基于演化历史的软件故障与日志关联关系分析与挖掘

池书琪[1] **何浩辰**[1]
(国防科学技术大学计算机学院，湖南 长沙 410073)[1]

摘要日志是软件故障诊断的重要手段。目前，利用日志进行软件故障诊断的工作主要可以分为两类：基于规则的方法和基于执行路径的方法。基于规则的方法主要通过挖掘和总结日志中的规则，这种规则一般是日志中的出错信息序列和软件组件之间的对应规则，或者日志信息和软件出错模式的对应规则，从而帮助开发者进行基于日志的软件故障诊断。基于执行路径的方法主要通过静态分析的方法来构建软件故障时的运行路径，从而帮助开发理解和诊断故障。但是，在软件规模较大时，这种方法容易产生路径爆炸问题。为了解决上述问题，降低路径爆炸的可能性，文章通过探索软件的演化历史，分析软件的演化历史中 patch 与故障日志的关系，得到如下两个结论：1）80%以上相似的故障日志所对应的 patch 里会操作相同的变量或调用相同的函数，2）70%以上的 patch 和第一条故障日志的距离不超过 2。通过这两个发现，可以帮助重建故障时路径的工作对路径进行筛选，从而避免路径爆炸。

关键词故障日志，故障修复，故障诊断

中图法分类号 TP311.5 **文献标识码** A

Analyzing and Mining the Relationships between Logs and Patches in Software Evolution History

CHI Shu-qi[1] HE Hao-chen[1]
(College of Computer, National University of Defense Technology, Changsha 410073, China)[1]

AbstractLogs plays an important part in bug diagnosis. Bug diagnosis based on logs either use rule-based methods or execution-path-based methods. Rule-based methods usually concentrate on mining rules, which are usually relationships between log sequences and software components, or patterns of errors in software, in logs, in order to facilitate bug diagnosis. Execution-path-based methods focus on reconstructing execution paths to help developers understand bugs with a static analysis method. However, execution-path-based methods are not able to infer every choice at branch instructions in failed execution, and this could result in path explosion. To remedy this situation, we explore the software evolution history, and analyze the relationship between logs and patches. We get two conclusions: 1) 80% of the similar logs have similar corresponding patches, 2)more than 70% patches has a distance of no more than 2 to the first error message. With our findings，we could help these execution-path-based method to avoid path explosion by pruning irrelevant paths.

KeywordsLog，Patch，Bug diagnosis

1. 引言

系统日志是故障诊断的重要手段，它可以记录程序运行时的动态信息，帮助维护人员分析检查、进而修复系统故障，提高系统运行的可靠性。在软件故障诊断的工作中，系统日志被广泛使用。

目前，基于日志的工作主要可以分为两类：基于规则的工作和基于执行路径的工作。基于规则的工作，主要是通过从故障日志中挖掘规则[1][2]来帮助开发者进行故障诊断。Logsurfer[1]利用人的领域知识，总结日志中的出错模式，从而帮助开发者诊断软件故障。Distalyzer[2]使用机器学习的方法，挖掘日志和系统组件的关联规则。并且使用这种规则来检测故障日志，从而帮助软件故障诊断。基于路径的方法[3][4]，主要通过重建软件运行时路径，来帮助软件维护人员理解和诊断软件故障。SherLog[3]利用静态分析技术，利用故障日志重新构建故障时的软件运行路径，从而帮助故障诊断。但是在软件规模较大时，这种方法容易产生路径爆炸问题。针对目前的相关

到稿日期：返修日期：
池书琪（1994-），男，硕士生，主要研究方向为操作系统和软件可靠性，E-mail: chishuqi16@nudt.edu.cn

工作现状，本文探索软件的演化历史，希望找出patch 与故障日志的潜在关联关系，从而进一步推动和指导软件故障修复工作。

本文通过人工调研，发现故障日志和patch 之间确实有关联关系。如图所示，在软件MariaDB 的故障管理系统中，故障 ID 分别为“MDEV-13591”和“MDEV-8195”的故障，他们的故障日志中有几条错误信息完全一致，而且他们 patch 中修改的地方都涉及同一个类型的变量“crypt_data”。这个例子表明：两个软件故障的日志相似，那么他们可能在故障修复上也具有一定的相似性。为了进一步验证这种故障日志和patch 之间的关联关系的普遍性，本文从 7 个大型开源软件的故障管理系统中收集了他们的故障报告和 patch，并对他们进行进一步的研究。这 7 个软件分别是：MariaDB, Squid, CUBRID, Httpd, OpenStack, OpenSSH, 和 Nginx。

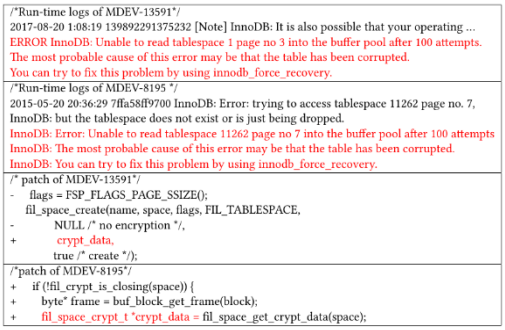

图 1 日志相似的 patch 相似的样例

Fig 1. A representative example of bugs (MDEV-13591 and MDEV-8195) and patches

在分析和挖掘软件故障日志和 patch 之间的关联关系的过程中主要有以下挑战：（1）故障日志的相似性判断。一般情况下，故障日志中的报错信息都混杂在普通信息（非报错信息，一般用于输出软件正常运行时的相关信息）中，普通信息阻碍了故障日志的相似性判断。本文研究了几个开源软件的软件输出机制，发现了这些开源软件在输出故障信息的函数的设置上存在一定模式，可以通过利用这个模式来区分日志中的普通信息和报错信息。（2）patch 的相似性判断。这个问题可以分两个方面，一方面，如何恰当的选择 patch 中特征，用于代表整个 patch 去判断其相似性；另一方面，patch 一般是一段仅包含少量上下文的代码段，并不包含完整的语义信息，这使得从中选取特征的工作更加困难。例如，仅凭图一中软件故障的 patch 片段，很难知道“crypt_data”这个变量的具体类型。本文通过人工研究了若干对故障信息相似的 patch，从 patch 相似修改中选取特征，用于相似 patch 的计算。另外，本文还通过结合结合源码的上下文分析来补全 patch 中缺失的上下文。

本文主要有以下的贡献：分析并挖掘了 7 款开源软件的故障日志和 patch 的关联关系，我们得到了两个结论：第一，在软件故障日志相似的情况下，80%以上的 patch 都是相似的。这个发现说明了相似故障的 patch 有很大程度上是有关系的。第二，故障日志和 patch 的位置一般比较接近。这对软件故障诊断工作提出了指导。本文还对现有的软件故障提交现状进行了分析，并提出了建议。

本文组织结构如下：第二章介绍了在MariaDB 软件中的针对故障日志和 patch 之间关系的调研工作；第三章介绍如何挖掘故障日志和patch 间的关联关系；第四章在 7 款开源软件中评估了故障日志和 patch 间的关联关系的情况；第五章对全文进行了总结。

2. 故障日志和 patch 关系调研

表 1 日志和 patch 关系分析

Table 1. Analysis on the relationship between logs and patches

软件	相同变量	相同函数	不相同	总计
MariaDB	17（85%）	0（0%）	3（15%）	20
Squid	28（61%）	12（26%）	6（13%）	46
CUBRID	21（67%）	6（19%）	4（14%）	31

本章的主要目的是调研历史故障日志和patch中是否存在关系。本文从3个开源软件MariaDB、Squid、CUBRID入手，人工分析了97对具有相似故障信息（除了时间戳和变量值外完全相同，如图1中两个日志中红色部分信息）的故障日志及其patch之间的关系。

如表1所示，在调研到的97对有相似故障信息的patch中，大部分patch之间都是有相似的修改的。在这些patch中的相似修复中，大部分的相似修复都是修改的代码涉及到对相同变量的操作，少部分涉及到相同函数的调用。为了验证和进一步研究这种关系存在的普遍性，本文在后面的章节中，在多个开源软件中故障日志和patch中存在的这种关联关系进行挖掘并验证。

3. 故障日志和patch的关联关系挖掘

3.1 关联关系挖掘工作流程

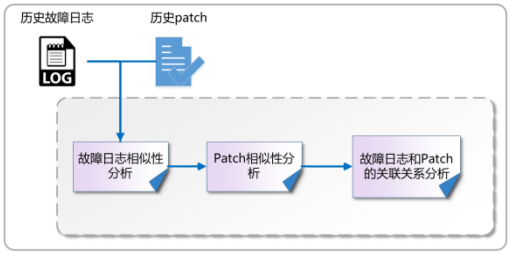

图2 关联关系挖掘流程图

Fig 2. Structure of mining relationship

如图2所示，首先在历史故障日志中分析故障日志的相似性，挑选出相似的故障日志对。然后，进行patch的相似性分析，判断相似的故障日志对应的patch是否相似。最后，如果故障日志对应的patch相似，那么提取这种相似关系，构建关联关系库。

3.2 软件故障日志的相似性分析

本节的主要目的是计算两个软件故障的日志是否相似。软件故障日志[5][3]作为非常重要的，它是唯一能在事后提供程序故障时的客观信息的材料。同时，故障时日志也是唯一可以客观反映软件故障特点的材料[6]。首先，本节这里的两个日志是否相似做出说明。相关工作中[7][8]，对于两个日志相似的一些分析如下：（1）如果两个日志相似，则它们中的报错信息需要相似。（2）一个日志中的信息与另一个日志中的信息完全相同，几乎不存在。从这两点出发，在本文中给出软件故障日志相似的定义：给定两个日志A和B，如果除了变量的值，时间戳等在每次运行得到的结果不同的值之外，剩余的报错信息的字符串完全相同，则认为日志A和日志B是相似的。

上面给出了本文对相似日志的定义，但是，如何判断两个软件故障的日志是否相似，首先需要解决两个问题。第一，如何从日志中的大量的普通信息中筛选出报错信息，因为几乎每个日志都含有普通信息，所以根据普通信息根本无法辨别两个日志是否相似。第二，如何判断筛选出来的故障信息的相似性，因为日志中的每条信息中，不但包含了不变的字符串值，也包含了变量的值、时间戳等在每次执行都不同的值。

软件故障日志中报错信息筛选。通过人工研究了几个开源软件中的日志输出机制，从而发现了一个现象：所研究的几个软件中，报错信息和普通信息一般使用不同的冗余等级，或者输出报错信息的函数名一般包含类似于“error”，“fail”之类的词语。总之，输出报错信息的函数一般和输出普通信息的函数在软件中是有区别的。本文利用这种区别，从而将日志中的报错信息和普通信息区分开来。具体的：

（1）首先按照字符串匹配的方法，将每条日志匹配到源码中它的输出函数（可能会一条信息匹配到多个函数）。

（2）根据本文的发现，从WordNet[9]上得到“error”，“fail”的所有近义词，利用这些词对步骤1得到的函数进行筛选：如果函数名或者其参数名包含了这些词，那么这个函数对应的信息就是报错信息（如果多个函数对应一条信息，那么只要有一个函数符合输出报错信息函数的特征，这条信息就是报错信息）。

两个日志中的故障信息的相似性判断。根

据本文对于相似日志的定义：除了时间戳、变量的值之外所有字符串都相同。考虑到对于报错信息而言，他们中存在着每次出错都会出现的信息和特定错误出现才会出现的信息。本文需要计算两个日志是否相似，很显然，在计算日志相似的时候对于不同报错信息所占的权重应该有所区分。本文使用信息检索中常用加权技术计算中常用的方法：TF-IDF（Term frequency－inverse document frequency）[10]，这种方法可以筛选出报错信息中的针对特定故障的信息。具体的：

（1）本节首先对得到的日志中的报错信息进行文本处理，去掉文本中的停顿词[11]、将驼峰法和使用连接符连接的词拆开，并利用 port stemming 算法将拆出来的词还原到其词根形式。

（2）然后，对处理出来的词进行排序，并将文本组成按照其出现次数组成一个向量的形式。

（3）计算每两个向量之间的余弦相似度[12]，如果大于 0.5，那么本文认为这两个向量对应的日志是相似的。（0.5 是通过人工筛选了 20 对相似的日志，通过计算其余弦相似度并取平均值得到）

3.3 软件 patch 的相似性分析

上一节介绍了如何计算故障日志的相似性，本章的主要目的是给定两个 patch，如何判断两个 patch 是否相似。要判断两个 patch 是否相似，主要有两个问题需要解决。第一，如何选定 patch 的特征，从而用于 patch 相似的判断；第二，因为 patch 一般是只包含少量上下文的代码段，不包含完整的语义信息，如何从中提取相应的语义信息，比较变量类型等。

Patch 的特征选取。本文通过调研 97 对日志相似的 patch，发现两个 patch 的相似一般表现在修改中涉及到相同的函数或者变量。从而，本文选择 patch 的特征为：被修改的语句中涉及到的变量及其类型，以及其中涉及到的调用的函数。相应的，本文给出对于两个 patch 相似的定义，给定两个 patch A 和 B：（1）如果两个 patch 中的修改的代码涉及相同变量，那么这两个 patch 相似；（2）如果两个 patch 中的修改的代码中调用了相同函数，那么这两个 patch 相似；（3）如果两个 patch 中的修改的代码既涉及相同变量，又调用了相同的函数，那么这恋歌 patch 相似。否则，两个 patch 不相似。

Patch 中的语义补全。本文使用结合源码的模糊匹配方法来补充 patch 中缺失的语义信息。具体的，首先，本文使用了一种叫做模糊匹配（fuzzy parse）的方法，具体的，利用模糊匹配的工具 srcML[13]先生成 patch 的抽象语法树，并标记出缺失类型信息的变量。其次，利用 LLVM[14]的 clang[15]工具，获得源码中对应文件的抽象语法树。最后，利用名称对应的方法，从源码的抽象语法树中提取相应的语义信息从而补充 patch 的抽象语法树。

解决了上述两个问题之后，只需要从 patch 的抽象语法树中提取出被修改了的语句涉及到的函数和变量以及类型，并将其作为特征。根据本文对于 patch 相似的定义，一旦两个 patch 中具有相同特征，则认为这两个 patch 相似。

3.4 故障日志和 patch 的关联关系分析

本节的目的主要是分析故障日志和故障修复之间的关联关系。本节通过分析计算这种关联关系存在的普遍性，希望用数据给出一个对这种关联关系的认识，并提取这种规则构建关联关系库。

我们使用下面的公式来计算这中普遍性：

$$P = \frac{\text{故障日志相似且patch也相似的故障对}}{\text{故障日志相似的故障对}}$$

我们使用 P 来衡量故障日志和 patch 之间关联关系存在的普遍性，通过 P 来反故障日志和 patch 之间关联关系的存在情况，从而给广大开发者提供一个关于故障日志和 patch 之间关系的认识，并且充分利用这种关系，从而在类似构建软件故障路径这样的工作中，避免路径爆炸的问题。

本文中的故障日志和故障修复之间的关联关系表现为：故障日志中出现了特定的故障信息，

那么在修复这个故障时，就需要修改与这个故障信息有关联关系的变量或者函数调用所在的代码。为了提取这种关联关系，我们首先寻找了相似的故障日志，分析其 patch 的相似性。如果它们的 patch 相似，那么我们提取两者的故障信息，以及两者 patch 的相同部分的变量或者函数调用，构成关联规则。

具体地，如果故障日志 A 和故障日志 B 相似，且它们的 patch 的相似部分为 C。那么从这两个故障日志构建的关联规则为：<<A 的故障信息，B 的故障信息>,C>

4. 实验评估

本章首先介绍用于评估软件故障日志和 patch 之间相关关系的数据集（4.1）；然后，文章分析了本文中使用到的数据集的情况，并进行了分析，给出了几点建议（4.2）；接着，文章计算数据集中的相似日志对应的 patch 的相似情况，并做出说明和分析（4.3）；之后，我们挖掘关联关系，并对关联关系进行验证（4.4）；最后，文章分析 patch 在日志输出序列中的分布情况（4.5）。

4.1 实验设置

表 2 用到的软件

Table 2. Subject Software

软件	Patch	故障	日志	版本
squid	1803	1803	600	176
Mariadb	4450	4450	1466	140
CUBRID	600	600	420	77
Httpd	2540	2600	38	63
Openstack	78000	80003	103	18
OpenSSH	540	2586	161	78
Nginx	200	1028	367	80

本文的实验是在有 8 个 Intel Xeon 2.33GHz CPU 和 8GB 内存的 Linux 机器上进行的。本文的数据集主要来源于以下 7 个软件：MariaDB, Squid, CUBRID, Httpd, OpenStack, OpenSSH, 和 Nginx。这 7 款开源软件都具有在其领域具有一定的代表性，使用范围广，并且经过了多年的开发和维护，积累了大量的故障和 patch。

本文利用一个开源的 java 爬虫框架，从这些软件的故障管理系统（Bugzilla, Trac 和 JIRA）爬取了大量的软件故障以及 patch（详见表 2），其中软件表示对应的软件名称，“patch”表示从这个软件的故障管理系统中得到的 patch 的数量，“故障”表示从这个软件的故障管理系统中得到的软件故障的数量，“版本”表示涉及到的软件的版本数，“日志”表示从软件的故障管理系统中得到的故障日志的数量。

4.2 数据集分析

从表 2 中可以看出，从其中一半以上的软件中得到的故障日志数量不超过 200。因为故障日志作为非常重要的软件故障的信息，这对于开发者而言，在理解和诊断软件故障时提高了难度。并且，对于评估相似故障日志以及 patch 之间的相关关系而言，是非常不利的。

为了了解为什么很多软件的故障日历没有被用户提交到相应的故障管理系统，本文详细对比了故障日志提交多的软件和故障日志提交少的软件。调查结果表明，很多情况下，故障提交网站并没有详细对需要提交的内容做出详细的介绍和说明。

如表 3 所示，这是 mariaDB 和 Nginx 两个软件的故障提交要求，从中明显可以看出，MariaDB 对于用于需要提交的内容做出了明确的要求，甚至细致到了需要提交的内容所在的位置。反观 Nginx，并没有做出明确的要求，只是很简单做了要求。从这两者的对比可以看出，在设计故障提交表格的时候，不能简单的将用户当作很有经验的人员，需要对需要提供的东西做出明确的要求。

为此，本文提出几点建议：

表 3　软件故障提交要求

Table 3. Instructions for Bug Submission

MariaDB	a. The environment (Operating system, hardware and MariaDB version) where the bug happened. b. Any related errors or warnings from the server error log file. Normally it is hostname.err file in your database directory c. The content of your my.cnf file or alternatively the output from mysqld --print-defaults or SHOW VARIABLES. d. Any background information you can provide (stack trace, tables, table definitions, data dumps, query logs). e. If the bug is about server producing wrong query results… f. If the bug about a performance problem, e.g. a certain query is slower on one version than on another, output of EXPLAIN EXTENDED <query> on both servers. g. A test case or some other way to repeat the bug. This should preferably be in plain SQL or in mysqltest format.
Nginx	Please describe the bug in full detail including all of the conditions that make it happen and include the relevant part of configuration as well. It would substantially reduce time and effort to understand and resolve the bug.

（1）在故障报告提交说明中，尽可能详细的要求用户提交修复故障可能需要的文件，要求用户提交复现的过程和相关文件

（2）不要将用户当作很有经验的开发人员，要尽可能细化需要提交的内容都在什么位置。

（3）故障提交的要求要放在非常显眼的位置，甚至可以要求用户必须先阅读才可以提交故障报告。

4.3　故障日志和 patch 的相关关系评估

本节主要对故障日志和 patch 的相关关系进行评估，由于本文中的数据集中有几个软件的故障日志太少，从中选择 3 个故障日志较多的软件来进行评估。具体的，对于每个软件，先计算日志的相似情况，然后计算相似日志对应的 patch 的相似情况。

表 4　相似日志对应的 patch 相似情况

Table 4. Pervasiveness of Similar modifications among similar logs

软件	变量相同	函数相同	完全不同	总计
Squid	39 （69.6%）	9 （16.1%）	8 （14.3%）	56
MariaDB	64 （56.1%）	37 （32.5%）	13 （11.4%）	114
CUBRID	49 （68.1%）	9 （12.5%）	14 （19.4%）	72

从表 4 的结果可以看出，根据本文对于相似 patch 的定义，在这 3 个软件中，超过 80%的相似的故障日志对应的 patch 是相似的，也就是 P 的值在每个软件中平均大于 80%。这个结果可以说明，在故障日志相似的情况下，两个故障需要修复的地方很有可能具有一定的相似性，这个结果为开发者在修复软件故障的时候提供了一些启示，可以一定程度上增进开发者对于软件故障的理解，可以帮助开发者在重建软件故障路径时，侧重于可能需要修复的位置，从而一定程度上减少和避免路径爆炸问题。

4.4 关联规则挖掘和验证

本节的主要目的是挖掘故障日志和 patch 的关联规则，并对其准确性进行验证。为了验证挖掘到的关联规则的准确性，我们将数据集按照其版本的先后顺序等分成两个部分，前面的版本组成训练集，后面的版本组成测试集。具体的验证方法为：首先在训练集中挖掘关联规则，然后在测试集中寻找与关联规则中所有故障日志都相似的故障日志，如果这些故障日志所对应的 patch 也满足规则（patch 中涉及规则中的变量和函数调用），那么这个规则是准确的，否则为不准确。如果测试集中找不到与某规则中故障日志都相似的故障日志，那么这个规则的准确性无

法验证。

表 5　规则的准确性验证

Table 5. Correctness of mined relationship

软件	规则数	准确	不准确	无法验证
Squid	36	16（44.4%）	2（5.5%）	18（50.1%）
MariaDB	73	39（53.4%）	5（6.8%）	29（39.8%）
CUBRID	30	10（33.3%）	2（6.7%）	18（60%）

我们挖掘 3 个软件 MariaDB、Squid、CUBRID 中的规则，并对其进行验证。如表 5 所示，有很大一部分在前面版本挖掘到的规则，在后面的版本中没有出现，有两种可能：由于前面的版本修复了，后面的版本没有再出现相似故障；或者故障还未被触发。

对于可以验证的部分，80%以上的规则都是准确的，我们分析了规则不准确的原因：大部分导致这种不准确的原因都是前面的修复未彻底修复完成。后面的 patch 事实上都修复了我们的规则对应的变量或者函数相关的上下文，或者相关的脚本文件（非源码文件）。事实上，我们的规则在帮助开发者进行故障定位时，也有一定的提示作用。

4.5　patch 在故障日志输出序列中的分布

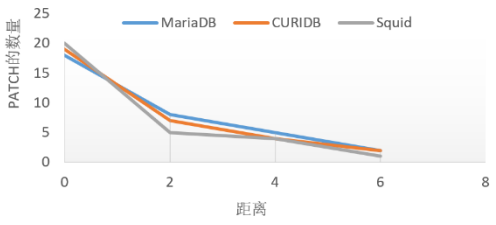

图 3 patch 于第一条报错信息的距离分布

Fig 3. The Distance between patch and first error message

本节首先定义如下的代码距离（distance），在程序的执行时，两段代码的距离为从一段代码跳转到另外一段代码所经过的最少的函数调用的次数。

为了衡量 patch 在软件故障日志中的分布，本文以故障日志中第一条报错信息所在代码为基准，并计算 patch 所在的代码和故障日志被输出位置所在的代码之间的距离。由于每个 patch 可能分布的位置比较分散，选择最小的距离作为 patch 和第一条报错信息的距离。在 3 个开源软件中，人工选择了 95 个 patch 以及对应的故障日志，来进行评估。

如图 3 所示，平均对每个软件来说，70%的 patch 和第一条故障信息的距离不超过 2。这个发现说明了，一般需要修复的位置可能距离第一条故障日志非常近，这个发现可以大大的加快开发者修复软件故障的速度，从而减少很多用于检查代码的时间，从而节省了工业界的软件维护成本。

5.　结束语

本文针对软件故障日志和 patch 之间是否存在关联关系这一问题。通过对 7 款开源软件的故障管理系统中的故障和 patch 的信息收集，对相似的故障日志对应的 patch 的相似性进行分析和计算，发现 80%以上的相似的软件故障日志对应的 patch 是相似的。同时，本文还发现 70%以上的 patch 和日志中第一条报错信息的距离小于 2. 同时，本文还分析了软件故障管理系统可能存在的问题，并提出了建议。

参考文献

[1]Prewett J E. Analyzing cluster log files using Logsurfer[J]. Proc.annual Conf.on Linux Clusters, 2003.

[2] Nagaraj K, Killian C, Neville J. Structured comparative analysis of systems logs to diagnose performance problems[C]// Usenix Conference on Networked Systems Design and Implementation. USENIX Association, 2012:26-26.

[3] Yuan D, Mai H, Xiong W, et al. SherLog: error diagnosis by connecting clues from run-time logs[C]// Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems. ACM, 2010:143-154.

[4] Zhang Y, Makarov S, Ren X, et al. Pensieve: Non-Intrusive Failure Reproduction for Distributed Systems using the Event Chaining Approach[C]// The, Symposium. 2017:19-33.

[5]Yuan D, Zheng J, Park S, et al. Improving software diagnosability via log enhancement[J]. Acm Transactions on Computer Systems, 2011, 39(1):3-14.

[6] Yuan D, Park S, Zhou Y. Characterizing logging practices in open-source software[C]// International Conference on Software Engineering. IEEE, 2012:102-112.

[7] Foyzul Hassan, Xiaoyin Wang. HireBuild: An Automatic Approach to History-Driven Repair of Build Scripts[C]// Ieee/acm International Conference on Software Engineering. IEEE, 2018:1078-1090

[8] Yuan D, Park S, Huang P, et al. Be conservative: enhancing failure diagnosis with proactive logging[C]// Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2012:293-306.

[9] Fellbaum C, Miller G. WordNet:An Electronic Lexical Database[M]. MIT Press, 1998.

[10] Wu H C, Luk R W P, Wong K F, et al. Interpreting TF-IDF term weights as making relevance decisions[J]. Acm Transactions on Information Systems, 2008, 26(3):55-59.

[11] Wilbur W J, Sirotkin K. The automatic identification of stop words[J]. Journal of Information Science, 1992, 18(1):45-55.

[12] Nguyen H V, Bai L. Cosine similarity metric learning for face verification[C]// Asian Conference on Computer Vision. Springer-Verlag, 2010:709-720.

[13] Collard M L. Addressing source code using srcml[J]. 2005.

[14]LLVM[EB/OL]. [2018-07-20]. http://llvm.org/

[15]Clang[EB/OL]. [2018-07-20]. http://clang.llvm.org/