第 9 章

生命周期管理

软件开发管理是很难的，项目常常被推迟交付，甚至还经常被取消。现代软件管理已经创建了大量降低风险的方法，最常用的、已被证明是有效的方法是迭代开发。有许多使用迭代方法的理论，它们通常被称为敏捷方法（Agile methodologies）。

本章不是完整的软件管理指南，因为这个主题需要整整一本书的篇幅（可以参考由 Addison-Wesley 出版的 *Agile and Iterative Development: A Manager's Guide* 一书。）

本章将给出一些关于如何基于迭代管理软件生命周期，以及如何使用少数工具来完成这一工作的技巧和总结。

9.1 不同的方法

在介绍迭代生命周期之前，先介绍几种软件业现有的开发模型。

9.1.1 瀑布开发模型

瀑布开发模型将软件视为一个整体，每个阶段都将在前一个阶段完成以后才开始。换句话说，所有工作将被组织为一系列阶段，通常包括：

- 需求分析；
- 总体架构及各软件部件组织形式的设计；
- 设计每个部件；
- 使用 TDD（测试驱动开发）方法编码每个部件（有些人不使用 TDD 方法）；
- 整合各部件进行整体系统测试；

- 部署。

因此，整个工作的组织看上去像一个瀑布，如图 9.1 所示。

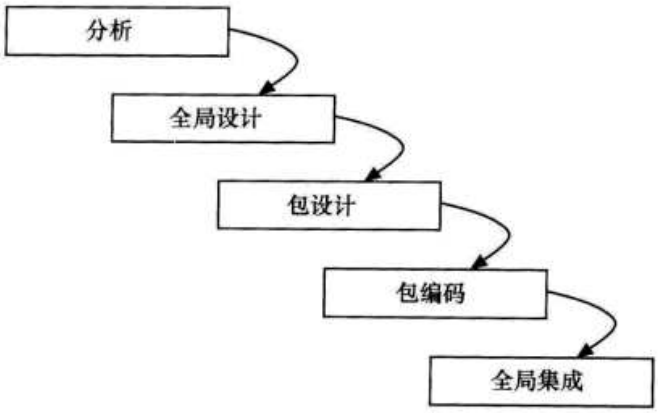

图 9.1

许多大公司都在使用这个模型，认真完成每个步骤并且进行评审，然后才开始下一步。这意味着设计完成之后，开发人员只需要实现这个设计。最后一步是将所有部件整合起来并进行总体测试。

这个模型很死板，因为在整体测试之前几乎不可能考虑到软件的所有方面。实际上，在最后一步常常会发现一些不一致或部件缺失，甚至因为设计缺陷而产生性能问题。

这样一个模型可能更适用于非常明确的目标环境以及一个有经验的团队，但对于大部分软件而言，这是不可能的。

9.1.2 螺旋开发模型

螺旋模型是基于原型之上的反馈。如图 9.2 所示，程序的第一个版本是根据原始需求创建的，不做任何修正。然后，这个原型根据接收到的反馈进行细化，程序的弱点和优点都能够精确地定位，这样它就可以被重构。

在几个周期之后，当所有人都认为原型符合需求时，就对其进行最后的细化并交付。

这个模型大大减少了风险，因为开发人员可以很早就开始软件编码，而不用像瀑布模型那样要先进行复杂的设计。在第一个周期之后，管理人员就能够对项目所需时间有概要性的了解。

9.1.3 迭代开发模型

迭代模型与螺旋模型相似，但它不把应用程序视为一个整体，而是关注于为系统增添一些新的功能，并且通过反馈来对原有的软件进行再造。

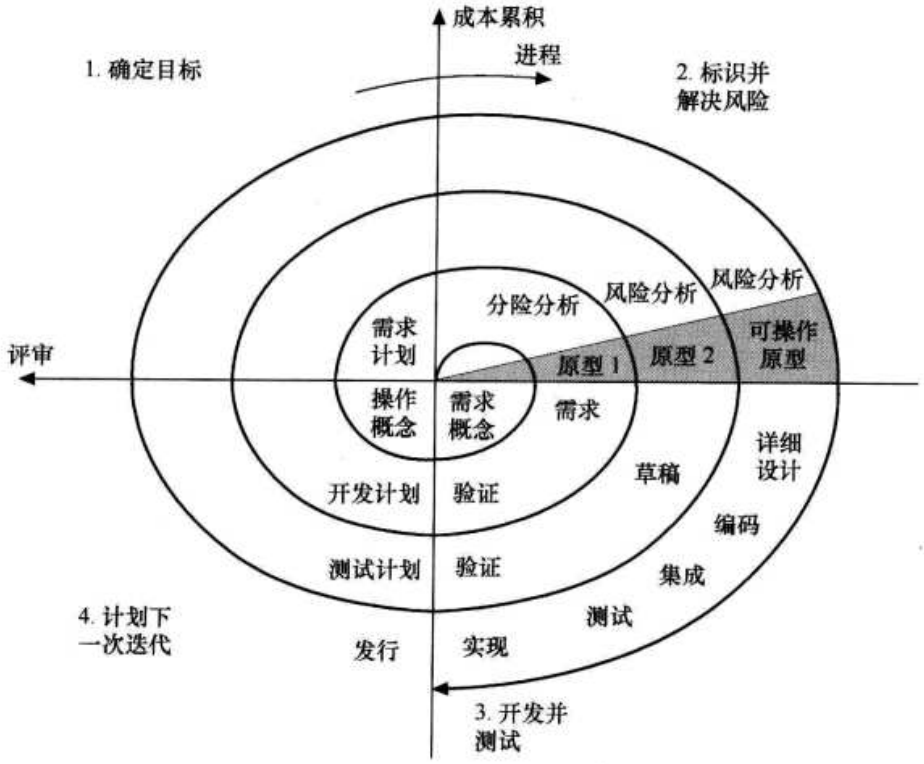

图 9.2

这意味着它和螺旋模型、瀑布模型都不同，在整个项目周期内都将存在分析、设计和编码工作，因为它们在每个周期中只关心功能的一个子集。

因此，一个项目被分成多个可以作为独立项目的迭代，在每个迭代中设计、构建和分发软件。所有涉及的团队，将通过跨学科的工作聚焦于当前迭代的功能。

在每个迭代中提供反馈，对于所有涉及的人员顺利地协同工作是很有帮助的。可以对每个功能在完成之前做很大的改进，往往与开始时的想法不一样。每个人都能从中获得一些思路，并且可以在下一个迭代中修正分析和设计。

因为这种渐进式方法关注于系统中更小的部件，所以提高了反馈的频度。一个迭代往往历时 1～4 周，软件在交付之前需要进行多个迭代。因为整个软件在每个迭代之后都将有所增长，所以它是增量式构建起来的。

基于迭代方法的理论很多，比如 Scrum 或 XP，这是敏捷方法的共同基础。

然而，迭代方法不是懒于设计的借口，它是有代价的。为了保持灵活性，敏捷的编程人员非常强调测试。通过编写测试，他们确保不会在修改时破坏原有的代码。

迭代方法中的经验法则是为每个迭代定义一个固定的长度，并且在结束时建立某些特殊的东西。

本章将提出一个作为许多开源项目共同基础的通用模型，并将通过一组可用于这一目标的工具来描述它。

9.2 定义生命周期

为一个项目定义生命周期，主要包括计划常规发行和保持进度两部分工作。在周期结束时，推迟实现某些未就绪的元素，以便跟上承诺的计划进度往往更好。这被称为火车方法：当火车离开的时候，要么上车，要么呆在那里。但是，有些项目会推迟发行日期而不删除某些功能，而这种方法往往更难预测。

项目开始之后，通过一个初始化阶段来定义整体计划。通常的工作是大家参加一个启动会议，共同定义软件交付时间，然后将剩余的时间分到各个迭代之中。

根据软件的特性，这些迭代可以在一至四五周之内完成。每次迭代应该具有相同的长度，以便在整个项目中保持相同的节奏，如图 9.3 所示。

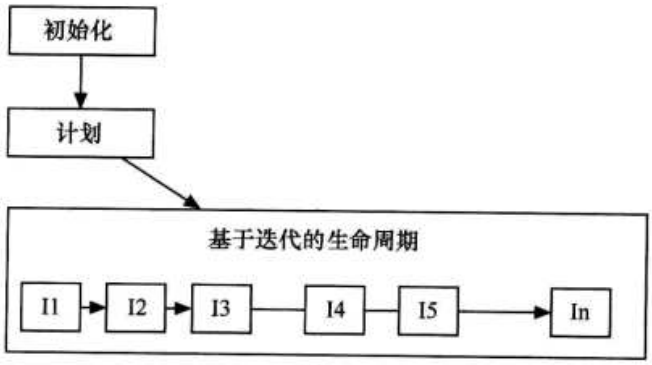

图 9.3

迭代长度根据项目达到稳定状态的时间而变化，它可能持续很长时间。

一次迭代是一个小的独立项目，它可能由以下 4 个阶段组成（如图 9.4 所示）：

- 计划阶段——定义所要做的工作；
- 分析、设计和测试驱动开发阶段——完成所要做的工作；
- 清理阶段——测试完成的工作并调试；
- 发行阶段——交付完成的软件。

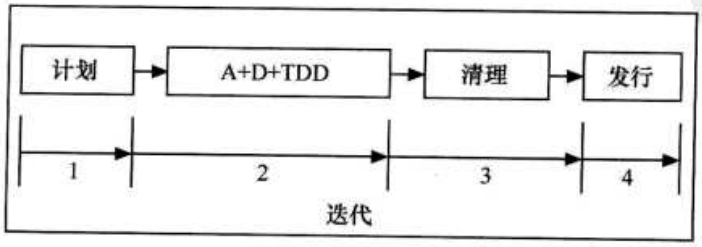

图 9.4

对于持续 2 周的一个迭代，也就是 10 个工作日，每个阶段的时间可以是：

- 1 天用于计划；
- 7 天用于开发；
- 1～2 天用于整体清理；
- 1 天用于发行。

9.2.1　计划

迭代的计划阶段将定义一系列必须完成的任务。一个任务是一个开发者所能完成的针对代码的一个原始操作。在给定剩余时间和开发人员数量的前提下，估计每个任务的周期以确定工作量是现实的。

这个计划由管理人员基于开发人员的反馈来完成，所有估计必须由实际工作的人来验证其正确性。在这个步骤的最后，迭代应该通过一个完整的任务列表被清晰地定义。

9.2.2　开发

开发阶段由每个开发人员所执行的任务组成，包括：

- 接受任务，每个人了解所需要处理的工作；
- 评审并改正预计的时间，这样管理人员知道估算是否准确，这个评审对于知道迭代的工作负荷是否现实也很重要；
- 编码；
- 完成时关闭任务；
- 以同样的方式执行另一个任务。

9.2.3　整体调试

整体调试阶段将关闭整个迭代，测试整个软件并完成剩余的任务。

执行最后这个步骤的最有效方法是聚集所有开发人员和管理人员，一起参加一个特殊的会议。

不能完成的任务将被推迟到下一次发行，在这个最后的阶段总能看到一些被掌声打断的展示。

9.2.4　发行

发行阶段由以下工作组成：

- 标记代码并创建发行，就像前一章中介绍的那样；
- 启动一个新的迭代。

9.3 建立一个跟踪系统

计划阶段定义了迭代内所需完成的任务。为了保持对这些任务的跟踪，可以使用一个跟踪系统。

这样的应用程序用来维护每个任务的相关信息，诸如：

- 负责人；
- 任务特性，包括新功能、调试、重构等；
- 到期日；
- 计划迭代；
- 状态，包括未解决、已完成等；
- 预计费用。

每个任务可以被放到一个迭代中，软件需要提供迭代的整体信息。在迭代的开发阶段，每个任务的状态由开发人员更新，这样就可以跟踪整体的状态。

9.3.1 Trac

Trac 是解决此类问题的一个很好的候选者。这个基于 wiki 的 Web 应用程序是个功能完整的问题跟踪系统，还提供了许多有用的功能（如图 9.5 所示）：

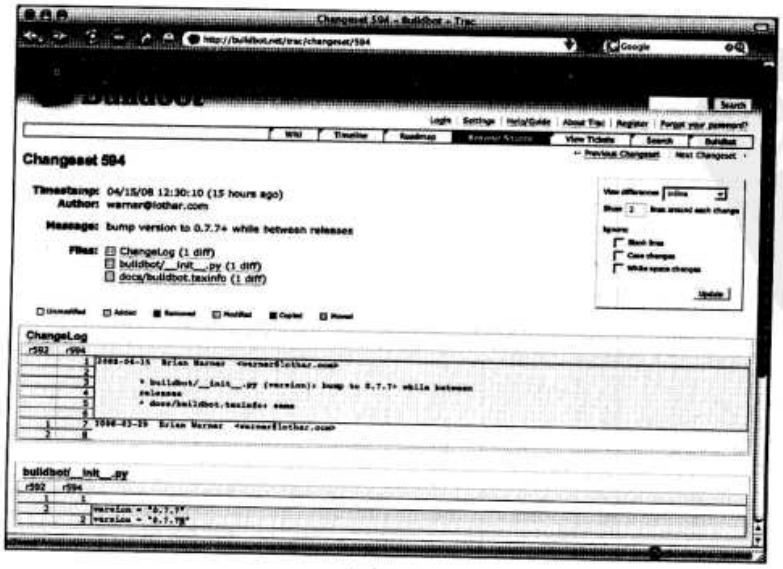

图 9.5

- 用户管理系统；
- 完全可编辑的基于 wiki 的界面；
- 允许为软件添加新功能的插件系统。

Trac 可以通过连接仓库和 Web 界面的插件来与大部分版本控制系统顺利地进行交互。现在可以通过 Web 浏览仓库树，用一条实时的时间线来显示提交，还有易于理解的 diff 报告。

在开放源码社区中，这个工具经常被用做项目网站，它为开发人员提供了处理代码所需的所有功能。以下是使用 Trac 的软件的应用例子。

- Plone　http://dev.plone.org/plone
- Buildbot　http://buildbot.net/trac
- Adium　http://trac.adiumx.com/

Trac 维护一个 Who Uses Trac?(谁使用 Trac？)页面 http://trac.edgewall. org/wiki/TracUsers。如果项目使用了它，可以将项目信息添加到该页面中。

它的界面风格非常简洁，也十分易于理解和使用。

1．安装

PyPI 上名为 pbp.recipe.trac 的 recipe 为配置一个与 Mercurial 服务器集成的 Trac 实例提供了一种便捷的方法。

在前一章中为 Buildbot 创建的 buildbot.cfg 文件里，可以添加一个新的 trac 小节，如下所示。

```
[buildout]
parts =
    buildmaster
    linux
    trac
[buildmaster]
...
[buildslave]
...
[trac]
recipe = pbp.recipe.trac
project-name = Atomisator
project-url = ${buildmaster:project-url}
repos-type = hg
```

```
repos-path = /home/mercurial/atomisator/repositories/unstable
buildbot-url = http://buildbot-atomisator.ziade.org/
components =
   atomisator.db      tarek
   atomisator.feed    tarek
   atomisator.main    tarek
   atomisator.parser  tarek
   pbp.recipe.trac    tarek
header-logo = atomisator.png
```

这个新小节将定义：

- 用做标题的 project name（项目名称）；
- 在 Trac 实例中用做主页的 project url（项目 URL）；
- 用以安装正确插件的 repository type（仓库类型），在本示例中是 hg；
- 指向仓库（必须在同一个服务器上）的 repository path（仓库路径）；
- 链接到 Trac 导航栏上的导航按钮的 Buildbot url；
- 将用于问题跟踪器的 component list（组件列表），带有组件名称和所有者；
- 指向一个图像的 header-logo，这个图像将被用来替代标题中的 Trac 图标。

在本示例中，图标将被放置在 buildout 文件夹中。

再次运行专用于 Buildbot 和 Trac 的 buildout 实例，如下所示。

```
$ bin/buildout -v
...
Project environment for 'Atomisator' created.
...
Try running the Trac standalone web server `tracd`:
   tracd --port 8000 /home/mercurial/atomisator/buildbot/parts/trac
...
Creating new milestone 'future'
Creating new component 'atomisator.db'
Creating new component 'atomisator.feed'
Creating new component 'atomisator.main'
Creating new component 'atomisator.parser'
```

这样，就在 parts/trac 中添加了一个 Trac 环境，在 bin 目录中添加了两个新的脚本：

- tracd　用于运行 Trac 实例的独立 Web 服务器；
- trac-admin　可以用来管理该实例的命令行 shell 程序。

尝试在 buildout 文件夹运行 tracd 脚本，如下所示。

```
$ bin/tracd --port 8000 parts/trac
Server starting in PID 24971.
Serving on 0.0.0.0:8000 view at http://127.0.0.1:8000/
```

Trac 实例应该可以在浏览器中以 http://127.0.0.1:8000/trac 访问到，如图 9.6 所示。

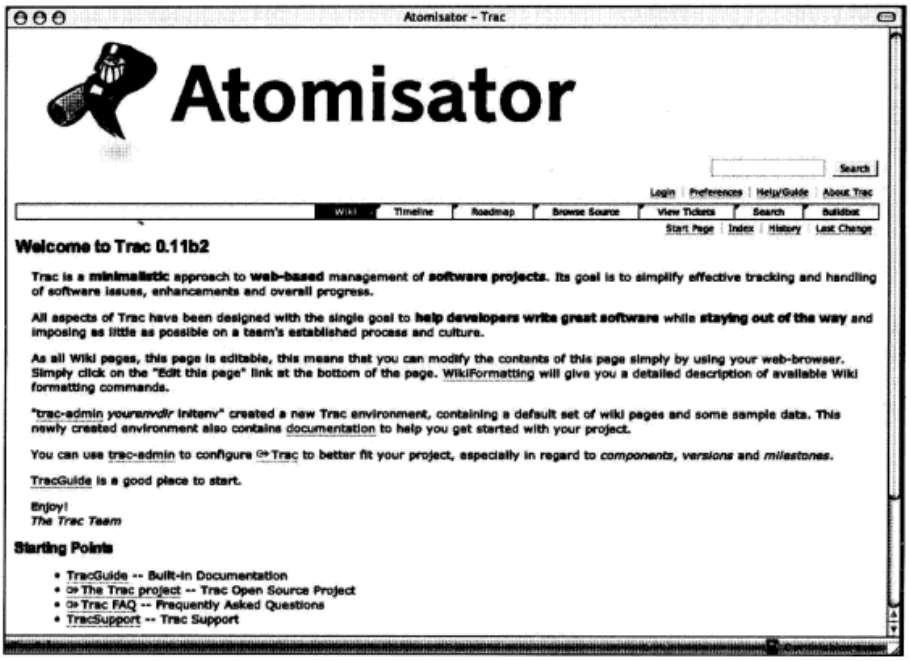

图 9.6

注意，通过 Buildbot 按钮可以访问 Buildbot 网页。

2. Apache 设置

和 Buildbot 类似，Trac 可以通过多个句柄与 Apache 挂钩。最简单的方法是使用 mod_python 句柄，这可以连接到 Trac 提供的前端上。

可以用以下命令将 mod_python 安装到 Debian Linux 上。

```
$ sudo apt-get install libapache2-mod-python
```

对于其他平台，可以参考项目页面http://www.modpython.org。

现在，可以在 Apache 配置中添加新的主机，如下所示。

```
<VirtualHost *:80>
  ServerName atomisator.ziade.org
  <Location />
    SetHandler mod_python
    PythonHandler trac.web.modpython_frontend
```

```
    PythonOption TracEnv /home/mercurial/atomisator/buildbot/parts/
trac
    PythonOption TracUriRoot /
    PythonPath "sys.path + ['/home/mercurial/atomisator/buildbot/
parts/trac', '/home/mercurial/atomisator/buildbot/eggs']"
  </Location>
  <Location "/login">
    AuthType Basic
    AuthName "Trac login"
    AuthUserFile /home/mercurial/atomisator/passwords
    Require valid-user
  </Location>
</VirtualHost>
```

在这个配置中，有几个需要注意的地方：

- PythonOption 定义一个 TracEnv 值，所以 Trac 系统知道实例所在的位置；
- PythonPath 选项指向脚本访问 Trac 模块时所需要的本地 buildout 目录；
- /login 小节的设置与之前为 Mercurial 创建的 passwords 文件挂钩，这样用户可以使用相同的用户名登录到系统中。

3. 权限设置

为了使用问题管理系统，需要定义几个组：

- manager（管理员） 能够完全管理 Trac 的人；
- developer（开发人员） 能够修改问题票（ticket）及 wiki 页面的人；
- authenticated（已验证的） 能够创建问题票的人；
- anonymous（匿名的） 能够查看所有内容的人。

这 4 个角色都已经在 Trac 设置好并可以使用了。这是一个默认的权限设置，或者是在 buildout 运行时由 pbp.recipe.trac 自动完成的设置。

剩下的工作是在每个组中添加一些人。Trac 提供了一个命令行使用程序，可以用来在实例中设置一些元素，如下所示。

```
$ bin/trac-admin parts/trac/
Welcome to trac-admin 0.11b2
Interactive Trac administration console.
Copyright (c) 2003-2007 Edgewall Software
Type: '?' or 'help' for help on commands.
Trac [parts/trac]>
```

现在，可以在组中添加一个用户。例如，定义 tarek 为管理员，bill 和 bob 为开发人员，命令如下。

```
Trac [parts/trac] > permission add tarek manager
Trac [parts/trac] > permission add bob developer
Trac [parts/trac] > permission add bill developer
```

这将允许 tarek 通过 Web 来管理项目，bill 和 bob 则能够处理问题票和 wiki 页面。

9.3.2　使用 Trac 管理项目生命周期

这些设置使 Trac 在生命周期的管理中很容易使用，可以通过 Web 界面或命令行工具来进行。

1．计划

在 Trac 中进行计划只需创建一个新的对应于迭代的里程碑（milestone），并确定它何时应该交付。管理员可通过 Web 界面或 trac-admin 命令行来添加它。后者更方便些，但是这意味着必须连接到执行那些任务的服务器上。

使用命令行来添加它，如下所示。

```
Trac [parts/trac] > milestone add atomisator-0.1.0
Trac [parts/trac] > milestone due atomisator-0.1.0 2008-08-01
```

相同的操作可以通过/admin/ticket/milestones 上的 Web 界面中的 admin 部分来执行。

接着，里程碑将出现在 roadmap 部分。现在可以通过在 Web 界面上单击 New Ticket（新增问题票）按钮，来为该里程碑添加问题票。

创建一个为 atomisator.db 定义任务的问题票，定义一个保存反馈条目的映射表，如图 9.7 所示。

除了摘要和描述之外，要提供的重要信息包括：

- Assign to（任务分配给...）　开发人员姓名（我们将其分配给 Bob）；
- Type（类型）　Task（任务），因为这是一个新功能；
- Component（组件）　atomisator.db；
- Milestone（里程碑）　atomisator-0.1.0；
- Estimated hours（估计工时）　8。

这个问题票将出现在里程碑中。

缺陷和改进问题票也是以相同的方式输入的。

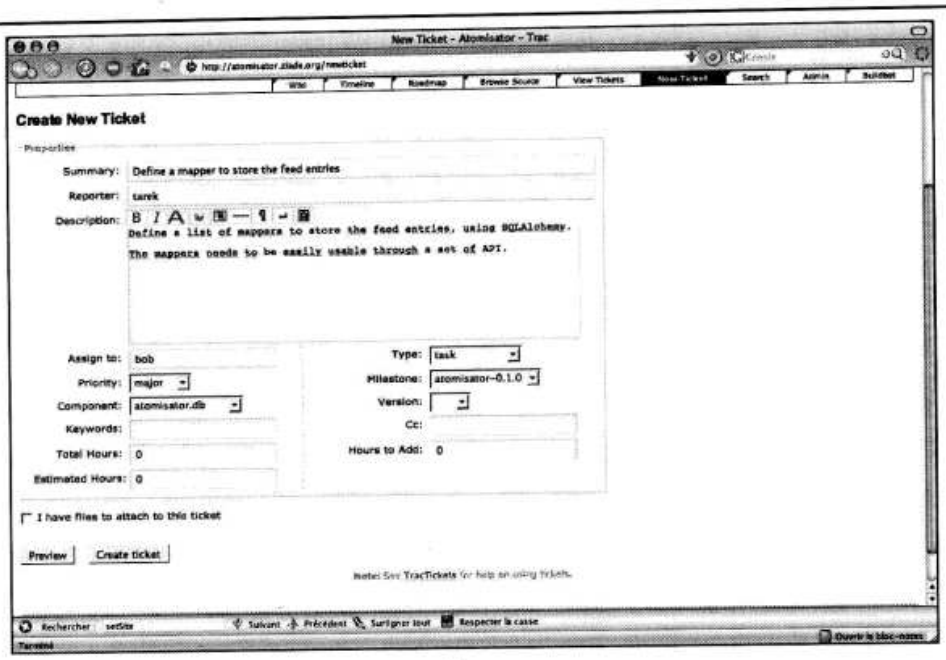

图 9.7

预计工时属于 Trac 的一个名为 TimeAndEstimation 的插件，pbp.recipe.trac 会自动安装它。

相关内容请参见 http://trac-hacks.org/wiki/TimingAndEstimationPlugin。

2．开发

每个开发人员可以通过/report 小节下的报告来查看他的任务，这可以通过 View Tickets（查看问题票）按钮来访问。My Tickets（我的问题票）将显示一个当前用户的问题票列表。

当 Bob 开始着手前面输入的问题票定义的工作时，他将执行 accept 操作。

任务完成时，他将填写 Total Hours（合计工时）字段并且执行 resolve 操作。

这种轻量级的时间管理不能替代真正的管理计划系统，但是能够提供任务所花费时间的有用指示。

时间跟踪是改进估算的重要内容。

3．清理

在团队结束一次迭代时，往往会有一些任务尚未完成，清理阶段是最大限度地解决小问题的好机会。有些团队会组织缺陷冲刺，并在一两天内完成缺陷的处理。

在这个阶段结束时，剩余的任务将推迟到未来的里程碑，除非它们特别受欢迎。在那种情况下，它们必须被修复，并希望能在该阶段结束之前完成。

这些操作都通过在 Web 界面上编辑每个任务来完成。

4．发行

发行由以下工作组成：

- 标记代码；
- 将对程序的修改从不稳定仓库中拉取到稳定仓库中；
- 创建发行仓库；
- 准备并交付一个发行版本；
- 将里程碑状态设置为 completed，以便关闭它；
- 创建新的里程碑并开始一个新的周期，有些团队会创建多达 3 个未来的里程碑，能够将问题票推进到具有优先级的未来里程碑中。

标记和拉取的工作发生在 Mercurial 端，这在前一章中已经介绍过，可以使用以下的命令集。

```
$ cd /home/mercurial/atomisator/repositories/unstable
$ hg tag -f -m "tag for 0.1.0 release" 0.1.0
$ cd ../stable
$ hg pull ../unstable
$ hg clone
```

最后，通过填充到期日来完成里程碑，如下所示。

```
Trac [parts/trac] > milestone completed atomisator-0.1.0 2008-08-01
```

Trac 可以用做信息的中心，创建的发行版本可以添加到 wiki 页面上，并在页面上公告。

9.4　小　　结

本章中介绍了以下内容：

- 各种管理软件开发生命周期的方法；
- 迭代开发方法的优点；
- 将项目组织为迭代的方法；
- Trac 的安装和使用；

下一章中将关注于软件文档。