

CENTRAL SOUTH UNIVERSITY

**本科生毕业设计**

**英文翻译及原文**

|  |  |
| --- | --- |
| 题 目 | 关注首条编译错误：同时处 |
|  | 理多个编译器错误信息 |
| 学生姓名 | 罗龙光 |
| 指导教师 | 李洪东 |
| 学 院 | 信息科学与工程学院 |
| 专业班级 | 信安1801 |

关注首条编译错误：同时处理多条编译器错误信息

Brett A. Becker Cormac Murray Tianyi Tao

爱尔兰都柏林大学 爱尔兰都柏林大学 复旦大学  
贝尔菲尔德学院，都柏林4 贝尔菲尔德学院，都柏林4 上海，中国  
[brett.becker@ucd.ie](mailto:brett.becker@ucd.ie) [cormac.murray@ucdconnect.ie](mailto:cormac.murray@ucdconnect.ie) [13302010026@fudan.edu.cm](mailto:13302010026@fudan.edu.cm)

Changheng Song Robert McCartney Kate Sanders

复旦大学 美国康涅狄格大学 美国罗得岛学院

上海，中国 斯托尔斯 普罗维登斯

[13302010026@fudan.edu.cm](mailto:13302010026@fudan.edu.cm) [robert@engr.uconn.edu](mailto:robert@engr.uconn.edu) [ksanders@ric.edu](mailto:ksanders@ric.edu)

摘要

为了帮助学生学习编程，一些计算机教育研究人员分析了新手在编译阶段产生的错误，以期帮助他们通过编译器反馈的信息来诊断这些错误。本文在之前的工作的基础上，应用了程序员们所熟悉的启发式方法 ：只关注第一个错误，忽视其他错误信息的方法，去分析了Blackbox数据集中的超过2100万条编译器错误信息。

我们发现，考虑所有的错误信息而得到的错误出现频率与以往一些研究的成果基本一致，但当我们只考虑第一条编译错误时，这些信息和频率就不同了。这些差异可能会对教学产生重要影响，并且可以作为工具并对未来的研究工作提供参考。

关键字

编译器错误信息；级联错误信息；后续错误信息；埋藏错误信息；新手语法错误；CS1；黑盒；新手程序员；调试

1 简介

计算机编程已被证明给学生带来了许多困难，它导致了学生的低积极性，低成功率，且提高了辍学率。调试是学习编程过程中的一个挑战，特别是，如何学习并理解编译器的报错信息。我们对于学生如何处理他们从编译器得到的反馈感兴趣：他们看到了什么，他们如何利用这些信息，他们如何进行调试改进。

新手往往被建议只关注编译器的第一个报错，正如Ben-Ari所说：“通常，一个小错误会导致编译器发出一连串的信息......。不要投入任何精力去修复多个错误信息！集中精力修复第一个错误信息 。”，Fincher和Barnes 也有类似的观点：“从第一个错误开始，在源文件中修正它，保存文件，然后重新编译。不要试图一次性修复所有的错误。如果你有级联相关的错误，或者有相同原因的多个错误，如拼写错误的标识符，这种方法就特别重要。”

这项工作的首要贡献是利用Blackbox数据集，大规模分析了将第一条错误信息与后续信息隔离的效果。此外，我们提出了一个基于学生调查的创新性结果，并提出了能使研究人员和教育工作者更精确地讨论编译器错误信息的术语。

1.1 学生的观点

1984年，DuBoulay和Matthew指出:“所有的新手程序员都发现，他们一开始编写的程序总伴随着一堆难以理解的报错信息被编译器所拒绝”。我们对106名正在学习CS1模块的软件工程学生进行了调查，其中73人（69%）做出了回应。这些学生使用Notepad++进行C语言编程，并使用GCC命令行进行编译。我们试图了解编译器错误信息（尤其是多个编译器错误信息）给学生们带来了的挫折、困惑的程度，并确定是否需要在这些问题上给学生们提供额外帮助。我们将这次调查的结果起名为三个标题：沮丧和困惑、误解与请求帮助。

**挫折和困惑**：85%的学生表示发现编译器错误信息他们挫折感的来源，92%的学生表示编译器错误信息是他们进步的障碍。64%的人报告说发现多个编译器错误信息会让他感到困惑。

**误解**：67%的学生说，当遇到只有一个编译器错误信息时，这个错误可能不代表代码中的真正错误。21%的人认为，第一个之后的编译器错误信息总是代表代码中的真实错误，11%的人认为这些错误信息从来不代表代码中的真实错误。当遇到一个以上的编译器错误信息时，只有22%的人试图修复第一个报告的错误信息，43%的人试图修复一个可能是也可能不是第一个报告的错误，34%的人试图同时修复一个以上的错误。

**请求帮助**：100%的学生表示，如果有专门针对处理编译器错误信息的额外帮助，将有助于他们学习编程。99%的学生认为，向他们解释多个“级联”的编译器错误信息的原因和性质会帮助他们更有效地处理编译器错误信息。

这些结果表明，需要对错误信息（包括多种错误信息）进行更深入的讨论并加以更多的关注。

1.2 编译器错误信息的分类法

为了便于讨论编译器的错误信息，我们引入了一个分类法，如图1所示。所有的信息都属于两种类型中的一个：第一类错误或随后的其他错误。第一条信息是编译器在编译失败后报告的第一条信息。第一条信息总是与代码中的真正错误相对应。即使是新手也知道，大多数编译错误会产生一些额外的错误信息；我们不知道这些非第一条错误信息的现有术语。我们使用术语 “后续 ”来指代编译器在第一条信息之后报告的所有信息。后续错误可以依次分解为两种类型。

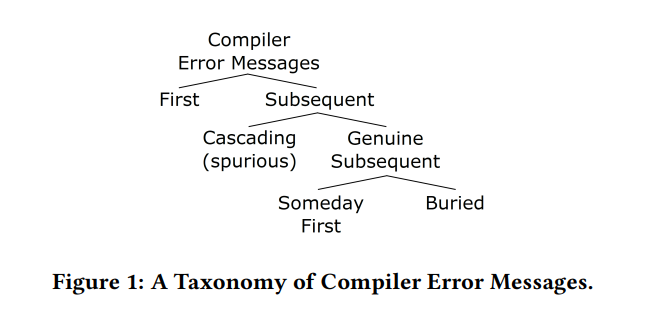
**级联式错误**：级联式错误信息是由一个与先前的错误信息相对应的错误引起的虚假信息，并没有提供额外的（有用的）信息。简而言之，这些是编译器混乱的结果。这个定义与Schorsch的描述一致。“由一个错误引起的多个错误报告，都可以通过修复第一个报告的错误而得到纠正。”

**真正的后续错误**:这些是由于代码中的真正错误而产生的后续信息。与级联错误信息不同的是，这些信息不是来自已经生成的错误。这些是与代码中的真正错误相对应的第一例消息。

随后，真正的后续错误信息被分为两类。

**someday first**: Someday first错误最终会成为编译后的第一条信息。

**buried**：相对于其他合法的错误，这些错误本应成为第一条错误信息，但却并没有，因为在它们到达错误列表的顶部之前发生了一些事情。例如，程序员可能会在没有编译器帮助的情况下看到并修复基本错误，或是在修复之前停止程序，又或是终止整个程序运行并重新开始。



1.3 一个激励人心的例子

考虑以下Java代码，它在第三行有一个错误： for循环的条件之间有一个普通的冒号而不是分号。

public class PrintNums {

public static void main (String[] args) (

for(int x = 0: x < 10; x++)

System.out.println(x)。

}

)

这是一个很容易犯的错误，要么是学习的时候不严谨，要么是一个简单的拼写错误。事实上，这两个符号经常处在键盘的同一按键上。当这段代码被编译时, 编译器会返回以下结果。

PrintNums.java:3: error: ';' expected

for(int x = 0: x < 10; x++)

ˆ

PrintNums.java:3: error: ';' expected

for(int x = 0: x < 10; x++)

ˆ

PrintNums.java:3: error: illegal start of type

for(int x = 0: x < 10; x++)

ˆ

PrintNums.java:3: error: illegal start of expression

for(int x = 0: x < 10; x++)

ˆ

PrintNums.java:3: error: ')' expected

for(int x = 0: x < 10; x++)

ˆ

PrintNums.java:3: error: illegal start of expression

for(int x = 0: x < 10; x++)

ˆ

6 errors

第一条信息准确地反映了代码中真正的错误。其他的是由于编译器被第一个（也是唯一的）错误迷惑而产生的级联信息。一旦原始错误（对应于第一个信息）被纠正，所有五个级联错误信息都会消失。花在级联错误上的时间被浪费了，由它们引起的困扰是没有必要的。

1.4 问题研究

我们想知道，如果我们不再试图帮助学生应对冗杂的错误信息列表，而是把注意力放在每次编译产生的第一条错误信息上，会发生什么变化。具体来说，我们的研究问题是以下这些:

Q1: 在所有的编译器错误信息中，第一条信息的比例是多少？

Q2：最频繁出现的错误信息的差异？

Q3：特定错误信息出现的频率的差异？

在接下来的章节中，我们将讨论本文所基于的相关工作（第2节），描述我们的方法（第3节），介绍我们的结果（第4节），讨论结果及其影响（第5节），考虑研究的有效性以及风险（第6节），并在最后讨论对教学、研究和未来工作方向的影响（第7节）

2相关工作

在本节中，我们将讨论与编译器错误信息的类型、它们的频率以及我们用来研究我们问题的Blackbox数据集的前期工作。

2.1错误和错误信息

编译器错误信息已经被研究了几十年了，它对于调试来说是必不可少的一环，且会给学生成功运行程序带来困难。尽管这些错误信息很重要，但对新手来说却特别的困难。

一些研究者分析了学生对错误信息的理解，开发了可以提供更友好的编译器错误信息的工具，或用附加信息补充编译器所提供的标准信息。其他研究人员使用学生的错误信息来对学生的编译或调试行为进行全方面研究，包括[1,2,6,10,12,20]。其他研究员，比如BenAri，写了一些指南，用于解释错误信息的含义，并举例说明会导致错误的代码和修复错误的方法。

在最近一项针对软件工程专业本科生的研究中，Barik等人[3]发现，学生们在阅读编译器信息上花费了大量时间。毫不奇怪的，学生所面对的错误信息难度越大，他们成功调试出给定程序的可能性就越小。

2.2第一条错误信息与随后的信息

为了帮助新手处理编译器信息，Munson和Schilling[26]建议他们去关注第一条错误信息。然而，他们发现他们的学生只有在48.2%的编译过程中尝试中这样做。

一些面向新手的编程环境，如CAP[30]和BlueJ[23]，它们通过只显示第一条编译器信息来迫使学生集中注意力。回想一下我们在第1节所给出的例子。BlueJ在面对这个编译错误，将只显示第一条错误信息，而不是六条信息，BlueJ的有以下特性：

BlueJ的编译速度非常快（对于2000行的代码，通常在一秒钟之内即可完成编译），以至于 "编译 "按钮可以作为的 "下一个错误 "按钮....这有几个好处：编译器更简单（因为不需要尝试错误恢复），错误信息更准确（因为避免了由其他的错误引起的不正确且不必要信息）[22，第8-9页]

2.3 明确各错误信息的频率

许多关于错误信息的研究的共同点是介绍 “最常见的 ”错误（在代码中），或错误信息（由代码中的错误导致）。这是基于以下合理的假设，即帮助学生解决他们最常处理的错误信息将是最有明显效果的。然而，在实践中，这些列表所包含的内容，特别是它们是如何被确定的，就大相径庭。另外，这些清单经常为其他清单提供信息，而且许多清单被用作进一步研究的基础。下面我们简要介绍一下其中的一些努力。关于许多这些 "常见错误/信息清单 "的更完整的总结，见[4]。

Hristova等人[19]制作了一份关于Java错误信息清单，该清单最初是收集了教育者们所认为的最常见的错误。Flowers等人[18]根据他们的教学经验，选择了九个常见的错误。Denny等人[14]使用一个定制的工具，从学生编码练习中收集错误。Becker等人[4，5，7]使用一个编辑器，旨在收集与加强由标准JDK编译器产生的错误信息。Chan Mow[12]统计了使用由JBuilder从三个类中收集的信息。Thompson[32]从Eclipse的工具化版本中收集信息。作为一个例子，这种信息列表在为其他研究提供支撑方面可以有很长的寿命，2017年Brown和Altadmri[10]使用了一组18个错误信息来支持他们的研究，这些错误信息来自Hristova等人[19]在2003年最初提出的20个错误。

从这些描述中可以看出，这些已有的信息表并不可直接互相比较：不是所表单有的都是基于编译器的错误信息（而是代码中的错误）；有些结合了某些相关的错误信息；有些使用不同的编译器，有不同的错误信息；还有一些甚至是模糊抽象的错误信息。总的来说，这些研究都忽略了一个事实，即许多学生只遭遇到了编译器所报告的第一条错误信息。本研究的一个目标是加深对于编译器所给出的第一条错误信息的研究。

2.4 为什么选择Blackbox做数据集

Blackbox是一个新的数据集，它常被用于研究新手如何学习编程的问题。它收集了学生与BlueJ的互动，比如源代码、编译器信息等。数据来源为世界各地的BlueJ用户。数以千计的用户已经选择加入数据收集计划，同意将他们的数据（匿名）用于研究。因此，现在比以前有了更多的数据源，可以查看各地学生程序的编译信息、代码修改操作和编译错误信息[11, 33]。该数据集时效性强，且规模如此大。现在研究人员仍在探索Blackbox的潜力。到目前为止，大多数论文都考虑了错误和错误频率[2, 10, 21, 29]。有一篇论文讨论了学生对GoTo-like控制结构的使用行为[31]。

3 研究方法

我们分析了美国的BlueJ用户在2016年1月1日至6月30日（含）期间的错误信息（不包括警告）。我们对数据进行迭代，首先选择所有bluejstart事件，从中检索出所有有效的session\_id值。对于每个session\_id，我们检索了所有的compile\_events。从这些事件中，我们从compile\_outputs表中检索出每个session\_id的所有错误信息，该表还包含一个标志，表明该信息是否显示给用户。这与JDK输出的第一个错误信息相对应，因为BlueJ只向用户显示第一个错误。所有没有显示的信息都是后续信息，其中包括所有的级联信息。

许多Java错误信息包含来自源代码的标记，它们可能是唯一的。例如，下面的错误信息包含一个与包名edu.univ.my.package对应的标记：包edu.univ.my\_package不存在

为了对这样的错误信息进行分类，我们使用存储在JSON文件中的正则表达式来处理所有信息。这使我们能够将所有同类错误消息归入一类。我们对所有其他包含独特标记的消息做了类似的处理。这些研究的结果是一个包含了202个错误信息的列表。除了那些含有潜在独特标记的信息，我们没有合并任何相似的错误信息。我们的研究中使用的源代码可在以下网址获得 <https://github.com/TTY112358/JavaErrorCollector>。

4 结果

我们分析了来自135,629个用户的21,511,098条错误信息，1,444,260个BlueJ会话，并计算了以下信息类别的频率：首次错误（F）；后续错误（S）；和所有错误信息（A）。

4.1 RQ1：第一条错误信息的比例

在我们的数据集中的21,511,098条错误信息中，28.5%（6,139,013条）是第一条错误信息。平均编译失败产生3.50条错误信息，其中2.50条是后续错误信息。当我们只关注第一个错误信息时，错误信息的分布开始出现不同的变化。

4.2 RQ2: 最频繁出现的错误信息的差异

教育者的建议和研究者的研究之间存在着一些差距：尽管教育者建议关注第一条错误信息已经有一段时间了，但大多数分析编译器错误信息的文献只考虑错误信息的总体频率。在本节中，我们将探讨在只考虑第一个错误信息的情况下，情况会有什么变化。

表1中显示了所有首条错误信息中的前85%，它们按照出现频率升序排列（RF），以及它们相应的频率百分比（%F）。我们把这组错误信息称为F\_85。我们选择85%，因为这包括了所有百分比频率大于1%的单个错误。与A\_85（所有错误信息的前85%）和S\_85（后续信息的前85%）相比，F\_85的规模提供我们这种只考虑第一条错误的基本信息。F\_85包含23条错误信息，A\_85包含17条，而S\_85只包含14条。

A85由表1中RA列的14条无阴影信息（也出现在F\_85中）和3条不在F\_85的信息组成（也不在表1中）。S\_85由表1中RS栏中的11条无阴影信息（也出现在F\_85中）加上同样的三条不在F\_85中的信息（也不在表1中）组成。下面展示了三种类型的错误的作为首个出现的（R\_A）和非首条错误信息出现（R\_s）信息频率的相应排序。

(1) Illegal start of type（R\_A=10，R\_s=10）。

(2) illegal character (R\_A = 13, R\_s = 11)

(3) non-static variable cannot be referenced from a static context（R\_A=17，R\_s=14）。

将A\_85和S\_85相比，最常出现的第一条编译错误信息出现了非常不同的情况；从表1中的错误信息频率的排名来看，R\_A和R\_S之间的相似度与它们和R\_F来比更高。只有在最常出现的错误上，三者保持了一致。此外，当只关注第一条错误信息时，我们会看到以往一些排名在后的错误信息来到了表单的前列。 F\_85中有9条信息不在A\_85中（在R\_A>14时，表1中RA有阴影），而F\_85中有12条信息不在S\_85中（在R\_A>14时，表1中R\_s有阴影）。

4.3 RQ3: 特定错误信息频率的差异

当我们只关注第一条错误信息时，个别错误信息的频率变化明显。在表1中，%F和%s的计算方法是：将排名第i条信息的出现次数分别除以第一条出现的信息和后续信息的总和。这些数值表示某条信息作为第一条错误信息或后续信息出现的概率。例如，表1显示，“missing return statement”作为第一条错误信息（3.20%）出现的可能性是作为后续信息（0.57%）的5.6倍。

为了方便这种比较，我们计算了%s和%F之间的对数变化：[log2（%s/%F）]。当这个值为1时，%s是%F的两倍；2表示%s是%F的四倍；3表示%s是%F的八倍。与简单的计算%s除以%f不同，以二为底取对数有以下又是：第一，它以0为中心，如果%s和%F相等，log2（%s/%F）就为0。第二，它是对称的，数值为正数就意味着%s大于%F，而负值意味着相反。为了类似地比较F和A，我们也计算了log2(%A/%F)。

图2显示了F\_85中的前23错误信息的log2(%s/%F)和log2(%A/%F)。可以看出，第一条出现的错误信息和后续错误信息，第一条出现的错误信息和所有的错误信息之间的差异都遵循着类似的趋势。有六条数据处于0和1之间，表明这些信息作为后续错误信息的可能性小于作为第一条信息的两倍。对于一条错误信息（即第12条），这种可能性在2到4倍之间。其余16条数据为负值，表明这些错误信息作为第一条信息比作为后续信息的可能性更大。对于一些信息来说，作为第一条信息出现的可能性比作为后续信息出现的可能性大近8倍。

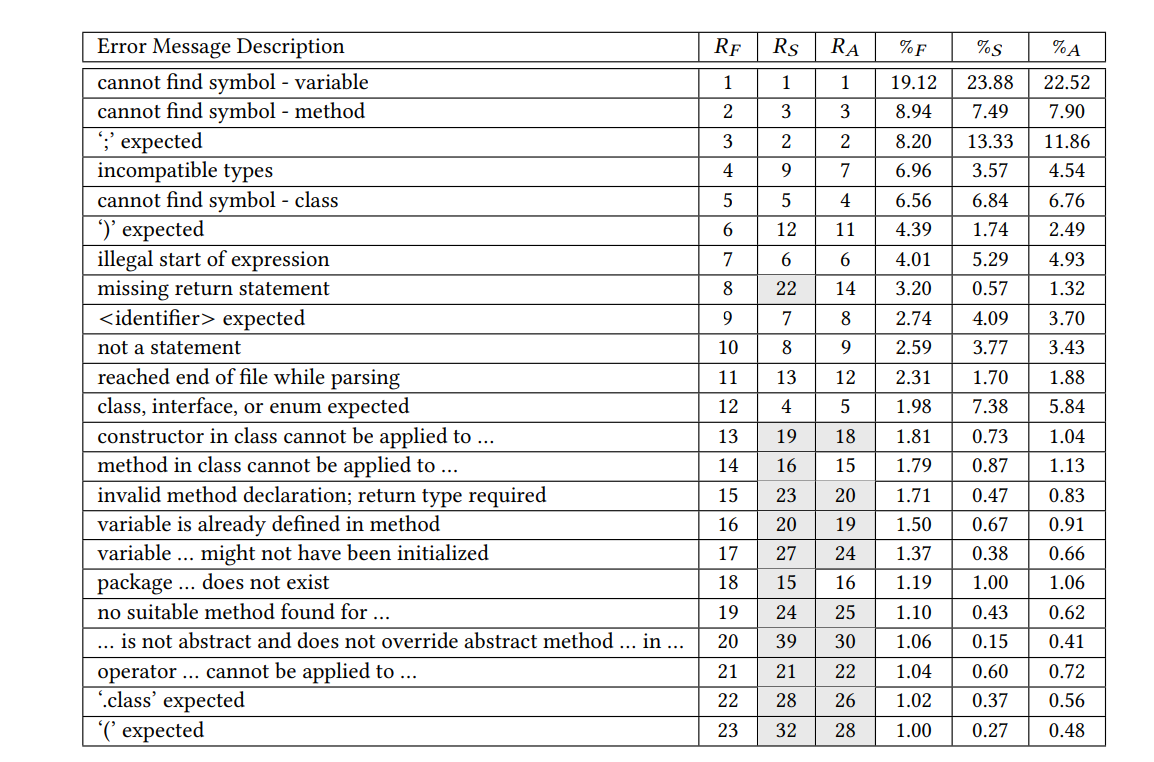
作为第一条信息更频繁的错误信息的一个例子是：“missing return statement”（比作为后续信息的可能性高5.5倍）。

5 讨论

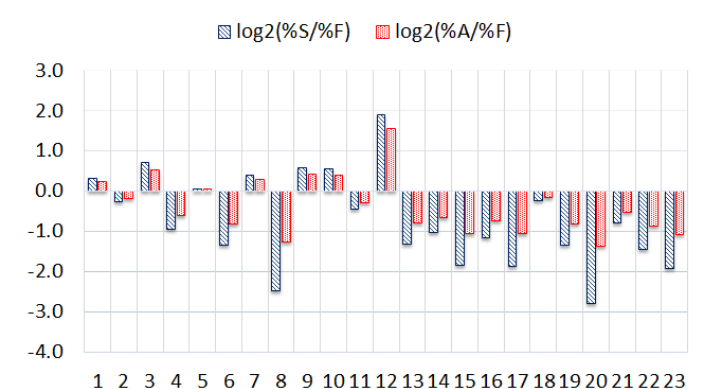
我们发现的首条错误信息所占的百分比（28.5%）、每次编译失败的平均错误信息数量（3.50）和后续信息的平均数量（2.50）与前人的研究相当。Murray[27]调查了2015年7月1日至2016年6月30日（含）间爱尔兰地区的Blackbox用户的信息，包括3708位用户和22440次会话。他发现首条错误信息占比23.1%，即每次失败的编译伴随着4.34条错误信息（即3.34条后续信息）。在1997年对42名Ada程序员新手的研究中，Moore等人[25]报告说，每次编译失败平均有4.06条错误信息（即3.06条后续信息）。

最频繁出现的首条错误信息表单与所有错误信息的表单有明显不同，也与之前研究发布的类似数据不同[11, 16, 20]。首条错误信息中有23条不同的信息，而全部错误信息有17条。首条错误信息中有9条不在所有错误信息的列表中，（如第4.2节所述）。在同时出现在两个列表中的错误，只有一个错误在两个表中的排序相同，而且它们所有的频率都有差异。例如，“missing return statement and ‘)’ expected”为作为第一条信息情况要常见得多，而“class, interface, or enum expected”作为第一条错误信息的情况则要少得多。

我们的“最频繁出现的第一条错误信息”列表包括了所有出现频率至少为1%的第一条错误信息。使用只显示第一条编译错误信息的编辑器的学生，平均在一个学期中会看到这些错误信息中的每一条1到19次，考虑到一个学期的编程教学中学生大概会遇到有100此失败的编译，因此，这些学生每节课会看到9条不同的错误信息，而这些信息并不在之前的文献所报告的频繁错误信息的总体清单中（这些表单是基于所有编译错误总体出现频率而定的）。此外，这些学生不会看到文献中提到的三种常见错误信息。这不仅影响了学生，也影响了CS1讲师和实验室的导师，因为他们有很大一部分时间是用来帮助学生纠正编译错误的。正如文献[14]的作者所指出的，作为教育工作者，我们对编译器错误的性质以及学生对它们的反应了解得越多，我们的教学工作就会越有效。



**图一**



**图二**

6 研究的局限性

Brown等人[11]已经描述了Blackbox数据集的局限性。值得注意的是，Blackbox只包括BlueJ用户，他们主要是新手,不过我们也主要是对新手感兴趣。另外，BlueJ用户没有看到一次编译后除了第一条错误的随后的消息；但我们所关注的是编译器消息，而不是用户对这些消息的反应，而且对于BlueJ和其他使用标准JDK的IDE来说，特定编译的基本消息基本上是一样的。BlueJ用户可能会使用不同的版本，因此与其他IDE的用户相比，总体消息频率不同。但我们认为这些影响可以忽略不计，因为我们的总体错误列表与其他研究人员发布的列表相似。

我们的结果也可能受到BlueJ是一个需要授权才能进行收集的数据集的影响。因此，这项研究的结果可能会受到志愿者偏见的影响。这也是我们与所有基于Blackbox的研究的一个共有问题。最后，我们的样本虽然很大，但仍然只是整个数据集合的一个子集。非美国的用户或不同的时间段的使用者有可能显示出不同的特征。

7 结束语

现在教育工作者们在对学生进行指导时，经常建议他们最好只关注第一条信息，也是一些新手编程环境所强制执行的。我们的贡献是一个更精确地讨论编译器错误信息的分类法，以及关于只关注第一条信息时错误信息情况时，各类数据指标时如何变化的。当我们只考虑第一条信息时，错误信息的出现频率和以往的研究相比是有区别的。我们分析了Blackbox数据集中的2100多万条编译器错误信息，发现所有错误信息中只有28.5%是第一条错误信息，而且最频繁的第一条信息列表与基于所有编译器错误信息列表有明显不同。

我们引入了一个新的分类法，使研究人员和教育工作者有可能更准确地谈论这些编译错误。在Blackbox数据集的所有编译器错误信息中，我们定义每次编译失败的第一条信息被标记为 "第一条信息"。 理论上，随后的信息可以分为三种类型：“级联信息”（虚假的错误信息，当真正的错误被修复后就会消失）；“真正的第一条信息”（对应于一个明显的错误，在编译中成为第一条信息）；以及 “被埋藏信息”（对应于一个明显的错误，但在编译后未成为第一条信息）。我们希望未来的工作也将根据这个分类法对错误进行分类。

虽然第一条错误信息与调查学生与编译器的互动的研究最为相关，但仅仅关注第一条错误信息，可能会导致被埋藏的信息被忽略，反而成为程序调试的绊脚石。设计和实现一种算法，从Blackbox中提取埋藏信息，将会是一个有趣的挑战。

感谢

伦敦大学国王学院和肯特大学的BlueJ团队提供的Blackbox数据集，使这项研究成为可能。

参考文献

1. Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. 2005. An Analysis of Patterns of Debugging Among Novice Computer Science Students. In ITiCSE '05. ACM, 84-88.
2. Amjad Altadmri and Neil C.C. Brown. 2015. 37 Million Compilations： Investigat­ing Novice Programming Mistakes in Large-Scale Student Data. In SIGCSE '15. ACM, 522-527.
3. Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. 2017. Do Developers Read Compiler Error Messages? (ICSE '17). 575-585. DOI:<http://dx.doi.org/10.1109/ICSE.2017.59>
4. Brett A. Becker. 2015. An Exploration of the Effects of Enhanced Compiler Error Messages for Computer Programming Novices. Master's thesis. Dublin Institute of Technology.
5. Brett A. Becker. 2016. An Effective Approach to Enhancing Compiler Error Messages. In SIGCSE '16. ACM, 126-131.
6. Brett A. Becker. 2016. A new metric to quantify repeated compiler errors for novice programmers. In ITiCSE '16. ACM, 296-301.
7. Brett A. Becker, Graham Glanville, Ricardo Iwashima, Claire McDonnell, Kyle Goslin, and Catherine Mooney. 2016. Effective compiler error message enhance­ment for novice programming students. Computer Science Education 26, 2-3 (2016), 148-175.
8. Mordechai Ben-Ari. 2007. Compile and Runtime Errors in Java. [http：//www.](http://www.weizmann.ac.il/sci-tea/benari/software-and-learning-materials/errors.pdf) [weizmann.ac.il/sci-tea/benari/software-and-learning-materials/errors.pdf.](http://www.weizmann.ac.il/sci-tea/benari/software-and-learning-materials/errors.pdf) (2007). Accessed 12 Jan 2017.
9. Benedict du Boulay and Ian Matthew. 1984. Fatal error in pass zero： How not to confuse novices. Behaviour & Information Technology 3, 2 (1984), 109-118.
10. Neil CC Brown and Amjad Altadmri. 2017. Novice Java programming mistakes： large-scale data vs. educator beliefs. ACM Transactions on Computing Education (TOCE) 17, 2 (2017), 7.
11. Neil Christopher Charles Brown, Michael Kolling, Davin McCall, and Ian Utting.
12. Blackbox： A large scale repository of novice programmers' activity. In SIGCSE '14. ACM, 223-228.
13. Ioana Tungalei Chan Mow. 2012. Analyses of student programming errors in Java programming courses. J. Emerging Trends in Computing and Information Sciences 3, 5 (2012), 739-749.
14. Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing syntax error messages appears ineffectual. In ITiCSE '14. ACM, 273-278.
15. Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. 2012. All syntax errors are not equal. In ITiCSE '12. ACM, 75-80.
16. Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. 2011. CodeWrite： Supporting student-driven practice of Java. In SIGCSE '11. ACM, 471-476.
17. Thomas Dy and Ma. Mercedes Rodrigo. 2010. A Detector for Non-literal Java Errors. In Koli Calling '10. ACM, 118-122.
18. Sally Fincher and David Barnes. 2006. Studying Programming. Palgrave Macmil­lan.
19. T. Flowers, C.A. Carver, and J. Jackson. 2004. Empowering students and building confidence in novice programmers through Gauntlet. In FIE '04. IEEE, T3H-10.
20. Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. 2003. Iden­tifying and Correcting Java Programming Errors for Introductory Computer Science Students. In SIGCSE '03. ACM, 153-156.
21. Matthew C Jadud. 2006. An exploration of novice compilation behaviour in BlueJ. Ph.D. Dissertation. Univ. of Kent.
22. Matthew C. Jadud and Brian Dorn. 2015. Aggregate Compilation Behavior： Findings and Implications from 27,698 Users. In ICER '15. ACM, 131-139.
23. Michael Kolling. 1999. Teaching object orientation with the Blue environment. J. Object Oriented Programming 12 (1999), 14-23.
24. Michael Kolling, Bruce 边ig, Andrew Patterson, and John Rosenberg. 2003. The BlueJ system and its pedagogy. Computer Science Education 13, 4 (2003), 249-268.
25. Guillaume Marceau, Kathi Fisler, and Shriram Krishnamurthi. 2011. Measuring the Effectiveness ofError MessagesDesigned for Novice Programmers.In SIGCSE '11. 499-504.
26. Deanne Moore, Allen Parrish, and David Cordes. 1997. Analyzing syntax error patterns among novice programmers. In ACM-SE '97. ACM, 188-190.
27. Jonathan P. Munson and Elizabeth A. Schilling. 2016. Analyzing Novice Pro­grammers' Response to Compiler Error Messages. J. Comput. Sci. Coll. 31, 3 (2016), 53-61.
28. Cormac Murray. 2016. A Comparative Study of Java Compiler Error Profiles Using the Blackbox Dataset. Master's thesis. University College Dublin.
29. James Prather, Raymond Pettit, Kayla Holcomb McMurry, Alani Peters, John Homer, Nevan Simone, and Maxine Cohen. 2017. On Novices' Interaction with Compiler Error Messages： A Human Factors Approach. In ICER '17. 74-82.
30. David Pritchard. 2015. Frequency Distribution ofError Messages. In PLATEAU
31. ACM, 1-8.
32. Tom Schorsch. 1995. CAP： An automated self-assessment tool to check Pascal programs for syntax, logic and style errors. In SIGCSE '95. ACM, 168-172.
33. Stewart D. Smith, Nicholas Zemljic, and Andrew Petersen. 2015. Modern Goto： Novice Programmer Usage of Non-standard Control Flow. In Koli Calling '15. ACM, 171-172.
34. Suzanne Marie Thompson. 2004. An Exploratory Study of Novice Programming Experiences and Errors. Master's thesis. Univ. of Victoria.

# Fix the First, Ignore the Rest: Dealing with Multiple Compiler Error Messages

# ABSTRACT

In order to help students learning to develop computer programs, several computing education researchers have analyzed the com- piler error messages generated by novices’ a1empts to compile their programs. The goal is to help students diagnose the errors they make through the messages generated by the compiler. This paper builds on that previous work by applying a technique based on a heuristic well-known to programmers – fix the first error and ignore the rest – to the analysis of over 21 million compiler error messages from the Blackbox dataset.

We find that the ranks and frequencies obtained by considering all error messages are generally consistent with previously published lists, but when we consider first messages only, these ranks and frequencies are different. These differences could have important implications for teaching, and can inform tool design and future research efforts.

**KEYWORDS**:compiler error messages; cascading error messages; subsequent error messages; buried error messages; novice syntax errors; CS1; Blackbox; novice programmers; debugging

# 1. INTRODUCTION

Computer programming has been shown to pose many difficulties for students, and it has been cited as contributing to low motivation, low success rates, and high dropout rates [4]. One of the challenging aspects of learning to program is debugging, and in particular, learning how to interpret compiler error messages. We are interested in how students work with the feedback they receive from the compiler: what messages they see, how they use them, and how that process can be improved.

Novices are often advised to focus on the first compiler error message only. As Ben-Ari [8, p. 6] states: “Frequently, a single small mistake will cause the compiler to issue a cascade of messages… Do not invest any effort in trying to fix multiple error messages! Concentrate on fixing the first error and then recompile.” Similarly, Fincher and Barnes [17, p. 68] state: “Start with the first one, correct it in the source file, save the file and then recompile. Don’t try to fix all the errors in one go. This approach is particularly important if you have cascading errors, or multiple errors that have the same cause such as misspelled identifier.”

The overarching contribution of this work is a large-scale analysis of the effects of isolating first error messages from subsequent messages, using the Blackbox dataset [11, 33]. In addition, we present motivating results of a student survey, and introduce terminology that enables researchers and educators to discuss compiler error messages with more precision.

# 1.1 Student Perspectives

In 1984 DuBoulay and Ma1hew [9, p. 109] noted: “All novice programmers find that their initial programs are rejected by the com- piler in a flurry of incomprehensible error messages.” Our present observations indicate that this is still the case. We conducted a survey of 106 Software Engineering students in their CS1 module, of which 73 (69%) responded. These students were programming in C using Notepad++ and compiling at the command line using GCC. We sought to explore the levels of frustration, confusion and misconception surrounding compiler error messages (particularly multiple compiler error messages), and to determine whether extra help on these topics is desired. We present the results of this survey organized into the three headings of frustration & confusion, misconceptions, and requests for help, below.

**Frustration & confusion:** 85% of students reported finding compiler error messages to be a source of frustration with 92% reporting compiler error messages as being a barrier to progress. 64% reported finding multiple compiler error messages confusing. Misconceptions: 67% of students reported that when confronted with only one compiler error message, this error may not represent a true error in the code. 21% believe that compiler error messages after the first always represent true errors in code, and 11% believe that these error messages never represent true errors in code. When confronted with more than one compiler error message, only 22% try to fix the first error message reported, 43% try to fix one error which may or may not be the first reported, and 34% try and fix more than one error at the same time.

**Requests for help:** 100% of students reported that having additional help specifically on dealing with compiler error messages would help them in learning to program. 99% of students think that having the reasons for and nature of multiple “cascading” com- piler error messages explained to them would help them deal with compiler error messages more effectively.

These results suggest that deeper discussion and increased focus on error messages (including multiple error messages) are required.

# 1.2 A Taxonomy of Compiler Error Messages

To facilitate the discussion of compiler error messages, we introduce a taxonomy, shown in Figure 1. All messages fall into one of two types: first or subsequent. A first message is the first message reported by the compiler in response to a failed compile. The first message always corresponds to a genuine error in code. As even novices learn early, most failed compiles generate several additional error messages; we are not aware of an existing term for these not-first error messages. We use the term subsequent to refer to all messages reported by the compiler after the first message. Subsequent messages can be broken down, in turn, into two types:

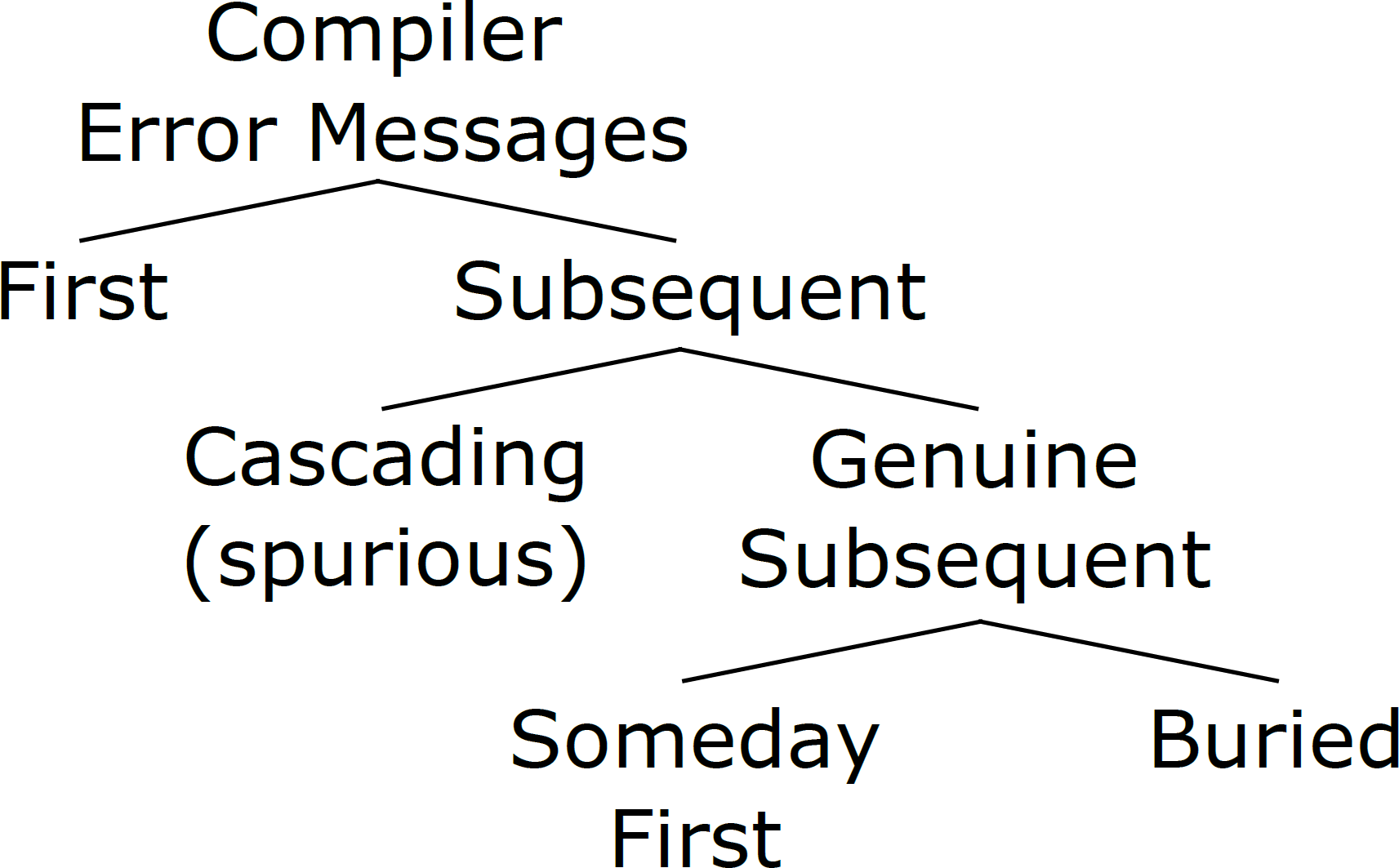
**cascading:** Cascading error messages are spurious messages caused by an error corresponding to an earlier error message, and provide no additional (useful) information. In short these are a result of compiler confusion. This definition is consistent Schorsch’s description: “multiple reported errors caused by a single mistake that can all be corrected by fixing the first reported error” [30, p. 169]

**genuine subsequent:** These are subsequent messages that arise due to a genuine error in code. Unlike cascading error messages these do not stem from an error for which a message has already been generated. These are the first- instance messages corresponding to genuine errors in code.

Genuine subsequent messages are then divided into two categories:

**someday first:** Someday first messages will eventually be- come a first message on a later compile.

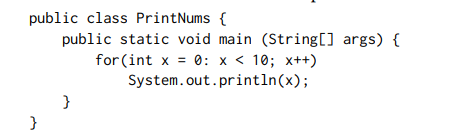
**buried: Buried** messages correspond to other legitimate errors that would have become first messages but do not, because something happens before they reach the top of the list. For example, the programmer might: see and fix the underlying error without help from the compiler, stop work on the program before fixing it, or delete the whole program and start over.



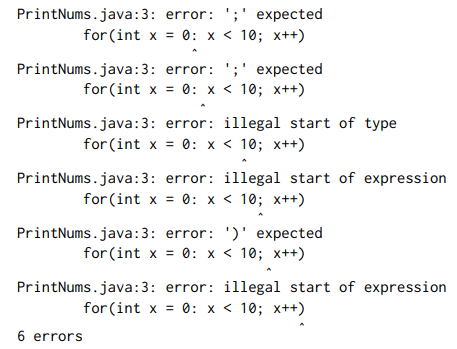
**Figure 1: A Taxonomy of Compiler Error Messages.**

# 1.3 A Motivating Example

Consider the following Java code which has an error on the third line: a regular colon (“:”) instead of a semicolon (“;”) between the initialization and the condition of the for loop.



This is an easy error to make, either as a result of misreading a book or notes, or as a simple typographical error. In fact both symbols are often on the same key. When this code is compiled (using javac, Java SE JDK 1.8.0 112) the following is returned by the compiler:



The first message accurately reflects the only genuine error in the code. The others are cascading messages resulting from the compiler being confused by the first (and only) error. Once the original error (corresponding to the first message) is rectified, all five cascading messages disappear. Time spent on the cascading messages is wasted, and confusion caused by them is unwarranted.

# 1.4 Research Questions

We wondered what changes if, instead of trying to help our students cope with long confusing lists of error messages, we focus on the first message generated by each compile. Specifically, our research questions are:

RQ1: Out of all compiler error messages, what proportion are first messages?

•

When first error messages are considered separately:

RQ2: How does the most-frequent list of messages change?

RQ3: How do the frequencies of specific error messages change?

In the following sections, we discuss the related work on which this paper builds (Section 2), describe our methodology (Section 3), present our results (Section 4), discuss the results and their im- plications (Section 5), consider threats to validity (Section 6), and conclude with a discussion of implications for teaching, research, and future work directions (Section 7).

# 2. RELATED WORK

In this section, we discuss previous work related to types of compiler error messages, their frequency, and the Blackbox dataset that we have used to investigate our questions.

# 2.1 Errors and Error Messages

Compiler error messages have been studied for several decades [4], are essential for debugging, and can present barriers to student success [7, 13, 15]. Despite their importance they present particular difficulties for novice students [20].

Several authors have analyzed student understanding of error messages, developed tools that a1empt to provide be1er compiler error messages, and/or supplemented standard messages with additional information [7, 13, 16, 18, 19, 24, 28, 32]. Other authors have used student error information to examine facets of student compilation or debugging behavior in general, including [1, 2, 6, 10, 12, 20]. Others, such as Ben-Ari [8], have wri1en guides that explain what error messages mean, with examples of code that would cause them and ways to fix them.

In a recent eye-tracking study of software-engineering under- graduates, Barik et al. [3] found that participants allocated a substantial portion of their task time to reading compiler messages, and that – not surprisingly – the greater a student’s difficulty with the error messages, the less likely he or she was to successfully debug the given program.

# 2.2 First Messages vs. Subsequent Messages

To help novices with compiler messages, Munson and Schilling [26] instructed them to focus on the first error message. Nevertheless, they found that their students did so on only 48.2% of compile a1empts.

Some novice programming environments, such as CAP [30] and BlueJ [23], have forced students to focus by only displaying the first message. Consider our example in Section 1. Run on the same example, BlueJ (3.1.7) displays only the first message (’;’ expected), instead of all six messages. Ko¨lling explains this decision in the context of Blue, a predecessor to BlueJ:

Compilation in Blue is so fast (usually under a sec- ond for a class of 2000 lines) that the “Compile” but- ton works as a de-facto “Next Error” bu1on…. This has several advantages: the compiler is simpler (because no error recovery has to be a1empted) and the error messages are more accurate (because incorrect messages caused by preceding errors are avoided) [22, pp. 8-9].

# 2.3 Determining Error Message Frequencies

Common to many studies on error messages is a presentation of the “most common” errors (in code), or error messages (resulting from errors in code), following from the reasonable assumption that helping students with the errors/messages that they most commonly deal with would be most productive. In practice however, exactly what these lists contain and particularly how they were determined vary considerably. Nonetheless these lists often inform others, and many are used as the basis for further research. Below we briefiy describe some of these efforts. For a more complete summary of many of these ‘common error/message lists’ see [4].

An early list of Java errors was compiled by Hristova et al. [19], which began as a faculty survey on what educators believed to be the most common errors. Flowers et al. [18] chose nine common errors based on their experiences with students. Denny et al. [14] collected errors from student exercises using a custom “drill and practice” tool. Becker et al. [4, 5, 7] used an editor designed to collect and enhance error messages generated by the standard JDK compiler. Chan Mow [12] counted messages collected from three classes using JBuilder. Thompson [32] collected messages from an instrumented version of Eclipse. As an example of how such lists can have long lifetimes in informing other research, in 2017 Brown and Altadmri [10] used a set of 18 errors derived from the 20 originally presented in 2003 by Hristova et al. [19].

From these descriptions it is clear that the frequency lists re- ported are not always directly comparable: not all are based on compiler error messages (but errors in code); some combine certain related error messages; some use different compilers that have different messages; and others are ambiguous. By and large these studies ignore the fact that many students only ever encounter the first error message reported. One goal of this research is to develop a picture of first error messages on a large scale.

# 2.4 Why Blackbox?

Blackbox is a novel dataset for investigating questions about how novices learn to program. It includes student interactions with BlueJ – source code, compiler messages, etc. – collected automatically from BlueJ users around the world. Many thousands of users have opted in, agreeing to have their interactions made available (anonymously) for research. Thus, it is now possible to look at the compile information, code revisions, and error messages for student programs on a much larger scale than before [11, 33]. The dataset is fairly new, and its scale so large, that researchers are still exploring Blackbox’s potential. Most papers so far have considered errors and error frequency [2, 10, 21, 29]. One paper has discussed students’ use of GoTo-like control structures [31].

# 3. METHODOLOGY

We analyzed Blackbox error messages (excluding warnings) from United States users for the period Jan 1 - June 30, 2016 inclusive. We iterated through the data by first selecting all bluej start events from which we retrieved all valid session id values. For each session id we retrieved all compile events. From these we retrieved all error messages for each session id from the compile outputs table which also contains a shown fiag which indicates if this message was shown to the user. This corresponds to the first error message from the JDK output as BlueJ only shows the first error to the user. All messages that are not shown are subsequent messages, which includes all cascading messages.

Many Java error messages contain tokens from source code and may be unique. For instance the following error message contains a token corresponding to the package name edu.univ.my package:

package edu.univ.my\_package does not exist

In order to categorize error messages such as this we processed all messages using regular expressions stored in JSON files. This allowed us to categorize all such messages into the general category package ... does not exist. We did similar for all other messages that contained unique tokens. This resulted in a set of 202 error message categories. Other than those containing potentially unique tokens, we did not combine any similar error messages. The source code used for our research is available at [h1ps://github.com/](https://github.com/TTY112358/JavaErrorCollector) [TTY112358/JavaErrorCollector.](https://github.com/TTY112358/JavaErrorCollector)

# 4. RESULTS

We analyzed 21,511,098 error messages from 135,629 users initiating 1,444,260 BlueJ sessions and calculated frequencies for the following message categories: first (F ); subsequent (S); and all messages (A).

# 4.1 RQ1: Proportion of First Error Messages

Out of the 21,511,098 error messages in our dataset, 28.5% (6,139,013) were first messages. The average failed compile generated 3.50 error messages, 2.50 of which were subsequent error messages. When we focus on the first error messages only, a different picture of error message distribution begins to emerge.

# 4.2 RQ2: Changes in the Most-Frequent Lists

There is a gap between what instructors advise and what researchers examine: although educators have recommended focusing on the first message for some time, most of the literature analyzing com- piler error messages only considers the overall frequency of error messages. In this section we explore how the picture changes when taking a first-error-message-only view.

The top 85% of all first messages are shown in Table 1, sorted by increasing frequency rank (RF ), along with their corresponding percent frequencies (%F ). We refer to this set of error messages as F85. We choose 85% as this incorporates all individual errors with percent frequencies 1%. The size of F85 compared to A85 (the top 85% of all error messages) and S85 (the top 85% of subsequent messages) provides initial insight into the shiks that occur when isolating first and subsequent error messages. F85 contains 23 error messages, A85 contains 17, and S85 contains only 14.

≥

A85 consists of the 14 unshaded messages in column RA in Table

1 (that also appear in F85) plus three messages not in F85 (and therefore also not in Table 1). S85 consists of the 11 unshaded messages in column RS in Table 1 (that also appear in F85) plus the same three messages not in F85 (and therefore also not in Table 1). These three messages are shown below with their corresponding ranks as overall (RA) and subsequent (RS ) messages.

illegal start of type (RA = 10, RS = 10)

illegal character (RA = 13, RS = 11)

non-static variable cannot be referenced from a static con- text (RA = 17, RS = 14)

Compared to A85 and S85, a very different picture emerges for the most-frequent first error messages; scanning the ranks of each message in Table 1 reveals that the ranks RA and RS are more similar than either is to RF . Only in the case of rank 1 does the same message appear at the same rank for all three lists. Further, when investigating first error messages only, more error messages are brought into play. F85 is larger than A85 and S85; There are 9 messages in F85 that are not in A85 (RA shaded in Table 1, where RA > 17), and there are 12 messages in F85 that are not in S85 (RS shaded in Table 1, where RA > 14).

# 4.3 RQ3: Changes in Message Frequencies

The frequencies of individual error messages change notably when looking at first messages only. In Table 1, %F and %S are calculated by dividing the number of occurrences of the message at rank i by the total number of first and subsequent messages respectively. These values indicate the likelihood that a given message will appear as a first or as a subsequent message. For instance, Table 1 shows that missing return statement is 5.6 times more likely to appear as a first message (3.20%) than as a subsequent message (0.57%).

In order to facilitate this type of comparison we calculated the log2 fold-change between %S and %F [log2(%S /%F )]. When this value is 1, %S is twice %F ; 2 indicates that %S is four times %F ; and 3 indicates that %S is eight times %F . Unlike the simple calculation

%S /%F , the log2 fold-change has two advantages: First, it is centered on 0 – if %S and %F are equivalent, log2(%S /%F ) is 0). Second, it is symmetric – positive values mean %S is greater than %F , and negative values mean the opposite. In order to similarly compare F with A, we also calculated log2(%A/%F ).

# 5. DISCUSSION

The percentage of first error messages we found (28.5%), average number of messages per failed compile (3.50), and average number of subsequent messages (2.50) were comparable to those found by others. Murray [27] investigated Irish Blackbox users from July 1, 2015 to June 30, 2016 inclusive, including 3,708 users and 22,440 sessions. He found 23.1% first messages, with an average of 4.34per failed compile (3.34 subsequent). In a 1997 study of 42 novice Ada programmers, Moore et al. [25] reported 4.06 average error messages (3.06 subsequent) per failed compile.

The list of most-frequent first messages is notably different from the corresponding list for all messages, and from similar lists that have been reported before [11, 16, 20]. There are 23 distinct messages in the list of first messages, as opposed to 17 overall. The first-messages list includes nine messages that are not on the over- all list, and omits three that are (as discussed in Section 4.2). Of those appearing on both lists, only one has the same rank, and all have differences in frequencies. For example, missing return statement and ‘)’ expected are much more common as first messages, and class, interface, or enum expected is much less common as a first message.

Our most-frequent first messages include all messages with at least 1% frequency. Students using editors that only show first error messages, or students following advice to only focus on first messages would (statistically) see each of these error messages from 1 to 19 times each in an average session, given 100 failed compiles in a programming session, which is quite realistic based on our data. These students would therefore see nine different messages per session not in the overall lists of frequent error messages reported by the literature (that are based on overall frequencies). Further, these students would not see three common error messages that do feature in the literature. This not only affects the students, but CS1 lecturers and lab instructors in particular, as a significant portion of their time is devoted to helping students rectify error messages. As the authors of [14] note, as educators the more we understand about the nature of compiler errors and how students respond to them, the more effective our teaching can be.

# 6. THREATS TO VALIDITY

The limitations of the Blackbox dataset have been characterized by Brown et al. [11]. Notably, Blackbox only includes BlueJ users and they are predominantly novices; but we are primarily interested in novices. Also, BlueJ users don’t see the subsequent messages; but we are looking at compiler messages, not user responses to those messages, and the underlying messages for a given compile are essentially the same for BlueJ and other IDEs that use the standard JDK. BlueJ users might generate a different sequence of program versions, and therefore a different overall message frequency com- pared to users of other IDEs. We believe these effects are negligible, however, since our overall list of errors is similar to lists found by other researchers.

Our results may also be affected by the fact that BlueJ is an opt-in dataset. Thus, the results of this study may be subject to volunteer bias. This is an issue we share with all studies based on Blackbox, and indeed, most computing education studies based on human subjects data. Finally, our sample, while large, is still only a subset of the Blackbox data. It is possible that non-United-States users or different time periods would show different characteristics.

# 7 CONCLUSIONS AND IMPLICATIONS

When teaching students, it is desirable to focus only on the first messages as often advised by educators and enforced by some novice programming environments. Our contributions are a taxonomy for discussing compiler error messages more precisely and data on how the error message landscape changes when focusing on first messages only. When we consider first messages only, message ranks and frequencies are different. We analyzed over 21 million compiler error messages from the Blackbox dataset and found that only ∼28.5% of all error messages are first messages, and that the list of the most frequent first messages is markedly different from the corresponding list based on all compiler error messages.

We introduced a new taxonomy to make it possible for researchers and instructors to talk about errors with more precision. The Black- box dataset includes all compiler error messages, with the first from each failed compile tagged as a “first message.” In theory, the subsequent messages can be broken down into three groups: the “cascading messages” (spurious messages, which disappear when some real error is fixed); the “someday first messages” (correspond- ing to a distinct error and become a first message on a later compile); and the “buried messages” (corresponding to a distinct error but never become first messages). Future work will look at categorizing errors according to this taxonomy.

Our results also have implications for research. While the first messages are most relevant to investigations of student interactions with the compiler, studies that use compiler messages as a proxy for (or stepping stone towards identifying) student errors might choose to look at first messages plus buried messages. Designing and implementing an algorithm to extract the buried messages from Blackbox would be an interesting challenge in itself.

# 8. ACKNOWLEDGMENTS

The authors would like to acknowledge the BlueJ team at King’s College London and the University of Kent for providing the Blackbox dataset that made this research possible.