动时间列在表 3.3 中。

表 3.3　旧木板房刷漆工程非关键工序机动时间

工序	LET	EET	持续时间	机动时间
2—4	6	2	3	1
3—5	11	6	2	3
4—7	18	6	1	11
5—6	12	8	0	4
5—8	15	8	4	3
6—7	18	12	0	6