计算机科学
COMPUTER SCIENCE

R语言及其核心包缺陷的实证研究

王子元，卜德欣，李凌菱，张霞

引用本文

王子元，卜德欣，李凌菱，张霞.R语言及其核心包缺陷的实证研究[J]．计算机科学, 2022, 49(12): 89-98.

WANG Zi-yuan, BU De-xin, LI Ling-ling, ZHANG Xia. Empirical Study on Defects in R Programming Language and Core Packages [J]. Computer Science, 2022, 49(12): 89-98.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220200181

R语言及其核心包缺陷的实证研究

王子元　卜德欣　李凌菱　张　霞
南京邮电大学计算机学院　南京 210023

摘　要　R语言提供了多种统计计算的功能,并被认为是最适合人工智能领域的程序设计语言之一。语言功能的正确实现是R语言程序正确运行的必要前提,但R语言中不可避免地存在着诸多软件缺陷。文中对R语言及其核心包中的历史缺陷进行了实证研究。通过分析R语言及其核心包中的7 020个缺陷报告发现:1)缺陷所涉及的35个R语言版本中R 3.1.2,R 3.0.2,R 3.5.0所含缺陷的数量较多,这些缺陷大量分布于Documentation,Graphics,Language等少数组件中;2)缺陷优先级整体较高的组件依次是Startup,Installation和Analyses,缺陷严重程度整体较高的组件依次是I/O,Installation和Accuracy,缺陷的优先级和严重性之间存在中等强度的秩相关;3)约78%的缺陷可在一年之内被修复;4)语义错误是缺陷最常见的根本原因,其中缺少功能和数据处理错误在各个阶段均占有较高的比例。这些发现揭示了R语言及其核心包中历史缺陷的一些基本规律,可在一定程度上帮助R语言开发人员提高开发质量,帮助R语言维护人员更高效地检测和修复缺陷,并帮助R语言的使用者规避潜在风险。

关键词:R语言;实证研究;软件缺陷;缺陷分布;缺陷修复;缺陷的根本原因
中图法分类号　TP311

Empirical Study on Defects in R Programming Language and Core Packages

WANG Zi-yuan,BU De-xin,LI Ling-ling and ZHANG Xia
School of Computer Science,Nanjing University of Posts and Telecommunications,Nanjing 210023,China

Abstract　The R programming language that provides a variety of statistical calculation functions is considered to be one of the programming languages most suitable for artificial intelligence. The correctness of the language implementation is a prerequisite for the correctness of the programs developed with such a language. However,there are inevitably many defects in the R programming language. This paper conducts an empirical study on defects in the R programming language and its core packages. By analyzing 7020 issues,we find that:1) Among all the 35 versions involved in these defects,there are the most defects in R 3.1.2,R 3.0.2 and R 3.5.0,and these defects are primarily distributed in a few components such as Documentation,Graphics,Language. 2) The components with higher overall defect priority include Startup,Installation and Analyses,and the components with higher overall defect severity include I/O,Installation and Accuracy. There is a significant intermediate correlation between the priority and severity of the defects. 3) About 78% of defects could be repaired within one year. 4) Semantic faults are the most frequent root cause of defects,in which the "missing feature" and "processing" are more than others. These findings reveal some laws of defects in the R programming language and its core packages. It can assist developers of the R programming language in improving their development quality,assist maintainers of the R programming language in detecting and repairing defects more effectively,and suggest users of the R programming language evade potential risks.

Keywords　R programming language,Empirical study,Software defect,Distribution of defects,Defect repair,Root cause

1　引言

R语言是一种用于统计计算的解释型程序设计语言,同时也被认为是最适合于人工智能领域开发的语言之一。据TIOBE公布的数据[1),]R语言长期居于热门程序设计语言的前十位。R语言中分类、聚类、统计建模、统计检验、时间序列分析、数据可视化等核心功能主要通过8个核心包(Package)来提供。

一段R语言应用程序若要正确运行,程序代码的正确性、R语言实现的正确性以及被调用包的正确性缺一不可。

1) https://www.tiobe.com/tiobe-index/

到稿日期:2022-02-27　返修日期:2022-06-03
基金项目:国家自然科学基金(61772259)
This work was supported by the National Natural Science Foundation of China(61772259).
通信作者:王子元(wangziyuan@njupt.edu.cn)

对R语言用户来说,如果所编写的应用程序由于R语言实现的问题或是被调用包的问题而不能正常运行,则会在调试程序时面临很多难题。但作为一类基础软件,R语言及其核心包也不可避免地存在诸多缺陷。深入探索和理解这些缺陷,对于保证和提高R语言本身的质量,乃至促进R语言生态的健康发展都具有重要意义。

R语言的开源为R语言及其核心包中缺陷的实证研究提供了必要的条件。但遗憾的是,尽管已有很多关于开源软件缺陷的实证研究[1-5],但针对R语言及其核心包中缺陷的实证研究目前仍是空白。

因此,本文对R语言及其核心包中的历史缺陷进行了实证研究。通过分析2021年4月前提交的7020个issue发现:1)缺陷较多地分布于R 3.1.2,R 3.0.2,R 3.5.0等版本,以及Documentation,Graphics,Language等组件中;2)约98%的缺陷的优先级为最低级别的P5,约67%的缺陷的严重性为中间层次的Normal,而Startup,Installation和Analyses组件中的缺陷整体优先级较高,I/O,Installation和Accuracy组件中的缺陷整体严重程度较高,缺陷的优先级和严重性之间存在中等强度的秩相关;3)大多数缺陷可在一年之内被修复,但也有近21%的缺陷需要超过一年的时间才能修复;4)语义错误、环境与配置、文档错误是缺陷较为常见的根本原因,而在语义错误中缺少功能的错误和数据处理错误则占有较高比例。

这些发现揭示了R语言及其核心包中缺陷的一些基本规律,可在一定程度上帮助R语言开发人员提高开发质量,帮助R语言维护人员高效地检测和修复缺陷,并帮助R语言使用者规避潜在风险。

总之,本文研究工作的主要贡献包括:1)收集并整理了R语言及其核心包中的历史缺陷,并进行了较大规模的实证研究;2)分析了R语言及其核心包中缺陷的分布规律,提示一些模块应得到更多关注;3)统计了R语言及其核心包中缺陷的修复时长,分析了修复时长与缺陷重要性间的关系;4)抽样分析了R语言及其核心包中缺陷的根本原因,提示了开发人员应避免的常见问题。

2 实证研究方案

本节首先列举了4个方面的研究问题,然后介绍了研究对象以及数据筛选和抽样的详细过程,最后针对每个研究问题给出了具体的研究方案。

2.1 研究问题

本文主要关注以下4个方面的研究问题。

RQ1(缺陷的分布):R语言及其核心包中的历史缺陷与版本之间是否存在相关性?这些缺陷较多地出现在哪些组件中?

RQ2(缺陷重要性):不同优先级和严重性的缺陷是如何分布的?缺陷优先级和严重性间是否存在相关性?缺陷所在组件与缺陷优先级间、缺陷所在组件与缺陷严重性之间是否存在相关性?

RQ3(缺陷修复):修复R语言及其核心包中的历史缺陷需要多长时间?缺陷的优先级、严重性是否会影响缺陷的修复时长?

RQ4(缺陷的根本原因):缺陷发生的根本原因是什么?随着语言的成熟,不同历史阶段缺陷发生的根本原因在分布上有何差异?

2.2 研究对象

R语言及其核心包的缺陷仓库位于R语言bug跟踪系统[1)]。我们将issue提交的时间范围限定为1998年8月至2021年4月,共收集到7020个issue。其中"Status"字段为"closed"且"Resolution"字段为"fixed"的issue共有4293个,它们表示缺陷已被正确修复,故缺陷报告被关闭。

2.2.1 重复的缺陷报告

人们可能会就同一个缺陷提交不同的缺陷报告,于是就出现了重复的缺陷报告[6]。在R语言及其核心包的缺陷仓库中,如果某一个缺陷存在与之重复的缺陷报告,则其报告中会被标记"Duplicate",且该标签之后会列举所有与之重复的缺陷报告的ID。例如,缺陷报告bug16755的"Duplicate"标签后列举了bug16763,bug16765,bug16770,bug16786,bug16781这5个ID,它们均与bug16775重复,并且也被标记了"Duplicate"标签。事实上,它们正是R语言缺陷仓库中重复次数最多的缺陷报告。

表1列出了重复缺陷报告的信息。表1中,第一行为重复次数,第二行为具有指定重复次数的缺陷报告的数量。可见,绝大部分(99.06%)的缺陷报告都是不重复的。

表1 重复的缺陷报告的分布

Table 1 Distribution of duplicate issues

#Duplicate	0	1	2	3	5
#Issues	6954	43	6	2	1

重复的缺陷报告往往是在较短的时间范围内被提交的,未发现同一个人反复就同一缺陷提交报告的情况。例如,bug16755及与之重复的缺陷报告均是在2016年3月10日这一天被提交的,这说明R语言用户的活跃程度较高。缺陷被重复报告一方面反映出了该缺陷所在的功能或模块的使用频率较高,另一方面也说明该缺陷的影响范围较广。因此,R语言及其核心包的开发人员、测试人员、维护人员均有必要对重复缺陷报告所涉及的功能或模块多加关注。

在后续工作中,我们通过扫描"Duplicate"标签来识别并剔除重复的缺陷报告,只保留主报告。

2.2.2 重新打开的缺陷报告

在软件开发和维护过程中,可能会将已关闭的缺陷报告重新打开,并对其作进一步的分析和处理。缺陷报告被重新打开的现象主要出现在以下两种情况下[7]:1)在缺陷被修复且通过了测试之后,开发人员将缺陷报告的状态修改为"closed fixed",但在后续使用过程中依然出现了bug,这说明缺陷未被完全修复,所使用的测试用例也不够全面;2)缺陷报告最早被开发人员认为是无效报告并将其关闭,但在后续的

1) https://bugs.r-project.org/bugzilla/

开发过程中发现此前的判断有误，故需要重新打开缺陷报告并进行处理。缺陷报告被重新打开的现象是难以完全避免的，但如果缺陷报告被多次重复打开且这一现象频繁出现，则说明该软件的开发过程的管理存在异常[8-9]，这会严重浪费开发人员和维护人员的精力。

在 R 语言及其核心包的缺陷仓库中，有些缺陷报告会给出缺陷状态变更的详细历史信息，而有些则不会。在我们所收集的缺陷报告中，共有 3622 个缺陷报告给出了状态变更历史信息。尤其是在 2010 年之后，这样的缺陷报告共有 3210 个，所占比例高达 88.63%。对于这 3622 个提供了状态变更历史信息的缺陷报告，我们利用“Added”标签判断“Reopened”字段的出现次数，以确定缺陷报告被重新打开的次数。对于未给出状态变更历史信息的缺陷报告，我们假设将其重新打开的次数为 0。

表 2 列出了缺陷报告重新打开次数的分布情况。表 2 中，第一行为重新打开的次数，第二行为具有指定重新打开次数的缺陷报告的数量，可见绝大部分(98%)的缺陷报告都未被重新打开。

表 2 缺陷报告重新打开次数的分布

Table 2 Distribution of reopened issues

# Reopen	0	1	2	3
# Issues	6871	78	4	1

重新打开次数最多的是 bug15212，该缺陷导致无法创建 22 个以上的线程，它被重新打开了 3 次。重新打开次数紧随其后的是 bug14231，bug16362，bug17668 和 bug180631。这 5 个缺陷报告被重新打开的原因都属于上述第一种情况。

无论是何种原因导致了缺陷报告的重新打开，都可以说明该缺陷的理解难度或是修复难度较高，对应的功能或模块可能更复杂，这就要求 R 语言及其核心包的开发人员和维护人员对这些功能和模块投入更多的关注。

2.3 研究方案

下文将针对每一个研究问题阐述实证研究的具体实施方案。

为了回答 RQ1，我们将统计缺陷在 R 语言不同版本中的分布情况，分析版本与缺陷数量之间的关系。此外，我们还将对近十年来 R 语言版本变化以及相应版本上缺陷数量的变化情况进行分析，寻找缺陷数量变化的规律。并且，我们将统计缺陷在不同组件中的分布情况，尤其是那些已经被修复的缺陷在组件中的分布情况。对于 R 语言的开发人员来说，缺陷较多的 R 语言版本和组件是值得关注的。

为了回答 RQ2，我们将统计 R 语言中具有不同优先级的缺陷和不同严重等级的缺陷的数量，并分析缺陷的优先级和严重性等级之间的相关性。此外，还将按照缺陷的优先级和严重性对缺陷所涉及的组件进行排序。基于“组件越重要则该组件中缺陷的优先级越高”和“组件越重要则该组件中缺陷的严重性越高”的假设，R 语言的开发人员需要关注缺陷整体优先级较高和严重性较高的组件。

为了回答 RQ3，我们将统计各个缺陷的修复时长，并分析缺陷修复时长与缺陷优先级、缺陷严重性之间的相关性。有 35 个缺陷报告由于缺少报告提交时间或是最后修改时间而无法计算修复时长，因此在本环节会被剔除。R 语言的开发和维护人员应合理分配用于缺陷修复的时间和精力，以使缺陷的生存期尽可能短。

为了回答 RQ4，首先将在“Component”字段或是状态标签中标记了“Wishlist”的 477 个报告剔除，因为它们的内容是提议添加新功能而不是报告缺陷。将缺陷报告按提交时间划分为 5 个组，表 3 列出了 5 组缺陷的数量，注意有 14 个缺陷报告由于未提供提交时间而在本环节被剔除。由于缺陷报告数量庞大，对所有缺陷的根本原因一一进行人工分析是不现实的，因此我们采取无放回抽样的方式在每组中随机抽取 50 个缺陷进行根本原因的人工分析。为了确保分析质量，人工分析会进行多轮迭代。R 语言的开发人员应在后续开发中重点关注经常出现的缺陷的根本原因。

表 3 不同时间段缺陷的数量

Table 3 Issues reported during each time

Time	Before filter Number/ Proportion	After filter Number/Proportion
1998-08－2000-12	624/8.97%	611/9.45%
2001-01－2005-12	1794/25.80%	1702/23.33%
2006-01－2010-12	1381/19.86%	1277/19.76%
2011-01－2015-12	1944/27.96%	1812/28.04%
2016-01－2021-04	1197/17.21%	1061/16.42%

3 缺陷的分布情况

本节将研究 R 语言及其核心包中历史缺陷的分布情况，包括在各版本、各组件中的分布情况。

3.1 缺陷在不同版本中的分布

R 语言官方规定，发行时间超过 5 年的版本被认为是 old 版本，而开发版本 R-devel(trunk)则体现了将出现在下一个 R 语言版本中的修订。后续的分析将不考虑 old 和 R-devel (trunk)版本中的缺陷。根据语义化版本控制规范[1)]，R 语言的版本号采用 x.y.z 的形式。缺陷所对应的版本号在缺陷报告中以“Version”标签标识。

表 4 列出了 3 个主版本中缺陷的数量，可见早期版本 R 2.y.z 中的缺陷最多，占比达 52.38%。

表 4 主版本中的缺陷数量

Table 4 Number of defects in each major version

Major Version	Number/Proportion
R 2.y.z	2710/52.38%
R 3.y.z	2271/43.89%
R 4.y.z	193/3.73%

表 5 列出了缺陷数量排名前十的二级版本中的缺陷数量，以及缺陷数量排名前十的标准版本中的缺陷数量。在 14 个二级版本中，R 3.0.x，R 3.1.x，R 2.15.x 的缺陷数量排名前三。在 35 个标准版本中，R 3.1.2，R 3.0.2，R 3.5.0 的缺陷数量较多，前两者都属于 R3.y.z 中相对早期的版本，而

1) https://semver.org/

R 3.5.0 则是 R 3.5.z 系列中的第一个版本。

表 5 各二级版本与标准版本中的缺陷数量(top 10)

Table 5 Number of defects in each minor version and normal version (top 10)

Minor Version	Number/Proportion	Normal Version	Number/Proportion
R 3.0.x	406/16.38%	R 3.1.2	156/8.77%
R 3.1.x	357/14.41%	R 3.0.2	153/8.60%
R 2.15.x	328/13.24%	R 3.5.0	145/8.15%
R 3.2.x	319/12.87%	R 3.0.1	125/7.03%
R 4.0.x	193/7.79%	R 3.0.0	113/6.35%
R 3.5.x	173/6.98%	R 3.1.0	105/5.90%
R 3.3.x	171/6.90%	R 3.1.1	77/4.33%
R 2.12.x	94/3.79%	R 3.2.0	75/4.22%
R 2.13.x	93/3.75%	R 3.2.1	73/4.10%
R 3.6.x	86/3.47%	R 3.2.3	72/4.05%

通过表 3 中的数据可以看出，2011－2015 年提交的缺陷报告数量最多，占总量的 27.96%；其次是 2001－2005 年提交的缺陷报告，其占比为 25.80%。近年来，缺陷报告的数量相对较少，一方面是由于 R 语言版本迭代速度相对变慢，另一方面是由于 2016 年 7 月 9 日之后只有“成员”(包括以前提交过缺陷报告的人)才可以在 R 语言缺陷仓库中提交新的缺陷报告。

3.2 缺陷在不同组件中的分布

剔除重复的缺陷报告后，我们共收集到 6 954 个带有组件标签“Component”的缺陷报告，它们均只涉及一个组件。存在部分缺陷报告的组件字段信息无法用于定位所在组件，如“Mac GUI/Mac specific”及“System-specific”分别表示发生于 Mac OS 平台及 Windows 和 Mac OS 之外的其他平台上的缺陷，这类缺陷共有 2076 个。此外，组件字段为“Misc”的缺陷也很多，这些缺陷无法被归类至某个特定组件。也就是说，共有 3212 个缺陷无法被归类至某一个特定组件，占缺陷总数的 46.19%。

表 6、表 7 分别列出了全部缺陷报告的组件分布情况，以及已修复且关闭的缺陷报告的组件分布情况。由表 6、表 7 可知，Documentation 组件中的缺陷最多，其次是 Graphics 和 Language。我们注意到，表 6 和表 7 中缺陷数量排名前十的组件是相同的，但 Analyses，Models，I/O，Accuracy，Installation 这 5 个组件在排名上发生了一定的变化。

表 6 全部缺陷报告的组件分布

Table 6 Distribution of defects in each component

Component	Number/Proportion
Misc	1 136/16.34%
Documentation	717/10.31%
Graphics	544/7.82%
Language	411/5.91%
Add-ons	351/5.05%
Accuracy	346/4.98%
Analyses	338/4.85%
Installation	337/4.85%
I/O	308/4.43%
Models	260/3.74%
S4methods	54/0.78%
Startup	51/0.73%

表 7 已修复且关闭的缺陷报告的组件分布

Table 7 Distribution of fixed defects in each component

Component	Number/Proportion
Documentation	595/13.86%
Misc	525/12.23%
Graphics	398/9.76%
Language	276/6.43%
Add-ons	237/5.52%
Analyses	223/5.19%
Models	183/4.62%
I/O	174/4.05%
Accuracy	171/3.98%
Installation	162/3.77%
S4methods	30/0.70%
Startup	27/0.63%

下面关注缺陷数量排名靠前的 3 个组件。

(1)Documentation 组件。文档是用户学习使用 R 语言的一个参考标准，文档组件的缺陷可能引起用户在使用系统调用时引发潜在的错误，对于任何一个大量开发者参与的开源软件系统来说，文档的编写、管理和维护都是较为困难的。文档组件中的缺陷共有 717 个，占全部缺陷的 10.31%。文档错误主要有两类：一类是文档的编写错误，例如 bug17466 报告了文档中对“deviance residuals”描述的错误，bug16169 指出了“stl”文档中“s.window”参数的错误，在“stl”文档中给出的示例“s.window=4”，而 Cleveland 等指出参数至少为 7；另一类是软件版本更新引发的错误，bug16196 在函数名和描述之间需要几个空格，这在" R 扩展"文件中没有说明。bug16269 是文档中未更新翻译页面的 URL 问题。

(2)Graphics 组件。Graphics 组件提供了一系列绘画接口，这些接口参考 canvas 绘画接口来实现。如果 Graphics 组件出现缺陷，则会直接影响 R 语言的绘图功能，而用于数据展示的图形环境正是 R 语言的特色之一。Graphics 组件中的缺陷共有 544 个，占全部缺陷的 7.82%。Graphics 组件中的缺陷均与画图有关，如图画颜色、画图边界、图形空间维度问题等。bug14511 报告了在“灰色”颜色模型下创建 PDF 会将填充颜色转换为灰色，但笔触仍然是彩色的问题。bug14833 报告了绘制对数轴图，并使用添加带有手动刻度线位置的附加轴时，下边界会超出绘制区域。bug1432 指出“filled.contour()”在矩阵维度 z 与参数 x，y 不匹配时未显示错误或警告消息的问题。

(3)Language 组件。由于 R 语言自身的问题导致使用 R 语言编写的程序不能正常运行，这样的缺陷往往会涉及 Language 组件。Language 组件中的缺陷共有 411 个，占比为 5.91%。Language 组件中常见的缺陷类型有：算法错误、方法调用错误。例如，bug16393 中说明若“x”是一个逻辑矩阵，那么在 R-devel 中，“! x”就变成了一个向量。bug1073 出现的原因是在“list”形式的数据中，“==”的操作未定义，而缺陷的报告者在“list”形式的数据中使用了“==”。

4 缺陷的优先级和严重性

本节将从优先级和严重性这两个角度，研究 R 语言及其核心包中历史缺陷的重要性分布。

4.1 缺陷的优先级

缺陷的优先级使用标签“Importance”来标识，共有 P1－P5 这 5 个级别。P1 是最高优先级，通常只有修复了这些高优先级的错误，软件系统才能发布。而优先级为 P2 的缺陷是较为严重的，需要在下一版本发布前修复。P1－P3 通常保留给非常重要的 API 崩溃、严重退化或损坏。P4 级别的缺陷需要正常排队等待修复。P5 是最低优先级[10]。

表 8 列出了各优先级缺陷的数量和比例。

表 8 缺陷的优先级分布

Table 8 Distribution of priorities of defects

Priority	Number/Proportion
P1	30/00.43%
P2	19/00.27%
P3	48/00.69%
P4	10/00.14%
P5	6 847/98.46%

优先级为 P5 的缺陷共有 6 847 个，比例高达 98.46%。其次是优先级为 P3 和 P1 的缺陷，但占比均不超过 1%。R 语言及其核心包中的绝大部分缺陷只具有最低优先级，说明其修复需求都不紧迫。

人们在直觉上认为，重要组件中的缺陷影响范围大、危害性强，因此应被指定较高的优先级。我们统计了各组件中不同优先级缺陷的数量和比例，并参考了 Sun 等使用的方法[11]，对组件的整体缺陷优先级进行了排序。给定组件 c 和缺陷优先级 $p\in\{1,2,3,4,5\}$，函数 $w(c,p)$ 为具有指定优先级 p 的缺陷在组件 c 中的占比，可使用式(1)定义任意两个组件 c_1 和 c_2 之间整体缺陷优先级的顺序。

$$\theta(c_1,c_2,p)=\begin{cases}>, & w(c_1,p)>w(c_2,p)\\ <, & w(c_1,p)<w(c_2,p)\\ \theta(c_1,c_2,p+1), & w(c_1,p)=w(c_2,p)\end{cases}\quad(1)$$

各组件中不同优先级缺陷的数量和比例如表 9 所列。而根据式(1)计算各组件的整体缺陷优先级次序，可以发现整体缺陷优先级最高的组件是 Startup，其次是 Installation 和 Analyses。

表 9 组件中不同优先级缺陷的数量和比例

Table 9 Number and proportion of defects with different priorities in each component

Component	P1	P2	P3	P4	P5
Documentation	2/0.28%	4/0.55%	8/1.11%	2/0.28%	705/97.78%
Graphics	1/0.18%	1/0.18%	7/1.27%	0	541/98.36%
Language	3/0.72%	0	4/0.96%	0	411/98.30%
Add-ons	0	2/0.57%	0	0	348/99.43%
Accuracy	2/0.57%	2/0.57%	4/1.14%	2/0.57%	341/97.15%
Installation	5/1.46%	1/0.29%	2/0.58%	2/0.58%	333/97.08%
Analyses	4/1.18%	1/0.29%	4/1.18%	0	331/97.35%
I/O	3/0.96%	2/0.64%	4/1.28%	1/0.32%	302/96.79%
Models	3/1.15%	0	2/0.77%	1/0.38%	255/97.70%
S4methods	0	0	1/1.82%	0	54/98.18%
Startup	1/1.96%	0	2/3.92%	0	48/94.12%
Translations	0	0	0	0	25/100.00%

4.2 缺陷的严重性

缺陷的严重性同样使用“Importance”标签来标识，共有 7 个级别。其中，Blocker 的严重程度最高，存在这类缺陷的版本不允许发行；Critical 缺陷包括严重的内存泄漏、数据丢失等；Major 缺陷会导致功能严重受损，严重程度更低一些的功能性问题分别用 Normal 和 Minor 表示，用户界面问题一般为 Trivial，而 Enhancement 的严重程度最低[12]。

表 10 列出了不同严重程度的缺陷的数量和比例。可见，Blocker 缺陷的数量极少，Normal 缺陷的数量最多。高严重性(Blocker，Critical，Major)的缺陷只有 504 个，比例之和为 7.25%，这说明 R 语言及其核心包中大部分缺陷都未造成严重危害。

表 10 缺陷的严重性分布

Table 10 Distribution of severities of defects

Severity	Number/Proportion
Blocker	56/0.81%
Critical	137/1.97%
Major	311/4.47%
Normal	4 663/67.05%
Minor	633/9.10%
Trivial	225/3.24%
Enhancement	929/13.36%

人们在直觉上认为，重要组件中的缺陷往往会造成比较严重的危害，因此应指定较高的严重等级。我们统计了各组件中不同严重程度缺陷的数量和比例，并参考了上文对组件整体缺陷优先级进行排序的方法，对组件的整体缺陷严重性进行了排序。给定组件 c 和缺陷严重性 $s\in\{1,2,3,4,5,6,7\}$(s 的取值由小到大分别对应 Blocker，Critical，Major，Normal，Minor，Trivial，Enhancement)，函数 $f(c,s)$ 为具有指定严重程度 s 的缺陷在组件 c 中的占比，可使用如下公式定义任意两个组件 c_1 和 c_2 之间整体缺陷严重程度的顺序：

$$\varphi(c_1,c_2,s)=\begin{cases}>, & f(c_1,s)>f(c_2,s)\\ <, & f(c_1,s)<f(c_2,s)\\ \varphi(c_1,c_2,s+1), & f(c_1,s)=f(c_2,s)\end{cases}\quad(2)$$

各组件中不同严重程度缺陷的数量和比例如表 11 所列。而根据式(2)计算各组件的整体缺陷严重性次序，可以发现整体缺陷严重程度最高的组件是 I/O，其次是 Installation，Accuracy 排名第三。我们注意到，其中 Installation 组件的整体缺陷优先级也是较高的，这说明该组件中的缺陷值得 R 语言的开发和维护人员重点关注。

表 11 各组件中不同严重程度缺陷的数量及其在该组件缺陷中的比例

Table 11 Number and proportion of defects with different severities in each component

Component	Blocker	Critical	Major	Normal	Minor	Trivial	Enhancement
Documentation	0	2//0.28%	6/0.84%	458/63.88%	76/10.60%	91/12.69%	84/11.72%
Graphics	4/0.74%	6/1.10%	22/4.04%	395/72.61%	54/9.23%	15/2.76%	48/8.82%
Language	4/0.97%	2/0.49%	30/7.30%	253/61.56%	53/12.90%	11/2.68%	58/14.11%
Add-ons	3/0.95%	8/2.54%	25/7.94%	251/79.68%	25/7.94%	7/2.22%	32/10.16%
Accuracy	9/2.60%	32/9.25%	30/8.67%	184/53.18%	37/10.69%	5/1.45%	49/14.16%
Installation	14/4.15%	12/3.56%	22/6.53%	179/53.12%	32/9.50%	10/2.97%	68/20.18%
Analyses	4/1.18%	14/4.14%	22/6.51%	235/69.53%	26/7.69%	3/0.89%	34/10.06%
I/O	2/8.12%	12/3.90%	26/8.44%	172/55.84%	43/13.96%	8/2.60%	45/14.61%
Models	4/1.54%	6/2.31%	16/6.15%	186/71.54%	16/6.15%	3/1.15%	29/11.15%
S4methods	0	3/5.56%	2/3.70%	35/64.81%	7/12.96%	1/1.85%	6/11.11%
Startup	0	3/5.88%	7/13.73%	25/49.02%	6/11.76%	2/3.92%	6/11.76%
Translations	0	0	0	13/52.00%	6/24.00%	2/8.00%	4/16.00%

4.3 缺陷的严重性与优先级之间的关系

人们一般认为“严重的缺陷的优先级也较高”。为了研究两者之间的关系，我们针对各个优先级统计了不同严重程度缺陷的数量及其在该优先级缺陷中所占的比例，如表 12 所列。

表 12 各优先级缺陷的严重程度及其在该优先级缺陷中的比例

Table 12 Number and proportion of defect with different severities for each priority

	Blocker	Critical	Major	Normal	Minor	Trivial	Enhancement
P1	8/26.67%	5/16.67%	5/16.67%	1/3.33%	8/26.67%	1/3.33%	2/6.67%
P2	0	2/10.53%	5/26.32%	5/26.32%	1/5.26%	0	6/31.58%
P3	3/6.25%	0	12/25%	21/43.75%	7/14.58%	0	5/10.42%
P4	0	1/10%	1/10%	5/50%	3/30%	0	0
P5	45/0.66%	129/1.88%	288/4.21%	4631/67.63%	614/8.97%	224/3.27%	916/13.38%

为了更准确地判断缺陷优先级与严重程度之间的关系，我们对两者进行了 Pearson 相关性分析以及 Spearman 秩相关性分析。表 13 列出了相关系数及相应的 P 值。Pearson 相关系数为 0.065，表明缺陷优先级与严重程度之间基本不存在线性相关的关系；而 Spearman 相关系数为 0.640 且 P 值非常接近 0，表明缺陷的优先级与严重程度之间存在中等强度的秩相关。这一结果与“严重的缺陷应被优先处理”的基本思想是吻合的。

表 13 缺陷优先级与严重性的相关性

Table 13 Correlation between priority and severity

	Correlation	P-value
Pearson	0.065	6.158×10^{-8}
Spearman	0.640	0

5 缺陷修复时长

我们将缺陷报告最后一次被修改的时间与缺陷报告被提交的时间之间的差值作为缺陷修复时长。其中，缺陷提交时间由“Reported”标签标识，最后一次修改时间由“Modified”标签标识。

表 14 列出了不同时间范围内被修复缺陷的分布。超过一半的缺陷在一天内被修复，约 78%的缺陷在一年之内被修复，修复时长超过一年的缺陷比例为 21.23%。对这些修复时间过长的缺陷进行研究，将是未来提高 R 语言质量的一个工作方向。

表 15 列出了缺陷修复时长的平均值、中位数、标准差以及最大值和最小值。通过研究发现，修复时长大于平均值的 1 421 个缺陷中有 1 131 个缺陷的首次修改时间距离提交时间较远。也就是说，缺陷报告未得到及时关注很有可能是导致缺陷修复时间过长的原因之一。

表 14 缺陷修复时长的分布

Table 14 Distribution of duration of defects

Duration/days	Number(/days)/Proportion
[0,1]	3 791/54.52%
(1,7]	677/9.73%
(7,30]	474/6.82%
(30,180]	378/5.44%
(180,360]	123/1.77%
(360,∞)	1 476/21.23%

表 15 缺陷修复时长的统计信息

Table 15 Statistics about duration of defects

(单位：d)

Mean	Median	SD	Max	Min
524.5	1	1 156.5	7 360	0

此外，不正确或不完全的修复会影响正常修复工作的秩序，也会导致缺陷修复时间的延长。例如，修复耗时最长的 bug412 在初次修复中未被完全修复，再次得到关注并被标记为“Unconfirmed”时为时已晚。

统计结果表明，缺陷修复时长与缺陷优先级之间、缺陷修复时长与缺陷严重程度之间均不存在显著的相关性关系，具体如表 16 和表 17 所列，这说明缺陷重要性的标记工作还有进一步提高的空间。缺陷仓库维护人员应结合多方面因素给缺陷贴上正确的重要性标签，这样开发人员才能准确地判断缺陷重要性，从而有序地开展缺陷修复工作。

表 16 缺陷修复时长与缺陷优先级之间的相关性

Table 16 Correlation between repair duration and priority

	Correlation	P-value
Pearson	0.276	0.653
Spearman	0	1

表 17 缺陷修复时长与缺陷严重性之间的相关性

Table 17 Correlation between repair duration and severity

	Correlation	P-value
Pearson	0.588	0.165
Spearman	0.536	0.215

6 缺陷产生的根本原因

6.1 缺陷根本原因的分析过程

如 1.3 节所述，我们对缺陷报告进行筛选、分组、抽样后，再进行根本原因的人工分析，分析对象为 5 个提交时间段内随机抽样的各 50 个缺陷报告。具体的分析过程包括根本原因类型标记和结果审核两个步骤。在根本原因类型标记阶段，对共计 250 个缺陷报告采用人工分析的方式来判断缺陷产生的根本原因，参考信息包括缺陷报告标题、详细描述、评论内容、拉取请求等。基于 Tan 等[1]以及 Wan 等[2]的研究，我们将缺陷根本原因分为如表 18 所列的几类，少量难以判断原因的缺陷的被标记为“其他”。在结果审核阶段，标记工作完成后对标记结果进行审核。

表 18 缺陷根本原因的分类及描述

Table 18 Description of roots causes of defects

根本原因类型	描述
内存	内存分配，内存管理错误导致内存泄漏、使用超过最大值等
语义	代码与需求规约或是程序员意图不符且不属于内存错误，可进一步细分
控制流错误	控制流程执行不正确
异常处理错误	没有适当的异常处理语句
数据处理错误	赋值语句、数值处理 、表达式求值中发生的错误
数据类型错误	基本数据类型转换出错或是函数需要支持其他数据类型
缺少功能	软件应有功能未被实现
函数调用错误	错误地调用了内部函数而引发的错误
外部调用错误	错误地调用了外部模块或是外部库而引发的错误
版本	版本更新引发的错误，不同版本向前兼容向后兼容等问题
平台	功能在 Windows，Linux，Unix 等不同平台不兼容
环境与配置	操作系统的配置错误以及其他影响软件功能的非代码错误
文档	文档针对某一功能的示例或注释有误
其他	因缺少信息而无法判断缺陷的根本原因

6.2 缺陷根本原因的示例

为了更好地理解不同类别的缺陷的根本原因，我们针对每一个分类各给出一个示例。

(1)语义

共有 7 类语义错误，分别举例如下。

1)控制流错误：bug421 因为 IF 控制信息错误而导致程序无法编译。

2)异常处理错误：bug17908 缺陷报告中出现异常时错误消息提及错误。

3)数据处理错误：bug319 报告函数 expand.grid(1:3) 返回值错误，应该返回“data.frame”类的对象，但返回了 NULL。

4)数据类型错误：bug500 定义 dimname 的类型无效，其类型必须是向量。

5)函数调用错误：bug411 约束优化器调用的参数值超出了约束范围。

6)外部调用错误：bug8422 外部库调用错误。

7)缺少功能：bug1073 在“list”模式下的“==”操作未定义。

(2)内存

bug14787 在尝试为 alpha blend 创建第二个缓冲区时系统崩溃。

(3)版本

bug15235 R 2.15.x 不支持 Makeinfo 5.0，因为 makeinfo 5.0 无法解析 R 的 doc 文件。R 2.15.3 中对此进行了记录。

(4)平台

bug6903 如果将 setwd() 放入已创建的目录，则“source()”即为 R 文件，在装有 OS X 的任何计算机上发生崩溃。

(5)环境与配置

bug1469 将无效的(零长度)参数范围传递给 loess.smooth 导致 R 崩溃。

(6)文档错误

bug9901 文档中“anlysis”出现拼写错误，应该为“analysis”。

6.3 缺陷根本原因的分布比例

表 19 列出了 5 个提交时间段内缺陷根本原因类别的分布情况，以及缺陷根本原因各类别所占比例在 95%置信度下的置信区间。由于一个缺陷的根本原因可能同时属于多个类别，对于这些同时标记了多个根本原因的缺陷报告会进行重复统计。

表 19 缺陷根本原因的分布

Table 19 Distribution of root causes of defects

根本原因类型	不同时间段提交的历史缺陷				
	2016-01－2021-04 数量/比例 置信区间	1998-08－2000-12 数量/比例 置信区间	2001-01－2005-12 数量/比例 置信区间	2006-01－2010-12 数量/比例 置信区间	2011-01－2015-12 数量/比例 置信区间
语义	30/60.00% [46.98%,73.02%]	27/54.00% [40.39%,67.61%]	26/52.00% [38.43%,65.57%]	15/30.00% [17.47%,42.53%]	37/74.00% [62.16%,75.87%]
控制流错误	5/10.00% [2.03%,17.97%]	1/2.00% [0,5.82%]	2/4.00% [0,9.32%]	2/4.00% [0,9.36%]	7/14.00% [4.61%,23.39%]
异常处理错误	1/2.00% [0,5.72%]	1/2.00% [0,5.82%]	0	0	1/2.00% [0,5.79%]
数据处理错误	7/14.00% [4.78%,23.22%]	7/14.00% [4.52%,23.48%]	13/26.00% [14.09%,37.91%]	3/6.00% [0,12.49%]	10/20.00% [9.17%,30.83%]
数据类型错误	1/2.00% [0,5.72%]	0	0	0	2/4.00% [0,9.30%]
函数调用错误	6/12.00% [3.36%,20.64%]	3/6.00% [0,12.49%]	4/8.00% [0.63%,15.37%]	1/2.00% [0,5.83%]	3/6.00% [0,12.43%]

(续表)

根本原因类型	不同时间段提交的历史缺陷				
	2016-01—2021-04 数量/比例 置信区间	1998-08—2000-12 数量/比例 置信区间	2001-01—2005-12 数量/比例 置信区间	2006-01—2010-12 数量/比例 置信区间	2011-01—2015-12 数量/比例 置信区间
外部调用错误	0	5/10.00% [1.81%,18.19%]	3/6.00% [0,12.45%]	4/8.00% [0.58%,15.42%]	0
缺少功能	10/20.00% [9.37%,30.63%]	10/20.00% [9.07%,30.93%]	4/8.00% [0.63%,15.37%]	5/10.00% [1.80%,18.20%]	14/28.00% [15.85,40.15%]
内存	2/4.00% [0,9.21%]	1/2.00% [0,5.82%]	4/8.00% [0.63%,15.37%]	2/4.00% [0,9.36%]	1/2.00% [0,5.79%]
版本	2/4.00% [0,9.21%]	1/2.00% [0,5.82%]	3/6.00% [0,12.45%]	2/4.00% [0,9.36%]	1/2.00% [0,5.79%]
平台	2/4.00% [0,9.21%]	3/6.00% [0,12.49%]	0	1/2.00% [0,5.83%]	1/2.00% [0,5.79%]
环境与配置	3/6.00% [0,12.31%]	7/14.00% [4.52%,23.48%]	5/10.00% [1.85%,18.15%]	7/14.00% [4.51%,23.49%]	4/8.00% [0.66%,15.34%]
文档	5/10.00% [2.03%,17.97%]	3/6.00% [0,12.49%]	7/14.00% [4.58%,23.42%]	3/6.00% [0,12.49%]	2/4.00% [0,9.30%]
其他	9/18.00% [7.79%,28.21%]	9/18.00% [7.51%,28.49%]	7/14.00% [4.58%,23.42%]	21/42.00% [28.51%,55.49%]	6/12.00% [4.66%,19.34%]

从表 19 中可以看到，语义错误的数量最多，共有 135 个，占抽样缺陷总数的 54.00%；其次是环境与配置导致的缺陷，占抽样缺陷总数的 10.40%；文档错误的数量排名第三，占抽样缺陷总数的 8.00%。语义错误中最常见的是缺少功能和数据处理错误。从时间变化的趋势来看，2016 年之前语义错误的比例呈逐渐下降的趋势，但 2016 年之后却大幅上升，一个可能的原因是近年来 R 语言的使用范围逐步扩大，使得用户对新功能的需求日益增长，进而导致功能性缺陷相关的语义错误在比例上有所升高。

7 启示

通过分析研究结果可得到如下的一些启示。这些启示可为 R 语言及其核心包的开发、测试、维护人员提供一定帮助，可为 R 语言用户提供一定参考，也可为相关领域的科研人员提供一些建议。

(1)缺陷的版本。缺陷数量最多的 3 个 R 语言版本分别是 R 3.1.2，R 3.0.2，R 3.5.0，前两者都属于 R3.y.z 中相对早期的版本，而 R 3.5.0 则是 R 3.5.z 系列中的第一个版本。由此可见，在软件迭代过程中应尤其注意那些变革较大的新版本。R 语言目前已发展至 R 4.0.0 版本，测试和维护人员当前应重点关注这一新版本中可能出现的缺陷。

(2)缺陷所在的组件。缺陷数量较多的组件依次是 Documentation，Graphics 和 Language，整体缺陷优先级较高的组件依次是 Startup，Installation 和 Analyses，整体缺陷严重程度较高的组件依次是 I/O，Installation 和 Accuracy。因此，R 语言及其核心包的开发、维护人员应该更多地关注这几个组件，加强对这几个组件的测试与维护。此外，对于使用 R 语言编写应用程序的用户，在调用上述组件所涉及的功能时也有必要加强单元测试，尽量避免由于 R 语言功能本身的缺陷而导致的软件失效。

(3)缺陷修复时长。近一半的缺陷可在一天之内被修复，大约 78%的缺陷可在一年之内被修复。修复时长超过一年的缺陷共有 1476 个，比例为 21.23%。这些缺陷修复时间过长的原因包括缺陷未能得到及时确认、缺陷报告不完善导致无法复现等。因此，R 语言及其核心包的开发和维护人员应及时审核用户提交的缺陷报告，以避免缺陷报告长期无人修复；而用户在提交缺陷报告时也应尽可能详细地描述缺陷发生的环境变量、前置条件、操作步骤等，以避免开发和维护人员就缺陷是否真实存在这一问题而产生分歧。此外，对于一些由于修复难度较高而导致迟迟无法得到修复的缺陷，可考虑寻求自动化的缺陷定位和缺陷修复工具的支持。

(4)缺陷优先级与严重性。缺陷修复时长与缺陷优先级、严重性之间没有体现出显著的相关性，但缺陷的优先级和严重性之间存在中等强度的秩相关。这说明维护人员在标记缺陷的优先级和严重性标签时具有较高的合理性。在后续的维护工作中，缺陷仓库的管理员应结合多种因素及时地为缺陷报告标记正确的优先级和严重性，这样开发和维护人员才能根据缺陷的优先级和严重性有序地开展缺陷修复工作。而对于那些被标记了高优先级、高严重性却长时间没有得到修复的缺陷，开发和维护人员也应加以关注。

(5)缺陷的根本原因。在 R 语言发展的早期，语义错误是最常见的缺陷根本原因。此后，随着 R 语言版本的迭代，由于平台、文档、环境配置等因素而导致的缺陷比例略有上升。但在近几年发布的版本中，语义错误的比例再次显著提高。语义错误中缺少功能的错误和数据处理错误在 R 语言发展的各个时期均占有较高的比例。这一方面提示了 R 语言开发人员在今后开发工作中应重点关注的方面，另一方面也提示了软件工程领域的研究人员有必要在该类错误的自动检测方面开展更多的研究工作。

8 有效性的威胁

本研究可能受到多种来自内部和外部的有效性威胁。内部威胁主要来自于缺陷的代表性，外部威胁主要涉及分析方法的正确性。

(1)缺陷代表性。本文在研究缺陷的各种特性时，均使用 1.3 节说明的方法进行数据的筛选与抽样，不存在人工干预的情况。我们在对缺陷产生的根本原因进行分析时，剔除了那些仅仅提出建议而未报告缺陷的 issue。我们采用无放回的抽样方式对缺陷进行随机取样，在统计各类型缺陷的根本原因时计算了 95%置信度下的置信区间。综上，本文研究的

缺陷信息是充分的。

(2)方法正确性。本文采取的研究方案充分参考了已有的实证研究工作。我们采用自动化分析与人工分析相结合的方式对收集到的大量缺陷报告进行分析，以减少误差。在分析缺陷根本原因的过程中，我们通过迭代的方式来降低个人主观性带来的误差，每一个缺陷报告均至少由两个人来完成根本原因的分析，如果出现意见不统一的情况还会进行讨论以达成共识。

9 相关研究工作

(1)缺陷的实证分析。Tan 等在 3 个大型开源项目 Linux 内核、Mozilla 和 Apache 中抽样了 2060 个缺陷进行缺陷特性的研究[1]。Wan 等对 8 个开源区块链项目中的缺陷进行了实证研究，手动检查了其中的 1108 个缺陷报告并进行了分类[2]。Razzaq 等研究了缺陷仓库中注释的特性，给出了缺陷判断时间和修复时间的计算方法，并提出了一个缺陷报告生命周期模型[3]。Bhattacharya 等对 Android 平台以及该平台上通信、工具、媒体等不同类别的 24 个开源应用中的缺陷进行了实证研究[4]。Saha 等对 4 个开源项目 JDT，CDT，PDE 和 Platform 中持续时间较长的缺陷从比例、严重性、分配、根本原因、修复的性质这 5 个角度进行了分析[5]。Sun 等从缺陷位置、触发缺陷的测试用例、缺陷持续时间、缺陷优先级等方面对 GCC 和 LLVM 两款编译器中的缺陷进行实证研究，并研发了一种用于检测编译器缺陷的工具 tkfuzz[10]。Sahoo 等从 Squid，Apache，Subversion(SVN)等 6 个服务器应用程序中随机选择了影响生产现场重现性的一些缺陷进行实证研究[11]。Le 等研究了等效模输入(EMI)，这种通用的编译器测试方法可以基于已有程序生成语义上等效的变异体，目前该方法已在 GCC 和 LLVM 中发现了数百个 bug[12]。Zaman 等分析了 Firefox 中的一些安全缺陷和性能缺陷[13]。Vijayakumar 等对 Firefox 应用程序组件上的缺陷修复进行了实证研究[14]。Li 等对 3000 多个缺陷和 4000 多个缺陷修复补丁进行了实证研究，分析了缺陷对代码库持续时间、补丁程序开发及时性、开发人员提供安全防护的程度等的影响[15]。Hanam 等挖掘了 134 个服务器端 JavaScript 项目中的 105000 个提交，对缺陷模式和修复模式进行了实证研究[16]。Nguyen 等对 10 个移动应用中的近 300 个与异常处理相关的缺陷及其修复进行了实证研究[17]。针对人工智能领域的应用，Sun 等对 3 个机器学习项目中的缺陷从缺陷类别、修复模式、修复规模、持续时间和软件维护这 5 个角度对 329 个已关闭的缺陷报告进行了分析，并检查了它们的演化过程[18]；Zhang 等从 Stack Overflow 问答和 GitHub 中收集基于 TensorFlow 的应用程序中的缺陷，并分析其症状和根本原因[19]；Zhang 等将微软深度学习平台上的 4960 个真实失效分为 20 类，并且针对其中的 400 个分析其根本原因和设计修复方案[20]。针对深度学习框架，Islam 等针对 Caffe，Keras，Tensorflow，Theano 和 Torch 分析 Stack Overflow 上的 2716 篇问答以及 GitHub 中的 500 个缺陷的修复，以了解缺陷的特征并发掘常见的反模式[21]；Du 等从缺陷分类、不同类型缺陷的频率和演变、缺陷类型与修复时间的相关性、Bohrbug 缺陷和 Mandelbug 缺陷等角度对 TensorFlow 的 2285 个缺陷进行了实证研究[22]。

(2)缺陷产生的根本原因分析。Grishma 等讨论了用于预测软件缺陷根本原因的 6 种聚类方法[23]。Hirsch 等通过挖掘和分析 GitHub 项目中的缺陷报告，提出了一种有监督机器学习方法，用于预测缺陷的根本原因[24]。Lal 等提出了一种机器学习方法用于分析缺陷的根本原因，并在开源系统 Eclipse 上进行了验证[25]。Jeffrey 等基于大多数内存错误涉及使用损坏的内存位置的现象，提出了一种自动分辨内存相关缺陷根本原因的方法[26]。Thung 等提出了一种机器学习和代码分析技术的组合，通过分析缺陷修复过程中代码的变更来识别缺陷的根本原因[27]。Dalal 等对一些历史上的严重软件故障以及一些仍在演化的软件项目中的故障进行了根本原因分析，此外还总结并比较了多种缺陷根本原因分析的方法[28]。

结束语 鉴于 R 语言在数据分析、人工智能等领域的广泛应用，本文对 R 语言及其核心包中的 7020 个历史缺陷进行了实证研究。通过分析缺陷的版本信息，发现 R 3.1.2，R 3.0.2，R 3.5.0 等版本中缺陷的数量最多，提示应关注变革较大的新版本；通过观察缺陷所在组件，发现 Documentation，Graphics，Language 等组件发生缺陷的概率高于其他组件，缺陷优先级整体最高和严重程度整体最高的组件分别是 Startup 和 I/O，提示应该加强对这几个组件的测试和维护工作；通过统计缺陷修复时长，发现部分缺陷修复时间过长的原因是缺陷确认时间过长以及缺陷报告撰写质量低下，这提示了提高缺陷修复效率应改进的两个方向；通过抽样分析缺陷产生的根本原因，发现语义错误最为常见，提示了开发过程中应对此类问题加以关注。这些发现揭示了 R 语言及其核心包中历史缺陷的一些基本规律，可在一定程度上帮助 R 语言开发人员提高开发质量，帮助 R 语言维护人员更高效地检测和修复缺陷，并帮助 R 语言的使用者规避潜在风险。

R 语言自身功能的正确性是各领域内 R 语言应用程序能够正确运行的必要前提。对于这类在特定领域内占有重要地位的程序设计语言，对其历史缺陷进行回顾性的实证研究，不仅有助于其本身质量的提高，也有助于开源基础软件质量的提高。在未来研究工作中，可面向 R 语言中常见的缺陷根本原因有针对性地开展缺陷自动检测、自动定位、自动修复等工作。

参 考 文 献

[1] TAN L，LIU C，LI Z M，et al. BugCharacteristics in Open Source Software [J]. Empirical Software Engineering，2014，19(6)：1665-1705.

[2] WAN Z Y，LO D，XIA X，et al. Bug Characteristics in Blockchain Systems：A Large-Scale Empirical Study [C]// Proceedings of the IEEE/ACM 14th International Conference on Mining Software Repositories(MSR 2017). 2017：413-424.

[3] RAZZAQ S，LI Y F，LIN C T，et al. A Study of the Extraction of Bug Judgment and Correction Times from Open Source Software Bug Logs [C]// Proceedings of the IEEE International Conference on Software Quality，Reliability and Security Companion(QRS-C 2018). 2018：229-234.

[4] BHATTACHARYA P，ULANOVA L，NEAMTIU I，et al. An

Empirical Analysis of Bug Reports and Bug Fixing in Open Source Android Apps [C]// Proceedings of the 17th European Conference on Software Maintenance and Reengineering. 2013: 133-143.

[5] SAHA R K, KHURSHID S, PERRY D E. An Empirical Study of Long Lived Bugs [C]// Proceedings of the Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Enginee-ring(CSMR-WCRE 2014). 2014: 144-153.

[6] YUE R R, MENG N, WANG Q X. A Characterization Study of Repeated Bug Fixes [C]// Proceedings of the IEEE International Conference on Software Maintenance and Evolution(ICSME 2017). 2017: 422-432.

[7] ZIMMERMANN T, NAGAPPAN N, GUO P J, et al. Characterizing and Predicting Which Bugs Get Reopened [C]// Proceedings of the 34th International Conference on Software Engineering(ICSE 2012). 2012: 1074-1083.

[8] SUN C N, DU J, CHEN N, et al. Mining Explicit Rules for Software Process Evaluation [C]// Proceedings of the International Conference on Software and System Process (ICSSP 2013). 2013: 118-125.

[9] CHEN N, HOI S C H, XIAO X K. Software Process Evaluation: A Machine Learning Approach [C]// Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering(ASE 2011). 2011: 333-342.

[10] SUN C N, LE V, ZHANG Q R, et al. Toward Understanding Compiler Bugs in GCC and LLVM [C]// Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016). 2016: 294-305.

[11] SAHOO S K, CRISWELL J, ADVE V. An Empirical Study of Reported Bugs in Server Software with Implications for Automated Bug Diagnosis [C]// Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering(ICSE 2010). 2010: 1-10.

[12] LE V, SUN C N, SU Z D. Finding Deep Compiler Bugs via Guided Stochastic Program Mutation [C]// Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications(OOPSLA 2015). 2015: 386-399.

[13] ZAMAN S, ADAMS B, E. HASSAN A. Security Versus Performance Bugs: A Case Study on Firefox [C]// Proceedings of the 8th Working Conference on Mining Software Repositories (MSR 2011). 2011: 93-102.

[14] VUIJAYAKUMAR K, BHUVANESWARI V. How Much Effort Needed to Fix the Bug? A Data Mining Approach for Effort Estimation and Analysing of Bug Report Attributes in Firefox [C]// Proceedings of the International Conference on Intelligent Computing Applications. 2014: 335-339.

[15] LI F, PAXSON V. A Large-Scale Empirical Study of Security Patches [C]// Proceedings of the ACM Conference on Computer and Communications Security(CCS 2017). 2017: 2201-2215.

[16] HANAM Q, BRITO F S D M, MESBAH A. Discovering Bug Patterns in JavaScript [C]// Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering(FSE 2016). 2016: 144-156.

[17] NGUYEN T, VU P M, NGUYE T T. An Empirical Study of Exception Handling Bugs and Fixes [C]// Proceedings of the ACM Southeast Conference(ACMSE 2019). 2019: 257-260.

[18] SUN X B, ZHOU T C, LI G J, et al. An Empirical Study on Real Bugs for Machine Learning Programs [C]// Proceedings of the 24th Asia-Pacific Software Engineering Conference (APSEC 2017). 2017: 348-357.

[19] ZHANG Y H, CHEN Y F, CHEUNG S C, et al. An Empirical Study on TensorFlow Program Bugs [C]// Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis(ISSTA 2018). 2018: 129-140.

[20] ZHANG R, XIAO W C, ZHANG H Y, et al. An Empirical Study on Program Failures of Deep Learning Jobs [C]// Proceedings of the 42nd International Conference on Software Engineering(ICSE 2020). 2020: 1159-1170.

[21] ISLAM M J, NGUYEN G, PAN R, et al. A Comprehensive Study on Deep Learning Bug Characteristics [C]// Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering(ESEC /FSE 2019). 2019: 510-520.

[22] DU X T, XIAO G P, SUI Y L. Fault Triggers in the TensorFlow Framework: An Experience Report [C]// Proceedings ofthe IEEE 31st International Symposium on Software Reliability Engineering(ISSRE 2020). 2020: 1-12.

[23] GRISHMA B R, ANJALI C. Software Root Cause Prediction using Clustering Techniques: A Review [C]// Proceedings of Global Conference on Communication Technologies (GCCT 2015). 2015: 511-515.

[24] HIRSCH T, HOFER B. Root Cause Prediction Based on Bug Reports [C]// Proceedings of the 31st IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW 2020). 2020: 171-176.

[25] LAL H, HOFER B, PAHWA G. Root Cause Analysis of Software Bugs using Machine Learning Techniques [C]// Proceedings of 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence. 2017: 105-111.

[26] JEFFREY D, GUPTA N, GUPTA R. Identifying the Root Causes of Memory Bugs Using Corrupted Memory Location Suppression [C]// Proceedings of the IEEE International Conference on Software Maintenance(ICSM 2008). 2008: 356-369.

[27] THUNG F, LO D, JIANG L X. Automatic Recovery of Root Causes from Bug-Fixing Changes [C]// Proceedings of 20th Working Conference on Reverse Engineering (WCRE 2013). 2013: 92-101.

[28] DALAL S, CHHILLAR R S. Empirical Study of Root Cause Analysis of Software Failure [J]. ACM SIGSOFT Software Engineering Notes, 2013, 38(4): 1-7.

WANG Zi-yuan, born in 1982, Ph.D, associate professor, master supervisor, is a member of China Computer Federation. His main research interests include software testing and so on.

(责任编辑:喻藜)