大数据时代的软件开发新技术

业界数据挖掘算法分析报告

南京大学计算机软件新技术国家重点实验室

State Key Laboratory for Novel Software Technology at Nanjing University

2016 年 12 月 11 日

目录

1 引言

当前软件开发活动造成了大数据化的软件资源，同时软件智能化开发也一直是软件工程追求的核心目标之一。近年来，在软件工程领域顶级会议 ICSE、FSE 等中出现了越来越多基于数据、知识驱动的软件开发智能化技术研究，并指出软件开发智能化技术能够显著提升软件开发者的工作效率与质量。

当前软件智能化开发成为热点的关键原因在于新时代带来的新环境：①开源软件提供了大量数据源；②机器学习和信息检索技术的发展提供了知识获取的核心技术支撑；③企业软件工程的广泛实践积累了大量的领域资源。但是从总体方法论和技术体系来看，目前的研究工作还是处于各自关注的技术点上，使用的数据均是针对问题本身设计和采集的，缺少大数据环境提供领域的智能化技术研究，也没有形成完善的技术体系和环境。

在此背景下，本报告试图调研业界互联网公司/软件公司的基于大数据分析的软件开发实践，为建立软件大数据环境，提高开发效率和质量，实现开发环境、方法的升级提供技术支持。报告主要调研问题包括：①业界使用数据分析驱动软件开发的具体应用场景有哪些？②结合应用场景所使用的数据分析方法有哪些？③各应用场景所采用和分析的数据有哪些？④完成这些任务所使用的数据分析工具有哪些？表 1-1 介绍了本报告调研的动机及问题分析。

表 1-1：研究问题

研究问题	研究动机及问题分解
数据挖掘**场景**	软件开发具有很长的生命周期，从前期的需求分析、软件构造设计；到中期的软件编码、测试；再到后期的软件更新与维护，其中一些环节如用户数据分析、缺陷分析、代码审查等通常包含大量具有隐藏价值的数据，研究数据分析在软件开发过程中的具体应用场景有助于提高软件开发过程的高效性与可靠性。 报告的研究内容包括但不限于： ① 数据分析与挖掘的具体应用场景有哪些？ ② 应用以后带来了什么效果？
数据挖掘**方法**	目前主流的数据分析与数据挖掘方法有：机器学习中的分类方法（线性回归、决策树、神经网络、支持向量机、贝叶斯）、聚类方法（K-Means、Apriori）、自然语言处理、概率统计模型等。 报告的研究内容包括但不限于： ① 结合具体应用场景的数据分析与挖掘方法有哪些？ ② 如何对应用场景的数据分析与挖掘方法进行评估？

	③ 有哪些指标？ ④ 指标的定义和具体的计算规则是什么？
数据挖掘**数据**	数据是所有研究与分析任务的源头。在真实的软件开发过程中，每个不同的应用场景都会面临不同的数据（文档、代码、日志等），不同的数据所需要的分析方法都是不同的，此外对数据的预处理与分析结果数据的后处理都是数据分析中至关重要的环节。 报告的研究内容包括但不限于： ① 结合应用场景的数据分析与挖掘方法需要哪些数据？ ② 有哪些不同的类型？ ③ 这些数据从哪里采集？
数据挖掘**工具**	好的数据分析工具能够帮助开发者高效地完成开发任务，对开发工具进行统计分析对软件开发效率的提升具有指导性作用。 报告的研究内容包括但不限于： ① 完成以上数据分析任务使用什么工具？ ② 这些工具是如何进行数据采集、数据分析与数据呈现的？

报告中通过信息检索筛选出业界公司数据驱动软件开发的相关文献（截止至2016 年 12 月 31 日前），对文献先后进行精读、细读，整理并汇总相关信息；接着按照应用场景对文献进行归类，分别回答表 1-1 中的主要问题，并探讨数据驱动软件开发的现状与未来发展。

通过研究，报告认为数据驱动的软件开发在未来的业界软件开发领域将是一个日趋重要的话题，在此基础上对软件设计构造、软件运维演化、软件项目管理这三个应用场景下，软件编码优化、软件缺陷预测、群智软件等细分领域的研究将是未来需要关注的热点；机器学习方法在软件工程领域的应用前景十分广泛；在未来的大数据环境下，开发人员与研究者会更加关注除传统代码类型以外的，诸如用户数据、开发过程等偏文本化与抽象化的数据。这也将对未来数据驱动软件开发的研究奠定了基础。

2 调研方法

本章介绍报告的主要文献调研方法。如图 2-1 所示，首先从文献、专利数据库中定位调研范围并进行搜索(文献收集)；接着，对收集到的文献进行基于内容的筛选(文献筛选)；最后，对筛选出的文献进行归类、分析、综合（文献归类总结）。以下分别介绍每一步骤。

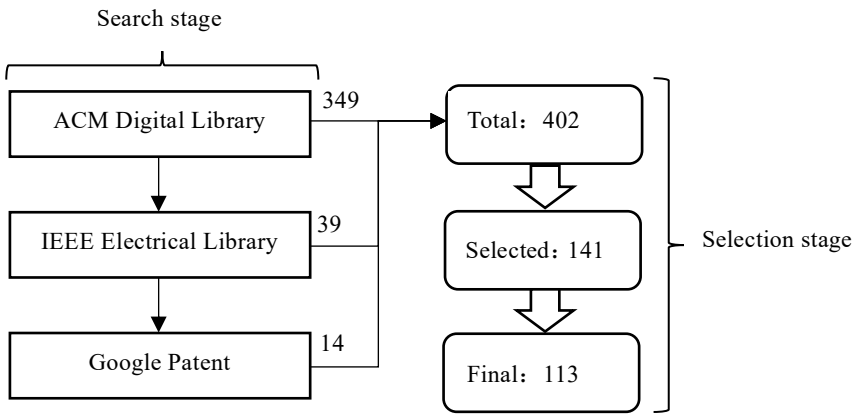


图 2-1：文献搜索与文献筛选流程

2.1 文献收集

表 2-1：文献收集范围

文献来源	ACM Digital Library, IEEE Electrical Library, Google Patent
发表公司	Amazon, Apple, Cisco, Google, IBM, Intel, Microsoft, Samsung, Sony
Primary CCS	Software Engineering
发表时间	2013.1-2016.12

文献收集阶段主要通过文献检索工具从文献数据库中搜索所有业界的与软件技术相关的文献。

表 2-1 列出了文献搜索范围，根据其中的关键词信息在对应数据库中进行搜索，得到 ACM Digital Library 文献 349 篇，IEEE Electrical Library 文献 39 篇，Google Patent 专利 14 项，共 402 篇相关文献。

2.2 文献筛选

文献筛选分为两步。第一步：对文献初步搜索得到的共 402 篇文献进行初选，通过文章标题、摘要等信息排除不符合调研范围的文献。第二步：对初步筛选后的文献进行精读，根据文献具体内容进一步排除不符合调研范围的文献。最终得到符

合调研范围的文献共 113 篇。

2.3 文献归类总结

在文献归类分析阶段，对筛选出的文献进行归类合并。

基本的归类策略是：在筛选出的文献中，根据文献标题、摘要与结论对文献进行整理，记录文献的发表会议/期刊、发表时间、公司、针对场景、采用数据、采用方法、工具等信息，从而探讨数据分析驱动软件开发的具体作用。

文献的分析策略是：按照应用场景将文献进行分类，分别研究具体应用场景下所使用的数据、方法和工具，并进行比较分析，从而针对性地探讨软件开发过程中各个阶段的重点数据与重点方法。

3 调研结果

3.1 文献概览

图 3-1 给出了近几年来数据驱动软件开发相关文献随时间变化的统计图。可以看到，截止 2015 年，有关文献的发表数目越来越多，这表明数据驱动的软件开发正在受到开发人员与研究人员日益广泛的关注。

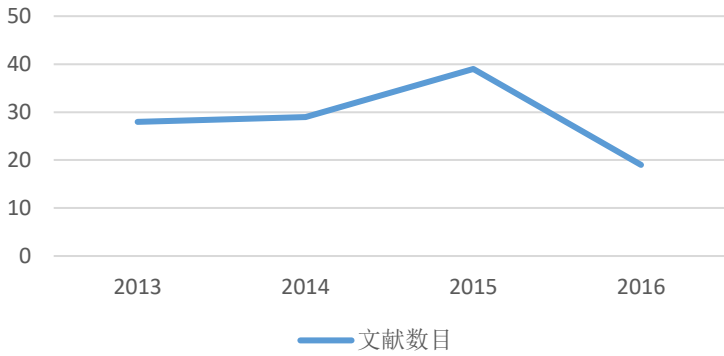


图 3-1：近几年发表文献数目汇总

图 3-2 给出了业界各公司发表相关文献数目的统计图。可以看到，Microsoft、IBM、Google 是发表数据驱动软件开发相关文献最多的三家业界公司，其中 Microsoft 发表的文献达到文献总数的一半。

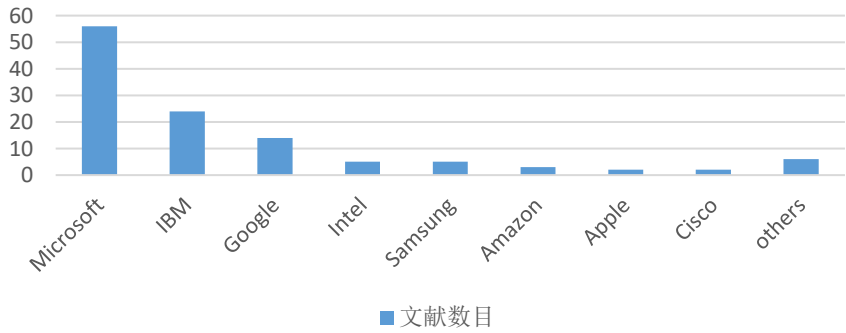


图 3-2 业界公司发表文献数目汇总

3.2 数据挖掘场景

从统计结果可知，业界数据驱动研发主要集中在 16 个具体应用场景，这 16 个应用场景可以进一步归纳为 3 个方面：

（1）软件设计构造

包括需求分析、软件编码技术、软件重构、文档自动生成等部分。通过数据分析与数据挖掘方法对需求、代码以及文档的自动生成，代码迁移与重构，以及软件开发过程中的代码审查问题。

（2）软件运维演化

包含运维数据的获取与管理、软件持续演化、缺陷预测、缺陷分析、缺陷修复、日志分析、问题报告分析、开发有过程分析、社区问答等部分。主要应用在发现软件漏洞和软件维护方面。

（3）软件项目管理

包括项目工作量分析、项目任务推荐、风险分析、智能协作开发与开发者画像。

表 3-1 给出了各个应用场景的主要内容及其归类。

表 3-1：应用场景分析与归类

应用场景	应用场景主要内容	应用场景归类
1.需求分析	发掘需求热点、用户反馈总结	软件设计构造
2.软件编码	代码搜索、代码自动补全、代码推荐、代码审查	
3.软件重构	项目重构、代码迁移	
4.文档自动生成	根据需求（规格化描述）自动生成文档	
5.运维数据管理	软件运行数据记录	软件运维演化
6.软件持续演化	软件维护、软件更新	
7.缺陷预测	缺陷分析、缺陷修复	
8.日志分析	开发日志分析、日志恢复	
9.问题报告分析	错误报告分析	
10.开发过程分析	开发人员交流分析	
11.社区问答	开发者、用户信息反馈	
12.工作量分析	项目 timeline 预估、项目人员分配	软件项目管理
13.智能协作	群智软件、众包、团队协作能力与稳定性	
14.任务推荐	缺陷修复人员推荐、任务分配推荐	
15.项目风险分析	错误代码的及时修改检测	
16.开发者画像	开发者能力预测、对关键开发人员的识别与评估	

图 3-3 给出了三个应用场景下的文献占文献总数的比重。从结果可知，软件运维演化是数据驱动研发的主要应用对象，数据驱动研发也广泛应用于软件设计构造。

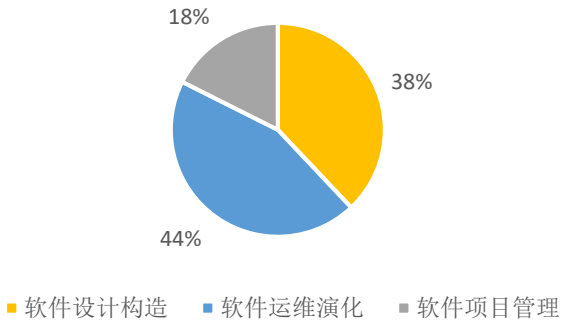


图 3-3 三个主要应用场景的文献数目

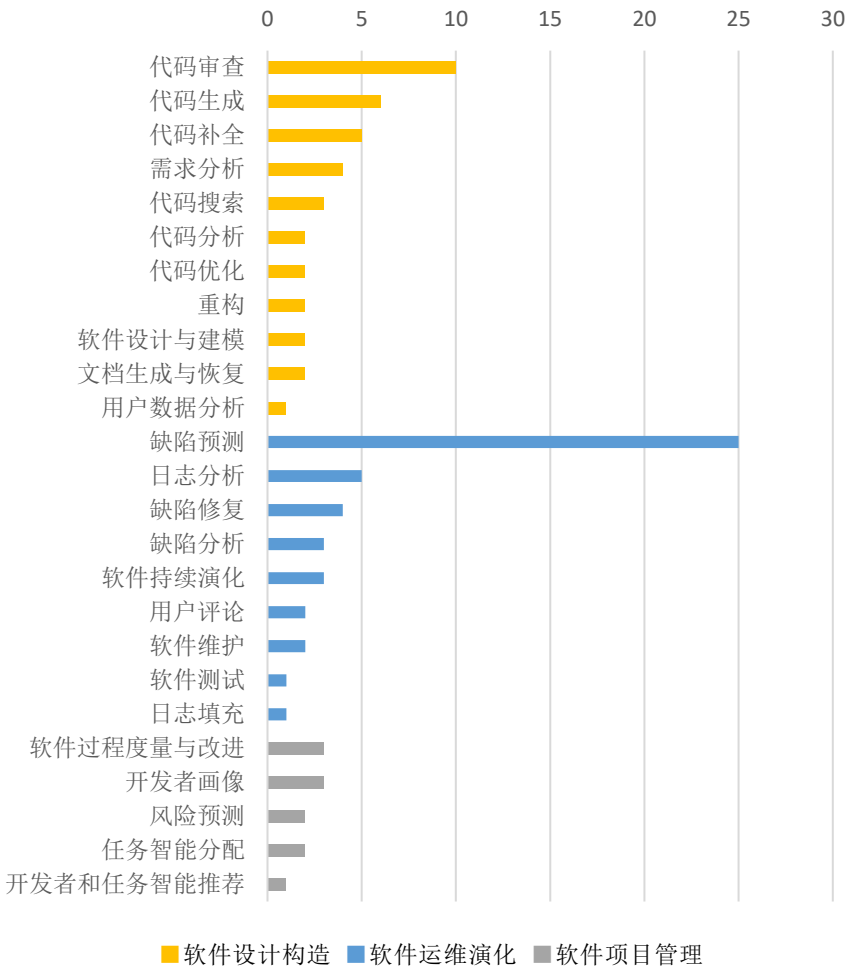


图 3-4: 各应用场景下的文献数目

图 3-4 给出了各个细分应用场景下的文献数目统计。在软件设计构造领域，软

件编码、需求分析是应用数据分析的主要环节；在软件运维演化领域，对缺陷的分析（缺陷预测、缺陷修复等）是该领域下非常重要的数据分析场景；在软件项目管理领域，数据分析主要在过程度量改进、开发者画像和任务推荐分配中发挥作用。

业界使用数据分析驱动的软件开发中，软件运维演化、软件设计构造是应用最频繁、最广泛的两个主要场景，其中运维演化领域主要使用数据分析与数据挖掘指导缺陷预测，设计构造领域主要使用数据分析与数据挖掘以提升编码质量。此外，需求分析、用户反馈分析、文档生成也是应用数据分析与数据挖掘技术的主要应用场景之一。

在这些应用场景下，数据分析与数据挖掘技术能够从以下方面对软件开发进行指导：

（1）提升编码质量，提高软件产品的性能效率、可靠性、可用性、可维护性和可移植性；

（2）智能分析软件缺陷，帮助缺陷修复，提升软件安全性；

（3）辅助分析用户需求与用户反馈，提升用户满意度；

（4）提升开发团队协作能力与稳定性，加快软件产品开发周期，提升开发效率；

（5）评估潜在风险与开发人员能力，协助、开发团队在软件开发中进行关键性决策。

4　软件设计构造

在软件设计构造这一应用场景下，需求分析、代码重构、代码补全、代码和文档生成以及代码审查是文献占比较大的几个方面，下面将重点分析其中的主要文献。

4.1　场景一　需求分析

多 agent 系统的商业化运用目前尚不成熟，主要问题在于难以将多 agent 系统投入到具体工业应用中。为此，微软提出一种从商业合同中提取需求的方法[19]。该方法可以从商业合同提取 6 种规则化关系：commitments (practical and dialectical), authorizations, powers, prohibitions, and sanctions，以帮助项目构建与建模，从而降低将多 agent 系统应用到工业界的障碍。

现有的从非结构化文本中提取信息的方法可以分为两种：基于 pattern 的方法（较简单）和基于机器学习的方法（较难，但是灵活性和鲁棒性更高）。该方法使用 NLP（特征提取）和机器学习方法（SVM）来提取上述关系。

图 4-1 描述了该方法的基本步骤。

（1）对合同进行预处理，去除 HTML 标签等；

（2）从预处理过后的合同中过滤出适合用于规格化的句子；

（3）对每个句子生成一个语法树；

（4）从语法树中进行特征提取，并训练了一个分类器用于筛选能真正代表规则化关系的候选项；

（5）用启发式函数提取规则化元素——主体、客体、前件、后件。

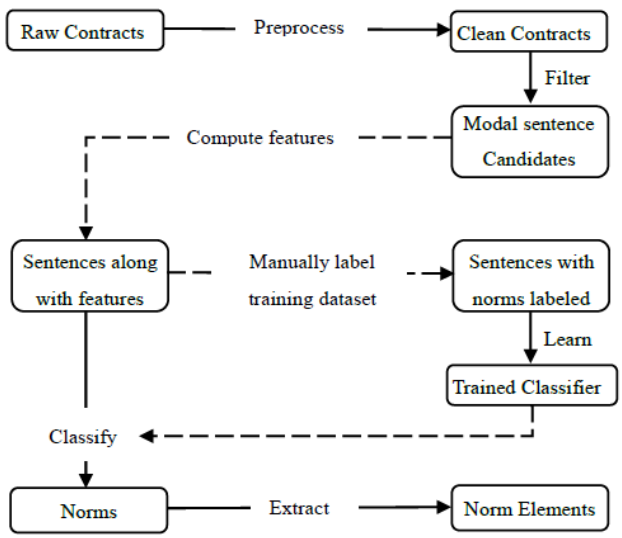


图 4-1 方法流程图

文章使用了 15 个特征，包括主语包含组织（Google）、条款信号（if）、情态动词（如 may, should）等，用作分类器的特征。接下来对其进行分类，考虑到 token 之间的独立性，可以用朴素贝叶斯分类器，此外，支持向量机、对率回归也是可用的分类方法，文章使用数据挖掘工具 Weka 对上述分类方法各自进行了实验。

为了验证该方法的性能，文章从真实商业合同中随机提取了 868 个句子，进行 10 折交叉验证，得到了平均 86%的查准率和 83%的查全率。

4.2 场景二 代码补全

Microsoft 共发表 3 篇代码补全相关的文献，分别运用贝叶斯[16]、神经概率语言模型[36]、自然语言处理[37]。Intel 发表 1 篇相关文献，使用概率调用自动机方法[23]。可以发现基于概率统计的方法在处理代码语言时仍是较为常见的方法。其中文献[16]对代码风格进行统一，文献[36]针对方法名类名推荐研究相应方法，分别获得 94%的精度与 state-of-the-art 效果。此外文献[37]对常见的代码语言模型进行分析研究，发现①面向开发者和具体应用的语言模型比面向整个代码库的语言模型性能更好；②时间性对语言模型的性能有很小的（几乎没有）影响。 而 Intel 提出使用概率调用自动机（Probabilistic Calling Automata, PCA）[23]分析函数调用之间的内在关系以及函数执行路径的性质，预测函数调用序列，比其他方法（CCT、TDAG、Pattern）提升了 30%-56%的整体序列精度。

编码习惯（coding conventions）能够反映代码的句法风格，不同的代码编写人员具有不同的编码习惯，不同的命名与格式决定了代码的可读性也不同。而在同一个项目中，由不同编码人员编写的代码带来的风格差异可能会导致项目代码难以阅读与维护。对此，微软提出基于统计的代码风格学习框架 Naturalize[16]。该框架的架构包含两部分：提议器和评分函数。提议器根据输入的代码片段给出一个能够代替输入代码片段的建议候选列表，在实际运行中由于会产生很多不切实际的候选提议，因此提议器中还包含了一个过滤器。评分函数将候选项根据代码自然性进行排序，其输入时一个候选的代码片段，返回一个相应的自然性度量值。

图 4-2 描述了 Naturalize 框架的架构。在用户界面中有一段连续的代码片段等待审查，由一些提议器返回一个候选列表，每个候选项都是将输入代码经过一些修改后的代码片段（比如有一个局部变量名被修改了）。这些候选项由评分函数进行排名（比如 N-Gram 模型），返回一个根据代码自然性排名的 top-k 结果。

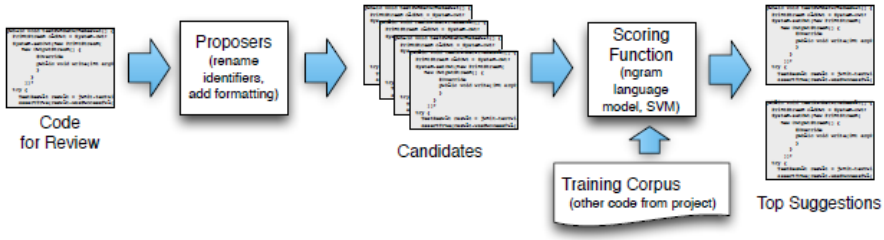


图 4-2Naturalize 框架

基于该框架设计了 4 个代码库分析工具（groups.inf.ed.ac.uk/naturalize）：

（1）devstyle：一个 Eclipse 插件，能对单个标识符或选定的一段代码中的命名和格式作出修改建议；

（2）styleprofile：一个代码审查助手，能够生成一个自动总结一段代码的编码习惯（风格）的简介，并作出标识符与格式更改建议以使该代码段具有更高的格式统一性。

（3）genrule：一个 Eclipse 代码格式化工具，能够根据 Naturalize 框架给出的编码风格生成相应的编码规则。

（4）stylish：一个 Git 预提交脚本，能够拒绝那些严重破坏代码风格一致性以及命名、格式自然性的提交。

使用该方法在 10 个开源项目上进行测试，对其中 5 个项目（elasticsearch,libgdx,junit,k-9,android-menudrawer）提出 18 项修改申请，有 14 项被项目作者采纳。

4.3 场景三 代码审查

Microsoft 发表文献 6 篇代码审查相关的文献[39],[7],[11],[55],[97],[78]，Google 发表相关文献 2 篇[58],[63]，Sony 发表相关文献 1 篇[62]。主要研究内容包括帮助理解代码审查、审查者智能推荐、代码审查评价等。微软提出一种根据审查者的历史审查记录、经验，自动推荐代码审查者的方法 cHRev[97]，其性能超过了现有方法 Revfinder、xFinder，与 RevCom 方法相当。文献[58],[63]提出不仅根据审查者的 GitHub 跨项目工作经验，而且根据其以往的 GitHub pull request 记录判断其作为潜在审查者的经验与可能性。值得注意的是，[58]与[63]是同一作者对审查者推荐这一领域研究的不同阶段产出文献。 文献[63]在 10 个商业项目和 10 个与这些项目相关的扩展库中进行探索性研究，接着用这些项目中的 17115 个 pull 请求进行验证，达到了 85%-92%的推荐精度、86%的查准率和 79%-81%的查全率。接着在文献[58]，作者将这一技术通过 C-S 模型实现，生成了一个 Google Chrome 浏览器插件。

本报告发现，Microsoft 与 Google 在这一问题上的解决思路是一致的：即通过代码审查人员的历史工作经验，筛选出最合适的代码审查人员。同时，从方法的角度看，由于软件设计构造问题中与开发人员直接交互的主要数据是代码，因此与代码语言处理相关的方法是这一领域内的主流方法，如基于贝叶斯分类器、统计概率模型的自然语言处理方法，此外还有少数文献使用了回归分析与图算法。此外还有如 N-Gram 模型、AST 树等传统软件数据分析方法。

总体来说，目前在软件设计与构造领域，开发人员与研究人员主要关注的还是与代码有关的技术，包括代码分析与优化、代码生成与补全、代码搜索与合并等，另外代码审查、需求分析、文档生成与恢复也是其重要组成部分。对于未来的发展与研究趋势，本报告认为业界仍然在巨头公司（特别是 Microsoft）的引领下，专注于软件设计过程中最重要的两方面：设计与编码，进行研究；同时，针对用户数据与用户需求，以及文档有关领域的研究也会呈积极向上的态势不断发展。同时，主要运用的方法，将会是以基于贝叶斯与统计概率模型为代表的方法为主，多种语言模型（如 N-Gram）辅助的发展态势。

- **其他主要文献介绍**

代码审查是业界公司进行软件开发过程中一个非常重要的环节，而代码审查人员的工作能力决定了代码审查的质量。对此，Google 提出一个代码审查者推荐工具 CORRECT[58][63]，该工具不仅考虑相关的跨项目经验，而且根据代码审查者在具体技术中的 pull-request 经验来评估其成为潜在代码审查者的可能性。其核心思想是：如果当前的一次 pull request 使用了与曾经的一次 pull request 相似的外部软件库或特定技术，那么可以认为这两次 request 是相关的，则之前一次 pull request 的代码审查者就有很大的可能成为当前 pull request 的审查者。

首先使用静态分析挖掘当前 pull request 中使用的外部库和其他技术信息，并识别出与当前 request 相关的以往 pull request，通过相似度对以往的 requests 进行评估排名，也即给出了审查者的推荐排名。

文章使用以下实验对该方法进行评估：首先，使用 Vendasta System 的 10 个子系统中的 13081 个 pull request 和 4 种流行的性能度量方式（Top-K、MRR、Precision、Recall）对该方法进行评估，得到了不错的实验结果：92.15% 的 Top-5 准确率，0.67 的 MRR，85.93%的查准率，81.39%的查全率。其次，与 State-of-the-art 方法 Thongtanunam 相比较，在 Top-5 准确率上提升了 11.43%，在查准率和查全率上各自提高了将近 10%。再次，在 GitHub 中的 6 个开源项目上进行实验，得到了 85.20% 的 Top-5 准确率，0.69 的 MRR，84.76%的查准率，78.73%的查全率，并且证明了

该方法是语言独立的，即不会因为不同的编程语言或项目是否开源而产生偏差。最后，计划进行 User Study，通过调查至少 10 个不同项目的至少 10 个开发人员来对该方法收集是否有用的反馈。

同样地，针对代码审查者推荐问题，Microsoft 提出一种自动推荐代码审查者的方法 cHRev[97]。该方法有一个基本前提：一个审查过代码中某个单元的审查者更有可能会帮助相应部分未来的审查。

cHRev 算法由以下步骤组成：

步骤 1：提取待审查源代码(Extract source code under review)。给定一个需要审查者审查的代码变更，提取出每个源代码文件。

步骤 2：生成审查者知识体系(Formulate reviewer expertise)。对于步骤 1 中的每个源代码文件，考虑在该文件上有多少曾经进行过多少次代码审查、是谁审查的、是何时审查的，生成一个审查者知识模型，该模型主要有以下三种度量：

（1）审查者提出的审查评论总数 C；

（2）审查者能够提出审查评论的有效工作日数 W；

（3）审查者给出的审查评论的最近时间 T。

步骤 3：评估并推荐审查者(Score and recommend reviewers)。根据步骤 2 中的信息对每个审查者给出一个评估并进行排名，这里用到了一个评估函数，其定义如下：

$$\mathrm{xFactor(r,f)} = \frac{RE_{(r,f)}}{FR_{(f)}}$$

其中，$RE_{(r,f)}$表示审查者 r 对文件 f 的审查量，$FR_{(f)}$表示文件 f 的审查总量。

$RE_{(r,f)} = \langle C_f, W_f, T_f \rangle$，其中$C_f$为审查者 r 在文件 f 上提出的审查评论总数，$W_f$为审查者 r 在文件 f 上提出评论的有效工作日数，T_f为审查者 r 在文件 f 上提出评论的最近日期。

$FR_{(f)} = \langle C'_f, W'_f, T'_f \rangle$，其中$C'_f$为所有审查者在文件 f 上提出的审查总数，$W'_f$为所有审查者在文件 f 上提出的所有审查所包含的有效工作日数，T'_f为在所有审查者对文件 f 提出的审查评论中的最近日期。因此审查者对文件的贡献程度可以表示为：

$$\mathrm{xFactor(r,f)} = \begin{cases} \dfrac{C_f}{C'_f} + \dfrac{W_f}{W'_f} + \dfrac{1}{|T_f - T'_f|} & if |T_f - T'_f| \neq 0 \\ \dfrac{C_f}{C'_f} + \dfrac{W_f}{W'_f} + 1 & if |T_f - T'_f| = 0 \end{cases}$$

将一个审查者对所有审查文件的贡献进行求和，即得到审查者的评估得分。文章将

cHRev 与其他方法进行了比较：如 REVFINDER 方法根据查找与待审查文件有相似文件名的代码的审查者来进行审查人员推荐；xFinder 方法根据历史提交记录进行审查人员推荐；RevCom 方法根据代码审查记录和提交记录结合进行审查者推荐。

文章使用 cHRev 方法对三个开源软件（Android, Eclipse, Mylyn）和一个闭源软件（Microsoft Office）上进行评估实验。其中 Android 数据包含从 2015 年 2 月 7 日-2015 年 3 月 26 日共 2052 次代码变更，2680 个代码审查者和 23181 条审查评论；Eclipse 数据包含 Gerrit 中从 2013 年 3 月 5 日-2014 年 11 月 28 日共 1854 次代码审查记录、10506 条审查评论和 3155 次提交记录。Mylyn 数据包含 Gerrit 中从 2012 年 3 月 2 日-2014 年 11 月 28 日共 1589 次代码审查，10157 条审查评论和 1838 次提交记录。Microsoft Offic 数据包含 2651 次代码变更，1886 次代码审查，10746 条审查评论和 845 名代码审查者。

在以上四个数据集上对 4 种方法进行实验评估，cHRev 方法在查准率、查全率、F-score 和 MRR 上都优于 REVCINDER 方法和 xFinder 方法，而 RevCom 方法并不比 cHRev 方法更好（但是 RevCom 是结合了审查记录和提交记录两方面数据的）。

同时，研究不同因素对代码审查带来的影响可以指导开发者更有效地进行代码审查。微软对此进行了研究[55]，调查哪些因素会使一次代码审查成为有用的（useful），并根据调查结果实现了一个分类模型用于对代码审查进行是否有用的反馈。该研究分为以下三个步骤：

步骤 1：在 Microsoft 内部进行了一次对代码审查有用性的探索新研究，采访了开发人员对“useful”的审查的理解，最终将有用性分为“有用（useful）”、“可能有用（somewhat useful）”和“没用（not useful）”三类。研究发现：①有用的评论通常会导致与评论位置很相近的代码被修改；②评论状态被标记为“Resolved”的评论通常是有用的，而状态为“Won’t Fix”的评论通常是没用的。

步骤 2：用步骤 1 中发现的结论，训练一个能够区分代码审查是否有用的分类器。从训练数据中提取出 8 个特征：状态、参与人数、评论数、作者是否评论、交互数量（类似于回帖）、是否导致附近的代码修改、关键词（fixed, bug 等）、情感分析（positive or negative），训练出一个决策树模型。

研究使用的训练数据来自 5 个 Microsoft 项目（Azure、Bing、Exchange、Office、Visual Studio）中的 844 条审查评论，其标签是由源代码作者人工标注的。文章使用的决策树模型在训练样本数据上进行了 100 次 10 折交叉验证，达到了 89.1%的平均查准率和 85.1%的平均查全率以及 16.6%的平均错误率。

步骤 3：将分类器运用到 Microsoft 项目的约一百五十万条代码审查记录中，研究其中哪些审查是有用的，进而探讨影响代码审查的主要因素到底有哪些。这里使用的是同样来自上述五个项目中的代码审查记录，通过研究这五个项目的不同作用域（开发环境、办公软件等）、审查是否有用等信息，从审查者和代码变更两个特征探讨不同因素对代码审查的影响。

研究得出的结论有：

（1）有相关审查经验的审查者能给出更多有用的审查信息；

（2）来自其他 team 的审查者比同 team 的审查者给出略微更多（0.2%-3.1%）的有用审查信息；

（3）代码审查有用性密度随开发时间增长；

（4）大型（代码变更很多的）代码审查往往会使评论有用率变低；

（5）在各种类型文件中（源代码、脚本、配置文件、构建文件），源代码文件产生有用评论的可能性最高 70%，构建文件最低 65%。

此外，在不同的软件公司中，有着各自不同的规章制度、公司文化、激励制度和时间管理模式，因此在项目开发过程中的代码审查也是在这些不同的环境下进行的。为了更好的理解这些差异所带来的影响，微软[11]对两个 Google 项目（Android、Chromium OS）、三个 Microsoct 项目（Bing、Office、MS SQL）以及 AMD 的内部项目进行研究，并与传统的软件检查（inspection）——朗讯内部项目和 Apache、Linux、KDE 等 6 个开源项目的数据进行对比，研究审查的时间间隔和参与审查的开发者数量。

研究发现，许多代码审查过程的特征都会收敛到一些相似的值——他们认为这些值解释了代码审查过程的一般规律；同时，文章提出了一种代码审查过程中对知识分享程度的度量方法。

收敛规则一：当代同行审查都遵从一个轻量化的弹性过程：

1.作者变更了代码，并提交等待审查；

2.开发者对变更进行商讨并给出修改建议，代码变更可以被提交多次来针对建议进行修改。

3.一个或多个审查者同意变更，将其并入 main 中。或者变更被拒绝。

收敛规则二：代码审查发生地很早、很快、很频繁。

收敛规则三：代码变更的量都较小。

收敛规则四：两个代码审查者是最优的缺陷检测配置。

收敛规则五：代码审查从单一的缺陷查找活动变成一个群体的问题解决活动。

研究发现，进行同行审查可以使开发者对项目提升 66%-150%的知识理解。

（1）当代同行审查会在代码提交后规律性地快速进行，而非整个项目完成以后进行；

（2）当代同行审查通常包括两个审查者，但这个数字不是一成不变的，故根据各种因素调整；

（3）同行审查的审查者更喜欢讨论、修改代码，而不是单纯地报告缺陷。

（4）各种审查工具的出现能够进一步确保软件质量。

4.4 场景四 代码搜索

随着软件工程的逐步发展，越来越多的软件开发积累了大量的源代码，许多代码搜索工具被用来在这些庞大的代码库中搜索那些能够被重复利用的代码，但是在实际中这些工具的准确性常常不能令人满意。对此，微软提出一种能够识别代码中潜在的 API 从而提供更准确的代码搜索的技术 – CodeHow[50]。CodeHow 的总体结构如图 4-3 所示，给定一个用户查询，首先将其提供给 API 理解部分（API Understanding），用来识别与查询相关的潜在 API；检索部分（Code Retrieval）根据理解出的 API 检索相关代码片段，得到原始的代码片段（Raw Code Snippets），使用静态切片将冗余部分与无关代码去除，得到最终的返回结果。

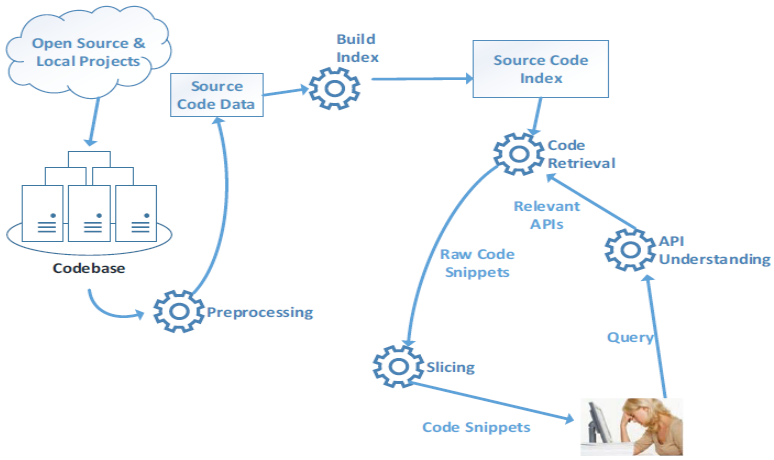


图 4-3 CodeHow 总体结构

图 4-4 描述了 API 理解部分的方法。首先从 API 在线文档中收集相关描述，接着计算该文本描述与查询之间的相似度以及 API 名字与查询之间的相似度，得到高相似度的结果。最后得到相应的 API 列表。CodeHow 利用检索的 API 列表及查询关键词进行关键词匹配，从而得到代码搜索结果。

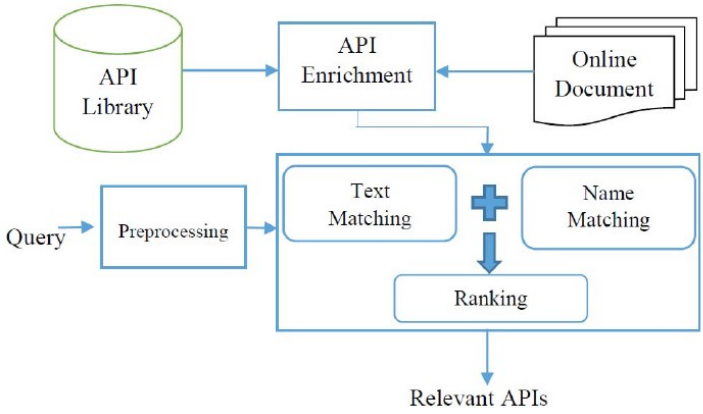


图 4-4 API 理解部分的结构

将 CodeHow 在一个 GitHub 项目中的 26K C#代码上进行验证，实验结果表明 Top-1 准确率达到 79.4%，超过了其他传统代码搜索工具。

4.5 场景五 代码重构

谷歌针对未充分利用目标（underutilized targets）的分解问题(NP-Hard)提出一种基于目标强连通分支的贪心算法[32]。用 40,000 个 Google Java 库中的目标进行测试，发现这些目标中有 19,994 个可以分解成两个目标；（2）有 1010 个目标在分解之后节约了 50%的调用时间。文献[17]提出一种基于搜索的自动迁移构建脚本的动态方法，并使用偏序缩减方法加速搜索重构序列，发现基于这种搜索能降低 46%的重构脚本大小，且搜索速度比普通搜索快 6.3 倍。

4.6 场景六 文档生成

文献[15]从代码数据着手，提出基于不变量来表示初始模型+调用追踪的模型接口技术，用于自动生成代码 API 文档，分析了分别只用其一和两者都采用的方法的三种方法，提出 SEKT、TEMI 两种算法，分别在测试集上达到 100%精度、34%查全率和 99%精度、65%-75%精度。 文献[43]从自然语言着手，提出使用交叉熵和 N-Gram 模型对自然语言的自然性（naturalness）进行度量，从而对软件领域中自然语言工件进行研究。将 StackOverflow 中的代码、评论、报告作为训练数据，对不同项目中的提交信息（commit messages）进行预测，达到 70%-90%的预测精度；对提交注释（commit comments）和源码注释（source comments）进行预测，分别达到 79%-93%以及 56%-78%的精度。

5 软件运维演化

在软件运维演化方面，开发过程分析全面的概括了软件开发过程中的优化和改进过程。缺陷预测、修复、分析占据的比例很大。而日志分析和软件维护也有许多实用价值很高的文献。

5.1 场景一 开发过程分析

微软公司提出了利用多模型的数据来帮助开发者通过在开发、部署、改良软件产品的循环中进行快速迭代的方法[2]。文献通过对测试者行为日志数据的分析，预测测试者下一步可能的动作。图 5-1 为该方法的流程图。

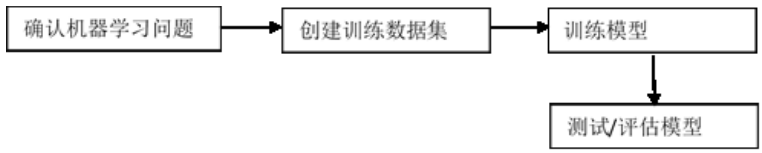


图 5-1 模型流程图

在进行日志数据采集时，微软的做法是将浮动的变量变成**特征流**。例如：在视频中的人脸的 X 轴坐标会被追踪为一个 Xcenter 的特征流，表明以 X 轴的中心替代 X 轴坐标数据。同时二进制的特征流表现为是或非的问题。这些特征流可以存储来自于传感器的信息或者运行时系统的组成部分。这些特征流可以以压缩的形式储存于硬盘中，并且可以被输出成 text 格式以供 matlab 或 excel 使用。

在训练模型的过程中，使用到了**决策树、线性 SVM、逻辑回归**等机器学习算法。将模型整合到系统中，在运行时执行。

Lync[72]是微软设计的一个可视化的系统，它通过在先前在**开发阶段的表现、评估过程、识别错误以及预测使用的时间**来帮助开发团队识别和优先处理软件表现方面的问题。在帮助大公司软件进行开发决策的过程中起到了重要作用。

现实生活中的性能表现很难测量出来。该文献讲述了如何测出真实世界中依靠**网络**的应用。在文献中，性能表现定义为在一个网络条件规定的情况下的所用时间。在开发时就让软件收集数据。得出的数据可以**帮助软件开发商**作出决定。使用 Engineering Intelligence (EI 分析)来调查数据性能表现。此方法已经在微软公司内部成功部署并且帮助决策。Lync 可以在开发的阶段就测量每个场景的表现。

该文献提出如下的问题：

（1）Lync 的性能如何在用户使用四人会议、八人会议和所有人会议的情况下测量。

（2）四个人谈话和一个人说话三个人听的性能表现差异在哪？

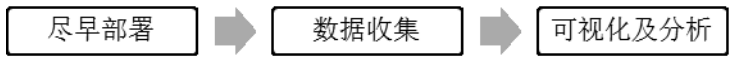
（3）在不同地狱或离服务器不同距离的情况下的性能差异？

（4）服务器的种类影响性能表现吗？

上述问题的问答可以帮助决策者**选择未来开发的重点**以及**将来预期的结果**。

该方法的目标就是获得并分析来自 lync 的真实使用数据并在早期开发中使用。

方法的总体流程如下：



Lync 早期系统帮助进行性能报告以及用户满意按钮选择。

数据收集使用内嵌事件驱动 API。每当事件发生时调用，必须输入事件的开头和结尾。开头为 UI 界面调用，结尾为 I /O 输出。数据保存时保存**整个事件的名字以及其组成部分**。数据被保存到服务器。图 5-2 为四个场景的结果视图。显示出每个场景的名字、组成部分以及其性能表现。

图 5-2 四个场景的结果视图

在分析数据时，应用允许用户检查性能表现的结果，并允许他们在更细的粒度上观察数据。图 5-3 展现了 EI 分析每个场景的性能数据，越往右的高斯曲线表示用户体验越差。

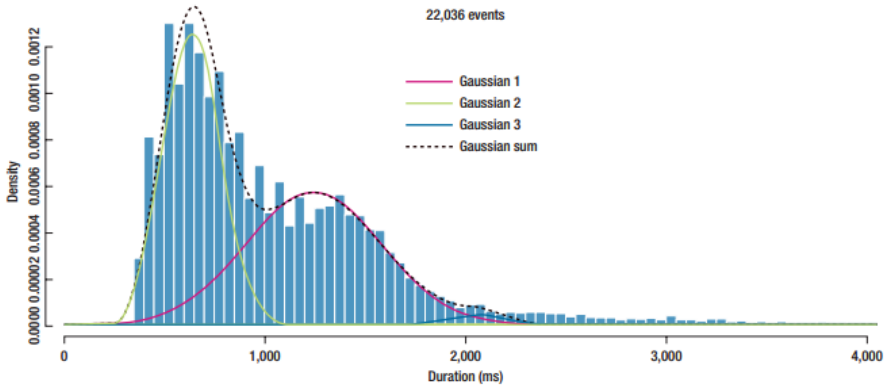


图 5-3 单个场景的分析图

EI 分析同时支持在不同的环境条件下的性能差异比较，图 5-4 为在 32 位和 64 位机器上应用的性能比较。

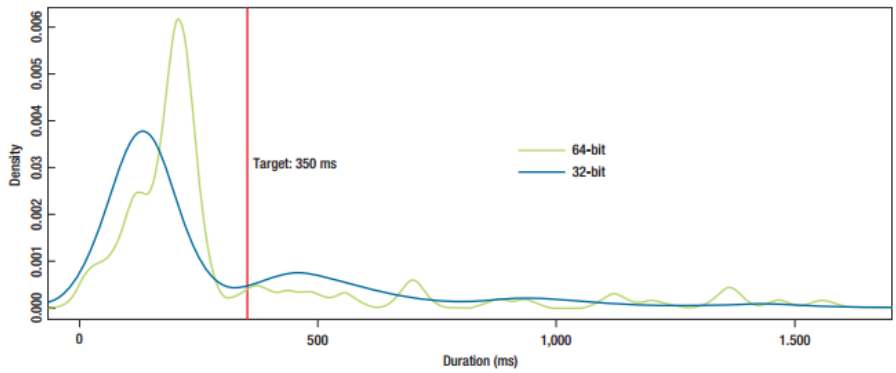


图 5-4 32 位和 64 位性能比较

EI 分析通过以上方法分析出不同场景的性能差异。EI 分析可以帮助提升用户满意度，在软件开发过程和决策制定时有着积极的影响。

CODEMINE[74]是微软提出的用于**收集和分析工程数据**的软件开发数据分析平台。一个尽早的、可信的数据可以帮助工程师和产品经理做出有**利于高质量软件系统开发的决策**。在微软，各个团队会使用数据来提升软件开发过程，例如：

（1）监督和报告产品安全

（2）风险评估

（3）分支结构最优化的版本控制

（4）搜寻 bug 和 debug 日志，加速新问题的调查

该文献指出，上述问题看似有着不同的目标，其实他们有着相类似的输入、输出以及工具所使用的方法。因此，微软公司创造出了 CODEMINE 来收集分析数据。如今，CODEMINE 已经广泛地部署在了微软的主要产品中，例如：Windows, Office, Bing 等。

图 5-5 展示了 CODEMINE 的主要数据类型，其中源代码占据了很大比例。在目前的 CODEMINE 版本中，源代码的改变、分支、整合是输出的主要特点。

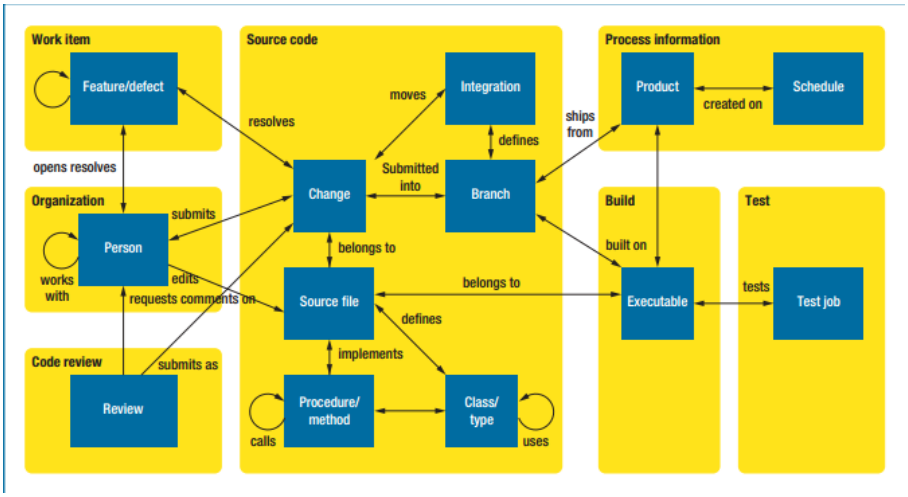


图 5-5 CODEMINE 数据类型

图 5-6 显示了 CODEMINE 的架构。系统主要包含以下几个部分：

1.数据存储

存储的数据类型不一定要相同，它们可以被分成各种类型。

2.数据载入

数据载入为读取源数据并直接把它们放到数据库的过程。

3.平台 API（数据模型）

平台聚焦于以下几种特征的数据：错误、特征、测试、人、属性和联系。

4.平台服务

平台服务包含一系列的与数据目录有关的工作，包括查询权限、实践日志、数据存档、数据发布等。

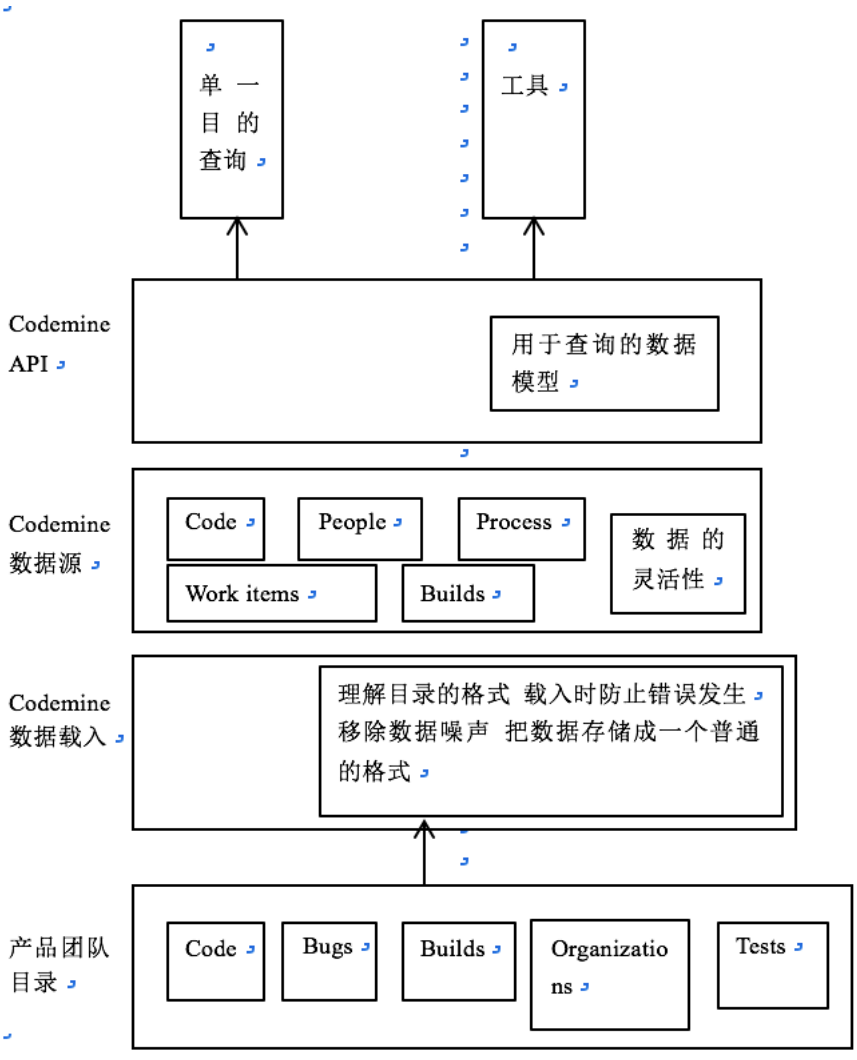


图 5-6 CODEMINE 架构

CODEMINE 平台的使用场景主要有：

1. 用于产品团队开发过程的报告工具或方法的数据源。
2. 在开发的过程中用于回答问题的分析方法。
3. 预测将来新的研究方向。。

5.2 场景二 社区问答

三星公司提出利用 Google Play 上**用户评论**以评定手机 app **安全性能**[52]。该方法使用了 AUTOREB 工具来自动评估 app 的安全度。它使用了机器学习的方法推断用户的评论以及安全性能的关系。在预测 app 安全性能的表现上十分出色。图 5-7

为该系统的框架。

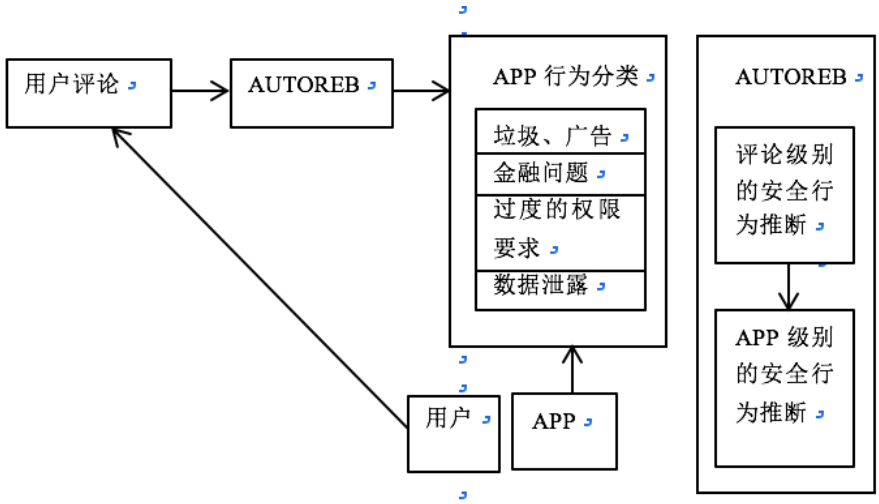


图 5-7 AUTOREB 系统框架

可以看到，AUTOREB 手机用户评论后评定 app 的行为，给出评论层面的安全推测，最后给出 app 层面的安全推测。

在该方法中，app 行为被定义为图 5-8 中的四个方面。

目录	行为描述
垃圾、广告	通知栏里的广告、垃圾邮件、通过 SMS 的垃圾、弹出广告、钓鱼软件等
金融问题	APP 内付款后得不到商品、APP 附加费用等
过度权限要求	APP 需要使用过多的权限要求
数据泄露	在未经用户同意的情况下读取个人数据，例如用户账号、联系人、位置等

图 5-8 app 行为

图 5-9 为 AUTOREB 推测引擎的方法过程

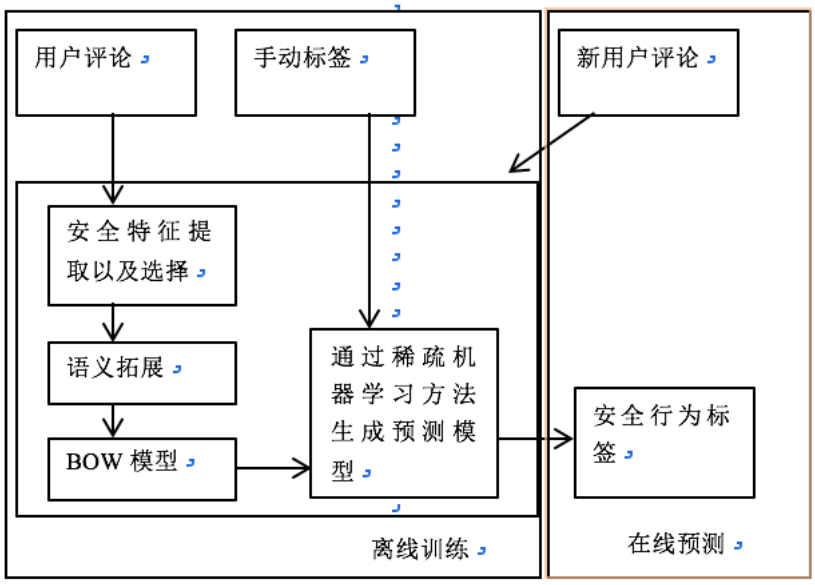


图 5-9 AUTOREB 推测引擎的方法过程

在**安全特征抽取**阶段，系统抽取和 4 个图 5-8 中的行为最相关的词语。

在**语义扩展阶段**，系统发现语义接近于安全特征的单词或者短语，并且通过**添加**相关的词语来拓展原先的评论。

在**训练模型阶段**，使用包括 KNN 以及 SVM 的分类器进行训练。（实验中选取**稀疏 SVM**）。

在**测试阶段**，给出 AUTOREB 分类的结果。

5.3 场景三 缺陷预测

缺陷预测场景内容广泛，可分为数据库缺陷预测、javascript 代码缺陷预测、方法函数接口缺陷预测、手机 app 缺陷预测、基于日志的缺陷预测等，以下便从这几个方面进行缺陷预测场景的分析。

（1）数据库缺陷预测领域：

文献[12]针对数据库语句的修复使用了传统的数据挖掘方法：半监督学习、SVM、决策树等，在不同的实例中进行了测试，在一般的数据中准确率 90%以上。可见方法使用较好。普遍性较强。文献[94]是在 oracle 数据中进行测试，使用了变异分析的方法，查错效率提升显著。普遍性较弱，专门针对 oracle 数据。

（2）Javascript 代码缺陷：

三星公司提出用于 javascript 代码的错误预测的方法[13],[35]。目前，javascript 应用广泛但因其动态的特性导致错误很难被发现。两篇文献均使用了工具 SAFEWAPI 以及 Dlint。在 javascript 的 web 应用上进行了测试。文献[35]的 Dlint 能找出 53 个

错误，其中 49 个错误均未被常规方法发现，证明其性能良好。

（3）方法、函数、接口的错误：

文献[5],[20],[42]均是研究代码中方法、函数或者 API 的缺陷预测情况，在这一方面的现状均可归纳为：目前的缺陷预测方法费时费力且费用昂贵。在这样的背景之下，三篇文献均提出了不同的方法。其中[5]、[20]文献分别提出了字节码操作以及崩溃栈分析这样非典型数据挖掘的方法。而文献[42]则使用随机森林的数据挖掘方法进行预测。三个方法具使用了各自工具。三篇文献均在开源的项目和数据上进行了测试，其中文献[5]预测的准确率为 80%-86%，而文献[20]错误定位的准确率大于 60%，比基于平常栈的方法更加优秀，回归率也上升了 20%。而文献[42]提出的方法准确率也十分可靠，f-measure 数据平均为 0.681。三篇文献各自方法都值得探索，同时文献中也提到未来会将方法运用到更多的项目上进行测试。

（4）手机 app 错误、恶意软件的查找与安全验证：

文献[61],[30],[52],[98]均是针对手机 app 应用场景错误审查以及安全验证的。目前的状况可以归纳为：目前恶意软件越来越复杂，与正常软件的相似度越来越低，且目前恶意软件的分类器效果不佳，导致恶意软件很难被发现，app 评级很难完成。在这样的背景下，四篇文献均提出了不同的方法应对这一局面。其中文献[61]尝试使用了诸多传统的数据挖掘方法（特征选择、KNN、SVM、朴素贝叶斯、线性回归、随机森林、AdaBoost、半监督学习方法）在安卓 APP 上进行测试，结果发现各方法表现各有千秋，但在特征选择方面基于语义的特征可以提升鲁棒性。而最终方法的准确率高达 67%-73%（传统方法只有 26%-55%）。提升较为显著，可深入进行研究。文献[30]则关注于手机应用自动找错功能。使用灰盒测试方法，使用 VanarSena 工具在 windows phone 应用报告上进行了测试，工具揭示了 2969 个错误其中 1227 个之前没有发现。性能提升也十分明显。可深入进行研究。文献[52]是基于用户评论评定安全性能，使用了众包自动聚合的方法，使用 AUTOREB 工具在用户评论上进行测试。预测准确率高达 94.05%。相较前两篇文献，此方法从不同的角度入手也取得了相当不错的结果。可深入进行研究。文献[98]也是关注恶意软件的查找，使用了传统的数据挖掘方法（逻辑回归以及神经网络）使用 MAP 工具，在恶意软件数据集上进行了测试，并未显示性能上的提升。较前三篇文献数据略显不足。

总体而言，手机 app 审查是一个目前相当实用的方面。就以上四篇文献的综述而言，前三篇文献提出的方法性能提升显著，可以作为下一步研究的目标。

（5）基于日志的缺陷预测场景：

文献[60]主要用于聚集日志来发现基于日志的错误识别，现状是在线系统崩溃

时，工程师需要查看日志解决问题，使用单个词搜索失败率较高。文献使用聚类的方法，使用 LogCluster 工具。在基于 Hadoop 的应用以及两个大型微软在线服务系上进行实现。比原先的方法更加有效，在 wordcount 和 pagerank 两个算法上比 keyword search 和 icse 之前发表的方法都提升了效率。可进行深入研究。

（6）常规的缺陷预测场景：

文献[44],[45],[47],[27],[81],[24]均是基于常规的缺陷预测场景。其中[44]以及[45]均使用了传统的数据挖掘方法（[44]为 PCA、随机森林，[45]为 PCA、逻辑回归）。其中[44]在 win8 系统崩溃报告上进行测试，结果回归率有所提升。而[45]的方法测试结果则不太稳定，未能超越一些之前发布模型。后三篇文献均使用了非典型的数据挖掘方法。其中文献[47]从运用的普遍性而言比较狭窄，使用程度可能较低。而[27]提出了一个新的时空协同网络模型，是根据测试人员的活动记录进行判定，具体的可实施度还有待商榷。文献[81]虽然没有使用传统的数据挖掘方法，但其栈追踪分析的方法应用比较广泛，测试结果与其余方法比较均较好，值得深入探究。文献[24]是研究错误曲线模型，运用了 weibull 模型和线性回归方法。在开源项目上测试效果良好，可视具体场景来选择研究。

5.4 场景四 缺陷修复

文献[4]基于语义分析而文献[84]使用了程序状态抽象、错误定位、修复合成方法。相比而言，文献[4]的方法准确率和速度更优秀，但文献[84]的方法也可以参考。两篇文献均有研究价值。以上两篇文献均是缺陷修复方面的文献，分别使用了不同的方法进行错误修复。

5.5 场景五 缺陷分析

文献[26], [67], [82]分别提出了不同的缺陷修复方法。文献[26]是用于分析时序软件工程数据，覆盖面较为狭窄，有相关问题时可以参考。文献[67]是用于故障识别，目前现状是故障识别是一个冗长和充满错误的过程。其使用了基于 BFS 的封闭项集数据挖掘，基于影响、改变检测和隔离功率的修剪方法，使用 iDice 工具。在微软在线服务系统中运用，证明其可以检测故障减少维护成本。使用程度比较高，可进行深入探究。文献[82]偏向于项目管理，是主要用于将 bug 分配给适当的开发者，现状是手动分配过程十分冗长。使用了特征提取、VSM(Vector Space Model)方法，使用了 BugFixer 工具。在三个大型开源项目和两个小的工业项目上测试，结果比同类型的方法在大型项目上表现更好在小型项目上表现相当。在需要分配 bug 任务时可进行深入研究。

5.6 场景六 日志分析

此场景并非使用日志分析缺陷，而是使用日志信息提升开发效率。微软在文献[38],[69],[77]中提出了不同的日志分析方法。其中[38]和[77]均偏向于日志分类和自动分析日志方面。两篇文献均涉及传统的数据挖掘方法，并都通过测试，性能有所提升，可进行深入研究。文献[69]是研究诸如 siri 以及 contana 提醒助手的软件日志，对这些日志进行学习可以设计一个系统来帮助人们完成工作和进行未来计划。使用数据驱动分析的方法。普遍性不高，可视情况进行研究。

5.7 场景七 软件持续演化

文献[25],[85]均为软件维护场景。相比较而言，文献[25]是从故障分析表中使用聚类方法分析预测数据，准确率提升显著，降低软件维护成本。可以深入研究。文献[85]是用于代码克隆一致性维护方面，使用了贝叶斯网络方法，在微软项目以及开源项目上进行测试，预测准确率高，可进行深入研究。

6 软件项目管理

经过筛选，本报告共挑出 11 篇项目管理方向相关的文献，其中 7 篇来自 IEEE 期刊，4 篇来自 ACM 期刊。

6.1 场景一 开发者画像

对开发商而言，修复一个问题的时间是衡量一个已广泛部署的商业软件质量最重要的指标，也直接影响到了用户的体验。

文献[70]通过对 5 年以来 IBM 中级产品的缺陷记录分析，提出了一个对开发者修复某个相关缺陷的能力进行预测的可行有效的方法。数据来源：1500 条产品缺陷记录（1,500 defect records from an IBM middle-ware product collected over a five-year period），主要的数据挖掘方法：LDA (Latent Dirichlet Allocation) 狄利克雷划分算法。

文章作者通过对 IBM 的某个中级产品约 5 年 1500 条缺陷记录的分析，其中包括对每个修复好的缺陷进行特征化主题分类，以及对每个开发人员在所涉及到的缺陷相关主题 topic 进行专业能力上的排序。分析发现解决缺陷问题的时间与每个开发人员及其在相关话题的专业能力明显相关。为了验证可行性，作者采访了团队的产品经理，他提供了他认为的程序员相关专业知识的能力排序用于对照。作者认为，该文章所提出的自动化的开发者能力排名方法不但有利于软件项目开发过程，也有助于其他软件生命周期。

这项研究的意义在于，对于一个商业软件项目的经理来说，一般他会将不同领域的缺陷修复问题分配给最熟悉这个可能会出问题的代码片段的开发人员，在这种情况下如果该开发人员在该问题领域有很好的专业知识就更好了。但是通常情况下，项目经理不一定清楚团队每一个成员擅长的技能和熟悉的知识领域，尤其是对大型的、全球化的团队而言。尽管有一些策略会让开发者自己选择缺陷去做研究工作，但是如果管理者知道团队成员的专业领域则会对整个开发工作有帮助，甚至在代码写出来之前就能提前分配好相关工作，可以很大程度上提升管理和开发效率。

该文献提出的方法首先对缺陷记录(defect records)进行分类，使用一个基于狄利克雷划分(LDA)的自动主题分析器，通过对缺陷描述的分析来进行分类工作。然后用已经分类的缺陷记录，计算每一个开发人员在每个缺陷主题下的解决问题时间（缺陷记录上的时间相减）。基于更快的解决问题表明更专业的相关技能的假设，该文章得到了两个排列(rank)：一个是每个主题领域下，开发人员的能力排名；还有一个是每个开发人员的不同主题的缺陷修复能力排名。尽管缺陷修复的时间可能受别的同期工作的影响，但是这种影响应该对每个参与分析的开发者是一样的，所以可以不

考虑。

最后将这个方法得到的结论和团队的项目经理给出的结论对照，发现其结果与项目经理给出的参考非常相近。该方法给出的 overall 排名为 C,D,B,A，项目经理给出的排名为 D,C,B,A。其中 D 与 C 之间的排名区别在于 D 经常分配到“project”类型缺陷，这类问题往往更难，因为文章中的方法没有对不同主题的缺陷进行困难的排名，所以没有办法预测到这一点。

文章中对于大数据的分析过程如下：

1. 筛选：对原始数据 19000 条缺陷记录进行多层面筛选，和开发团队讨论后，先去掉了时间过于久远的缺陷记录，再去掉一些不活跃的开发人员，只保留项目核心的开发人员 4 人，匿名为 A、B、C、D，这四个人解决了整个数据集中 15%的缺陷，成为了本文主要研究的对象。

2. 主题划分：使用狄利克雷划分，一个十分流行的主题建模方法，通过对缺陷的描述使用 LDA 方法，得到了 4 个缺陷主题，人为的检查了这四个主题的结果后，将其分为“code”, “install”, “logic”, and “project”来指示 LDA 的结果。

文献[80]通过对开源代码和大量错误报告的分析，提出了一个基于网络关系分析的方法来识别一个项目中修复漏洞的关键开发人员。数据来源：开源代码和商业软件项目的错误报告(98304 bug reports across 11 open source and 5 commercial software projects)，主要的数据挖掘方法：MEG(Minimal Essential Graph) algorithm 基本图算法，Directed Tossing Graph 有向再分配图。

文章提到解决漏洞是软件维护过程的重要目标，而在修复漏洞的这个环节中，一般情况下谁通过代码的改变解决了漏洞，谁就是这个环节最重要的参与者。然而还有很多人在这一环节并不提交任何代码，比如 reporters，做漏洞汇报工作的，还有了解深层次的技术来提供深层思路解决漏洞的人，还有 bug-tossers 将漏洞分配给适合解决这一类问题的开发者的人，他们都是对修复漏洞有很关键贡献的人。尽管不提交代码的人参与了漏洞的修复工作，但是并不是他们所有人都对解决漏洞有关键性的帮助。这篇文献通过研究 98304 个漏洞报告（来自 11 个开源代码和 5 个商业软件项目），来定义 non-committers 非代码提交者，和识别这些人中关键地担当漏洞修复催化剂的开发者。文章中提出了一个基于关系网络分析方法来构建一个 MEG 基本图来识别项目中非代码提供者中的关键人物。

文章作者先表明了参与了评论的开发人员也在解决 bug 中扮演一定的角色，没有其他参与修复漏洞的人的贡献，提交代码解决问题的那一个关键的开发者可能也无法想到解决漏洞的办法。通过对收集到的漏洞报告的分析，文章先给出了 Essential

Non-Commuters 的定义，从修复漏洞的任务分发，对于漏洞的评论两个角度给出了关键的非代码提供者的定义，并称其为 catalysts(催化剂)。然后提供了一个构建 MEG 基本图的算法，应用于从漏洞报告中建立的有向再分配图 TG，从构建得到的 MEG 中，又得到了每个项目中的非代码贡献者的总数量、catalysts 的数量以及 catalysts 平均分配的漏洞数量、non-catalyst 的数量以及他们平均分配的漏洞数量。

文章中大数据的分析过程如下：

1.数据收集：收集了 Apache Software Foundation 提供的 11 个开源代码和 5 个 IBM 公司提供的商业项目中的总共 98304 个漏洞报告。5 个商业项目中有两个：CA 和 CADW 由 IBM 研究室开发，其他三个来自于 IBM 软件组。11 个开源代码使用 JIRA 作为事件追踪记录系统，5 个商业项目则使用 Rational Team Concert（RTC）。

2.数据筛选：定义了结点与边的含义后，从原始数据中构建了一个 directed tossing graph 有向分发图，再提供一个算法用于从该图中构建一个 MEG 基本图，得到最后的结果。

文献[28]提出了一种方法通过检测开发人员编程困难问题时的状态，尽早阻止其写出错误代码来解决软件开发过程中的编程缺陷问题。这种方法通过分析生理心理学传感器和脑电图中的数据来进行预测，提供了新一代的困难问题的检测方法，并且具有很高的准确度。数据来源：生理心理学传感器数据(eye-tracking, EDA, and EEG)，主要的数据挖掘方法：机器学习(包括 boosted decision trees、linear SVMs、logistic regression)，朴素贝叶斯分类器，用到了 Weka(基于 JAVA 的机器学习的分类工具包)和 NASA 的任务负荷测量评估工具。

6.2 场景二 开发者和任务的智能推荐

文献[96]提供了一个名为 iHDev 的方法来推荐最适合的可以实现所改变的需求的开发人员。通过机器学习方法分析所给的改变需求文档，并对应上一段源码，然后挖掘与这段源码有关联的开发人员并排序，推荐给项目团队。对比研究表明该方法和以前通过提交记录和源码作者信息来推荐开发者的方式相比更具效率。数据来源：开源系统 Mylyn 以及开源的 Eclipse 项目（interaction log），主要的数据挖掘方法：KNN 邻近算法，机器学习算法(包括但不限于 boosted decision trees、linear SVMs、logistic regression)。

6.3 场景三 智能协作、任务分配

文献[72]通过对协作网络中的一些开发团队的源码库等开发记录进行大数据研究分析，提出针对大型全球化开发的商业软件团队的协作能力的提升方法和指出影

响团队稳定性的因素。数据来源：大规模商业软件开发记录(500 kLOC part of a commercial large-scale enterprise software product source code)，主要的数据挖掘方法：clustering algorithm 聚类算法，Louvain algorithm，Modularity measure 模块化度量方法。文献[76]开发了一个基于频繁模式挖掘的可拓展的方法，用于识别开发团队任务库中经常共同编辑的字段模式，减少了大量的解译任务库记录的工作。数据来源：微软产品团队任务资料库(包含 801,621 团队提交记录)，主要的数据挖掘方法：频繁模式挖掘，aRules package（用于挖掘关系和常见项目集合的计算环境）。

6.4 场景四 项目风险预测

文献[75]提出了一个方法通过给每一段改变的代码设置一些可以衡量的指标，然后使用经过大量现有数据训练的统计学模型来区分错误和无错误的代码改变，来尽早的检测出错误的代码修改。数据来源：项目 version control system (VCS)版本控制系统与项目 Bug Tracking Database（BTD）BUG 追踪数据库，主要的数据挖掘方法：SZZ 算法。文献[22]通过对 200,000 个 Windows 用户数据分析挖掘出影响软件稳定性可靠性的因素。文章发现软件可靠性和其他软件，游戏或者文件共享软件，软件配置，软件使用频率都有关系，提醒了项目开发人员，需要对这些因素有所了解。数据来源：200,000 微软 windows 用户的可靠性数据(reliability data from more than 200,000 users of Microsoft Windows and associated crashes of the most frequently used applications with software and hardware features)，主要的数据挖掘方法：频繁项目挖掘，Logistic Regression，Influence Networks 关系网络。

6.5 场景五 软件过程的度量和改进

传统的应用测试方法如建模工作负荷测试或控制环境测试已经无法对当前复杂环境中的协作软件性能进行精确的评估了。文献[73]提出了一个方法通过分析工具在应用开发过程早期提供一个应用的直观的性能表现评估，使开发团队尽早发现缺陷，改进主要性能。其使用的数据来源主要是大规模企业交流系统(Lync)的用户数据 large-scale enterprise communication system，主要的数据挖掘方法是 EI Analytics（reporting website 一个分析报告网站） EI 分析工具。文献[2]提供了允许系统开发者通过机器学习方法快速建模的一个多模态数据收集，可视化，标记，学习的框架，用于快速迭代产品。主要的数据挖掘方法：machine learning techniques 机器学习(包括但不限于 boosted decision trees、linear SVMs、logistic regression)，log explorer 记录挖掘工具，Multiple label generation mechanisms 多标签生成方法。文献[7] 利用基于数据流的贝叶斯推断对基于概率的项目进行审查验证，提出了一个新的基于数据

流的贝叶斯推断算法来计算基于概率项目的后验概率，用于项目的分析和验证，并提出了许多标准评估了新算法的结果。数据来源：Algebraic Decision Diagram(ADD)代数判定图(图状数据结构)，主要的数据挖掘方法：data flow analysis techniques 数据流分析方法，贝叶斯推断。

综合所有应用场景，可以看到项目管理相关内容中与软件过程度量和改进，开发者画像主题相关各有 3 篇文献。不难看出面向开发者的推荐筛选、评估等问题是项目管理领域比较重要也比较热门的内容之一。一个好的软件必不可缺好的开发者。如何找到擅长解决某一特定问题（比如某类缺陷）的开发人员来负责软件某一部分的开发，如何找到面对特定的需求变化有很强实现能力的程序员等等这些面向开发者的话题，将会是项目管理任务的重点，尤其是在涉及到大型、广泛部署的商业软件，在管理大型、全球化的开发团队时，面向开发者的问题所带来的效率和质量上的影响更显重要。而数据挖掘则在其中扮演的重要一环，应当受到更多重视，团队和企业也都将因此提高开发效率和收益。

大数据驱动着项目管理模式的更新和改进。与传统的项目管理工作模式相比，文献里提出的新的方法更加智能化自动化，也更加精准。而未来在项目管理方面，挖掘优秀、专业的开发人员将会是热门主题之一。

7 数据挖掘算法

从文献统计数据来看，数据驱动软件开发的主要方法主要分为两大类：①基于机器学习的数据分析技术和②传统软件数据分析技术。图 7-1 给出了两种方法在文献中的比重。其中机器学习的方法占 55%，可见机器学习技术已经在软件数据分析领域得到广泛的使用。

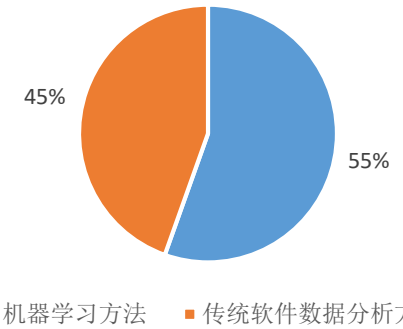


图 7-1：机器学习方法与传统软件数据分析方法在文献中所占比重

7.1 算法介绍

本节主要介绍文献中出现较多的几种机器学习方法与软件数据分析方法。

7.1.1 贝叶斯（朴素贝叶斯、贝叶斯网）

贝叶斯分类器的分类原理是通过某对象的先验概率，利用贝叶斯公式计算出其后验概率，即该对象属于某一类的概率，选择具有最大后验概率的类作为该对象所属的类。对于朴素贝叶斯分类器，假设所有属性都是独立的，及每个属性独立地对分类结果产生影响，则可以通过最大化各类别下所有属性的条件概率之积来求得该对象在所有分类下的后验概率。

IBM 基于独立微结构特征与贝叶斯网络提出一种编译器自协调框架，使开发者不需要花费过多时间针对不同平台进行编译器优化，并大大降低了编译时间[57]。

7.1.2 决策树

决策树是一类常见的机器学习方法，基于树结构来进行决策。在进行决策时，将问题或一系列的判断分解成一步步的子判断，根据一系列的子判断得到最终结论。常见的决策树算法有 ID3 算法和 C4.5 算法。

IBM 针对缺陷修复场景提出了基于 ID3 决策树的修复方法，并在 7 个真实项目代码中得到良好运用[12]。

7.1.3 线性回归、对率回归

给定有若干属性描述的样例，线性模型试图学得一个通过属性的线性组合进行预测的函数，使得预测函数与真实标记之间的差别尽可能小。当需要进行分类任务时，只需要找一个单调可谓你数将分类任务的真实标记与线性回归模型的预测值联系起来，当这个函数取对数几率函数 y=1/(1+e^(-x))，则将该模型称为对数几率回归模型。

IBM 研究影响软件错误曲线的因素以降低维护成本，运用线性回归，准确地只用外部发布的属性来描绘错误曲线[24]。

7.1.4 支持向量机

支持向量机（SVM）是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，其学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。对于一个划分任务而言，SVM 算法的目标是找到一个最优的划分超平面，使得距离超平面最近的异类分类样本到超平面的距离之和最大，即寻找具有最大间隔的划分超平面。

Microsoft 提出通过特征提取与 SVM 分类器从商业合同中提取规则化需求信息，指导软件建模[19]。Samsung 则提出通过 SVM 分类器从用户评论中自动学习理解与移动应用安全相关程序行为[52]。

7.1.5 频繁项集挖掘

Apriori 是一种常见的频繁项集挖掘算法，该算法的基本思想是：首先找出所有的频繁集，这些项集出现的频繁性至少和预定义的最小支持度一样。然后由频繁产生强关联规则，这些规则必须满足最小支持度和最小可信度。然后使用第一次找到的频繁及产生期望的规则，产生只包含集合的项的所有规则，其中每一条规则的右部只有一项，这里采用的是中规则的定义。一旦这些规则被生成，那么只有那些大于给定的最小可信度（阈值）的规则才被留下来，成为数据中可能的强关联信息。

7.1.6 数据降维方法

主成分学习（PCA）是一种常用的降维方法，该方法的核心思想是通过正交变换将原数据投影到一个新的维度空间，使得每个信息的相关性尽可能小，降维后的数据能够便于进行分析和计算。

IDM 使用逆向选择建模，并用 PCA 将数据降到 8 维，研究组织度量标准在创建缺陷模型时的作用[45]。

7.1.7 集成学习

随机森林是一种基于 Bagging 的集成学习方法。具体来说，传统决策树在选择划分属性时在当前节点选择属性集合中的一个最优属性，而随机森林是从该节点的属性集合中随机选择包含 k 个属性的自己，从这个子集中进行最优属性划分。

Samsung 在进行缺陷预测时运用随机森林算法，提升了错误检测率的同时降低了资源使用[42]。

7.1.8 N-Gram 模型

N-Gram 模型是一种常用的连续语言模型。利用上下文中相邻词间的搭配信息，假设第 N 个词的出现至于前面 N-1 个词相关，而与其它任何词都不相关，整句的概率就是各个词出现概率的乘积。这些概率可以通过直接从语料中统计 N 个词同时出现的次数得到。

IBM 使用 N-Gram 模型与交叉熵对自然语言文本进行度量，并提出基于 N-Gram 模型的文本预测工具，对软件项目中的提交信息、提交评论、源码注释进行预测[43]。

7.1.9 抽象语法树

抽象语法树（AST）是源代码的抽象语法结构的树状表现形式，这里特指编程语言的源代码。树上的每个节点都表示源代码中的一种结构。之所以说语法是“抽象”的，是因为这里的语法并不会表示出真实语法中出现的每个细节。比如，嵌套括号被隐含在树的结构中，并没有以节点的形式呈现；而类似于 if-condition-then 这样的条件跳转语句，可以使用带有两个分支的节点来表示。

Microsoft 基于 AST 树提出一种进行实时无冲突代码合并的算法 diffTree，并开发了相应编程工具[40]。

7.2 算法清单

表 7-1 给出了使用机器学习技术的相关文献清单。

表 7-1：文献中的机器学习方法清单

机器学习方法	引用文献编号
贝叶斯	[7],[16],[28],[38],[57],[61],[85]
NLP	[9],[19],[37],[51],[99]
概率统计模型	[36],[51],[64],[66],[80]
特征选择	[9],[22],[56],[61],[95]
PCA	[44],[45],[88],[93]
决策树	[12],[38],[78],[95]
随机森林	[42],[44],[61],[93]

对率回归	[22],[38],[45],[98]
特征提取	[21],[82],[95]
SVM	[21],[61],[38]
频繁项集挖掘	[22],[67],[76]
线性回归	[24],[61]
聚类	[25],[60],
kNN	[61],[96]
关系网络	[22]
图算法	[32]
神经网络	[98]
LDA	[70]
半监督学习	[61]
AdaBoost	[61]

除了机器学习方法以外，传统软件工程中也有许多数据分析技术也在业界广泛使用，表 7-2 给出了文献中使用的部分方法。

表 7-2：文献中的传统软件数据分析技术

软件数据方法	**引用文献**
N-Gram 模型	[37],[43]
Partial Program Analysis 部分程序分析	[46],[68]
基于 AST 的代码合并	[3],[40]
模型扩展表 MSP	[14]
符号逻辑分析	[18]
模块化程序生成	[113]
调用追踪	[15]
基于搜索重构的二阶段迁移	[17]
基于 operational patterns 的代码生成	[101]
信息检索向量空间模型	[50]
基于 API 调用的自动建模	[13]
交叉熵	[1]
字节码操作	[5]
崩溃栈分析	[20]
weibull 模型	[24]
众包自动聚合	[52]
时空协同网络模型	[27]
灰盒测试	[30]
基于 BFS 的封闭项集数据挖掘	[67]

数据驱动分析	[69]
逆向选择建模	[45]
栈跟踪分析	[81]
变异分析	[94]
VSM	[82]
FCA	[77]
TWINY	[79]
MEG	[80]
Louvain 算法	[72]
SZZ 算法	[75]
语义分析	[4]
跨项目的实例迁移技术	[53]
改进变化分类技术	[47]
关键预测指标与客户体验度量数学建模	[54]
提出一种新的针对顺序的模式查询语言	[26]
程序状态抽象、错误定位、修复合成	[84]
设置阈值分类	[86]
FixCache 以及 Rahman 方法	[71]

7.3 性能评价指标

对于数据分析方法的评价，主要有以下几种性能度量：

（1）精度(accuracy)

对于分类任务，精度是分类正确的样本数占样本总数的比例，对于样例集合 D，精度定义为：

$$\mathrm{acc}(f;D)=\frac{1}{m}\sum_{i=1}^{m}\mathrm{II}\left(f(x_i)=y_i\right)\ ,\mathrm{II}\left(X\right)=\begin{cases}1 & X=true\\ 0 & X=false\end{cases}$$

其中，f为预测函数，m 为样本容量，x_i为输入样本，y_i为样本真实标记。

（2）查准率(Precision)、查全率(Recall)

对于分类问题，可将样例根据其真实类别与预测类别的组合划分为真正例(true positive)、假正例(false positive)、真反例(true negative)、假反例(false negative)四种情形，令 TP、FP、TN、FN 分别表示其对应的样例数量，则有：

表 7-3. 预测分类结果

真实情况	预测结果	
	正例	反例
正例	真正例 TP	假反例 FN

反例	假正例 FP	真反例 TN

查准率 P 与查全率 R 的定义为：

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

查准率反映了预测正例中真实正例的比例，查全率反映了真实正例中被正确预测的比例。

8 数据挖掘数据

表 8-1 是通过对文献数据进行归类整理，将数据分为以下几类。

表 8-1：数据驱动软件开发所采用的数据

数据类别	具体数据
开源项目或程序	开源项目、代码库中的代码
工业应用	工业界（大多是企业内部）应用平台代码
文档	需求文档、
日志、审查记录	开发日志、运行日志、审查人员的审查日志
论坛	GitHub、Stack Overflow 等论坛
用户数据	用户主动反馈的数据、用户使用数据
错误报告	软件错误运行报告
其他	已有数据集、传感器数据等

图 8-1(a)给出了各种数据构成的比重，图 8-1(b)-(d)分别给出了在软件设计构造、软件运维演化、软件项目管理三个应用场景下所采用的数据。报告发现文献中超过 60%文献数据来自开源项目和工业应用。

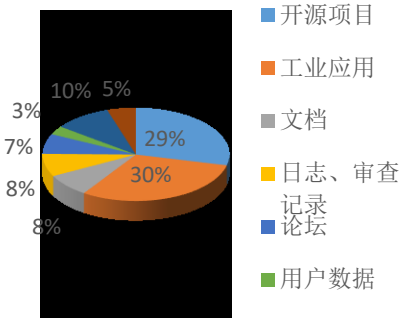


图 8-1(a) 数据驱动软件开发所需数据

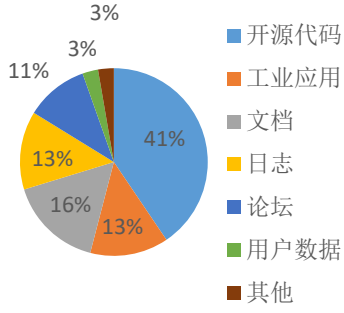


图 8-1(b) 软件设计构造中的数据

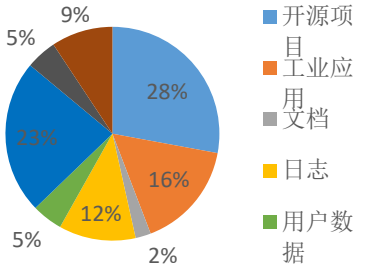


图 8-1(c) 软件运维演化中的数据

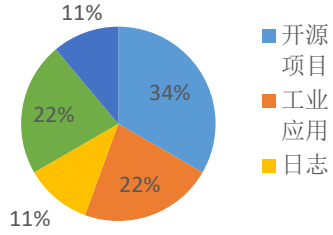


图 8-1d) 软件项目管理中的数据

本报告发现，业界使用数据分析驱动的软件开发中，59%的数据是来自开源项目或工业应用，33%的数据是来自用户或开发过程中的文档、日志等。可以看到，在软件设计构造领域，最主要的分析数据是代码，而在软件运维演化领域，错误报告占据了相当一部分比重。而在软件项目管理中，当涉及到软件改进或者面向开发者的评估推荐时，很多情况下都是从开源项目的源码或者一些工业应用的产品的开发记录中去挖掘需要的信息，比如评估开发者对某一特定缺陷的修复能力，可以通过对他在一些开源项目中提交过的相关代码信息挖掘出他对该缺陷修复的专业程度。所以，在项目管理领域，开源项目和工业应用开发记录这些数据是有很大挖掘空间。

这些数据可以分为两大类：代码数据和文本数据。代码数据包括开源项目和工业应用，文本数据包括需求文档、开发日志、错误日志等。这些数据有些是直接来自软件源代码，这部分数据是比较容易获取的，并且通常符合一定的结构规范（如编码规范、编程语言语法），因此分析过程相对有规可循；而有些是在软件开发过程中由开发人员或用户产生、维护的记录，这些数据相比而言更加复杂，但可能包含的信息价值对软件开发者来说是巨大的。

同时，本报告统计了两种不同类型数据在近三年来的文献中所占比重的变化，发现针对第二类数据的文献比重正在逐年上升，这表明这类数据正在受到开发人员与研究人员日益广泛的关注。

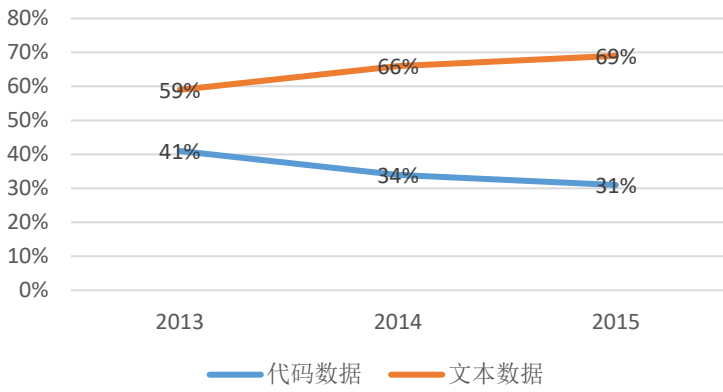


图 8-2 两种主要数据使用比例随时间变化情况

9 数据挖掘工具

本报告收集了文献中进行数据分析任务所使用的工具，以及文献作者所提出的新工具，表 9-1 给出了其中一些数据分析工具及其简介：

表 9-1：数据挖掘工具汇总

工具名称	工具说明
aRules package	挖掘关系和频繁项目集的计算环境
EI Analytics	EI 分析工具(a reporting website 一个分析报告网站）
log explorer	log 记录挖掘工具
the NASA Task Load Index (TLX)	任务负荷测量评估(for assessing cognitive load)
Weka	基于 JAVA 的机器学习工具包
SEMFIX	基于语义的程序修复工具
SAFE WAPI	分析 web API 和 web JavaScript 应用的工具
DLINT	动态检测 JavaScript 代码错误工具
RECRASH	监视可能出现错误的方法
CrashLocator	定位崩溃
LogCluster	聚集日志工具
REMI	API 测试的错误预测
VanarSena	自动 app 找错工具
SPQL	序列规律查询语言
iDice	发现问题工具
learning to log	日志引导工具
AUTOREB	自动评估 app 的系统
FranC	软件维护排名框架
AutoFix	自动 debug 技术
MAP	恶意软件提示处理器
BugFixer	推荐开发者错误报告
Tricorder	代码分析器整合平台
NATURALIZE	一种通用语言框架
KLEE	符号执行引擎
TouchDevelop	基于 web 的实验测试环境
CUDD	决策图工具包
FlashProg	用于文本数据提取的 web 应用
A framework for multimodal data collection, visualization, annotation and learning	基于 ML 的多模态数据处理框架

Eclipse JDT	Java 开发工具

这些数据分析工具可以分为以下几类：

（1）数据分析工具：如 Weka 是一款免费的基于 JAVA 的开源机器学习以及数据挖掘软件。

（2）平台架构工具：如 Tricorder[33]提供了一个代码分析器的集成平台；

（3）开发环境工具：如 TouchDevelop[40]提供了基于微软云服务的移动编程环境；

（4）数据收集工具：如 LogCluster 用于日志的聚集。

这些工具能够帮助开发人员更高效地进行数据分析，一定程度上提升软件开发效率与软件质量。

参考文献

[1] Chockler H, Even K, Yahav E. Finding rare numerical stability errors in concurrent computations[C]//Proceedings of the 2013 International Symposium on Software Testing and Analysis. ACM, 2013: 12-22.

[2] Thompson A L, Bohus D. A framework for multimodal data collection, visualization, annotation and learning[C]//Proceedings of the 15th ACM on International conference on multimodal interaction. ACM, 2013: 67-68.

[3] Meng N, Kim M, McKinley K S. LASE: locating and applying systematic edits by learning from examples[C]//Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013: 502-511.

[4] Nguyen H D T, Qi D, Roychoudhury A, et al. SemFix: program repair via semantic analysis[C]//Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013: 772-781.

[5] Kim S, Zimmermann T, Premraj R, et al. Predicting method crashes with bytecode operations[C]//Proceedings of the 6th India Software Engineering Conference. ACM, 2013: 3-12.

[6] Grönroos C, Ravald A. Service as business logic: implications for value creation and marketing[J]. Journal of Service Management, 2011, 22(1): 5-22.

[7] Claret G, Rajamani S K, Nori A V, et al. Bayesian inference using data flow analysis[C]//Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013: 92-102.

[8] Kumar R, Satyanarayan A, Torres C, et al. Webzeitgeist: design mining the web[C]//Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2013: 3083-3092.

[9] Le V, Gulwani S, Su Z. Smartsynth: Synthesizing smartphone automation scripts from natural language[C]//Proceeding of the 11th annual international conference on Mobile systems, applications, and services. ACM, 2013: 193-206.

[10] Yi J, Chen Y, Li J, et al. Predictive model performance: Offline and online evaluations[C]//Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2013: 1294-1302.

[11] Rigby P C, Bird C. Convergent contemporary software peer review

practices[C]//Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013: 202-212.

[12] Gopinath D, Khurshid S, Saha D, et al. Data-guided repair of selection statements[C]//Proceedings of the 36th International Conference on Software Engineering. ACM, 2014: 243-253.

[13] Bae S G, Cho H, Lim I, et al. SAFEWAPI: Web API misuse detector for web applications[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 507-517.

[14] Miyashita H, Tai H, Amano S. Controlled modeling environment using flexibly-formatted spreadsheets[C]//Proceedings of the 36th International Conference on Software Engineering. ACM, 2014: 978-988.

[15] Krka I, Brun Y, Medvidovic N. Automatic mining of specifications from invocation traces and method invariants[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 178-189.

[16] Allamanis M, Barr E T, Bird C, et al. Learning natural coding conventions[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 281-293.

[17] Gligoric M, Schulte W, Prasad C, et al. Automated migration of build scripts using dynamic analysis and search-based refactoring[C]//ACM SIGPLAN Notices. ACM, 2014, 49(10): 599-616.

[18] Kuchta T, Cadar C, Castro M, et al. Docovery: toward generic automatic document recovery[C]//Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. ACM, 2014: 563-574.

[19] Gao X, Singh M P. Extracting normative relationships from business contracts[C]//Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems. International Foundation for Autonomous Agents and Multiagent Systems, 2014: 101-108.

[20] Wu R, Zhang H, Cheung S C, et al. CrashLocator: locating crashing faults based on crash stacks[C]//Proceedings of the 2014 International Symposium on Software Testing and Analysis. ACM, 2014: 204-214.

[21] Tulsian V, Kanade A, Kumar R, et al. MUX: algorithm selection for software

model checkers[C]//Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 2014: 132-141.

[22] Bird C, Ranganath V P, Zimmermann T, et al. Extrinsic influence factors in software reliability: A study of 200,000 windows machines[C]//Companion Proceedings of the 36th International Conference on Software Engineering. ACM, 2014: 205-214.

[23] Zhao Z, Wu B, Zhou M, et al. Call sequence prediction through probabilistic calling automata[C]//ACM SIGPLAN Notices. ACM, 2014, 49(10): 745-762.

[24] Nguyen T T, Duesterwald E, Klinger T, et al. Characterizing defect trends in software support[C]//Companion Proceedings of the 36th International Conference on Software Engineering. ACM, 2014: 508-511.

[25] Mani S, Sankaranarayanan K, Sinha V S, et al. Panning requirement nuggets in stream of software maintenance tickets[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 678-688.

[26] Sun C, Zhang H, Lou J G, et al. Querying sequential software engineering data[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 700-710.

[27] Miranskyy A, Caglayan B, Bener A, et al. Effect of temporal collaboration network, maintenance activity, and experience on defect exposure[C]//Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, 2014: 27.

[28] Fritz T, Begel A, Müller S C, et al. Using psycho-physiological measures to assess task difficulty in software development[C]//Proceedings of the 36th International Conference on Software Engineering. ACM, 2014: 402-413.

[29] Youn S, Gu C, Kim J. Probabilistic bug localization via statistical inference based on partially observed data[C]//Proceedings of the 51st Annual Design Automation Conference. ACM, 2014: 1-6.

[30] Ravindranath L, Nath S, Padhye J, et al. Automatic and scalable fault detection for mobile applications[C]//Proceedings of the 12th annual international conference on Mobile systems, applications, and services. ACM, 2014: 190-203.

[31] Gulwani S, Marron M. NLyze: Interactive programming by natural language for spreadsheet data analysis and manipulation[C]//Proceedings of the 2014 ACM

SIGMOD international conference on Management of data. ACM, 2014: 803-814.

[32] Vakilian M, Sauciuc R, Morgenthaler J D, et al. Automated decomposition of build targets[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 1: 123-133.

[33] Sadowski C, Van Gogh J, Jaspan C, et al. Tricorder: Building a program analysis ecosystem[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 1: 598-608.

[34] Sadowski C, Stolee K T, Elbaum S. How developers search for code: a case study[C]//Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015: 191-201.

[35] Gong L, Pradel M, Sridharan M, et al. DLint: Dynamically checking bad coding practices in JavaScript[C]//Proceedings of the 2015 International Symposium on Software Testing and Analysis. ACM, 2015: 94-105.

[36] Allamanis M, Barr E T, Bird C, et al. Suggesting accurate method and class names[C]//Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015: 38-49.

[37] Saraiva J, Bird C, Zimmermann T. Products, developers, and milestones: how should I build my N-Gram language model[C]//Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015: 998-1001.

[38] Zhu J, He P, Fu Q, et al. Learning to log: Helping developers make informed logging decisions[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 1: 415-425.

[39] Barnett M, Bird C, Brunet J, et al. Helping developers help themselves: Automatic decomposition of code review changesets[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 1: 134-144.

[40] Protzenko J, Burckhardt S, Moskal M, et al. Implementing real-time collaboration in TouchDevelop using AST merges[C]//Proceedings of the 3rd International Workshop on Mobile Development Lifecycle. ACM, 2015: 25-27.

[41] Mayer M, Soares G, Grechkin M, et al. User interaction models for disambiguation in programming by example[C]//Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology. ACM, 2015: 291-301.

[42] Kim M, Nam J, Yeon J, et al. REMI: defect prediction for efficient API

testing[C]//Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015: 990-993.

[43] Sridhara G, Sinha V S, Mani S. Naturalness of Natural Language Artifacts in Software[C]//Proceedings of the 8th India Software Engineering Conference. ACM, 2015: 156-165.

[44] Theisen C, Herzig K, Morrison P, et al. Approximating attack surfaces with stack traces[C]//Proceedings of the 37th International Conference on Software Engineering-Volume 2. IEEE Press, 2015: 199-208.

[45] Caglayan B, Turhan B, Bener A, et al. Merits of organizational metrics in defect prediction: an industrial replication[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 2: 89-98.

[46] Vinayakarao V, Purandare R, Nori A V. Structurally heterogeneous source code examples from unstructured knowledge sources[C]//Proceedings of the 2015 Workshop on Partial Evaluation and Program Manipulation. ACM, 2015: 21-26.

[47] Vinayakarao V, Purandare R, Nori A V. Structurally heterogeneous source code examples from unstructured knowledge sources[C]//Proceedings of the 2015 Workshop on Partial Evaluation and Program Manipulation. ACM, 2015: 21-26.

[48] Vidales P. Unleashing the True Power of Mobile Systems: Big Data and Analytics[C]//Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems. ACM, 2015: 3-3.

[49] Cheng Y, Iqbal M S, Gupta A, et al. Cast: Tiering storage for data analytics in the cloud[C]//Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing. ACM, 2015: 45-56.

[50] Lv F, Zhang H, Lou J, et al. CodeHow: Effective Code Search Based on API Understanding and Extended Boolean Model (E)[C]//Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on. IEEE, 2015: 260-270.

[51] Allamanis M, Tarlow D, Gordon A D, et al. Bimodal modelling of source code and natural language[C]//Proceedings of the 32th International Conference on Machine Learning. 2015, 951: 2015.

[52] Kong D, Cen L, Jin H. Autoreb: Automatically understanding the review-to-behavior fidelity in android applications[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015: 530-541.

[53] Qing H, Biwen L, Beijun S, et al. Cross-Project Software Defect Prediction Using Feature-Based Transfer Learning[C]//Proceedings of the 7th Asia-Pacific Symposium on Internetware. ACM, 2015: 74-82.

[54] Rotella P, Chulani S, Goyal D. Predicting field reliability[C]//Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM, 2015: 986-989.

[55] Bosu A, Greiler M, Bird C. Characteristics of useful code reviews: an empirical study at Microsoft[C]//Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press, 2015: 146-156.

[56] Zhu J, He P, Fu Q, et al. Learning to log: Helping developers make informed logging decisions[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 1: 415-425.

[57] Ashouri A H, Mariani G, Palermo G, et al. Cobayn: Compiler autotuning framework using bayesian networks[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2016, 13(2): 21.

[58] Rahman M M, Roy C K, Redl J, et al. CORRECT: code reviewer recommendation at GitHub for Vendasta technologies[C]//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ACM, 2016: 792-797.

[59] Baluda M, Pistoia M, Castro P, et al. A framework for automatic anomaly detection in mobile applications[C]//Proceedings of the International Workshop on Mobile Software Engineering and Systems. ACM, 2016: 297-298.

[60] Lin Q, Zhang H, Lou J G, et al. Log clustering based problem identification for online service systems[C]//Proceedings of the 38th International Conference on Software Engineering Companion. ACM, 2016: 102-111.

[61] Chen W, Aspinall D, Gordon A D, et al. More semantics more robust: Improving android malware classifiers[C]//Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks. ACM, 2016: 147-158.

[62] Shimagaki J, Kamei Y, McIntosh S, et al. A study of the quality-impacting practices of modern code review at Sony mobile[C]//Proceedings of the 38th International Conference on Software Engineering Companion. ACM, 2016: 212-221.

[63] Rahman M M, Roy C K, Collins J A. CoRReCT: code reviewer

recommendation in GitHub based on cross-project and technology experience[C]//Proceedings of the 38th International Conference on Software Engineering Companion. ACM, 2016: 222-231.

[64] Karim M R, Alam A, Didar S M, et al. Applying data analytics towards optimized issue management: an industrial case study[C]//Proceedings of the 4th International Workshop on Conducting Empirical Studies in Industry. ACM, 2016: 7-13.

[65] Barik T, DeLine R, Drucker S, et al. The bones of the system: a case study of logging and telemetry at Microsoft[C]//Proceedings of the 38th International Conference on Software Engineering Companion. ACM, 2016: 92-101.

[66] Raghothaman M, Wei Y, Hamadi Y. SWIM: synthesizing what i mean: code search and idiomatic snippet synthesis[C]//Proceedings of the 38th International Conference on Software Engineering. ACM, 2016: 357-367.

[67] Lin Q, Lou J G, Zhang H, et al. iDice: problem identification for emerging issues[C]//Proceedings of the 38th International Conference on Software Engineering. ACM, 2016: 214-224.

[68] Mani S, Padhye R, Sinha V S. Mining API Expertise Profiles with Partial Program Analysis[C]//Proceedings of the 9th India Software Engineering Conference. ACM, 2016: 109-118.

[69] Graus D, Bennett P N, White R W, et al. Analyzing and Predicting Task Reminders[J]. 2016.

[70] Nguyen T T, Nguyen T N, Duesterwald E, et al. Inferring developer expertise through defect analysis[C]//2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 1297-1300.

[71] Lewis C, Lin Z, Sadowski C, et al. Does bug prediction support human developers? findings from a google case study[C]//Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013: 372-381.

[72] Caglayan B, Bener A B, Miranskyy A. Emergence of developer teams in the collaboration network[C]//Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on. IEEE, 2013: 33-40.

[73] Musson R, Richards J, Fisher D, et al. Leveraging the crowd: how 48,000 users helped improve Lync performance[J]. IEEE software, 2013, 30(4): 38-45.

[74] Czerwonka J, Nagappan N, Schulte W, et al. Codemine: Building a software

development data analytics platform at microsoft[J]. IEEE software, 2013, 30(4): 64-71.

[75] Tarvo A, Nagappan N, Zimmermann T. Predicting risk of pre-release code changes with checkinmentor[C]//2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2013: 128-137.

[76] Bettenburg N, Begel A. Deciphering the story of software development through frequent pattern mining[C]//Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013: 1197-1200.

[77] Fu Q, Lou J G, Lin Q, et al. Contextual analysis of program logs for understanding system behaviors[C]//Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, 2013: 397-400.

[78] Mukadam M, Bird C, Rigby P C. Gerrit software code review data from android[C]//Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on. IEEE, 2013: 45-48.

[79] Mukherjee D, Garg M. Which work-item updates need your response?[C]//Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on. IEEE, 2013: 12-21.

[80] Mani S, Nagar S, Mukherjee D, et al. Bug resolution catalysts: Identifying essential non-committers from bug repositories[C]//Proceedings of the 10th Working Conference on Mining Software Repositories. IEEE Press, 2013: 193-202.

[81] Wong C P, Xiong Y, Zhang H, et al. Boosting Bug-Report-Oriented Fault Localization with Segmentation and Stack-Trace Analysis[C]//ICSME. 2014: 181-190.

[82] Hu H, Zhang H, Xuan J, et al. Effective bug triage based on historical bug-fix information[C]//2014 IEEE 25th International Symposium on Software Reliability Engineering. IEEE, 2014: 122-132.

[83] Whitmore J, Türpe S, Triller S, et al. Threat analysis in the software development lifecycle[J]. IBM Journal of Research and Development, 2014, 58(1): 6: 1-6: 13.

[84] Pei Y, Furia C A, Nordio M, et al. Automated fixing of programs with contracts[J]. Ieee transactions on software engineering, 2014, 40(5): 427-449.

[85] Wang X, Dang Y, Zhang L, et al. Predicting Consistency-Maintenance Requirement of Code Clonesat Copy-and-Paste Time[J]. Software Engineering IEEE Transactions on, 2014, 40(8):773-794.

[86] Chaudhari D, Zulkernine M, Weldemariam K. FRanC: A Ranking Framework for the Prioritization of Software Maintenance[C]//Software Security and Reliability-Companion (SERE-C), 2014 IEEE Eighth International Conference on. IEEE, 2014: 31-40.

[87] Wnuk K, Gorschek T, Callele D, et al. Supporting Scope Tracking and Visualization for Very Large-Scale Requirements Engineering-Utilizing FSC+, Decision Patterns, and Atomic Decision Visualizations[J]. IEEE Transactions on Software Engineering, 2016, 42(1): 47-74.

[88] Caglayan B, Turhan B, Bener A, et al. Merits of organizational metrics in defect prediction: an industrial replication[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 2: 89-98.

[89] DeLine R. Research opportunities for the big data era of software engineering[C]//Big Data Software Engineering (BIGDSE), 2015 IEEE/ACM 1st International Workshop on. IEEE, 2015: 26-29.

[90] Greiler M, Herzig K, Czerwonka J. Code ownership and software quality: a replication study[C]//Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press, 2015: 2-12.

[91] Goldstein M, Segall I. Automatic and continuous software architecture validation[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 2: 59-68.

[92] Zhou H, Lou J G, Zhang H, et al. An empirical study on quality issues of production big data platform[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 2: 17-26.

[93] Theisen C, Herzig K, Morrison P, et al. Approximating attack surfaces with stack traces[C]//Proceedings of the 37th International Conference on Software Engineering-Volume 2. IEEE Press, 2015: 199-208.

[94] Gay G, Staats M, Whalen M, et al. Automated oracle data selection support[J]. IEEE Transactions on Software Engineering, 2015, 41(11): 1119-1137.

[95] Zhu J, He P, Fu Q, et al. Learning to log: Helping developers make informed logging decisions[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 1: 415-425.

[96] Zanjani M B, Kagdi H, Bird C. Using developer-interaction trails to triage

change requests[C]//Proceedings of the 12th Working Conference on Mining Software Repositories. IEEE Press, 2015: 88-98.

[97] Zanjani M, Kagdi H, Bird C. Automatically Recommending Peer Reviewers in Modern Code Review[J]. 2015.

[98] Ozsoy M, Khasawneh K N, Donovick C, et al. Hardware-based Malware Detection using Low level Architectural Features[J].

[99] Masuda S, Matsuodani T, Tsuda K. Automatic Generation of UTP Models from Requirements in Natural Language[C]//Software Testing, Verification and Validation Workshops (ICSTW), 2016 IEEE Ninth International Conference on. IEEE, 2016: 1-6.

[100] Mao F, Cai X, Shen B, et al. Operational pattern based code generation for management information system: An industrial case study[C]//Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2016 17th IEEE/ACIS International Conference on. IEEE, 2016: 425-430.

[101] Incremental whole program compilation of code. 2015 Microsoft

[102] Translating natural language descriptions to programs in a domain-specific language for spreadsheets. 2014 Microsoft

[103] System and method for providing visual representations of email to enable efficient email processing. 2013 Google

[104] Dynamic sessionization of analytics data. 2013 Google

[105] Detecting undesirable content. 2013 Google

[106] Know your customer (kyc). 2013 Google

[107] Associating a task with a user based on user selection of a query suggestion. 2013 Google

[108] Clustering geofence-based alerts for mobile devices. 2013 Google

[109] Event-based user behavior timeline, predictions, and recommendations. 2013 Amazon

[110] Image analysis for user authentication. 2015 Amazon

[111] Programmatic event detection and message generation for requests to execute program code. 2014 Amazon

[112] Automatically customizing a computer-executable application at runtime. 2014 Apple