克隆代码分析方法研究 *

王克朝[1, 2a]，朱宸光[2b]，王甜甜[2a]，苏小红[2a]

(1. 哈尔滨学院 软件学院，哈尔滨 150086；2. 哈尔滨工业大学 a. 计算机科学与技术学院；b. 软件学院，哈尔滨 150001)

摘 要：针对已有克隆代码检测工具只输出克隆组形式的检测结果，而难以分析克隆代码对软件质量的影响的问题，提出了危害软件质量的关键克隆代码的识别方法。首先，定义了克隆代码的统一表示形式，使之可以分析各种克隆检测工具的检测结果；接下来，解析源程序和克隆检测结果，识别标识符命名不一致性潜在缺陷；然后，定义了克隆关联图，在此基础上检测跨越多个实现不同功能的文件、危害软件可维护性的克隆代码；最后，对检测结果进行可视化统计分析。本文的克隆代码分析工具被应用于分析开源代码 httpd，检测出了 1 组标识符命名不一致的克隆代码和 44 组危害软件可维护性的关键克隆类，实验结果表明，本文方法可以有效辅助软件开发和维护人员分析、维护克隆代码。

关键词：克隆代码；克隆代码分析；克隆代码维护；缺陷检测

中图分类号：TP311.5

Research on code clone analysis approach

Wang Kechao[1, 2a], Zhu Chenguang[2b], Wang Tiantian[2a], Su Xiaohong[2a]

(1. School of Software, Harbin University, Harbin 150086, China; 2. School of Computer Science & Technology, Harbin Institute of Technology, Harbin 150001, China; 3. School of Software, Harbin Institute of Technology, Harbin 150001, China)

Abstract: Most existing clone detection tools only output the detection results in the form of clone sets, lacking clone analysis. To solve this problem, we proposed a key clone recognition approach. The approach can analysis the clones which decrease the quality of software. Firstly, our approach defines a unified clone representation form to support the analysis of various clone detection tools. Then, the approach analysis the source code and clone detection results to detect identifier renaming inconsistent clones, which are potential bugs. Next, the approach detects the clones diffusing in various functional different files, which may decrease the maintainability of software. Finally, the approach visually analysis the detection results. The proposed analysis tool has analyzed the open source code httpd, which detected 1 identifier renaming inconsistent clone, and 44 clones diffusing in various functional different files. The experimental results show that it can facilitate the analysis and maintenance of code clones.

Key Words: code clones; clone analysis; clone maintenance; bug detection

0 引言

研究表明，在大型软件系统中克隆代码（也称重复代码、拷贝-粘贴代码）约占代码总量的 7-23%[1]。

从软件维护的角度而言，克隆代码可以分为三类：第一类是出于软件可重用的角度引入的，这些克隆代码是良好软件设计的体现，是无害的；第二类是导致软件缺陷的克隆代码，例如在拷贝-粘贴-修改的过程中忘记或错误地修改了某些变量等[2-4]；第三类是虽然没有直接导致软件缺陷，但是会影响软件的可维护性的克隆类，例如在采用模型-视图-控制器（Model View Controller, MVC）模式进行系统设计时，如果模型子系统中和其它子系统间存在克隆代码，则意味着它们的业务逻辑是相交的，不满足 MVC 分层、独立的要求。如果要修改模型子系统，则需要修改、重新编译视图和控制器子系统。因此这种跨越多个实现不同功能的系统的克隆代码会危害系统的可维护性。

可见，后两类克隆代码直接危害软件的质量，因此需要在检测出所有克隆代码后，进一步识别克隆代码所属的类别，有针对性的指导软件开发和维护人员对其进行维护。如修正软件缺陷，重构危害软件可维护性的克隆代码。

目前已有数目众多的克隆代码检测工具[5,6]，但是这些工具

基金项目：国家自然科学基金资助项目（61202092，61173021）；高等学校博士学科点专项科研基金资助项目（20112302120052）；黑龙江省普通高校青年学术骨干项目（1254G037）；黑龙江省自然科学基金资助项目（F201127）；哈尔滨科技创新人才研究专项资金项目（RC2013QN010001，2014RFQXJ062）

作者简介：王克朝（1980-），男，河南南阳人，副教授，博士研究生，主要研究方向为程序分析、软件错误定位；朱宸光（1992-），男，山东临沂人，本科生，主要研究方向为克隆代码检测；王甜甜（1980-），女，辽宁丹东人，副教授，博士，主要研究方向为软件自动化调试、计算机辅助教学；苏小红（1966-），女，哈尔滨人，教授，博士，主要研究方向为程序分析、图像处理.

通常缺少对克隆检测结果的进一步分析。本文研究克隆代码相关缺陷及危害软件可维护性的克隆代码的检测算法，开发了克隆代码分析工具。

1　相关研究

目前已有的克隆代码检测工具，如 IClones[7]、CCFinder[8]、NiCAD[9]、PMD/CP、Simian、CPbugdetector[10]、CMGA[11]等，为克隆代码的检测奠定了良好的基础，但仍存在如下问题：

（1）大多以文件的形式输出克隆代码组，而对克隆代码的属性及分布缺少统计分析。

（2）大多不支持克隆代码相关的缺陷检测。

（3）不支持分析克隆代码是否危害软件可维护性。

Viscad[12]、SolidSDD[13]这些工具研究了克隆代码检测结果的可视化及分析技术，但它们不能检测克隆代码相关的缺陷，也无法识别危害软件可维护性的克隆代码。

综上所述，孤立的检测结果无法为克隆代码的维护提供有效指导。只有在分析各个克隆代码之间的分布和关联关系之后，才能识别出那些真正具有危害性的克隆代码，从而帮助软件开发人员有针对性地分析、维护克隆代码。

具体地说，克隆代码分析还需解决以下问题：

（1）如果能直接利用各种已有克隆代码检测工具的检测结果，则有助于提高克隆代码的分析的效率及准确性，但是存在的一个难点问题是“各个工具的检测结果表示形式都不相同，如何统一对其进行表示，以使后续的分析具有通用性？”

（2）源代码片段中某些标识符在拷贝-粘贴代码中可能被忘记或错误修改，如何检测这种缺陷？

（3）如何识别危害软件可维护性的克隆代码？

（4）如何支持检测结果的可视化统计分析？

2　克隆代码分析框架

为了解决以上问题，本文提出的克隆代码分析框架如图 1 所示。

首先，定义了克隆代码检测结果的统一表示形式（Unified Clone Representation, UCR），开发格式转换工具，将各种克隆检测工具的检测结果，都转换成统一的表示形式，从而可灵活配置克隆代码检测前端，可支持多种克隆检测工具，目前支持 IClones、CCFinder、NiCAD、PMD/CP、Simian、CPbugdetector。

接下来，对 UCR 形式克隆检测结果进行分析，识别如下两种危害软件质量的克隆代码。

一是检测容易导致软件缺陷的克隆代码，解析克隆类及相关的源程序信息，进行标识符命名不一致性的检测，输出可能含有软件缺陷的克隆类，为开发人员修正程序提供参考。

另一种是识别危害软件可维护性的关键克隆类，定义了克隆关联图、关键克隆类等概念，在此基础上给出关键克隆代码的识别方法。

最后，对检测结果进行可视化统计分析，以图的形式直观的给出克隆类、相关缺陷及关键克隆代码的统计信息，辅助开发和维护人员分析克隆代码。

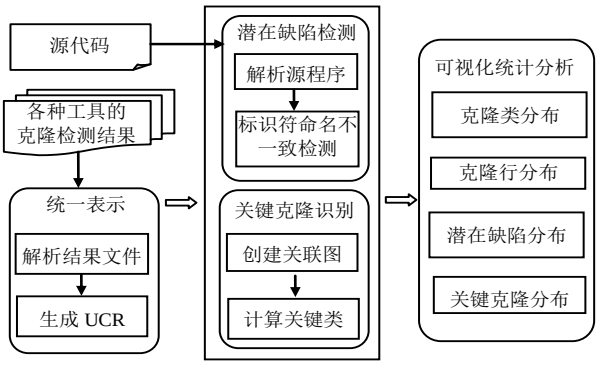

图 1 克隆代码分析框架

3　关键技术

3.1　克隆代码检测结果的统一表示形式

定义克隆代码检测结果的统一表示形式（UCR）如图 2 所示。

其中 CloneClass 表示克隆类，ClonePair 表示克隆代码对，一个克隆类由 2 个以上的 Clone 构成，一个克隆代码对由 2 个 Clone 构成，一个 Clone 可以包含多个不连续的克隆代码片段 Fragment，每个代码片段有起始位置和结束位置以及源代码文件路径，同一个 Clone 中相邻 Fragment 间的代码行是 Gap。这使得 UCR 表示法不但可以表示连续的克隆代码片段，还可以表示删除了语句的克隆代码片段。因此无论是何种类型的克隆代码检测工具的检测结果，只要结果中包含克隆类/对以及克隆代码的起始和结束位置信息，均可转换为 UCR，基于这种统一表示形式可以方便地自动分析各个工具的检测结果。

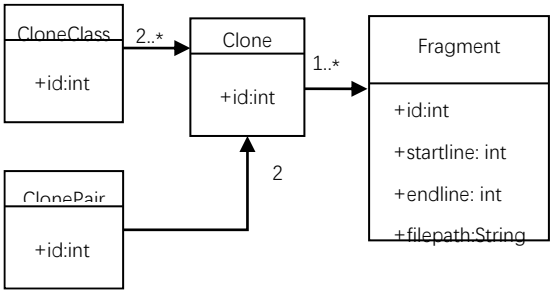

图 2 克隆代码检测结果的统一表示形式

3.2　克隆代码相关缺陷检测

定义（标识符命名不一致） 给定克隆代码对 $\text{ClonePair} < Clone1, Clone2 >$、$\forall Idx \in Clone1$，$Clone2$ 中与标识符 Idx 对应的标识符的集合记为 $Mapset(Idx) = \{Idy \mid Idy \in Clone2, Map(Idx, Idy)\}$，其中 $Map(Idx, Idy)$ 表示标识符 Idx 和 Idy 在克隆代码中具有相同的位置。

如果 $|Mapset| \geq 2$，即 Idx 在拷贝-粘贴代码中的所有出现位置上被修改成了两个或两个以上不同的名称，则称标识符 Idx 命名不一致。

标识符命名不一致性通常伴随着软件缺陷，例如，如果标识符 Idx 在拷贝-粘贴代码的所有出现位置，除了一两处跟原名称相同，剩下的位置都被命名为另一个名称 Idy，则意味着很可

能开发人员想把拷贝代码中的 *Idx* 都改为 *Idy*，但是却忘记修改了某处。还有一种情况是在粘贴代码后，错误的将某个标识符命名为其它标识符。

标识符命名不一致检测算法描述如下。

输入：源程序、UCR 格式的克隆检测结果

输出：标识符命名不一致列表 List

$$UnchangedRatio(Idx) = \frac{num(Idy = Idx)}{\sum_{\forall Idy \in Mapset(Idx)} num(Idy)}$$

1：词法分析程序，识别标识符信息。

2：Foreach ClonePair $< Clone1, Clone2 >$

{

解析 token 流，匹配 $Clone1$ 和 $Clone2$ 中的标识符;

foreach Idx in $Clone1$

{

$Mapset(Idx) = \{(Idy, num) \mid Idy \in Clone2, Map(Idx, Idy)\}$

if ($UnchangedRatio(Idx) > Threshold$)

$\text{List} = \text{List} \cup \{< Idx, Clone1, Clone2 >\}$

}

}

在获得两个 Clone 中的标识符映射关系后，用 $UnchangedRatio(Idx)$ 判定标识符 *Idx* 是否存在命名不一致性，其中 $num(Idy)$ 是标识符 *Idy* 在映射中出现的次数，$num(Idy = Idx)$ 是 *Idx* 没有被重新命名的次数，$\sum_{Idy \in Mapse(Idx)} num(Idy)$ 是 *Idy* 在映射中出现的总次数。

3.3 关键克隆代码的识别算法

软件开发人员经常将实现同一功能的程序代码写在同一个文件中，各个不同的文件间应具有独立性。而跨越多个实现不同功能的文件的克隆代码妨碍了这种独立性，不利于软件维护[14]。因此，本文关键克隆代码识别算法的目标是检测跨越多个实现不同功能的文件中的克隆类。

如图 3 所示，克隆类 1、4、5、6、7、8、9 由于其所有克隆代码片段都在同一个文件中，可以认为它们仅在同一个子系统内有克隆代码，没有跨越不同子系统间的克隆代码，所以判断它们是出于软件重用角度考虑引入的克隆代码片段，是无害的克隆代码；而克隆类 2 和 3 则跨越了多个实现不同功能的文件，其克隆代码片段在多个子系统内有分布，这样的克隆代码可能会危害软件的可维护性，它们属于要识别的关键克隆代码。

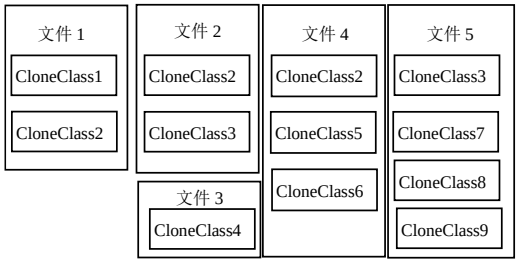

图 3 克隆代码类-文件分布图示例

本文定义了克隆关联图，抽象表示克隆代码类在文件中分布的关联关系，在此基础上检测关键克隆类，相关定义如下。

定义（关联的克隆类）如果克隆代码类 *A* 包含的某个代码片段与另一克隆类 *B* 包含的某个代码片段在同一个文件中，则称 *A* 和 *B* 是关联的克隆类。

定义（克隆关联图）将克隆类以及它们之间的关联关系抽象成一个无向图 *G*=<*V*,*E*>，*V* 中的每个节点表示一个克隆类，对于节点 *A* 和 *B*，如果他们是关联的克隆类，则在 *G* 中用一条边将节点 *A* 和 *B* 相连,即 $AB \in E$。

图 3 的克隆关联图如图 4 所示。由于克隆类 2 的其中一个片段和克隆类 1 的片段在同一个文件 1 中，所以把节点 1 和节点 2 用一条边相连；由于克隆类 2、5、6 都有一个片段在文件 4 中，所以将这几个节点两两互连，其他节点以此类推，就生成了这个克隆关联图。克隆类 4 不和任何其他克隆类关联，因此没有表示在关联图中。

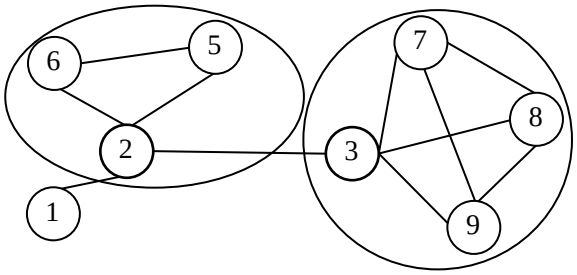

图 4 克隆关联图示例

定义（关联环） $CR \subset G$ 是克隆关联图 *G* 中节点数大于 2 的子图，如果 *CR* 中的所有节点两两关联，且不存在其它关联环 *CR*′满足 $CR \subset CR'$，则称 *CR* 是 *G* 的一个关联环。

图 4 中，2-5-6，3-7-8-9 是两个关联环。关联环中的节点位于相同的文件中，共同实现某项功能。

定义（关键克隆类）如果克隆关联图 *G*=<*V*,*E*>中的节点 $v \in V$ 连接了 2 个及以上的关联环，则称 v 为关键节点，v 对应的克隆类为关键克隆类。

关键克隆类的识别算法如下。

输入：UCR 形式的克隆检测结果 *UCRf*

输出：关键克隆类列表 *CloneList*

1: *G*= ParseCloneResult(*UCRf*);

//解析 *UCRf*，创建克隆关联图 *G*

2: foreach node *s* in *G*

{//计算 *V* 中各节点最短路径

foreach node *t* in G

calculate $Djstr(s,t)$;

}

3: foreach node *v* in *V*

{//用 Betweenness Centrality 判定关键节点

calculate $BC(v) = \sum_{s \neq v \neq t \in V} \frac{Djstr_{st}(v)}{Djstr(s,t)}$

if($BC(v) > threshold$)

$$CloneList = CloneList \cup \{v\}$$

}

通过计算克隆关联图中的各个节点 v 的 Betweenness Centrality（记为 $BC(v)$）来判定其是否是关键节点。$BC(v)$ 表示所有的节点对之间通过节点 v 的最短路径条数与所有节点对之间最短路径总条数的比值的总和。衡量了节点 v 对于整个图的连通性的贡献程度。计算公式如下。

$$BC(v) = \sum_{s \neq v \neq t \in V} \frac{Djstr_{st}(v)}{Djstr(s,t)} \quad (1)$$

$Djstr(s,t)$ 表示从节点 s 出发到达节点 t 的最短路径总条数，$Djstr_{st}(v)$ 表示从节点 s 出发到达节点 t 且经过节点 v 的最短路径条数。

例如，对图 4 计算得到 BC(2)=25,BC(3)=20 明显高于其它节点的 BC 值（均为 0），因此克隆类 2 和 3 被识别为关键克隆类。

4　克隆分析工具的应用

开发了克隆代码分析工具，并对开源代码 httpd2.2.2，共 737 个文件，280,792 代码行（http://www.apache.org/），进行了实验分析。

克隆分析工具目前支持主流的克隆代码检测工具，包括 IClones、CCFinder、NiCAD、PMD/CP、Simian 和 CPbugdetector。对于其它新的克隆代码检测工具，可通过扩展解析前端的格式转换工具，将新工具的检测结果转换为 UCR 表示形式即可。本实验分析基于 CPbugdetector 工具的克隆代码检测结果。

4.1　标识符命名不一致的检测结果

在\srclib\apr\locks\unix\proc_mutex.c 文件中检测出一组标识符命名不一致的克隆类，如图 5 所示。左侧代码中的标识符“rv”，在右侧代码中 2、7、8 行保持了原名字“rv”，但是在第 9 行被重命名为“errono”。将该结果提供给开发人员，以确认是否含有“错误修改了标识符缺陷”。

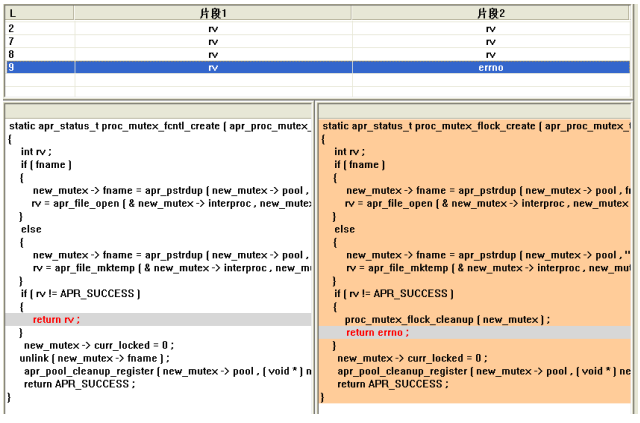

图 5 httpd2.2.2 中的标识符命名不一致检测结果

4.2　可视化统计分析

包括克隆代码片段的数量及分布统计、克隆代码行数统计、标识符命名不一致统计、关键克隆类统计。例如，httpd2.2.2 克隆代码片段的分布如图 6 所示，克隆代码行的分布如图 7 所示，可以方便地获知 core.c 文件的代码克隆情况最严重。

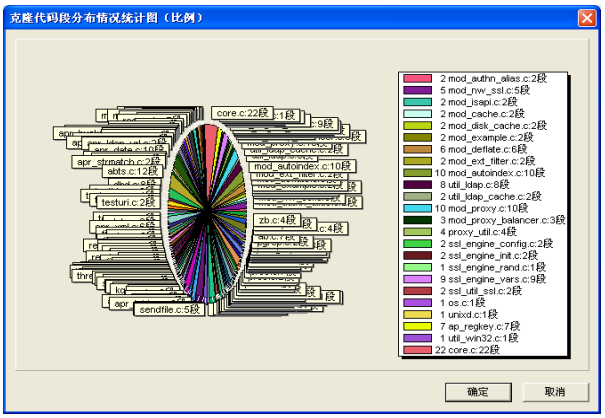

图 6　克隆代码片段分布统计

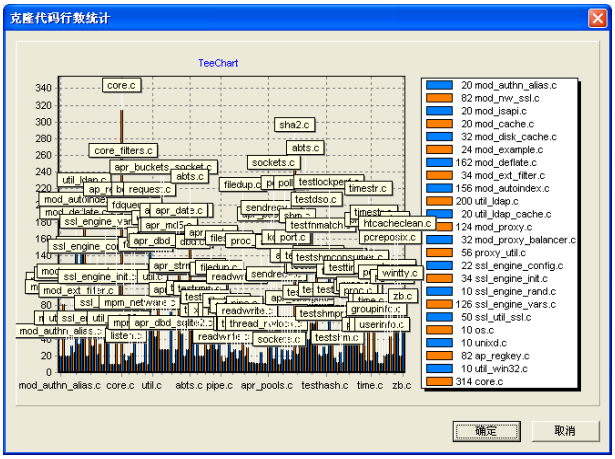

图 7　克隆代码行分布统计

4.3　关键克隆代码的识别

在检测出的 httpd2.2.2 的 201 组克隆类中识别了 44 组关键克隆类。通过分析关键克隆类发现，server\mpm 的子目录\worker 和\experimental \event 目录下均含有 fdqueue.c 文件，且这两个文件中的大部分代码都是关联克隆类。还有多个克隆类跨越了 srclib\apr\user\win32\ 路径下的 userinfo.c 和 groupinfo.c 文件。软件维护时可对这两个文件提取公共的函数，进行重构。

5　结束语

本文从软件维护的角度，将克隆代码分为三类，定义了关联克隆图、关键克隆类等概念，并在此基础上提出了危害软件质量的潜在缺陷和关键克隆代码的识别方法，开发了克隆代码分析工具。解决了已有克隆检测工具无法为软件维护提供有效参考的问题。

后续工作将进一步研究多版本中克隆代码的演化分析和维护。

参考文献

[1] Roy C K, Cordy J R. An empirical study of function clones in open source software[C]//Proc of the 15th Working Conference on Reverse Engineering. 2008: 81-90.

[2] Li J, Ernst M D. CBCD: Cloned buggy code detector[C]// Proc of the International Conference on Software Engineering. 2012: 310-320.

[3] Jiang L, Su Z, Chiu E. Context-based detection of clone-related

bugs[C]//Proc of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. 2007: 55-64.

[4] Li Z, Lu S, Myagmar S, *et al*. CP-Miner: finding copy-paste and related bugs in large-scale software code[J]. IEEE Trans on Software Engineering, 2006, 32(3): 176-192.

[5] Roy C K, Cordy J R, Koschke R. Comparison and evaluation of code clone detection techniques and tools: a qualitative approach[J]. Science of Computer Programming, 2009, 74(7): 470-495.

[6] Wang T, Harman M, Jia Y, *et al*. Searching for better configurations: a rigorous approach to clone evaluation[C]//Proc of the 9th Joint Meeting on Foundations of Software Engineering. 2013: 455-465.

[7] Gode N, Koschke R. Incremental clone detection[C]//Proc of the 13th European Conference on Software Maintenance and Reengineering. 2009: 219-228.

[8] Kamiya T, Kusumoto S, Inoue K. CCFinder: a multilinguistic token-based code clone detection system for large scale source code[J]. IEEE Trans on Software Engineering, 2002, 28(7): 654-670.

[9] Roy C K, Cordy J R. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization[C]//Proc of the 16th IEEE International Conference on Program Comprehension. 2008: 172-181.

[10] 王倩. 基于序列挖掘的 C 克隆代码及相关软件缺陷的检测[D]. 哈尔滨, 哈尔滨工业大学, 2009.

[11] Wang T, Wang K, Su X, *et al*. Detection of semantically similar code[J]. Frontiers of Computer Science, 2014, 8(6): 996-1011.

[12] M. Asaduzzaman. Visualization and Analysis of Software Clones[D]. Saskatoon: University of Saskatchewan, 2012

[13] Voinea L, Telea A C. Visual Clone Analysis with SolidSDD[C]// Proc of the 2nd IEEE Working Conference on Software Visualization. 2014: 79-82.

[14] Fukushima Y, Kula R, Kawaguchi S, *et al*. Code clone graph metrics for detecting diffused code clones[C]//Proc of the Asia-Pacific Software Engineering Conference. 2009: 373-380.