Part 1

THE SURVIVAL MIND-SET

第一篇

求生心态

第 1 章

欢迎来到软件项目求生训练中心

许多软件项目的生存机会看来不大，其实不然。想要生存的第一步是确定以良好的方式开始进行。有好的开始，生存的机会自然就大得多了。

4

你可能很难相信，一般人对软件产品的要求要比软件项目严格多了。使用软件的人希望软件产品可以连续使用好几个钟头，可以连续执行好几百万个程序命令而历久弥新。可是软件开发者对软件项目反而不抱太大期望。使用者与客户也许会抱怨项目慢了一个月、三个月甚至半年才推出，也许抱怨程序不好用或缺乏几项重要功能。可是如果软件产品计划中的主体如期推出，即使不惜血本，大部分消费者还是会认为开发产品的项目成功了。我们看过太多失败的例子，所以我们认为只有完全像是扶不起的阿斗的项目才算是失败的。

多年来，软件工业的高层人士对软件项目的要求总是爱之深、责之切。一个成功的项目应该尽可能满足成本与时间的需求，以追求高质量的产品为目标，不要瞻前顾后。确定了这点，就现阶段的技术还可以控制在百分之十上下的水准。一般的软件项目主管都可以做得到，即使是项目外的其他人，如高阶主管、经理、客户、投资人和使用者代表一样可以发挥相同影响力。

一名Construx Software Builders的首席软件工程师请我去看他们一些失败的案例。在专家眼里，失败的原因通常很明显。中型软件项目的失败（20 000~250 000行源

代码）其实很容易避免。此外我发现软件项目不是不能达到最短时程、最低成本、最佳质量或任何其他目标择一力臻完美。

并非以上所有目标都能同时完成，本书想要告诉大家的是力求在众多目标之间取得平衡，让一个低成本而高质量的产品能如期推出。

求生需求

软件项目求生第一步就是确认生存的基本需要。Abraham Maslow观察出人类的需求依照程度由低到高，以自然阶层的形态呈现。最低程度的需求称做“生理需求”，这是人类生存所必需的最低要求。在我们满足上层的需求前，必须先满足图1-1中在虚线以下部分的较低程度需求。所以要先满足对食物、空气、水的生理需求之后，我们才能够追求“归属感”与爱自尊与自我实现的满足。

如同许多软件专家一样，我发现类似的需求阶层也可以套用在软件项目上。软件项目有一组基本需求必须先被满足，逐步攀爬到需求金字塔的上层，就可以大幅改善项目的质量与生产力。

项目团队必须满足“一定会完成项目”的最低层次需

求，接着再来考虑有关时间和预算目标百分之十上下的问题。而且项目小组必须在有限的预算和时间之内，以现有的技术水准，努力推出预定计划中的最佳软件。

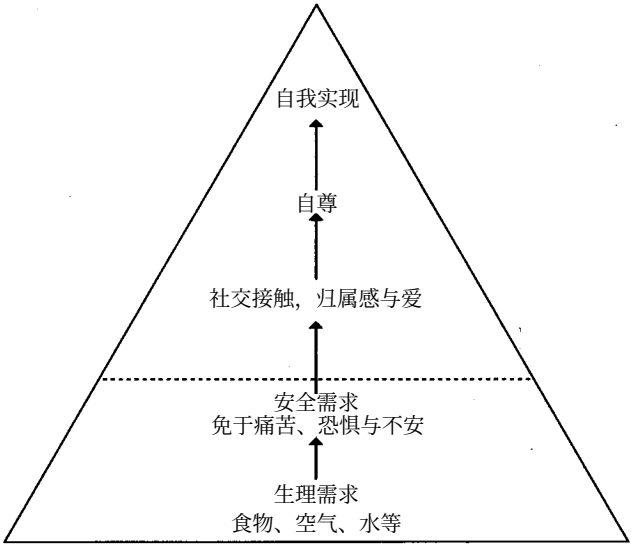


图1-1　Abraham Maslow的人类需求阶层图。在满足较高阶层的需求前，必须先满足较低阶层的需求

如图1-2所描述的，软件项目的需求阶层与制作项目的个人需求大相径庭。举例来说，开发人员会将他们的个人自尊摆在健全的团队动力之前。但就项目而言，健全的团队动力要比开发人员个人自尊更重要。

本书针对软件项目需求中下阶层的部分讨论，只有当上层需求的方向影响下层需求的满足时，才会提到上一阶层的部分。

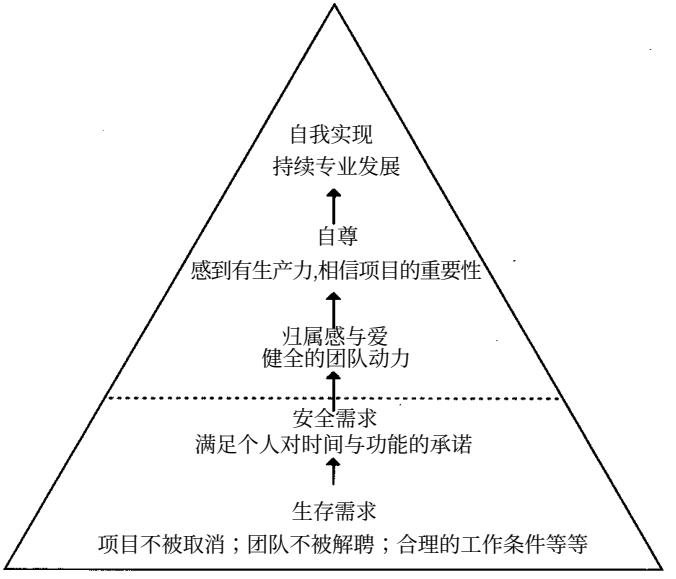


图1-2 软件项目需求阶层图。项目的需求大致与参与项目的个人相仿

求生权利

处境艰难的项目威胁到各相关人士的生存需求。客户担心项目到底能不能推出，结果会不会太慢或太贵。

主管担心客户会不会取消项目而导致失败，或者开发人员能不能够完成。开发人员担心他 (她)会不会丢掉饭碗，或是被迫牺牲数百小时的休闲时间表示他 (她)真的全心投入工作了。每一种情形，每个人都退回到项目需求阶层的最底层部分 —— 担忧是否能满足他们个人承诺的安全需求。这样的反应反而让他们放弃追求金字塔上层可达成最高质量与生产力的东西。

如汤玛斯·富比世（Thomas Hobbes）在十七世纪所作的观察，17世纪时，专制制度下生活的人孤僻、穷困、难受、粗野而短命。软件项目求生的第一步就是让各相关人员能同意彼此以文明的方式对待。总结出一个结果，当文明的规范套用到软件项目上时，就成了“客户的人权法案”。当项目不是针对特定客户而开发时，本来属于客户的权利就会成为产品主管、行销代表或使用者代表所有。

客户的人权法案

我有以下权利:
1. 设定项目目标，并让项目朝目标进行。
2. 了解软件项目要花费的时间与成本。

(续)

3. 决定产品中要有哪些功能。
4. 在项目进行中对需求做合理改变，并了解进行改变所需的代价。
5. 清楚可靠地了解项目进行的状态。
6. 经常被告知会影响成本、时间表或质量的风险所在，并获得解决基本问题的选择权。
7. 可探知项目过程中完成的部分。

各种权利中，我最了解的就是美国人权法案中列举的各项权利。那些权利不只是“权利”而已，而是民主代议制度的基本要求。同样这些软件的项目权利不只是让软件项目的推动更有乐趣，而是让项目运作的必要条件。

与美国人权法案相似，个人在软件项目中拥有的权利也是他在项目中所需负起的义务。我们都享有言论自由，不过我们对言论自由的义务就是让别人也有言论自由的权利，即使我们不认同他人的言论或认为他人的言语侵犯了我们的权利。在一个软件项目中，客户的权利来自尊重下列这些项目小组的权利：

项目小组的人权法案

我有以下权利：

1. 了解项目目标，并理清各顺序不明的目标之间的优先度。
2. 了解我要建立的产品细节，并理清产品定义不明之处。
3. 随时让客户、主管、行销人员或其他负责决策的人能探知软件功能已完成的部分。
4. 以技术上负责任的方式进行项目中各阶段的工作，特别是不因压力而提早进行项目中的程序写作。
5. 估计被要求完成工作所需的努力与时间。这项权利包含只提供项目各阶段理论上约略可行的时间与成本类型，与争取建立有意义的估量所需要的时间，以及项目需求变化时重新测算各项时间与成本的权利。
6. 将项目的推动状态明确报告给客户与上层主管得知。
7. 获得一个免于被打扰而分心的生产工作环境，特别是在进行项目的关键部分时。

项目成功的第一步在于让相关各方尊重彼此对项目成功的权利。第二步则在于让项目彻底满足各方人士生存需求，使每个人都感受不到生存上的威胁。至于怎么做，就是本书接下来要讨论的了。

求生检查

每一章结尾都有个求生检查，列出你能从项目外进行

检查的项目特性，以衡量项目的健全与否。

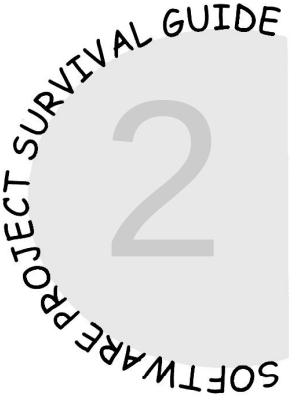
成功的关键以楷体字表示，标上一个大拇指的符号(👍)，显示这些特性的项目在朝着成功方向前进。陷阱则以炸弹的符号(💣)标示，有这些特性的项目就危险了。

求生检查

- 👍 项目的求生需求获得满足。
- 👍 客户与项目小组同意尊重彼此对软件项目的权利。
  - 💣 原则上同意尊重彼此的权利，实际上却没做到。

第 2 章

软件项目求生测验



一个简短的测验可以测出软件项目的健全程度。如果测验指出项目处于危险状态，你可以采取一些拉高积分的步骤来改善这种情形。

14

本章提供一个测验用来评估项目成功的机率有多大。项目能不能如期在预算内完成，结果会超出预期还是大失所望，都可以从以下测验中看出来。

求生测验

需求

_____ 1. 项目中有清楚、不含糊的着眼点或任务叙述吗？

_____ 2. 所有团队成员都相信项目中提到的着眼点是实在的吗？

_____ 3. 项目中有提到获利细节与如何计算获利的部分吗？

_____ 4. 项目中有使用者接口雏形来详细示范实际系统拥有的功能吗？

_____ 5. 项目中有没有软件该有的详细明文规格？

_____ 6. 项目小组从项目初期是否就与未来可能实际应用该软件的人交换意见，并在过程中持续了解他们的反应？

规划

_____ 7. 项目有无详细明文规划过？

_____ 8. 项目的工作条款中是不是包含安装程序，并将资料从早期的系统中更新，与其他软件进行整合，和客户面谈，及其他“琐碎”项目？

_____ 9. 接近完成的阶段时是否有很正式的重估时间与预算成本？

_____ 10. 项目有无详细的架构与说明文件？

_____ 11. 项目有无一套详细明文纪录表格，除了系统测试以外，还要求设计与实作相互检验的品质确保计划？

_____ 12. 项目有无一套详细的阶段性软件完成规划流程，说明各阶段实作与完成的软件部分？

_____ 13. 项目规划中是否事先保留例假日、成员出勤与训练所需的时间是否一并考虑在内？

_____ 14. 项目规划中是否包含由开发团队、品质确认团队与技术写作团队 —— 这些负责进行工作的人们所认可的时间表？

项目控制

_____ 15. 项目是否仅有一名拥有决策权限的主管，并充分获得全力支持？

_____ 16. 项目主管的工作负担是否仍有余力让他（她）兼顾项目？

_____ 17. 项目是否有清楚明确的完成点，来观察执行成果是否完成或未完成（这样的完成点称作“二元完成点”）？

_____ 18. 你能简单找出已经完成的各个“二元完成点”吗？

_____ 19. 项目有无供成员能够评论主管与上层主管好坏的匿名渠道？

_____ 20. 项目中有无明文规定来规范产品变更规格？

_____ 21. 项目中有无变动控制布告栏，用来决定该不该接受被提出来的变更项目？

_____ 22. 项目中有无对所有团队成员公开的信息规划，包含努力度与时间估计、任务指派与预期进度检查？

_____ 23. 所有源代码受自动修订控制系统的管理吗？

_____ 24. 项目环境设定有无包含完成项目所需的基本工具，像缺陷追踪软件、源代码控制系统与项目管理软件？

风险管理

_____ 25. 项目有无一份最新风险控制表格？这份风险控制表最近有无更新过？

_____ 26. 项目有无一名项目风险负责人？

_____ 27. 如果项目采用转包制，有无一套规划来管理各承包组织及其管理人员的办法（如果项目不采用转包制度，这题就给满分）？

人事

_____ 28. 项目小组拥有完成项目所需的复杂技术吗？

_____ 29. 项目小组对使用软件的事务环境有专业见解吗？

_____ 30. 项目中有无一名能够带领项目成功推行的技术主持人？

_____ 31. 有完成工作所需的足够人手吗？

_____ 32. 每个人是否都合作愉快？

_____ 33. 每个人是否都尽心投入项目进行中？

总和

_____ **原始分数**：将每个答案得到的分数加总起来。

- **加权比例**：如果项目团队中只有三名或更少的全职开发人员、质量确认人员或第一级主管，这里的加权比例就是1.5。如果全职人员有4到6人，这里的比例就是1.25。否则加权比例就是1。
- **最后得分**：将原始分数乘上加权比例。

求生测验的评分

这对大多数项目来说都是个艰难的测验，许多项目都少于50分。底下就是得分所代表的意义：

表2-1　求生测验评分表

分数	解　释
≥90 成果出众	得到这样分数的项目几乎可以保证能满足各方面的目标，包括时间、预算、质量与其他方面的要求。套用第1章的项目需求阶层中的观念，这样的项目完全满足了自我实现的需求
80~89 优秀	这种程度的项目表现比一般水准要好得多。这样的项目很有可能做出接近时间、预算与质量目标的软件
60~79	这范围内的得分代表软件开发成效比一般水准要

(续)

分数	解　释
好	好。这样的项目较有机会满足时间或预算目标，不过大概不易同时达到这两个目标
40~59 普通	一般项目都是拿到这个范围内的分数 得到这个分数的项目团队通常都会出现压力大而不安的情形，产品花费的成本及工时均高，功能却比要求做到的要少。这样的项目最能从本书提供的规划方式中获益
< 40 危险	得到这分数的项目在需求、规划、项目控制、风险管理与人事上都有明显缺陷。这类项目该注意最后到底能不能完成

求生测验的说明

每个“是”的答案都可以替项目拿到三分，你也可以酌情给分；如果是“大概”的情形可给两分；如果是“有点，不全是”的情形给一分；如果你的项目刚开始，请依据拟定的项目计划答题；如果项目正在进行中，请依据项目中真正发生的情形作答。测验结尾处会告诉你不同的得分所代表的意义。

上述的求生测验建立一套将来可供作比较的检测基

准。如学校学期刚开始时的测验一般，当你做过这样的测验后，你再花时间研读与学习新课程，到了学期末再做一遍同样的测验。如果老师已尽到教学责任（而且你也认真学的话），你的得分就会改善。

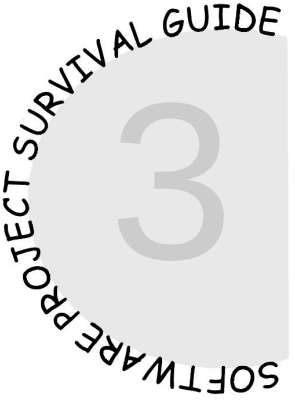
做为一个好测验，这些“期初与期末”测验应该涵盖课程的所有范围。这里的软件项目求生测验包含了软件项目求生的所有要旨。当你看完本书，好好规划你的下一个项目，把这里的测验再做一遍，你的下一个项目得分会改善，而且项目的存活率将会跟着提高。

求生检查

- 项目在求生测验中得到的分数应该在60分以上。
- 项目得到的成绩少于60分，可是规划了可以改善成绩的正确行动。
- 项目团队没依循正确行动的规划。

第 3 章

求生概念



良好的开发程序对软件项目的存在是重要而必须的。有了良好的开发程序，软件开发人员可以将大部分时间用在让项目更稳定的生产性工作上。如果开发程序规划不良，开发人员将会花费大部分时间修正错误。项目成败的关键在于拥有良好的准备工作，并让有见识的项目出资者了解，项目人员投注了足够的精力在准备工作上，以减少后续发生的问题。

任务开始前，你会看到最重要的演示文稿。本章描述让软件任务成功的关键要素。

“开发程序”的威力

这是一本有效的软件开发程序的书。“软件开发程序”这字眼可以代表许多不同的东西，而底下就是这字眼所要表示的事情：

- 把所有需要的项目纪录下来。
- 使用系统化的做法来控制产品的增加与更改。
- 推动所有要求、设计与原始码系统化的技术性审查。
- 在项目初期就发展系统化的品质保证计划，其中包含测试计划、审查规划与错误追踪计划。
- 建立一套产品功能组件发展与统合的实作规划。
- 使用自动化源代码控制系统。
- 在包括需求分析、构架确定、细部设计与实作阶段末尾等各个重要过程完成后重新评估成本与时间的表格。
- 这些开发程序都有着明显的快速成效。

对软件开发程序的负面观点

“开发程序”这字眼被一些软件开发界的人仅当成一个不实用的词汇。这些人认为“软件开发程序”是死板、苛刻而没效率的。这种看法的论点是，最好的项目执行方式是聘请你能找到最佳人才，给予他们要求的所有资源，然后放手让这些你找来的人处理他们最专精的东西。照这种观点，不受任何程序约束的项目才能够特别有效率。持有这种观点的人把项目的推动过程想像成了如图 3-1中所描绘的样子：

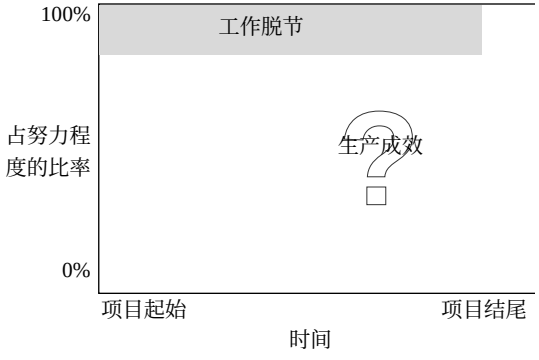


图3-1　对忽略开发程序可以增加项目生产性工作比例的错觉

抱有这种论点的人承认“工作脱节”或非生产性的工作占有一些份量。开发人员会做错事，他们承认这一点，可是这些错误可以很快有效地修正过来当然这比“开发程

序”所要花费的整体成本要少。

依据这样的看法，如图3-2，在项目中加上程序设定只不过是多余的，还会耗去生产性工作时间。

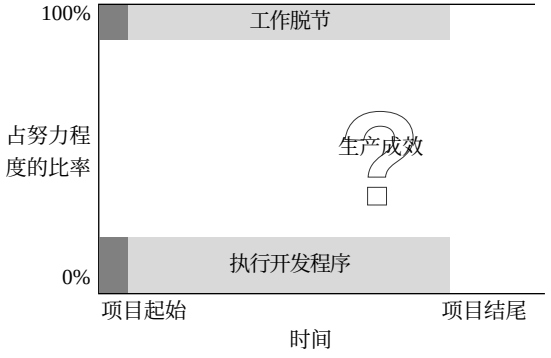


图3-2 对于注重开发程序的项目的生产性工作量的错觉
（开发程序的执行被当成纯粹多余的）

这样的说法有着直觉性的吸引力。在项目开始（图中暗灰色部份），对执行开发程序的关切用掉了一些生产性工作的时间。如果这种情形从头到尾持续下去（图中亮灰色的部分），继续花时间去执行开发程序就不合理了。

不过根据软件业界的经验，在中型项目中，图3-2所示的情形不会在项目中持续下去。没在开始建立有效的执行开发程序的项目在一段时间之后还是被迫建立执行程序，而且花费时间更多而得到好处更少。下面就是些愈早

设定开发程序愈好的情形：

- **变动控制**：项目推行到一半时，团队成员同意非正式地做个由主管或客户直接提出来的大幅变动。他们一开始并没有系统化地控制项目的变动，结果产品范围扩大了25%~50%以上，而预算跟工时也跟着追加了。
- **质量保证**：在初期没设定好消除错误程序的项目，陷入了似乎永无止境的测试－改错－修改－重新测试的冗长循环中。项目末尾的测试结果找到了相当多的错误，“更改控制布告栏”上每天都排着不同错误修正的顺序，“功能实践团队”每天都开会排定不同错误修正的顺序。由于错误太多了，软件推出时还有许多已知（虽然不严重）错误。最糟的情形下，软件质量可能永远达不到推出水准。
- **不受限制的审查**：项目中太晚发现的重大错误让软件在测试阶段还须被重新设计并重写，使项目在本来不受任何规划控制的情形下严重脱轨。
- **错误追踪**：太晚开始追踪项目中的错误情形，忘了来修正一些已知的错误，而让这些可能很好修正的错误跟着产品一起推出。

- **系统整合**：不同开发人员发展的组件到项目末期才整合的结果，使得组件之间的接口缺乏协调，而要花费更多精力才能让这些组件步伐一致。
- **自动化源代码控制系统**：源代码修定控制建立太晚，让开发人员意外将自己的程序版本盖过源代码正本或其他人的源代码档案。
- **时程安排**：在缺乏时间表的项目中，开发人员常被要求每周或经常重估剩余工作所需花费的时间占整体开发工作时间的比例。

一个起初就对各种开发程序漠不关心的项目，会让开发人员在此后觉得他们把时间都花费在开会跟修正错误上，而非用在加强软件功能上。他们知道项目进度脱节了，当他们发现不能满足时间底限时，他们的求生念头开始萌生，使他们退回“单独开发模式”——只求满足个人的时间底限。他们不再去跟主管、客户、测试人员、技术写作者跟开发团队中的其他人打交道，使得项目纪律荡然无存。

远远不如图3-1所示水准稳定的生产性工作比例、不太注重开发程序的中型项目一般都经历了如图 3-3中所示的生产力变化方式：

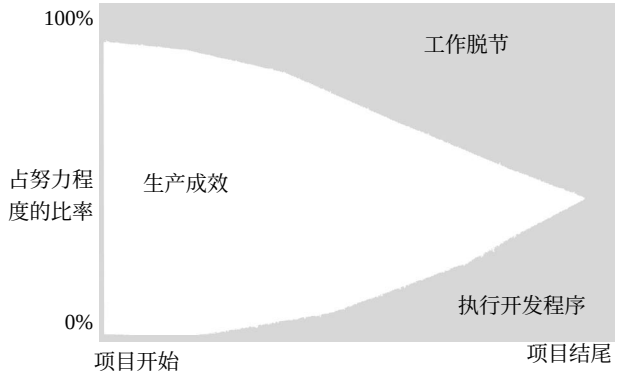


图3-3　不注重执行开发程序的实际经历。当项目环境愈来愈复杂时，工作脱节情形与所需的处理程序也跟着一起增加

图中，项目经历了推动中工作脱节状况递增的情形。等项目推动到途中，团队才会明白工作脱节耗去了许多时间，如果能进行一些对应的开发处理程序是会有好处的，不过已为时晚矣，对项目的伤害都已经出现了。项目团队试着增加开发处理程序的功效，可是这样的努力做得再多也弥补不了工作脱节的严重情形。有时努力步伐太慢一样会让工作脱节得更严重。

较幸运的项目还可以在仍有残余的工作时间里把产品赶出来，不幸的项目就没办法在工作脱节之前完成产品。这种情形持续数周或数个月后，这些项目一般都会因为主管或客户明白项目推动进度受阻而遭到取消。如果你认为

项目中的开发程序设定是件不必要的额外负担，想想看一件被取消的项目可能让你更划不来吧。

补救程序

幸运的是对这样凄凉的场景，还有些变通的补救办法，最令人高兴的是完全不用依赖那些“死板而没效率的开发程序”。[1]当然，有些软件开发程序确实死板而没效率，所以我不建议把这些处理方式用在项目中。本书所要描述的是使用能够增进项目弹性与效率的那些开发程序。

当这些开发程序被用到项目中时，项目的工作效率比例就会变成图3-4中的样子。

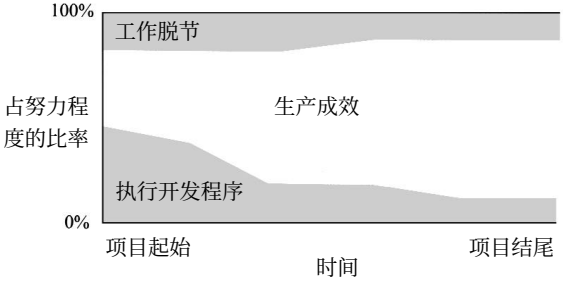


图3-4　及早注重开发程序的项目提供的经验就如本图所示。当团队从开发程序中获得经验，并适应好开发环境后，那些在开发程序与工作脱节状况所浪费的时间就大幅缩减了

[1] 译注：“死板而没效率的开发程序”原文为rigid, inefficient processes，缩写成R.I.P.，刚好跟请死者“安息”的rest in peace的缩写相同。

在项目进行的头几周，程序导向的团队似乎比害怕使用开发程序的团队更没生产力，因为两边工作脱节程度都差不多，而程序导向的团队会花费较多的时间来执行开发程序，这项投资稍后就会在项目中显出成效了。

等项目进行到一半，先前就重视开发程序处理的团队跟之前比较，工作脱节的程度会开始降低，程序的执行也会开始顺畅起来。同时害怕使用开发程序的团队则开始明白工作脱节已无法掩饰，开始自己寻求补救办法。

等项目接近尾声时，程序导向团队正处于顺畅运作状态，很少出现脱节的情形，而程序的执行也感觉不出有什么费时。少许的工作脱节状况难免存在，因为为消除这些脱节现象所花费的精力实在不划算。当上述提到的开发程序都被执行后，程序导向团队比起害怕使用开发程序的团队会更早进入状态。

项目一开始对开发程序所做的努力终究有所回馈。

明确注重改善开发程序的组织，可以在几年中把产品

的上市时间缩减了一半，错误发生率降低了60%~90%倍。在五年里，洛克希德公司（Lockheed）把产品开发成本降低了75%，时间则缩减了40%，而错误发生率也降低了90%。在六年半的时间里，Raytheon公司的生产力提升了三倍，投资回报率（Return On Investment, ROI）达到八倍。Bull HN公司在经历四年的软件开发程序改进后，投资回报率也达到四倍。Schlumberger公司在花三年半的时间改善软件开发程序后，把投资回报率提升到接近九倍。NASA的软件工程实验室在八年中把每个任务的平均成本缩减了一半，错误发生率降低了75%，同时大幅度增进了每个任务中使用软件的复杂度。类似成果也出现在休斯、Loral、摩托罗拉、全录与其他注重系统化改善软件开发程序的公司中。

最好的消息是，你猜猜看，这些生产力、质量与时程表现的改善花费的成本是多少？只花费了整体开发成本的2%的代价而已 —— 通常每名开发人员每年约花费1 500美金而已。

开发程序对创意与士气

对于系统化开发程序的一个反对意见是认为这样会限制程序写作人员的创意。程序员们确实很需要有创意，不

过主管与项目主持人也需要让项目维持在可掌握的范围内，让项目进展看得出来，而且满足时间、预算与其他目标。

系统化开发程序会限制开发人员创意的说法，是由于对开发人员创意与达成项目目标管理两者间一些矛盾的想法所致。两者并行不悖的环境当然是可能的，而且许多公司都已经做到了，就像建立一个能够同时满足不同目标间的协调并完成这些目标的环境一样可行。

注重开发程序的公司已经发现有效率的程序可以支持开发人员的创意与士气。在一个针对约五十家公司的调查中，发现只有少数开发程序导向公司的人将自己公司评为“士气良好”或“士气高昂”。在对软件开发程序付出更多注意的组织中，约有一半的人将自己公司评为“士气良好”或“士气高昂”。而在最注重开发程序的组织中，有 60% 的人将自己公司评为“士气良好”或“士气高昂”。

当程序写作人员最具生产力时，他们的感觉也最好。好的项目领导者建立清楚的目标远景，并利用一套开发程序构架让程序员们觉得自己拥有不可思议的生产力。程序员们讨厌阻碍工作的各种障碍，他们不得不抛开眼高手低、毫无章法的脆弱领导者。程序员们欣赏有远见、有见识与有魄力的开明领导者。

在此对开发程序与创意之间所谓矛盾的适当响应就是，本书中没有任何一个程序规划会以任何方式限制程序写作人员的创意，最多是提供一个支持架构，让程序员们能够自由地对相关的技术性工作发挥创意，让他们免于分心在那些消耗注意力的琐事上。

过渡到系统化开发程序

如果一支项目团队目前没采用系统化程序，一个最简单的过渡办法就是描绘出目前的软件开发程序的细节，将没用的部分标示出来，试着改正那些地方。虽然项目团队有时会宣称他们目前没有规划开发程序，每个项目团队实际上都有自己的开发程序做法（如果他们说目前没采取开发程序的做法，他们大概只是没有一套好的开发程序规划）。

最粗糙的开发程序做法一般是这样的：

1. 讨论要写出怎样的软件。
2. 写出些程序来。
3. 测试程序，找出缺陷。
4. 除错，找出错误的根源来。

5. 修正缺陷。

6. 如果项目还没完成，回到第一步。

本书提供了一套更精巧的软件开发程序。

建立系统化软件开发程序的障碍之一，就是项目团队害怕自己会在面对太多开发程序项目时出错，害怕他们执行这些程序后，会变得太官僚化，对项目执行造成太大负担。这通常不构成显著的危险，因为：

- 使用本书做法的项目会有个相当精细的开发程序，不会带来太多负担。
- 软件项目常常比第一眼看起来的要大得多。多数项目的问题出在本身，而不是出在开发程序中。
- 比较好的做法是开始先设定较多的开发程序以供执行、再视情况调整，比开头毫无章法其后再追加额外程序项目要容易。
- 多设定些程序项目所花的成本与时间代价要远低于不重视这档子事。接下来，就告诉你为何如此。

上下游

好的软件开发程序可及早处理项目中的问题。

你有时会听到经验老到的软件开发人员提起软件项目

中的“上游”跟“下游”部分。“上游”这字眼只是用来表示项目的开始部分，“下游”则是指项目的末尾部分。

“上下游”的分法对于思考项目上的问题上是很有用的。开发人员在项目开始所做的，在末尾未必有收获。如果一开始工作处理得当，末尾的收获就会健全并有助于项目成功。如果开始工作做得不好，后来的结果就可能严重影响成果，不客气地说，甚至可能让项目无法完成。

研究人员发现诸如规格或构架上的错误，事后才修正会比一开始就修正错误要花费多出50~200倍的代价。图3-5说明了这种效应。

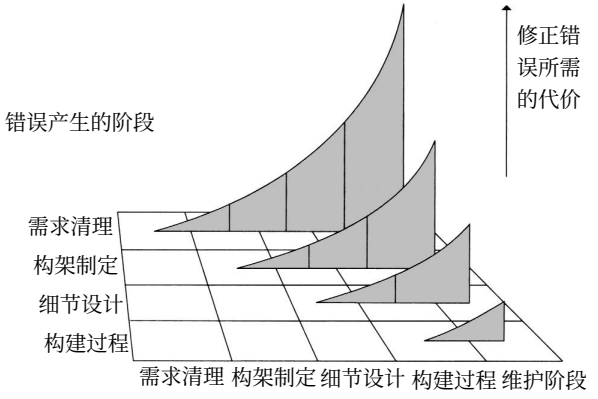


图3-5 错误所耗费的代价随产生错误与修正错误的时间距离而成正比。有效率的项目采用“阶段围堵”的做法——把错误在产生的阶段内找出来并修正

要能让规格中的一句话很容易变成几个设计方式，而这些设计方式之后能再变成几百行的源代码、几打的测试情形、许多页的使用文件、线上求助画面及许多技术支持部门的操作说明等等东西。

当项目团队有机会阶段性修正错误，立刻修正错误会比过一阵子再去修正受影响之处要有意义。这种在错误发生的阶段内找出错误并马上修正的构想称作“阶段围堵”策略。

成功的项目借由仔细审查、要求规格与构架来制造机会修正上游问题。

在上游动作时还没写出任何程序，找寻及修正错误，虽然可能让项目“该做的工作”表面上看来被拖延了，实际上却刚好相反。这些工作是在替项目的成功铺路。

过多程序项目的错误会越发的增加项目的负担，不过缺乏程序规划的问题会让错误变成必须花费50~200倍的效率成本才能修正的大麻烦。因此宁可多设定开发程序事项，也比不做开发程序设定来得好。

不确定性的角锥

上游错误要多花费50~200倍的代价才能在下游修正的一个原因是，上游决定的范围比下游的要广。

在项目初期，项目团队面对的是大问题，例如要不要同时支持Windows NT或Macintosh或是只支持Windows NT就好了，还有报表格式是使用者自订还是固定格式。在项目进行中，项目团队面对的是中型问题：有多少子系统；如何按一般情形处理错误状况；或是如何把一个打印例程从过去的项目移植到现在的项目中。

到了项目末期，项目团队面对的是小问题，像是要采用哪一种技术算法，或者要不要让使用者能够取消还没完成的动作等等。如图3-6所示，软件开发是个持续渐进的程序，将问题由大到小分开处理，从决定大势走向到解决个别的小问题。推动软件项目所花费的时间，就是彻底思考并解决问题所需要的时间。

本图中的角锥状图形为项目中决策方向未定的事务量。软件项目中对问题的决策尺度由大到小。除非团队成员完成了前一个阶段的大部分工作，不然无法知道一个特定阶段中到底有多少需要决定执行方向的问题。

项目团队最擅于处理大型的决策问题，不过有时无

法预见（而且不可预知的）的问题会在后头因为大型决策不当而出现，若要中途取消一个行动则表示项目团队得重新设计一个常式、一个模块甚至一整个子系统。

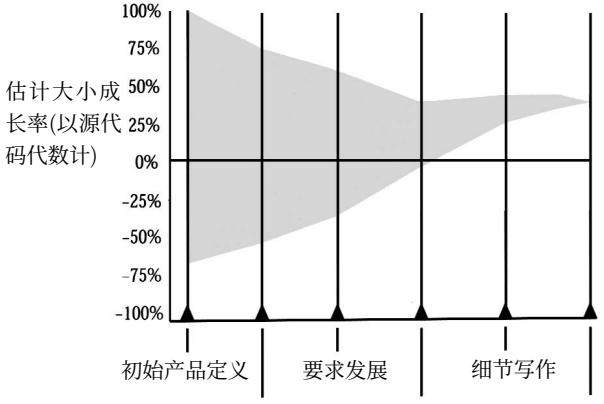


图3-6　本图中的角锥状图形为项目中决策方向未定的事务量。软件项目中对问题的决策尺度由大到小。除非团队成员完成了前一个阶段的大部分工作，不然无法知道一个特定阶段中到底有多少需要决策执行方向的问题

在某一尺度的决策敲定后，应可以事先精确预估下一尺度将面对的决策种类。不过项目团队或许可以先做好前头的决定，但对项目后头的决策问题只可能作出一般性的猜测。如果你想知道软件开发的工作为何，了解项目团队所有该思考的问题，并明白一个团队得在充分了解下一个阶段工作的内容之前便决定好现阶段中所有的执行方向，

就成了重要的事情。

不确定性的角锥对项目预估的意义

不确定性的角锥对软件项目的预估有着强烈的意义。它不光表示在前期阶段精确评估项目的困难性，还说明那在理论上不可能办到。在需求规格阶段末尾，项目的范围还得由一大堆在构架阶段、细节设计阶段和构建阶段要作出的决策所决定。宣称能够预估未定决策对项目影响有多大的人如果不是先知，就是对于软件开发的本质不太了解的人。

另一方面，寻求控制决策方向以符合项目预计时间或预算目标的人是明智的。只要你愿意痛下决心砍掉一些规划功能，在项目初期就确立精确的时间表和预算目标，并在之后一直按部就班。成功的关键在于以这样的方式达成目标要求，做法包括在项目初期设定明确而不冲突的目标，保持产品概念非常有弹性，并主动追踪控制项目中的开发工作。

在项目开头，你能制定实在的成本和时间目标，或是制定充实的功能组合，但是你不能强求两边的目标都要达成。

求生检查

- 项目领导阶层了解良好开发程序的重要地位，并愿意支持这些程序的执行。
- 项目的开发程序一般倾向尽早在上游阶段找出愈多问题愈好。
- 项目领导者了解项目前期的预估状态难免不精确实需要在项目进行中再做修正。

第 4 章 求生技能

软件项目本来就复杂，而复杂的软件项目若无细心的规划就不可能成功。一个良好策划过的项目会被有效控制着，其进度操控自如，且会照顾到参与项目进行者的福利。软件项目本质上也是危险的，缺乏风险管理就不可能获得成效。从头保持跟项目的使用者联系，努力将产品维持在满足客户的要求之上，并尽全力把焦点放在找出项目关键问题的解决方法上。

小型项目可以只靠着毅力与运气而完成，中型和大型项目则需要更多系统化的做法。本章将概述一些让中型项目达到成效的技巧。

规划

许多技术工作者宁可直接做自己的工作而不想花时间规划一下工作方向。许多技术主管缺乏足够的技术训练并且不相信自己可以规划出改善项目的方式。由于两边都没人对项目好好规划，结果就是项目常常窘况百出。

没有一套规划方式是项目中最严重的错误之一。由于上一章中描述的上下游效应，想在修正成本较低的上游阶段将问题解决掉，一套有效的规划方式是必要的。一般项目约80%的时间花费在未经规划的重复工作上。

软件开发的成功归功于让所有错误变成一堆细心规划过后的小错误，避免出现未经规划的大错误。研究四种设计方式后，放弃其中三种，最多也不过是造成三个小错误而已。可是没做好设计工作而必须把程序重写三遍的结果，却严重到可能造成三个大错误。由于在下游阶段修正上游造成的错误比在上游阶段就修正好问题要多花费50~200倍

的代价，细心协调过的项目就有机会以1/200~1/50的代价将许多错误修正好。

项目中哪里找得出时间来进行规划？很简单，把大部分项目花在未经规划的重复动作上的时间中的一小段拿来用在项目初期的规划上，避免之后将时间花费在高代价的重复动作上。尽管并非所有上游阶段用去的时间都会确实省去下游阶段该处理的麻烦，但是省下的麻烦还是很多。在上游阶段质量保证的经验法则是：用在进行技术检阅等早期规划工作上的一小时可以省下下游阶段3~10小时的工作时数。从不同的观点来看，一名开发人员每天花在项目要求规格或架构上检查的时间一般会省下后一阶段中3~10天的工作时数。

软件规划的例子

一个项目团队该如何规划项目，以在一定预算内完成软件？底下是一些项目规划的具体工作项目：

- 一个软件开发的规划反应项目进行的方式。把规划方式纪录下来可以让项目的资助者透过这份规划来了解项目。
- 项目评估提供了项目规划的基础。一份仔细的评估

可以确切地定出项目的规模，从而确切地定出预算上限、人员需求与时间需要。差劲的评估会低估项目在各方面的需要成本，使项目难以顺利而有效地完成。

- 在每个主要阶段末尾作个修正评估，可以中途修正项目成本，并让项目的进行以稳健的步伐前进。
- 一份包含技术审查与测试的质量保证规划，可确保项目不会被代价高昂而找不出错误的测试、除错和修正周期压垮。
- 一份详细规划出软件构建的方式，可确保软件解决方案有效地在各阶段以提高对客户价值和降低风险的方式进行。

除了以上这些明确的规划方式，软件项目的几个主要活动也可以被规划处理。

- 详细订出项目团队试着要解决的问题，确定正确解决问题的方式。
- 构架设计出解决问题方式的大体规格，建立正确的解决方案。
- 细节设计是关于如何建立项目软件的综合规划，以正确的方式建立正确的解决方案。

规划审查

规划方式对项目的成功是如此的重要，一些专家说项目的成败完全决定于项目初期10%的时间内。不管确实比例是10%还是多少，在项目的最早期，项目团队应该已经订出使用者接口雏形、细部要求跟包含成本与时间预估的细部项目规划，而这些信息可以用来决定要不要让项目进行下去。

两阶段出资方式

一些组织中对软件项目经费的问题在于项目主管在完成一些探索性的工作前就得要求上头支出必要经费。这样的请款要求必然失准，因为对软件了解不足。软件业界的经验是，在项目初始阶段的估计或多或少，可能会跟实际的要求差距达四倍。

一个较好的办法是让项目主管把经费要求分作两个阶段。项目主管先在项目完成初期10%到20%的探索阶段要求第一次经费。到了阶段结尾，项目团队进行一次规划审查，同时资深管理阶层或是客户决定要不要继续推动项目，然后再拨项目剩下的部分经费。这时项目成本仍有可能变动，不过先前已经完成的部分会让整个成本变动范围从四

倍缩小到50%左右。

准备规划审查

在进行规划审查之前，得先要有下面的准备：

- 决定好项目关键决策者
- 前景叙述
- 该软件的业务状况
- 初步努力与时程目标
- 初步努力与时程目标估计
- 十大风险列表
- 使用者接口风格简介
- 使用者接口细部雏形
- 使用手册/使用需求规格
- 软件质量保证计划
- 软件开发细节规划

上述每一项准备在本书其他部分会更详尽地解说。

如果这些项目都没准备好，表示还没有足够信息来决定项目的质量如何，就不用进行规划审查了。如果项目团队一直都没能够做好这些规划审查的准备，失败的风险很大。

做好这些准备所需要的时间依据软件所需要的工作量而定。当一般使用者知道他们要的软件是怎样的情形下，这个准备时间可能只需要软件总开发时间的10%。一般情况下，这段准备时间会占总开发时间的 10%~20%。在一些项目中，这项工作最困难的部分是帮助一般使用者理清他们要的是什么，所以偶尔这部分的工作会占用总开发时间的25%以上。对规划审查的经费要求与计划应该将这项工作的变动考虑进去。

规划审查的一般项目

规划审查应该集中在下列各项目上：

- 本来的产品概念仍然可行吗？
- 发展一个满足项目前景叙述的产品可能吗？
- 该软件的业务状态在更新、更精确的成本与时间估计出炉后依然差距不大吗？
- 项目的主要风险可被克服吗？
- 使用者与开发人员都能够同意使用者接口的细节雏形布置吗？
- 使用说明/使用需求规格是否完整而稳定，足以支

持更进一步的开发工作？

- ◆ 软件开发计划是否完整，并适合进行更进一步的开发工作？
- ◆ 完成项目所需的预估成本为多少？
- ◆ 完成项目所需的时间安排如何？

在项目开头10%~20%的完成部分应该足以解答这些问题，而且这些问题的答案应该足以让客户或上层主管决定是否要继续拨款进行项目的后面阶段。

规划审查的主要好处

让软件组织将软件项目经费分成两阶段处理至少有三种好处。首先，被取消的项目常被看成失败的案子，可是一个进行了 10%~20%就取消的项目反而应该视为彻底成功。完成 80%~90%才被取消的项目所耗用的经费足以负担许多在探索阶段进行 10%~20%就取消的项目。

再者，将庞大的经费延迟到项目完成 10%~20%之后再拨款的话，可以对项目所需的庞大经费做出更可靠的要求。

最后，要求项目主管先完成项目的 10%~20%才能要

求拨款进行下面的部分，可以让这些主管做好与对项目成功息息相关的上游工作。这些工作往往被省去或忽略掉，带来的损失只有到项目下游阶段必须花费昂贵的成本来修正许多错误时才会突显出来。如果项目团队被要求在下游工作之前完成最重要的上游工作，项目的整体风险就会大大地降低。

风险管理

风险管理是一种特殊的规划方式。进行过大中型软件项目的人都亲身体验到许多事情都可能出错。最成功的项目就是采取积极步骤以免屈服在这些问题的困扰之下。你也许是完美主义者，可是对软件项目而言，就如人们说的，你可以有最佳期望，但是应该要有最坏准备。

几种最严重的软件项目风险与规划方式息息相关：

- 没规划好。
- 没照着规划方式做好。
- 没在项目环境变化后修正好规划方向。

在软件项目中不采取积极的风险管理就是忽视软件业界在过去几十年中从几千次教训中总结的一个经验：软件开发是高风险的活动。如果项目采取积极风险管理

的方式，就可以避免或降低许多风险，而这些风险有许多如果没处理好，就可能使项目陷入瘫痪中。

本书求生策略中包含（并在本书其他部分谈到）的做法比起其他方式风险更低，并能够增强对其他类型的项目风险的发现与控制，而被选录。对软件项目中要不要采取风险管理策略，你没什么选择余地。如Tom Gilb在《软件工程管理原理》（Principles of Software Engineering Management）一书中所说的，如果你不积极面对软件项目中的风险，你就会碰到这些问题。

项目控制

本书的一个主旨是，软件项目可被控制，从而迎合时间、经费跟其他目标。对一些人来说，这种项目“控制”的点子简直惨无人道，令人想象起一名专制的项目主管以皮鞭和铜链展现威权的样子。

项目“控制”的反面就是项目失控。

人们也许，会认为项目即使没人控制方向，也会顺利

地进行下去而不会失控。我的想法和经验都告诉我那是不可能的。

有些人反对项目控制，因为他们认为这表示控制项目成员的行动。事情并不是这样，项目控制所控制的是项目本身。下面是一些我所指的“控制”的意思：

- 选用一种软件生命周期模型，如本书中使用的阶段完成模型，给项目中技术性工作一个轮廓构架。
- 管理需求的改变，只接受必要的改变。
- 确立设计与程序写作标准，使设计方式与源代码以彼此一致的方式产生。
- 建立项目的细节规划，使每一名开发人员的工作朝项目目标前进而不会妨碍到其他人的工作。

控制并不是良好的技术性工作的附属品。项目控制必须透过积极的项目管理与持续渐进的控制行动来落实。项目控制的具体行动将在本书其他部分谈到。

项目洞悉力

与项目控制密不可分的是“洞悉力”的概念，这是指决定项目真实状态的能力。项目是否进行在时间、预算目标的10%范围内，项目是否按部就班地达到质量目标，或

是远远落后目标？如果项目团队答不出这样的问题，这支团队就缺乏足够的洞悉力来控制项目。

下面是一些改进洞悉力的方法：

- 用前景叙述或目标来订出项目的概括目标。如果你不知道项目的行动方向，项目大概就推动不了了。
- 在项目完成10%后做一次规划审查，看看项目可行与否，以决定是否该完成这个项目。
- 经常比较实际进度跟规划进度，以决定规划方式是否有用，是否需要修正。
- 用二元完成点决定该做的事情是否完成。完成点只有“做好了”和“没做好”两种状况，因为事实证明，从“完全做好了”让步到“90%完成”只会让项目状态信息的质量从“很好”降到“极糟”。
- 定期将产品做成可推出状态，以决定产品的真实状况，并控制质量水准。
- 在每个阶段末尾修订估计数据，依据这些新信息来改善规划方式，并更进一步了解规划方式中的假设状况。

许多项目团队最后发现项目洞悉力并不是自动得来的。如果你想要有良好的项目洞悉力，得从一开始就把这

一点放在项目规划中，而如果你想要有个成功的项目，你就得有良好的洞悉力。

人因工程

软件开发需要创意和智能、原创与持久再加上高度的干劲。任何对软件开发有效率的做法是，如果项目团队没进入状态，项目就不可能成功。成员们必须进入状态，但是他们有可能士气低落，或是缺乏一个可以让麻雀变凤凰的灵感，甚至连团队的核心都可能在推出产品之前就离开工作岗位。

Tom DeMarco与Timothy Lister让“人因工程（peopleware）”这个显示出在软件开发过程中人类成员才是最重要的字眼变得热门起来。如果你不是名软件开发者，人因工程的一些准则也许会吓倒你。

照开发人员的兴趣分派工作

一般来说，软件开发人员的最大动力就是工作能与个人兴趣相符。如果开发人员发现工作有趣，他们就会发挥高度干劲。如果他们觉得工作很无趣，就会趣味索然。Robert Zawacki研究了十五年后指出约60%的开发人员的

生产力来自个人有兴趣的工作上。要让人们发挥最佳生产力，请依据开发人员的个别兴趣来分派工作。

有些人的干劲来自不可能达成的目标。
可是开发人员有时太钻牛角尖了，如果目标达成不了，反而会让他们失去干劲。

让开发人员知道你真的欣赏他们的成就

就像其他人一样，开发人员也喜欢别人对他们的赏识。如果项目主持人真心赞赏开发人员，他们对项目就会更加投入。如果主管对他们的赞美虚伪而做作，他们也以虚应对。不要用敷衍手法、离谱的目标或金钱诱惑来挑动开发人员的干劲。

提供思考导向的办公室空间

软件开发既要有发现也要有发明，缺一不可。整体过程中最好的环境就是既轻松又可以自在思考的场所。开发人员要如同数学家或物理学家般聚精会神。试想如果爱因

斯坦坐在办公桌前听着他的主管责骂他说，“爱因斯坦，我们现在就要看你的相对论！快把它写出来！”你想他做得到吗？（译注：爱因斯坦发表相对论时还只是个邮局职员）况且软件开发人员远不如爱因斯坦般聪明，他们需要更有助于工作的环境。

避免采用开放式工作隔间

有个老生常谈的论调，说开放式的工作隔间有助于软件项目间的沟通。问题在于这样的隔间沟通起来反而杂乱无章，连带影响生产力。有效率而良好的沟通还不如在茶水间放部汽水机来进行。开放式工作隔间有助沟通的论调乍听起来很吸引人，可是经不起确切考验，而且软件研究资料清楚显示开发人员只有在一两个人的办公室内最具生产力。

一些研究发现开发人员在隐密、安静、一两个人的办公室内工作和在开放式隔间的办公室内相比，前者的生产力比后者要高出2.5倍。软件开发是个需要高度思考的活动，如果电话响个不停，广播系统播个没完，人们在你四周走来走去，每隔几分钟就来打扰，怎么可能专心思考问题。

身为主管的工作要求是每隔几分钟就转移注意力以达到管理效果。

一般软件开发人员要求几个钟头内最好不要转移注意力，才能专心。许多组织没办法提供开发人员安静隐密的办公室。他们有的是空间不足，有的是要留给副总裁之类的人用。有些组织发现，将他们的开发团队安排到另一个环境去会更好些。其他组织则试着让开发人员带着耳机工作杜绝干扰，或者干脆让他们在家工作。至于其他没办法提供开发人员安静隐密而不受干扰的组织，只好自求多福了！

使用者参与度

软件使用者参与的程度对软件项目很重要。软件成功的关键在于设计出一般使用者爱用的产品。如果没有使用者的参与，软件开发人员常常会先考虑技术层面而忽略使用者的需要。这样的话或许软件产品中塞进过多功能，使用者实际用到的却没几种。

要想开发出使用者爱用的软件产品惟一方法，在于问

问使用者要的是什么，把开发中的产品先告诉使用者，并问问使用者喜不喜欢这东西，聆听使用者的心声并参考他们的意见。

也许项目团队得努力好几次才能了解使用者要的是什么。使用者们也必须了解开发人员让他们看的雏形跟未来的成品也许相差十万八千里。有时要分辨出哪些可能是未来的“使用者”都有困难。这些都会在第8章中提到。

由于使用者的参与消除了一大变量——摒除了“差不多先生”的心态，这样一来项目所费的时间就少得多了。如果使用者没从头参与，以后对项目产品进行检视时，一定会指出有好些地方没照顾到他们的需要。这时开发团队就得面对一个艰难的抉择：依循原来的预算与时程目标而忽视使用者意见，或是在项目的下游阶段接纳使用者意见而让预算跟时间超出预期限制。通常开发人员会妥协，先采用可立即修正的使用者意见，而将难以立即采用的意见留到程序的下个版本去处理。说老实话，让使用者在产品还具可塑性时愈早参与愈好，而且最好是在那些不被喜欢的东西出现以前。

由于“做了比没做好，早做比晚做好”的观念，使用者对自己参与开发过程的项目反而比没让他们参与的期望

更深。表面上，愈早让使用者参与开发的软件愈能满足使用者需求与期望，而这也成了一种质量的认定方式。实际上这样的软件缺陷的确更少，因为它们在开发过程中方向变动较少，而让产品构架、设计方式与实作更稳定。如果在项目中途才加入使用者需求，被迫将没想到的功能加进体质不良的现有构架中时，软件质量一定会大幅下降。

除了及早参与、持续参与外，很少有软件项目一开始就找出真正良好的解决方案，一般都必须在使用者坐下来说“是的，这就是我要的软件”以前做出不同版本的使用者接口雏形。一个良好而成熟的使用者接口雏形可以让项目推出后广受欢迎，不过软件的适用性得在开发时就调练好。参照使用者所需导向而做低廉、小型的修正，就不用到了项目末期再大兴土木以致付出高昂代价（本书使用的阶段性完成策略提供了这样的过程中修正方式）。

参予开发过程的使用者用不着很多，在Jakob Nielsen的《Usability Engineering》一书中就指出，当软件适用与否的测试人数在3~9个时，价格获益比最好。

在1994年，Standish Group对8 000件以上的软件项目进行检查。结论是使用者的参与是项目成功最显著的因素。在那些失败的项目中，缺乏使用者反应是失败的主要因素。

计算机软件的专家指出快速开发项目能够成功的最重要因素在于让一般使用者全程参与开发过程。

让使用者参与项目开发过程可说是重要的软件项目求生技能。

产品简化主义

成功的软件项目开发从需求制定到产品推出、继而被接受，都需要一种“化繁为简”的定位方针。因为软件开发工作相当费神，若能将项目彻底简化必然事半功倍。功能规格、设计与实作都应该简化。许多开发人员沉迷于复杂性，误以为“数大便是美”，事实上只有简化项目才是成功的唯一出路。

当开发人员致力寻求项目目标流程的简化时，可说又向成功迈进了一大步，对一般使用者也可能造成影响。一个功能通常可分两小时版、两天版和两周版，开发人员应该一开始先弄出两小时版，通常这个版本的解决方案最简单直接，问题最少。开始实际操作后，如果解决方案还不

够，他们才应制作两天版甚至两周版的解决方案，再看看这样是否可以解决问题。解决问题的一贯态度是，先简单后复杂，才不致于因噎废食。

法国作家Voltaire说一篇散文不是在“增一字则太多”时完成的，而是在“减一字则太少”时完成的。软件项目亦同，应该是从软件中去除不必要的东西以进行简化，而不是不断加上更复杂的东西。

焦点放在推出软件

有效率的开发团队全心全力着重产品的推出。微软公司尤其重视“产品推出”。对产品推出有功的开发人员会获得一个“产品推出”奖，感谢他们的努力。在微软工作了许多年的开发人员一般都有一大堆产品推出奖。这样简单的做法强调一件事，微软公司不是靠着开发软件赚钱，而是靠推出产品赚钱，当然这也是大部分生产软件的公司赚钱的方式。

对于无论是替内部使用者开发软件或对一般大众发行软件的开发人员来说，焦点是一样重要的。一个清晰的前景描述可以让任何软件开发团队对产品推出的目标相同。如果开发人员到了项目结尾都还对产品的看法有歧义，将

被迫花费上许多时间、努力与金钱来让他们的看法一致。

一个清楚的构架也有助于让开发团队在目标上步伐一致。反之，就难以同心同德。如果一套良好的产品构架确立了，项目的开发焦点自然集中在技术层次上。

软件开发团队必须确保每个技术性决定都朝着简化系统功能的方向前进。如果是开发大学教育项目，也许有必要让项目任意复杂化。不过当团队在开发商用产品时，他们的任务是提供最简单的问题解决方案，任何违反这些原则的决定都应被否决掉。

本书所要传达的信息是软件开发工作本质上主要为了满足实际目标，这个工作有着浓厚的美学标准与科学成分。有效率的软件开发人员了解软件项目不是为了让他们有个筑梦堡垒，而他们也依此排定各项条件的优先级。微软公司的经验显示，这样得到的行动方针能够在项目团队中培养。没有共识的开发人员对于项目是一个累赘，对组织可说没什么用处。

求生检查

- 项目团队在项目初期就规划了一套解决高成本潜在问题的方法。
- 项目借助规划审查，在项目完成约 10% 时决定要不要继续进行下去，以此强调上游工作的重要性。
- 项目做到了积极的风险管理。
- 项目规划强调洞悉力和项目控制。
- 项目规划包含使用者从头到尾的参与。
- 项目在高生产力的环境中进行，或是假设项目成员生产力恐有不足而先做准备。
- 项目规划寻求由简而繁的办法，而不是反过来做。

第 5 章

看看成功的项目

软件项目是发现与发明的过程。发现与发明融合为一的最佳方式是透过“阶段性完成”的做法，将产品的功能分阶段完成，而最重要的功能最早完成。当项目进行时，许多活动交互重叠，把产品由抽象概念转化成具体成果。项目进行中的源代码倾向以S形曲线而非线性成长，而大部分的程序代码都是在项目中间第三部分完成的。追踪程序代码的成长提供对项目状态的洞悉力。执行良好的项目也可以由一名上层主管选择最有成效的一组来进行追踪。

本章描述一个成功的项目由两万尺的高空鸟瞰时的样子。它提供了通过不同的角度透视项目流程、人员、进展活动、程序代码成长与主要成效。

思考阶段

在讨论软件项目依照规划阶段分期完成的方法之前，我认为先对项目整个过程进行通盘了解会有用处。

软件项目如图5-1，被切分成三个概念阶段。在项目初期，焦点摆在“发现”，特别是发现使用者的真正需要。透过技术性调查、与使用者访谈和建立接口雏形，把不确定性的概念转换成确定的观念，这就是第一阶段的特色。

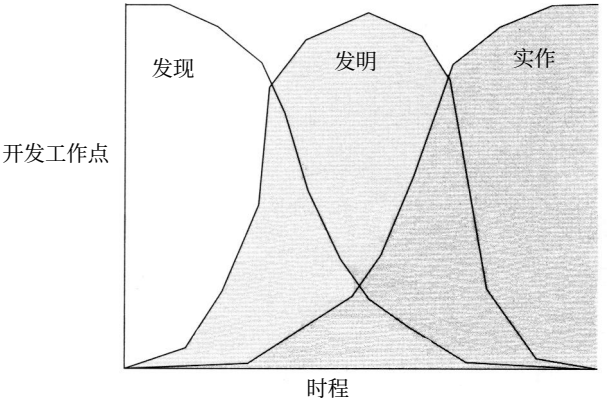


图5-1 软件项目的概念性阶段。在不同阶段有不同类型的工作，不过各类型工作都会完成到某种程度

在项目进行中期，焦点移到了“发明”上。往大方向看，开发人员要发明软件构架与设计方式。细节的地方，如每个函数式或对象类别也不能忽略。

如同发现阶段般，发明阶段的特征在于将不确定的概念转换成确定的观念。如果还有别的特征，就是发明阶段的不确定性要高得多。在发现阶段，开发人员可以确定答案“就在”某个地方。可是在发明阶段，就不能以此类推。

在项目的最后部分，焦点又转移了，这次摆在实作上。不同于发现与发明阶段的是，实作阶段的不确定性少多了，故可发掘出许多已确定的观念并可实现成具体成果。

如图5-1所描述的，发现、发明与实作在软件项目中各自以一定程度进行着。太严格分段规划反而不能有效运作，好的项目计划必须让发现、发明与实作一起出现。

项目流程

在某些软件开发方式中，项目团队几乎是秘密完成开发的大部分工作。技术性项目常提供如“完成90%”之类的状态报告。对顾客来说，如果完成项目的90%就花去了90%的时间，那剩下的10%可能会花去另一个“90%”的时间。

本书提供的项目规划依循着“阶段性完成”的轮廓进行。由于它将项目中开发的软件分阶段完成，而不是到了项目结尾才一次完成，这种方式称做“阶段性完成”。图5-2说明了这种方式。

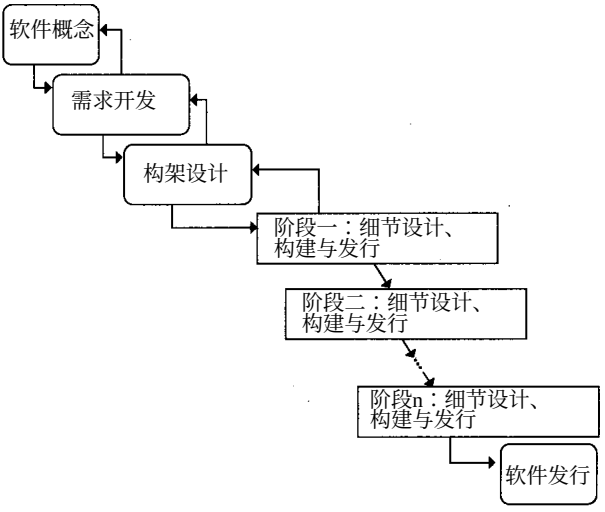


图5-2 阶段性完成规划。项目被详细定义与构架，然后在以下的阶段中分期完成各项功能

如你从这图所看到的，阶段性完成强调项目规划与风险降低。项目团队先发展软件概念，分析汇集需求；再完成构架设计。这些工作透过积极的风险管理与精心规划，朝着消除风险的目标前进。

在每个实作阶段中，项目团队进行细节设计、程序写作、除错与测试（以阶段1到n表示），在每个阶段都建立出可能推出的产品。这三个阶段也描绘在图中，不过你可能想随心所欲的操控项目的完成，有些项目可以只用三或四个阶段，而有些则有更充裕时间，使得每个星期都能推出新的软件。

分段性完成的好处

阶段性完成提供下述5种好处。

1. 关键功能更早出现

阶段性完成的项目中各阶段一般都先规划产品最重要的功能。如果使用者期望特定功能，就表现出他们不想等到产品完工了才看到这些东西，他们只想在第一阶段就先睹为快。比起一些仓促完工的冒失项目，阶段性完成对一个有着时间压力的项目可说是个宝贵的做法。

早期降低风险这种做法强调项目的规划与风险管理。将产品分阶段完成并随时整合起来的方式，可降低项目末期才整合失败的技术性风险。同时将可用的软件尽早提供给一般使用者参与，借着产生具体进展成效来降低管理风险。

2. 早期预警问题

当你及早计划各完成版本的软件时，你就会提早得到明白的进度报告。无论各个版本是否如期推出，工作质量可明显从推出版本看出。当开发团队碰上了麻烦，你也会先知先觉，不必等到项目“完成90%”了还摸不清头绪。

3. 减少报告负担

阶段性完成也提供一个长久有效办法来消除开发人员花在建立临时性进度报告和传统的固定性进度报告。

产品的动态比任何书面报告更能精确反应项目状态。

阶段性完成能提供更多选择。项目团队在每个阶段末尾评估可推出产品并不代表产品非推出不可，不过如果真的有必要，产品可以随时推出，而且将产品完成到待命状态，所需的功能应有尽有。如果你没采用阶段性完成的做法，你就没有这种选择。

4. 阶段性完成可降低估计失误

阶段性完成可通过对推出产品各部分功能逐步完成来避开估计错误的问题。与一次对整个项目做出大型评估相

比，项目团队可以灵活地分别对几个小版本做小型评估。在每个版本中，项目团队可以从评估中吸取教训，重新校正做法，并改善项目现况，使项目的未来预估更为精确。

5. 阶段性完成均衡了弹性与效率

分阶段完成产品让项目团队有精确的时间来决定软件中该更改哪些东西，这些时间来自阶段衔接的空档。在空档内决定软件规格的变更，让开发团队不必一直考虑软件改变，却能保证让项目不致遗漏该考虑变更的东西。

阶段性完成的代价

从前述所列的好处中，阶段性完成的做法听来似乎毫无缺点，其实则不然。阶段性完成的做法要付出相当代价。因为项目团队需要时间准备各种可推出的软件，在每个阶段重复测试已经测试过的功能，推出软件前进行相关的版本管制工作，提供试用的不同版本软件没预料到的问题的解决方案（如果阶段性完成的软件真的拿出去给人使用），还有规划阶段性发行这种做法的好坏等等，都会提高项目的负担。

其中的某些代价并不真的是额外多出来的它们不过是让本来隐藏在项目末尾的一些成本提早显现出来而已，找出

缺陷并修正错误就是这类的隐藏成本。阶段性完成项目的有些工作人员在一开始会抱怨他们把时间都花在修正错误上。实际上他们是在修理早晚都得修理的东西，而且这些在项目中先找出来的错误愈早修正，花费的成本就愈低廉。

其他代价也许是多余的，如对外发出很多不同版本的软件，会增加项目的整体负担和总成本。

阶段性完成并不是万灵丹，不过总合起来，那些额外的负担相对于明显改善了的状态、质量与时间的匹配、精确预估与降低风险等来说，不过是一点小小的付出而已。

规划阶段

图5-2的阶段性完成流程图说明项目早期活动，如需求分析与构架设计都是连续不断进行的。在投入构架设计的阶段之前先完成大部分需求分析工作，以及在进行细节设计与实作之前先定好整体构架，都是重要的事情。不过实际上，这些活动都是在同时间内重叠进行的，这是不可

避免而且必然的事情。图 5-3所描绘的，就是这种重叠进行的情形。

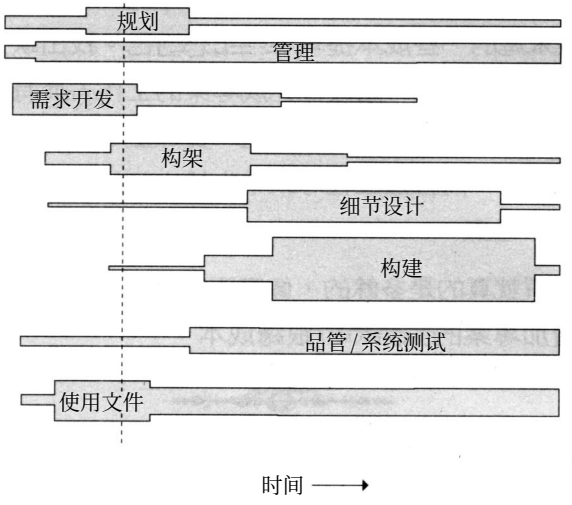


图5-3 典型重叠活动型态。需求开发与构架的前期活动不像后期一些的细节设计、程序写作与整合测试活动一样，重叠那么多部分条状图的粗细表示在项目的阶段中需要投入的人力的多少

从本图中，建立项目团队应该在开始构架设计前先做好大部分需求开发的工作。在开始细节设计以前，应该先完成大致的构架工作。这么做，是由于错误修正的成本会随着时间的延后而提高。80/20比例的规则可以应用到这上头来：在开始进行构架工作前，先完成80%的需求开发工作，而在开始细节设计以前，先完成 80%的构架设计。

80%并不是随便选定的比例，只是基于一条良好的经验法则：这样可以让团队在明白表示不可能一次把这些工作完全做好时，还能在项目初期先满足大部分需求。

细节设计、程序写作、整合与测试差不多在同一时间内完成，因为阶段性完成的做法会在各阶段中产生设计、实作、整合与测试的小循环。使用文件的撰写会提早进行，因为使用者需求先被开发好了（这在第8章中会提到），而且这些工作会持续进行，管理与规划也会持续处理。这样的复杂性也许会让人以为这种做法只适用于大型项目，不过实际上几乎任何项目都可以应用这些法则，只是程度上多寡的问题而已。

汇聚人力

从人力观点来看，在阶段性完成项目中应有两个阶段。在第一阶段中，项目还在酝酿中，团队正在开发需求和建构构架。在这阶段的整体努力程度一般被认为要低于主要实作阶段。前述项目完成10%~20%后决定要不要继续进行下去就是在这阶段中处理的。这一阶段的工作人员应该是技巧高超的资深开发人员。

第二个阶段是阶段性完成时期。在这阶段中，项目团

队处理细节设计、软件构建与测试。技巧高超的资深开发人员在这一阶段也扮演重要地位，不过项目也需要质量保证人员、技术文件写作人员和资历较浅的软件开发人员。

在项目后半50%里，你可以看出质量保证、细节设计、程序写作的人员要求保持相对顺畅。在一次做完所有事情的做法中，项目通常得花费更多人力，而且到了项目末期可用人力会变少。阶段性完成的项目则在大部分时间里有着较流畅的人力比例，使资金流向、雇用、训练、测试和运算资源的使用跟着流畅起来。

图5-3中所描绘的动态并没有真正显示出各项活动花费了多少时间，图5-4说明了一个项目中的这些活动所需时间的分布情形。

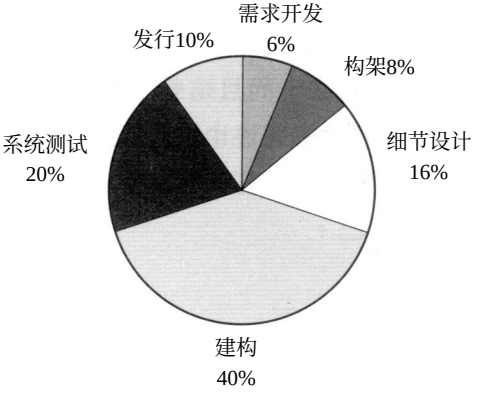


图5-4　按消耗的努力比例描绘的动态分布图

注意，图5-4的“发行”活动不像在图5-3中是分开列举的。它包含取得所有项目资助者同意发行的时间，建立项目的最后记录，建立项目的历程报告和其他项目结尾的活动。

图表中的具体数据都是依照经验法则得来的，所以不是很严格的数字。不过它们还是提供了一些实际的概念。需求开发跟构架的上游活动消耗了项目努力中相对较少的部分，而下游的构建跟系统测试活动则消耗了太多的时间和精力。上游活动对下游活动起着极大的杠杆作用，两者都需兼顾是很重要的。

图5-5补足了上图，说明项目活动间的典型时间分布。

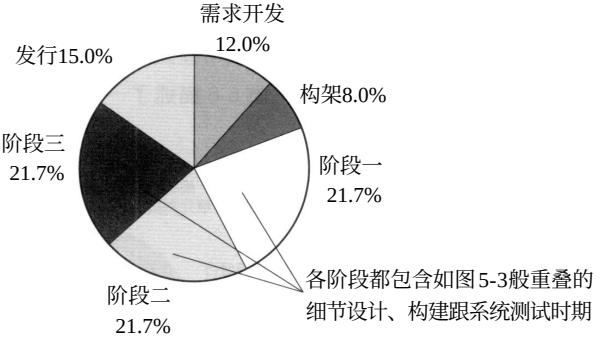


图5-5 假设将项目切成三部分，按消耗时间比例分布的活动状态（由于四舍五入的问题，图中的数字加起来不会刚好是100%）。软件项目中的时间精力分布不会刚好跟本图一样

从这些图表中得到一个很重要的概念是，项目中进行某一活动的时间比例不会跟耗用的精力成正比。需求开发工作一般会耗去项目12%的时间，可是只占用6%的努力。由于上游活动比较抽象而需要更多的深思熟虑，所以必须以较慢的步调进行。

程序代码增长曲线

前面的图表应该已经说明在项目初期有许多不会产生任何程序代码的工作要做。其实项目的头三分之一是用来详细了解需求与发展高质量的构架方式，好让项目团队能够好好检测项目规格，然后一次把产品做好，程序代码可能会较慢写出来。项目中间的三分之一主要在建立项目软件上，在这一阶段程序码会快速产生出来。项目的后面三分之一，则将焦点摆在检查前面阶段写出来的程序代码是否上得了台面。这一阶段着重错误修正和根本程序代码的更动。如同开头三分之一，程序代码增加得很缓慢。图5-6描述了一个执行良好的项目中程序码增长的方式。

黑线表示正常的程序代码增长方式，阴影部分则表示正常变化量。在项目中期程序代码增长的变化量仿照过渡版本提升现有程序代码品质而产生的。图 5-6中的项目在

最后推出产品以前，推出两个过渡版本。

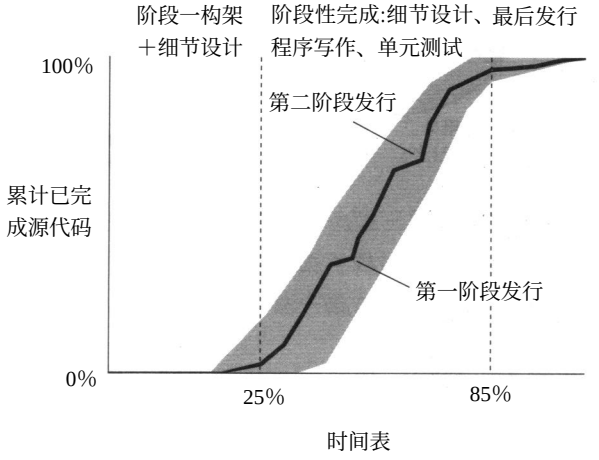


图5-6　典型的项目程序代码增长方式。大部分的程序代码都在项目中间三分之一的时间内增加而成的

一旦你了解了程序代码增长的方式，你就能很精确的估计项目的现况。执行良好的项目每周都会记录项目的程序代码。如果你认为项目快到结尾，新程序码几乎也停止增加，这时项目就已经可以准备发行了。如果新的程序代码还不停的加入项目中，那项目就还处于执行中期的三分之一，还没接近推出的成熟阶段。

同样的道理，如果开发人员在他们完成构架设计前加入太多程序代码，你几乎可以保证会在系统测试阶段停留

许久，那是为了要让开发人员修正他们在系统设计还不是很成熟时所造成的错误。

一些项目犯下的严重错误是将产品在只完成了85%左右时就发行出去。

如果项目主持人不了解图中所说明的软件开发方式，他们会假设新的程序代码开发量开始降低时就可以立刻把程序推出去发行了，特别是项目执行受到明显的时间压力时。这个将产品只有85%完成度就仓促发行的决定，表示软件终究未臻完善，这样的决定就像搬砖头砸自己的脚。

主要完成点与推行点

有时候，前几节中描述的一般软件开发方式可缩减成以图像表示详细的完成时间点和推出时间点。主要完成点以极高水平来追踪项目执行进度，图 5-7概述了本书中使用的高层次阶段和完成点的划分方式。

这里的通用项目要点几乎可以应用到任何规模的项目上。实际上每个阶段的初步工作常可以比图中所说的更早开始进行，而后头的工作可以比图中所示的要晚结束。本图说明了每段时间中主要强调的重点都放在个别的活动上

（图5-3已经更完整的提到过各项活动重叠的情形了）。

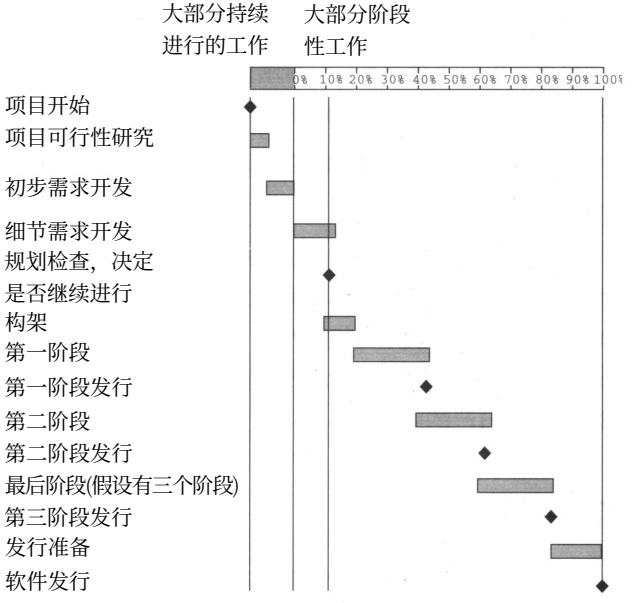


图5-7　高层次项目阶段与完成点划分方式。一个执行良好的软件项目能够依循这里的一般性规划，不管项目规模如何

上层主管与客户有时对于软件项目中的完成点模糊不清、无法依据而感到泄气，不过如果你依循着本书建议的做法，以完成点的方式追踪项目进度，将可以对项目状态得到良好的了解。表5-1列出这些完成点与图5-7的高层次阶段划分方式相对应的详细活动。

表5-1　高层次完成点与推行点的划分方式

◆ 项目初期
☐ 找出项目关键决策者
☐ 建立、检视并按照前景叙述进行项目
☐ 建立软件的业务状况评估
☐ 建立、检视初步努力与时间目标
☐ 团队中拥有2~3名资深开发人员
☐ 建立、检视变更管制规划，定案
☐ 建立、检视十大风险清单，并以此避开已知风险
☐ 开始纪录软件项目的过程
项目这时期所进行的工作没有的明显终结点。这时期的工作是用来了解项目规模而进行的。这时期耗用的时间和精力会随着项目的不同而有极大变化
◆ 项目开启／可行性研究完成
☐ 质量保证主导者就位
☐ 文件说明主导者就位
☐ 找出重要使用者，进行访谈
☐ 建立简单的使用者接口雏形，由使用者审查直到能被接受后定案
☐ 使用者接口格式简介建立，经过检阅后作为日后说明文件的基础
☐ 建立第一次项目评估（精确度在 -50%~范围线+100%的范围内），检视评估结果后定案
☐ 建立初步软件开发规划，经过审查后，据以执行项目过程

(续)

☐ 更新十大风险清单

☐ 更新软件项目记录

项目在这部分的工作也是不受结尾限制的，而且依照项目性质而决定

◆ 初步需求开发完成

在这时期，项目中开放性结尾的工作都已经完成了，现在需要一些检查来决定要不要继续进行项目

☐ 建立细节使用者接口雏形，经过检视后定案

☐ 建立使用说明/使用规格，经过审查后定案

☐ 建立软件质量保证计划，经过检查后定案

☐ 建立细节软件开发规划，经过检查后定案

☐ 更新项目评估（精确度在-45%~+75%的范围内）

☐ 更新十大风险清单

☐ 更新软件项目记录

到这一阶段中约花费12%的项目时间跟6%的人手精力。这些比例不包括在项目开启/可行性研究与初步需求开发上的工作时间跟努力成效的花费

◆ 细节需求开发完成

◆ 审查规划，决定要不要继续进行项目

☐ 开发团队大致就位

(续)

☐ 管理人员大致就位

☐ 完成使用说明/使用规格后撤除撰写文件的人力（除非有别的文件产品要写）

☐ 建立软件构架文件，检查后定案

☐ 建立软件整合程序，检查后定案

☐ 建立阶段性完成规划，检查后定案

☐ 建立第一阶段的软件测试项目，检查后定案

☐ 更新使用说明/使用规格

☐ 更新项目评估（精确度在-30%~+40%）

☐ 更新十大风险清单

☐ 更新软件开发规划

☐ 更新软件项目记录

到这阶段，约耗费20%的项目时间与14%的精力

◆ 构架完成

☐开发团队完全就位

☐ 品管人员完全就位

☐ 初步阶段规划完成

☐ 建立第一阶段的细节设计文件，检查后定案

☐ 建立包括小型完成点的第一阶段的细节软件构建规划，检查后定案

☐ 建立下一阶段的软件测试项目，检查后定案

(续)

☐ 更新第一阶段软件测试项目

☐ 建立第一阶段软件建立指示（产生档案）

☐ 建立第一阶段软件源代码，检查后定案

☐ 创造安装程序，检查后定案

☐ 更新使用说明/使用规格

☐ 第一阶段“功能完成”产品

☐ 更新项目评估（精确度在-20%~+30%）

☐ 更新十大风险清单

☐ 更新软件项目记录

假设项目分三个阶段，到这阶段约耗用 45%的项目时间与40%的努力成效

◆ 第一阶段程序代码完成

☐ 各项活动同上

到这阶段约花费65%的项目时间与65%的努力成效

◆ 第二阶段源代码完成

☐ 建立最后阶段细节设计文件，检查后定案

☐ 更新所有阶段软件测试项目

☐ 更新所有阶段软件源代码

☐ 更新所有阶段软件建立指示（产生档案）

☐ 更新安装程序

(续)

☐ 如果软件是个业务系统，完成使用文件（清楚的使用说明），完成使用者训练，使用团队准备上路

☐ 整合已完成的“功能完整”产品

☐ 更新项目评估（精确度在-5%~+5%）

☐ 更新十大风险清单

☐ 更新软件项目记录

到这阶段，约花费85%的项目时间跟90%的努力成果

◆ 最后阶段源代码完成（如果分三个阶段，这就是第三阶段）

☐ 建立发行检查项目，检查后定案

☐ 所有成员同意发行产品，不再更议产品

☐ 完成功能正确的产品

☐ 完成功能正确的安装程序

☐ 完成最后测试项目

☐ 完成软件正本的复制

☐ 项目成果媒介（源代码、建立环境等）归档存放

☐ 更新最后软件项目记录

☐ 建立项目历程文件，检查后定案

到了这阶段，用去100%的项目时间与努力成效

◆产品发行

有时人们会想，为何软件项目要花那么久的时间。表

5-1中的推进项目清单有助于回答这个问题。每个推进项目都代表一些必须完成才能让项目有效推进的重要工作。

大部分执行成效糟糕的项目最后都得做同样的事情，由于它们缺乏管理，执行起来也缺乏效率，最后花费代价更多却没得到什么好处。

人们抗拒“开发程序”的一个理由是开发程序导向的项目在一开始就让人们看到一张这样的工作清单，使他们觉得“把这些事情都做完的话，项目就永远做不完了！”事实是如果项目中不做这些事，就得花更久时间才完成得了。即使小型的项目也得处理表5-1中列出来的绝大多数工作项目，虽然有些工作在小型项目中做起来不会花什么时间。我们也不希望做那么多工作，不过对软件项目而言，忽视一定得做的事情恐怕会功亏一篑，而如果人们知道一个项目中从开头就有哪些事情必须做的话，每个人都会好过得多。

求生检查

- 项目采用阶段性完成方式。
- 上层主管、顾客或两者都依据程序代码增长曲线盯紧项目进度。
- 上层主管、顾客或两者都留心着主要完成点和推行点的达成。
- 可从本书网站 http://www.construx.com/survivalkit/ 取得本表的电子档案。

Part II

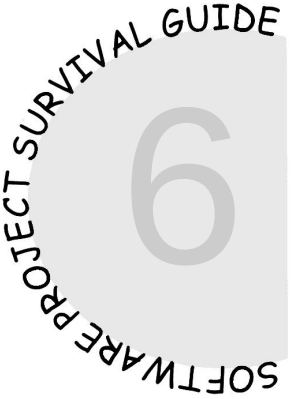
SURVIVAL PREPARATIONS

第二篇

求生准备

第 6 章

命中移动目标



有效率的项目能控制变动；没效率的项目则被变动控制。成功控制变动的关键包括成立一个变动控制小组，在项目中限制预定事项的变更，对主要成果进行变动管制。

如果你确切知道目标的移动路径，你就容易命中移动的目标；你省下跟着目标移动的精力，而命中对你而言是静止的目标。随着市场变动与科技发展，软件项目常需瞄准移动的目标。有些变动是不可避免的，其他的则是可以掌握的。不积极控制变动的项目简直是将自身置于敌暗我明风险之中。控制变动并不是事后补救措施，而是项目规划中影响软件项目成功的一个关键整合部分。如Gene Forte所说，“控制变动”与“放任变动”形成显著对比。

“控制变动”是评估、驾驭与确认项目进行中可能出现任何变动的重要法门，也是确定所有项目资助者都了解变动所会带来影响的做法。控制变动要先确认变动对系统的影响程度。

变动管制程序

在最基本层次上，变动管制处理需求和源代码变动。较先进的项目将变更管制透过项目活动（项目规划、评估、

需求、构架、细节设计、源代码、测试规划与文件）进行整合。基本的变动管制程序包含下列步骤：

1. 工作成果（如项目需求）一开始在变动管制程序没介入的情形下进行开发。在这时期，可以任意对工作成果进行变动。
2. 工作成果提交技术检查会议，决定开始的开发工作是否可以宣告完成。
3. 当初期开发工作完成后，工作成果就提交“变动小组”。变动小组一般包含与项目相关的人员与代表：例如项目主管、市场人员、开发人员、品管人员、文件部门和使用者代表（这小组按照功能，称作“战略小组”、“沙皇撤换小组”等类似的名称，取决于项目团队将自己当成执行委员会、军事团体或帝俄贵族而定）。在小型项目中，变动小组可能只有一两名或三名成员。在集团化公司的最大型项目中，这样的小组膨胀到 30人以上。重要的目的是有个确保所有重要观点都被中央决策单位考虑到。 在提交变动小组后，工作成果就被“定案”了。在这时，对成品的任何进一步变动都要经过比第一步骤更系统化的变动程序来进行。

4. 对工作成果进行修订管制。“修订管制”是指使用一套能够电子化存放多个版本内容的软件修订管制程序（又被称作“版本管制”或“源代码管制”）。尽管这套系统最常用在源代码上，大部分修订管制系统都可以存放任何可以以电子档案存放的东西——文件、项目规划、电子表格、流程设计、源代码、测试项目等等。
5. 对工作成果的更进一步变动都以系统化的方式进行：
   1) 透过变动提案提出。一个变动提案描述工作成果的问题、变动方式建议、直接受影响的关系人的看法（成本和收益间的考虑）。在最小的项目中，这种做法都是好点子，因为这样提供了比人们的事后回忆更为可靠的一项决策方法。
   2) 变动小组指明会受到变动影响的人员，将变动提案交给这些人员进行检查。
   3) 相关人员各自估计提议的变动在各自负责段中需要多少成本，可产生多少好处。
   4) 变动小组结合各方面估计，排定变动提案的优先级——决定要接受、否决还是延后处理。

5) 变动小组将提案结果通知相关人员。

我说过这个过程有点正式，看起来似乎很官僚。不过如果你细心阅读每一步骤的说明，你会发现这些步骤只是将常识规范化而已——在变动之前评估变动影响，让受影响的人检查变动的提案，并在变动提案被认可后通知相关人员。不然还能怎么做？不要在变动之前评估影响？不要让受影响的人先行检查？不要让受影响的人知道提案通过了？这听来滑稽，可是这就是没采用系统化变动管制程序的项目中经常出现的现象。

变动管制益处

变动管制程序有几个显著好处。它的主要好处在于按照程序办事，——保护项目免于不必要的变动，保证变动提案都被系统化地考虑过。不必要的功能是软件开发风险中最严重的一种，这样的功能增加了软件复杂度，在设计与程序中产生不稳定的效果，而且增加所需的成本与时间。

变动管制增进软件的决策质量，确保相关人员都参与决策。同样这种方式也增进了必要变动的透明度，确保相关人员都在考虑变动时被通知，并让他们知道变动提案何时通过，如此使得项目团队追踪进度的能力也随着提升。

变动管制也提供了“模糊完成点”的反制。软件项目中一个潜在的危机就是项目团队达到一处完成点后，即使并没有真正满足完成点的要求，也宣告目标已经达到。项目团队也许没有真正建立一份完整的构架文件，可是当构架文件的完成日到期时，他们就宣告他们完成目标了，不管做好的东西称不称得上是一套架构，特别在他们有时间压力，却又野心勃勃地要如期完成时。

在变动管制下，构架（或其他工作成果）必须被检查、认可，并在完成后置于变动管制系统的保管下。如此一来一套含糊的构架要想瞒过所有相关人员检查通过完成点就困难多了，而且当这些人知道有什么更进一步的变动要进行时，变动的提案一定会经过系统化的变动管制程序的处理。

这样消除了模糊完成点，也改进了状态可见度。如果你知道完成点很实在，那项目一旦达成完成点目标时就是个有意义的指针。

描述这种好处的一般方式就是变动管制的改善项目一定要说明清楚。相关人士得在工作成果上看过并签名，才能让这些东西归档。对已定案成果，提案变动的人得说明为何需要变动，而且他们的理由会成为项目永久记录的一

部分。反对变动的人亦得解释为何不需变动，而那些理由也一样得被永久记录。项目常碰到的麻烦就是缺乏说明。

在成功的项目中，项目成员主动寻求影响自己与他人工作的说明。

变动管制的一个必然好处是版本划分

自动化修订管制软件让项目成员更易于取得任何项目中的任一版本文件。他们可以取得初期项目规划、修订过的规划版本、目前的项目规划版本，也可以重现任何曾经发出给顾客的产品版本。这比许多未经变动管制的项目要好多了，你将拥有项目的详细历程记录，并能够重新追寻估计、雏形、设计及源代码这些在项目过程中开发过的工作成果。

一个自动化的修订管制系统让所有项目文件在任何时候都可以公开取得。任何需要检视项目规划、需求、设计方式、程序写作标准、使用者接口雏形或其他工作成果的人都可以从修订管制系统中取得需要的工作成果。不必担

心遗失这些工作成果，也不必担心在需要时看不到工作成果。要取得一份文件的目前版本并不需要去找出原始拥有者。虽然这看来是个小优点而已，只要是曾在可以公开取得项目文件部门中工作过的软件开发人员是绝不希望再去不公开项目文件的部门里工作的。

大部分修订管制软件也能替项目状态评估产生概略信息。这些软件能够产生关于源代码每周在项目中增加、变动或删除的行数。

一般变动管制问题

第一次应用变动管制的机构通常会面对如何处理的几个常出现的问题。

如何考虑变动

变动管制小组一般会考虑下列因素来决定如何解决一个变动提案：

- 变动的预期获益如何？
- 变动如何影响项目成本？
- 项目时程如何受变动影响？
- 软件品质如何受变动影响？

- 变动如何影响项目资源分配？变动会增加项目重要部分的人员工作负担吗？
- 变动可以延后到项目的下个阶段或是下个软件版本再进行吗？
- 这时在项目中进行变动会不会让软件出现不稳的危险？

何时考虑变动

变动小组——一般在项目初期阶段每半个月开会一次，在项目中期则在两阶段之间进行，接近软件推出日期时则常召开。

变动小组可以在任何时候通过变动提案，不过项目人员会要求评估影响的项目中应该试着限制提案次数。在项目初期，在需求开发与构架阶段可以随时提出变动提案并加以考虑。越到末期，开发团队就越不应该受到变动要求的打扰，只能有一些重要的变动要求才可以去影响开发团队。变动小组的部分工作是作为开发团队与变动提议者之间的缓冲，评估变动影响，使项目团队不必从工作上分心。在项目的后期阶段，变动会议应该收集变动提案来进行分批考虑。在阶段性完成的方式下，阶段之间的时间是考虑

分批变动的好时机。

大致上，该多久考虑一次变动取决于开发团队、顾客、主管跟其它项目资助者的性质。如果资助者对整齐有效的运作发放奖金，他们一般会要求尽少变动。如果资助者对于重大问题的速审速决能发给奖金犒赏，他们当然会常考虑变动。

如何处理小变动

变动小组应该区分出如修正小缺陷等不会产生重大影响的变动。这些变动可被考虑且被变动管制小组一致接受，或者变动管制规划可以自动化地通过这些变动。举例来说，变动管制规划也许自动化通过可以修正系统当机器或软件计算精确度受影响时。

如何处理政治问题

对于没接触过正式变动管制程序的人来说，这个程序开头似乎很麻烦。在熟悉这程序后，你将发现它真是“物美价廉”。对于小型变动，通常可以在变动会议进行时就评估好，而完成步骤5(1)~5(2)的时间也许就只有几分钟或更短。

进行变动管制的一个初期效果是比起预期的变动要少得多。有些人会觉得不可能有任何变动会通过得了。尽管

变动会议在项目中看来也许有点官僚化，事实上却永远可以自由选择更多的提案。在进行变动管制以前，变动的影响通常没被通盘考虑过；一旦这些变动都被完整考虑了，就只有少数的变动会被认为是有必要的。这段时期代表着从变动控制项目到项目控制变动的重要转型期。

有些极力反对变动管制的意见来自那些过去不给足够时间通盘考虑影响、却又强行通过有利于己的变动的人士。在开始采用变动管制的做法后，这些人再也不能如往常一般让变动意见强行过关。变动管制的一个好处就是使变动的影响得以规范化。

习惯于各行其事的人们还是可以在系统化变动管制下各得其所，不过他们得经过详细清楚的说明程序来得到自己想要的东西。

开发人员得明白，这对某些人来说是困难的转变，而且应该预先做好准备。

要把哪些工作成果纳入变动管制之下

变动管制规划应该包含一份纳入变动管制的工作成果清

单。至少这份清单应该包括表6-1中的工作成果项目。

表6-1　纳入变动管制的工作成果

工作成果
变动管制规划本身
变动提案
前景叙述
十大风险清单
软件开发规划，包括项目成本与时程评估
使用者接口雏形
使用者接口风格说明
使用说明/需求规格
管质规划
软件构架
软件整合程序
阶段性完成规划
个别阶段规划，包括小型完成点的时程安排
程序写作标准
软件测试项目
程序源代码
产品中使用的媒体资料，包括图形、声音、影像等等

(续)

工作成果
软件编译建立指示（程序建立档案）
各阶段细节设计文件
各阶段软件构建规划
安装程序
使用文件（简单的使用手册）
发行检查清单
发行认可文件
软件项目记录
软件项目历程文件

表5-1中的每一项工作成果都应该在“定案”后纳入变动管制下，如“高阶完成点与施行项目”。

这份工作成果清单代表纳入变动管制下健全的最小施行项目。当你发现一组如本清单中的工作成果时，你也许会认为，“那会带来许多负担与额外的工作”。也许没错，它大概会让项目多几个百分比的负担，不过并没有其他能够提供项目状态可见度、降低风险和项目控制的好办法，简短说来，就是大幅增进项目成功的机率，而不用或多或少的建立与控制这些工作成果。由于这些工作提供的绝大

好处，就事论事，接受这种特殊负担的决定不光是付出一个好代价，而且是付出一个出色的代价。

对任何项目来说，第一次建立这些工作成果需要许多努力。在第二个或第三个项目里，开发团队就从先前版本的类似成果中直接修改出其中的许多项目来产生符合现状的版本。

交付变动管制

为了让变动管制顺利运作，项目与项目所属组织必须做变动管制的工作。这些工作必须达到几个层次。

软件变动管制运作必须先经规划。本章描述的变动管制规划（程序跟工作成果清单）应该在明文的变动管制规划中表示出来。软件开发规划（下一章中将提到）应该将变动管制规划当成正式软件开发程序的一部分。

必须给项目成员一些时间来执行各自的变动管制任务。至少，各项目成员最后应该花些时间来评估变动提案的影响。有些项目成员也会花时间出席变动会议。

组织必须全面接受变动会议的决定。如果项目主管获市场行销部门同意可越俎代庖，或者软件开发人员不通过变动管制程序就自行变动软件，变动管制就毫无意义了。

求生检查

- 项目拥有变动会议。
  - 变动会议的决定可被主管人员、市场行销人员或顾客驳回。
- 项目有份众人认可的明文规定的变动管制规划。
  - 没给项目团队成员足够时间执行变动管制规划的内容。
  - 工作成果没有真正纳入变动管制。
- 变动提案在处理以前由所有项目相关人员评估。
- 变动小组将变动提案处理的结果通知项目相关的人员。
- 变动小组让项目团队分批评估变动提案，使得开发人员不致因持续受到变动要求的干扰而分神。
- 本章的一些文件模板可以在本书网址http://www.construx.com/survivalkit/ 的网站上取得。

第 7 章

初步规划

成功的项目都会及早开始规划。初步规划包括订定目标前景、找出一名主持推动者、设定项目规模、管理风险和安排使用人力。这些初步规划陆续都会被涵盖于软件开发项目中。

你也许认为在确认项目的需求以前，没什么好规划的，其实不然。在投入需求的开发工作以前，初步规划是有用而且重要的。下面就是应该先处理好的事情：

- 项目目标前景
- 主持推动者
- 项目规模目标
- 公开规划与进度
- 风险管理
- 人事策略
- 时间安排

这些方面的讨论构成了本章的大部分内容。

项目目标前景

在项目开始进行以前，需要先设定一个共同目标。如果没有共同目标，将无高效率的合作默契的团队。一项对75个团队进行的研究发现，在各团队有效运作的项目中，都对自己的目标有着清楚的认知。

享有一个共同目标是放之四海而皆准的原则。对项目目标的认同有助于简化问题的处理，也有助于集中焦点，避免浪费时间在细枝末节上。共同目标可使团队成员之间

建立起信任感，因为他们知道彼此都朝着同样的目标前进。团队可以自行决定，然后执行已决定的事务，而不用争吵与重新复议。有效率的团队能建立起相当程度的合作默契，发挥出1+1＞2的效果。

有效率的目标可以产生激励效果，要达成这一点，必须先振奋团队士气。团队需要面对不断的挑战与不停的任务。团队不该订立出庸俗无奇的目标。像“我们想要建立市场上第三好的网络网站设计程序，并比标准落后25%的时间完成”这样的目标无法令人信服。

对挑战的响应往往夹带情绪，而且很大程度受到工作安排方式或性质的影响。不妨换个方式来表达刚刚那个乏味的目标：“我们要建立一个网络的网站设计程序，让我们市场占有率在头六个月内从0爬升到25%的地位。我们的资金不足，所以我们要设定功能与期限来达成目标，好让我们用一支小巧而高效率的团队，在资金耗尽以前，在市场上立足”。如此一来，一支目标明确、非常棒的团队就会卓然成形。

好了，振奋士气的前提已达成，前景叙述也可达成，销售与行销人员也向不可能的目标挑战，可是软件开发人员却将这样的目标视为不合逻辑而兴味索然。同样的道理

也应用在分击可行、合却不利的情形下。短时程、低成本及多功能也许分别可以达成，但是不可能同时达到。这样的目标会被大部分开发人员当成只有作假才能办得到，自然容易半途而废。

在一些项目中，你没办法在初期分辨哪些目标可以达成。一个常见而且致命的状况是管理阶层订立的目标，开发团队认为不可行。如果管理者继续坚持目标是可行的，团队士气就会受到打击。如果你是一名软件项目主管，当开发人员开始告诉你项目目标不切实际时，你就得小心了。

确定不用处理的事情

前景目标应该很清楚能判别出软件中该做与不该做的事情。一个像“建立世界上最好的文书处理程序”的前景叙述也许有激励作用，可是开发团队却认为不要只是告诉他们应该把所有想像得到的功能都纳入软件中，Microsoft Word 在Windows上第一版的开发工作就受到这类混沌不明的前景所束缚。那项产品开发了五年，比起原先规划的多了四年。一个如“建立世界上最易用的文书处理程序”的前景叙述虽然能振奋团队士气，但是对于哪些工作应该排除则交待不清。

等量齐观地建立一套排除跟纳入的项目是建立前景叙述中最困难的一部分，可是这样的效果对于前景叙述不可或缺。

一组良好的、排除性的前景叙述有助于产品简化目标，而这将有效地驾驭项目风险。

保证达成目标

项目的前景目标应该正式明文记录。当团队成员投入工作时，前景目标也应该是第一个纳入变动管制的文件项目。一份前景叙述如果在项目进行中草率更动，就不能提供最高原则的指示作用了。当开发中的软件逐渐成熟时，前景叙述也应该能够有效变动。随着时间的推进，不同项目的前景叙述整合后就会变成新项目的重要资源，尤其是项目被分类出各种不同效果时。

主持推动权

主持推动权是赋予拥有整个项目最后决策权威的人或小组的。许多调查发现有效率的主持推动权对于项目

成功是不可或缺的。因此项目规划中应该找出主持推动者。这个人或小组应该对功能的完成、使用者接口设计的认可、及软件是否推出的决定负责。如果决策权力掌握在一个小组手上，小组中的每个人都应该代表不同利害关系，如管理、行销、开发、品管等等。有时这个小组就是第6章“命中移动目标”中所描述的变动管制小组。

在我开始工作的初期，我在一个有五位老板（一种失序的管理体制）的项目中工作过。这些老板常常以不同方式更改我的工作内容，让我花了两年才完成一件预定一年即可完成的项目。我觉得像个傻瓜一样被人从五个方向同时拉着，而我们的软件看起来就像以五种不同方式扭曲过了的傻瓜模样。确定决策应该来自同一处，不管是一个人或一个变动管制小组。有个单一清楚的决策权威是有效运作的项目基础。

项目规模目标

在项目的多项工作完成前，对预算要求、时间表、人手安排与软件功能特性有个概念是有用的。这并不是预先估计结果，只是个实验性的目标。当项目推进时，项目团队会

为想像中的软件投入大量的工作，然后会出现几种可能性：

- 你发现一开始的项目预算跟时间目标与想像中十分吻合。
- 你发现一开始的项目预算跟时间目标无法达成预期功能，必须增加预算和时间。
- 你发现开始的项目预算跟时间目标无法达成预期功能，必须缩减功能。

如我在图3-6中解释过的，软件开发是个持续改善的过程，从中间的每个过程都会了解到更多软件技术。执行良好的项目难以避免地会在功能、预算与时间安排上进行一些取舍。

最好的机构能在项目进行中经常重新估计目标，并定期依据重新估计结果调整项目规划。

世界上最成功的一个软件开发机构，NASA的软件工程实验室（SEL）在需求规格订定后才建立最初的预估目标，并在项目中如表7-1的估计点改善估计结果至少五次。在每个估计点，开发团队都会进行一次基本估计。然后

由基本估计与表7-1中的“上限”跟“下限”数字相乘来表示。此外，SEL的《经理手册》(Manager’s Handbook)建议基本估计一般会在项目进行中成长40%（这种关系描绘在图3-6中）。

表7-1　项目进行中的工作量估计改善

估计点	上限	下限
需求定义与规格完成	×2.0	×0.50
需求分析完成	×1.75	×0.57
初步设计结束	×1.4	×0.71
细节设计结束	×1.25	×0.80
创作结束	×1.10	×0.91
系统测试结束	×1.05	×0.95

考虑开发完成后的项目估计结果中的不确定性，在产品概念时期，想要建立最佳的时间表与预算目标的依据，在于帮助开发团队找出无法顺利完成目标的因素，及早消除这些因素。这样一来就可解决过度复杂与华而不实所带来的主要风险。

更详细的估计本章稍后会再提，并在第11章“最后准备”中进一步解释。

公开规划与进度

在不成功的软件项目中一个常见的特征就是规划的秘密进行。通常没有任何规划人员想秘密进行，可是他们也没有着意让项目中的其他人员参与规划内容，这种事常常发生。缺乏开发人员、测试人员与文件写作人员这些实际进行工作者参与的规划就不能处理所有问题，而且规划内容不切实际，无法依循，最后项目只得以放任自由的方式进行——这就是失去控制了。

项目规划应该经过那些要执行规划内容的人的检讨与认可。这可不是开玩笑的，难道一名主管建立的规划内容得经过他的部属认可吗？绝对是的。如果项目团队不认可，规划本身实际上就不会被遵循。

有效率的软件项目主管如同一名交响乐团指挥、一名协调团队工作的人。主管本身并不具备完成项目各部分的所有专业知识，所以才必须将各参与者的意见统合起来。

软件项目在项目的管理阶层与执行团队产生对立时，就不可能成功。主管不应该哄骗或催赶开发团队进行工作，而应尊重他们的意愿。开发团队对管理阶层的唯一要求就是有效率的协调活动，好让他们的工作努力不致浪费。精明的项目主管会在大量的工作开始进行以前先确定规划方

案并经过项目团队的认可。

在一个健全的项目中，所有规划内容都会进行公开检讨 —— 生产力、时程表、时程垫补、风险、工作指派与每个规划项目。

公开进度指标

一旦经过规划检查、认可并纳入变动管制中，项目的核心指标就清晰可见了。目标是让所有项目资助者都能看到项目基本状态。他们看到的应该至少包含下列信息：

- 已完成的工作清单。
- 缺陷统计。
- 十大风险清单。
- 用去的时间百分比。
- 用去的资源百分比。
- 给上层主管的项目管理状态报告

有一个达成这个目标的方法就是建立项目企业内部网络网页，提供包含项目规划、追踪信息、技术工作成果与项目成品在内的一般项目信息。图 7-1 中就是一个这样的

网站范例。让这些内容在网站上公开，在企业内部网站上，你可以取得项目的最新信息，却不用了解该如何使用项目的修订管制系统。

图7-1 包含软件项目最新状态、规划信息与项目关键工作成果的企业内部网页

成功的软件项目没有秘密，无论消息好坏都必须在各阶层成员间流通而不受限制。

风险管理

“风险”在字典上的定义是“可能失去东西或受到伤害”。软件开发是一项包含可能损失的活动，无论开发过程如何进行都以超出预算与时间延长的形式出现。软件开发工作进行的方式很少能保证让开发工作一定成功，需要冒一定的风险。如图7-2所示，我发现一般项目几乎不关心风险管理，结果遭受了极大的风险。成功的项目由于承受了少量负担来管理风险，因而降低了暴露在风险下的危险。

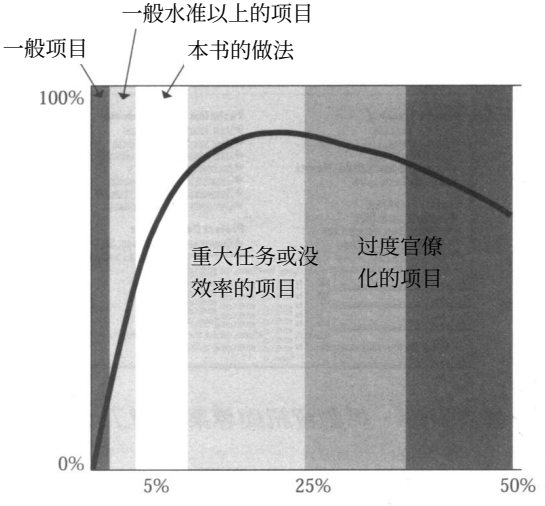


图7-2　风险层次的变化视风险管理的负担而定

投入少量精力在风险管理上能带来巨大的好处。在

本书的做法中，约 5%的项目工作用来进行“ 风险管理 ”的活动。这一少许支出让时间与预算目标满足的机会大幅增加。本书的做法应该能给大部分项目 50%~75%如期完成的机会。

成功的机构积极找寻以少量的负担换取降低大量风险的各种方法。

某些项目需要更进一步降低风险。为了达成这点，他们被迫缩小要求结果，如图右边的亮灰色区域中的那些。不过在其他项目中，如受限于纸上作业影响的美国国防部的项目，就在图中极右方，必须投入大量预算以确保项目的如期完成。

由于软件开发本质上存在的危险性，对于大部分中型到大型项目要达到百分之百迎合时间与预算目标，而不加上大量安全限制是不可能的。在一些程度的负担下（如图中约略描绘的），这些负担本身就成了项目的风险，也降低项目满足时间与预算目标的机会。

达成风险管理

风险管理的成功取决于如何达成、发展、执行与检验每一个步骤。如果遗漏任何一点，风险管理就没效果了。

风险管理的达成包含三个要素。首先，项目规划必须在做法中写下如何进行风险管理，如本书所描述的。第二，项目预算必须包含解决风险所需的经费。如果没有经费，就无法达到风险管理的目标。第三，评估风险时，风险的影响也必须纳入项目规划中。有些项目执行风险评估，可是却不作出任何对策，那就和把你的电话转到语音信箱上，却从不去检查语音信箱中的留言差不多了。

发展风险管理的能力有点难又不会太难。只要项目团队按照本节描述的去执行就行了。但是有些机构却阻挡了风险相关信息流通给上层主管及需要的人，这节的规划方式就有助于改善此问题。

本书的做法采用了一些主要用来管理风险的方式。所描述的其他做法虽然不是专门针对风险管理，可是已经为了风险管理的好处而纳入其中。建议做法包括下列这些：

- 在软件开发中规划风险管理（这正是本节在说明的东西）。
- 找出风险管理者。

◆使用十大风险清单。

◆对每种风险建立一份风险管理规划。

◆建立一个匿名风险反馈通道。

这些做法在接下来数节中详细说明。

风险管理者

项目应该找出一名风险管理者，在项目中负责随时处理冒出来的风险，最好是项目主管之外的人员担任。风险管理者必须有点胆小（“成天怕天塌下来”）而且有点神经质（“那功能根本无法执行”）。风险管理者的功用是在规划会议上与检讨规划内容时扮演魔鬼代言人。他必须不断寻找可能出现的任何状况，并试着挑选各个风险的管理策略。同时风险管理者应该受到管理阶层的尊重，不然可能就变成项目“无的放矢的悲观主义者”了。

一名资深的开发人员或测试人员经常要能担任风险管理者，因为项目主管的工作是引导项目成功进行，要项目主管接受风险管理者工作所需的悲观态度是困难的。这种方式就像为何要请独立测试者来测试程序一样。对开发人员来说，又要试着让程序运作又要同时挑剔程序运作是难以办到的。这两种工作都需要不同的观点，而项目主管跟

风险管理者的工作也是如此。

并非挑选一名风险管理者就足够了。机构内的信息流通方式与奖赏标准应该鼓励任何项目人员把风险找出来。

十大风险清单

十大风险清单是关键的风险管理工具，清单上列举出在任何时候碰到的最大风险。维护一份风险清单的简单动作，有助于让项目主管将风险管理牢记在心中。

项目团队应该在开始工作以前规划一份初步的风险清单，内容随时更新。清单上是不是刚好有十种风险并不重要，可以只有5种或15种风险，重要的是内容要经常维护。项目主管、风险管理者与老板应该每两周甚至更短的时间内检查清单上的项目。这种检查应该安排在他们每半个月的工作日程上，好让这项活动不致半途而废。更新风险清单，排定风险重要性、更新“风险解决进度”专栏，会强迫他们经常留心风险，并在风险变化时收到警讯。

为让项目得以生存，这份清单应该对所有项目成员公开。应鼓励不在管理阶层链上的人勇于发现问题并提出警告。这份清单可以列得很简单，如表7-2这样。

表7-2　十大风险清单范例

本周	上周	上榜周数	风险	风险解决进度
1	1	5	拖泥带水的需求	使用者界面雏形用来收集高质量需求规格 需求规格纳入明确的变动管制。阶段性完成的做法会提供一些变动所需功。
2	5	5	需求或开发人员华而不实的毛病	前景叙述具体说明软件中不包含哪些东西 设计强调精简。检查项目清单来检查“额外的设计或实作”
3	2	4	发行软件质量太低	开发使用者界面雏形来确认使用者会接受软件采用规范化的开发过程。 对需求、设计和程式进行技术检查。 测试规划确认所有功能都涵盖在系统测试中。

(续)

本周	上周	上榜周数	风险	风险解决进度
				系统测试由独立测试员进行
4	7	5	不可能达成的时间表	项目避免在完成需求规格前以决定日程安排 在最省事的上游时期进行寻找和更正问题的检讨 在项目进行中数次重新估计时间表 积极追踪项目进度，确保时程延误及早发现 当整个项目花费的时间比预期要久时，阶段性完成的做法可以让部分功能先被完成
5	4	2	不稳定的工具延误时间	只在项目中使用一两项新工具，其余都延用前面项目的工具
6	-	1	人员流动率高	鼓励开发人员投入的项目目标

(续)

本周	上周	上榜周数	风险	风险解决进度
				主动的细节项目规划、清楚的期望目标 定期重新估计，修订规划，避免大量加班
7	3	5	开发人员与顾客间的摩擦	提供开发人员高生产力、高动力与高记忆力的工作环境 使用者界面雏形让开发人员与顾客在细节看法上达成一致 阶段性完成的方式让顾客看到渐进的项目进度
8	6	5	非生产性的工作环境	在完成使用者界面雏形后，将开发工作换到不同地方的隐密工作室进行 需要经费认可来更换工作处所

风险追踪工具

建立十大风险清单的一个办法是将风险输入缺陷追踪

系统中（与项目缺失分开处理）。缺失追踪系统一般能将风险项目标示为已解决或尚待处理状态，也能指定解决问题的项目团队成员，并安排处理顺序。你可以将各风险项目依序列印出来，按照缺陷存在时间与负责处理者等资料来进行排列。这样，缺失追踪系统使维护风险清单的工作不那么单调。

细节风险管理规划

十大风险清单上的每一个风险项目都应该有一份细节处理方式，规划内容不需要十分详尽，可以只有一两页，可是应该包含表7-3所列问题的答案。

表7-3　风险管理规划

为何需要这份风险管理规划
为何这个风险需要规划管理？描述风险发生机率、结果与严重性
如何规划管理风险
一般如何解决这项风险？描述解决风险的一般方式是列出或描述该考虑的事项
将采取哪些对应步骤
将采取哪些步骤来解决风险？列出具体步骤和处理问题时将产生的结果，并加上不能如期解决问题而使风险提高时的状态描述

（续）

谁来处理问题
谁该负责完成各个处理步骤？列出负责各步骤完成的特定人选
何时完成各步骤
各步骤何时完成？列出各步骤的完成日期
需要多少经费
问题的解决需要多少经费？列出排除风险的各步骤的成本、风险管理规划大纲。规划应该简短，而能说明谁该如何采用怎样的步骤在何时何地来排除风险

匿名风险反馈通道

把好消息传遍整个项目团队与管理层链很少会有问题，可是坏消息却不然。项目团队应该建立匿名风险反馈通道，让项目成员可以反馈项目状态和风险。这个“反馈通道”应该和休息室的意见箱用法一样简单。如果开发人员不及时把程序交给测试人员，相关的测试人员就可以反馈这个状况。如果测试人员交给文件写作者的程序版本没有完整测试过，相关的文件写作者就可以报告这种情况。如果项目主管对上层主管夸大了项目进度，相关的开发人员也可以反馈这种不实情况。

如果你身为上层主管，有些技术人员在项目进行10%到20%后晓得项目行不通，你当然希望在当时就晓得这回事，而不是让技术人员在花了预定时间的 150%后才来说“我试过想告诉你，可是我的直接主管不让我说。”

如图7-1中的项目网站首页所示，一个匿名风险反馈通道可以整合进项目的网站首页中。采用这种做法时，所有项目成员都可以在任何时候得到一份匿名反馈的风险清单。

人事策略

要支持第4章中提到的人因工程求生技能，我建议采取“人尽其才”的做法。这种做法的背后构想是，项目主管应能对项目人力资源的强化或损失负责。如果项目结尾时，有五名开发人员辞职不干了怎么办？这对公司来说是实际的损失，而且项目主管应该负起和损失了美金250 000一样的责任。项目的整个开发团队若有着强化技能而且士气高得不可思议，那对机构本身很有好处，而且应该在主管的功劳簿上大记一笔。

人事开发

项目规划应该包含增加人手的明确规范。以下是依据

人尽其才方式的规划特点：

- 依据留住项目人员的表现进行评估。
- 所有项目成员都有机会获得晋升。
- 开发人员相信项目目标，而且在项目中觉得公司变得越来越好。

募集人手

募集一个中型项目团队的最有效办法，就是在需求阶段以资深人员成立团队，然后在构架与设计阶段逐渐增加人手。在需求分析和构架时期，大约2~5个人就能做出最好成果。在进度达到10%~20%的规划检查以前，还不需要扩增人手。较大的团队会倾向减弱软件的概念整合度，而在没提升生产力的情形下加快项目进行速度。越往后头的细节设计与构建阶段中，更大的团队则有更高的生产力。不过新进人员必须在构建阶段初期就加入团队中，不然效果有限。

对小型项目（约七名或更少开发人员），使用从头到尾项目人员数都没变化的平坦模式就可以了。

1. 新开发人员：可用与好用

在增加项目人手时，不要只因为有人就拼命采用，要

采用那些够格的。在一个为期一年的项目中，最好等上一两个月来雇用一名够水准的程序员，这样比随便找个水准较差的程序员要好多了。

软件工程研究最普遍的一个现象就是，最有效率的开发人员跟最差的开发人员之间的生产力至少有十比一的差距。这并不是意味着最有效率的开发人员要高出十倍的生产力，而是说他们比那些没本事的人要高出五倍的生产力，因为最没生产力的程序员实际上还会让项目进度落后。所以不要有人可用就好，不要雇用最差的开发人员来工作。

最好等有生产力的程序员出现，而不要期待最先找到的程序员变得有生产力。

2. 团队动力

研究员B. Lakhanpal 1993年发表了他在十年里得到的一项有趣的研究结果。研究报告中检视了31支团队，企图找出开发人员独自的能力和团队向心力何者对生产力更有影响。研究报告发现团队向心力比起项目成员的独自能力对生产力更有影响。

这项研究的结果令人惊奇。团队通常只是由技术上能够胜任工作的个别成员组成。可是这份研究报告却说当团队成立时，至少应该注意到团队成员能否合作愉快。一旦具有向心力的团队成形了，细心的主管自然就会在项目结束时多想想该不该拆散整支团队，亦或让这支团队继续进行下个项目。

一旦项目开始进行时，就不要留着惹麻烦的开发人员。在另一份对75个项目进行调查的研究中，Larson和LaFasto发现，团队成员对带领者最大的抱怨，就是没将惹麻烦的家伙从团队中剔除掉。团队成员都晓得谁在惹麻烦，从而给他们的团队带领者极不好的评价。矛盾的是，团队领导者倾向在这方面给自己较高的分数。

如果你觉得团队中有个不好相处的家伙，就尽快跟项目主管、上层主管、人事部门或任何必要的负责单位协调，把那个人调离项目团队，或者把他炒鱿鱼。这种削减团队向心力的负面效应会影响到每个项目成员，而且造成的伤害比带来的正面贡献要大得多了。更进一步，在我看过的例子中，当难以相处的成员离开后，我发现离开者的工作表现水准跟他的人际互动关系一样糟糕。晚一点剔除掉那样的团队成员只会害了大家。

3. 募集人手的关键问题

下面是一些初步的项目规划在增添人手时所应该处理的关键问题：

- 项目主管有一个以上同等规模项目的软件开发经验吗？
- 项目的资深技术人员是否有从相似的成功项目中，获得建立中的软件类型所需的知识和经验？
- 大部分团队成员的水准都差不多，对团队生产力的期望符合成员能力水准吗？
- 团队成员是否合作愉快？

团队组织

不同机构使用不同的团队组织方式，当然有些方式会比较有效率。最有效率的软件组织同意资深的软件开发人员与主管人员平起平坐，他们对待项目主管的方式就像职业球队中对经理的待遇一样，他们对团队的成功当然重要，可是没有比明星球员来得更重要。

1. 项目团队组织

除了最小型的项目以外，一个项目团队需要区分出不同成员扮演的角色。即使小型项目也需要这些角色，只不

过可能都是由同一个人扮演的而已。

- **项目主管** 负责包含开发、品管与使用者文件编修在内的项目细节技术工作指挥者，项目主管承担发展软件开发规划的工作，常成为开发团队与上层主管间的沟通桥梁。
- **产品主管** 负责在实务层次整合项目成果，在商用软件产品中，这项工作包括行销、产品包装、使用者文件编撰、使用者支持与软件开发。在内部项目中，这项工作包括与开发中系统的未来使用者一起定义出软件规格，进行使用者训练与支持，还有转换到新系统的规划过程。
- **构架者** 负责在设计与实作层次整合软件概念。
- **使用者接口设计者** 负责在使用者可见的层次整合软件概念。在内部项目中，这个角色可以由使用者支持、使用文件编撰、开发人员或产品管理阶层的人来担任。在商用产品中，则应该由一名使用者接口方面的专业人员来担任。
- **使用者联络人** 负责在项目进行中与使用者沟通，在雏形开发、展示新发行版本、导引使用者意见回馈等等工作上与使用者进行沟通。这个角色可以由

开发人员、产品主管或使用者支持人员担任。

- **开发人员** 软件细节设计与实作的负责人员。开发人员负责让软件能够运作。
- **品管 / 测试人员** 负责规划与主导测试活动，建立细节测试规划和进行测试。他们负责找出解决软件故障的所有方法。如果项目够大，这些人会有自己的小组领导者或主管。
- **开发工具制作者** 负责发展程序建立描述指令，维护源代码管制系统，开发项目中需要的特定工具等等工作。
- **版本建立管理者** 负责维护与执行每日整建工作（在第14章中说明），并在源代码造成新建立版本故障时通知负责的开发人员。
- **风险管理者** 被指派来留心风险的出现，如本章前面提到的。
- **使用者文件编修专门人员** 负责生成使用者的辅助说明档案、纸上文件与其他说明文件。

你也许会认为上面列出了许多角色，不过这些角色在大部分项目中并不一定会正式划分出来。即使是非正式负起日常事务运作的角色，只要明确标明出来，在规划上都

是有用的。在大型项目中，每个角色指派一两个人，就能让每个角色的工作性质显露出来。实际上更大型的项目会比上头列出的角色多得多。在一人扮演多重角色的小型项目中，可能会低估每个人在项目规划时的责任范围。举例来说，在小型项目中就容易忽略主要开发人员也要与使用者沟通。明确各角色的责任，有助于确保每个人的完整责任范围均被考虑到。

大部分的角色都可以依照项目性质混合扮演，不过开发人员跟品管人员的角色不应该跟其他工作混在一起。品管角色需要魔鬼代言人的心态，而开发人员是不可能或难以调适成那种心态的。

2. 特勤小组

执行良好的项目会在生命期中找出短期优先工作。项目团队也许需要决定程序库的升级能否消除从前就有的问题，也许需要深入评估竞争对手的新版本功能，或是否根据使用者意见需要扩充项目的使用者接口雏形。

在这样的状况中，项目主管可以指派一组“特勤小组”来盯紧问题。一支特勤小组是一小队快速完成单一工作的人员（通常只有一两个人而已）。需要的工作时间通常是在两个星期内，往往只要几天。一旦他们完成了指派任务，

就解散回到原来正常工作中。初步规划里头应该包含特勤小组处理项目开头时尚未决定的任务所需的时间。

注意，不同开发人员对被指派为特勤小组的反应不尽相同。有些开发人员当成是对他们在项目中贡献的认可，而在没获得任务指派时觉得不舒服。一些团队成员喜欢偶尔换个不同的工作做做看。其他人则不喜欢在主要工作中分心去做别的事情。

让特勤小组人选在项目成员间轮流指派是个不错的办法。如果项目主持人总是指派最有效率的开发人员去当特勤小组，就可能让那些开发人员因为分心而延误项目的主要工作。其他情况下，他们会找到可以把短期任务中做得很好的人选，这时就可以让这些人有更多机会参与不同性质的特勤小组工作。

时间管理

项目初期也是开始细节时间安排的时机——掌握项目成员如何利用时间。时间管理在目前项目的控制上是个关键部分，而且可以为精确估计与规划未来项目奠定基础。

时间管理资料让你能够比较预估时间跟实际花费时间，借此改善未来。你可以利用项目中不同活动所花费的时间来规划未来项目。你可以检查做得不好的工作所花费

的时间量，来看看有没有在错误发生不久时就进行修正。如果组织中没有自己的时间管理分类方式，你的项目可以采用表7-4中的分类方式。

时间管理应该在项目初期尽早进行，不然就会失去供未来借鉴的宝贵资讯。时间管理分类需要如表 7-4那样尽可能仔细。稍有遗漏，时间管理资料就没办法包含足以提供未来项目规划指引的细节。使用时间管理方式的项目通常会使用几种网络程序之中一种，让团队成员能够从自己的办公桌上输入自己的时间资料（在本书的网站上，可以找到几种时间管理程序的网站连结）。

表7-4　时间管理分类范例

类　别	活　动
管理	规划
	处理顾客/使用者关系
	处理更改
行政	停工时期
	人力准备
程序开发	建立开发程序
	检查开发程序
	修订开发程序
	对顾客或团队成员进行开发程序的教育

(续)

类　别	活　动
需求开发	建立规格
	检查规格
	修订规格
	反馈规格中找到的缺陷
使用者界面雏形	建立雏形
	检查雏形
	修订雏形
	反馈雏形中找到的缺陷
构架	建立构架
	检查构架
	修订构架
	反馈构架中找到的缺陷
细节设计	建立设计
	检查设计
	修订设计
	反馈设计中找到的缺陷
创作	建立创作
	检查创作
	修订创作
	反馈创作中找到的缺陷

(续)

类　别	活　动
增添元件	调查/增添元件
	处理元件增添
	测试/检讨增添的元件
	维护增添的元件
	反馈增添的元件缺陷
整合	自动化版本建立
	建立维护版本
	建立测试版本
	建立发行版本
系统测试	规划系统测试
	建立系统测试说明
	建立自动化系统测试
	执行人工系统测试
	执行自动化系统测试
	反馈系统测试中找到的缺陷
软体发行	准备各Alpha, Beta版或各阶段发行版
	和支援最后发行工作
资料表格	收集管理资料
	分析测量资料

注：1. 在本书网站上提供这份时间管理表的电子文件档。

2. 本书网站上也有一份注解过的详细软件开发规划文件。

软件开发规划

在这时建立的各种规划比起在项目结尾的要粗略些，不过还是应该被明文记录下来。低效率的软件项目常常有重复不断检讨相同问题的特征，因为这些问题构成项目的规划基础，将这些问题的讨论记录下来就是一件重要的工作。一份称作软件开发规划的文件应该说明在本章描述的那些规划考虑项目，包括决策权威、项目规模、规划与进度公开、风险管理、人事策略与时间管理。一旦定出了规划的草稿，规划文件应该经过检讨，并由项目主管、品管小组跟开发团队签名同意，然后如同其他重要的工作成果一般，将这份文件纳入变动管制系统中。

一旦项目中的规划都建立好了，就能够更轻松地修改成适于未来项目使用的内容，所以在最初的项目中应该花费较多的功夫来进行规划作业。

求生检查

- 项目有个清楚的前景目标。
- 没提供决定软件功能去留的工作方针。
- 项目找出了一名拥有项目最后决策权威的主持推动者或一个推动委员会。

- 对项目进行规划，而且依据规划进行的进度对所有项目团队成员和上层主管公开。
- 项目指派一名风险管理者。
  - 风险管理者就是项目主管。
- 项目有一份十大风险清单。
  - 十大风险清单没反应现状。
- 对十大风险清单上各项风险制定风险管理计划。
- 如有必要，项目宁可聘请水准优良的人员，而不是找到人就匆忙聘请。
- 项目时间管理在需求开发之前就着手进行。
- 以上所有考虑都正式记录在软件开发规划文件中。
  - 软件开发规划文件没纳入变动管制中。
  - 开发团队没能确实依循软件开发规划中描述的方式工作。

第 8 章

需求开发

在需求开发阶段，软件概念通过不同的使用者接口雏形版本与使用文件的建立而卓然成形。这种做法有可能促进最佳组合的汇集，替出色的构架工作奠定基础，消除耗时的细节需求文件，使项目合理化进行，并让使用者文件避开关键部分。

软件需求开发是以项目汇集顾客需求，并转换成系统必要规格的部分。

需求开发包含三个相关活动：

- 通过调查可能使用者之系统需求，检查竞争产品的功能，通过互动雏形等行动来汇集使用者的可能需求。
- 要求具体化，将汇集的需求项目记录到具体媒介上，如需求记录文件、项目记事板、交谈式使用者接口雏形或其他媒体上。
- 需求分析，寻找这项需求间的共性，将这些项目分析成基本特性。这属于初步的设计活动，就不在本章继续讨论了。

在称呼这些活动为需求开发时，我有点偏离了一般的术语。这些活动通常被称为“规格”“分析”或“汇集”。我用开发这字眼来强调需求工作通常不只是将使用者想要软件做什么写下来就行了。需求并不像地底下的铁矿那样等待发掘。使用者的心思就像丰富的需求来源，不过项目团队必须先播种，才能在需求收获以前建立这些需求。

需求汇集最困难的部分不是记录使用者要什么，而是在探索性、开发性地帮助使用者找出他们到底要什么。

需求汇集与规格化是毫无止境的活动。你只有在对使用者的要求达到一个清晰稳定的了解之后，才能说项目的需求开发完成了。忽略掉需求活动是种代价不小的错误。对于每个错误列出的需求，你将在下游时期付出高达50~200倍的修正代价。

本章描述一种支持这些目标的需求汇集和规格化方式。

需求开发概览

下面是我对需求开发工作所建议的一般步骤：

(1) 找出一组关键性使用者。

(2) 访问一般使用者，建立一组初步需求。

(3) 建立一套简易交谈方式的使用者接口雏形。

(4) 对重要的一般使用者展示简易的使用者接口雏形，

取得他们的意见回馈。继续修订简单的使用者接口雏形，保持简化，并持续对使用者展示和修订界面，直到一般使用者对于软件概念也能感到有兴趣为止。

(5) 开发一套依据使用者的接口雏形外观和使用感受所编撰的使用者接口风格说明，检查其内容，并纳入变动管制系统下保管。

(6) 完全扩充使用者接口雏形，直到该雏形能够完全展示出软件各部分功能。让使用者接口雏形能够涵盖整个系统层面，不过要尽可能的只触及表层内容。该雏形应该涵盖功能的范围，而非真正实作那些功能。

(7) 将完全扩充了的雏形当作定案规格，纳入变动管制系统中处理。然后要求开发出来的软件除了经过变动管制程序通过的修改之外，外观上一律符合雏形外貌。

(8) 依据使用者接口雏形，写下详细的一般使用文件。该文件将变成详细的软件规格，而且应该纳入变动管制的管辖下。

(9) 建立一套分别记录算法与其他软硬件互动关系的

非使用者接口需求文件，也一齐纳入变动管制系统中。

这组步骤对于交谈式软件的开发有着重大影响。如果软件是个没有使用者接口的嵌入式系统（如汽车行进控制系统），本章的讨论就派不上用场了。

需求开发程序细节

下面是需求开发程序的各步骤细节。

找出一组关键使用者

收集需求信息的第一步是先找出能够定义软件需求准则的使用者。你找到的使用者必须能够指出哪些功能是重要的，并且说服你相信那些功能是重要的。同样的，如果他们认为一个功能是多余的，你就应该相信那个功能是真的可以拿掉的。你所找出来提供意见的使用者中最好同时包含计算机功力高深的使用者跟一般使用者。

如果项目开发的是内部使用的软件，项目主持人应该甄选几个真正的使用者，并让他们参与项目的开发工作。如果项目开发的是对外发售的软件，项目主持人就得规划彼此的互动关系以确认谁是真正的使用者。

找出使用者意见是重要的成功因素。如果你找不到合适的使用者以致项目停顿怎么办？那也许是一种看不出来的幸运吧。与其花了大量时间与金钱建立出使用者不想用的软件，还不如花时间寻找能给你提意见的使用者。

使用者访谈

开头进行的使用者访谈应该引出初步的系统需求，用来作为建立初步的简单使用者接口雏形的基础。

20多年的设计经验使我们领悟了一件事情，那就是软件开发人员本身对于设计出使用者喜欢的软件并不是很在行。可是软件开发人员在协助使用者发掘自己的偏好方面可以做得很好，因为使用者本身对于如何设计出自己喜欢的软件也是一窍不通。

建立简单的使用者接口雏形

让雏形接口愈简单愈好。这项活动的重点在于投入更多努力以前，先让使用者从许多不同的方法中挑出自己想要的东西出来。项目团队应该开发足以让使用者观看及感受将来开发出来的软件是什么样子的接口雏形出来。例如，你要做出一个报告表格的雏形，并不用真的

让这个使用者接口的雏形能够印出报表。开发人员应该在文书处理程序中做出一份报表，告诉使用者说，“这就是你按下程序的打印按键后，会产生出来的结果。”

当项目在一个缺乏良好雏形开发工具的环境中发展时，可以考虑在一部执行Windows的机器上仿真目的平台的外观和使用感。如真正的开发环境是一台IBM大型主机，就可以用Visual Basic来模仿出黑底绿字的那种使用者界面。

雏形化的动作应该由1~3名资深开发人员组成的小队进行。这些开发人员应该对于以最少工作量展示软件外貌与感受的过程相当熟练。如果开发软件任何单一部份的雏形都需要花费超过几个钟头的时间，那开发出来的雏形中就包含了太多功能深度了，应该让雏形程序的功能层面愈浅愈好。

以这种方式开发使用者接口雏形让使用者们可以看到他们指定的具体软件外貌，减少使用者在原先不知道软件外观的情形下，看过软件后改变心意的机会。这种雏形化的方式降低了怪异的需求出现的机会，而那是传统上软件项目面对的最严重危机之一。

让使用者了解雏形就真的只是个“雏形”。
建立使用者接口雏形的一个风险就是让使用者对于项目未来进度产生不实际的期望。

如果可以，就采用使用者纸上记事板

一个使用者接口雏形化工作的低技术做法是使用纸上记事板。采用这种做法，开发人员或一般使用者借由描绘出画面、对话窗、功率列与其他他们希望在软件中看到的使用者接口组件的图形，开始进行雏形化的过程。开发人员与一般使用者开小组会议，并在活动挂图上画出画面样本。他们在活动挂图上不断重画雏形图样，直到大家都得到彼此同意的软件外观与功能，获得使用者接口的详细规格。

这种做法有好几种优点。使用者们可以自行进行一些工作，而不用知道如何使用雏形开发软件。记事板上的雏形可以迅速产生更动，代价也不昂贵。纸上记事板也消除了一些最常见的软件雏形化风险。对开发者来说，他们消

除了过度扩展不必要的雏形部分与花费太多时间适应雏形化工具的危险性。对使用者来说，记事板消除了让使用者认为有着实际外观的雏形程序，差不多就是真正的软件的问题。

纸上记事板的惟一缺点，就是有些开发人员和一般使用者没办法在纸上描绘出软件的样子。由于建立雏形的主要目的就是协助使用者将要建立的软件可视化出来，因而这种缺点就比较关键。如果你发现纸上记事板不能帮项目资助者把软件可视化，就毫不犹豫回头用电子式的雏形建立方法吧。

不断修订雏形，直到使用者对软件感到兴致盎然为止

雏形的第一版很少能够满足使用者的期望。开发人员应该让充满期望的使用者了解，唯有他们的意见加入才能让雏形修订得更满足他们的期望。他们应该对使用者解释雏形接口只是过程产物，他们希望获得使用者的批评指教。如果使用者困惑了，或是觉得软件会不好用，那并不是使用者的错误；而是在开发正式的软件前的雏形开发阶段所必须修正好的问题。

让简单的雏形化接口精致化，直到使用者认为依据它

所建立出来的软件最后将令人振奋，而不仅只是能满足他们的“要求”而已。这看来似乎像是开发人员彻底将一段毫无节制的时间花在开发一个可能没啥前途的东西上头，可是这样的上游工作对于避免下游昂贵的代价来说是很好的投资。如果开发团队是在新的领域而非熟悉的领域（就是说，在一个经验不满两年的环境中）进行开发，雏形开发的工作将会更加耗时。

这也是个提醒使用者们，雏形就只是个“雏形”而已。在一个我工作过的机构中，当开发人员们提醒使用者们说他们看到的只是个“雏形”时，使用者们会开始一齐唱着，“那只是堆烟雾跟镜子而已，要看到成果还要花很久的开发时间。我们知道，我们晓得，我们以前听过这样的话了。”使用者们已经习惯这样的提醒了，因为开发人员们很尽责地教导他们了解这一点。

建立一份风格说明

一旦使用者同意了雏形接口的一般外貌与感受，开发人员就应该建立一份使用者接口风格说明，对应用程序的外观与感受设定一套标准。这份说明文件包含画面图形在内应只有数页的长度，不过完整得足以指引雏形化过程中

剩余的工作。文件中应该包含标准按键，如确认键、取消键和线上求助按键的大小与位置，允许使用的字体种类，错误信息的格式，一般动作的注记按键，一般设计标准以及其他使用者接口中常见的元素。发展完成后，使用者界面风格说明就应该经过检查，并纳入变动管制系统下保存。

在项目中及早处理设计风格问题有助于使用者接口的一致性，也避免开发人员持续改变使用者接口以及使用者漫无限制地提出要求，以致最后完成的软件中出现许多更动。

完全扩展雏形接口

将雏形开发工作涵盖到整个系统层面，使开发人员能够将软件可视化到一个有助于构架和设计工作的程度。项目的前景叙述让开发人员能够协调一致地工作；详细的使用者接口雏形则提供完整的见解，让他们能够将工作在数以千计细小的层面上协调好。

为了确保雏形能真正处理软件的所有功能。这里有一份包含各功能层面部分列表：

◆所有对话框包含标准对话框，如档案开启、档案储

存、打印等等。

- 所有图形输入画面。
- 所有图形与文字输出。
- 与操作系统的互动，包括输出/入资料到剪贴簿中
- 与其他厂商开发产品的互动，像是将资料从其他程序中读取和输出，提供能够嵌入到其他程序中的组件等等。

当雏形定案时，对开发团队来说，大概还不能决定那样的雏形在技术上是否可以具有示范的功能。这应该是雏形接口很自然的一种特性，因为将抽象而不存在的功能在雏形接口中展示出来是很简单的事情（例如，你可以很容易依据使用者提供的文字叙述所描绘出的真实图形的系统做成一个会动的雏形软件，可是事实上要写出这样的软件在以目前的技术是完全不可能办到的）。

应该先整理出一份可能无法成形甚至有危害功能的清单，向一般使用者解释清楚并签名同意后，纳入变动管制系统中，这原就该是“需求规格”的一部分。很少有项目能幸免于难，所以在项目中务必要事先决定哪些功能其实是无法成形甚至有害的。

记住，那只是个随时可丢弃的雏形

虽然雏形接口的发展涵盖了整个软件层面，重要的是由此能展示软件的各项功能，并得到事半功倍之效。记住，雏形接口只是个细枝末节，虽然有用但却不成气候。

一个良好的雏形接口是迅速拼凑出来的程序成果，就好象好莱坞的电影场景中将一块大木板的表面漆成房子的样子，可是实际上，木板后头并没有房子存在。

你不应该把使用者接口雏形当作软件的真正基础，就好比你不能将好莱坞电影场景中的房子布景当成真的房子一样。

在依照这种方式开发一套雏形接口时，无论如何都不要把它当成可发行软件的程序基础。一套雏形接口是以用完可丢的前提开发出来的。在可望推出软件中使用那样的程序代码，就好比企图将好莱坞电影场景中的屋子布景盖出真正的房子来一样。

开发雏形接口，要选择一个合适的软件程序，避免让雏形程序代码出现在最后。举例来说，项目团队预计要将

软件以C++或Java实作，只要将雏形以Visual Basic开发，就可以避免雏形接口成为真正软件的基础。

将完全扩展了的雏形当成定案规格

当雏形接口开始被列入考虑时，整个开发工作就该进入调整步调的阶段了。雏形接口在这时必须够稳定，而且团队必须融入软件之中。让项目的决策者在雏形规格上签名同意，并将之纳入变动管制中。组织将会依据这份规格来估计、规划、安排人事、处理构架、设计制作和开发。决策者必须承担起规格的制定与系统化管制所造成的变动。

系统化管制所造成的变动不代表“冻结”需求。相对的，项目团队会召开会议来处理如何应对。最不明智的做法就是毫无策略、不经规划地以一不作、二不休的方式处理所有更动。

一旦雏形定案了，好处也随即产生。除了曾提到的需求稳定之外，一份完整的使用者接口雏形，能够使构架、设计与实作流程、搭配测试规划开发工作同时进行。不然这些活动通常会卡在关键时刻，除非实作过程进行良好，不然这些工作无法开始。

依据雏形详细写出一般使用文件

在将雏形纳入变动管制前，就可以开始制作详细的使用文件了（使用说明/规格）。这是最后交给软件使用者的文件，一般都是在项目末尾才进行，不过本书的做法是在项目初期就开始着手制作。

有些人也许会反对建立一个完整的使用文件，因为比起单纯的技术规格而言，要花费太多的更新时间。这种说法没错，如果只在乎需求规格，建立完整的使用文件确实花时间，但是别忘了一分耕耘，一分收获。

及早开发的使用文件减去了建立单独的技术规格的过程。制作出来的文件对使用者而言比传统的技术规格要容易了解，并能根据使用者对软件反馈的意见改进质量。使用文件的及早开发，自使用者接口雏形展示的功能开始，逐步接近与真正软件的距离，消除其中的差异性。

这种做法能及早取得一般使用者的拥护与认同，技术文件写作者往往比使用者本身更擅于表达观点。

这类的规格较易更新。许多软件规格变成“中看不中用”的文件，它们是在需求开发时期为了满足项目规划而建立的，可是在需求更改后却没有跟着更新，甚至在写好后就乏人问津。相反当开发团队了解规格最后得让每一位使用者

都能接受时，赶紧修正规格文件就成了刻不容缓的事！

这种做法也解决了需求规格应该涵盖“何种内容”或“何种功能”的问题。如果文件写作者认为使用者需要什么，使用说明/规格中就会有那些东西。除此之外都留给开发人员去自行决定。

许多产品都有多种的文件，包括使用说明、入门文件、参考手册与线上辅助说明。这里头唯一需要在本阶段建立的就是说明软件各项功能的文件。它被当成线上辅助说明的一部分，而且是对开发团队最有用处的参考文件。

区分一套非使用者接口需求文件

本章描述的做法对于着重使用者观感的中小型项目最有效果。大部分项目在使用者接口雏形或使用说明中，没办法提供设计与实作准则来具体描述功能。至于细节算法的规格与其他软硬件的互动关系、效能要求、记忆体使用方式，与其他不常用的需求都可以分别记录在另一份需求文件中（或当作使用文件/规格的附录）。在开发出来后，那份文件（或附录）需要经过检查、定案，并纳入变动管制中。

关于雏形风格说明的范例，可以参考本书网站上的例子。

求生检查

- 项目团队能找出一组可胜任使软件成形的使用者。
- 开发人员建立好几个版本的基本雏形，直到使用者满意为止。
  - 雏形工作在使用者表现还很冷淡时就停下来。
- 开发人员将使用者接口雏形扩展得符合使用者的详细需求。
  - 雏形提供的功能不够广泛、也不够通俗，而且在最后重新修正深层功能时浪费时间。
- 项目团队完成使用说明 /规格，并将此当作细节需求规格。
- 完整开发的雏形定案了,并纳入变动管制中。
  - 项目团队试着将雏形程序代码在真正的软件中使用。
- 区分一份非使用者接口需求文件，而后纳入变动管制中。

第 9 章

质量保证

软件品管(质量管理)的重要性绝不亚于软件测试，在一个高效的项目中应该包含测试、技术检查和项目规划，这些都是为了及早发现缺陷并修正它。

人们在谈到质量时，往往包含的不只一样东西。质量可以用来表示系统不会当机，可以表示软件是否达到使用者的期望，或者程序合用不合用的程度。质量可以表示对特定需求的吻合度，或是在开头对正确需求规格的实作程度。一个良好的定义是“软件可以满足在设计中明白交代出来的要求和没明确指出的需求”。本章将解释如何保证这类型的质量。

质量为何重要

即使你的软件不必多么可靠，缺陷还是得受控制，因为这影响到开发速度、开发成本与其他项目特质。第 3章“求生概念”描述项目的上下游效应：在下游找出错误并修正要比在上游高出50~200倍的代价。知道为何该重视质量了吧，不过伴随而来的还有其他冲击。

人们有时会认为他们可以在目前的项目质量上取巧，到下个项目再来更正。在小型项目中（时间一两个月或更短的项目），这么做当然骗得过去。可是在时间较长的项目中，往往弄巧成拙。你不可能把所有不良效果通通延后到下个项目上；许多效果会直接影响到目前的项目。

软件质量在推动事务的成本上也有着冲击。低质量的软件增加一般使用者支持负担。顶尖公司，如微软注意到这一点，所以向一般使用者的求助电话收费，以支付造成问题的软件带来的支持负担。

开发低质量软件，根基一旦受到侵蚀自然也会增加维护成本。你也许认为你的程序只会被使用3~5年，可是一般程序实际上可以很长寿，需要前后十任的维护程序员来支持。由于程序50%~80%的生命期成本会在程序的初次发行之后出现，就经济角度来看，在第1版的软件中就打下成功的基础总是比失败好多了。

最后，你的工作中是否会出现低质量软件，完全掌握在你手中。我的强烈印象是顾客不会去管这类高低质量的软件是何时到他手中的，只会记得他们喜不喜欢用那个软件。

快速地解决问题固然很好，但有些前辈提醒我们，当快速的结果是被遗忘时，敷衍了事的代价影响深远。

品管规划

本书的一个主旨就是想要让一个软件项目生存下去就得有求生技巧。其中包含了品管要求，它得做到下例几点：

- 软件品管活动必须经过规划。
- 软件品管活动的规划必须明文订定。
- 软件品管活动必须在软件需求活动时期或更早就开始进行。
- 进行软件品管工作的独立小组必须存在。依据项目规模，这“小组”也许只有一个人，也许有两名开发人员，交互替对方的项目进行品管工作。
- 小组成员必须经过软件品管活动的训练。你不能光是指着一名没什么经验的程序员说，“你过来，你现在就去品管部门工作”。
- 必须有适当的经费提供给软件品管活动。

品管规划的要素

一个有效率的品管规划也许包含缺陷追踪、单元测试、源代码追踪、技术检查、整合测试和系统测试。

缺陷追踪在这些品管活动中进行。项目团队保留一份

详列每项已知缺陷及其源头、发现时间、解决时间、解决方式（修好了或还没修理）等等的追踪记录。缺陷追踪是项目追踪与控制关键的一环。完整的缺陷信息有助于决定项目上市的日期、质量以及开发程序的改进。

单元测试是由撰写程序的人来进行非正式的源代码测试。“单元”可以是一个副例程，一个模块，一个对象类别，甚至更大型的程序设计项目。单元测试通常非正式地进行，但却是在单元与源代码正本整合、交付独立的检查测试过程以前必须进行的工作。

源代码追踪是在一个交谈式除错器中一行一行地追踪执行源代码。这项工作是由写程序的开发人员进行的。许多程序员发现这项活动在找寻程序写作缺陷上有着无比重大的意义，我个人的经验也是如此。有趣的证据显示：开发人员在整合程序以前，若能够先在除错器中追踪执行程序状况的项目，比起不使用除错器的项目，容易整合得多。

技术检查是由开发人员来检查别人完成的工作。这些检查被用来确认使用者接口雏形、需求规格、构架、设计与其他技术性工作成果的质量。新的源代码与源代码更动都应该被讨论到。技术检查一般由开发团队带领。品管人

员在检查过程中的角色，是确保检查过程中出现的缺陷，必须密切追踪并完成修改。整合测试是在新开发的程序代码与其他已经完成整合的程序代码合并时进行的。这种测试工作是由新程序代码的开发人员非正式地进行的。

系统测试是指执行软件以找出缺陷。系统测试由一个独立测试机构或品管小组进行。

将这些品管活动合并在一起进行看来似乎负担较少，实际上刚好相反。多层次品管活动的要点在于尽可能及早找出缺陷，降低修正成本。

缺陷追踪

缺陷追踪是记录和追踪有关缺陷从发现到解决过程的工作。缺陷从个别层次——一个个的缺陷开始到统计层次，以开放缺陷数、缺陷解决率、修正缺陷平均需要时间等等数据来进行追踪。

及早在项目中开始追踪缺陷，可及早了解消除错误的重要性，并在项目进行中提供软件缺陷数量的最精确信息。

缺陷追踪应该在项目中及早开始，至少在需求规格定案、工作成果纳入变动管制后。当程序员在实作时期发现

一项设计有缺陷和失误，那缺陷和失误就开始受到追踪了，因为设计方式早就定案了。如果同一位程序员在还没定案的新程序中发现程序写作错误，那样的错误就不用被追踪。但是如果程序员在程序经过检查定案后才发现同样的缺失，那样的错误就该被追踪。

缺陷报告应该纳入变动管制中。让所有缺陷和失误都公开成为软件质量的宝贵资料，让项目团队能将软件中剩余的缺陷和失误具体化。缺陷和失误数量的变动可用来追踪测试活动的进度。

缺陷追踪活动听来也许相当费力，事实上有许多特定化的工具可以让这些工作轻轻松松地完成。

表9-1列出了每项缺陷应该被追踪的信息。

表9-1　缺陷报告中追踪的信息

缺陷代号（一个数字或其他唯一的标识符）
缺陷说明
制造缺陷的步骤
平台信息（微处理器类型、内存量、磁盘空间、显示卡等等）
缺陷的目前状态(还没修好或修正好了)
发现缺陷者
发现缺陷日期

(续)

严重性（1~4表示，或者以致命、严重、表面等等的字眼表示）
缺陷产生阶段（需求、构架、设计、构建、缺陷测试项目等等）
发现缺陷阶段（需求、构架、设计、构建等等）
缺陷更正日期
缺陷更正者
更正缺陷所需代价（人员、小时）
修正的工作成果或产品（需求叙述、设计流程、程序模块、使用说明、测试项目等等）
解决方式（延后工程修正、延后工程检查、延后品管确认、修正、判定不当、无法重现等等）
其他说明

收集这些缺陷信息，让项目团队能够追踪与评估项目状态，以及做成对未来项目有用的图表与报告（我们在第16章“软件发行”会有更多说明）。

技术检查

由于可被利用在上游工作成果中找出缺陷并加以修正的功用，就控制成本与时间表而言技术检查至少与测试工作一样重要。“技术检查”一般包括浏览、调查与程序阅

读等几种检查。

除了兼具有品管的功能外，这些检查也是新手与资深开发人员互相交流经验的机会。资深开发人员能够修正新开发人员程序中的问题，而新开发人员能够从阅读资深开发人员的程序中获得更多经验，也有机会让新开发人员向老的设计思想提出挑战。

一般检查方式

检查依循下列方式。

通知与传递　工作成果的作者通知检查人员（如项目规划、需求、使用者接口雏形、设计、程序代码或测试项目）已经可以准备检查了。在正式设定中，作者将给予检查会议主持人一些说明，让主持人决定检查工作成果和出席检查会议的人选。

准备　检查人员检查工作成果，最好使用过去最常见的错误检查清单来辅助进行。检查会议应该在检查人员完成工作成果的检查后召开。

检查会议　作者、主持人（如果有的话）跟检查人员聚会检视工作成果。

检查报告　在会议后，作者或主持人应该记录检查会

议的统计结果，说明检查量、缺陷发现数与种类；检查会议进行时间、工作成果通过或没通过检查过程。

后续工作 作者或其他人采取必要的更改，这些更改经过检查后，工作成果才被宣告为正式通过检查。

成功使用检查方式的关键

为了最佳结果，留意下列重点。

1. 及早在项目中开始检查

在需求、构架、设计阶段建立的技术说明应该都被检查过。品管说明，如品管规划与测试项目应该由品管与技术人员的检查。检查工作应该在实作阶段持续进行，所有细节设计与源代码都应该经过检查。对工作成果的管理，包括项目日程与软件开发规划，也应该被检查过。

2. 让技术检查着重在找出缺陷

在技术检查时，焦点应该放在找出问题来。将会议时间用来建立与评估解决方案只会浪费大多数人的时间，这些最好另外处理。

3. 让技术检查维持技术性

如果检查失去焦点，就容易变成技术性的敲锣打鼓

竞赛了。任何类型的权威人物出现在技术检查会议中都会变成焦点，因此管理阶层或顾客都不应该出席检查会议。为管理阶层或顾客的利益进行检查也许相当适切，但是这些都不是技术检查所要讨论的重点，所以应该分开进行讨论。

4. 记录下检查过的项目有哪些

追踪设计与程序的检查进度也可变成另一项项目衡量方式。通过追踪每周检查的模块数，你可以了解有多少模块等待检查。如果项目检查的模块数比每周检查的平均模块数还多出许多，以便赶上期限，那些检查过程可能就会被匆匆略过；测试过程可能会在那些模块中找到超出平均数量的缺陷。

5. 记录检查过程中发现的缺陷

经过检查的工作成果会有一份检查报告。列出找到的缺陷，并提供一份修正和查验修正结果的时间表。

6. 在检查进行中查验工作成果

技术检查的一个常见弱点，就是项目没依照检查过程中发现的缺陷进行善后。因此在技术检查中找到的缺陷应该一样纳入缺陷追踪系统中。直到如其他缺陷一样修好为止。

7. 对项目团队公布检查结果

虽然检查会议应该只包含技术人员，但是检查结果应该公布出来（在项目团队中，非真的对外公布），好让其他项目成员也能利用这些结果，避免再犯。

8. 在时程安排中加入检查和修正检查所需的时间

如果项目主持人预期检查会议是在开发人员的工作时间之外进行，检查会议就不能有效运作。在成功的项目中，检查过程是开发人员正常工作的一部分，应该在时间表中安排时间。

如果检查过程有效率，就会在规划、设计、测试项目或源代码中详细检查出问题来，时间表中应该列出修正这些问题所需的时间。

系统测试

检查活动对于评估软件上游品管有着关键意义，系统测试则对评估软件下游品管有着关键意义。下列是成功进行软件系统测试的关键。

系统测试应该由独立测试人员带领

为求效率，软件须由开发人员以外的人来进行测试。

开发人员也许可以在自己的程序中找出相当数量的缺陷，可是要彻底找出缺陷，就必须将心态从让软件可以运作改成让软件无法运作来进行，而很少有开发人员能够在同个项目中同时有两种心态的。

测试规划应该在需求已知时尽快开始

有效率的系统测试取决于有效率的规划。测试项目必须如源代码般被设计、检查和实作。如果你不要让测试变成软件发行的关键障碍，就应该让测试规划尽早开始——最好在需求了解后就开始。

在第一阶段开始系统测试

在阶段性完成方式中，可执行软件会在第一阶段中出现，而系统测试那时就应该开始进行了。

系统测试应该以需求追踪表格准确涵盖完整的需求范围

软件系统测试应该规划成能够涵盖软件全部功能。这一般是经由“需求追踪”的程序加以确定的。在需求追踪程序中，会建立一个大表格，表格各行标上测试项目，各列则标上需求项目。在行列交会处标上“1”表示行的测

试项目有助查验列的需求完成。没标上“ 1 ”的行则表示列的测试项目查验不了任何需求项目，是可以被删除的。一列之中没有“ 1 ”的标记的话，则表示该需求项目没有任何测试可以查验得了。各行各列都应该至少有一个“ 1 ”才行。建立需求追踪表格是一项单调的工作，却是确保软件的整个范围都被测试过的最佳方式。

提供系统测试适当的资源

适当测试计算机软件所需的资源，会依据软件种类而异。对高质量商用软件而言，经验法则是每个开发人员分配一名测试者，这是微软公司与其他顶尖软件公司的比率。这个数量的测试人员是必须的，因为大部分测试套件必须是自动化的，使品管人员能够在软件更改时经常将软件功能从头到尾测试过，这是每天都要进行的工作。有些重大任务的业务系统也需要经过那么多的测试。对于性命悠关的软件，需要更多测试人员。航天飞机的飞行控制软件平均每一名开发人员需配十名测试员。

内部业务系统就不需要那么超级可靠的质量，而只需要少许的测试参与者，大概每三到四名开发人员搭配一名测试者就行了。测试项目不用完全自动化，软件也

不必非常可靠，需求的资源就比较少了。

打破测试上瘾的循环

测试本身对软件质量的增进——如同你步行减肥一般没多大帮助。

测试意味着发现软件系统的质量水准，而非软件质量的担保。

当测试与缺陷修正合并时，测试与修正的组合对于软件质量的担保才构成意义，可是并不是非常有效。更有效的方式是将上游的品管活动，如使用者界面雏形化和技术检查，与下游的测试活动相结合，这种方式更经济、有效。缺乏上游品管活动导致许多机构染上了致命的“测试上瘾”的轮回中。他们的软件质量如此不良，因为在上游阶段没做好品管工作，让大量的缺陷遗漏到下游阶段才被发现。由于缺陷数量如此庞大，这样低质量的项目需要许多测试人员，因而更难分配品管资源给

下个项目。当愈来愈多资源透入下游测试工作，而愈来愈少资源分到上游预防活动时，下游就出现愈来愈多问题，从而形成了恶性循环，同时也让下个项目中需要更多下游测试工作，因而造成测试上瘾的结果。惟一打破测试上瘾循环的方法就是咬紧牙关，分配给项目应有的上游品管资源。一旦这种做法的获益开始浮现，下个项目将可正常取得上游资源了。

Beta版公开测试

在本书建议的做法中，一般品管问题是由内部独立的测试机构进行的。公司向外部的 Beta版测试者发行软件有不同原因，而这些原因可以总结成表 9-2中各项目。有些列出的理由是技术性的，大部分则是非技术性的。重要的是了解这些进行 Beta版公开测试的技术性理由，除了兼容性测试这个例外以外，都可以透过 Beta版公开测试以外的过程得到更好的满足。

表9-2　进行Beta版公开测试的理由

1. 专业咨询。机构对专业使用者展示软件，看看这些专家对于软件该怎么改进有何建议，以及最感兴趣的是什么部分

(续)

2. 杂志评鉴。有些机构将软件在发行给一般大众以前先拿给杂志评鉴者，以讨好那些评鉴者

3. 建立顾客关系。有些机构对重要顾客发行Beta版软件，以便让顾客了解他们享有优惠待遇

4. 证明与掌握控制。机构有时对一般大众发行软件以取得对软件的赞赏意见，用来作为行销宣传。其他机构则搜集顾客意见，好让他们能够列出软件中受欢迎和不受欢迎的部分，而在行销宣传中强调最受欢迎的特点

5. 依据顾客使用方式强化使用者接口设计。有些机构将他们接近完成的软件拿给顾客使用，好让他们能够观察顾客使用软件的方式，据以调整软件内容并克服使用障碍

6. 兼容性测试。有些机构发行Beta版软件给顾客，让他们能够判断出软件在更多与内部测试中不同类型的软硬件平台上执行情况

7. 一般质量管制。有些机构假设愈多人使用软件，就能在发行软件以前找出愈多缺陷，而将软件尽可能多地提供给使用者

资料来源: 节录自 Testing Computer Software, 2d Ed. (Kaner, Falk, Nguyen 1993) and Software Project Dynamics (McCarthy 1995).

在Beta版公开测试过程中的专业咨询能起的作用太

少，也太晚了。如果你需要专业咨询，在处理使用者接口雏形的需求开发时期就该得到那些专家的意见了。强化使用者接口的设计也是同样的原因，应该在需求时期就处理好，而非到了Beta版公开测试才进行。

一般质量管制曾经是主要的Beta版公开测试理由，但是软件公司发现外部的Beta版公开测试代价并不便宜。当公司首次送出Beta版测试软件时，大部分收到软件的人都不会回反馈任何缺陷，也不会送回任何意见。所以机构开始限制给予Beta版软件者的数量。不过接着他们就发现自己被要求改变软件的意见给淹没了，而仍然没收到任何缺陷报告。在经历了那些过程后，机构开始明白Beta版公开测试带来太多低质量的意见反馈，而不能满足有用的品管目的，虽然这对行销多少有点帮助。

如果你要从真正的一般使用者得到反馈，不如聘些使用者代表参与项目开发，并使他们在监督下使用软件，而非让他们参与Beta版公开测试。录下他们使用软件作业过程的录像带，好向开发团队重现他们遇到的所有问题。受监督的使用者将产生非常深入的反馈意见。总之以管理性与价值衡量，我不赞成为了一般品管目的而进行外部Beta

版公开测试。

一次有效率的Beta版公开测试活动需要进行大量协调，而且通常分散了投注在机构内部可以带来更多品管获益的资源。

如果你的软件最后会散播给上千名使用者，你大概会希望在最后发行以前先进行一些外部的测试发行。这些外部发行，并不是为了一般品管的目的，而是为了特定的兼容性测试。即使最富有的机构也不可能完全测试现代办公桌上各种不可思议的软硬件搭配方式。一旦软件通过系统测试，惟一进行兼容性测试的务实做法，就是把软件发行给一组宽宏大量的外部使用者。

品管规划涵盖的工作成果

品管规划应该指出需要检查或测试的工作成果。表9-3中描述了用在本书各项工作成果上的品管方式。

表9-3 本书工作成果的建议品管方式与责任

工作成果	品管人员	开发人员	文件写作人员	管理阶层	顾客或行销人员	一般使用者
变动管制规划	●	●	●	●	●	○
变动提案	●	●	●	●	●	○
前景叙述	●	●	●	●	●	○
十大风险清单	●	●	●	●	○	
软件开发规划，包含预估结果	●	●	●	●	○	
使用者界面雏形	●□	●	●	○	●	●
使用者界面风格说明	●	●	●	○	○	●
使用说明/需求规格	●■	●	●	○	●	○
品管规划	●	○		○	○	

(续)

工作成果	品管人员	开发人员	文件写作人员	管理阶层	顾客或行销人员	一般使用者
软件构架	○	●		○	○	
软件整合程序	○	●	○	○		
阶段性完成规划	●	●	●	●	●	○
各阶段规划，包含小型完成时间表	●	●	●	○	○	
程序写作标准		●		○		
软件测试项目	●	○		○		
可执行软件	■	■	■	■	■	■
(新开发)源代码	○■	●■		○□		
(更改过)源代码	○■	●■		○□		

(续)

工作成果	品管人员	开发人员	文件写作人员	管理阶层	顾客或行销人员	一般使用者
媒体，包含图形、声音、影像等等	○■	●■	○	○	●	○□
软件建立指示 (建立描述档案)		●				
细节设计文件	○	●		○		
各阶段软件构建规划安装程序	○	●	○	○		
	○■	●■	○	○	○	■
使用文件 (简易使用手册)	○■	●■	●	●	●	●■
发行检查清单	●	●	●	●	○	
发行认可	●	●	●	●	○	

(续)

工作成果	品管人员	开发人员	文件写作人员	管理阶层	顾客或行销人员	一般使用者
软件项目记录	○	●	○	○		
软件项目历程文件	○	●	○	○	○	

注：一定要检查 (●) 或测试 (■)，可以检查(○) 或测试 (□)。

如表中所示，纳入变动管制的每一项工作成果都被检查过，有些还经过测试。一些工作成果，如使用说明，是以逐字键入使用说明中描述事项的方式来查验软件是否如说明中所描述的运作。

具体义务在不同项目中可能有相当差异，特别是依据特定文件写作者、主管、顾客、市场人士和参与项目的一般使用者的技能与兴趣而有不同。

支持活动

除了明确的品管工作，品管小组也参与项目的软件开

发规划、各种标准与程序的准备与检查。品管小组会检查软件开发活动来核对检查结果、单元测试与源代码追踪这些工作进行情形。

软件品管小组定期将活动结果报告给开发人员与管理阶层，并定期与资深管理阶层检查活动成果。

软件发行标准

在许多我所知的机构里，软件开发组织多半独断决定何时发行软件，这就好象把狐狸丢入鸡笼子中一样。开发人员与开发主管急于赶上时程目标，而且相信他们的软件有着优异质量。事实上应该要有一组检查项目与衡量标准来控制这种倾向和想法，而品管小组能够提供这些。

因此品管规划必须以具体说明软件的发行标准。发行标准可以像是“没有可重现的软件当机错误”或“平均每八个钟头出现一项缺失”，或者“更正了95%的已知错误”，“没有没修正的第一和第二级严重性的错误”。发行标准必须是可量化的，好让品管小组能够报告软件何时准备发行了，而不会引起敏感的政治争议。

在本书网站上有一份这些工具的列表。

求生检查

- 项目有一套明文得到认可的品管规划。
  - 项目不依循明文规划执行。
- 品管与需求工作同时开始。
- 缺陷追踪软件在需求开发时期就开始运作，而缺陷从项目开始就进行追踪。
- 开发人员在设计和程序代码被当成“完成”以前检查所有设计方式和程序代码。
  - 所有的设计方式或程序代码都能通过检查，说明这些检查都敷衍带过。
  - 开发人员在把自己的程序代码提交检查以前，不进行追踪和单元测试，增加了检查过程中必须追查的缺陷量。
- 品管规划要求独立的品管小组。
  - 没有经费成立独立的品管小组。
- 品管规划包含可测量的标准，用以决定软件是否准备发行。

第 10 章

构架

软件构架可说是组成项目的技术结构。良好的构架可以简化项目，不良的构架让项目如同海市蜃楼。良好的软件构架文件，以现行的构架加以变化、重复利用来自其他系统或买来的组件，在符合标准功能领域的做法下，说明整体程序组织。将每一项系统需求条列详细列出，来降低潜在的下游成本。

构架设计阶段好比兴建房子一般，软件构架阶段可说是参考模型，提供构架团队探索构建软件的不同方法，而不必要花时间和经费去实地尝试不同做法，减少负担。构架时期也被称作“系统构架时期”、“设计时期”、“高阶设计时期”和“上层设计时期”。一般说来，构架描述会放在“软件构架”文件中。

在构架阶段，构架团队将系统分割成主要的子系统，具体规划子系统彼此间的互动关系和上层技术规划的文件。它也针对系统执行中主要的设计问题，像错误处理、内存管理及字符串储存的做法。构架阶段通过定义细节设计时期所使用的结构，来替细节设计阶段铺路。

在小型项目中，构架和设计也许在同一个进度进行，不过在大部分项目中，构架应该被独立看待。 The Mythical Man-Month的作者Fred Brooks报告指出，“拥有一名系统构架设计者，是朝向概念整合最重要的一步……在一个软件工程实验室授课超过二十次后，我开始坚持只要有四个人的学生团队即应该选出一名主管和一名构架设计者。在这么小的团队中定义不同角色也许有点极端，不过我发觉这种方式可以最有效运作，而且对于小型团队的成功最有贡献”。

本章的讨论，假设软件构架是由一小队设计者开发出来的，不管这些人如何进行，本章描述的问题应该被小心提防，并在进行细节设计和构建过程以前解决掉。

缓缓进入构架阶段

构架工作应该在需求开发完成约80%时开始进行，不可能等到需求开发工作全都完成。只要需求达到80%之后，项目已经强韧得足以支撑软件构架的建立。80%得自经验法则，而且项目主持人需要以具体的判断来评估此时是否足以开始发展软件构架。

在构架团队开始进行以前，项目团队、上层主管与顾客应该举行第4章中描述的规划检查会议。同时等待经费的支持，万事齐备后才开始全心投入。

良好的构架的特色

当构架团队投入构架开发工作时，他们将面对一组核心设计问题，这当然是项目的构架中所不能欠缺的。

系统概述

一个系统构架需要先有概括描述。开发人员才能从上

千个细节甚至一打以上的模块或对象类别中建立出一致的轮廓。构架应该深入考虑主要的设计方式，列出这些方式被考虑的理由，以及不被采用的理由。

概念整合

构架的目标应该要能清楚说明系统概念。对以非妥协性为主要目标的系统与以调适性为主要目标的设计而言，虽然两者功能一致，其实是大不相同。一个良好的构架应该能适合问题的需要。在构架设计开始一段时间之后，设计者应该建立一套良好的构架来处理问题，好让其他人感觉到：“没错，就是这样；除了这么做，你还能怎么做”？Harlan Mills 将质量当成“深度简化”。要知道，愈复杂的构架其实就是愈糟糕的构架，别期待它对你有帮助。

小心琐碎的构架—也就是试图能处理所有想像得到的问题的一种构架。现行的趋势是，构架团队会找机会尽量简化，最佳的构架文件应该简短，着重流程，一般少于100页。

最受欢迎的软件工程书《The Mythical Man-Month》的主要内容是大型系统的根本问题在于概念整合。当你检视构架时，你应该为解决方案的自然、简单感到骄傲。构

架不应该看来杂乱不堪。

子系统与组织

构架应该先定义程序中的主要子系统。子系统是功能主要的划分方式，如输出格式化、资料储存、分析、使用者输入等等。大部分系统应该包含5~9个子系统。如果太复杂，系统就难以被了解。图10-1说明在应用程序中这类设计工作的适当繁杂程度。大部分构架拥有5~9个上层子系统。良好的构架在子系统间有着相对少量的互动关系。

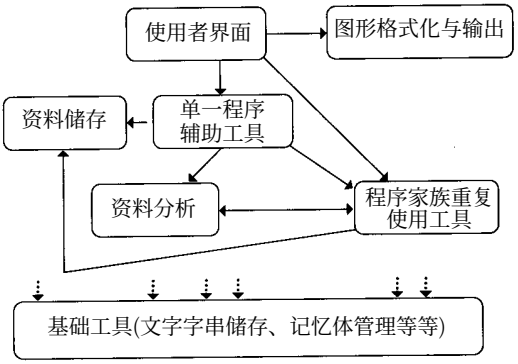


图10-1　子系统构架流程的例子

除了如图10-1中的流程，构架应该描述各个子系统的任务，并提供每个子系统中各模块或对象类别的初步列表。

最后的模块或对象类别清单会在细节设计与构建过程中产生出来。

构架应该描述不同子系统间相互的沟通方式。在图10-1中，只有少许子系统间沟通可以进行。图10-2说明毫无限制的子系统彼此沟通会造成构架变得难以控制。子系统间可能会以各种组合进行互动，这样一来将破坏简化的目标。一个良好的构架应该将子系统间的沟通降到最低。

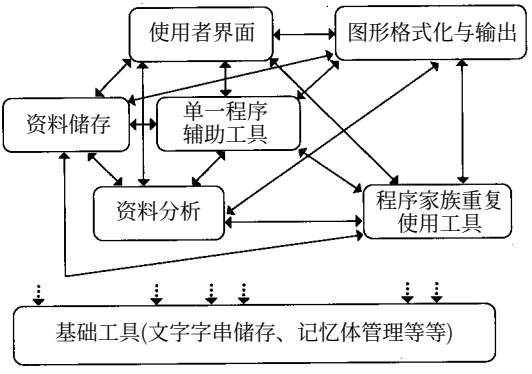


图10-2 毫无规范限制子系统间沟通的构架例子

子系统沟通的限制，一般不会由标准软件开发工具自动设限。所以在细节设计和程序检查的检查清单上应该加列一行“遵从构架指示”的项目。从没在良好开发构架项目中工作过的开发人员，有时会抗拒构架放进自己程序中

的限制。

一旦你明白构架的目标是降低复杂度，构架设计者的工作就一目了然，大体上就是决定软件中该放入及去除哪些东西。

构架标记方式

在大型项目中，应该采取一套标准的标记方式，像是Booch/Rumbaugh Unified Modeling Language （UML，统一模型化语言）标记方式。至于小型项目，只要每个人都确实了解流程意义，公开构架流程就行了。

在项目后面的细节设计时期，开发人员会用与构架阶段相同的标记方式或是适于深入细节设计工作的方式来注记。如果在构架阶段采用 UML，他们多半会想继续下去。开发人员可能也会以伪代码来表达个别例程的设计方式。不管使用哪一种标记方式，都应该在构架时期协调好一套方法，好让整个项目都能采用那套标记方法。

环境变化与变更策略

成功构架的一个重要特色，在于标明最可能变更的程序领域。有些项目中最致命的影响是没料到后期在设计完成后才发生的变动。即使一切进度都在掌握中，处理起来还是相当棘手。

构架应该列出程序中最可能变动的部分，说明构架如何应变。最常见的因素是需求不稳，所以构架中至少应该包含一串稳固的需求项目。

另一个常见的问题在于支持技术的变动——开发人员发现公司给他们的物件类别链接库不合用。项目确实不应该只依赖任何单一来源的技术上（如编译器厂商，硬件平台等等）。当无法避免时，构架应该事先做好防范，好让开发人员能够有修改依据。

在考虑环境变化时，考虑八成可能发生情形，然后停下来冷静一下。构架团队不可能预设每一种变化，当考虑环境变化达到八成以上的程度时，就差不多足够了。

重复使用分析与外购 vs.自建决策

缩减软件时程跟降低成本的最有利方式之一，未必要自行构建软件，可以购买或重复使用现有软件。这样一来

会大大影响构架设计的方式。也就是说可以采用外购组件或重复使用内部源代码，甚至连细节设计、测试项目、规划方式、文件及构架组件都可纳入考虑范围。

标准功能领域的构架途径

除了系统组织的问题，构架应该着重在对细节实作有全面性影响的设计决策上。构架应该深入这些决策领域。下面是这些领域中最常见的项目。

- **外部软件接口**　软件是否可与其他软件沟通？采用哪一种方式？程序接口间传递的数据结构为何？
- **使用者接口**　使用者接口如何与系统的其他部分分开，好让使用者接口上的修改不致影响系统的其他部分？
- **数据库组织**　数据库的组织和内容为何？
- **资料储存**　非数据库资料如何存放？使用的档案格式如何？主要的资料结构有哪些？
- **关键算法**　软件的关键算法为何？这些算法是否良好，或者在软件项目任务中如何定义这些新的算法？
- **内存管理**　不同程序部分的内存配置策略为何？

- **字符串储存** 文字字符串如有错误信息，如何储存与取用？
- **并行处理/执行绪** 这是个多执行绪的软件吗？如果是，它的并行性、可移植性与相关问题如何处理？
- **安全性** 软件需要在安全的环境中操作吗？如果是，对软件的设计有何影响？
- **语言区域化** 软件会在其他国家使用吗？如何处理不同的文字字符串、不同字符集和可能的页面书写方式（从由左至右变成由右至左书写）？
- **网络** 软件如何支持网络上的多人操作？
- **可移植性** 软件如何在一个以上的环境中执行（例如在Unix和Windows NT上）？
- **程序设计语言** 构架上允许软件以任何程序设计语言开发、或是要求以特定语言实作？
- **错误处理** 构架中包含同一的错误处理策略吗？

需求追踪性

构架团队应该建立一个如第9章“质量保证”中测试项目的“追踪性表格”。

这种需求追踪表格的建立通常很烦人，也可能是令人泄气，因为当构架团队完成构架时，他们会找到一些没被子系统涵盖的需求项目。如果是这样，真值得庆贺！不管烦人与否，项目花在找寻和更正构架错误的时间，比起旷日废时的后期动作上，要少50~200倍。

如果你能清楚软件项目难免有坏消息，还是早点知道的好，因为上下游效应迟早会影响到项目成本。

在需求开发时期，项目团队应该建立一份具风险规划功能的领域清单，因为那些领域的功能能不能写出来都是未定的。当构架阶段完成，清单上的项目应该大部分都被解决掉了 —— 构架团队应该拟出实作这些东西的大概流程，或者事先考虑无法完成的可能性，而将这些项目预先拿掉。

支持阶段性完成规划

对于本书内容中的项目最重要的是构架必须支持阶段

性规划，应该能提供阶段性规划中如何开发与完成的方式。具体说来，不应该依赖无法独立运作的子系统构架。将系统各部分的依赖性找出来，安排一套开发计划，以支持阶段性规划的做法。

构架时期通常是你在阶段性完成的项目中碰到一些阻力的时候。基于顾全大局的立场只得做出一些“不完美”的构架决策。重要的问题是“为何不完美”？一个到项目末尾之前都无法完成关键功能的构架，无助于转移风险，也无法追踪所需的明确进度，这实在称不上是完美的构架。一个构架如果依赖高风险的实作策略，也称不上是完美的。构架更不能只依据静态的系统目标来设计，而忽略了动态的开发过程。

如何分辨构架完成与否

要想了解构架团队如何完成构架是项挑战，构架团队可能在这阶段中打转而见不到任何成果。因此要制定目标，并紧密监控过程。

一方面，切勿宣告构架完成了——直到构架不再想更改。你不必真的停止构架的更改，不过构架团队应该对这点有信心。

另一方面构架总有缺陷，时机成熟时，构架团队就应该投入项目剩下的工作。在程序实作深入进行以前，问题不会被赤裸裸地揭露，你也别期望在构架阶段就完全了解问题的全貌。

依循设计Algol程序设计语言的团队获得的建议：追求“最佳”境界只会妨碍“良好”境界的达到。

如果你试着追求最佳境界，你常会一事无成。努力简化所有必备要求，别太担心找不到最佳的解决方案。

软件构架文件

一旦构架完成，就应该描述在软件构架文件上，将文件提交变动管制会议进行检查，视需要修订，然后定案执行。软件构架文件会变成控制未来设计与开发工作的标准，所以应该确切检查构架方式，并保证可行。

当更进一步的工作完成时，构架需要重新修订。修订时，应该透过标准的变动管制程序进行更新。依照那些程

序可以确保构架不被任意修改，也有助于突显构架缺陷，并比较下游细节设计与构建缺陷的真正代价与差距。

求生检查

- 构架团队建立一份软件构架文件。
  - 软件构架文件没有进行变动管制。
  - 软件构架文件没有在设计和构建过程中根据变动来进行更新，且不再精确描述程序的构架。
  - 开发人员找不到项目构架的内容。
- 构架强调简化更甚于精巧。
- 构架支持阶段性完成规划。
- 构架方式针对所有项目需求，并在完成的需求追踪表格中注明。

第 11 章

最后准备

在项目团队将需求规格定案，并开始构架设计时，他们可以较项目初期制定更多细节规划。这时期称为“最后准备时期”。一些稍早制定的规划需要修订，以反映需求开发阶段中改变的软件概念。其他规划则需要等到使用需求规格定案后才接着进行。

本章描述的准备工作在项目需求定案而构架工作进行时完成。这些准备工作包含：

- 建立项目预估。
- 写出阶段性完成规划。
- 进行持续规划工作。

除了这些相关的部分，项目团队需要在细节实作前进行另一类的规划，以便每个阶段开头时顺利进行。

项目预估

在需求规格定案后，项目团队可以进行有意义的工作量、成本与时间预估。

进行软件预估时，请将以下经验法则牢记在心：

- 精确估计软件项目是可行的。
- 精确估计要花的时间。
- 精确估计需要的定量方式，最好有一套软件预估工具协助进行。
- 最精确的预估资料来自于以前完成的项目。
- 预估结果需要随项目进度改善。

预估程序

有效率的机构遵循系统化的预估程序。预估程序应该

包含下列特色。

预估程序确定一套明确程序可以避免野心勃勃的项目主管、上层主管、市场人员向顾客恫吓，并采用不切实际的工作量与工作时数。预估程序的价值就在于项目预估必须照章行事。

一旦所有项目资助者都同意估计程序的做法，你就可以对影响估计的因素（功能和资源）进行合理的交涉，而不用对估计结果（预算和时间）进行无谓的争吵。

预估结果应该由一名专业的估计师或最资深的开发、品管与文件撰写人员建立。准确的估计需要有专业素养。如果你可以找到一名专业估计师，就再好不过了。如果找不到这样的专家，就应该由最熟悉类似工作的人来进行。不管找不找得到这样的专家，项目预估结果应该包含最熟悉项目的评估人的意见。

1. 预估结果应该包含所有正常活动的时间

表11-1列出一些应该包括在项目预估中的明显与不甚明显的活动。

表11-1　应该包含在项目预估结果中的活动项目

明显的活动
构架设计
细节设计
一般规划
各阶段发行规划
程序写作
程序测试
建立使用文件
建立安装程序
建立旧版资料表格
更新转换程序
较不明显的活动
与顾客或一般使用者互动的过程
向上层主管、顾客和一般使用者展示软件或软件的雏形
检查规划、估计、构架、细节设计、阶段规划、程序、测试项目等等

(续)

较不明显的活动
修正检查和测试过程中找到的问题
维护修订管制系统
维护执行每日建立版本的描述指令
评估更改提案影响
答复品管问题
答复文件写作问题
支持过去项目
团队协调
接受技术训练
训练软件支持人员
假日
休假
周末
病假

有些项目采用短视的态度，蓄意将表11-1中的许多活动排除在规划之外。那样对小型项目也许管用，不过在至少数星期的项目中，那些活动将会以另一种方式渐渐出现。但是因为没安排好，逐渐拉大的缝隙出现在规划（预计要

做的事情）跟现实（实际发生的现象）之间，对项目造成致命的危机，项目团队不再认真看待项目目标。到那时，项目团队就丧失有意义的规划、追踪进度与控制项目进展的能力了。

项目规划不应该假设团队的工作会有所超前，如果项目规划假设团队会加班工作，项目的进度就有可能倒退。一个从开头就加班的项目，就好比在冬季爬山而不携带预备的食物和御寒衣物一样。如果够幸运，当然一切都没问题，不过哪个有理性的人会依赖运气来过日子？

要降低时程延误的风险，在项目开头投入更多资源，不要建立一套依赖加班赶工的规划。

2. 预计结果应该由预估软件建立

一套商用软件预估工具可以当成是估计过程中的客观权威。它可提供工作项目明细、各角色的定位、为特定项目类型与规模确定的详细时程安排。

使用预估软件可以避免在成本与时间估计上的争议所带来的负面影响。假使有一名项目资助者反对一项初步预

估，因为估计的成本和时间太高了，行销人员就会调整目标，使项目的成本与时间能大幅降低。

在这样的情形下，预估软件就能扮演公正的第三者。项目假设性的变更可以输入其中，然后软件可以判断出项目成本与时间的变化效果。

3. 预估应该依据以往完成的资料

你所拥有的最佳预估资料，是机构中过去完成的项目资料。好的软件预估程序允许利用过去完成的项目资料来更正估计结果。

预估人员有时会错用过去项目的规划资料。除非确实资料无法取得，不然尽量避免如此。

4. 留心开发人员可能不接受预估结果

在碰到麻烦的项目中，我常发现开发人员认为项目预估一开始就不切实际。但却没机会在预估结果出现在上层主管或顾客之前，先检查估计资料。开发人员不买预估资料的账是项目目标难以达成的最大警讯。最糟情形是，那还代表着开发人员与管理阶层的敌对关系，表示项目除了在预估结果不精确造成的问题之外，还有着严重的动力与士气问题。

5. 项目团队应该在几个特定时期重新评估

我在第3章“求生概念”中提到过，在早期阶段就想要精确估计一个项目是不可行的。在第7章“初步规划”中说过，最有效率的软件组织规划，必须在项目进行中的几个时期分别修订估计结果。以两段式出资的方式规划检查，将重新评估纳入项目规划中。

虽然在本书中，是在构架设计的预估部分进行，不过预估结果应该在各阶段末尾产生出来：

◆初步需求开发（在使用者接口雏形发展好后）。

◆细节需求开发（在使用说明/需求规格完成后）。

◆构架设计。

6. 预估结果应该纳入变动管制

项目团队在各阶段完成估计后，预估结果应该经过检查、签名同意，定案后纳入变动管制中。将文件纳入变动管制的一个优点是，估计结果必须经过所有相关人员检查和认可——无论挑剔的顾客、主管或市场人员都不能单方面的加上一份立意良好但缺乏根据的预估。同样开发团队也不能在半夜偷偷摸摸的把时间表换掉——时间表的更改必须让所有受影响的人都一目了然。

7. 完成目标

项目团队应该利用新建立的估计结果，定下主要完成

点的目标，要牢记在心的是这些目标都会经过修订，更精确反映出项目进展。软件开发规划应该包含下列主要完成点的日期：

- 构架完成。
- 第一阶段完成。
- 第二阶段完成。
- 第三阶段完成（假设只有三个阶段）。
- 软件发行。

除了主要完成目标，在项目进行时，软件开发规划应该随时更新，以涵盖随后进行的阶段完成目标。细节内容将在下一章中提起。

估计中的非技术考虑

软件项目估计有着双面问题。问题的第一面是估计本身就有着技术上的困难。前一节中的描述已概略说明。

问题的另一面则是估计结果会受到来自市场人员、主管、顾客与其他项目资助者的压力，更难产生。

采用市面上最先进的预估软件，将估计结果依据以往表现进行评估，然而只因“时间太久了”而让无知的主管把估计结果拦腰斩半的机构实在很多。

要求在“更短时间内完成相同项目”就好比试着挤压篮球，企图让整个球变小一样。好一点的情况是你可以暂时把球挤压成不同形状；坏一点的情况是你把球里头的气挤了出来，让球无法继续使用。

软件项目也可以暂时被挤压成不同形状，通常是项目前端先受到挤压，将工作往项目后端挤。如项目在起始阶段花的时间较少，则有可能会出现较多的缺陷。以后不仅得更正所有在前头制造的错误，还得花费更高代价去修正问题。与挤压篮球不同，挤压一个软件项目的估计结果只会让整个项目变得更大，不会让它变小。

如果你要确保软件项目的成功，就告诉其他项目资助者，任意更改成本与时间估计，其他不做相应更改的代价将会是什么。如果你是名高层主管，你要求开发团队将时间缩减一半，你得注意项目团队的情况未必跟你假设的一样。当你告诉他们把时间缩减一半时，最好也告诉他们只要完成一半的软件就行了。

阶段性完成规划

在本书建议的规划方式下，软件最后是分阶段完成的，而最重要的功能应最先完成。图11-1提供了阶段性完成的

观念概览。（与图5-2有些相似。）阶段性完成让软件可以在需求和构架设计发展好后分阶段完成，把最重要的功能在最开头的阶段中先做出来。

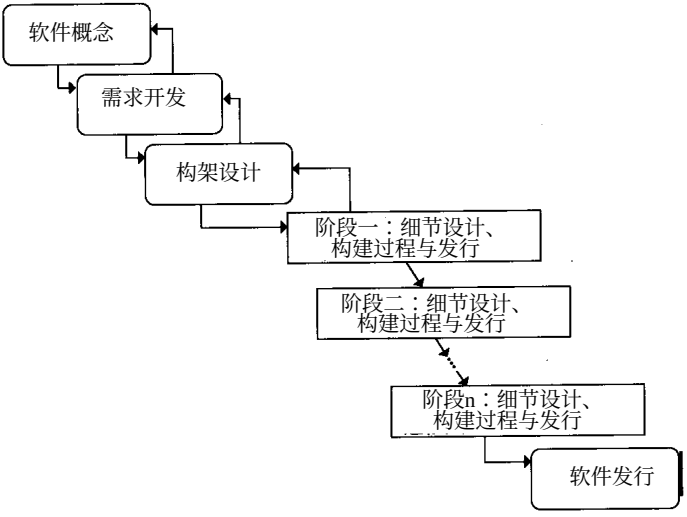


图11-1　阶段性完成的概念

由于最重要的功能最先做好，使用者的需求就愈快满足。阶段性完成不会真正削减完成软件需要的时间量，可是能够削减软件完成似乎需要的时间。每个阶段性完成的成果提供了项目进展的确实证据，如果你曾经碰到过项目时间似乎无限延长的经验，这也可以让你放心多了。

阶段性完成不会自然而成，它需要坚实的构架，细心的管理与详尽的技术规划。那些工作对项目来说都是良好的投资，因为这样几乎去除了项目延后完成、整合失败、功能错误和顾客、主管与项目团队之间的摩擦等常见的项目危机。

将项目分割到各阶段中

在一系列阶段中，最先完成的应该是软件框架。在接下来的工作中，项目团队细心在软件中加入更多功能。最后软件在最终阶段完成。如图11-1所示，在各阶段中，项目团队进行完整的细节设计、建构及测试循环，然后完成各阶段目标。将软件质量提高到可推出成品的水准，避免项目累积的工作量过大以致在你真正要推出软件时造成失控是很重要的。

第一次发行工作的独特之处，在于试着将构架中的错误找出来，并要求打好足够的基础。避免将项目规划成在第一阶段就想将所有构架基础打好 (如果这样就降低了阶段性完成这一做法的风险管理效益)。如果构架设计完善，在完成第一阶段的功能时，所需进行的只有很少量的基础工作。

作为一般目标，应试着将软件的功能依照重要顺序完成。以这种方式确定完成目标，可以强制人们安排好工作顺序，并有助于将非基本功能延后到项目后期，从而去除华而不实的东西。如果你在安排发行顺序上成效良好，先期的完成结果将降低发行稍有延迟时的时间压力，因为使用者已经看到最重要的功能了。

阶段主题

确定各阶段内容，也许会引起耗时的协商逐一功能的问题。一个决定哪些功能放到哪个阶段完成的好办法，是在各阶段确定一个主题。这些阶段主题与项目情景相关，可以说就是项目团队投入各阶段工作后所要完成的目标。

一个开发文字处理器的项目团队，也许会将阶段主题定义成文字编辑、基本格式安排、进阶格式化、周边工具与整合等等项目。这样安排主题太过僵化。安排主题是要易于决定各阶段要完成的功能，即使一个功能落在灰色地带，例如自动列表编号可能就被归类到进阶格式化或周边工具的范围内，你还是可以将它更轻松地归类，只要确定两个主题中哪一个较适合确切就行了。不必每个阶段都考

虑安排主题的问题，后头的两张表格说明依据阶段主题安排的阶段性完成规划大纲。

表11-1　依据阶段主题安排的阶段性完成规划范例

阶段	主题名称	说明
阶段一	文字编辑	完成包括编辑、储存和打印的文字编辑功能
阶段二	基本格式化	完成文字和基本段落格式功能
阶段三	进阶格式化	完成包括所见即所得的页面安排与画面格式化工具的进阶格式化功能
阶段四	周边工具	完成包括拼写检查、辞典、文法检查、断句换行和邮件合并在内的周边工具功能
阶段五	整合	完成与其他软件的完全整合

当你使用阶段主题的方法时，项目团队大概无法依照确切的顺序安排来完成各项功能。所以将阶段主题按照重要性安排，然后按顺序完成阶段主题。

发行工作主题对内部使用的软件项目和公开出售的商

用软件项目一样管用，下面的表格说明内部使用软件项目的阶段性完成规划大纲。

表11-2　顾客帐目系统的阶段性完成规划范例

阶段	主题名称	说明
阶段一	数据库	完成账目数据库，可以在数据库中存取帐目信息
阶段二	帐单	完成打印跟找寻标准帐单与国际帐目资料的功能，可以启用收受付款功能
阶段三	网络	完成网络资料输入功能
阶段四	延伸报告	完成管理摘要报告与分析报告功能
阶段五	自动化	完成无人月结处理功能

阶段主题的使用不应该造成发行规划的省略。文字处理器与帐目系统的例子只是显示了各发行主题的大概内容。开发团队还是需要安排在各发行中要完成的功能规划。如果没有做到这些，你就不会确切了解各阶段完成的是什么，而且失去这种做法的优点了。

类似阶段性完成的规划方式

有些项目依照一种百分比的方式，规划在软件完成80%、完成90%、完全完成时各发行一次。这种方式并没有列出各阶段的详细内容，光是完成的百分比并不足以引导阶段性发行规划。如果一个比例目标有着和主题发行目标同等程度的详细规划，使用这种百分比的做法当然可被接受，虽然这样的做法不能决定将哪些功能放到哪个主题下进行。

相似的，将项目分成Alpha、Beta和最后发行版，并不是真正的阶段性完成方式。不如这样说吧，那种方式是让质量低到在三个阶段内就必须正确运作的软件，在Alpha发行时期完成完整功能的软件规划。这跟真正的阶段性完成方式比起来，是个代价高昂的做法，项目团队必须花费Alpha、Beta和最后发行阶段的大量下游时间，来更正原本可以在上游阶段即可更正的缺陷。

推出各发行版本

项目团队不必依据阶段性完成的做法，完成各种给顾客的发行版本。在本章稍早的文字处理器例子中，项目团

队也许不会在阶段三或四、五以前向顾客发出软件版本，不过团队还是可以利用阶段性完成，作为追踪进度的辅助工具，掌控品管缺陷，降低整合问题的风险。项目团队可以从阶段性发行的做法中取得许多好处，即使他们完成的东西只对品管人员、市场行销人员跟其他项目资助者发行，让这些人了解项目的进展。

项目团队对顾客发行软件的频率，取决于有多少顾客，以及与顾客之间的关系。如果项目只有几名内部顾客，发行程序就不用正式化，而且团队可以每周或更经常地推出已完成的软件部分。如果团队有成千上万的顾客，发行程序就必须正式多了，而且团队不应该规划比两、三个月更频繁的发行时间。更频繁的外部发行工作，将让项目承受外部发行的负担，关闭还不能完全运作的功能，撰写发行说明，处理发行程序，指导使用者安装程序，以及应付求助电话，而对于降低项目的技术或管理风险并无明显益处。

各阶段性发行工作，应该满足最终的软件质量目标。阶段性发行的一个好处就是让软件质量免于不知不觉地长期下滑。将软件先行推出而不将质量提升到必要水准，将有碍软件提升到必要的水准。

修订阶段性完成规划

阶段性完成规划应该提交给变动管制，检查定案后如其他重要的工作成果一样，纳入修订管制系统的保管下。不过这份文件不应该被冻结。在项目后期完成的工作将显露出初期阶段性规划没考虑到的问题，因而必须对规划进行一些修正。届时阶段性完成规划就应该被更新，反映经过变动管制程序修订的规划内容。

持续规划工作

最后准备时期比较着重在规划上，所以从日常的项目活动中回头仔细检查稍早发展的规划方式是重要的事情。

风险管理

在项目初期就开始的明确风险管理工作，应该在项目中持续进行。当项目工作定义愈来愈清楚后，风险也变得愈来愈明确。十大风险清单现在应该已经更新过好几次了，在最后准备时期，找出以前不够明显的新风险，找出与成本、可用计算资源、可用人力与项目不确定的技术方面相关的风险。

本书建议的许多做法获得选用的原因都是由于它们

拥有控制最常见、最关键的项目风险的能力。变动管制规划、软件整合程序、经常调整项目估计与阶段性完成的做法都能降低项目风险，尽管这些都不是主要的风险管理工作。

项目前景

项目前景仍然适切吗？在一些情形下，需求开发阶段中得到的许多信息都显示出前景需求不再符合项目设定。也许项目团队本来同意的前景规划不再有用，可是他们没办法在具体的新前景规划正式出现之前，将工作以同样的新看法重新调整步伐。检查项目规划中的前景叙述，如果必要，把它修订得能够为阶段规划、构架和细节设计与实作阶段提供指导方向。

决策权威

在包含变动管制规划与初步估计等项目计划与目标初步规划明朗时，检查决策的权威性。如果决策权威不在意或不同意项目规划，最好在细节设计和实作开始进行以前，把所有问题都搞定。

人手安排

最后准备时期也是进一步安排人手、检查项目团队是否健全的适切时机。以下是几个评估项目人手安排适当与否的准则。

- ◆项目士气应该高昂。如果没有，找出原因来更正。
- ◆项目不应该有任何麻烦成员。如果有任何项目成员会制造问题，在他们对团队士气造成更多打击、伤害到项目成员生产力、而且还有时间找寻新成员以前，把他们剔除掉。
- ◆团队安排的方式应该是有效的方式。如果不管用，就重新安排。
- ◆如果有某些成员显现出弱点，提供补救训练。
- ◆检查有没有可能充实软件开发规划中要求的项目成员。如果不行，将这一点当成十大风险清单上的一项风险。
- ◆检查项目是否在知人善任上有着强烈的正面评价。项目是否以对机构本身显露出更有价值或更没价值的人力资源安排方式进行着？如果项目将成员数不断在实作开始之前扩充，项目迟早惹上大麻烦。

更新软件开发规划

在初步规划阶段建立的软件开发规划，应该经过修订与更新，反映在最后准备时期的规划成果。修订过的规划应该经过项目主管、开发团队、品管人员与文件写作团队的检查与认可，然后纳入变动管制中。

你可以从本书的网站上下载一份预估程序的样本。

你可以免费从本书的网站上下载作者公司的预估软件Construx Estimate。

站上也有其他商用预估软件的网址链结。

求生检查

- 项目团队在初步需求开发完成后建立第一份估计结果报告。
  - 估计没包含正常活动，如例假日与周末。
  - 开发人员不相信估计结果。
- 估计结果在细节需求开发后更新，并在构架设计阶段后再次更新。
- 项目有套依照主题安排阶段性发行的阶段完成规划。
  - 规划不包含细节。

- 项目前景、风险管理、决策与人员规划符合现状。
- 项目的软件开发规划符合现状，而且继续被遵循着。

Part III

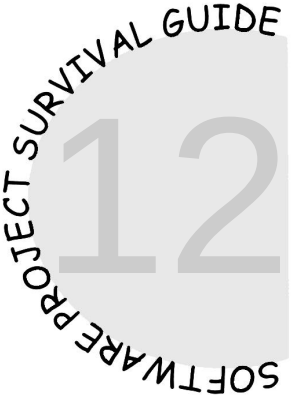
SUCCEEDING BY STAGES

第三篇

逐步迈向成功

第 12 章

阶段的初步规划



阶段初步规划在每个阶段初期描绘出该阶段的详细过程。项目团队通过阶段规划来描述细节设计、程序写作、系统检查测试、整合以及其他该阶段将进行的活动。最耗费功夫的规划工作是建立一连串步骤，让团队得以按部就班完成阶段任务。这个工作需要的功夫虽多，却是值得的，因为它能让你看到进展以及降低风险。

采用阶段性开发的项目，其中每个阶段都可算是小型项目。包括规划、设计、构建、测试与发行准备。软件业人士发现小型项目所承担的风险远小于大型项目，而阶段性完成的做法正是将大型项目的风险转化成较小的风险。

为何要进行阶段规划

我的一名客户公开招标一个软件产品的第 2版。当他提出提案时，一个厂商承诺产品将在五个月内完成。另一个厂商则承诺产品将在九个月内完成，原因是考虑此项目的规模需要较久的时间，另外还考虑了客户需求的优先级，提出会先在三个月内把最优先要完成的东西拿给客户看，他预计在六个月内推出第二阶段成果，最后在九个月内完成整个产品。第二个厂商说，我的客户和顾客最急需的功能需求会在第一阶段内完成，我将建议我的客户把这第一阶段的成果当成一个 1.5版本的产品发行。在评估这些提案后，我的客户认为规模风险并不是个严重的问题，而选择了第一个厂商承诺在五个月内推出产品的提案。

最后所有相关者都后悔未曾像第二个厂商一样关注规模问题。被选定的厂商对开头几个阶段的推出日期偏差了

百分之百到百分之两百。六个月后，这厂商将项目重新估计成九个月内可以完成，九个月到了又重估十二个月内可以完成，等十二个月到了，他们还在为质量问题伤脑筋，再也没人相信这家厂商的估计结果了。最后，十二个月过去了，我的客户没有半个2.0版或甚至1.5版的软件可以推出。

不可靠的预估常出现在软件项目开发中，而阶段性的做法有助于降低估计错误造成的伤害。如果我的客户安排了三段式推出计划，也许不会真如第二个厂商承诺的刚好三个月，可是应该会在三、四个月内，获得一部分的版本。通过阶段性完成规划，我的客户应该已经在三、四个月后对顾客提供最重要的新功能了，而不是花了一年的开发时间还交不出任何东西来。

阶段性发行强迫开发团队“收敛”软件，即把软件完成到一个可发行状态，在项目进行中重复多次这样的工作，使项目更易于管理。这种做法降低了低质量软件、缺乏可见性与时程延误的危险，这些可降低的风险对那些在每个阶段初期没做什么规划的人而言，应该是一大刺激吧。

阶段性规划概要

在每个阶段，项目整体上以软件开发规划中描述的方

式进行，细节则受着阶段初期定下的阶段规划引导。每个阶段规划在定好后，就加进软件开发规划之中。除非项目规模庞大，不然每个阶段规划应该都只有几页纸的长度。各个阶段，时程表与工作指派都应该在以下阶段性规划的工作中安排好：

- 需求更新。
- 细节设计。
- 程序写作。
- 建立测试项目。
- 更新使用文件。
- 技术检查。
- 更正缺陷。
- 技术协调。
- 风险管理。
- 项目进度追踪。
- 整合与发行。
- 阶段末简讯。

接下来本书将对这些行动更具体地加以描述。

需求更新

在项目早期阶段中，实际上的需求应该和项目团队在

雏形化与需求开发过程中设计出来的样子相符。在之后的阶段里，才会对软件有更深的理解，市场状况的改变与其他因素可能要求需求规格必须更改。在每个阶段初期就应该分配好时间(特别是后期阶段的初期)来评估需求更改的可能。

细节设计

在每个阶段初期，开发人员做出足以支持稍后将完成的实际功能所需的细节设计。如果细节设计工作发现构架中的缺陷，项目团队会透过更改转致程序来修定系统构架。

程序写作

开发人员把要完成的软件在这阶段中写出来。程序写作由日常的程序建立与程序测试所支持，因为现在大部分的软件项目中，建立软件细节设计的开发人员通常也是撰写那部分程序的人，所以细节设计的缺陷容易出现在写出来的程序中。

建立测试项目

测试人员应该建立必要的测试项目，来进行功能测试。

测试项目可以和程序构建同时进行，可以采用需求开发阶段中建立的细节使用者接口雏形，加上程序代码开发人员在完成程序以前制作的测试用程序来进行检测（这做法会在第14章“构建过程”中详细讨论）。

更新使用文件

更新使用手册以说明实作出来的软件。建立辅助档案与其他类型使用文件。

技术检查

开发人员参与设计与程序检查。构架后期是认真开始设计与程序检查的时候，这些时间也应该包含在阶段的规划中。

更正缺陷

开发人员对在测试与检查阶段找到的缺陷进行更正。阶段性做法的一个优点就是它能够通过在各阶段末期强迫提升软件质量到可推出的水准来降低产品质量低劣的风险。各阶段中找到的错误必须在同一个阶段内完成修正，而且这些修正必须经过测试或技术检查的确认，以降低质

量低劣的风险。

技术协调

每个项目都需要经过开发人员与测试人员的协调，而且时程规划中应该拨出时间来进行这样的协调工作。大型项目的主管也需要协调不同开发小组之间的行动。开发人员一般也得对技术文件写作者提供实作上的说明，并检阅他们写出来的技术文件。在使用本书中的做法时，项目已经建立好完整的使用者接口雏形与使用手册了，所以项目中需要的协调工作就会比其他项目所需要的少。

一些需求和使用者接口上的改动在进行软件细节开发时几乎是不可避免的，开发人员也得对测试人员和技术文件写作者解释这些更改的地方。

风险管理

阶段规划应该时时考虑风险。项目主管应该检视项目目前十大风险清单，询问目前项目规划中有无妥善解决目前面对的项目风险。许多时候，项目头尾的十大风险会有所不同，所以需要不同的规划来避开风险。每一章节末尾的“求生检查”提供了一些办法，特别是在各阶段末尾可

能碰到的风险警告。

项目进度追踪

追踪已经完成的项目进度是各阶段中的主要管理工作。下一节“阶段里程碑”就是在详细讨论这些工作。

整合与发行

在各阶段结束，开发团队推动软件到一个可发行状态，他们会将要发行出去的程序代码整合起来，修正缺陷，然后将软件提升到符合发行质量的程度（这里的意思是说，他们将软件的安装程序调试得可以运行，内容相关的线上辅助说明会显示正确内容，也就是一个正式发行出来的软件中应该有的工作等等）。

这时软件可能以最适于业务原则的方式“发行”出来：

- 可以简单地宣告软件“已发行”，让开发团队庆祝完成这个阶段的发行工作，然后立刻进行下一阶段的工作。
- 可以对某些内部或外部使用者发行。
- 可以发行给内部使用者，让他们检视并评估目前成

果。

◆可以发行给一整群内部或外部使用者。

阶段性成果在多大范围内发行应该依据业务因素考虑，而非技术因素。软件在各阶段结束后的技术质量应该让软件可以发行给任何人，可是软件本身可能没加入太多足以广泛发行的新功能，或者项目团队不想承担管理外部发行所需的成本和时间。

另一方面，如果外部使用者急于一览新功能，而开发团队也能够依序发行软件，在各发行版本推出时对外部使用者公开发行阶段完成版的软件也许是个较好的决定。

不管对使用者发行与否，软件也应该拿给品管小组测试。除非经过独立验证过，不然贸然宣告软件已经适合发行是没什么好处的。

阶段末简讯

在阶段结束时，项目团队应该停下来检视一下进度，并作出必要的修正。项目应该依据日期检查进度，找出实际已完成的进度，了解更改哪些部分可以使项目进行得更顺畅。项目团队会对他们开发中的软件产生更深的了解，这些应该足以在各阶段末尾建立更精确的成本与时间估计。

阶段里程碑

在各阶段中一项重要管理工作就是积极的进度追踪。阶段规划必须为进度追踪奠下基础，进行进度追踪的方法则是建立一组详细的里程碑，使项目主管可以用来标示阶段中已完成的进度。

阶段里程碑是开发人员与其他项目成员必须经常满足的目标 —— 至少以每星期完成一个阶段里程碑的频率进行，时间短一些会更好。“ 每天 ” 也许是最小的里程碑时间了。每个里程碑都是 “ 二分性 ” 的，表示它只有完成与未完成状态，没有 “ 90%完成 ” 这回事。

要了解为何里程碑只能是二分性才有效，你可以想想下面的情形：假设你估计你需要一百桶喷漆来粉刷自家墙壁。如果你每隔三十秒被打扰一次，而将使用中的喷漆桶丢到一大堆桶子中，每次你回来继续上漆时，你只是拿一个最顺眼的桶来用。一个小时之后，如果有人问你用了多少喷漆，这时你有一大堆桶，有些完全空了，有些还是满满的，而大部分的则只用了一半，你怎么估计你用掉了多少喷漆？

如果你等桶里的漆用完了才换桶子，把空桶和还没用的漆桶分开放，你就很清楚用了多少桶漆。同样的道理也

可以用到软件项目管理上，如果你知道每个工作完成了没有，你就可以很容易估计出多少工作做完了，还剩多少工作没做。

阶段里程碑帮助团队将焦点放在最重要的工作上。如果只规划了长程里程碑，项目进行时就很容易随便浪费一天、两天、一星期或两星期的时间。人们把时间花在有趣或看起来有生产力而实际上对项目的推动没啥帮助的事情上。如果只采用长程里程碑的规划，开发人员会在开始时就比规划慢几天或一星期，到最后就更不在乎时程安排了。

“项目怎么慢了一年才完成？”.....

“因为每天都慢些时间。”

(Frederick P. Brooks)

使用短程里程碑时，你将了解项目会不会立刻碰到赶不上整体时间表的问题，如果项目一开始跟不上里程碑安排的步骤就有问题了。这样的预警让项目团队能够及早重新安排时间表，或以其他方式调整项目规划。

建立完整的里程碑清单

注意里程碑清单中将每个发行软件的必须工作都列上去。软件项目最常出现的错误就是忽略了必要的工作。不要让开发人员在脑海中、白板上或可粘留言便条上留下任何“时间表外的”工作清单。我处理过一个项目，其中有个开发人员发誓说他“几乎完成”而且即将“完全完成”所有指派给他的工作。当我们在接近发行日期、再次规划项目剩下的部分时，这名开发人员才说他还有大约需要六个星期才能完成的零星工作。他还严重低估了所遗漏的工作，实际漏掉的工作其实需要将近四个月的时间，而不是他说的六个星期。

这样的时程问题是这名开发人员的错吗？部分是的。不过他的主管可以要求更详尽的进度报告来避免这样的问题，而且他的主管应该这样要求的。

一定要将每件工作都放进里程碑清单中。细节设计，检查程序，更正检查中找到的缺陷，将工作成果与其他开发人员的整合在一起，整理应急的修正方式等等。当最后一个里程碑标上“完成”记号时，项目就应该是完成状态。未完成工作所带来的风险已经在本书描述的做法中多方面处理了。遗漏掉的工作在细节设计工作时就应该被找出来。

如果细节设计阶段也没找到这样的缺陷，那在建立阶段里程碑时也应该会处理好。

如同所有良好的软件项目规划，本书的做法是依据现实中会出现在项目里的错误而制定的。项目的成功取决于项目团队对快速轻松找出并修正这些错误的态度。

达到必要质量

在每个工作成果上进行技术检查有助于确保表里如一的达到阶段里程碑的要求。技术检查降低了开发人员把没做完的工作说成做完了的机率。如果有模块应该“完成”了，就该检查一下这个模块就究竟有没有真的完成了。

技术检查也有助于防范常见的累积工作量造成的质量问题。当开发人员们宣称程序“写好了”而且质量很好，你可以对项目已经完成多少有个清楚的了解。没有技术检查的把关，则低质量的工作成果也被当作完成，则会因为将更正错误的工作慢慢累积到项目末尾而伤害项目状态的

可见度。这些累积了的工作被大量隐藏，直到软件功能完成，质量问题才冒出来，软件不得发行时，这些隐藏起来的工作量才会被显露出来。

将低质量成果宣称为完成品的一个更危险的效应是，除了将工作推迟到项目末尾，还增加了提升软件质量水准的整体工作量。由于所有可能的连带影响，要对三、四个低质量组件的互动关系除错比将各组件分别除错要难得多了。

实际上，真正的伤害比上述的还要巨大。一个典型的中型项目可能有1 000~2 000个例程与250~500以上的模块组成。

如果质量管理工作拖到项目末期才做，开发人员就不只是面对三四个低质量组件除错时的互动关系了，他们除错时会面对数以千百计的低质量组件的互动关系。

你了解了这个关系后，就不难了解一个大体上似乎已经完成了的程序为何能够拖上好几个月的时间还达不到可以推出的质量。如果程序中每个元件的质量都很糟糕，那实际上，这软件也许永远也不可能达到可推出的

质量水准。

如何确定阶段里程碑

在确定阶段里程碑时，项目团队规划出下个重要目标的途径，阶段里程碑只安排从目前可预见范围内的详细路径。在每个阶段应该建立两次阶段里程碑。团队应该先定义一组细节设计的阶段里程碑，然后在完成细节设计后，定义第二组里程碑，让他们能够在阶段末尾推出软件。

确定阶段里程碑会花时间，可是许多人觉得只要定义好这些里程碑，工作就完成一半了。

小型项目的阶段里程碑

小型项目团队有时会认为他们不需要定义阶段里程碑，因为他们的项目小到只需要少许的管理。可是小型项目成本与时间的相对风险和大型项目是一样的。预计一个月完成的项目多拖了一个月虽然比预计一年完成的项目多拖了一年不显著，可是拖延的比例是一样的。使用阶段里程碑所需的工作量也与项目大小成比例。相对说来，阶段里程碑对小型项目提供与大型项目一样多的可见度与控制的好处。

政策考虑

有些开发人员会将阶段里程碑当成微观管理，这是正确的。更具体地说，阶段里程碑是微观的项目追踪方式。并不是所有的微观管理都是不好的，而这也不是开发人员讨厌的那种微观管理，只是更能真切有效地追踪项目进度。

有些人不了解他们的工作细节，而那些人会觉得这种做法让他们倍受威胁。如果项目主管将他们的反对意见圆满地处理掉，试着依据阶段里程碑的方式来进行工作将会是一种学习途径，而且他们会在熟悉使用阶段里程碑来安排工作的过程中渐渐得心应手起来。

项目主管应该让人们确实将自己的阶段里程碑制定出来，让他们能够控制自己工作的细节。整个阶段里程碑的做法需要的只是让项目成员告诉主管他们的工作细节是什么。然后将进度成果披露于公开可见的项目规划中，让自己的主管了解自己的进度。

在项目初期制定好阶段里程碑。如同其他的项目控制方式一样，阶段里程碑也可以在项目进行中先紧后松地执行。如Barry Boehm与Rony Ross发表于1989年7月IEEE Transactions on Software Engineering上的Theory-W

Software Project Management: Principles and Examples一文中所说的，“先硬后软比先软后硬要有成效”。你可以预期一开始会有些阻力，不过注意那些抗拒阶段里程碑做法的开发人员在各方面所付出的代价。这样的开发人员可以说是失去控制的——你没办法对这名开发人员的工作进行状态有清楚的了解。那名开发人员抗拒管制所花费的代价也会让你了解他对项目的贡献。

如何处理项目跟不上阶段里程碑的情形

如果实实在在地规划里程碑而项目团队也投注焦点在上头，要跟上时程，应该只要花费正常的工作时间跟少量的O’Connell 在《How to Run Successful Projects II: The Silver Bulle》一书中所谓的加班手术时间——在特定处赶上特定阶段里程碑所多花的几个钟头。

跟不上阶段里程碑，预示着项目整体上有赶不上时间表的风险存在。如果你发现项目经常跟不上里程碑，不要试着让项目团队加班卖命跟上每个里程碑，那样只会让项目进度拖得更慢。

如果一名开发人员全神贯注后还需要花费额外的加班时间来跟上里程碑的时间表，那名开发人员的时间表就应该

重新修正了。那名开发人员没得到足够的时间缓冲，而任何被更严重低估的工作项目会将整个项目的时间表都搞砸。

那名开发人员也需要获得足够的时间，避免仓促行事或做出掉以轻心的决定使项目花更久的时间。工作伶俐而努力固然很好，努力工作而不伶俐就糟了。重新修正时间表，让开发人员能够在正常的每天八小时工作中跟上自己的工作底限，将该名开发人员剩下的阶段里程碑的时数日期乘上时间表已经顺延了的长度。

当无法完成阶段里程碑时，重新修正小型时间表只是几种选择之一。你可以削减软件的功能，去除开发人员分心的因素，将项目的工作重新指派给轻松完成他们自己里程碑的开发人员，或是采取其他修正措施。

阶段规划与管理风格

有些项目主管会反对本章提出的紧迫管理方式。他们喜欢放手让手下做事的管理风格，因此他们不需要有阶段性完成的负担，详细的进度追踪负担，经常评估风险的负担，或是在项目状况变动时重新规划的负担。

在选择紧迫盯人或是放手而行的风格时，有一点要谨记在心，“放手去做”的意思是让项目的工作人员

放手去做，不是放任项目不管。采取放手去做的方式对于项目中的有些工作人员也许是最好的选择，有效率的项目领导者自然需要对不同项目需要的人手有具体的了解。

不过对项目本身采用放任的方式将招致项目的失败。放手管理会让项目多花费50~200倍的代价，而这些代价通常到了软件应该要推出的前夕才会浮现出来，因为这种做法没能实时处理一些重大风险，这些因素都会让原本可以成功的项目失败。

有效率的软件项目管理受益于系统化应用紧迫盯人策略。有些最有效率的做法需要额外工作，可是交换而来的是例外风险的降低、状态可见度的提高与时程的有效控制。细心的阶段规划包括阶段里程碑的使用，就是这样的做法。

求生检查

- 在各阶段初期进行该阶段活动的规划。
- 阶段规划包含需求检查，细节设计，程序写作与程序检查，测试条件建立，使用文

件的更新，阶段内错误修正，技术协调，整合与发行，风险管理，项目进度追踪与其他重要活动。

- 项目团队建立一系列协助追踪阶段进度的阶段里程碑。
  - 阶段里程碑清单不包括所有活动。
  - 项目没有确实照着阶段里程碑清单进行进度追踪。
- 项目主管对项目本身采取紧迫盯人的做法。

第 13 章

细节设计

细节设计是扩展从系统构架阶段开始的设计工作，针对许多相同问题进行更详细地处理。细节设计需要的工作量取决于项目规模与开发人员的素质。细节设计的审查提供显著的质量与成本获益。第一阶段的细节设计需要一些特别的工作，像是在一个构架设定以前先验证构架方式的质量。

各阶段的细节设计工作包括开发人员投注于设计阶段中将完成的细节部分。对后面阶段工作也许必要，不过重点应该是在接下来将要实施的项目上。如果构架定义良好，开发人员将不用思索太多稍后阶段中将要进行的工作状况就能建立出细节设计蓝图，而且能够照着构架处理使不同阶段的工作彼此相容。

重新检查构架

项目团队会重新检查构架阶段中处理过的一些地方。

程序组织

在构架时期，设计者关切系统层次的程序组织问题。在细节设计时期，设计者关切的则是对象类别与例程层次的程序组织问题。

重复使用分析

开发人员再次检查重复使用设计中现有组件与市场可取得组件的可行性，不过这次是对更细处的可重复使用性作检查。

需求解析

如我在第8章所提到的，项目团队在需求规格制定时期有时必须跳过一些需求设定。任何影响阶段性发行工作的需求必须在细节设计时先解决。如果此时还不能处理，只好留到后面阶段的细节设计程序，但是项目领导者应该留心上下游关系的效应，这些工作愈往后拖，将为前头未能决定出来的需求规格花费的工作和时间愈多。

需求可追踪性

细节设计必须与所有相关的需要呼应，如构架设计必须满足需求一样。有时这种情形被称为“向下需求流程”，因为需求流程一路从需求开发到构架阶段，再一路到细节设计、程序写作与测试项目的建立。需求可追踪性的有效运作是项目成功最重要的活动之一，关于这点我会在本章稍后再提起。

构建规划

开发人员应该建立一套细节构建蓝图和构建阶段的阶段里程碑清单。如果你进行细节设计时，开发人员表示他们没办法建立一套细节规划，那你就应该特别留意这一点。

这是他们细节设计不够周密的重要警讯。他们得继续进行细节设计的工作，直到他们能够建立构建阶段的阶段里程碑为止。

修正构架上的缺陷

即使构架方式杰出，而且检查人员已经费尽心思挑出构架的漏洞，在细节设计阶段还是可能找出构架方式上的一些缺陷。开发人员可能发现必须填补的漏洞，需要解决的矛盾，以及可被消除的寄生组件。这在第一阶段的细节设计时期也许不成立，因为这时项目只能依循着构架的蓝图进行。在稍后阶段的细节设计时期，这种情形比较常见，这时开发团队已经对构架了解更多，更清楚构架的优点和不足。

一个项目需要多少细节设计工作

设计工作与该工作的程序量取决于两个因素：项目开发人员的专业素质和项目的困难度，图13-1说明了这点。

如图所示，经验法则指出，如果项目有专业开发人员，而且项目本身很简单（只要花几个月来构建的那种），细节设计工作就不用做得太正式。设计工作可以跟构建过程

同时进行，开发人员也可以将许多构建工作与设计活动合并进行。

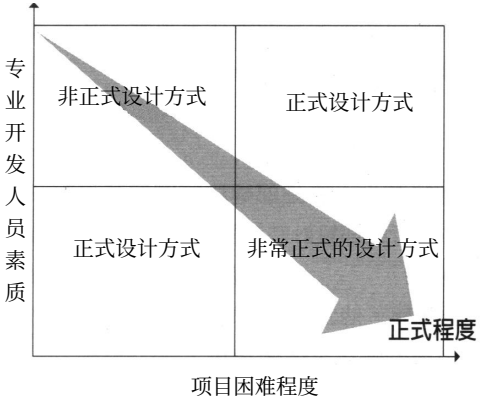


图13-1 细节设计程序、项目困难与开发人员专业素养之间的关联

如果项目的开发人员资历浅薄，而且项目难度较大（需要花超过好几个月时间来完成，包括应用开发人员不熟悉的技术，而这些技术又要求高稳定性，或是要求开发新的应用领域），项目团队就应该在系统各部分组件设计时采用正式的设计方式，并在开始构建之前先检查订好的设计方式。在这样的程序量下，可以预期看到系统各对象类别的模块流程图和各例程的文字说明。除了最根本的例程，所有例程都应该以伪代码写好，并在执行之前先检查好。

碰到中间情形时——资深开发人员进行困难的项目，或是资历浅的开发者进行简单的项目——通常需用到一些设计手续，而且我认为愈多手续比没进行好，这有几点理由。

一个理由是判定一名开发人员的专业素质可能很难。我在一个项目中担任过顾问，客户和一名对象导向设计的项目人员签约，要他协助处理该客户接手的第一件重大对象导向项目。这名项目人员宣称有五年的对象导向开发工作经验，他的资历让人觉得他应该可以把整个工作做得很好。当我们完成需求开发工作，开始进行设计工作时，项目主管对这名专家说，“你怎么不开会主持设计过程，指导我们建立一个对象导向设计方式的步骤？”

这项目人员的回答是，“这正是此项目中让我的客户紧张的地方，因为我不知道该怎么解释我作了哪些事情。我只知道我一定作得出来”。这里清楚地出现了一个红色警告，虽然他有好几年的经验，这位对象导向“专家”对于物件导向设计，显然知道得不会比看过几篇文章的团队成员多。

如果你设定了太多设计手续，虽然项目团队还是会产生出一套有效的设计方式，可是项目团队用来建立设计流

程图、写出伪代码或在程序之外分别检查细节设计所进行的各项手续将会增加项目的负担。

如果你设定过少的设计手续，你会让构建和测试工作冒着低质量与容易出错的风险。那样会增加后面项目的工作量，增加时程与预算风险。

有经验的人都会同意过多的设计手续比过少手续要好。

成功的项目寻求以增加小量的工作负担来交换项目风险的降低，这也是一种典型做法。在这例子中，额外进行的工作是否形成“负担”都还是令人质疑的，没错，这种做法会增加说明文件，但毫无疑问对以后进行系统维护的开发人员有所帮助，而且现在开发系统的人可能就是以后维护系统的人。

技术检查

当某一特定部分的设计者认为设计工作完成后，已经完成的设计方式应该经过大家的检查。两三名原来设

计者之外的开发人员应在开会以前先检查设计的方式。如果有三个以上的人对设计进行检查，也许会找出更多缺陷，这样作会使成本效益迅速下降。你当然不会在至少有两名检查人员可用时，只叫两个人来检查设计方式。许多软件工程研究发现，不同的检查者可以找出不同的设计缺陷；如果你只用一个人去检查，几乎可以肯定设计中会有重大缺陷没被找出来。

开会前的检查工作是很基本的，因为大部分潜在缺陷都可以在开检查会议前被发现。如果项目团队被迫选择取消会前检查或是会议本身，应该取消的是检查会议本身，然后让检查人员个别对设计者报告自己找到的设计缺陷。

在设计会议上要检查的项目依各组织的情形而不同。项目小组应该试着从经验中获取最佳检查项目数量。由试着检查一个对象类别、几个例程或是一次不高过一百行伪代码的设计量做为起点。检查小部分，而不要将检查者的时间切得太零碎，叫一个检查者看太多东西只会让他承受过大负担而伤害检查的水准。

找出功能缺陷

在开检查会议时，每个细节设计组件都应该检查下列

特性：

◆正确性：设计是否如预期运作？

◆完成度：设计成果是否适用于所有预期用途？

◆清晰性：设计方式是否易于被其他人了解？

复杂的设计方式与错误的增加相关。设计方式应该尽可能简化，以避免初期开发阶段的错误。更重要的是软件维护研究发现，维护程序员必须花掉修改程序的时间来了解原来的程序。由于一般会由维护程序员进行维护，在设计检查时多花一些时间来了解清晰性将可为日后省下许多时间。

找出需求缺陷

设计方式也该评估需求可追踪性。这是说，检查设计方式时要确定所有必要需求都已获得解决以及是否有不必要的组件。

1. 遗漏需求

很重要的事是找出被遗漏的需求，因为这样有助于避免对需求设计遗漏的错误。如果设计方式忽略了一项需求，系统在项目后头就得以更高的代价来回溯满足那项需求。

2. 不必要的功能

找出不必要的功能也是很重要的，因为不必要的功能会增加项目的成本与时间。增加一点额外功能可能影响不大，不过增加这样的小功能，即使使用者看不到这东西都可能增加以下负担：

- ◆增加程序写作、测试程序与除错时间。
- ◆增加复杂性，让系统更容易出错。
- ◆额外的系统测试项目。
- ◆在追踪错误系统中找出并修正、追踪额外的缺陷。
- ◆额外的使用文件。
- ◆额外的使用者支持训练。
- ◆额外的使用者支持电子邮件和电话服务。
- ◆未来版本的程序中必须支持的额外功能。

我有次检查一个用来处理精密分析功能程序的设计。使用的分析运算基本上很复杂，应该是世界上第一个那样写的系统。要把那样的设计跟实作最简化的设计比较应该是重大的挑战。

项目团队决定让系统的不同组件以很奇怪的“异步”的方式互动。当系统“同步”运作时，程序的一部分会呼叫另一部分，等待被呼叫者完成工作，然后呼叫者才继续

进行自己的工作。“异步”运作下，程序的一部分呼叫另一部分后，可以立刻进行自己的工作，然后定期检查被呼叫者是否完成工作。

当被呼叫者完成工作后，呼叫者再采取对应的行动。

可想而之，异步的做法比起同步运作要复杂多了。如果程序的几百个地方套上这样额外的复杂性，可以想象程序异步运作得多花多少经费和时间。异步处理是个“奇怪”功能，可是并不是必需的，比已经够复杂的系统的需要增加50%~100%工作量，而对出钱开发这系统的人来说，根本看不到这种异步的功能有什么用处，如果让他们选择是否要采用保证提高项目预算50%~100%的异步方式，他们一定会说：“不要”。

除了成本的因素，不需要的程序代码也会对软件本身造成灾难性的后果。

在欧洲共同体开发的亚利安五型火箭1996年第一次发射时，由于一些不必要的调整性程序而让发射失败。那些程序被用在前几型亚利安火箭上，只是没从亚利安五型上移除。

细节设计时期可以使项目成本减少一半或加倍，决定软件更可靠或更不稳定的关键时期，而这都取决于开

发团队寻求的是简化设计的方法或是让软件更广泛、更复杂而具技术挑战性的方法。当通盘考虑项目内容时，几个奇怪的不必要的功能就足以改变项目最终的成本。

因为这可以省下额外预算与不可靠功能实作的时间，技术检查可以进行许多次。

项目目标的检查

设计检查也是开发人员能够讨论让设计最符合项目目标的时候。如果项目的目标是以最低成本完成软件，检查人员常可以找出让设计人员以更便宜的方法进行改进。如果目标是最后的适应性、可移植性或其他目标，检查人员也常常可以找到完成这些目标的更佳途径。

项目初期订下的前景目标在这阶段变得相当重要。如果眼光敏锐，就会在设计检查时指出向目标迈进的方向。如果前景不明，就不能提供这样的行动方针，而设计检查也就不能以这样的方向改进软件。

交叉训练

设计检查的一个好处是提供关键开发人员出了问题的保险。如果开发人员的设计方式被两名以上的检查人员看过，他们就会熟悉原设计者的工作成果，并能够在原设计者出了问题后接替他的工作。原设计者如果出了意外，也许令人难过，不过并不会让项目因此取消。设计检查也对愚蠢的设计方式提供保险。你可以把这些检查看做是“笨蛋保险”。

检查与生产力

通过检查找出缺陷可省下庞大的潜在成本。约有60%的缺陷常出现在设计时期，而项目团队应该试着在设计时期就把这些缺陷都清除掉。在细节设计时期不着重在找出缺陷上的决定，将使找出缺陷并修正的行动延后到项目末期，使这些错误到时得花费昂贵成本和更多时间来进行修正。

由于可以及早在开发周期中使用技术检查——可以使开发时间减少10%~30%。一个对大型程序的研究发现，花在检查上的每小时可以避免平均33小时的维护时间，而且比对程序进行测试要有效率达20倍以上。

细节设计文件

开发人员应该替每个程序组件建立细节设计文件。依据项目规模，每个细节设计文件可能会涵盖一个对象类别、一组对象类别或是一整个子系统。对大部分中型项目（本书所讨论的项目规模），就适于给每个子系统一份细节设计文件。

这些文件并不需要特别正式。依据项目的正式性层次来决定恰当的做法，这些文件可以是活页式的设计流程图，设计流程图跟伪代码的组合，或是包含介绍、设计流程图、伪代码、需求可追踪性表格跟其他相关项目的小型文件。

不管细节设计有多正式，每个细节设计文件都应该在对应的设计方式通过检查后归属于更动管制的管辖。有些更动管制管辖下的工作成果，像需求规格、构架方式以及程序写作标准，不经常更新。因为对这些工作成果的更动会对成本与时间表产生重大影响，所以应该管制这部分成果难以更动。其他工作成果，如源代程序代码就需要常更新，而这些东西的更动管制程序就应该更简化。如果细节设计文件是在子系统层次撰写的，项目进行到各阶段的子系统更新时，就会连带更新细节设计文件许多次。对细节设计文件的更动管制程序应该像源代码的更新管制程序般

简化，而不该像上游的工作成果管制得很严格。

第一个阶段的特别考虑

在项目的第一个阶段，设计上会有一些特别考虑。在中型跟中大型项目中，构架方式应该在阶段性开发以前完成。在小型项目或由极资深开发人员参与的项目中，构架方式有时会放到第一阶段处理。

不管构架方式是分开或合并到第一阶段中，第一个阶段都应该对系统构架的健全性和风险进行探索。

了解构架风险的最佳方式取决于了解系统的“水平”和“垂直”面。假设你有个产生五种图表的分析系统，你可以定义一组图表，而且能够编辑、列印、储存或重新加载这些图表。系统的一个“水平面”会触及这五种图表的各种功能，可能呈现出五种图表的大致面貌。系统的一个“垂直”面则包含由上而下的图表功能构架：编辑、打印、储存与重新加载。

如果项目面临强烈的技术风险（应用尖端技术，开发团队不熟悉开发工具，或是二者兼有），放在第一阶段的重点在系统垂直面上。这样让团队运用程序中使用的技术，能够对构架上需要做到的事情先演练一番。如果项目面对

强烈的技术风险（在了解透彻的环境中使用熟悉的工具），确定团队在第一阶段中建立的系统部分会动，而且探索所有子系统间的重要接口与互动关系。通常第一阶段实作应该涵盖80%的系统层面和20%的功能程度。

开发人员可以先解决系统中最艰难的部分。他们不必将那些部分全部完成，不过他们应该让那些部分完成到足以显露出主要问题，并确定他们拥有详尽规划来解决这些风险。

求生检查

- 项目团队替每个子系统建立细节设计文件，将构架考虑到细节层次，并将这些文件置于变动管制的管辖下。
  - 细节设计不被检查或很敷衍地检查。
  - 细节设计文件无法解决需要追踪性的问题。
- 细节设计工作的正式性似乎相当适合项目规模和开发人员的专业素质。
  - 细节设计工作太不正式。

- 细节设计检查着重在找出功能缺陷、需求错误与更满足项目目标的方式。
- 项目第一阶段的细节设计找出构架的潜在问题。

第 14 章

构建过程

构建过程是开发团队带给系统生命的刺激时刻。如果在这阶段之前的活动都有效进行了，构建过程将会是开发人员通过每日整建与测试来赋予系统功能的美好时刻。在构建过程中，成功的项目团队渐渐警醒于找出让软件简化并控制更改项目的方法。当项目主管追踪包括阶段里程碑、缺陷、十大风险清单与系统本身在内的进度指针时，项目进度将十分易见。

在构建过程中，开发人员开始建立运作程序的源代码，赋予系统新生命。每个开发人员填补细节设计阶段的缺陷，建立源代码与单位测试项目，对源代码进行除错，并将源代码与项目的正本档案整合在一起。

在执行良好的项目中，再构建过程累积的功能提升了整个项目的进度。当开发人员各自进行其程序建立工作后，项目开始以高效率运转，主管与使用者们看到每日累积的新功能，对系统产生渐增的信心。

构建过程可以是软件项目中最快乐的阶段，也可以是最具争议性的阶段。本章描述如何让构建过程保持生产力与乐趣。

源代码质量

软件构建的方式对于系统的成本有重大影响。把事情复杂化的程序和正确而有益的程序之间的区别在于应用详尽的程序写作技术。如此详尽的技术必须应用在程序代码开始构建时，因为一个不可弥补的错误是不可能回溯补救而变成一个具有扩充性程序的。

程序写作标准

建立优美程序而不让问题复杂化的关键之一是建立一套程序写作标准。程序写作标准的目标是让整个程序如单一毛料织成的毛毯，而非东钉西补成的拼凑品。当项目进行时，让程序源代码有个一贯外观和质感有助于开发人员阅读彼此的程序代码。当原来的开发人员离开项目而由新的开发人员接管软件时，如果软件依循一致的风格写作，新的开发人员就可以迅速了解程序内容。

程序写作标准一般针对下面各项目：

- 对象类别、模块、例程与例程内部程序代码的编排。
- 对象类别、模块、例程与例程内程序代码的批注。
- 变量名称。
- 函式名称，包括日常操作名称，例如自对象类别或模块中取得和设定资料。
- 例程源代码的最大行数长度。
- 对象类别内最大例程数。
- 允许的复杂度，包括对goto叙述、复杂的逻辑测试、巢状循环等等的使用限制。
- 内存管理、错误处理、字符串存放等等的程序代码

层次构架加强标准。

- 使用工具与链接库版本。
- 工具与链接库的使用方式。
- 源代码档案的命名方式。
- 开发机器、项目建立机器与源代码管制工具的源代码目录结构。
- 源代码档案内容（例如每个C++对象类别各自存放在一个档案中）。
- 如何标示未完成的程序代码（例如加上“ TBD ”批注，TBD = To Be Done，留待完成之意）。

在一个组织中，大部分程序写作标准的内容在不同项目间应该都是一样的，至少使用同样的程序设计语言。主要的不同点在于依据个别项目构架方式的加强处理。

最好的程序写作标准应该是简短的通常不超过 25页。没有必要将项目中每一个项目都标准化 —— 开发人员不可能记得那么多标准，也不可能通通做到。

建立程序代码写作标准可能有很大的争议，但是这些标准被依循的程度要比这些标准的具体细节重要多了。程序代码写作方式的标准对项目的主要贡献是产生项目标准或公司标准的源代码。

程序代码写作标准一般在程序检查中执行。程序检查的主要作用是找出程序中的缺陷，不过附带作用则是将所有源代码依据项目的程序写作标准进行规格统一。

如果组织中有个独立的维护组织，让维护组织中的成员参与程序检查会有些好处，因为这名外来成员对于程序是否易于维护有着强烈兴趣。

项目目标

如细节设计过程，在构建过程中，开发人员应该持续寻找最佳化项目目标的方法。构建过程提供许多让项目更简化或更复杂、更稳定或更脆弱、更灵巧或更笨重的机会。

开发人员在构建过程中会确实做出几千个影响这些因素的低级决定。这并非夸大，一个中型程序的75 000行程序代码或许可被组织成3 500个子程序。开发人员会在每个例程中作出几个影响软件复杂度、稳定度与表现的决定。他们会决定该如何说明例程，该使用哪种程序写作风格，该如何处理错误，该如何检验例程正确运作，并且决定例程真正进行的工作目的。一个清楚的项目前景有助于确保这几千个决定中的多数能与项目目标配合。缺乏前瞻的做

法则将使这些决定大部分都毫无特定焦点。

软件构建将完成非常细节程度的工作，回溯这几千个决定等于是将整个系统改写。

特别重要的是如果你没有在第一次决定时弄对方向，你就没有第二次机会了。

单纯

构建过程为开发人员提供简化程序以降低复杂性的机会。项目很少因为设计与实作不够复杂而取消。许多项目被取消，是因为它们太复杂，以致没有人能够弄懂这些软件到底在做什么，即使改变或扩充系统，也容易造成无数的副作用，而这些副作用多到让整个系统难以再继续扩充下去。

软件整合程序

软件整合程序是构建软件成功与否的重大支柱，这程序规划了将新开发的程序代码加入项目软件主体的方式。

表14-1列出了我建议的整合程序。

表14-1　建议整合程序

1. 开发人员发展一份程序代码
2. 开发人员单元测试这份程序代码
3. 开发人员在交谈式除错器中逐行追踪执行程序，这包括所有例外与错误情形的处理程序
4. 开发人员将初步测试好的程序代码与项目软件主体的个人复制版本进行整合
5. 开发人员提交程序代码进行技术检查
6. 开发人员将程序代码交给测试人员进行非正式的测试项目准备
7. 程序检查完成
8. 开发人员修正检查过程中指出的任何问题
9. 修正后程序的再检查
10. 开发人员将最后的程序代码整合进项目软件主体中
11. 该程序代码宣告“完成”，在项目活动清单中标记完成

完成就是做好了

整合程序提供重要的项目管制，确保一份程序代码宣告“完成”时才真正“完成”了。当你看到一份状态报告说“90%的模块完成了”，你可以相信那些模块真的都完

成了，而不用担心其中还有很多工作没做好。

习惯于“部分实作”的开发人员必须调整其工作习惯来适应这种做法。那些开发人员会发现因为他们的程序部分依赖其他程序所提供的功能，而这些程序还在等待完成中，只好写出一个速成拼凑版的必要功能实作，再继续写他们本来进行中的东西。这个速成拼凑版的实作往往有所限制，而且开发人员得在他们的主程序代码中加上突破那些限制的解决方法。细心的开发人员会记得回过头把速成拼凑的东西重新整理一遍，再修订主程序去除那些原来搭配拼凑出来的例程所必要的解决步骤。

由于这种方式本身在项目进行中重复了许多遍，开发人员最后会得到一份冗长的“等待处理”事项清单。这样的开发方式对于迅速作出许多可见功能是个好方法，可是这样会让项目必须持续进行到整理阶段，而其中的工作量将是很难预计的。

使用表14-1中提供的整合程序要求纪律，对于习惯部分开发实作策略的人来说是个显著的调整。这样的整合程序，使得进行下个模块开发之前可以降低风险、增加可见度与控制的优点，让这做法变于实施。这个整合程序对不习惯的开发人员似乎麻烦，可是能够避免缺陷未

能及时发现。

其他开发人员成果的稳定基础

这个整合程序也解决了非到项目后期才看得见质量的重大风险。在控制不良的项目中，源代码在质量确立以前就整合进主项目中。如果缺陷没有立刻出现，开发人员就略过那段程序代码，继续进行下个部分的工作。到了项目末尾，已经整合好的程序代码质量低到了让问题开始浮现台面，而由于各个程序代码都没被要求达到高质量标准，修正一个问题通常会制造两三个额外的问题。这时项目进入了微软公司所谓“无限缺陷”的地步——错误出现得比修正的速度还快，这种趋势如果没有更正过来，就真的会产生出问题无穷多的软件来。因为这种潜在质量问题的风险而被迫取消的项目数量，即使没破千也有几百。对此软件整合程序是积极以增加小额项目负担来换取项目风险大幅下降的一种做法。

每日整建与冒烟测试

补足软件整合程序的一个做法是利用表14-2提供的每日整建与冒烟测试程序。

表14-2 每日整建与冒烟测试程序[①]

1. 合并修改程序。开发人员将自己的源代码与项目主体源代码比较，找出最近由其他开发人员修改的程序跟新增或修订过的程序中冲突与不协调之处。然后将自己修改的程序代码与项目主体源代码合并，经由自动化源代码管制工具处理，使开发人员能够得知任何不协调的地方
2. 建立与测试个人项目版本。开发人员建立并测试项目的个人版本，以确定新实作的功能仍然如预期运作
3. 执行冒烟测试。开发人员在个人版本的软件上执行目前的冒烟测试项目，确定新的程序代码不出故障
4. 记录归档。开发人员将自己的源代码归档到项目主体源代码中。有些项目可以确立新程序代码加入项目主体的时间、例如新的程序代码必须在早上7点以后，下午5点以前完成归档
5. 产生每日完成版本。整建团队（或整建者）从项目主体源代码产生完整的程序代码执行码版本
6. 进行冒烟测试。整建团队进行冒烟测试来评估目前的完成版本是否稳定得足以通过测试
7. 立刻修正任何问题！如果整建团队发现任何让新版本程序通不过测试的错误（使程序故障的错误），就会通知程序故障的开发人员，

(续)

让那名开发人员立刻修正问题。修正目前版本的问题是项目最优先要做的事情

① 这个程序由开发人员依照表14-1建议的第十步骤开始进行。这里假设开发人员已经检查过需要更动的源代码档案，而且可能建立好新档案了。

在每日整建与冒烟测试程序中，整个程序每天都会重新编译出新版本。这代表程序的每个源代码档案都重新编译、连结并合并成一个执行程序。产生出来的执行程序接着会进行“冒烟测试”，一个相当简单的测试，看看程序能否接受测试条件，或者在接受测试时有故障。“冒烟测试”一词来自电机工程界，方法为将机器打开，看机器在运转时会不会中途冒烟烧掉。

每天花点功夫来进行整建与冒烟测试的工作，找出让完成版本坏掉的开发人员，确保他们立刻修好坏掉的源代码。在小型项目中，每日完成的版本可由一名质量保证人员兼职进行。在较大型的项目中，也许就需要一名专职人员或一组人来进行每日整建的工作。

冒烟测试应该随着项目进展而更新内容，以跟上软件

开发的脚步。由于冒烟测试每天都会进行，所以常常是自动化处理的。冒烟测试不是一种消耗性测试，不过内容应该涵盖足够的软件功能，以提供每天产生的版本程序是否稳定得足以通过测试的评估。如果软件稳定度不够通过测试，冒烟测试就会让软件当掉。

每日整建与冒烟测试的做法降低了团队项目面对最大风险的可能性——当不同团队成员组合或“整合”自己分开写好的程序代码时，组合起来的程序不会动作。它也解决了低质量软件的风险。通过每天至少对整个软件测试一遍，质量问题就不会掌握项目存活的去向。项目团队将软件推进到已知的良好境界，然后一直保持软件的质量水准，软件就不至于恶化到让耗时质量问题出现的地步。

每日整建也让项目进度的监督工作容易一些。当项目团队每天重新编译系统时，已完成与未完成的功能清楚可见，技术人员与非技术人员都能够简单地从软件表现得知距离整体完成还有多远。

这种每日整建与冒烟测试的做法已经在不同规模的项目中成功运用，包括庞大的Microsoft Windows NT 3.0项目

中——该项目的源代码行数超过500万行。对大型项目来说，每日整建与冒烟测试特别重要，因为大型项目中整合程序失败的风险是如此明显。

第一阶段的特别考量

在第一阶段构建过程中，开发团队应该建立系统的骨架，一个足以支持其他系统功能的构架，一般涵盖了建立包括菜单、工具列与其他在开发过程中与实际功能搭配的使用者接口的外表。

建立系统骨架需要一些在第一阶段中开发出来的基础功能。一般说来，如错误处理程序、字符串处理函式与内存管理程序等低级工具，需要在其他实作开始进行之前先完成。如图14-1所示，基本上这表示开发出系统中一个呈现T形的部分——系统的完整宽度与狭长的垂直面。之所以被当成“T”形，是因为整个使用者接口的宽度都已经开发了（不过没有一个元素完全会动），而且整个支持基础的深度也完全开发好了（不过没有一个基础功能会被其他功能呼叫）。系统的宽度与深度足以满足此阶段的功能需求。

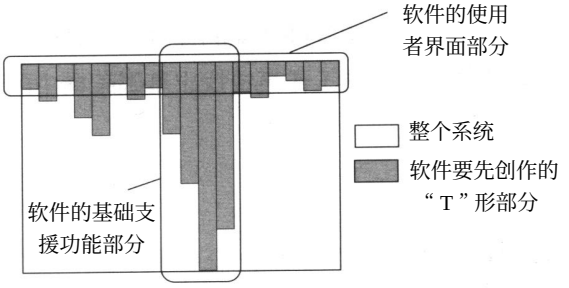


图14-1　在第一阶段以“T”形开发系统骨架

避免未成熟的基础开发工作

基础功能以许多名称出现，像“基础程序”和“基底对象类别”，其实这些都是指一样的东西。

要让项目快速进行，跟“T”形基础功能一样重要的是开发真正可见的完整功能面。特别重要的是，不要试着在某些实际功能还没完成前开发整个基础构架。理论上，在进行任何可见功能的实作之前先行开发基础功能也许是有效率的，可是实务时，主管、顾客和开发人员会紧张，如果在软件执行以前就花了太多时间的话，会发生什么糟糕的事情难以预测。

基础开发工作也可能变成一个完美理想构架的研究计划，通常是指许多实际用不着的功能跑进项目中。项目团队应该只考虑目前阶段已经完成的实际功能所需要的基础

功能。

在完成整个基础功能以前先开发实际可见的功能是最有效率的方式，因为这避免让基础功能的建立变成花哨的研究计划，同时比纯粹开发基础功能的工作更快找出基础构架中的问题所在。

进度追踪

进度追踪对构建过程是非常重要的。在早期阶段，工作无法分割成小部分进行，然而在构建过程中，工作可被切割成以数天时间为单位或更短的阶段里程碑来作，以便让项目的工作进度都能保持记录。

收集状态信息

自动化工具确实减轻了追踪阶段里程碑所需的工作。阶段里程碑清单应该包含在电子规划工具，如 Microsoft Project或一个电子表格中。它应该存放在项目的修订管制系统中，可以透过项目的网络首页来存取，这让所有项目团队成员都能够取得这份清单。当一名开发人员或测试者完成一个完成点后，应该立即检查修订管制系统的完成点清单，将已经完成的工作项目更新，并标记清楚，再将完

成点清单归档。

时间管理系统也应该以相似的方式维护。有些组织有项目人员填写时间表，然后让管理人员执行各个时间项目。自从人们熟悉自动化系统后，线上执行数据项所花的时间就少得多了。项目成员应该将自己所做的时间安排，每周至少输入时间管理系统一次。如果不这样做，时间管理就变得不精确了。

可见性

有效率的规划与时间管理信息的收集，对项目有着相当大好处。项目主管应避免让等待处理的信息变成项目执行的瓶颈。

其他让项目公开化的理由，是不同项目成员对事情有着不同的影响。当你将台面上与台面下的意见与问题结合时，你就确保项目主管与上层主管能够倾听新问题，才能达成公开化的目标。

每周更新项目进度追踪

在这段期间，每星期项目主管应该检查一次项目状态。这个检查应该包括以下内容：

- 从项目规划、错误追踪与时间管理工具中收集概要资料。
- 将实际完成的阶段里程碑与规划中的进行比较。
- 将实际找到的缺陷与预测应该有的缺陷进行比较。
- 将实际工作与规划工作进行比较。
- 检查更新十大风险清单。
- 检查任何透过匿名意见回报管道发表的意见。
- 检查由项目变动公布栏认可跟提出的变更项目，并检查项目规划中的变更项。

目的累积效果

根据这次检查，如果实际结果与规划中的情形严重脱轨或是出现新问题，项目主管都应该采取修正的措施。

每周收集的资料与分析，也可在各阶段末尾用来更新软件项目记录，以建立软件项目历程的基础。

与顾客和上层主管沟通

检查进度时项目主管应该事先经常与上层主管、顾客、使用者和其他项目资助者沟通，加强他们对项目怀的信心。如果等着他们来询问项目成果时，大概已经开

始对进度感到紧张了。到那时，项目主管所能做的只是说服他们，不过项目主管可能从此以后就让他们失去信心了。

项目主管应该在项目资助者们开始紧张以前，主动说明项目进展情况。大部分人宁可每星期一次，也不要每个月才一次的“ Everything is OK ”。采取积极态度，将被视为更合作、负责而且诚实的合作伙伴。

控制变动

当软件开始运作时，使用者和主管们会发现软件和他们预期的不同。有些人能理解差异存在，因为他们知道软件还在开发中。至于问题的出现是因为错误，还是因为有内容还没完成？这是无法从单纯使用软件来分辨的。

有些人理解差异实际上会造成软件的变更，而且在构建时期要求变更的压力从各方面涌现。

变更软件的压力愈大，谨慎面对改变就越重要，不然项目就会失去控制。

如果项目这时已经不在控制下，失控的变更将更容易使项目脱离时间表和预算目标。

项目变动公布栏对于管制变动是个重要的方式。如果变动公布栏有效运作，栏内的决定一定会被所有项目资助者，包括上层主管、市场人士、顾客与其他提议变更的人所尊重。在项目初期的需求开发阶段实施变动公布栏的做法，会使其在项目达到构建时期时得到合法性。所有人都会习惯将变更意见发表到公布栏上，通知受影响的人们考虑，然后接受变动公布栏上的决定。对人们来说，在软件预计要推出的前三个星期才头一次整合一个系统化的变动控制程序到工作中是很困难的。如果变动公布栏的设立只是为了否决人们的意见，自然不容易树立公布栏的威信。一个长期设立的变动公布栏应避免这类问题，而且公布栏本身就是在项目初期设立的最佳帮手。

保持焦点

项目主管在这阶段的部分工作是避免开发人员分神——特别是避免让他们因为使用者、上层主管、市场人士等等的软件变更要求而分神。项目主管应该了解所有

变更要求应该透过正式的变动控制管道表达，而非直接传达给开发人员，好让开发人员不致从主要工作中分心。

这就是构建过程中所有要做的事情

构建过程在软件开发中是相当关键的。良好的构建过程可以让项目节约一半的时间完成。良好的构建过程为快乐的软件维护工作奠定基础；不好的构建过程只会让历任维护者叫苦连天，花费倍增。

从软件项目求生的观点，项目最终成功或失败的大部分基础，在项目到达构建时期时就已经确立了。如果团队彻底调查了使用者需求，细心勾绘出设计方式，建立良好的构架，安排好阶段性完成规划，小心评估项目成本与时间，有效控制变动，构建过程就会因为进展平稳、过程顺畅而表现卓著。

如果项目的上游阶段已经有效进行了，构建过程将会大幅进展。

构建过程是管理不良的项目开始翻身的时期。这样的项目在早期阶段不怎么在乎需求开发、构架分析与细节设计所出现对于快速的进展，通常在短时间内即产生会动的程序代码。问题只有在测试人员和一般使用者执行的程序代码量够多时，才会突显出来，人们开始发现程序中的大量错误。到那时候，必须花费庞大代价来修正。早期快速进展的不良做法开始原形毕露，被当成构思不良的快捷方式看待。开发资源一开始即被限制着，以至于开发进程开始拖延，预算开始追加，还留下许多严重的错误、缺陷未被发掘出来，在开发人员、测试人员、主管们与客户们之间产生许多痛苦的对立。

在执行良好的项目中，相同的问题都在代价很低的上游阶段被找出来。大部分在构建过程中找到的问题都是构建上的问题，不用花什么功夫，也不用面对毁灭性的对立，就可以修好了。

求生检查

- 项目有一套程序写作标准。
- 程序写作标准透过对所有程序代码的技术

检查实行。

- 项目有一套软件整合程序。
  - 开发人员没依循这套程序。
- 项目团队在第一阶段建立一套系统骨架。
  - 项目团队在建立任何可见功能以前陷入建立完整基础功能的泥淖中。
- 项目采取每日整建与冒烟测试。
  - 每日完成版本的程序比起没有采用这做法要更常坏掉。
  - 冒烟测试没和每日整建工作一起做，也没测试软件的完整功能。
- 项目主管每周追踪相关进度指针，包括阶段里程碑，错误数量，变更报告，时间管理资料及十大风险清单。
- 项目状态资料公开呈现给所有项目成员取用。
  - 状态信息不常报告给顾客或上层主管。
- 变动通过变动控制公布栏的使用持续控制着。
  - 变动控制公布栏到了构建过程开始前都还没设立好，以致权威感尽失。

第 15 章

系统测试

系统测试与构建阶段同时进行或略慢半步。系统在这阶段从头到尾测试一遍，先找出缺陷后由开发人员修正。继而由测试人员帮助开发人员确保系统质量仍然能够维持在和新程序代码整合，开发人员则快速修正错误来帮助测试人员。

系统测试要通过确认从头到尾的功能正常才算完成。要确认所有必要项目的质量都在水准以上。在一些项目中，系统测试被延到项目末尾才进行。本章节从逻辑上与上一章的软件构建是分开的，不过时间顺序上系统测试与构建活动应该同时进行。

测试哲学

测试经常是软件项目中最关键的过程。因为直到软件大部分完成之前，没有人规划进行测试工作，在那种状态下，就不可能快速做好足以完整测试软件的工作，测试过程往往被省略掉，而项目带着许多尚待发掘的缺陷就推出。

在本书的做法中，系统测试是以密集步伐伴随软件开发的。测试项目应该比软件实作本身稍早一步或同时进行。在真正开始开发之前，是不可能这么做的。可是再依照本书做法进行的项目里，测试人员将有个完全的使用者接口雏形与详尽的使用手册，用来开发自己的测试项目。测试人员同时也在软件提出来进行程序检查时，收到软件的非正式版本，测试人员应该在软件通过程序检查阶段后不久就准备好自己的测试项目。

系统测试在软件项目中的地位

系统测试在本书的做法中并不是事后添加的，可是系统测试工作拖到末尾再进行的项目中，质量多半不佳。如图15-1所示，大部分项目将质量保证工作延后到项目末尾才进行。

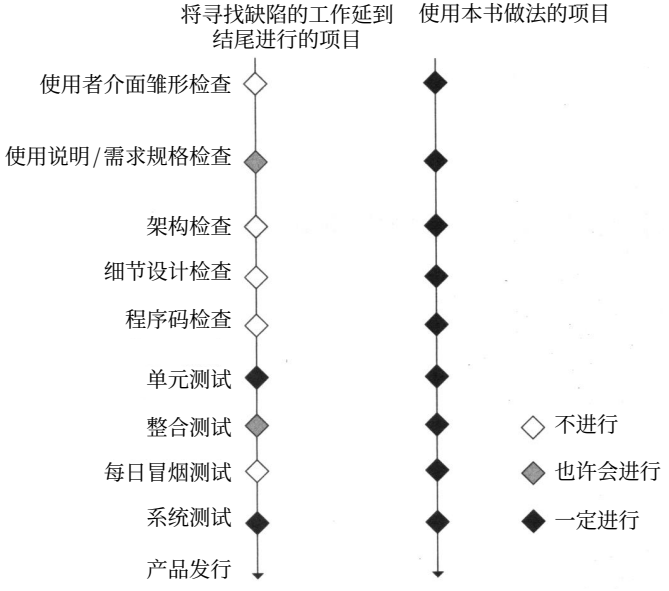


图15-1　使用比本书做法要传统得多的功能开发工作

本书的做法：任何特定功能到了系统测试时期，就已经经过检查了，如使用者接口雏形、使用说明/需求规格、系统架构、细节设计与程序等。开发人员也做过单元测试

和整合测试，以及每日冒烟测试。这些程序都不会留下太多隐藏的错误，所以系统测试就变得比较不重要了。

系统测试的程度

系统测试应该在系统的整个范围内进行。测试项目应该设计成能够确保实作每个必要项目，其每一行程序代码都执行无误。

在完成项目的每个阶段中，系统测试包含现阶段与前阶段需求规格完成后的“回归测试”阶段再加上新的测试项目，以涵盖新实作的必要项目，确保新的程序代码不会破坏旧程序代码的功能。

测试小组对每日整建的帮助

一般说来，测试小组会每天进行冒烟测试，检验每天建立的执行版本能否通过测试。冒烟测试决定每天建立的程序版本，是否够稳定到能进行系统测试。这些测试一般花费半小时以内的时间，如果当天建立的程序没能通过冒烟测试，测试小组就会要求开发小组立刻进行修正。

开发人员对系统测试的帮助

开发人员需要快速修理错误，好让测试团队不用等到程序版本十分稳定的状态下。每周项目进度追踪应该列出一张尚未解决缺陷数量的图表。这些缺陷数目应该维持得很小，如果可以要求开发人员把未解决的缺陷数压低到10以下（或其他合理的数字），你就能够维持软件的质量水准。这种做法有助于让快速而草率的程序员们，在解决一大堆自己弄出来的错误之前，不再进行新功能的开发。

开发人员有时会在错误找出来时，不想立刻修正。他们会说花时间修理错误会让他们分神，让他们的工作缺乏效率。别相信这套鬼话。他们不明白，写错的程序使测试人员跟其他开发人员失去工作效率。不管你怎么看待这个问题，找出已知问题是浪费时间的。如果问题几小时前才被发现，浪费的时间就当作是进行软件项目中不可避免的工作而已。不过当错误是在几天前就找出来的，问题应该早就被修正好了。这样浪费掉的时间是完全不必要的，而且是开发人员可以避免的。

策略性质量保证

系统测试的用处不光是修正特定错误，还能当作改进软件质量的策略应用。通常约80%的系统错误出现在20%的例程中。几项软件工程研究显示这些例程对系统总成本产生不相称的影响。

使用错误追踪系统来确认主要制造系统错误的例程。任何产生超出10个错误的例程都该被标示出来。作一番改进设计检查，使它们能够符合质量标准。并重新设计跟实作最有问题的那些例程。

在一个典型项目中，约80%的时间是花在规划之外的重复工作。施行些小策略，规划中的重复工作将能大幅改善软件质量与项目的整体生产力。

求生检查

- 系统测试与构建过程准备同时进行。
- 测试人员对每天建立的程序版本进行冒烟测试，并在测试失败时让开发人员修正缺陷。

- 每天建立的程序版本经常没能通过冒烟测试，因为开发人员没对自己写的程序做好适当的单元测试工作。
- 开发人员在得到反馈后没迅速修正错误。

帮开发人员测试标着“容易出错”的例程，以检查与重新设计或重新实作。

第 16 章

软件发行

在各阶段末尾将软件驱策到可发行程度，是驾驭整合失败和不良质量的基本方法。直觉上，要决定软件是否可以发行是件困难的事情，有几个简单的统计技巧可以协助进行这样的决定。发行阶段可能是令人兴奋的时刻，而发行检查清单的使用则可避免在发行阶段内发生问题。

在每个阶段结尾时，项目团队会象征性或实质性的发行软件。无论如何，这对各阶段结束时，软件是否已驱策到一个可推出状态是重要的。缺陷应该减少到让软件能对外公开发行，并完成应该处理的问题，例如使用文件应该与实际建立的软件一致等等。

严肃看待发行工作

将发行工作与下一个阶段的细节设计时期重叠进行通常是个好点子。在不使用设计与程序检查方法的项目中，开发人员在发行阶段为了更正缺陷而手忙脚乱。如果项目在这阶段完成良好的品管，就不会有太多发行相关的工作让开发人员整天忙个不停，而他们会更热切期望开始进行下一阶段的工作。

在这种情形下，将软件修正到可推出状态反而成为次要工作。开发人员们很快开始新领域的工作，进行新设计与新实作会更有生产力与成就感。

在各个阶段结束时，项目团队将驱策软件达到可推出状态当成第一优先的工作，是非常重要的。

阶段性完成做法的成功，依赖将软件推动至一个可推出质量的水准，并做到超出所有必须之外的品管与开发工作。将软件推动至一个可推出状态，必须消除可能累积工作的死角，并提供项目状态的精确可见度。如果在大部分项目团队成员都已经开始下个阶段的工作时，发行仍然继续拖上几个星期或几个月，就会丧失决定项目真正状态的能力。甚至让软件开始慢慢滑向低质量，而且可能永远回不了头。

我审核过一个开发人员本来预计要在阶段中完成软件的项目。当开发人员接近第一阶段末尾时，他们觉得时间不够将软件驱策到可推出状态，就决定直接进行第二阶段的开发工作。当我审核的团队检查项目进度时，已经比预定时程要慢上好几个月了，开发人员几乎被困在延长的测试/除错/修正/测试循环中。他们修好的每个缺陷似乎又产生至少一个新的错误。

这个项目的问题根源在于累积了大量低质量的程序代码。在开发人员加上新程序代码时，他们没有分辨新问题是来自新的程序代码还是来自低质量的旧程序中。那样大大增加了找问题所需的时间，而且使他们更容易出错。最后通过停止新程序代码的开发，再让开发人员多花一个月

的时间全力修正错误，才摆脱了那样的困境。

开发人员在第一阶段结束时决定“没有时间”将软件驱策到可推出状态，他们所做出的代价是最昂贵的。他们大概在做出那个决定时就已经延误时程了，这时更是火上加油。如果那些开发人员按照原定规划将软件在第一阶段结束时驱策到可推出状态，他们的工作量将大大减少。

开发人员可以在目前阶段的发行时期，开始进行下一阶段的细节设计工作，不过在发现前一阶段工作中的缺陷时，他们得先丢开目前的设计工作，去修正先前的错误。

何时发行

要不要发行软件是个危险的问题，答案就在早点发行低质量软件和晚点发行高质量软件之间的夹缝中。在这种情形下，“现在软件好得可以发行了吗？”和“软件何时才好得能够发行？”的问题对公司的生存变得重要了。几个技巧能够将这个抉择以稳健的基础决定，而非只是以直觉判断。

缺陷数量

在最基本程度上，这类的缺陷追踪，能使你适度了解

并发行软件之前还有多少工作要做。你能依顺序得到缺陷数量的摘要报告：“2个重要缺陷，8个严重错误，147个表面缺点”等等。

通过每周新出现的与解决掉的缺陷数，你能得知项目距离完成阶段还有多远。如果每周出现的新缺陷多余当周修正的缺陷，项目大概距离完成还很远。图 16-1是一个“开放缺陷”的例子。公开这张图表会强调控制缺陷是件高优先工作，而有助于让潜在的质量问题在处理控制之中。

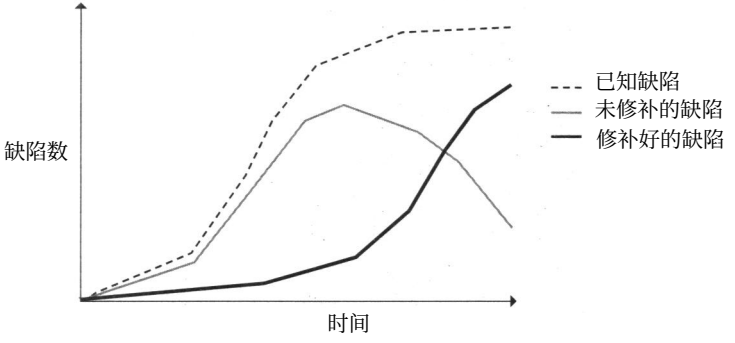


图16-1 “开放缺陷”图

如果项目的质量水准在控制之中，而项目进度稳定，开放缺陷数就应该在项目中期以后开始降低，而且保持低水平。“修正”缺陷曲线跟“开放”缺陷曲线交会处，在心理学上有着重要性，因为这一点说明了缺陷逐渐修正得

比发现速度要快了。如果项目的质量水准失控，而项目蹒跚难行，你就会看到一条逐渐攀升的开放缺陷曲线，表示在加上更多新功能之前，必须要有些步骤用来改善现有设计和程序水准。

个别缺陷增加的工作量统计

依照缺陷种类分类的资料，会提供目前项目跟未来项目中，剩余缺陷修正工作的基础。当你收集这个信息时，在项目中期你也许会说像这样的话，“项目中有230个开放缺陷，开发人员平均三个小时修正一个缺陷，所以项目大概还要花700个小时来修正。”

缺陷发现与修正时期的资料，也给你一个开发过程效率的衡量方法。如果95%的缺陷在产生的同时期修正，项目就可说有个非常有效率的过程。如果95%的缺陷是在产生后的一段时期才发现，项目就还有许多改善空间。

缺陷密度预测

判断程序是否可准备发行的最简单办法，是检查它的缺陷密度，即每行程序代码的缺陷数。假使软件的第一个版本GigaTron 1.0有100 000行程序码，品管小组在软件发

行前找到650项缺陷，软件发行后又被找到 50项新缺陷。那软件在生命期中的缺陷数就是 700，有着每千行程序代码（Kilo Lines of Code，KLOC）七项缺陷的缺陷密度。

假设GigaTron 2.0多了50 000行程序代码，品管小组在发行前找到400项缺陷，而发行后又被找到75项新缺陷。那这次发行的总缺陷密度就是 475项缺陷除以50 000行新程序代码，每千行程序代码有9.5项缺陷。

现在假设你正试着决定GigaTron 3.0是否测试得足够推出了。这个版本里头多了 100 000行新程序代码，而品管小组到目前为止已经找到600项缺陷了，也就是说每千行程序代码有6项缺陷。除非你有好理由认为开发过程已经获得改善了，不然你的经验应该会让你期望出现每千行程序代码有7~10项缺陷的结果。项目团队应该试着找到的缺陷数，依照要求的质量水准而有所不同。如果你想在发行软件以前消弥95%的缺陷，就得在发行前找到650~950项不等的缺陷。这种技巧指出软件还不到能成熟发行的时候。

你拥有的过去项目愈多，对于发行前缺陷密度的目标也会愈有信心。如果你只有来自两个项目的资料，而范围从每千行程序代码7~10项缺陷，那对第三个项目是否会像

第一个或第二个项目的判断，就有很大的摇摆空间了。不过如果你已经追踪过十个项目的缺陷资料，而且发现它们的平均生命期缺陷发生率是每千行程序代码有7.4项缺陷，标准差为0.4，那你手上就有着很好的判断准则了。

缺陷分堆

另一种简单的缺陷预测技巧是将缺陷报告分成两堆，称为A堆和B堆，测试团队在这两堆缺陷中分别追踪，缺陷数可以从不同的缺陷堆中的共同缺陷数估计出。两者差异基本上是不固定的，你可以在项目中期将测试团队拆成两半，分别处理两堆已知缺陷。其实你怎样拆测试团队并不打紧，只要两边都独立运作，而且两边都测试着软件的全部范围即可。一旦两边产生了差异，两个测试团队就分别追踪A堆跟B堆的缺陷数，找出两堆同时出现的相同缺陷数量。任何时刻的缺陷数量为：

$$缺陷数_{Unique} = 缺陷数_{A} + 缺陷数_{B} - 缺陷数_{A\&B}$$

缺陷总数可以用底下这简单的公式估计：

$$缺陷数_{Total} = (缺陷数_{A} * 缺陷数_{B}) / 缺陷数_{A\&B}$$

如果GigaTron 3.9项目在A堆中找到400项缺陷，在B堆中找到350项缺陷，有150项同样的缺陷同时出现在两边，

如图16-2所示，那已经找到的缺陷总数就是 400 + 350 - 150 = 600。大概的缺陷总数会是 (400 * 350) / 150 = 933。这技巧提示还有约333项缺陷等待发掘（约为估计总量的1/3），而项目的品管还有待加强。

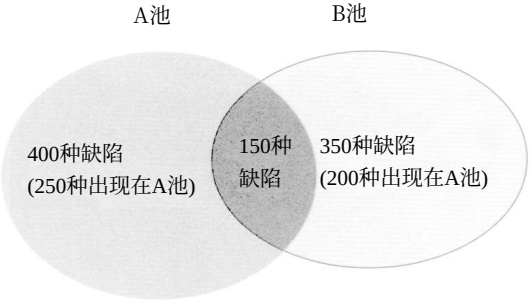


图16-2　缺陷分堆

缺陷分堆技巧让缺陷追查的工作分两边进行，还要找出共同的缺陷，测试小组还要在两边都涵盖到软件的范围来进行测试，明显地增加负担。由于会增添这样的负担，这技巧只适用于那些需要在发行前决定还剩多少缺陷等待发掘的项目。

缺陷播种

如图16-3所示，缺陷播种的灵感来自一种高度发展的统计技巧，在统计总体中抽样，再由取得的样本进行总体

的估计。例如要统计一个池塘中鱼的数目，生物学家会在一些鱼身上标上记号，再把它们放回池子中。然后再从池子中捕获一些样本，比较其中有记号和没记号的比例，以估算出池塘中鱼的总数。

缺陷播种就是故意在程序中放入一些错误，让别的小组去找。被找到的人为错误和加入的人为缺陷总数的比例，和已经找到的缺陷总数，便可计算出跟总体缺陷数量相关的。

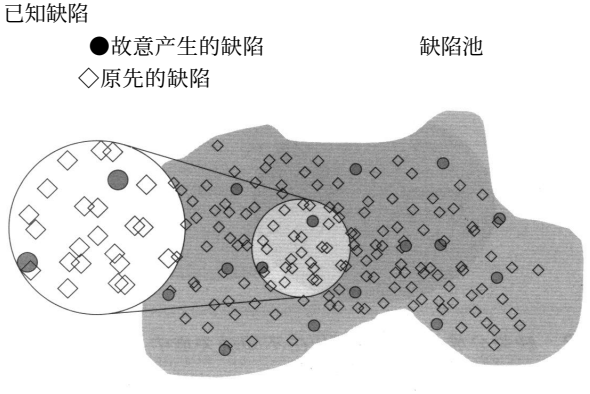


图16-3　缺陷播种

假设GigaTron 3.0中被开发团队故意放入50项错误。在最佳效果下，这些人为错误应该被设计成涵盖软件功能的各个层面，而且包含不同的严重性从随机性错误到表面

性错误都有。

假设当你看过人为缺陷报告后，你感觉测试工作已经快完成了。你发现600个已知缺陷中有31个是人为产生的，那你就可以依据下列公式估计缺陷总数：

$$缺陷数_{Total} = (缺陷数_{SeededDefectsPlanted} / 缺陷数_{SeededDefectsFound}) \times 缺陷数_{FoundSoFar}$$

这种技巧建议GigaTron 3.0总共有约(50 / 31)×600 = 968项缺陷。这表示还有约400项缺陷等待发掘。

要使用缺陷播种的技巧，开发人员必须在测试开始以前就将蓄意制造的错误放到程序中，才能弄清楚测试工作是否有效。如果测试团队使用人工方式，开发人员就应该在测试开始以前将错误制造好。如果测试团队使用自动化测试方式，开发人员可以在任何时候将错误放到程序中，借着自动化测试找到缺陷数量。

使用缺陷播种方式的常见的问题是，忘了将人为加入的错误移除。另一个常见的问题，则是移除设计不良的人为错误反而带来新错误。要避免这些问题，确实将所有人为错误，在进行最后的系统测试和软件发行之前移除。有些项目需要让人为错误维持简化，允许那些只有一行程序代码就修正的人为错误加入程序中。

缺陷模型化

对软件缺陷的处理来说，没坏消息通常就是坏消息。如果项目到了后期还只有少数几项缺陷被发现，人们会倾向认定“我们最后终于把项目弄好，而且建立了一个几乎没错误的程序了！”现实中，“没坏消息”通常是测试工作做得不够，而不是开发团队绩效优异的结果。

有些精密的软件项目估计与控制工具，包含缺陷模型化的功能，可以预测你在每个项目阶段中找到的缺陷数量。通过实际找到的缺陷数和预测的结果，可以估计项目在找寻缺陷的工作上是否赶上进度。

发行决定

如果依照本书的做法，你会得到用来决定软件何时发行的实在信息。

这些信息来自下列各项：

- 程序代码增长统计与图表（参见图5-6）。
- 已经完成的二分性小型完成点的详细清单。
- 缺陷追踪系统未经处理的缺陷列表。
- 累积缺陷统计与图表。
- 各缺陷处理工作量的统计。

◆缺陷密度预测。

◆缺陷分堆技巧。

◆缺陷播种技巧。

◆缺陷模型化。

评估这些准备就绪的指针组合，将比个别估计技巧提供的结果，让你更有信心。单独检视GigaTron 3.0的缺陷密度统计，你应该会预期得到700~1000个生命期中的总缺陷数，而项目团队应该在软件发行前移除650~950项缺陷，以期盼在发行前去除95%错误的目标。如果项目团队找到了600项缺陷，缺陷密度的信息可能会让你宣告软件“快可以发行了”。可是缺陷分堆分析估计GigaTron 3.0总共会产生出约933项缺陷。比较两种方法的结果，你应该预期总缺陷数的范围应该偏向缺陷密度的上限，而非下限。因为缺陷播种技巧也将缺陷总数评估为900多，对GigaTron 3.0来说似乎有证据显示，它应该会出现相对高的总缺陷数，所以项目应该继续进行测试工作。

缺陷追踪与沟通

在项目中公开这一节讨论的状态类型与质量信息，有助于让项目进度上轨道。项目团队应该在如项目休息室、

项目主管办公室门口或项目内部网站的网页上，公开处公布大略的缺陷信息。

发行检查清单

在阶段末尾，即使是最佳项目中，最严重错误出现的原因经常只是由于一时疏忽。人们想把工作做好，觉得工作乍似完成，就想跳过很明显的复杂部分。

在我软件职业生涯的最早期，我是一家对客户提供保险费率报价程序的保险顾问公司的项目经理。以今日标准，当时的那些程序都很简单，只用约三个人力月的工作量就可以写好了，而且大部分只需要更少的开发时间。即使是这么简单的程序，我们还是在软件发行上碰到了最常见的问题——忘了一些我们在将软件发行给客户之前所要做的事情。从中我们得到一个难能可贵的经验，建立了一份发行检查清单，其中项目如下：

这是我以前发行检查清单的一部分

- ◆ 在送出软件以前，复制所需的磁盘份数。
- ◆ 列出所有该收到程序的人跟要寄出去的磁盘数。
- ◆ 在寄给客户的包装上贴好邮票。

如果你看了上面几行，你大概可猜得到我们有时会少

复制程序份数；有时会不晓得客户到底收到了哪一个程序；有一次，我们甚至忘了贴上邮票。更精密的作业需要更精细的发行程序，不过任何发行程序的根本，都是一份发行之前要做好的检查工作清单，而检查清单难免会有许多经常被遗漏掉的工作。简单程序的发行检查清单也许只有几个项目，对于极端复杂的程序，如Windows 95，就可能有超过两百个以上的发行检查清单项目。

表16-1列出一些应该出现在对公众发行的中型软件产品的发行检查清单样本。这份清单并不着重在测试上，到了要发行才担心测试的问题就太慢了。重点是摆在出门以前一些容易被遗漏的项目上。

表16-1　发行检查清单样本

活动	负责人员
工程活动	
以最新的版本信息更新版本字符串	开发人员
从软件中移除除错和测试程序代码	开发人员
从软件中移除人为错误	开发人员

(续)

活动	负责人员
品管活动	
检查目前缺陷清单上的问题是不是都解决了	测试人员
对最后建立的程序版本进行冒烟测试/回归测试	测试人员
从CD ROM上将程序安装到没安装过程序的机器上	测试人员
从磁盘上将程序安装到没安装过程序的机器上	测试人员
从网站上将程序安装到没安装过程序的机器上	测试人员
在安装过旧版本程序的机器从CD ROM/磁盘上安装（升级安装）	测试人员
检验安装程序是否建立正确的Windows系统登陆项目	测试人员
检验第一次安装程序的机器反安装的情形	测试人员

(续)

活动	负责人员
发行活动	
固定要送出去的档案清单	发行团队
与所有发行档案的时间日期同步	发行团队
准备完成程序磁盘（“ 金碟 ”）	发行团队
检验所有档案都在金碟上	发行团队
病毒扫描所有发行媒体	发行团队
表面扫描发行磁盘正本的坏扇区	发行团队
建立开发环境的备份，	发行团队
将开发环境纳入更动管制	
文件说明活动	
检验金碟上的自述文件版本	文件说明人员
检验金碟上的辅助说明	文件说明人员
档案版本	
其他活动	
检验版权、授权及其他	产品主管，法律顾问
法律文件	

如果项目团队是对内部使用者发行软件，检查清单内容会不一样，可是构想是相同的：检查清单应该列举在匆

忙发行新系统时，是不该被忘掉的重要工作。

项目团队应该将过度发行版本的检查清单，跟完成版本的结合在一起。这些清单上的项目当然不同，可是会有许多共同点。

发行认可文件

除了发行检查清单，大部分机构会让所有相关人员签名同意软件的发行。如果要对公众发行软件，重要的就是在发行之前有个检查，避免不良软件流入市场。顶尖机构要求软件在能够发行之前获得品管认可。品管有时会被来自开发背景的项目主持人视为次要活动，可是品管是避免发行令人困窘而又昂贵的沉重技术的万灵丹，可避免让组织面对法律问题的更糟状况的防线。在软件发行前要求品管认可，我认为是适合的，而且不要在对软件适合推出有信心以前强迫品管人员认可发行。

有些人担心品管人员会把持软件，直到所有臭虫都被修正为止，而这就是我在第9章中提到过的，此时是检测发行标准派上用场的时候了。当项目有可检测的发行标准时，如“ 95%的最严重问题和次严重的问题都修正时 ”——品管人员的工作，只是回报软件是否满足每个人稍早

都同意的发行标准。

除了品管小组，也经常需要其他资助者来批准新软件的发行。表16-2列出一份代表性的资助者清单。

表16-2　发行认可文件样本

识别名称	
程序名称	_____
程序版本	_____
项目代号	_____
签名同意	
我愿意担保软件可以发行了：	
子系统1的主任工程师	_____
子系统2的主任工程师	_____
…	…
子系统n的主任工程师	_____
工程主管	_____
产品主管	_____
品管主管	_____
文件说明主管	_____
国际工程主管	_____

(续)

使用者支持主管	_____
行销主管	_____
法律顾问	_____

使用发行检查清单和发行认可文件不能保证软件的完美发行，可是不使用这些文件则几乎导致发行失败。发行检查清单易于使用，易于更新，而且可以区别出强势发行和最后标上“退货”的软件之间的差别。

在本书网站上有一份目前市面上的软件项目估计与控制工具清单。

求生检查

- 项目团队将阶段末推出时期视为第一优先。
- 用统计技巧来协助发行决策。
  - 只使用一种统计技巧。
- 项目团队利用发行检查清单来避免软件发行阶段的疏忽。
- 项目团队使用发行认可文件来保证各项目资助者同意软件可以发行了。

第 17 章

阶段完成记录

每个阶段末尾提供了修正方向与获得经验的机会。在项目进展时愈来愈精确的预估，可作为下一阶段规划的基础。到目前为止得到的项目经验，应该保存在软件项目记录中，好让未来的项目也能从其中获益。

阶段性完成规划可分两种方向进行。可以将阶段划分成决定性的，主要提供软件最重要功能，可最先完成。阶段的内容也可以在每个阶段末尾重新订定，先完成最重要的功能，但是在各阶段末尾，依然提供改变项目执行方向的机会。

不管何种做法，在各阶段结束时，都提供了学习阶段中获得的经验，并继续应用的机会。而且能更快速得到经验与专业知识的方法。

总结更动会议

在本书建议的阶段性完成做法中，项目可以只在两个阶段间的空档，考虑变更功能来简化更动评估的工作。项目应该持续进行更正错误的工作，不过项目主管（在更动会议成员之外）不应该拦阻功能变更提议。如果更动次数不受限制，项目成员会发现自己的更动评估结果会每隔几天就产生重大影响，而累积成重大的困扰。

可试着将功能更动评估延到项目结束再进行，也由此把提案分类而作了更好的更动顺序安排。如果更动会议每周评估一次更动提案，可能就会通过某些不重要的更动，因为那周可能没有其他提案出现，而批准半天额

外的工作，还可能造成某些伤害。甚至造成严重的时间脱序，特别是当那些原来估计半天就可以做好的事情，却花费了一整天的时间。更动分类使得判断更动顺序更容易些，况且，有时两个看来大得难以个别进行的相关更动合并，就容易多了。

在发行程序版本后，是个考虑更动提案的好时机。整个项目团队在完成软件发行以前，通常都卖力工作着，在那之后再来花一两天时间，考虑更动提案会是个比较适宜的时机。

估计修正

阶段末尾也是检查项目时程进展、与规划时程差距、修订项目预估的好时机。

重新检查开头用来产生估计的那些不同因素。项目规模是否还是跟本来预估的差不多？如果不是，就应该依据对项目大小的进一步了解来重新调整。功能项目变更了吗？软件表现、稳定度或是适于完成的水准改变了吗？开发团队是否如预期的具有生产力？估计中是否省略了必要的工作？这些都是估计修订时应该考虑进去的因素。

错的不是低估项目规模或是高估项目团队生产力，错的是没发现估计错误而且没更正错误的估计。

重新估计生产力

假使项目本来规划在六个月内完成，让第一阶段在八个星期内完成。如果第一阶段实际上花了十个星期才完成，开发团队该如何更正对项目剩下部分的估计？

- 假设时程后面可以弥补失去的两个星期，让原来六个月的总估计维持不变？
- 在总时程中加上两个星期，建立一份六个半月的时程规划？
- 将整个开发时间乘上延误的时间比例，在这例子中是乘上25%，把整个时程重新估计成七个半月？

第一个选项是最常见的做法。通常忽略而花费掉额外时间的工作不多，项目成员会发誓不让同样的疏忽出现在别次估计中。人们通常会说服自己说项目的开头部分比预期的花费要更久，因为该阶段中要熟悉的东西比他们预期

的多。他们说服自己，已经在该阶段中学到够多教训了，所以下个阶段只会进行得更快些。

不要陷入这种推理之中！软件开发本质上是个勘爬陡峭解决问题的学习曲线的活动，而学习曲线本身并不会在第一阶段结束后就消失。

1991年对超过300个项目的调查发现，项目很难追回失去的时间，只会更加延误。

第二个选择看来似乎合理些，不过估计错误通常是肇因于遍及全项目时程内的系统因素，如对采用完美假设感到压力，或是匆忙进行项目估计而忽略了许多小动作。软件组织的实际经验可能比项目初期的估计要更精确。项目团队可以在错误的生产力假设以外的因素中，寻求时间延误的原因，不过在单一项目中，这些因素通常是难以修正的。

除了少数例外，最精确的修订时程方式应该是第三个选项。那个选项将项目的实际经验衡量的比初期做出的估计更精确，而且依据实际表现而非预估表现。

“重新估计”或“延误”

在某些组织中，重新估计被误认为是一种“延误”，其实它是对于建立中的软件在详尽了解后不得不作的改进。

“重新估计”跟“延误”两个词有不同的涵义。重新估计是在项目初期对于软件项目精确估计后，重新了解所进行的计算和规划事件。重新估计在项目规划中会更进一步订定出进行的阶段。定期重新估计可表示出开发团队的能干。

延误则是没能满足目标，表示管理团队没充分照表操作，或者开发团队没有完成托付任务，以致发生不可预测的迟滞，赶不上主要完成点的情形。如果资助者不了解从项目初期重新估计规划，会认为重新估计就是发生延误了。

确实让所有项目资助者从第一天起了解，一个健全成功的项目开发团队会以预先决定的步调，在项目进行中让预估结果更精确。

对项目规划的评估表现

项目团队在阶段中有依照项目规划中要求的程序进行

吗？他们有进行技术检查，依照整合程序进行，观察变更管制计划，维护时间管理资料，并在其他方面依照规划方式进行吗？原因为何，为何没有？是因为规划方式无法完成，还是团队本身对于阶段中所该处理的事情不够努力？

一个最常见的软件项目管理问题就是没有真正想照着项目规划去做，放弃项目规划。

本书的做法有效消除了许多开发团队放弃项目规划的理由：

- 规划是公开的。在某些项目中，团队没照着规划方式进行，只是因为他们找不到整个规划内容。
- 规划是可靠的。开发团队参与了规划建立，就应该相信这些规划是实在而可行的。
- 规划是有人情味的。规划代表着将项目方向以少量的加班时间进行安排的真诚尝试。这不是驱赶团队进行没薪水可拿的加班工作的伎俩。
- 规划一直在更新。规划不能跟项目的现实脱节太远，以至于让规划变得不切实际。

忠于规划。如果规划不可能照着作，就重新规划。让项目失控的一个确切办法就是放弃本来的规划却不重新规划。

项目文件保存

在各阶段结束时，整个用来建立软件的环境应该被记录起来。组织常需要重新建立软件的旧版本，如果旧版本的项目组件已经找不到了，这可就糟了。项目应该至少依照下列项目保存起来：

- 发行包装的副制品，包括CD或磁盘片，印好的使用文件，附带套件和包装方式。
- 源代码。
- 随软件发行的数据库或数据文件。
- 用来建立软件的资料文件。
- 项目中使用的自行开发工具。
- 商用链接库。
- 图形资源、视讯资源、声音资源与其他包含在软件内的资源。
- 编译器、连结器、资源编译器和其他用来建立软件的工具。

◆辅助说明档案和其他文件的原始文本文件。

◆程序编译建立描述档案。

◆文字描述。

◆测试资料。

这些项目的一份复制品应该跟项目记录一起保存，而其他复制品则存放在安全的保管设备中。

工作文件、电子表格、项目规划和其他由变更管制处理的项目也应该和编辑用的文书处理程序、电子表格程序、项目规划软件和其他用来建立它们的软件一起记录归档。

阶段完成记录不是定期备份的替代方式。如果你在开发软件，不经常备份档案就好像经营便利商店，却将现金都放在柜台一样，你辛勤努力的结果在发生意外或遭到蓄意破坏时，都显得不堪一击。

更新项目记录

有效率的组织会在项目进行时收集项目相关资料，使他们能够获得有助益的资料。在项目进行的各阶段结尾收集这些信息，回顾该项目的人就可以从项目进展中，在最后时程表现与预算高低之外，得到一个动态的开发过程轮廓。

项目记录保存在软件项目记录中，项目主管应该在各阶段末尾说明项目状态，更新项目记录。表17-1说明软件项目记录的大纲。

表17-1　软件项目记录内容

软件项目记录应该包含下列关于各阶段结尾的活动与结尾时的项目状态信息
项目目前的时间与工作估计
在阶段中由更动会议认可的时程与工作调整
阶段进行中的重大日期、背景与决策
各阶段主要工作完成的规划日期与实际日期
阶段中进行的技术检查结果（通过与否的状态跟缺陷分析）
时间管理资料
程序代码
错误数量
更动提案提出和接受件数

当表17-1中列举的各数据项收集好，软件项目记录更新好，就应该在更动管制系统中更新好这些东西。这份文件，如同其他项目规划文件，应该在项目中公开取得。

更新软件项目记录需要的时间少得惊人（如果项目追

踪资料是最新的，就只需要一两个小时），而且可以在对未来的项目规划的基础上得到重大益处——这包括对目前项目未来各阶段在内。

求生检查

- 项目团队将功能变更要求延到阶段末尾。
  - 项目也将缺陷更正延到阶段结束。
- 项目团队在各阶段结尾重新估计工作、时间与成本。
  - 重新估计时假设估计错误的情形只是单纯增加项目时间，而非将整个时程数全面乘上一个数字。
  - 重新估计被认为是“延误”而非重新“估计”，因为资助者在项目开始时没了解重新估计的性质。
- 在各阶段结束时，项目主管在软件项目记录中纪录项目状态。

Part IV

MISSION ACCOMPLISHED

第四篇

完成任务

第 18 章

项目历程

软件项目历程中包含的信息，对于将来的项目是有用的。软件项目历程使用来自软件项目记录的资料，这些资料在各阶段结束时进行更新，并从中提取一般教训。在执行良好的项目中，项目历程所需的大部分资料都会慢慢准备好，并让项目历程易于建立。

根据项目长度与规模，推出软件后，可以招待项目团队好好吃顿晚餐，放他们半天假，或是送他们去夏威夷好好渡个假。不管项目是成功得令人手舞足蹈还是失败得提不起劲来，那也是从项目中获取了经验，为将来的成功奠定基础。

汇集项目资料

一些项目在结束时，会举行一次项目检查会议，其他项目则收集项目成员认为有哪些做到了跟没做到的电子邮件。不管汇集资料的方式如何，重要的是将项目成员的意见感想汇集起来，而且最好在项目完成后就收集那些信息。最佳情形下，在发行软件的 15~30天内收集好项目成员对项目的感想。如果更久之后才进行，项目成员会开始忘记重要的想法，让项目更难以组合或重组。

项目检查会议

项目检查会议是项目成员不客气的讨论想法的宝贵时刻，而且那些想法对组织本身会有重大的好处。

如果项目团队举行项目检查会议，应该确定有个客观

的会议主持。有时项目检查会议会变成牢骚会议。这些牢骚都是还没发表过的，项目成员会想讨论这些东西。一名客观的会议主持会确保各个重要议题都被讨论到，并避免让会议陷入单一问题的泥淖中。

项目检查问卷

有些项目有使用三段式项目检查问卷的良好习惯。问卷的第一部分包括不同项目属性的评分。“ 你对项目在更动需求上的管制评价如何？等级从1~5，1表示太严格了，5表示太宽松了，3代表刚刚好 ”。这让项目成员对项目的评估结果可以从问卷调查中收集出来。

第二部分则包括对可能需要改进的特定领域的指定问题。这部分可能包含如下的具体问题，“ 我们在这项目中第一次使用阶段性完成的做法。在你的意见中，这么做对于达成我们项目的目标有何帮助？这种做法还有哪些可以改进之处 ”？这部分的问题可以在项目检查会议时拿出来直接讨论。第三部分是自由意见的发表，可以用来征求不受限制的意见。除了这方式提供的结构，这种问题也可以用来征求匿名意见，在某些环境中可能比面对面询问更能获得坦率的意见。

软件项目历程文件

项目历程在称为软件项目历程的文件中定形。一份好的软件项目历程收集包括目标跟项目中发生的定量信息，以及团队成员的意见，做好了哪些事情和哪些事情没做好的感想。

软件项目记录中的定量信息跟项目检查中的意见想法将构成软件项目历程的文件内容，如表18-1中勾绘出来的轮廓。

表18-1　软件项目历程的内容

项目介绍
说明软件用途、顾客、前景叙述、详细目标跟其他一般信息
历程概览
对各个阶段说明工作成果、完成点、面对的主要风险、日程规划、人事层次与其他项目规划信息
说明下列阶段：
使用者接口雏形化跟需求信息汇集
构架设计
质量保证规划
一般阶段规划

(续)

从最初阶段到第n阶段的细节设计到推出的活动（包括细节设计、构建过程、系统测试与阶段发行）

最后推出软件

项目资料

说明项目中使用的组织结构，包括执行赞助者，项目参与者名单，他们的角色，以及在项目过程中参与的层级

软件项目历程也应该包含下列关于项目的确实资料：

软件推出时实际花费时间和工时

软件推出时的时间管理资料

软件推出时的子系统数

软件推出时的源代码行数

软件推出时的重复使用程序代码行数

媒体量（声音、图形、影像等等）

软件推出时的缺陷数量

软件推出时，提出和接受过的更动提案数

估计日程与实际日程的比较图

估计工作量与实际工作量的比较图

项目程序代码每周增长曲线图

项目缺陷产生与修正数量周计图

获得的教训

(续)

说明项目中获得的教训
规划。规划的方式有用吗？项目团队有照着规划执行吗？项目人手的质量够吗？每个部门的人手充足吗
需求。需求完整吗？这些需求稳定或经常变更吗？这些需求易于了解或容易被误解吗
开发过程。设计、实作跟单元测试做得怎样？每日整建有没有用？软件整合有效吗？推出的软件版本运作如何
测试。测试规划、测试项目开发跟冒烟测试开发做得怎样？自动化测试有用吗
新技术。新技术对于成本、时间与质量的影响如何？主管和开发人员对于这些影响的看法一致吗

一旦表18-1的这些项目都收集好，软件项目历程内容组合好了，就应该保存下来留作将来参考之用。

准备未来项目使用的项目历程结果

项目历程有个一再发生的问题，在于这些文件在完成存放后，就被遗忘了。要从项目历程中获得最大益处，将项目历程分析的结果以最少两种方式包装起来：

◆替未来项目建立一份规划检查清单。如果项目团

队已经有份规划检查清单，应该将项目历程报告中提到的主要问题解决方式，加到现有清单中。检查清单中应该同时包含该做和不该做的项目。

- 将项目中提到的主要风险列入一份十大风险清单样板中，让下个项目可以利用这模板作为列出初始风险清单的基础。

将项目历程的分析转成易于使用的格式提高了开发团队用于建立项目历程的时间与工作价值，替下个项目奠定更为成功的基础。

散发软件项目历程的复制文件

完成的软件项目历程对每个团队成员的个人回忆提供有用的补充。一份装订印好的项目历程文件提供了成就感，每个项目成员都应该收到一份项目历程文件。

求生检查

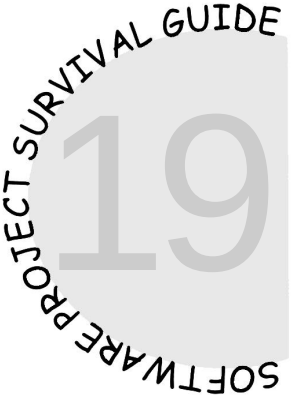
建立一份软件项目历程报告，其中包含项

目的目标与意见概述信息。

- 软件项目历程没在项目完成的15~30天内做好。
- 将软件项目历程发给每个项目成员。
- 项目历程结果包装成项目规划检查清单和下个项目使用的初始风险清单。

第 19 章

求生小技



本书的关键指导方针，综合自世界最有效率的软件开发组织NASA软件工程实验室的指导原则。本章结尾将提供进阶阅读与其他参考资源。

本章概述软件项目成功所需要的事项，将本书的主旨提炼成几页精华。第一节概观NASA内一个软件开发组织使用的方法。第二节则说明你应该考虑纳入自己的软件求生工具中的资源。

NASA的成功检查清单

位于NASA高达德太空飞行中心的软件工程实验室（Software Engineering Laboratory，缩写成SEL)在软件开发实务界居于顶尖地位将近20年，该实验室是世界上最具竞争力、也最成功的软件开发组织。在1994年，该实验室获得不凡生产力与软件质量的认可，SEL成为第一个赢得美国电气与电子工程师协会 IEEE颁发的软件程序成就奖的组织。

如果你认为那奖项的获得是由于NASA的软件必须超级可靠，NASA的教训可能就不适用于你的组织了，好好再想想看。SEL依循的做法几乎和任何软件开发机构的做法相同，而且应该遵循的。这些做法使SEL达到和一般信息系统项目差不多的生产力，质量水准却超出 10~20倍。换个不同的方式说，一般信息系统需要 14个月和110个人力月数来完成一个 10万行程序代码的信息系统， NASA

SEL要完成同样规模的系统所需要的时间跟人力也差不多，可是这样的系统中大约只有50个错误。

SEL出版的《Recommended Approach to Software Development》一书将SEL 20年里得到的教训，萃取成软件项目要成功所需的九件应做和八件不应做的事。这些事项就在此提出。

NASA SEL提出软件项目要成功所应做的事项

一个成功的项目所应做到如下的九件事：

1. 建立并遵循一套软件开发规划

在项目初期，准备一套软件开发规划，说明项目前景，确定团队结构，定义开发方法。这套规划应该包含估计主要完成点及其他用来追踪进度的衡量方法。这套软件开发规划应该有在各主要阶段进行更新的活动文件。

2. 授权项目人员

以项目前景指派开发团队成员，确定项目人手安排，提供成员一个高生产力的工作环境，清楚分配任务与完成任务所需的权力。

3. 简化官僚体系

建立需要的最小程序负担，以满足项目目标。有良好

理由才需要开会与进行纸上作业。如 NASA所言，“ 开更多会、写更多说明、加上更多管理并不等于会更成功 ”。

4. 定义需求底线，管理需求变更

尽早定下需求。纪录一份可能改变需求或未定义需求清单，估计各项需求成本与时间影响，以安排需求顺序。试着在构架阶段解决这些项目，或至少在细节设计阶段解决这些问题。

5. 采取阶段性评估项目体制与进度，视需要重新规划方向

经常将项目进度与项目规划与过去相似的项目进行比较。如果项目进度明显与项目规划脱节，重新进行规划。在重新规划时小心考虑降低工作规模，认真分析不切实际的美化预估状况。

6. 定期重新评估系统规模、工作量与时间表

项目的每个新阶段提供软件建立的新信息。不要坚持维持原始估计，在每个主要完成点完成时试着改善估计结果。估计是一门不精确的学问，错的并不是开发团队低估了项目规模或高估自己的生产力，而是没规划定期检查估计的精确度及修正项目进度的估计。

7. 确定并管理阶段变化

有些项目在从需求开发转换到构架过程、构架过程到阶段规划，从一个阶段结束到下个阶段开始的中间浪费了时间。项目团队应该在完成目前阶段之前几个星期就开始考虑下个阶段的工作，好让团队能够做个有效率的转换。

8. 培养团队精神

即使项目中包含不同机构或公司的人员，都该强调项目工作人员朝向的一致目标。清楚定义每个人的责任，强调整个项目在个人责任内的部分。确实以同样方式沟通项目状态、项目风险与其他项目管理问题。

个人成功无损团队，个人失败无益团队。

9. 以少数资深人员开始进行项目

以一小群能够在项目中带头的资历深厚老手开始进行项目。让他们确实建立一个目标，确定软件概念，发展一个完成项目的方式，让他们在其他新手加入开发行列之前大致统一步伐。

NASA SEL提出软件项目要成功所不该做的事项

成功的项目所不该做的如下八件事：

1. 不要让团队成员以非系统化的方式工作

高质量软件的有效开发,不是感情化的不可管理过程。那是个创意的过程，而且获益自己制订原则、做法、方法与技术的合理应用。坚持团队采用系统化的开发方式。

2. 不要确定不合理的目标

确定不合理的目标比完全不订目标要糟糕。如果团队不相信目标能够达到，团队成员就不会投入工作，只会上班打卡，然后等到下班时间。如果他们随便做做，他们会在上游阶段酿成下游阶段必须花费昂贵代价才能弥补的错误。设定合理、适度挑战性的目标，团队就会全力赶上目标而不会危害到项目效率。

3. 不要还没衡量影响及获得更动会议的认可就做出更动

估计每项更动的影响更重要，项目可以承受的小更动不用重新安排时程。项目宏观上跟微观上都需要一个更动历程记录。即使在特定更动不会造成重大影响时，未经管制的小更动加总起来最后也会造成追加成本与时间延误。

4. 不要花俏的功能

只要按照项目的要求去做，开发人员、主管与顾客不

要以为小而简单的变更会使软件更好，这些更动常会造成长远的影响。不要让多余的复杂性以花俏的面貌出现在项目中。

5. 不要人浮于事，特别是项目初期

以小型资深团队开始进行项目。只有在工作需要处理时，才加入人手。这条准则是允许在初期使用新手。在需求开发与初期构架时期，资历相对浅薄的人员可以检讨文件，研究工具和链接库功能，处理许多其他需要良好技能而还用不着高手出面的工作。

6. 不要假设阶段中期的时间延误可以在后头弥补过来

一个常见错误是假设生产力在项目从阶段开始到末尾的过程中会提升。生产力也许会有些提升，不过在任何阶段中都没有足够时间能把失去的时间追回来。一般情况，不要假设时间的延误可以在项目后头的任何时候弥补回来。如果项目没在延误的情形出现时马上弥补回来，你大可放心，但延误的时间一定赶不上本来的时程了。

7. 不要为降低而降低标准成本或缩短时间

降低标准容易造成项目中的错误，而且理想的项目成本跟开发时间都取决于错误的消除。降低标准也容易引起消极效果，大部分开发人员都非常重视质量，标准的降低

等于告诉他们说顾客或上层主管不在乎质量。

8. 不要假设有大量文件说明就保证成功

不同项目需要不同类型的文件支持。依照项目规模、时间与系统的预期生命期来决定需要文件的数量与种类。避免出现那种25 000行程序代码的程序就要5 000~10 000页文字，而100 000行程序代码的程序就要多到40 000页文字的美国国防部型文件说明。

其他求生资源

该书是本软件求生指南。你也许会把它当成急救工具放在背包或货车里。就如同医院比你的急救箱的医疗器材更完备，其他软件开发资源的世界也比这本小小的求生指南为你提供更多的软件求生工具。

参 考 书

这里有一些内容非常好并提供实务建议的书籍。

Recommended Approach to Software Development, Revision 3, Document SEL-81-305, Greenbelt, Maryland: NASA Goddard Space Flight Center, NASA, 1992. 软件工程实验室的Recommended Approach大概是我看过执行软件项目最实际的概览书籍。它是针对飞行动力项目而写的，可是许多建议可以更广泛地应用开来。本书描述每个项目阶段的出入标准，给定各阶段中开发团队、开发团队与其他主要团队活动的切实摘要说明。这本书令人振奋的主旨，在于软件项目并不是神秘的、难以驾驭的，虽然它们极端复杂，还是可以通过努力应用前一个项目中得到的教训来加以驾驭。你可以写信到Software Engineering Branch, Code 552, Goddard Space Flight Center, Greenbelt, Maryland 20771免费索取本书。你也可以从SEL在http://fdd.gsfc.nasa.gov/seltext.html的网站下索取本书的文件档案。

Manager's Handbook for Software Development, Revision 1, Document SEL-84-101, Greenbelt, Maryland: NASA Goddard Space Flight Center, NASA, 1990. 这本SEL

的Manager's Handbook与Recommended Approach一样管用。这一本内容比较短，其方针特别着重于软件项目管理的事务上。你可以通过通过与 Recommended Approach一样的方式取得本书。

Fergus O'Connell. How to Run Successful Projects II: The Silver Bullet. London: Prentice Hall International (UK) Limited, 1996. 该书提供了对软件项目所需事项的完善而完整地检查。它涵盖的范围与本书相同，而且内容与本书相当搭配，很少有重复的地方。鉴于本书在提供一个项目技术构架的完整轮廓， O'Connell 的书着重于一名项目主管必须处理的许多具体工作。这本书包含许多范例项目文件与规划项目。如果你看完这本书后，问过自己："这些看起来都很好，可是我该怎么做"？那 O'Connell 的书对你来说就是个正确的选择。

Tom Gilb. Principles of Software Engineering Management. Wokingham, England: Addison-Wesley, 1988.Gilb的书通过对软件项目采用定量的风险导向做法，将"工程"带入软件工程的管理。这是一本国际知名的软件项目管理顾问的经验总结。这并不是一本教导关于软件项目成功的常见想法的书籍，这本书中经常采用作者的做

法。不过我认为本书中的许多地方不依照常见的想法，是因为Gilb的观念是对的，而常见的想法是错的。

Lawrence H. Putnam and Ware Myers. Industrial Strength Software: Effective Management Using Measurement, Washington: IEEE Computer Society Press, 1997. Putnam与Myers写过一本描述如何使用软件尺度来管理各种软件开发工作的参考书籍。对照 Gilb那一本书，Putnam与Myers重点不在具体项目上而在编制功能上。书中第21章“ Shining Shadow Loses its Luster ”是块瑰宝，戏剧性地描述定量做法对软件项目管理的威力。

Tom DeMarco and Timothy Lister. Peopleware: Productive Projects and Teams. New York: Dorset House, 1987. Peopleware将程序设计对人们来说是最先完成的工作，而计算机只是次要的这件事情彻底讲明。这一本读起来挺有趣，提供的是关于那些成功与失败的软件团队的故事，值得记取教训。

Alan M. Davis. 201 Principles of Software Development, New York: McGraw-Hill, 1995.201 Principles 提供一个对软件开发的关键问题容易阅读的介绍。其他书籍讨论发生在项目中的关键问题，Davis的书让你认清这些问题。

Steve McConnell. Rapid Development. Redmond, WA: Microsoft Press, 1996.如果你喜欢我这本书，你大概也会喜欢我在Rapid Development一书中必须要提的东西。这一本书中包含一般化，而又是常见问题、风险管理、生命周期规划、日程安排、行动方式、团队合作与其他与快速软件开发相关的话题讨论。

网络资源

我的公司在网址 http://www.construx.com/ survivalguide/ 的地方设了一个关于这本书的网站。其中包含本书建议的许多样本文件的例子，如软件开发规划，使用者接口风格说明，评估程序，发行完毕表格以及其他许多文件。这里也提供其他软件开发资源网站的链结，如目前源代码管制工具，时间安排工具，估计软件与错误追踪工具的链结。这里也提供本书求生检查的电子文件案与第 2章软件项目求生测试的电子表格版本。

尾　声

碰，碰，碰…

软件开发人员、各级主管与顾客们几十年来不停地敲着头面对同样的问题。许多中型项目由于不合理的原因而失败，那些项目并没有用到半点先进的技术，也没有采用其他学科的顶尖研究成果。那些项目只是被自己的笨重给压垮了。

软件项目的存活不是一种意外的结果。要让一个项目成功所需的努力并非特别困难或耗时，只是需要从项目开始进行的第一天就勤奋努力到最后一天而已。

软件开发的方式已经进步到没几个中型软件项目的失败是应该的了。当开发人员、各级主管跟顾客停下足够长的时间，不再拿头去撞击同样的老问题，让他们自己吸收本书中描述的各类软件项目求生技能后，他们的项目就会成功了。