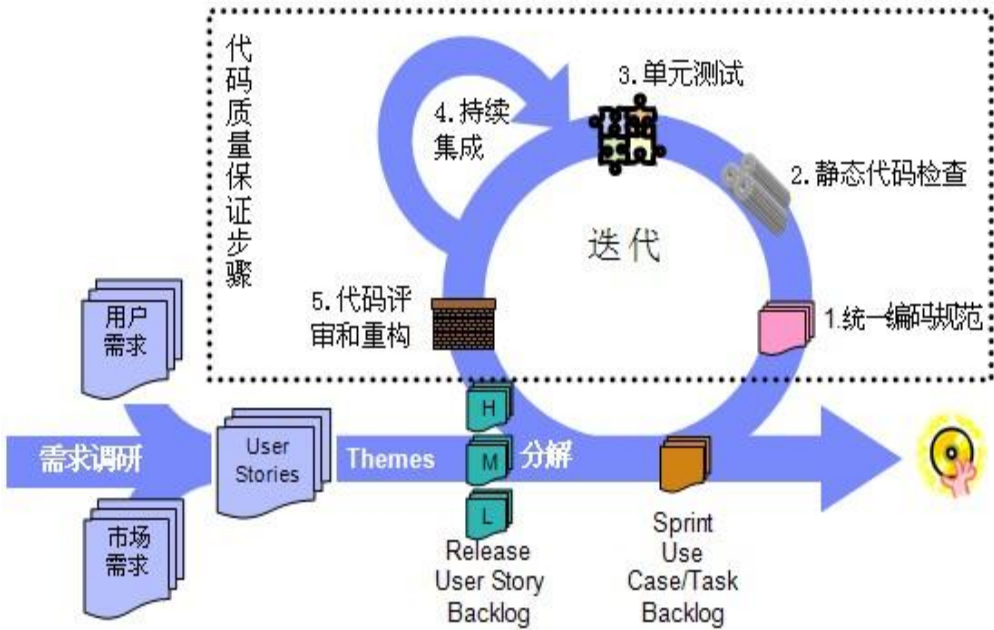


敏捷开发中编写高质量 Java 代码

敏捷开发的理念已经流行了很长的时间，在敏捷开发中的开发迭代阶段中，我们可以通过五个步骤，来有效的提高整个项目的代码质量。

Java 项目开发过程中，由于开发人员的经验、Java 代码编写习惯，以及缺乏统一的标准和管理流程，往往导致整个项目的代码质量较差，难于维护，需要较大的测试投入和周期等问题。这些问题在一个项目组初建、需求和设计均具有不完全可预期性和完备性的全新项目中将尤为突出。

如图 1 所示，敏捷开发过程经历需求调研，用例分析和用例分解，进入开发迭代阶段。在每个迭代过程中，可以采用以下步骤来保证和提高整个项目的代码质量：统一编码规范、代码样式；静态代码分析(staticcodereview)；单元测试；持续集成；代码评审和重构 (Review&Refactor)。下文将针对每个步骤和其所使用的工具、方法进行详细描述。



步骤一：统一编码规范、代码样式

规范统一的编码会增加项目代码的可读性和可维护性，但实际情况往往是项目组内的 Java 代码开发人员的编码风格常常各不相同，这可能是由于不同的经验习惯或者缺乏编码规范方面的学习造成的。这样一来，其他项目成员或者维护人员在阅读项目代码时就需要花费更多的时间来理解代码作者的意图，所以制定并采取统一的编码规范就显得很重要。编码规范主要应包含以下几个方面：

- ◆一般规则和格式规范。例如代码缩进、程序块规范、每行最大代码长度等。
- ◆命名规则。例如包名、类名、变量、方法、接口、参数等命名规范
- ◆文档规范。例如类文件头声明、类注释、成员变量和方法注释等规范。
- ◆编程规范。例如异常、并发、多线程等方面的处理方式。
- ◆其他规范。例如日志格式、属性文件格式，返回值和消息格式。

项目的编码规范可以参考已有的一些 Java 编程规范书籍和其他相关资料并结合项目的本身来制定，可供参考的书籍有《Java 编程风格》(英文书 名为：TheElementsofJavaStyle)。编码规范要形成文档，而且要简洁明了，并组织项目成员一起学习，确保所有成员正理解所有条目。

一旦编码规范确定，就可以利用 Eclipse 自身提供的功能来控制代码样式和格式。具体做法是，点击 Eclipse 的 Windows->Preference 菜单项，在打开的 Preferences 对话框的左侧栏中找到 Java 节点下的子项 CodeStyle(如图 2)，该项 和它的子项允许您对 Java 代码的样式进行控制。

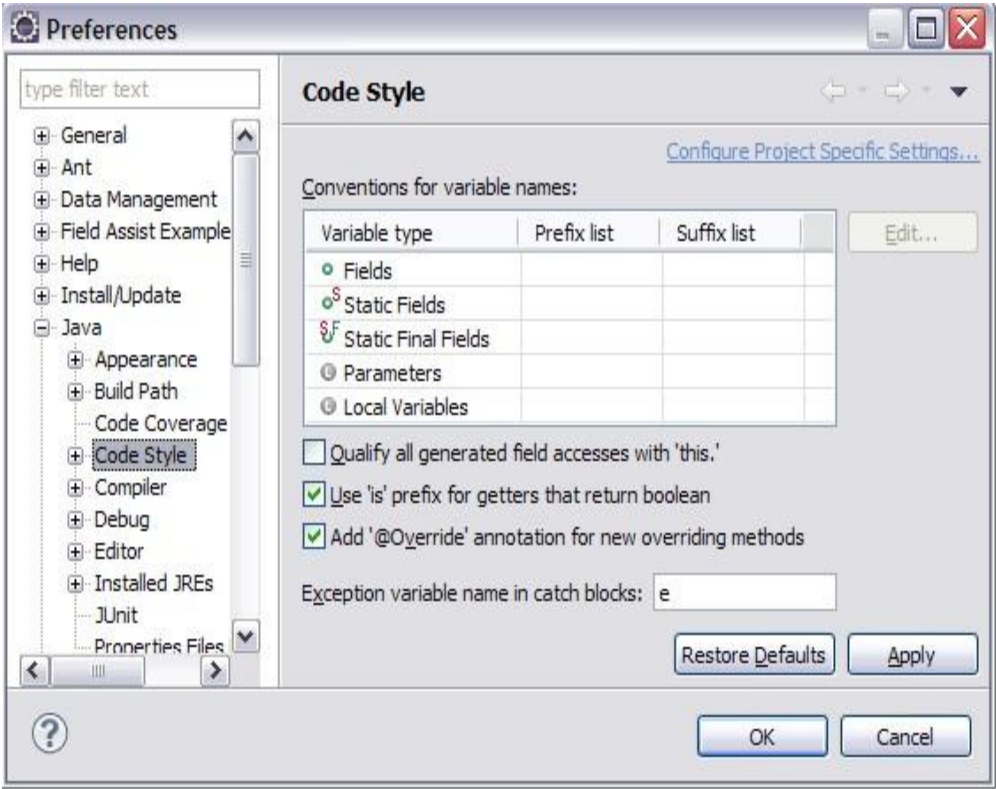


图 2.Eclipse 代码样式设置窗口

例如，为了使用自动格式化工具，可以在 Eclipse 提供的默认代码格式配置的基础上建立自定义的格式。在 Formatter 面板中，点击 New，输入新的名字并选择一个默认的配置作为初始化格式，如图 3 所示。

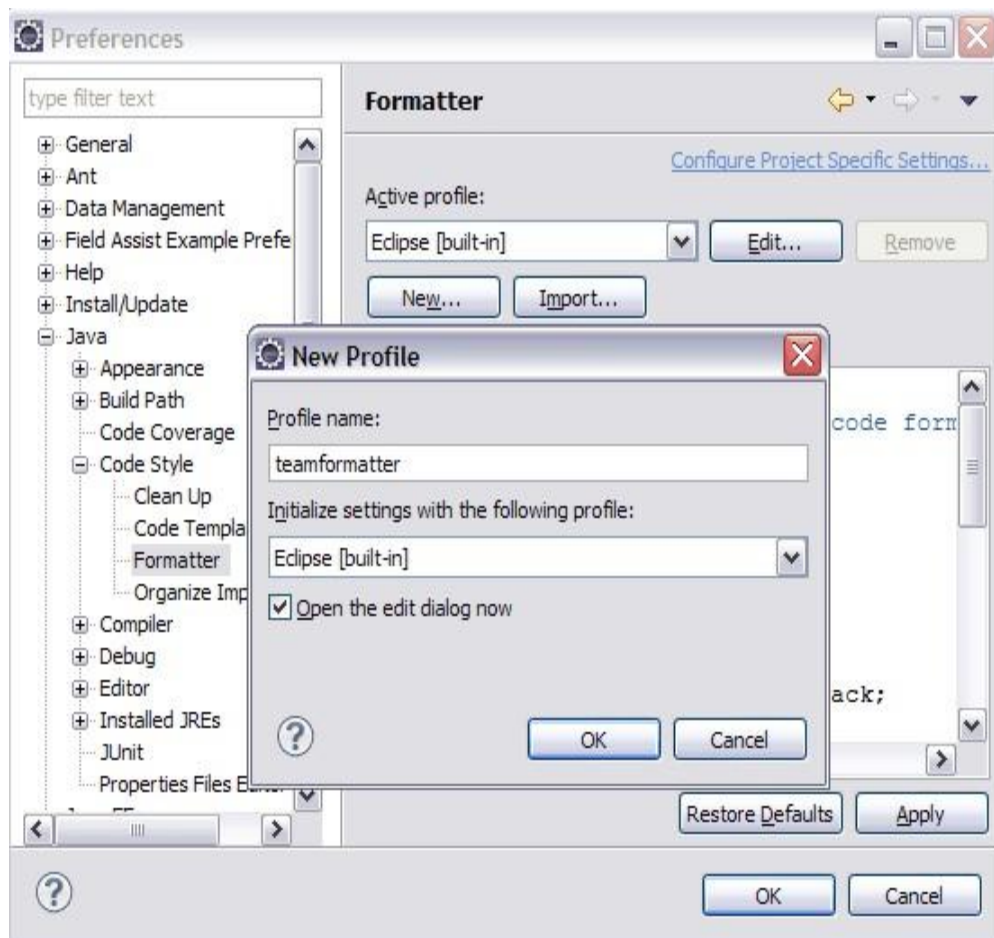


图 3.创建新的代码格式配置

单击 OK 后就可以在新打开的窗口中进行修改定制自己需要的格式。如图 4 所示。

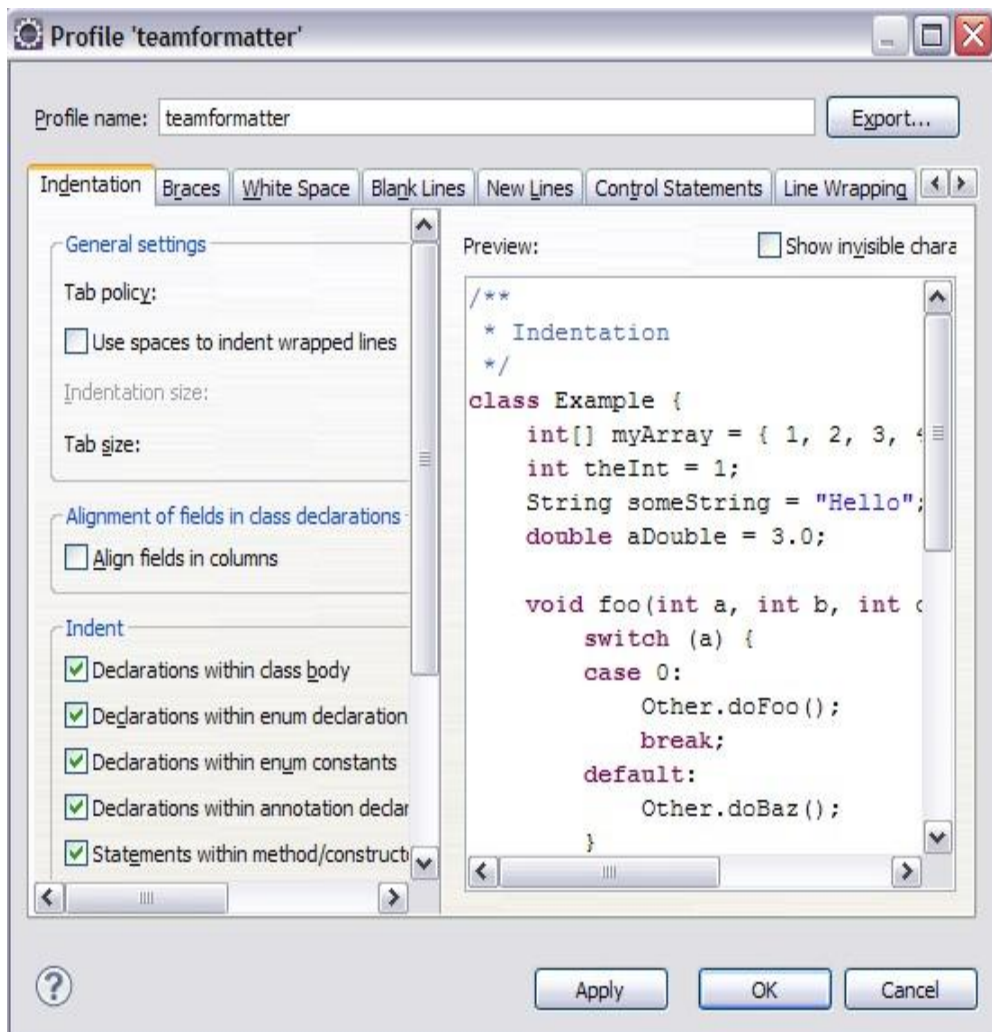


图 4.创建新的代码格式配置

修改完成后点击 **Apply** 保存所作修改。同时可以点击 **Export** 将当前的格式定义导出成一个 XML 文件，这样项目组的其他成员就可以很方便地通过点击图 3 中的 **Import** 按钮来导入该 XML 文件来使用同一个代码格式定义。

这样每次在提交代码到版本控制服务器前就可以通过 Eclipse 界面里的 **Source->Format** 菜单来对代码进行格式化，从而使整个项目的代码具有相同的格式。同样可以通过对 **CodeStyle** 下的其他项目进行设置来帮助对 Java 代码的样式进行控制。将所有这些样式文件导出成 XML 文件后，同编码规范一起归档，供所有项目成员使用。

## 步骤二：静态代码分析

在完成源代码的开发以后，下面要进行的工作就是审视和测试代码。除了通过运行测试代码来检查功能之外，还能利用一些静态分析工具来快速、直接地提高代码质量。静态代码分析工具并不需要运行代码，可以直接对 Java 文件和 Class 文件进行分析，通过一些检查条件的设置，快速找到代码中的错误和潜在缺陷。现在的静态分析工具很多，有 FindBugs、PMD、IBMRationalTool，等等。在这里，选择 FindBugs 作为静态代码分析工具。FindBugs 可以和日常开发工具 Eclipse 进行集成，在开发过程中，就可以方便地开始静态代码的检查。通过检查 Class 文件或者 JAR 文件，将字节码和一组缺陷模式进行对比，来发现可能存在的代码问题。在 Eclipse 的开发环境中，用插件安装的方式安装了 Findbugs 后，在 Eclipse 的配置选项中就会多出来 FindBugs 的配置选项。可以对自己的项目进行配置，选择需要的 Detector 检查代码。

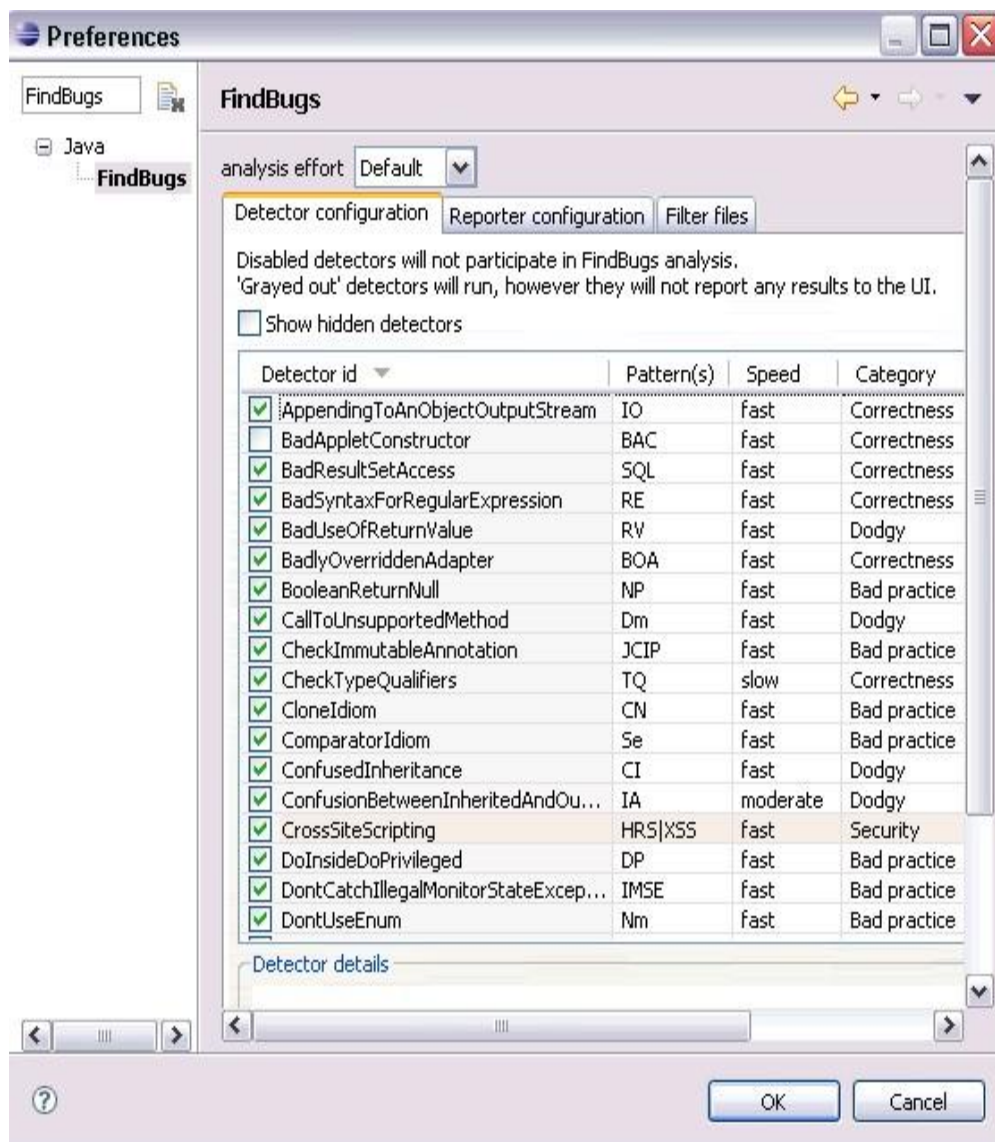


图 5. FindBugs 的配置选项

设置好自己的规则后，在需要检查的代码文件夹上点击右键，就可以启动 FindBugs 检查。代码可以是一个项目，也可以只是几个文件。



图 6. 运行 FindBugs

检查完毕后，会出现 FindBugs 视图，把所有检查的结果根据错误分组展示。点击结果里面的每一个错误，会自动打开对应的代码。当根据规则改正了所有的错误，或者说潜在错误，这些代码也就通过了静态代码检查。FindBugs 的检查结果可以是 XML 文件，也可以是文本文件，便于项目的集成管理和检查保存。



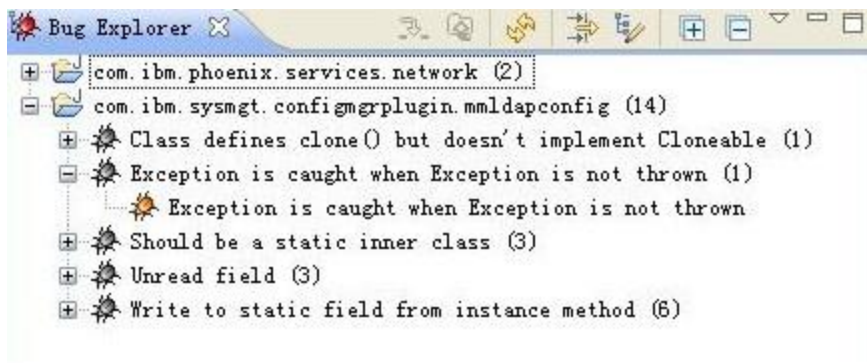


图 7.FindBugs 检查结果

### 步骤三：单元测试

#### 单元测试用例设计和评审

单元测试是软件开发过程中重要的质量保证环节，在此环节中，设计和评审对于保证整个单元测试过程的完整性和有效性来说十分重要。设计阶段需要具体考虑要对哪些代码单元进行测试，被测单元之间的关系，测试策略，以及单元测试用例设计等，并最终输出《单元测试用例设计》文档，用来指导具体的单元测试执行。在用例设计中，通过对代码单元输入和期待输出的定义来保证该单元的功能正确性，边界值的测试和异常测试非常重要。同时也配合测试用例和功能块的匹配方法来衡量用例设计的完整性。

在用例设计完成之后，下一步的工作就是进行测试用例的评审。个人的理解和经验始终是有限的，用例评审可以借集体之力，对用例设计进入查漏补缺，进一步保证测试用例的有效性。由于单元测试属于白盒测试范畴，它主要通过代码的逻辑结构进行分析来设计测试用例，因此，评审员的选择最好以理解代码逻辑结构为前提，如果评审员来自相关模块，还能够有效的发现模块相关性和依赖性所带来的问题。

#### 模拟对象技术

在实际项目中，开发人员自己的代码往往需要和其他的代码模块或系统进行交互，但在测试的过程中，这些需要被调用的真实对象常常很难被实例化，或者这些对象在某些情况下无法被用来测试，例如，真实对象的行为无法预测，真实对象的行为难以触发，或者真实对象的运行速度很慢。这时候，就需要使用模拟对象技术(Mock)，利用一个模拟对象来模拟我们的代码所依赖的真实对象，来帮助完成测试，提高测试覆盖率，从而提高代码质量。模拟对象技术利用了面向接口的编程中，由于代码直接对接口进行调用，所以代码并不知道引用的是真实对象还是模拟对象，这样就可以顺利的完成对代码的测试，模拟技术有很多种，如 jMock, EasyMock, Mockito, PowerMock 等等。其中 Mockito 消除了对期望行为的需求，避免了这些代码的大量初始化。

```
//You can mock concrete classes, not only interfaces
LinkedList mockedList = mock(LinkedList.class);

//stubbing - before execution
when(mockedList.get(0)).thenReturn("first");

//following prints "first"
System.out.println(mockedList.get(0));

//following prints "null" because get(999) was not stubbed
System.out.println(mockedList.get(999));
```

图 8.Mockito 示例

在模拟对象过程中，先模拟一个需要调用的 `List` 对象 `LinkedList`，再设定这个对象的行为，当调用 `get(0)` 的时候，返回“first”。这样，测试代码就可以利用这个对象来测试我们的功能代码，需要调用和返回值的时候，可以顺利的得到模拟对象的返回值。也需要对模拟对象 进行错误情况的模拟，保证代码对错误的处理的正确性。

**测试覆盖率分析**

为了衡量单元测试的质量和覆盖的范围，需要对单元测试的代码进行测试覆盖分析。常用的衡量测试覆盖率的指标主要有语句覆盖率、分支覆盖率、路径覆盖率、条件覆盖率和方法覆盖率等。具体采用哪些指标可以根据项目的实际情况来定，以避免因过高的指标增加了代码开发人员的工作量而影响了项目整体的进度。

EMMA 是一款比较流行的开源 Java 测试覆盖率分析工具，支持类、方法、代码行、基本代码块等多种类型的测试覆盖率分析，支持将覆盖率分析结果导出为多种格式的报告，并采用多种颜色来高亮显示不同的覆盖率状态。EclEmma 是一款基于 EMMA 的 Eclipse 插件，方便在 EclipseIDE 中进行测试覆盖率分析。如图 9，在测试用例写好后，可以在右键点击测试类，选择 `CoverageAs->JUnitTest`。

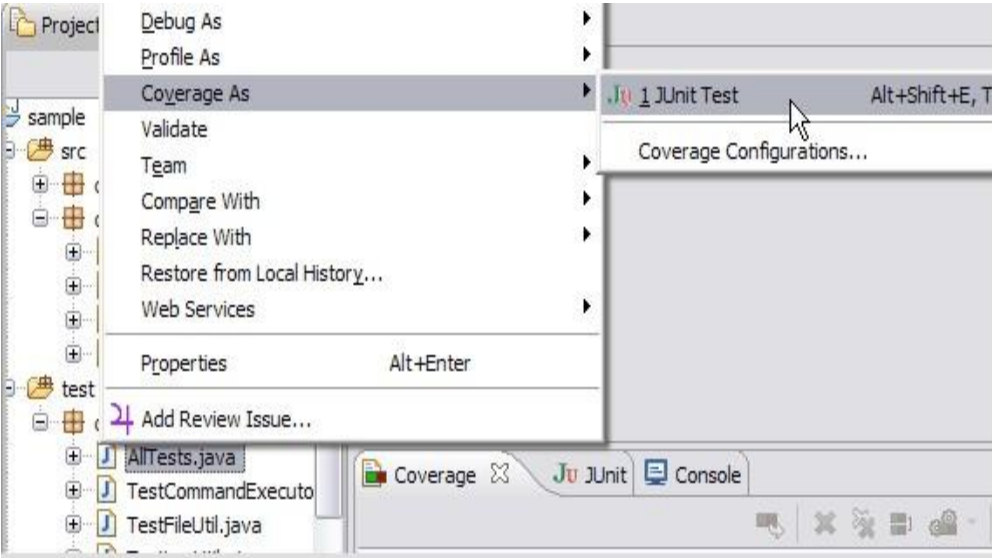


图 9.运行测试覆盖分析

单元测试跑完后，`Coverage` 视图中会显示所选择的测试的覆盖率。双击打开某一具体的类后，可以看到高亮显示的覆盖分析结果，如图 10 所示。红色代表测试没有覆盖到该行，黄色表示部分覆盖，绿色的行表示该行在本次测试中被覆盖到。

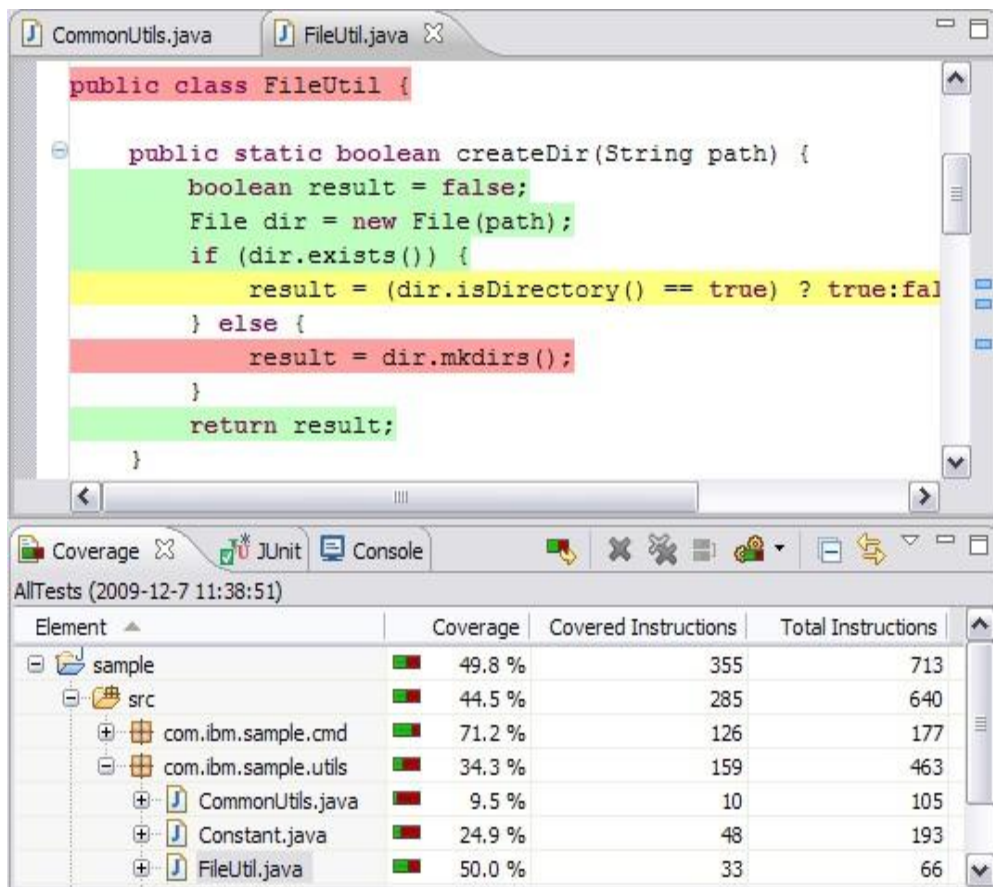


图 10.查看测试覆盖分析结果

在 Coverage 视图中可以通过点击鼠标右键将测试覆盖分析的结果导出成需要的格式，例如 HTML。

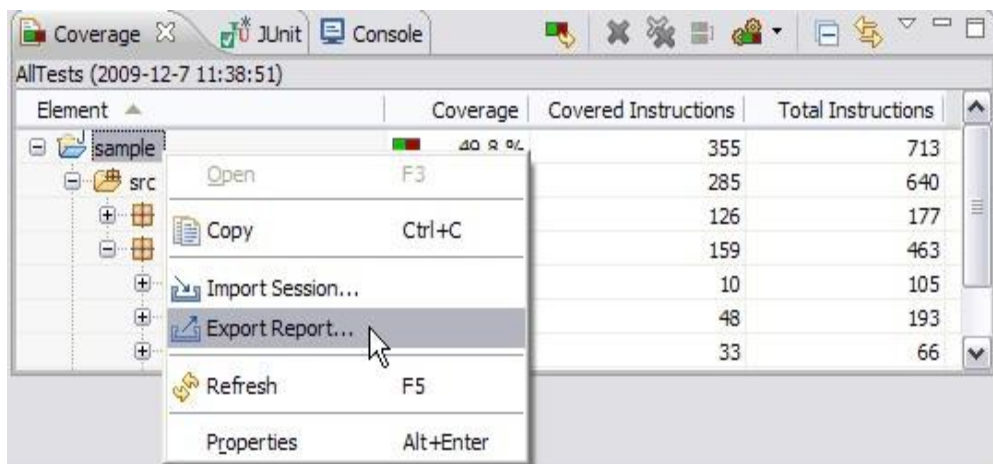


图 11.导出测试覆盖分析结果

图 12 显示了导出的 report。



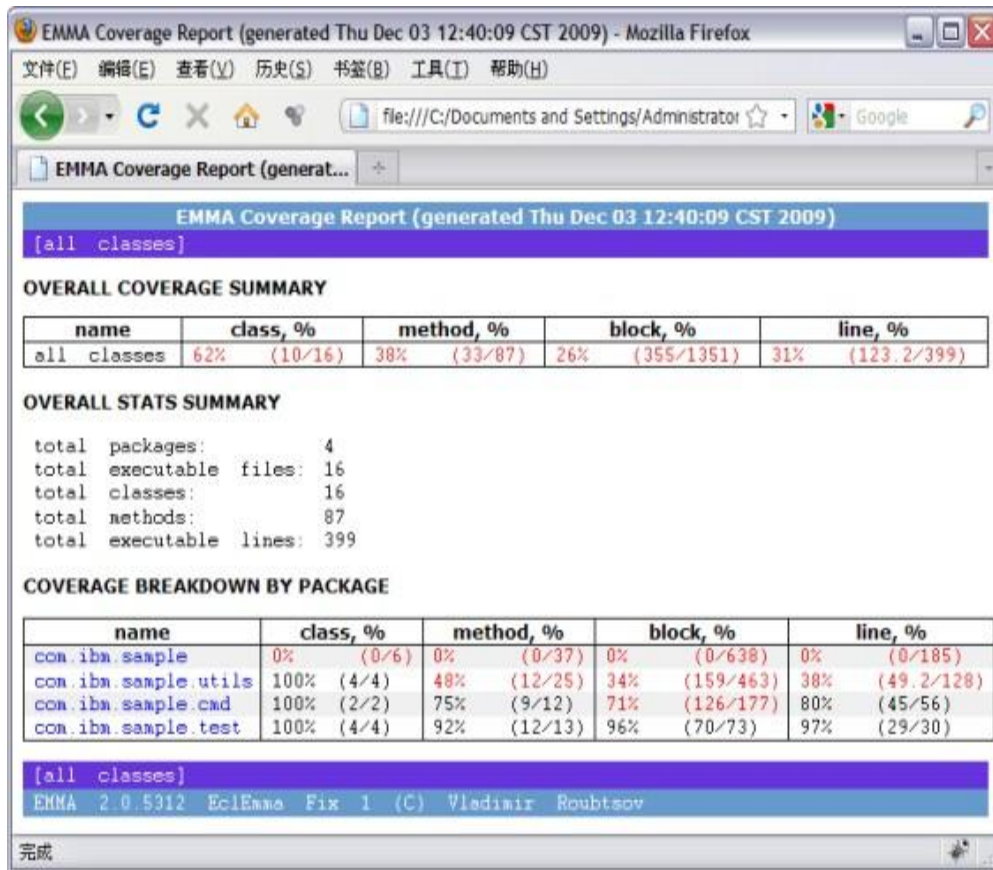


图 12.测试覆盖分析报告

为了保证单元测试的有效性和质量，可以规定一个测试覆盖率的下限，例如所有的包和类的覆盖率必须达到 **80%**以上。不过值得注意的是，不要单纯追求高覆盖率，要同时注意测试用例的质量，如果测试用例本身就写的有错误，那么即使测试覆盖率很高也没有意义。

#### 步骤四：持续集成

持续集成(ContinuousIntegration)是利用一系列的工具，方法和规则，做到快速的构建开发代码，自动的测试化，来提高开发代码的效率和质量。利用自动构建工具，随时都能把提交的代码构建出来，提供一个可以测试使用的版本，让用户和开发人员同时看到相同的功能，尽早的发现问题和错误，也可以尽快的得到测试人员和用户的反馈。

要做到持续集成，就要利用一系列工具，把开发过程中的重复工作自动化。搭建自动的构建服务器，自动的进行单元测试和发布新版本，一个集成的服务器可以提供构建过程的结果报告，自动通知开发人员构建结果，并且保存历史数据。 **IBMRationalTeamConcert(RTC)**可以提供工作任务的管理，项目计划的安排，代码版本管理控制，自动构建可用版本，生成构建结果报告。这些过程构成了项目的持续集成过程，其中，版本的自动构建和代码的自动单元测试是持续集成的关键过程，RTC在这些过程上提供了有力的支持。

#### 自动构建

RTC 提供了 **buildengine** 来负责构建 **build**，首选，启动 **buildengine**，并和 RTC 服务器建立了连接。再创建项目的 **build** 定义。在这个定义中，需要设定编译哪些模块的代码，需要跳动哪个 **ANT** 文件来启动编译，和一些编译过程中的参数的设定。当这些都准备好了，编译对于项目而言，就变成一个简单的事情。

可以看到，通过在 build 定义上，点击请求构建，就可以触发一次构建过程。选择需要的构建参数，这个过程就会在后台运行。每一个开发人员，做了少许的代码改变和提交，都可以触发新的构建过程，来保证我们代码的有效性。申请一个新的构建的过程如图 13、图 14 所示。

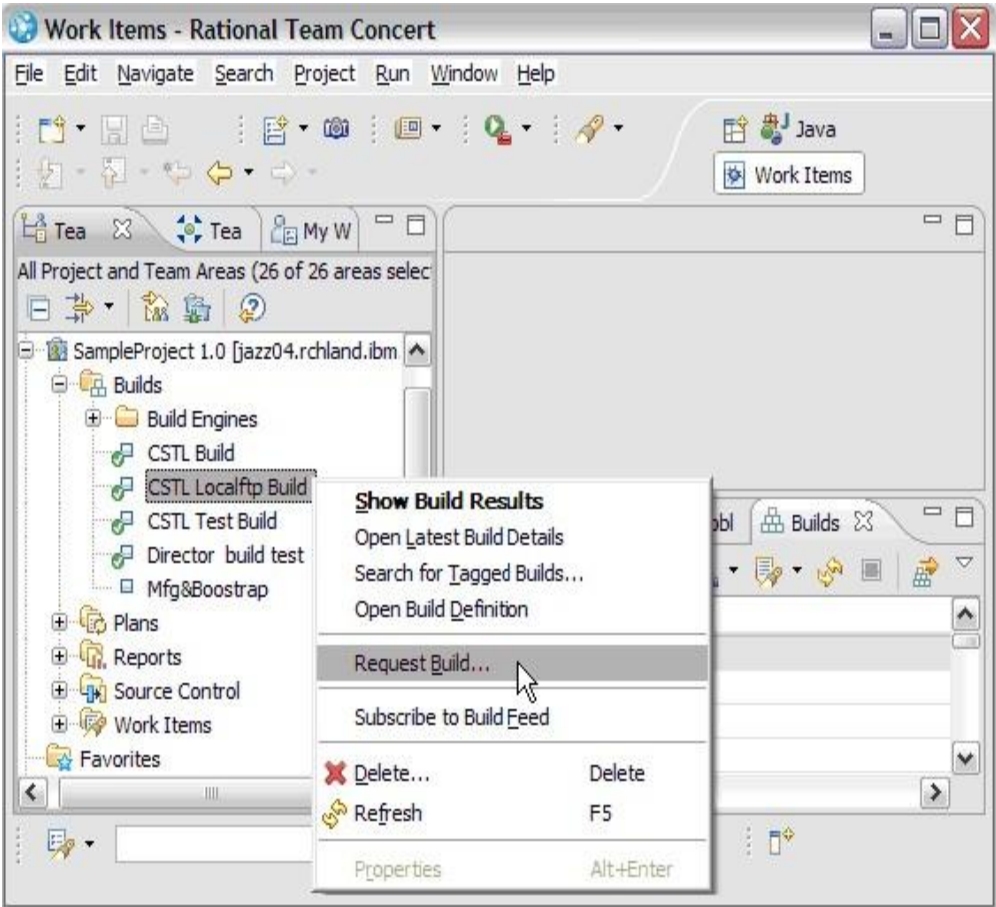


图 13. 申请一个新的构建

**Request Build**

Request that a build be executed on the first available build engine.

Build:

▼ Build Options

☐ Personal Build

Check this option to make this a personal build. The build will run using a workspace you specify, and will not impact the build definition status. Personal build alerts will only be received by you, not other members of your team.

Repository workspace:\*

Component load rules:

▼ Build Properties

Name ▲	Value
build.dest	C:/phoenix_build/phoenixdest
build.lib	C:/phoenix_build/phoenixresource
build.temp	C:/phoenix_build/phoenixtemp
fetch.dir	C:/phoenix_build/phoenixfecth
report	false

Build Property Description:  
<select a build property to see its description>

After submitting the request:  ▼

图 14. 构建申请界面

当构建结束后，RTC 服务器会提供构建结果报告。开发人员可以查询到这次构建的详细信息。

20091127-1744

### Build CSTL Localftp Build 20091127-1744


Save

✓ **Completed**


Duration: 2 minutes, 43 seconds


Start Time: 2009年11月27日 下午05:44:45


Completed: 2009年11月27日 下午05:47:28

Status Trend: 

**Reported Work Items**

 None reported against this build

 [Create a new work item](#)

 [Associate an existing work item](#)

**Contribution Summary**

Logs: [1 log](#)

Work items: [2 included in build](#)

Snapshot: [CSTL Localftp Build 20091127-1744](#)

Repository Workspace: [SampleProject 1.0 Team Stream Workspace](#)

Changes: [Show changes](#)

**General Information**

Requested by: Yong Kui Wang

Build Definition: [CSTL Localftp Build](#)

Build Engine: [CSLTPhoenix](#)


Build History: [538 builds](#)

Tags:

☒ Deletion allowed

**Associated Release**

Released builds are available as choices in the work item "Found In" field.

 [Create a release to associate with this build](#)

Overview Activities Logs Properties

图 15. 构建结果

整个开发过程中，构建版本的过程应该是无数次的，通过每次构建，都可以得到当时代码的编译情况，并且可以得到一个可运行的软件版本。在构建定义上，RTC 支持设置构建计划。定时自动的触发一次构建。

CSTL Localftp Build

Build Definition

ID: CSTL Localftp Build Project or Team Area: SampleProject 1.0 Team

Save

Browse...

**Schedule**

Schedule for automatically running this build.

☒ Enabled

**Build Time**

☐ Continuous interval in minutes:

☒ At: AM 08 : 00

**Build Days**

☒ Monday ☒ Tuesday ☒ Wednesday ☒ Thursday ☒ Friday ☒ Saturday ☒ Sunday

Select All

Deselect All

Overview Schedule Properties Jazz Source Control Ant

图 16.构建定义

### 自动单元测试

构建可以自动了，重点提高代码质量的单元测试呢？如果每一天的代码，每一个版本的代码，都已经通过了我们的单元测试，这样我们就能对代码的质量 有了基本的保证。在构建脚本的自动调用过程中，通过 ANT 的脚本，可以加上 JUnit，EMMA，FindBugs 的 ANT 脚本调用，每一次的构建，都可以把这些检查工作自动的进行一遍测试。这些测试都要生成测试结果报告，RTC 不能提供这些报告的展示，就可以利用 Hudson 这个开源工具，集成测试报告 来方便查阅。



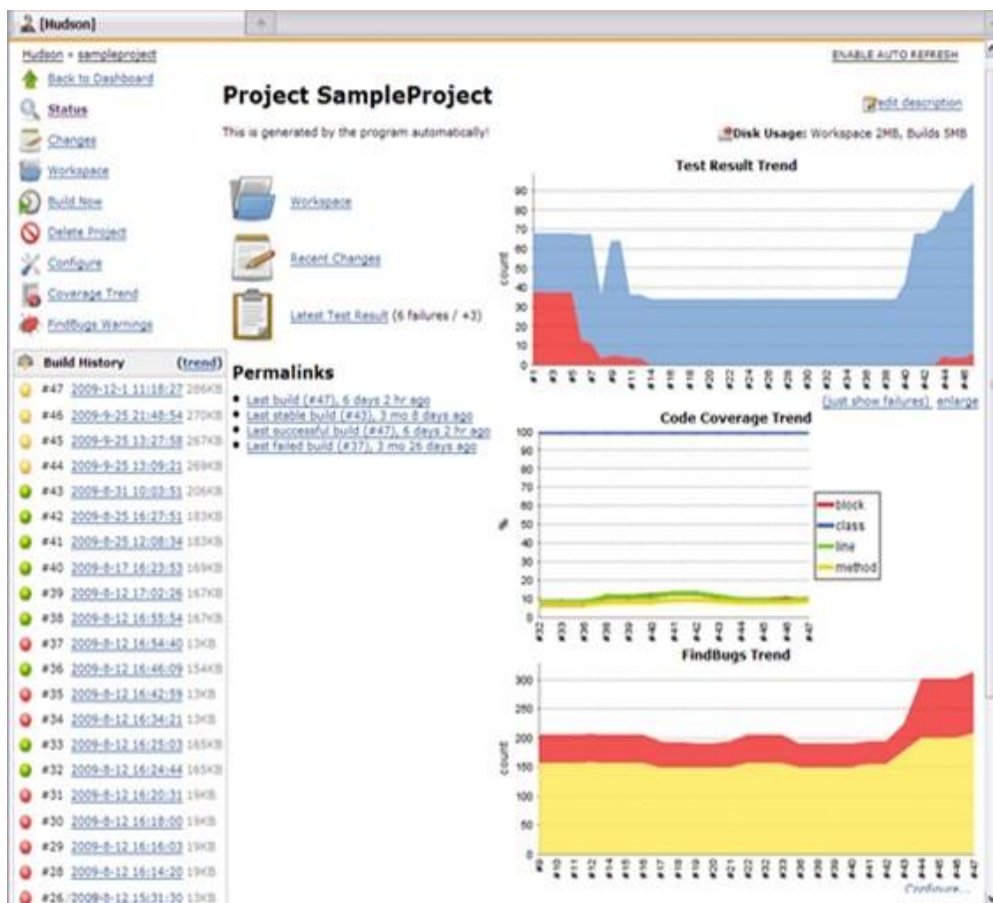


图 17.自动测试报告

#### 步骤五：代码评审和重构

代码评审(CodeReview)是 Java 项目开发过程中的一个重要步骤，代码评审可以帮助发现静态代码分析过程中无法发现的一些问题，例如 代码的编写是否符合编码规范，代码在逻辑上或者功能上是否存在错误，代码在执行效率和性能上是否有需要改进的地方，代码的注释是否完整正确，代码是否存在 冗余和重复。代码评审还可以帮助新进入项目组的成员快速学习和了解项目，促进经验分享，同时也能保证项目成员的良好沟通。代码评审主要包括两种形式，同级 评审(PeerReview)和小组评审(GroupReview)。同级评审主要指项目成员间的互相评审，小组评审是指通过召开评审会议，项目成员一起 对项目代码进行评审。

为了提高代码评审的有效性和效率，可以借助一些外部工具，比较常用的代码评审工具有 Jupiter 和 CodeStriker。Jupiter 是一款开源的 Eclipse 插件，允许成员将评审意见定位到真实代码的具体行，由于代码评审的结果以 XML 文件的形式保存，所以可以把结果提交到版本管理服务进行共享。图 18 显示了使用 Jupiter 进行代码评审的界面。

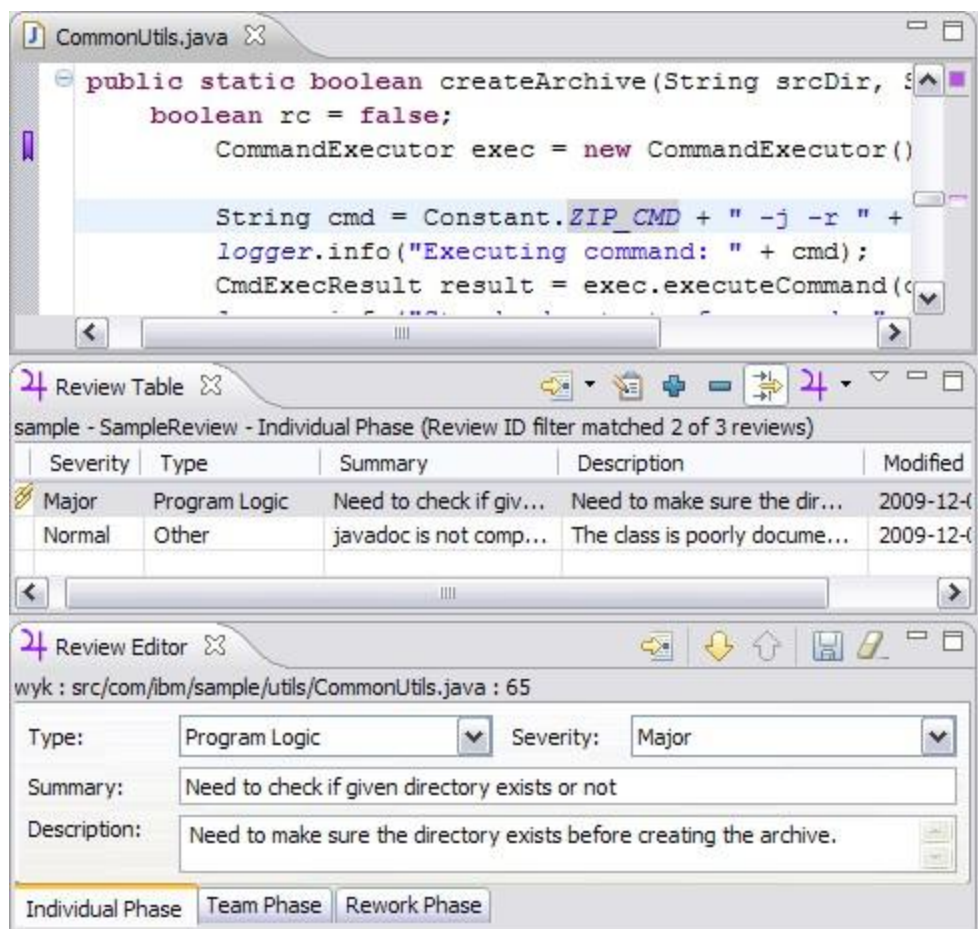


图 18.Jupiter 代码评审界面

在代码评审任务创建后，Jupiter 将代码评审分成三个阶段，个人评审阶段(IndividualPhase)、团队评审阶段 (TeamPhase)和问题修复阶段 (ReworkPhase)。在个人评审阶段，评审成员将发现的代码问题或者缺陷记录下来，每个问题都会作为一个记录保存在评审表格中。在团队评审阶段，团队的全部或者部分成员会一起对个人评审阶段发现的问题进行定性，如果问题确实存在，就将该问题分配给某个成员去解决，并在 Jupiter 中将该问题设置成相应的状态。在问题修复阶段，团队成员会修复属于自己的问题，并将相应的记录设置成已解决等正确的状态。

Codestriker 是一款基于 Web 的常用代码评审工具，对代码的评审可以针对某一具体行，也可以针对整个代码文件，评审意见会被保存在数据库中。评审人员可以同时看到其他人的评论，代码作者也可以针对某一具体的评论回复。Codestriker 支持邮件通知，还可以同版本控制服务器进行集成，以跟踪和显示文件内容的改变。图 19 显示了 Codestriker 的界面。

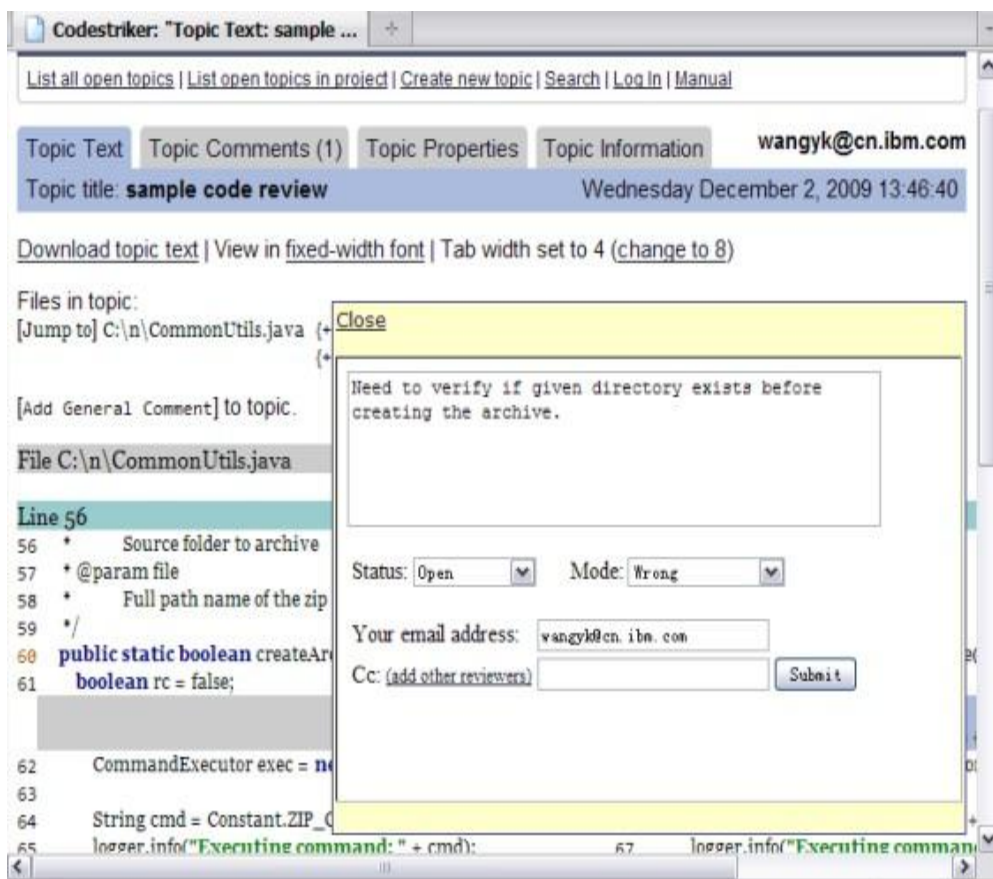


图 19.Codestriker 报告界面

在实践中对所有代码进行小组评审会比较费时，所以可以根据实际情况来挑选一些核心代码进行小组评审，或者在项目的前期安排较多的小组评审，等项目组的成员对代码评审的标准和要求有较好的理解，进行代码评审的经验提高后，就可以逐渐减少小组评审的次数，从而达到大部分代码即使只进行同级评审也能保证很好的质量。

通过代码评审发现的问题要通过代码重构及时解决掉，较小的不涉及多人代码的重构可以由项目成员自己借助 **Eclipse** 的重构功能完成，不同项目成员写的实现相同功能的不同代码要通过讨论整合成公共的类或者方法。比较复杂的或者比较高层次的重构工作，例如整个项目层面的代码组织形式的改变需要由整个项目组共同讨论完成。

## 结论

软件开发没有一成不变、万能通用的流程和方法，希望大家能从本文得到启发和收益，结合您的实际项目特点，实践以上步骤和方法，并加以完善和改进，共同打造高效高质量的 **Java** 代码，为您的项目成功奠定坚实的基础。