Soot 开源项目的软件演化过程分析实验报告

组员：MF1833060-沙猛 MF1833061-沈思远

我们通过分析大型开源软件在开发过程中的不同版本间代码的相似度来探索软件演化规律。

1 实验设计

我们的实验对象选择的是 Soot。Soot 是一个针对 Java 代码的优化框架，可以用于分析和转换 Java 字节码。然后通过构造软件不同版本下的代码抽象语法树进行代码的相似度比对，并通过 4 个研究问题阐述我们本次实验的研究发现。

如上所述，我们需要构建 java 代码的抽象语法树，并且进行相似度比对，主要的实验方法可以分为以下两个阶段：

1. 抽象语法树生成：针对项目中的每个代码模块，生成该模块对应的抽象语法树
2. 代码相似度评估：为了能够进行代码结构的相似度比较，我们需要将抽象语法树转化为容易处理和比较的字符串，对于任意两个抽象语法树转化后得到的字符串，我们选择使用编辑距离来体现模块的相似度

1.1 抽象语法树生成

首先我们需要将项目分成若干模块，我们分析的源代码根目录为 soot/src/main/java。规定在根目录的任一包含.java 文件的子文件夹组成模块（**dir/*.java**），也即一个 package 中的所有.java 文件构成一个模块，然后对于模块内的每一个文件，使用 JDT 生成对应的 AST。

1.2 代码相似度评估

然后我们将按照 JDT 的 AST 类中定义不同的节点类型，给每类 AST 节点赋值一个字符，然后通过前序遍历的方法将遍历过程中遇到的每个节点的字符值拼接起来，生成的即为该 java 文件的字符串。由于 Soot 代码结构较为复杂且代码量较大，导致每个 java 文件生成的对应字符串均较长。如果此时将模块内所有的 AST 字符串拼接起一个超长字符串，然后进行不同版本间的字符串编辑距离的计算，极易造成程序运行出现内存不足的情况。为了降低计算字符串编辑距离的时间和空间复杂度，我们首先计算模块内每个 AST 字符串在不同版本间的编辑距离，相加后得到模块的总距离，然后除以所有 AST 的字符串拼接后的字符串总长度。对于模块m，不同版本间相似度计算公式如下所示：

$$Similarity = 1 - \frac{\sum_{i \in m} editDistance(AST_{a,i}, AST_{b,i})}{\sum_{i \in m} max(AST_{a,i}, AST_{b,i})}$$

其中i表示模块内的文件计数，$AST_{a,i}$表示表示 java 文件i在版本a中生成的 AST 字符串，$editDistance$为计算字符串编辑距离的函数，max函数返回两个字符串的最大长度。$Similarity$为最终计算出的模块相似度。

2 实验评估

由于 Soot 的项目历史较长，其中的项目架构在项目演化过程中进行了很多变化，这给我们划分出 Soot 的模块带来了很大麻烦，为了方便进行模块分析和实验效率的考虑，我们选择了 Soot 项目的最近 938 次 commit 作为我们研究 commit 的对象，并且我们收集到了 31 个 Soot 发布版本作为我们探索项目在不同发布版本演化规律的对象，经过模块划分，

我们划分出 132 个模块。接下来我们将通过对 4 个研究问题的讨论进行实验结果评估。

2.1 RQ1：连续的 commit 间，模块 AST 的变化有多大?

	package	0	0.0~0.8	0.8~0.85	0.85~0.9	0.9~0.91	0.91~0.92	0.92~0.93	0.93~0.94	0.94~0.95	0.95~0.96	0.96~0.97	0.97~0.98	0.98~0.99	0.99~1.0	1
2	soot	2	0	1	38	0	0	0	0	3	0	0	5	9	249	630
3	soot.asm	2	0	1	0	5	2	1	36	0	0	1	2	5	59	823
4	soot.baf	2	0	0	0	0	0	0	0	0	5	0	0	0	28	902
5	soot.baf.internal	2	0	0	5	0	0	0	0	0	0	0	0	0	0	930
6	soot.baf.toolkits.base	2	0	0	0	0	0	5	0	0	4	1	0	0	7	918
7	soot.coffi	2	0	0	0	0	0	0	0	0	0	0	0	9	17	909
8	soot.dava	2	0	0	0	0	0	0	0	0	0	0	0	0	13	922
9	soot.dava.internal.asg	2	0	0	0	0	0	0	0	0	0	0	5	0	6	924
10	soot.dava.internal.AST	2	0	0	0	0	0	0	5	0	0	0	0	0	0	930
11	soot.dava.internal.javaRep	2	0	0	0	5	0	0	0	0	0	0	0	0	0	930
12	soot.dava.internal.SET	2	0	0	0	0	5	0	0	0	0	0	0	5	1	924
13	soot.dava.toolkits.base.AST	2	0	5	0	0	0	0	0	0	0	0	0	0	0	930
14	soot.dava.toolkits.base.AST.analysis	2	0	5	0	0	0	0	0	0	0	0	0	0	0	930
15	soot.dava.toolkits.base.AST.interProcedural	2	0	0	0	0	0	0	0	0	0	0	0	5	0	930
16	soot.dava.toolkits.base.AST.structuredAnalysis	2	0	0	0	0	0	0	0	0	0	0	0	5	0	930
17	soot.dava.toolkits.base.AST.transformations	2	0	0	0	0	0	5	0	0	0	0	0	0	3	927
18	soot.dava.toolkits.base.AST.traversals	2	0	0	5	0	0	0	0	0	0	0	0	0	0	930
19	soot.dava.toolkits.base.DavaMonitor	2	0	0	0	0	0	0	0	0	0	0	0	5	0	930
20	soot.dava.toolkits.base.finders	2	0	0	0	0	0	5	0	0	0	0	0	0	6	924
21	soot.dava.toolkits.base.misc	2	0	0	0	0	0	5	0	0	0	0	2	3	2	923
22	soot.dava.toolkits.base.renamer	2	0	0	0	0	0	0	5	0	0	0	0	0	0	930
23	soot.dexpler	2	0	0	0	0	0	0	0	0	0	3	15	12	73	832
24	soot.dexpler.instructions	2	0	0	7	0	0	0	0	0	0	0	0	0	37	891
25	soot.dexpler.tags	2	0	0	0	0	0	0	0	0	0	0	0	0	0	935
26	soot.dexpler.typing	2	0	0	0	0	0	0	0	0	0	0	0	0	5	930
27	soot.grimp	2	5	0	0	0	0	0	0	0	0	0	0	0	6	924
28	soot.grimp.internal	2	0	0	5	0	0	0	0	0	0	0	0	0	0	930
29	soot.grimp.toolkits.base	2	0	0	0	0	0	0	5	4	1	0	0	0	1	924
30	soot.javaToJimple	2	0	0	0	0	0	0	0	0	0	0	0	0	29	906

由于模块数过多（132），上图为部分模块数据，第一列为模块名，后面的 14 列为 14 个区间来分别计数不同程度的相似度。其中区间 0 和 1 比较特殊：0 表示模块完全不同，这通常发生在创建模块的 commit，由于之前模块不存在，任何非空模块与空模块的相似度均为 0；1 表示模块完全相同，这说明我们研究的 java 源代码在语法结构上没有发生任何变化，这中间可能发生了标识符改名、或者其他相关配置文件的更改，这些更改并未被我们被记录到。通过上图的实验数据发现对于每一模块而言，在绝大多数情况下，模块 AST 发生的变化极小（0.01），这与我们之前的直觉相符。因为复杂的软件开发过程从来就不是一蹴而就的事情，对于多人参与的大型项目开发过程来说，每次对项目的修复和完善过程都应该是较小的，方便在开发过程中出现问题时能够随时回溯到之前变化不大的 commit 斑斑中。

2.2 RQ2：连续的发布版本间，模块 AST 的变化有多大?

我们通过 git tag 收集到了 Soot 的 31 个版本，由于在版本变迁中代码根目录发生变化，由 soot/src（24 个发布版本）变为 soot/src/main/java（7 个发布版本），为了能够完整分析所有连续的发布版本间的模块 AST 变化，我们对处理的代码根目录进行了版本适应，然后进行相似度比较，部分结果如下图所示：

	release	0	0.0~0.1	0.1~0.2	0.2~0.3	0.3~0.4	0.4~0.5	0.5~0.6	0.6~0.7	0.7~0.8	0.8~0.9	0.9~1.0	1
1	soot	0	0	0	0	1	0	0	0	2	3	23	1
2	soot.asm	1	22	0	0	0	0	0	0	0	0	7	0
3	soot.baf	0	0	0	0	0	0	1	0	0	1	17	11
4	soot.baf.ir	0	0	0	0	0	0	0	0	0	1	12	17
5	soot.baf.to	0	0	0	0	0	0	0	0	0	3	7	20
6	soot.coffi	0	0	0	0	0	0	0	0	0	0	21	9
7	soot.dava	0	0	1	0	0	0	1	1	1	1	7	18
8	soot.dava.	1	3	0	0	0	0	0	0	1	0	5	20
9	soot.dava.	1	3	0	0	0	0	1	0	1	0	5	19
10	soot.dava.	1	3	0	0	0	0	0	0	3	0	6	17
11	soot.dava.	1	3	0	0	0	0	0	0	0	0	8	18
12	soot.dava.	1	3	0	0	0	0	0	0	0	1	3	22
13	soot.dava.	1	12	0	0	0	0	0	0	0	1	4	12
14	soot.dava.	1	14	0	0	0	0	0	0	0	0	2	13
15	soot.dava.	1	12	0	0	0	1	1	0	0	1	2	12
16	soot.dava.	1	12	0	0	0	0	0	2	0	0	5	10
17	soot.dava.	1	13	0	0	0	0	0	0	0	2	2	12
18	soot.dava.	1	3	0	0	0	0	0	0	0	1	2	23
19	soot.dava.	1	3	0	0	0	0	0	0	0	0	9	17
20	soot.dava.	1	3	0	0	0	0	0	0	0	0	9	17
21	soot.dava.	1	12	0	0	0	1	0	0	0	0	4	12
22	soot.dexpl	1	21	0	0	1	0	0	0	0	0	7	0
23	soot.dexpl	1	21	0	0	0	0	0	0	1	1	3	3
24	soot.dexpl	1	21	0	0	0	0	0	0	0	0	0	8
25	soot.dexpl	1	22	0	0	0	0	0	0	0	0	1	6
26	soot.grimp	0	0	0	0	0	0	0	0	1	0	8	21
27	soot.grimp	0	0	0	0	0	0	0	0	0	3	7	20
28	soot.grimp	0	0	0	0	0	0	0	0	1	0	5	24
29	soot.javaT	1	8	0	0	0	0	1	0	0	0	13	7

我们可以通过分析连续版本间的时间间隔以及相似度可以发现，相似度高的版本的时间间隔基本在 1 年之内，而且年代越久远的连续版本变化相比于最近的连续版本变化来说带动更大。这说明在项目草创初期，由于功能模块不够多，代码量相对较少，参与人员较少，每次修改代码的相对相似度值通常较高。而且 Java 每次重大的版本变化也会带来为了能够分析语言的新特性而增加的许多功能模块代码。在项目较为成熟之后且 Java 语言版本变动不大的阶段，发布版本间的相似度较高，这可能说明是修复 bug 版本，而项目的主体功能模块没有大的结构性变化。

2.3 RQ3：在什么情况下，相近两次 commit 会使得模块 AST 发生大的变化？

	package	commit1	commit2
1	soot.util.queue	80f0e2f1ff4add55dad0ac06bcdca61a1a752d30	5fb34c4458f79e8fe606737f6f8b4530dfe2e993
2	soot.util.queue	bd6e9418b53d51e6210ce59ce5457c74691b0c20	095f6751a0d35a11155477e3c6a1179e5bfbf3af
3	soot.toolkits.exceptions	38697cd8340d666bbf9831cbe17b9d26c6956424	e801483604f8ff07102ceb504df7c199396b321d
4	soot.toolkits.exceptions	e801483604f8ff07102ceb504df7c199396b321d	5e39ecd8569ae92f118fd5f3eae07a2fe758b0aa
5	soot.toolkits.exceptions	736319acd4ec3629189c67dede5f32ea1180e05a	1ba9cc93e6dcd27cc162ddc095a67185b94f8441
6	soot.jimple.toolkits.pointer.util	2fbe80749438332afeef125c005251dd8b5c5774	6db08b7d2e67e5ba609b474146e3a595e0d6ae50
7	soot.grimp	2fbe80749438332afeef125c005251dd8b5c5774	6db08b7d2e67e5ba609b474146e3a595e0d6ae50
8	soot.jimple.toolkits.annotation.profiling	2fbe80749438332afeef125c005251dd8b5c5774	6db08b7d2e67e5ba609b474146e3a595e0d6ae50
9	soot.jimple.toolkits.pointer.util	7e979339bfea2da738aa6db7887d3775d53f5c62	ae1cf8c4126771873414a127b89e59d0ba4c0d57
10	soot.grimp	7e979339bfea2da738aa6db7887d3775d53f5c62	ae1cf8c4126771873414a127b89e59d0ba4c0d57
11	soot.jimple.toolkits.annotation.profiling	7e979339bfea2da738aa6db7887d3775d53f5c62	ae1cf8c4126771873414a127b89e59d0ba4c0d57
12	soot.jimple.toolkits.pointer.util	177d800d38aaef062345689719929d3dcdcd153b	49ee53916804c697e55ea1dadc169b853b2d75ca
13	soot.grimp	177d800d38aaef062345689719929d3dcdcd153b	49ee53916804c697e55ea1dadc169b853b2d75ca
14	soot.jimple.toolkits.annotation.profiling	177d800d38aaef062345689719929d3dcdcd153b	49ee53916804c697e55ea1dadc169b853b2d75ca
15	soot.jimple.toolkits.pointer.util	90cb3e20e8392f9c8b1e16b08ddb2e73b593c8be	5c8bfba2594942e89be978167febf533177879bd
16	soot.grimp	90cb3e20e8392f9c8b1e16b08ddb2e73b593c8be	5c8bfba2594942e89be978167febf533177879bd
17	soot.jimple.toolkits.annotation.profiling	90cb3e20e8392f9c8b1e16b08ddb2e73b593c8be	5c8bfba2594942e89be978167febf533177879bd
18	soot.jbco.name	04b0333a936b119b06a42f0fb7bb38cfeeaea5dc	c6a82204bda14e27567f1ad85ad0e3c65a902ed6
19	soot.jbco.name	0d284994597440cd78ca172eb16fa612ee917f85	6e7b4ae50ff2d064a074abc91296c68a2e806054
20	soot.jbco.name	6e7b4ae50ff2d064a074abc91296c68a2e806054	a689cd1a47dd02463774ffca4444c2cb337b151d
21	soot.jbco.name	a927f16fc2203ddebd231d83e6fda85e8ab122e2	36b6fc37bd90dca5fb04cc2715b73d5d16784f09
22	soot.util.queue	dc2ac1d3784ca9faddb544250a35f1e94980b7c4	013fc1cea86ca6c9b56c2afed8ef41a205d3ac94
23	soot.util.queue	013fc1cea86ca6c9b56c2afed8ef41a205d3ac94	c174ccd931429c69c92ef6d04a1f136adc22c2c7
24	soot.util.queue	6822179006b8934ea45f2622423fb2660dc92401	f28e984039e2a93f1d4eecdc6530f601700fd0ae
25	soot.plugins.internal	d660d117d5ec858152e9655fd30a40e15f1f6db6	500ef1f11f5dd310893f65bc6bc5b1edac1f11c8
26	soot.plugins.internal	500ef1f11f5dd310893f65bc6bc5b1edac1f11c8	98d3190e9160f1206ab63dac005b788daa0a35bd
27	soot.plugins.internal	67b4c068af5a82d704f3adc30d58930e3730b109	16032a916335dbfa7bda19cae397c1d07507a66d
28	soot.jimple.toolkits.ide.icfg.dotexport	eba3ca4ebac00f7269df370b9524ac0a77e8550d	7276ecd69a3d50ebc750208aa8c29174207fe446
29	soot.jimple.toolkits.ide.icfg.dotexport	7276ecd69a3d50ebc750208aa8c29174207fe446	9738a664582e12ec904c42bfa3decdf5b53d072e

我们收集了相似度小于 0.8 的部分连续 commit 如上图所示。首先模块从没有到处创建的过程会使得模块 AST 发生大的变化。然后我们通过人工查看变化较大的连续 commit 的 log 信息，我们发现这种大的变化可能是由多种情况导致的，比如，合并其他分支的代码、添加新的功能、在代码量不大的模块里面进行文件 reformat 等等，下图给出了一个模块变化较大的 git diff 的部分示意图。

```
diff --git a/src/main/java/soot/jbco/name/AbstractNameGenerator.java b/src/main/java/soot/jbco/name/AbstractNameGenerator.java
deleted file mode 100644
index 7c6aa077a..000000000
--- a/src/main/java/soot/jbco/name/AbstractNameGenerator.java
+++ /dev/null
@@ -1,39 +0,0 @@
-package soot.jbco.name;
-
-import soot.jbco.util.Rand;
-
-/**
- * Abstract class that implements {@link NameGenerator#generateName(int)}.
- *
- * @author p.nesterovich
- * @since 21.03.18
- */
-public abstract class AbstractNameGenerator implements NameGenerator {
-
-    @Override
-    public String generateName(final int size) {
-        if (size > NAME_MAX_LENGTH) {
-            throw new IllegalArgumentException("Cannot generate junk name: too long for JVM.");
-        }
-
-        final char[][] chars = getChars();
-
-        final int index = Rand.getInt(chars.length);
-        final int length = chars[index].length;
-
-        char newName[] = new char[size];
-        do {
-            newName[0] = chars[index][Rand.getInt(length)];
-        } while (!Character.isJavaIdentifierStart(newName[0]));
-
-        // generate random string
-        for (int i = 1; i < newName.length; i++) {
-            int rand = Rand.getInt(length);
-            newName[i] = chars[index][rand];
-        }
-        return String.valueOf(newName);
-    }
-
-    protected abstract char[][] getChars();
-
-}
```

2.4 RQ4：开发过程中，模块 AST 有没有可能发生连续的大的变化？

在我们收集到的 938 个 commit 中没有连续的大的变化，这可能是由于我们收集到的都是最近的 commit，此时 Soot 项目已经规模较大，开发流程已经比较成熟，所以模块 AST 未发生连续的大的变化。我们猜测在项目早期阶段可能会出现这种连续的大的变化

3 小结

本文主要介绍了我们对 Soot 项目软件演化的分析过程，我们主要通过生成 Java 源代码的抽象语法树来进行模块间相似度比较，这是一种代码语法结构上的相似度比较。然后我们通过 4 个研究问题分别进行我们的实验，并且对于实验结果给出了我们的探索结论或者对未来探索方向的猜测。当然本文实现的 AST 相似度比较工具也可以扩展到其他领域进行扩展使用，比如说代码克隆检测、泛化重构中的识别出概念上相似的类等等。本次实验的源码已上传于 https://github.com/NJUcoder/software-maintenance-and-evolution。下一步可以进一步分析距离现在时间更长的 commit 记录或者通过其他方式比如函数调用图分析或者是数据类分析进一步证实我们的结论和验证我们的猜测。