文章编号：1672-5913(2016)05-0100-04 中图分类号：G642

基于实时重构的代码质量教育

李光杰[1,2]，牛振东[1]，刘 辉[1]
（1. 北京理工大学 计算机学院，北京 100081 ；2. 北京工业大学 耿丹学院，北京 101301）

摘 要：介绍软件代码质量的重要性以及当前代码质量教育方面工作的不足，提出将代码坏味检测工具引入程序设计课程实践教学可以提高代码质量教育的观点。文章设计一个实验方案对基于传统说教和代码坏味检测工具两种方式的代码质量教育效果进行了对比研究，最后对实验结果进行分析和总结。

关键词：软件工程；代码质量；软件重构；实时监控；计算机教育
DOI:10.16512/j.cnki.jsjjy.2016.05.026

1 背 景

早在 30 年前，美国软件工程师 Brooks E P 就在《人月神话》一书中介绍了代码质量的重要性[1]。代码质量一直被企业看作提高竞争力的重要策略之一。提高代码质量不仅可以提高生产率，也可以提高用户的满意度。代码质量方面的专家和学者在如何提高代码质量方面做了大量工作，Sun Myung Hwang[2] 认为对软件工程师教育的缺失是造成代码质量问题的主要原因，因此他设计了软件过程和代码质量教育的大纲。

IEEE2014 对软件质量的定义如下：软件质量是指软件产品满足预定需求的程度[3]。鉴于代码质量的重要性，人们提出了多种方法来衡量代码的质量。评价代码质量的指标一般包括可信性、完整性、可用性、有效性、可维护性、灵活性、重用性等[4]。为了提高代码质量，人们提出了很多质量模型，如 Bansiya 和 Davis[5] 提出的 QMOOD 质量模型包括可重用性、灵活性、可理解性、功能型、可扩展性和有效性 6 个较高层次质量属性；Dromey R G[6] 提出的质量框架由影响质量的产品属性、高层质量属性以及连接产品属性和质量属性的方法 3 个主要元素组成；Jim McCall[7] 提出的 GE 质量模型利用产品修订、转变和操作特性 3 个中层视角来定义和识别软件产品的质量；Kaotao Ka[8] 等人关注重构代码的可维护性；Eva Van Emden[9] 等人研究通过检查代码坏味自动评估代码质量，同时开发了 jCOSMO 工具自动检测和提示代码坏味。

软件代码的质量对软件项目的效益、成本、生命周期等都有重要影响。因此，与编程相关的课程都投入了大量课时介绍代码质量的重要性以及提高代码质量的方法和注意事项。在计算机编程课堂上需要重点阐述命名规范的重要性，比如标识符名称尽量有意义、类型由大写字母开头、变量名由小写字母开头等。在软件工程课程上，教师通常要花大量的精力强调软件演化的必然性以及代码质量（尤其是代码的可扩展性）对软件演化的重要影响。但传统的说教方式教学效果并不理想，其主要原因包括：①教师在评定学生的成绩时主要看程序是否实现了预定的功能需求，忽视代码质量；②软件质量比较抽象，很难量化衡量比较；③学生以及部分教师缺乏足够的工程经验，难以理解代码质量的重要性。代码质量对软件项目的影响，需要足够的迭代演化周期才能变得更为明显，而学生往往没有任何工程经验，难以理解软件演化的必然性，也难以理解代码质量对软件演化的影响。

为此，我们提出基于代码坏味检测辅助代码质量教育。基于实时监控的代码坏味检测工具

基金项目：北京理工大学全日制硕士专业学位研究生课程建设重点项目（ZHSK2014-A02）；北京理工大学学位与研究生教育发展研究课题（2013Y08B03-011）；北京理工大学 2015 年度研究生教学团队项目（YJXTD-2015-A03）。
第一作者简介：李光杰，女，副教授，博士研究生，研究方向为软件工程，liguangjie_er@126.com。

通过实时检测学生编程时的不良设计，及时警告其中的代码质量问题。此外，该工具还可以提出可行建议解决这些不良设计。通过一段时间的训练，学生对代码的质量有一个感性的认识，知道什么样的代码是“坏”的设计，也知道如何避免这些坏的设计，从而培养出较好的编程习惯。最后，对比该方法与基于课堂说教的方式，表明前者的效果显著优于后者。

2 相关工作

2.1 软件重构

软件重构的概念由 OPDYKE W F 于 1992 年在其博士论文《重构面向对象框架》中首次提出，指在不改变代码外部行为的情况下，通过改善代码内部结构以提高代码质量，尤其是代码的可扩展性与可维护性 [10]。软件重构的基本流程是：首先确定哪里需要重构及应用哪种重构方法，在保证不改变软件原有行为的情况下执行重构，接下来评估重构对软件质量及成本等方面的影响，最后保证重构代码与文档、需求规范等内容的一致性 [11]。基本的重构包括移动方法、重命名方法、添加类、添加方法、提取方法和上移方法 6 种，其他复杂重构均可由这 6 种基本重构类型组合或变形得到。微软工程师 Kim 等人对软件重构的意义进行了定量分析，结果表明重构的价值明显 [12]。

2.2 代码坏味

软件重构的第一步是确定哪里需要重构。代码坏味就是用于检测哪里需要重构的一种重要技术。代码坏味（bad smells）的概念由 Martin Fowler 于 1999 年提出，用于表示需重构的代码片段。常见的代码坏味包括命名不合理、重复代码、长方法、大类、长参数列表、依恋、纯数据类、发散式变化、散弹式修改等 [13-15]。笔者主要以编程初学者常犯的、影响潜在代码质量的两种典型代码坏味（命名不合理和长方法）为例进行研究。

1）命名不合理问题。

通常来说，标识符占源代码总量的 70% [16]，选择能够反映代码功能的有意义的标识符名称有利于提高代码质量。如果标识符名称过于简单或无意义，则其他程序员很难理解或维护源代码。解决命名不合理问题的基本步骤为：提取标识符名称并分割为单词，然后对单词进行词性标记，利用解析器分析语法，最后利用相关技术进行语义分析并找出不符合常规的命名。例如，对于标识符 displayMenuBar，首先用单词分割工具 INTT 将其分割为 display、Menu 和 Bar 3 个单词，然后利用词库 WordNet 将这 3 个单词分别标记词性为 VB、NN 和 NN，再利用语法解析工具 Stanford Parser 抽象动名词短语树。

2）长方法问题。

程序的方法体越长越难以阅读和理解，出现问题时也不易发现和处理。在编程过程中，如果觉得要对某些代码进行注释，则可能是因为方法体过长。此时最好将需要注释的代码抽取成一个新的方法，并根据这段代码的功能对新方法进行合理命名。

2.3 实时代码坏味检测

代码坏味检测是代码重构工作的第一步，研究人员提出了多种自动或半自动代码坏味检测的方法。例如，Travasssos 等人提出了一系列手动标识代码坏味的阅读技术 [17]，Tourwe 等人利用 SOUL 语言定义了逻辑规则作为检测代码坏味的算法 [13]，Moha 等人提出了 DSL 语言指定代码坏味 [14]，Munro 提出了基于度量的 Java 代码坏味检测方法 [15]。流行的代码坏味检测工具有 Checkstyle、PMD、FindBugs 等。

但目前所有的代码坏味检测工具都是独立提供的，需要软件工程师手动调用，无法推动开发人员主动重构，尤其是没有经验的编程初学者。由刘辉等人 [18] 开发的实时代码检测工具 InsRefactor 能够及时检测并提示代码坏味问题，从而驱动开发人员主动解决。笔者应用的 InsRefactor 实时代码检测工具由监视器、代码坏味检测器、重构工具、反馈控制器以及坏味视图 5 部分组成，用于检测和提示 Java 代码坏味，如图 1 所示。在 Eclipse 中安装完 InsRefactor 插件并打开 Smell View 视图后，当出现代码坏味问题时，Smell View 视图将以红色字体逐条显示每个代码坏味的所属类型及相应的问题描述。当双击 Smell View 视图中的某条具体提示信息时，光标将自动定位到代码编辑窗口中的相应位置。

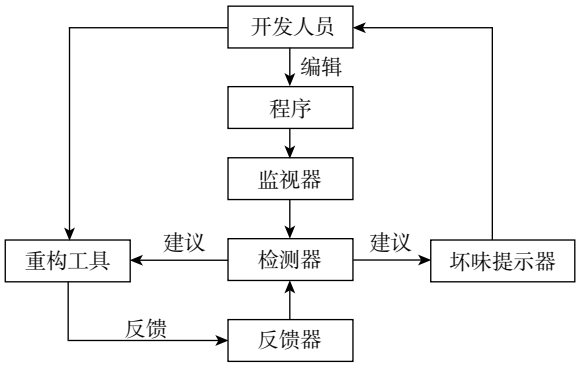

图 1　InsRefactor 工具工作原理图

3　实验设计

笔者的主要研究目标是确定实时监控工具在提高代码质量方面是否比传统说教方式更有效，因此，我们设计了一个实验来对比这两种方法的效果。

3.1　参与对象

在选取参与者时，我们希望参与者有一定的编程基础，但都是 Java 语言的初学者，因此所有参与者均为计算机专业二年级学生。我们事先检查了所有学生之前的 C 语言笔试成绩和学生提交的 C 语言课程实验报告，然后选取 10 名笔试成绩在 80~85 分之间且实验报告成绩为良好的学生作为实验参与者，并随机将这 10 名学生平分到实验组和控制组。

3.2　实验过程

控制组采用传统的说教模式。本实验为控制组提供了一个视频文件用于介绍常见代码坏味问题、危害及相应解决方法，并告诉学生在软件开发过程中可以随时观看视频，尽可能避免出现类似的代码坏味问题。

实验组采用基于实时代码坏味检测的实践教学模式。首先简单介绍实时监控代码检测工具，然后介绍具体的使用方式。

接下来，我们设计一组 Java 程序开发任务。考虑到实验参与者是 Java 初学者，因此实验任务的代码量不是很大，要求学生开发一个基于命令行界面的售票管理系统。系统主要包括班次管理、售票管理和退票管理三大功能。其中，班次管理包括增、删和查询功能；查询模块设定 3 种查询条件；售票和退票管理根据用户输入的班次和票数实现售退票功能。实验任务的代码量约 700 行，额定编程时间为 6 小时。

4　结果和分析

4.1　实验结果

在学生提交的程序中，实验组平均编程时间为 4.5 小时，控制组平均编程时间为 5 小时。我们仅对命名不合理和长方法这两类代码坏味进行了统计和分析。实验组共出现代码坏味 34 个，命名不合理问题 33 次，长方法问题 1 次；控制组共出现的代码坏味 264 个，其中命名不合理问题 255 次，长方法问题 9 个，具体数据见表 1。

表 1　控制组和实验组编程时间及代码坏味总量

分组	平均编程时间	代码坏味总个数
控制组	5	264
实验组	4.5	34

4.2　实验分析

通过分析，我们得出以下 3 条结论：

(1) 实时代码坏味检测工具可以有效地解决命名不合理和长方法两种代码坏味。命名不合理和长方法是初学者常出现的问题，借助实时代码坏味检测工具，Java 初学者出现这两个问题的数量明显减少，其中命名不合理降低 88.2%，长方法问题降低 88.8%。

(2) 基于实时代码坏味检测工具的实践式教学有助于初学者建立良好的编程习惯。实验组的平均编程时间比控制组减少 10%，说明借助实时代码坏味检测工具，实验组能更好地设计程序，抽取和分解长方法，降低整体开发时间。

(3) 基于实时代码坏味检测工具的实践式教学可以有效地提高代码质量教育效果。对比数据不难发现，传统说教式很难在代码质量教育方面起作用。原因在于初学者一般以 IDE 不报错为标准，很少自觉检查代码坏味问题；而当重构工具实时提示代码坏味问题时，他们会不断思考原因并尽量避免坏味问题的出现，从而很好地实现代码质量教育目标。

4.3　讨论

实验的局限性包括以下两个方面：

1) 实验方案相对简单。

如果实验任务过于简单或者参与者非常熟悉，则产生的代码坏味数量减少；如果实验任务太难，则不适合初学者。考虑到实验者都是 Java 初学者，我们设计的任务代码量均在 600~800 行之间，编程难度不大，只能检测并辅助解决命名不合理和长方法这两类代码坏味问题，无法对大类、特征依赖、散弹式修改等类型的代码坏味问题实施质量教育。笔者正在考虑对具有良好 Java 知识基础的学生进行基于实时代码检测工具的大规模程序实践式教学，从而观察在包含更多类型代码坏味问题时的质量教育效果。

2）实验规模相对较小。

为了保证实验参与者具有相同的编程基础，我们主要以 C 语言笔试成绩和实验报告成绩为依据，选取笔试成绩为 80~85 和实验成绩良好的学生作为实验参与者。由于客观条件的限制，实验共选取 10 名参与者；也由于实验规模较小，结果数据可能会偏高。笔者将设计更大规模的实验对此进行验证和完善。

5 结 语

我们通过两组对比实验研究传统说教方式和基于实时代码坏味检测的实践式教学在代码质量教育中的作用。实验结果表明，软件重构工具在代码质量教育中起到明显作用，使命名不合理和长方法两类代码坏味问题分别降低 88.2% 和 88.8%，有助于培养初学者良好的编程习惯。同时，由于受实验数据量、实验方案代码量以及实验参数设置等因素的影响，我们仅研究命名不合理和长方法两类代码坏味问题，笔者将在下一步工作中继续探索将 InsRefactor 重构工具引入程序设计类课程的实践教学以及有效地解决更多类型的代码坏味问题。

参考文献：

[1] Frederick P.Brooks J.人月神话[M].北京:清华大学出版社,2002:68-70.
[2] Sun M H. Essential contents for software development process and software quality education[J].Engineering Systems Modelling and Simulation,2014(6):1-2.
[3] IEEE Computer Society.IEEE Standard for Software Quality Assurance Processes[EB/OL].[2016-03-28].http://www.doc88.com/p-7065214863897.html.
[4] Yas A. Alsultanny, Ahmed M. Wohaishi. Requirements of Software Quality Assurance Model[C].Second International Conference on Environmental and Computer Science.2009:19-23.
[5] Bansiya J,Davis C G.A hierarchical model for object oriented design quality assessment[EB/OL].[2016-03-28].http://www.doc88.com/p-6981861681596.html.
[6] Dromey R G.A model for software product quality[EB/OL].[2016-03-28].http://www.doc88.com/p-5465210569273.html.
[7] McCall J A,Richards P K,Walters G F. Factors in software quality[EB/OL].[2016-03-28].https://www.researchgate.net/publication/235031005_Factors_in_Software_Quality_Volume_I_Concepts_and_Definitions_of_Software_Quality.
[8] Kataoka Y,Imai T.A quantitative evaluation of maintainability enhancement by refactoring[EB/OL].[2016-03-28].http://link.springer.com/chapter/10.1007%2F978-3-540-85279-7_20.
[9] Emden E V,Moonen L.Java Quality Assurance by Detecting Code Smells[C].Working Conference on Reverse Engineering.2002:95-97.
[10] Opdyke W F.Refactoring object-oriented frameworks[D].Illinois:University of Illinois at Urbana-Champaign,1992:18-35.
[11] Fowler M,Beck K,Opdyke W.Refactoring: Improving the design of existing code[M].London:Addison-Wesley Professional,1999:46-59.
[12] Kim M,Zimmermann T.A field study of refactoring challenges and benefits[EB/OL].[2016-03-28].http://citeseerx.ist.psu.edu/showciting?cid=15067308.
[13] Tourwe T,Mens T.Identifying refactoring opportunities using logic meta programming[EB/OL].[2016-03-28].http://www.doc88.com/p-7394513234508.html.
[14] Moha N,Guhneuc Y G.Decor:A method for the specification and detection of code and design smells[J].Software Engineer ,2010,36(1):20-36
[15] Munro M.Product metrics for automatic identification of ‘ Bad Smell ’ design problems in Java source-code[EB/OL].[2016-03-28].https://www.researchgate.net/publication/4175576_Product_Metrics_for_Automatic_Identification_of_Bad_Smell_Design_Problems_in_Java_Source-Code.
[16] Deissenboeck F,Pizka M.Concise and consistent naming[J].In Software Quality Control,2006,14(3):261-282.
[17] Travassos G,Shull F,Fredericks M.Detecting defects in object-oriented designs: Using reading techniques to increase software quality[EB/OL].[2016-03-28].http://doi.acm.org/10.1145/320384.320389.
[18] Liu H,Guo X.Monitor-based instant software refactoring[J].Software Engineer ,2013,39(8):1112 - 1126.

（编辑：孙怡铭）