section1

通过复制和粘贴源代码来重用是常见的做法。所谓的软件克隆就是结果。有时，这些克隆体会稍加修改以适应新的环境或目的。

有几个作者报告了7%到23%的代码重复[1]、[2]、[3]；在一个极端情况下，作者报告了59%的代码重复[4]。

代码克隆的问题是，必须在每个副本中修复原始代码中的错误。其他类型的维护更改，例如扩展或修改，也必须应用多次。

然而，代码被复制的地方通常没有文档记录。在这种情况下，人们需要检测它们。对于大型系统，只有通过自动化技术才能实现检测。

因此，提出了几种自动检测克隆的技术[1]、[3]、[4]、[5]、[6]、[7]、[8]、[9]、[10]、[11]、[12]。技术的丰富性要求定量分析评估。

这个本文介绍了2002年进行的一项实验，该实验评估了基于8个大型C和javaprogram的6个克隆检测器（总共将近850kloc）。

实验中有几位研究人员将他们的工具应用到这些系统上。他们的克隆候选人由作者之一，即斯特凡·贝隆作为独立的第三方进行评估。

所选择的技术涵盖了克隆检测的最新技术。这些技术处理文本、词汇和语法信息、软件度量和程序依赖图。

图1列出了参与者、他们的工具以及他们需要的信息类型杠杆作用

本文的剩余部分组织如下：

下一节描述了我们评估的技术和克隆检测的相关技术。

第3节给出了评估中使用的克隆类型的操作结构定义。

实验装置见第4节，结果见第5节。

第六节介绍了克隆检测评价的相关研究。

section 2

文本比较

为了提高性能，使用字符串的哈希函数对行进行分区。只比较同一分区中的行。结果显示为点图，每个点表示一对克隆线。

在这些点图中，克隆可以被发现为特定的模式视觉上。连续的在点图中，直线可以自动地总结成更大的克隆序列，

如不间断的对角线或斜线。Johnson[13]使用Karpand Rabin[14]基于指纹的高效字符串匹配令牌比较。

令牌比较

贝克的技术也是一种基于直线的比较。不用字符串比较，而是通过后缀树有效地比较行的标记序列。

首先，整行的每个标记序列由一个所谓的函子总结，函子从标识符和文字的具体值中抽象出来[1]。

函子唯一地刻画了这个标记序列。赋值函子可以看作是一个完美的散列函数。标识符和文字的具体值被捕获为这个函子的参数。

这些参数的编码是从它们的具体值中提取出来的，而不是从它们的顺序中提取出来的，

这样就可以检测到只在参数的系统重命名中不同的代码片段。如果两行的函子和参数编码匹配，那么它们就是克隆。

函子及其参数汇总在后缀树中，后缀树以紧凑的方式表示程序的所有后缀。后缀树可以建立在时间和空间线性输入长度[7]，[15]。

后缀树中的每一个分支都表示程序后缀，它们的开头都是相同的克隆序列

Kamiya等人通过规范化标记序列来增加表面上不同但等价的序列的召回率[9]。

由于没有考虑到语法，所发现的克隆可能会重叠不同的语法单元，而这些语法单元不能通过函数抽象来替换。

在前处理[16]、[17]或后处理[18]步骤中，如果块分隔符已知，则可以找到完全落在语法块中的克隆。

度量比较。

Merlo等人收集代码片段的不同度量并比较这些度量向量，而不是直接比较代码[2]、[3]、[12]、[19]。

这些度量向量的允许距离（例如，欧几里德距离）可以用作类似度量的提示代码。特定基于度量的技术也被用于网站中的克隆[20]，[21]。

抽象语法树的比较

Baxter等人。基于哈希函数划分程序抽象语法树的子树，然后通过树匹配比较同一分区中的子树（允许一些差异）[8]。

Yang[22]早期提出了一种类似的方法，使用动态规划来发现同一文件的两个版本之间的差异

程序依赖图（PDG）的比较。

函数的控制和数据流依赖关系可以用程序依赖图来表示；克隆可以被识别为同构子图[10]，[11]；因为这个问题是NP-hard的，

Krinke使用近似解。

其他技术。

Marcus和Maletic使用潜在语义索引（一种信息检索技术）来识别出现相似名称的片段[23]。

Leitao[24]通过组合专门的比较函数（类似的调用子图、交换运算符、用户定义的等价关系以及转换为规范的语法形式）

将句法和语义技术结合起来。每个比较函数都会产生一个证据，该证据在一个产生克隆可能性的证据因子模型中汇总。

Wauler-等人（25）和Li等人（26）将相似的片段搜索作为数据挖掘问题。语句序列汇总到项集。

一种自适应的数据挖掘算法搜索频繁项集。

### section 3

主要是一些相关的定义

- what is clone？

粗略地说，根据给定的相似性定义，如果两个代码片段足够相似，就形成一个克隆对。不同的相似性定义和相关的容忍度允许不同种类和程度的克隆。如果B包含A的功能，那么一段代码A与另一段代码B相似；换句话说，它们具有“相似”的前提条件和后置条件。我们把这样的一对称为（A，B）语义克隆。不幸的是，检测语义克隆通常是不确定的。

另一个定义：

相似性的另一个定义是考虑程序文本：如果两个代码片段的程序文本相似，它们就形成一个克隆对。这两个代码片段在语义上可能等价，也可能不等价。这类克隆通常是复制粘贴的结果，也就是说，程序员选择一个代码片段并将其复制到另一个代码片段位置。复制和粘贴是一种常见的编程实践，也是特别重用的一个例子。本实验所评估的自动克隆检测仪发现了程序文本中相似的克隆，因此本文采用了克隆对的后一种定义。

- 关于克隆程度的相关定义

类型1是没有修改的精确副本（空白和注释除外）。

类型2是语法上相同的副本；只更改了变量、类型或函数标识符。

类型3是经过进一步修改的副本；状态更改、添加或删除。

注解：

一些工具报告了所谓的参数化克隆[6]，这是类型2克隆的一个子集。

两个代码片段A和B是参数化克隆对，如果有从A的标识符到B的标识符的双射映射允许在A‘中进行标识符替换，而A’是到B的1型克隆（反之亦然）。

区分参数化克隆需要我们在评估工具提出的克隆对时检查一致的重命名。因为验证是完全手动完成的，而且不是所有的工具都有这种区别，所以我们没有区分参数化克隆和其他类型2克隆。

1型克隆和2型克隆被精确地定义并形成一种等价关系，而3型克隆的定义是模糊的。有些工具认为，如果两个连续的1型或2型克隆之间的间隙低于一定的线阈值，则它们一起形成一个3型克隆。另一个精确定义可以基于Levenshtein距离的阈值，即将一个字符串转换为另一个字符串所需的删除、插入或替换的数量。

最终定义

**定义1**.克隆（对）是三元组（f1；f2；t），其中f1和f2是两个相似的代码片段，t是相关的相似类型（类型1、2或3）。

事实上，在评估中，我们进一步限制了上述定义，因为附加的要求是可以通过函数调用替换克隆，即它们在语法上是完整的。

一些工具报告了处于不同语法测试级别的代码片段（例如，由两个不同的连续函数体的部分组成的片段），

这些片段可以通过宏进行索引替换；但是维护程序mer永远不会想替换它们，因为替换会使程序难以理解.

所以，下一个问题是，“代码片段到底是什么？“我们可以将一系列标记视为一个代码片段。

然而，令牌的概念因工具而异（例如，是否考虑了预处理器令牌？）并不是所有的工具都报告令牌序列。

我们对代码片段的定义是基于文本，而不是标记。标记可以映射到文本上，而源文本是一个不那么有争议的参考点

（由于宏和预处理器指令的存在，人们可以使用预处理的或原始的文本），因此它是唯一没有争议的，而不是根本没有争议的）。

程序文本可以通过文件名和行、列信息来引用。不幸的是，并非所有工具都报告列信息。

因此，为我们的评估定义代码片段的最小公分母是文件名和行信息。

**定义2**.代码片段是一个元组（f；s；e），它由源文件的名称f、片段的起始行s和结束行e组成。两个行号都包含在内。

4.1.1试运行

测试运行的目标是确定主运行的潜在问题。测试阶段分析了两个小型C程序（bison和wget）和两个小型Java程序（EIRC和spule），

在测试运行中，我们注意到一些工具会在代码片段的开始行和结束行之前或之后报告一行，如果这些行只包含一个大括号。

在实践中，这种差异是相关的，但它使来自不同工具的克隆的比较复杂化。因此，主运行的源代码是“规范化”的，删除了空行。

删除了只包含左大括号或右大括号的行，并将大括号添加到上面的行中，注意单行注释等（参见图2）。

使用布局信息[12]来检测大括号的工具可能会受到这种规范化的影响，但为了使比较更容易，所有参与者都同意该规范化。

4.1.2主运行

主运行包括对四个用C编写的程序和四个Java程序的分析。这些程序的源代码大小从11K到235K不等SLOC.

图.3概述了作为实验有些工具可以配置，我们把主运行分为强制运行和自愿运行两部分。

强制部分必须使用特定工具的“默认”设置来完成，而在自愿运行中，

每个科学家可以根据自己对主题系统的实验来调整其工具的设置，以获得最佳结果

这些工具由参与者在固定的时间段（五周）内操作，结果由Stefan Bellon收集和评估。所有参与者一致认为，

只有至少6行长的克隆被报道。较小的克隆往往更假。一些工具在进行分析之前应用了预处理器；另一些则直接处理原始程序文本。

4.2基准

我们将参与者的个人结果与参考语料库“真实克隆”进行了比较，类似于信息检索中的评估方案。

工具建议的每一个克隆对都将被称为候选克隆，参考语料库的每一个克隆对都将被称为参考克隆。

我们将参与者的个人结果与参考语料库“真实克隆”进行了比较，类似于信息检索中的评估方案。工具建议的每一个克隆对都称为候选克隆，参考语料库的每一个克隆对都称为参考克隆。

创建这样一个参考语料库的明显而简单的方法是：

1.通过不同工具报告的候选人联盟，

2.不同工具报告的候选人交叉，

3.N个工具联合发现的候选对象。

这三种方法都有不足之处。第一种选择将导致每个工具的精度为1，因为工具报告的所有候选对象都存在于参考语料库中。此外，我们在参考文献中得到许多虚假的假阳性。第二种选择具有相反的效果：对所有工具的要求是1，我们获得了许多虚假的真阴性（一个工具不能检测到某个克隆就足够了）。第三种选择是前两种选择之间的折衷，也没有真正的帮助。除了我们必须证明所选的N值是正确的以外，还有一些工具报告相同的假阳性，或者只有N-1个工具发现了真阳性。

相反，我们手动构建了参考语料库。StefanBellon作为一个独立的政党（以下简称为asoraclein）

研究了所有325935份提交的候选人中的2%，并通过插入推荐的候选人（有时在稍微修改他们之后）建立了一个参考语料库。

在下面，我们将使用术语oracled来表示Stefan Bellon查看的所有候选对象，以决定是否接受它作为克隆。

请注意，oracled包括拒绝和接受的原样或各种形式。

一个自动选择过程确保了他不知道哪些工具推荐了候选人，而且这2%是平均分配的，因此没有工具被提及或歧视。

尽管我们希望将超过2%的候选人分类，但考虑到我们的时间限制，这是不可能的：将第一个1%的候选人分类需要44小时，

将第二个1%的候选人分类需要33小时。我们在实验设计中预见到了这个问题，并采取了两种对策。

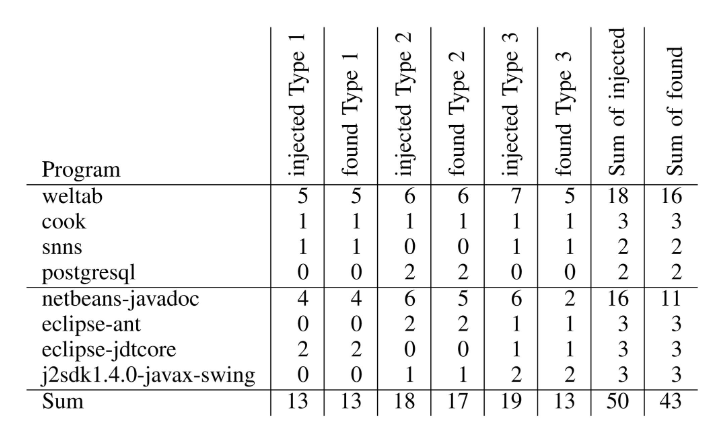
首先，在1%的候选人被神谕后进行一次评估。然后，又有1%的人被神谕。有趣的观察（如第5.3节所示）是相对定量结果几乎相同。

第二，我们注入克隆，我们没有透露给参与者在给定的程序。注入的克隆体帮助我们更好地了解潜在的召回。

图4显示了有多少克隆类型的克隆对被注入到程序中，以及有多少克隆对是通过合并工具找到的。

注入的克隆在程序中的分布并不均匀，因为Stefan Bellon开始在两个程序中引入许多克隆，然后注意到他将超过他的时间限制。

将克隆对注入程序后，它们也被添加到参考语料库中。



4.3评价指标的方法

本节定义了用于比较自动克隆检测工具的度量。这种评估是基于克隆对而不是克隆的等价类，因为只有1型和2型克隆，潜在的相似性函数是自反的、对称的和传递的。3型克隆的相似性是不可传递的：如果A是B的3型克隆，而B是C的1型克隆，那么A和C之间的相似性可能太低，不能称之为3型克隆。此外，有些工具报告它们的克隆不是类而是克隆对。

为了确定候选对象是否匹配引用，我们需要一个精确的度量。实际上，我们并不坚持完全重叠的代码片段，而是允许候选克隆对和引用克隆对之间有“足够大”的重叠。

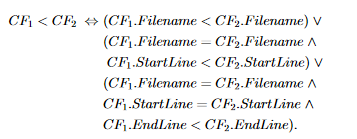
**定义3**。重叠是共同代码与两个代码片段（CF1和CF2）的比率，即它们的交集与其并集相关。让行（CF）表示代码片段CF的行集合；然后，重叠（CF1；CF2）定义为：



**定义4**.包含是一个代码片段的代码与另一个代码片段的代码之比。让行（CF1）表示第一个代码片段的行集合，行（CF2）表示第二个代码片段的行集合；然后，包含的（CF1；CF2）定义为：



现在，我们使用上面的两个定义来创建两个指标，来告诉我们候选人在引用中的命中率。要使以下两个定义起作用，我们必须确保组成克隆对的两个代码片段CF1和CF2的顺序如下：



注释：*CF1的文件名小于CF2的文件名，或者说尽管两者的文件名相等但是CF1的开始行小于CF2的开始行，或者说在文件名与开始行相等的情况下，CF1的结束行小于CF2的结束行*

因此，对于一个有效的克隆对CP=（CF1；CF2；t），CF1<CF2必须始终保持不变（错误顺序的候选代码片段只需交换即可满足此标准）。

**定义5**.两个克隆对CP1和CP2之间的良好值定义如下



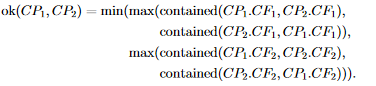
两个克隆对CP1和CP2满足以下条件，存在p∈[0，1]使得

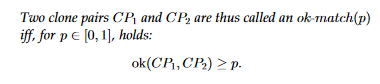


成立，则称CP1和CP2是好匹配（p）-->原文标识good-match（p）

我们使用最小重叠度，因为它比最大重叠度或平均重叠度更严格。

**定义6**.两个克隆对CP1和CP2之间的OK值定义如下





注解：good（CP1，CP2）以及ok（CP1，CP2）

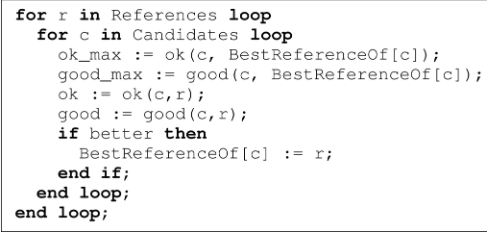
Ok-match（p）：表示CP1与CP2的相似度最少是P\*100%

Good-match（p）：与上述相同，主要是为了避免CP1远大于CP2。

中间的垂直线表示线性源代码。第一行在顶部，最后一行在底部。参与克隆的代码片段由填充的矩形表示。左侧代表第一个克隆对；右侧代表第二个克隆对。虚线箭头表示代码片段是如何被复制的。假设左侧是克隆候选，右侧是参考语料库中的克隆对。候选的第一个代码片段比引用的对应代码片段短一行，并且开始和结束的时间比引用的对应代码片段早。然而，候选者的第二个代码片段完全包含在引用的相应代码片段中，但短了两行。

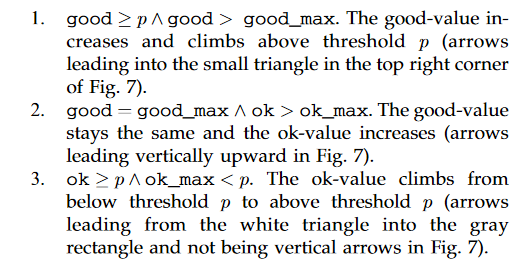
这个不等式意味着对于相同的p值，一个好匹配（p）比一个好匹配（p）更重要。在我们的实验中，我们决定使用p=0.7的值。因为在实验中克隆的可接受长度的阈值是6，所以选择p=0.7允许两个6行代码片段移动一行。例如，如果一个克隆对的片段从第1行开始到第6行结束，而另一个克隆对的片段从第2行开始到第7行结束，则重叠度为5/7>p=0.7。这种选择适应了被评估工具行报告中的一对一不一致。因为这两个度量基本上都是重叠的度量——从两个片段的角度来看是好的，从较小片段的角度来看是好的，所以出于一致性的原因，我们选择对这两个度量使用相同的阈值

最后，必须建立从候选人到参考人的映射。每个候选对象都映射到它最匹配的引用。用于建立该映射的算法的思想如图6所示（实际上，使用了更有效的实现）。



有两个维度可以优化从候选到引用的映射：good和ok值

图6中的谓词better用于定义优化准则。predicate better的动机是将引用映射到候选者，以获得尽可能高的好值和尽可能高的ok值，而不损害好值（即，匹配仍保留在good类别中）。图7用于详细地解释比特的组成。在图7中，您可以通过算法期间最佳匹配参考的路径（good，ok）坐标看到每个组成部分的方向：如果可能，它会向good/ok值空间的右上角移动。对角线下方的阴影部分无关紧要，因为good<=ok始终有效。如果下列属性中至少有一个为真，则batter为真：



基于上述定义，我们可以定义几种衡量检测质量各个方面的指标。在以下定义中，T是表示参与工具之一的变量，P是表示所分析程序之一的变量，以及“t”是表示观察到的克隆类型的变量。这三个变量都有一个特殊值“All”，分别表示所有工具、程序和克隆类型。此外，还有一个特殊值“unknown”，因为某些工具无法对克隆类型进行分类。首先，我们介绍一些基数。然后，我们用这些基数来定义调用和精度。

定义7。

让Cands（P；T；“t”）表示通过工具T找到的P程序的t拷贝类型的候选人

让Refs（P；“t”）表示程参考语料库中对于程序P的t克隆类型的参考。

让OKRefs（P；T；“t”）表示通过工具T使得ok>0.7的程序P和克隆类型“t”的引用。

让GoodRefs（“P”，“T”，“t“）表示通过工具T满足good>0.7的程序P和t克隆类型的引用。

让DetectedRefs（“P”，“T”，“t“）表示OKRefs或者GoodRefs的同义词，根据情况进行选择。

让FoundSecrets（“P”，“T”，“t“）表示工具T找到的关于程序P的（被主动注入的）t类型克隆的数量。

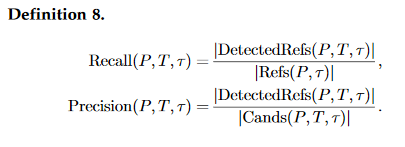
让OracledCands（“P”，“T”，“t“）表示被oracle（第三方）使用工具T调查出来的关于P和类型t的所有候选。

让RejectedCands（“P”，“T”，“t“）表示被oracle（第三方）使用工具T调查出来的关于P和类型t的但是good<0.7或者·ok<0.7的，在人工调查之后是分别进行的

让TrueNegativeRefs(“P”，“T”，“t“)表示所有的关于工具T，t类型和P程序的参考资料库，没有候选达到ok>0.7 或者good>0.7的那些references在评价之后

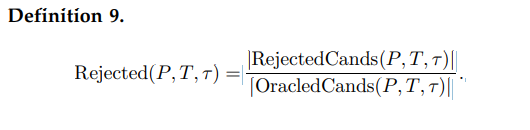
参考集是由至少一个工具提出的克隆形成的，这些克隆由第三方确认，并由我们手动注入的克隆形成。因此，如果没有找到注入的克隆，或者第三方修改了克隆对，因此生成的引用与任何工具集中的候选对象（甚至是导致克隆首先放入引用集中的候选对象）都不匹配，则所有工具都会出现真正的负值。

基于上述基数，我们可以定义如下：

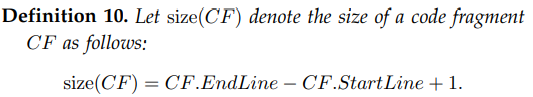


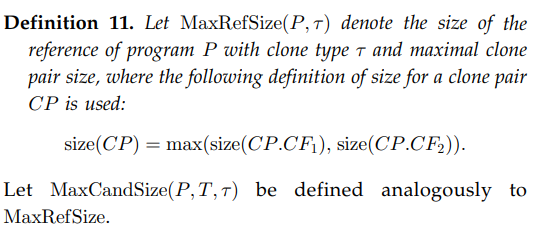
召回指标是有意义的，即使我们oracled只有2%，但精度指标适用于我们不完整的参考集将只产生一个下限的实际精度。

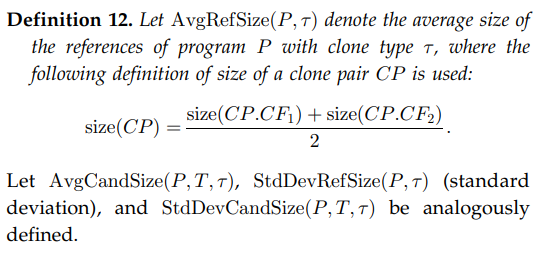
这就是为什么我们使用一个不同的指标来了解推荐候选人的质量，这个指标基于我们实际验证的候选人，如下所示



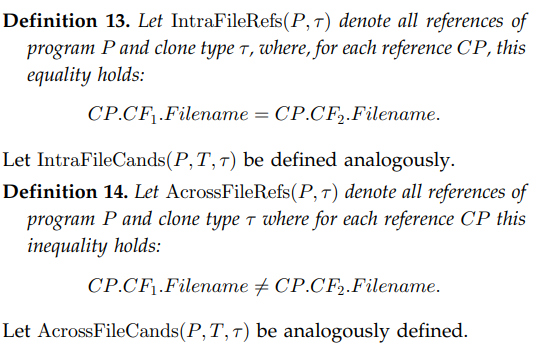
4.3.2 Clone Size Aspects





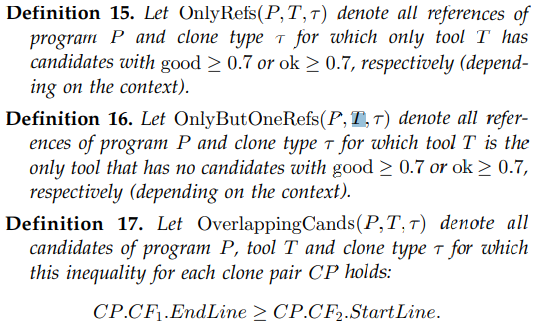


4.3.3 Cloning Scope



4.3.4 Distinctiveness of Tools

本节中定义的测量允许我们调查工具的独特贡献及其独特的缺陷。



5 EVALUATION

本节介绍实验结果。首先，我们给出了参考集。然后，我们来仔细看看其中一个分析过的程序。最后，看看其他几个分析程序的结果。

我们必须注意，Merlo等人在这个实验中使用了两种不同的克隆检测技术：基于度量的克隆检测技术用于功能克隆，基于令牌的克隆检测技术用于类型1和类型2克隆。

5.1 The Reference Set