# 一种基于多指标排序的库迁移推荐方法

何昊

2021 年 1 月 19 日

**摘要**

在软件项目中的广泛采用第三方库既有益又有风险。一个已经正在使用的第三方库可能会被其维护者抛弃，可能与许可证不兼容，或者可能不再符合当前项目要求。在这种情况下，开发人员需要将库迁移到具有类似功能的另一个库，但是迁移决策通常基于观点和经验，并且在信息有限的情况下不够理想。因此，已有研究提出了几种基于指标过滤的方法来从现有软件数据中挖掘库迁移，但是它们在准确率和召回率上很难两全，限制了它们在支持迁移决策方面的实用性。

在本文中，我们提出了一种新颖的方法，该方法利用多个指标来排名并因此推荐库迁移。给定要迁移的库，我们的方法会首先从大量的软件库中生成候选目标库，然后通过组合以下四个指标来对它们进行排名，以从开发历史中捕获不同维度的证据：规则支持度，消息支持度，距离支持度，和 API 支持度。为了评估该方法的性能，我们基于以前的工作中从 21,358 个 Java GitHub 项目中恢复了 773 个真实的迁移规则。实验表明，我们的指标可有效地识别出真正的迁移目标，并且我们的方法明显优于现有工作，MRR 为 0.8566，top-1 精度为 0.7947，top-10 NDCG 为 0.7702，top-20 召回率 0.8939。为了证明我们方法的通用性，我们手动验证了 480 个其他广泛使用的库的推荐结果，以相近的性能从中确认了 231 条新的迁移规则。与本文相关的源代码和数据位于：`https://github.com/hehao98/MigrationHelper`。

## 1 引言

现代软件系统在很大程度上依赖于第三方库来提供丰富且易于使用的功能，从而降低了开发成本并提高了生产率 [1,2]。近年来，随着开源软件的兴起和程序包托管平台的出现，例如 GitHub [3]，Maven Central [4] 和 NPM [5]，开源库的数量呈指数增长。例如，Maven Central 中新发布的 JAR 数量在 2010 年为 86,191，在 2015 年为 364,268，在 2019 年则超过 120 万 [4]。如今，开源库可以满足各种各样的开发要求，并已在工业界项目和开源软件项目中广泛采用 [6]。

但是，第三方库会在软件演化过程中引起问题。首先，第三方库可能会出现可持续性方面的失败 [7, 8]. 它们可能会因维护者缺乏时间和兴趣，维护困难或被竞争对手取代而被维护者抛弃。[7]. 其次，第三方库可能具有许可证限制 [9, 10]. 如果项目在早期原型制作过程中使用了 GPL 许可的库，则必须在项目作为专有软件发布之前将其替换。最后，由于缺少特性、性能问题等，第三方库可能无法满足新的项目需求。一个处于初期阶段的 Java 应用程序可能会为了简便而使用 `org.json:json` [11]。当这个应用程序日渐成熟并需要伸缩性的时候，通常必须迁移到另一个 JSON 库以获得更丰富的功能或更高的性能。图 1是其中一个例子。为了解决以上任何一个问题，软件项目必须用一些其他功能类似或等效的库（即**目标库**）替换一些已经使用的库（即**源库**）。此类活动在相关文献中被称为 **库迁移**（Library Migration） [12–19]。

但是，找到好的目标库并在许多候选库之间进行选择通常并不容易 [12, 20, 21]。社区维护的库列表，例如 `awesome-java` 和 AlternativeTo [22] 通常信息量不够，并且包含低质量的库，而

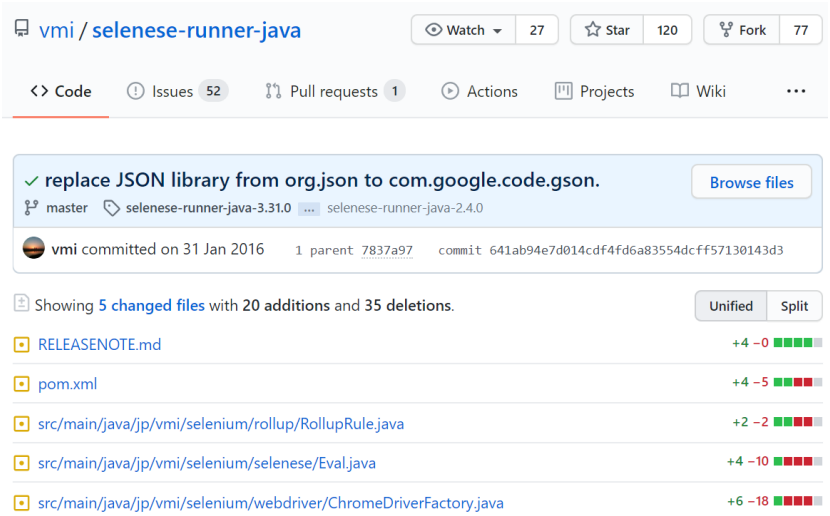



图 1: JSON 库之间的一个真实迁移例子

博客和文章往往仅包含人们的观点 [20]。易于计算的指标（例如，受欢迎程度和发布频率）用途有限，并且在不同的应用领域中指向性不同 [21]。实践中，工业界项目通常依赖领域专家来做出迁移决策 [23]，而开源项目仅在核心开发人员在讨论中达成共识时才迁移库 [15]。无论哪种情况，都不能保证迁移是适当的，具有成本效益的或对项目有利的。

为了应对这些挑战，研究人员提出了从现有软件数据中挖掘库迁移的几种方法 [12,14,17]。其合理性在于，过去发生的迁移实践能够为开发人员做出迁移决策提供有价值的参考和指导，并且如果数据量足够大且质量足够高，则算法可以自动的“众人的智慧”中找到最佳决策。他们的方法首先从大量项目中挖掘候选迁移规则，然后根据频率 [12] 或相关代码更改 [17] 来筛选这些候选规则。但是，由于两种原因，它们的方法要么召回率较低 [12,17]，要么精度不高 [12,14]。首先，他们都定义了全局过滤阈值，但一个全局阈值对所有真实的迁移规则不总有效，因此很难达到性能平衡。其次，他们的过滤指标中没有考虑到一些有价值的信息源。现有方法的实用性也受到限制，因为精度低会导致人工检查结果的工作量较大，而召回率低可能使开发人员可能因为错过迁移机会而无法做出最佳决策。

为了提高这些方法的有效性 [12,14,17]，我们提出了一种从现有软件开发历史中自动推荐库迁移目标的新方法。我们将此问题形式化为*挖掘和排名*问题，而不是挖掘和过滤问题，因为我们注意到相对排名位置不仅更重要，而且对指标和参数更改更加健壮。给定一个源库，我们的方法首先从大量软件开发历史中构建的依赖项变更序列 (在 2.2节中定义) 中挖掘候选目标库。之后，方法根据以下四个精心设计的指标组合对候选库进行*排名*：规则支持度，消息支持度，距离支持度和 API 支持度。这些指标旨在从数据中获取不同的证据来源，并根据证据确定可能的迁移目标。最后，方法返回排序后的最重要的目标库，它们的指标以及相关的迁移实例，以供人工检查。我们的方法是作为 Web 服务实现的，项目维护人员可以使用它来寻求迁移建议或支持其迁移决策。

为了实现我们的方法，我们收集了 21,358 个 Java GitHub 仓库的版本控制数据，并从其依赖项配置文件中成功提取了 147,220 个依赖项变更序列。为了支持指标计算，我们还使用 Libraries.io [24] 从 Maven Central [4] 中收集 Maven 构件，并对每个收集的 Maven 构件收集了其 API 信息。为了评估我们方法的性能，我们从收集的 21,358 个仓库中恢复并扩展了已有研究的迁移规则 [14]，并获得了包含 773 条迁移规则的真实数据集。我们使用 190 个源库作为我们方法的

查询输入，得到了 243,152 条候选规则及其对应指标。这些候选规则中包括上述所有真实迁移规则。然后，我们估算并比较每个指标的生存函数，并为我们的方法和许多其他基准方法计算了平均倒数排名（Mean Reciprocal Rank，MRR）[25]，top-$k$ 精度，top-$k$ 召回率和 top-$k$ 归一化折损累积增益（Normalized Discounted Cumulative Gain，NDCG）[26]。最后，为了确保我们的方法具有通用性并发现更多的真实迁移，我们在另外 480 个流行的库中试验了我们的方法。我们的实验评估表明，这些指标可有效帮助从其他库中识别出真正的迁移目标库，将这四个指标结合在一起的方法可以达到 0.8566 的 MRR，0.7947 的 top-1 精度，0.7702 的 top-10 NDCG 和 0.8939 的 top-20 召回率。此外，我们还可以以相近的性能确认了 661 条新的真实迁移规则。我们的方法已部署在一个工业界内部使用的第三方库管理工具中，用于为黑名单中的库推荐迁移目标。

总而言之，我们在本文中做出了以下贡献：

1. 将库迁移推荐问题形式化为*挖掘*和*排序*问题，
2. 提出了一种新的基于挖掘依赖项变更序列的多指标排序方法，该方法明显优于现有方法，
3. 对 Maven 管理的 Java 项目实现并系统地评估了该方法，表明该方法可有效地对 Maven 管理的 Java 项目提出库迁移建议，
4. 提供了从 1,651 个开源 Java 项目中发现的最新的真实迁移数据集，其中包含 1,384 个迁移规则和 3,340 个相关的代码提交，可用于促进库迁移相关的进一步研究。

# 2 问题背景

在本节中，我们首先简要介绍我们的方法的背景。在那之后，我们定义通用术语以及在本文其余部分中使用的一个库依赖模型。然后，我们给出对库迁移推荐问题的定义。最后，我们讨论 Teyton 等人 [12,14] 和 Alrubaye 等人 [17] 提供的现有方法。

## 2.1 库迁移

迁移是软件维护和演化中的常见现象，可能涉及源于各种动机的不同开发活动。软件开发中常见的迁移案例包括：从旧平台迁移到现代平台 [27,28]，从一种编程语言到另一种编程语言 [29–35]，从一个库版本到另一个库版本 [36,37]，从一个 API 到另一个 API [13,16,38–43]，或从一个库到另一个库 [12,14,15,17,44,45]. 在本文中，我们使用术语**库迁移**来指代将一个库替换为具有相似功能的另一个库的过程，与这些研究 [12–19] 保持一致。

库迁移通常需要两个步骤。第一步是确定要迁移到哪个库以及是否值得进行迁移。第二步是通过修改源代码中的 API 调用，更改配置文件等来进行真正的迁移。Kabinna 等人 [15] 研究了 Apache 软件基金会（Apache Software Foundation，ASF）项目中的日志库迁移。他们发现 ASF 项目进行日志库迁移主要是为了实现上的灵活性、特定的新功能和更好的性能，但由于没有人提交补丁或缺少维护者的共识，某些项目最终还是没有迁移。在当前情况下，我们方法的主要目标是为迁移的第一步提供*可解释的*和*基于证据*的自动化支持，以便开发人员可以根据工具的建议为他们的项目做出最合适的迁移决策结果。如前所述，实现该目标的现有方法存在精度低或召回率低的问题，限制了它们的实用性。

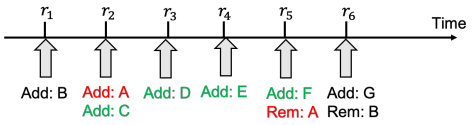



图 2: 一个依赖项变更序列的例子

## 2.2 本文使用的术语和库依赖模型

假设 $\mathcal{P}$ 为待分析的项目集，而 $\mathcal{L}$ 为待分析的库集。项目 $p \in \mathcal{P}$ 具有一组修订（Revision）$Rev(p) = \{r_1, r_2, ..., r_n\}$，其中每个修订具有零个，一个或多个父修订版本。如果 $r_i$ 发生在 $p$ 中的 $r_j$ 之前，我们定义 $r_i < r_j$，反之亦然。对于 $Rev(p)$ 中的每个修订 $r_i$，这个修订中项目会使用一组库 $L_i \subset \mathcal{L}$，我们将其称为 $r_i$ 的**依赖项**。通过与其父修订版本（例如 $r_{i-1}$）进行比较，我们可以如下提取 $r_i$ 的依赖项变更[1]

$$L_i^+ = L_i - L_{i-1} \tag{1}$$

$$L_i^- = L_{i-1} - L_i \tag{2}$$

其中 $L_i^+$ 是已添加库的集合，而 $L_i^-$ 是已删除库的集合。通过按拓扑顺序对所有修订进行排序并汇总依赖项更改，我们可以为项目 $p$ 构建**依赖项变更序列** $D_p$。

$$D_p = L_1^+, L_1^-, L_2^+, L_2^-, ..., L_n^+, L_n^- \tag{3}$$

对于库 $a \in \mathcal{L}$ 和库 $b \in \mathcal{L}$，我们定义 $(a, b)$ 为**迁移规则**，当且仅当 $\exists p \in P$, $r_i, r_j \in Rev(p)$，$p$ 在 $r_i$ 和 $r_j$ 之间进行了从 $a$ 到 $b$ 的迁移，$b \in L_i^+$ 且 $a \in L_j^-$。请注意，迁移既可能只发生在一个修订（$r_i = r_j$）中，也可能会跨越多个修订（$r_i < r_j$）。在随后的论文中，我们将 $a$ 称为**源库**，将 $b$ 称为**目标库**。

图 2显示了一个依赖项变更序列的例子，其中项目在修订 $r_5$ 中执行了从库 $A$ 到 $F$ 的迁移。在这种情况下，根据我们的定义，$L_5^+ = \{F\}$ 和 $L_5^- = \{A\}$，$(A, F)$ 是迁移规则。

## 2.3 库迁移推荐问题定义

令 $R$ 为所有迁移规则的集合。给定项目集合 $\mathcal{P}$、库集合 $\mathcal{L}$、每个项目 $p \in \mathcal{P}$ 的依赖项变更序列集合 $D_p$、以及一组库查询（即待迁移的源库）$Q$，*库迁移推荐问题*的目标是以一种既完整（为了客观性）又准确（为了最小化人工检查工作）的方式识别迁移规则。对于 $a \in Q$，一种方法应当能找到**迁移目标**的集合 $T = \{b | (a, b) \in R\}$。我们将此问题视为两步挖掘排序问题。在挖掘步骤中，一种方法应从所有依赖项变更序列中生成**候选规则**集合 $R_c$。在排序步骤中，对于 $(a, b) \in R_c$，它应计算置信度值 $conf(a, b)$，并按该值对 $R_c$ 进行排序。它应确保排名靠前的候选规则中的目标库更有可能成为真实的迁移目标。

[1]我们将在第 4节中讨论如何处理多个父修订版本的情况。

表 1: 最相关工作的总结与比较 [12,14,17]，更多的性能比较参见第5节

| 方法 | 图2中 $A$ 的候选规则 | 选择迁移规则的方法 | 报告的性能 |
|---|---|---|---|
| Teyton 等人 [12] | $(A,F)$ | $conf_T(a,b) \geq 0.06$ | 以 0.679 的精度在 38,588 个软件仓库中确认了 80 条迁移规则 |
| Teyton 等人 [14] | $(A,C)$, $(A,D)$, $(A,E)$, $(A,F)$ | 所有候选规则 | 以 0.019 的精度在 15,168 个软件仓库中确认了 329 条迁移规则 |
| Alrubaye 等人 [17] | $(A,F)$ | $RS(a,b) = 1 \wedge AC(a,b) > 0$ | 以 1.000 的精度在 16 个软件仓库中确认了 6 条迁移规则 |
| 我们的方法 | $(A,C)$, $(A,D)$, $(A,E)$, $(A,F)$ | 按 $conf(a,b)$ 排序（公式15） | 以 0.795 的精度在 21,358 个软件仓库里确认了 1,384 条迁移规则 |

## 2.4 已有方法

尽管已有几种解决库迁移推荐问题的方法[2]，他们采用了挖掘和过滤的思想。Teyton 等人 [12] 将候选规则定义为同一版本中添加和删除的库的笛卡尔积的集合。更形式化地说，对于项目 $p$，

$$R_{ci}^{p} = \{(a,b) | (a,b) \in L_i^- \times L_i^+, r_i \in Rev(p)\} \tag{4}$$

$$R_c = \bigcup_{p \in \mathcal{P}} \bigcup_{r_i \in Rev(p)} R_{ci}^{p} \tag{5}$$

然后，他们将置信度值定义为规则发生的修订个数除以相同源库或相同目标库的所有规则的最大数目，如下

$$conf_T(a,b) = \frac{|\{r_i | (a,b) \in R_{ci}^{p}\}|}{\max(|\{(a,x) \in R_c\}|, |\{(x,b) \in R_c\}|)} \tag{6}$$

它捕获最常见的规则，并在我们的第一个度量 $RS$（第3.2.1节）中被部分重用。他们的方法需要手动指定阈值 $t$，并且仅当 $conf_T a,b) \geq t$ 时，他们才将候选规则 $(a,b)$ 视为迁移规则。在评估中，通过设置 $t = 0.06$，他们在 38,588 个软件仓库中确认了 80 个迁移规则，精度为 0.679。

在其后续工作 [14] 中，他们还使用以下候选规则定义来考虑跨越多个修订的迁移

$$R_c = \bigcup_{p \in \mathcal{P}} \{(a,b) | (a,b) \in L_j^- \times L_i^+, r_i \leq r_j\} \tag{7}$$

根据这个定义，给定 $A$ 作为源库，图2中的 $(A,C)$，$(A,D)$，$(A,E)$ 和 $(A,F)$ 都将被视为候选规则。为了涵盖尽可能多的迁移规则，他们手动验证了从 15168 个软件仓库生成的 17,113 个候选规则，并在其中成功确认了 329 个迁移规则。

Alrubaye 等人 [17] 提出了一个名为 MigrationMiner 的工具，该工具使用与 [12] 中相同的候选规则定义，但通过其相对频率以及 API 替换发生在代码更改中来过滤候选规则。他们的过滤策略可以在我们的框架中被有效地描述为：规则支持度 $RS(a,b) = 1$ 和 API 计数 $AC(a,b) > 0$（有

[2]注意，同一两位作者在方法粒度也有工作 [13,16]，但我们比较的是他们库粒度的工作 [12,14,17]。对于 [17]，我们在比较期间忽略了其方法粒度的功能。

关定义请参见第3节）。该工具在 16 个软件仓库中进行了测试，以 100%的精度确认 6 个迁移规则。表1概括了现有方法，并与我们的方法进行了比较。

# 3 我们的方法

在本节中，我们将在较高层次上介绍推荐库迁移目标的方法。如前所述，我们使用一个挖掘步骤和一个排名步骤来解决此问题。我们首先描述如何为每个依赖项变更序列挖掘候选规则。然后，我们详细介绍如何设计四个排名指标。最后，我们描述了最终算法的置信度值，排名策略和伪代码。

## 3.1 候选规则挖掘

给定一个库 $a$，我们推荐算法的第一步是找到一组候选规则 $\{(a,x)\}$，其中 $x$ 可能是一个可行的迁移目标。为了实现较大的覆盖范围，我们遵循类似 [14] 的过程来挖掘候选规则 $R_c$（公式7）。由于这么做会产生许多误报，因此在指标计算期间，我们将特别考虑从同一修订（公式4中的 $R^p_{ci}$）中提取的候选规则。我们还为每个候选规则 $(a,b)$ 收集了所有相关的修订对，定义为

$$Rev(a,b) = \{(r_i, r_j) | a \in L_j^- \wedge b \in L_i^+ \wedge r_i \leq r_j\} \tag{8}$$

## 3.2 四种排序指标

### 3.2.1 规则支持度

对于每个候选规则 $(a,b)$，我们将规则计数 $RC(a,b)$ 定义为在同一修订中 $a$ 被删除和 $b$ 被添加的次数：

$$RC(a,b) = |\{r_i | (a,b) \in R^p_{ci}, \forall p \in \mathcal{P}, r_i \in Rev(p)\}| \tag{9}$$

我们将规则支持度 $RS(a,b)$ 定义为：$(a,b)$ 的规则计数除以 $a$ 作为源库的所有候选规则的规则计数的最大值：

$$RS(a,b) = \frac{RC(a,b)}{\max_{(a,x)\in R_c} RC(a,x)} \tag{10}$$

该指标基本上是等式6中 $conf_T(a,b)$ 的重用。因为直觉上，在相同修订版本中经常出现的依赖项更改更可能是库迁移。我们忽略了第二个分母，因为它只能通过一次全量挖掘（$Q = \mathcal{L}$）来计算。考虑到当前开源库的数量，代价是昂贵的。

### 3.2.2 消息支持度

除了以规则支持度为表征的“隐式”频率提示之外，我们还观察到开发人员在修订中留下的消息（例如提交信息或发行说明）所提供的“显性知识”非常有价值。因此，我们设计以下启发式算法来确定修订对 $(r_i, r_j)$ 是否似乎陈述了 $a$ 到 $b$ 的迁移：

1. 首先，我们将 $a$ 和 $b$ 的名称划分为可能有用的部分，因为开发人员经常使用缩写名称在这些消息中提及库。

2. 对于 $r_i = r_j$，我们检查其消息是否说明从 $a$ 迁移到 $b$。

3. 对于 $r_i \neq r_j$，我们检查 $r_i$ 的消息是否在说明引入了库 $b$，以及 $r_j$ 的消息是否在说明库 $a$ 的删除或清理。

我们通过库名称部分的关键字匹配，以及常用于迁移、添加、删除和清除的不同动词来实现检查。例如，我们认为诸如 “migrate”、“replace”、“switch” 之类的词暗示了迁移。我们还使用在5节中发现的真实迁移提交迭代地完善了关键字匹配策略。

令 $h(r_i\ r_j) \mapsto \{0,1\}$ 为上述启发式函数，其中如果消息被我们的启发式算法判定为迁移，则 $h(r_i, r_j) = 1$。我们将消息计数 $MC(a,b)$ 定义为修订对 $(r_i, r_j)$ 的数量，这些修订对的消息证明开发者进行了从 $a$ 到 $b$ 的迁移，在 $r_i$ 中添加了 $b$，并在 $r_j$ 中删除了 $a$：

$$MC(a,b) = |\{(r_i, r_j)|(r_i, r_j) \in Rev(a,b) \wedge h(r_i, r_j)\}| \tag{11}$$

然后我们进一步如下定义消息支持度：

$$MS(a,b) = \log_2(MC(a,b) + 1) \tag{12}$$

### 3.2.3 距离支持度

前两个指标没有考虑没有任何有意义的消息的多版本迁移，但是这种情况仍然很常见，因为并非所有开发人员都编写高质量的消息。由于我们观察到大多数真正的目标库是在删除源库的附近引入的，我们设计了距离支持度，以惩罚距离删除原库很远的地方出现的候选目标库。令 $dis(r_i\ r_j)$ 为修订 $r_i$ 和 $r_j$ 之间的修订数（如果 $r_i = r_j$，则为 0）。我们将距离支持 $DS(a\ b)$ 定义为所有修订对 $(r_i\ r_j) \in Rev(a\ b)$ 的平方距离倒数的平均值：

$$DS(a,b) = \frac{1}{|Rev(a,b)|} \sum_{(r_i,r_j)\in Rev(a,b)} \frac{1}{(dis(r_i, r_j)+1)^2} \tag{13}$$

### 3.2.4 API 支持度

库迁移的另一个重要信息源是在两个库之间执行的实际代码修改和 API 替换。为此，我们将 API 计数 $AC(a,b)$ 定义为 $Rev(a,b)$ 中的所有代码修改块[3]中添加了 $b$ 的 API（即对公共方法和字段的引用），并删除了 $a$ 的 API 的代码修改块个数。然后，我们将 API 支度持定义为：

$$AS(a,b) = \max(0.1, \frac{AC(a,b)}{\max_{(a,x)\in R_c} AC(a,x)}) \tag{14}$$

设置最低阈值的原因是，为了使 $AS$ 在某些迁移规则未检测到 API 更改的情况下更加健壮。发生这种情况的原因可能是代码更改没有与依赖项配置文件更改同步发生，或者因为某些库只需要配置文件就能使用。例如，对 100 个真实的迁移提交（第5.2节）进行的手动分析显示，有 45 个迁移提交不包含任何相关的 API 更改，其中 26 个提交仅修改 `pom.xml` 文件，12 个提交只修改配置文件，而其他提交则使用了一种源库和目标库的 API 没有在同一个代码修改块中被同时添加和删除的方式修改了代码。

## 3.3 推荐迁移目标库

我们的推荐算法结合了以上介绍的四个指标，使用以下简单乘法生成所有候选规则的最终置信度值：

$$conf(a,b) = RS(a,b) \cdot MS(a,b) \cdot DS(a,b) \cdot AS(a,b) \tag{15}$$

[3]代码修改块是由版本控制系统的 diff 算法提取的一组发生在相同位置的添加和删除的代码行。

**Algorithm 1** 迁移目标库推荐算法

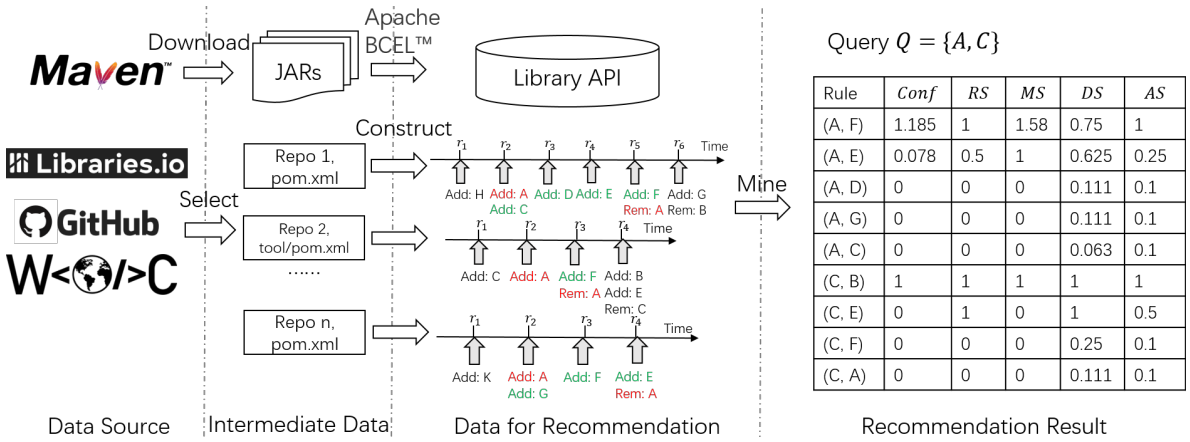
**输入：** 项目集合 $\mathcal{P}$，库集合 $\mathcal{L}$ 和待替换库查询集合 $Q$

**输出：** 对于 $a \in Q$，按 $conf(a,b)$ 排序过的 $R_c$

1: 初始化：$Rev(a,b) \leftarrow \emptyset$
2: **for** $(a,b) \in \mathcal{L} \times \mathcal{L}$ **do**
3: 初始化：$RC(a,b) \leftarrow MC(a,b) \leftarrow DS(a,b) \leftarrow 0$
4: 初始化：$AS(a,b) \leftarrow 0.1$
5: **end for**
6: **for** $p \in \mathcal{P}, a \in Q, b \in R_c^p, (r_i, r_j) \in Rev(a,b)$ **do**
7: $Rev(a,b) \leftarrow Rev(a,b) \cup (r_i, r_j)$
8: **if** $r_i = r_j$ **then**
9: $RC(a,b) \leftarrow RC(a,b) + 1$
10: **end if**
11: $MC(a,b) \leftarrow MC(a,b) + h(r_i, r_j)$
12: $DS(a,b) \leftarrow DS(a,b) + 1/(dis(r_i, r_j) + 1)^2$
13: $AC(a,b) \leftarrow AC(a,b) + \text{getAPICount}(r_i, r_j)$
14: **end for**
15: **for** $(a,b) \in R_c = \bigcup_{p \in \mathcal{P}} R_c^p$ **do**
16: $RS(a,b) \leftarrow RC(a,b) / \max_{(a,x) \in R_c} RC(a,x)$
17: $MS(a,b) \leftarrow \log_2(MC(a,b) + 1)$
18: $DS(a,b) \leftarrow DS(a,b) / |Rev(a,b)|$
19: $AS'(a,b) \leftarrow AC(a,b) / \max_{(a,x) \in R_c} AC(a,x)$
20: **if** $AS'(a,b) > AS(a,b)$ **then**
21: $AS(a,b) \leftarrow AS'(a,b)$
22: **end if**
23: $conf(a,b) = RS(a,b) \cdot MS(a,b) \cdot DS(a,b) \cdot AS(a,b)$
24: **end for**
25: **return** 对 $a \in Q$，按 $conf(a,b)$ 排序过的 $R_c$

对于 $Q$ 中的每个库查询 $l$，方法按照该置信度值从最大到最小对相应的候选规则进行排序。如果某些候选规则的置信度值相同，我们将使用 $RS$ 对其进行进一步排序。最后，将推荐结果以及按每个库查询分组的相关修订对返回，以供人工检查。算法 1 提供了我们的迁移推荐算法的完整说明。在初始化所需的数据结构（第 1-5 行）后，我们将生成候选规则，并在所有项目及其依赖项变更序列（第 6-14 行）的一次迭代中累积必要的数据以进行指标计算。之后，通过对所有候选规则的另一次迭代来计算指标和置信度值（第 15-24 行）。

几个重要的优化可以应用于算法 1 中。首先，第 2-4 行的初始化可以按需进行，以避免过多的内存消耗。其次，对于长度为 $n$ 的依赖项变更序列，可以通过反向遍历序列的同时保留一组已删除的库和相关修订，从而以 $O(n)$ 复杂度高效地完成第 6 行中的迭代。最后，我们可以通过对所有项目的所有修订的一次迭代为 $\mathcal{L} \times \mathcal{L}$ 中的所有 $(a,b)$ 预先计算 $AC(a\ b)$，以避免按需计算 $AC(a,b)$，因为这么做很慢且包含大量重复计算。




| Rule | $Conf$ | $RS$ | $MS$ | $DS$ | $AS$ |
|---|---|---|---|---|---|
| (A, F) | 1.185 | 1 | 1.58 | 0.75 | 1 |
| (A, E) | 0.078 | 0.5 | 1 | 0.625 | 0.25 |
| (A, D) | 0 | 0 | 0 | 0.111 | 0.1 |
| (A, G) | 0 | 0 | 0 | 0.111 | 0.1 |
| (A, C) | 0 | 0 | 0 | 0.063 | 0.1 |
| (C, B) | 1 | 1 | 1 | 1 | 1 |
| (C, E) | 0 | 1 | 0 | 1 | 0.5 |
| (C, F) | 0 | 0 | 0 | 0.25 | 0.1 |
| (C, A) | 0 | 0 | 0 | 0.111 | 0.1 |

图 3: 我们的方法实现概览，其中有一些样例数据和结果

# 4 方法实现

在本节中，我们将详细介绍该方法的重要实现细节，包括设计考量，实现环境和数据收集过程。图 3 概述了我们的实现。

## 4.1 设计考量

由于 Java 的流行度和工业重要性，加之先前的工作 [12, 14, 17] 都针对 Java 实现，因此我们选择对 Java 项目实现我们的方法。我们选择仅对 Maven [46] 托管的项目挖掘依赖项变更序列，因为这些项目可以通过解析 `pom.xml` 文件得到准确的依赖项。每个 `pom.xml` 文件都有一个 `<dependency>` 节，开发人员可以在其中使用组 ID（Group ID）、构件 ID（Artifact ID）和版本号来声明一个项目使用的库。在编译期间，大多数已声明的库都是从 Maven Central [4] 下载的，该网站为大多数 Java 开源库提供了中央托管服务。我们将具有相同组 ID 和构件 ID 的构件（artifact）视为一个库，并将版本字符串视为标记一个库的不同版本。推荐算法目前不会考虑版本号。尽管多个（组 ID，构件 ID）可能引用同一个库，但我们不处理此类别名，因为我们观察到，组 ID 和构件 ID 的重命名通常伴随重要的库更改，例如主要版本更新，有意的依赖项屏蔽等。因此，我们将别名之间的迁移视为有效的迁移规则。但是，我们不考虑与源库具有相同组 ID 的目标库，因为我们注意到它们通常是具有紧密的关系，例如是子组件或辅助库，它们不太可能成为真正的迁移目标。

## 4.2 实现环境

我们在具有 2 个 Intel Xeon E5-2630 v2 CPU，400GB RAM 和 20TB 存储的 Red Hat Linux 服务器上实现我们的方法。这个服务器可以访问 World of Code [47]，一个用于存储开源版本控制数据的数据库，其中包括来自 GitHub 的几乎所有软件仓库。我们使用 World of Code，因为与直接从 GitHub 克隆并使用 `git` 进行分析相比，它在构建依赖项变更序列时它提供了更高的性能。我们使用 Java 收集库和 API，并使用 Python 构造依赖项变更序列。数据是使用 Java 程序和 Python 脚本的构造的，并存储在本地 MongoDB [48] 实例中。最终推荐服务使用 Spring 框架 [49] 在 Java 中实现。

### 4.3 数据收集

#### 4.3.1 获取库的元数据和 API

我们使用 Libraries.io 数据集 [24]（最后更新于 2020 年 1 月）来获取 Maven 构件及其元数据的列表。在 2020 年 10 月从 Maven Central 更新了最新版本信息之后，我们总共获得了 184,817 个不同的库用于挖掘候选规则，并获得了 4,045,748 个不同的库版本用于 API 提取。然后，从 Maven Central 为每个版本下载相应的 JAR 文件（如果有的话）。下载的 JAR 文件的总大小为 ~ 3TB，但在提取 API 后可以将其安全删除。对于每个 JAR 文件，我们使用 Apache Commons BCEL™ 库 [50] 提取所有公共类及其字段和方法，并将每个类作为文档存储在 MongoDB 中。每个文档都描述了一个唯一且紧凑的类签名对象，该对象描述了其字段，方法，继承关系等。它们由从其所有属性计算出的 SHA1 索引。我们还维护库版本和类之间的映射。最后，我们总共获得 25,272,024 个不同的类。在我们的服务器上使用 16 个并行线程，整个下载和提取过程大约需要 3 天。

#### 4.3.2 构造依赖项变更序列

我们使用 Libraries.io 数据集提供的 GitHub 软件仓库列表，在该列表中，我们选择具有至少 10 个星标和一个 `pom.xml` 文件的非 Fork 软件仓库，从而获得了 21,358 个软件仓库。我们设置 10 个星标的阈值以确保收集的软件仓库具有足够的质量。然后，我们从 World of Code 版本 R（最新更新为 2020 年 4 月）中检索和分析软件仓库的提交和 Blob。对于不同的子项目或子模块，软件仓库可能在不同的路径中具有多个 `pom.xml` 文件，因此我们将每个文件视为跟踪一个项目的依赖关系，并选择为每个子项目构造一个依赖项变更序列。对于每个 `pom.xml` 文件，我们将提取修改该文件的所有提交，分别解析其旧版本和新版本，以查看是否添加或删除了任何库。对于合并提交，由于设计和性能限制，我们仅将其与旧版本之一进行比较。对于并行分支，我们按时间对更改进行排序并将它们合并为一个依赖项变更序列。然后，我们清除由合并提交或并行分支引入的重复更改。过滤掉只有一个修订的依赖项变更序列后，我们最终得到 147,220 个不同的依赖项变更序列。

#### 4.3.3 预计算 API 计数

如第3.3节所述，有必要为 $\mathcal{L} \times \mathcal{L}$ 中的每个 $(a.b)$ 预先计算 API 计数表。其原因在于，推荐期间的按需计算不仅耗时，而且效率低下，会针对不同的候选规则进行重复的 Java 文件差异分析。因此，我们在维护一组当前候选规则的同时，遍历每个软件仓库的所有 Java 文件 diff，以查看其中一些是否应增加其 API 计数值。对于每个软件仓库以及不同 Java 文件对的代码分析，都能以高度并行的方式完成整个计算。最后，我们获得了 4,934,677 个具有非零 $AC$ 值的库对。整个计算需要大约两周的时间才能完成（16 个线程），但是我们认为可以对其进行进一步优化，并且只需要运行一次，因为当有新的软件仓库数据进入时，该表可以进行增量更新。表 2 概述了我们使用的最终推荐数据库。

## 5 方法评估

在本节中，我们首先介绍用于评估的两个研究问题。然后，我们描述了如何为不同的评估目的构建两个基本事实数据集。最后，我们给出了每个研究问题的方法和结果。

表 2: 推荐数据库的统计数据

| 数据类型 | 数量/大小 | 构建所需时间 |
|---|---|---|
| GitHub 软件仓库（$\mathcal{P}$） | 21,358 | 几分钟 |
| 带有文件差异的代码提交 | 29,439,998 | 约一天 |
| 分析过的 `pom.xml` 文件 | 10,009,952 | 约一天 |
| 依赖项变更序列（$D_p$） | 147,220 | 几小时 |
| 库（$\mathcal{L}$） | 185,817 | 几分钟 |
| 库的版本 | 4,045,748 | 几小时 |
| Java 类 | 25,272,024 | 约三天 |
| 非零 API 计数 | 4,934,677 | 约两周 |
| 数据库大小（压缩后） | ~100GB | 约三周 |

## 5.1 研究问题

我们的主要目标是评估算法 1 在推荐库迁移目标方面的有效性。由于该算法主要依赖于根据四个指标（等式15）计算出的置信度值，因此我们首先验证每个指标的有效性，从而形成第一个研究问题：

- RQ1：$RS$、$MS$、$DS$ 和 $AS$ 在确定实际迁移目标方面的效果如何？

然后，为了证明我们的方法具有良好的整体性能，我们提出第二个研究问题：

- RQ2：与基准方法和现有方法相比，我们的方法的性能如何？

## 5.2 真实数据集

我们将两个真实数据集用于不同的评估目的。第一个以以前的工作 [14] 为基础，在我们收集的 21,358 个软件仓库中进行恢复得到。这套真实数据集对其中的源库的迁移目标库具有相当好的覆盖范围，因此我们可以计算准确的性能指标并比较不同的方法。第二个是根据我们算法在 21,358 个软件仓库上的推荐结果手动验证的，以 480 个最受欢迎的库作为查询。我们使用该数据集来验证我们的方法可以很好地推广到新数据集上，并以可比的性能确认真实的迁移。在下一节中，我们用 GT2014 指代第一个数据集，用 GT2020 指代第二个数据集。

### 5.2.1 GT2014

我们选择基于 [14] 中的数据集来恢复新的迁移规则数据集，因为自其发表以来，库迁移的情况可能已经发生了重大变化。[14] 中的迁移规则数据集由 329 个“缩写”的库名称对组成，它们可能对应于多个组 ID 和构件 ID。他们通过人工归纳解决了前述库别名问题（4.1部分），但我们认为在收集到的 18 万个库中重复此过程过于昂贵。由于 [14] 中未提供缩写前的 Maven 工件对，因此我们选择将这些规则人工映射回组 ID 和工件 ID。为了覆盖尽可能多的迁移规则，我们将所有组 ID 和工件 ID 的笛卡尔积视为可能规则。我们还对规则进行反向和传递扩展，直到饱和为止（即 $(a,b) \in R \Rightarrow (b,a) \in R$，$(a,b) \in R \wedge (b,c) \in R \Rightarrow (a,c) \in R$），因为直觉上，一个功能类别中的库可以用相同类别中的任何其他库替换。扩展后，我们获得了 3,878 个“可能的”迁移规则。

在第 2.2节中，我们将迁移规则定义为可以在一个项目中确认至少一个迁移的规则。根据此定义，前述收集的规则可能仍包含许多误报。因此，我们收集这些规则的所有相关提交，通过提交消息（commit message）过滤它们，并使用以下方法人工标记它们是否是真正的规则：

1. 如果库 $a$ 和库 $b$ 提供明显不同的功能，我们将 $(a, b)$ 标记为否。
2. 对于其他规则，我们通过应用第3.2.2节中的消息匹配算法来手动检查 “最可能的” 提交。如果存在表明从 $a$ 迁移到 $b$ 的提交消息，我们将 $(a, b)$ 标记为真。
3. 否则，我们将检查易于理解的小提交中的代码变更，当我们发现至少一个可理解的提交进行了从 $a$ 到 $b$ 的迁移时，将 $(a, b)$ 标记为真。

标记过程由具有至少一年 Java 开发经验的三名学生完成。他们被要求在 Google 上搜索任何不熟悉的库，阅读相关网站，并咨询我们社交网络中的 Java 老手，直到他们熟悉这些库为止。为了减轻人工标记带来的有效性威胁，我们采取了保守的标记策略，并确保那位具有 3 年 Java 经验和 1 年行业经验的学生仔细检查了所有带有真标签的规则。我们还手动检查前 20 条推荐结果，并使用相同的标记过程添加所有新的迁移规则。最后，我们获得了 773 个迁移规则的真实数据集，其中包含 190 个不同的源库，这些库分布在来自 1,228 个软件仓库的 2,214 个提交对中。这套迁移规则主要用于性能评估和与其他方法的比较，在本节的其余部分中表示为 $R_t$。

在 RQ1 和 RQ2 中，我们使用 $R_t$ 中的上述 190 个源库作为对我们方法的查询（$|Q| = 190$），它返回 243,152 个候选规则（$|R_c| = 243,152$ ）以及相关指标，提交和置信度值。请注意，候选规则的数量明显大于真实迁移规则的数量（$|R_t| = 773$），因此一个有效的排序算法非常重要。使用预先计算的 API 计数数据，这个推荐只用几分钟就能完成。

### 5.2.2 GT2020

我们根据使用的软件仓库数量选择了最受欢迎的 500 个库。过滤掉 $R_t$ 中的现有源库后，我们得到 480 个库。我们将它们用作我们方法的查询并收集其推荐输出，从而产生 383,218 条候选规则。为确保我们确认的迁移规则对广大受众有价值，并保持合理数量的人工检查工作，我们仅检查符合以下条件的候选规则：

1. 出现在前 20 条推荐结果中，
2. 已添加到多于 10 个软件仓库中，
3. 具有非零置信度值。

过滤之后，我们需要在 12,565 个 `pom.xml` 文件更改，2,353 个提交对和 1,313 个软件仓库中标记 4,418 个候选规则。我们重复如上所述的标记过程，并针对 480 个查询中的 231 个（48.125%），成功地在 1,233 个提交和 785 个软件仓库中确认了 661 个迁移规则。在 RQ2 中，我们还使用 GT2020 验证并比较了我们的方法与 231 个库的现有方法的性能。

通过合并 GT2014 和 GT2020，我们从 14,334 个 `pom.xml` 更改，3,340 个提交对和 1,651 个软件仓库（占 21,358 个软件仓库的 7.73%）中获得了 1,384 个迁移规则。如果我们考虑提交数量排名前 20%的大软件仓库，则在 4,271 个软件仓库中，有 1,092 个（25.57%）至少进行了一次库迁移。这两个百分比均高于 [14] 中报告的百分比（5.57%和 9.95%），这表明自 2014 年以来，Java 开源项目中的库迁移更为普遍。数据集可在我们的网站中下载（请参见摘要）。

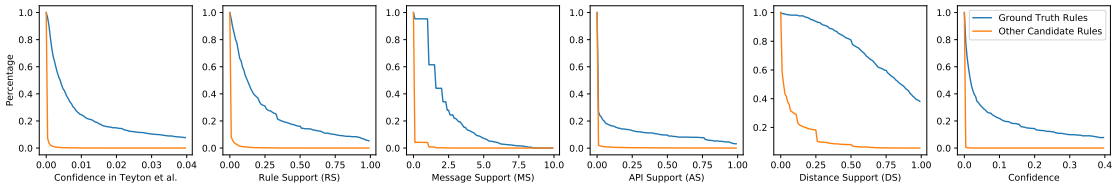



图 4: 真实迁移规则和其他规则的生存函数，这里 $AS$ 并没有被加上 0.1 的阈值

表 3: 使用 GT2014 数据集对不同方法的性能进行比较

| **方法** | **MRR** | **Precision@1** | **NDCG@10** | **Recall@20** |
|---|---|---|---|---|
| Teyton et al. | 0.7133 | 0.6368 | 0.6056 | 0.7257 |
| Teyton et al.' | 0.7335 | 0.6757 | 0.6909 | 0.6391 |
| Teyton et al." | 0.8858 | 0.8737 | 0.8909 | 0.1759 |
| Alrubaye et al. | 0.9412 | 0.9412 | 0.9412 | 0.0540 |
| RS Only | 0.7208 | 0.6474 | 0.6073 | 0.7270 |
| MS Only | 0.7619 | 0.6737 | 0.6619 | 0.7736 |
| RS · MS | 0.8275 | 0.7579 | 0.7436 | 0.8616 |
| RS · MS · DS | 0.8401 | 0.7737 | 0.7589 | 0.8680 |
| RS · MS · AS | 0.8379 | 0.7737 | 0.7479 | 0.8745 |
| 我们的方法 | 0.8566 | 0.7947 | 0.7702 | 0.8939 |

### 5.3 RQ1：$RS$、$MS$、$DS$ 和 $AS$ 在确定实际迁移目标方面的效果如何？

为了回答 RQ1，我们为 [12] 中的置信度、推荐用的四个指标、以及我们方法的最终置信度值绘制了生存函数。对于指标 $M$，其生存函数定义为 $y = P(M \geq x)$，其中 $P()$ 是概率函数。我们使用 GT2014 来估计图 4中真实迁移规则 $R_t$ 和其他候选规则 $R_c - R_t$ 的概率。我们可以在图 4中观察到，真实迁移规则通常对于所有度量都有较高的生存概率，这表明了每个指标的有效性。我们还可以观察到，任何过滤阈值 $x$ 都会导致大量真实迁移规则损失，或大量其他候选规则的生存。因此，尽管每个指标都有效，但是我们得出结论，任何基于过滤的方法本质上都是无效的，我们应该利用相对排名位置而不是绝对指标值来从大量候选对象中选择迁移规则。

### 5.4 RQ2：与基准方法和现有方法相比，我们的方法的性能如何？

对于性能评估，我们使用以下常见的质量指标来评估排名问题：均值倒数排名（MRR）[25]，前 $k$ 精度，前 $k$ 召回率和前 $k$ 归一化折损累积增益（NDCG）[26]。这些指标也已用于评估软件工程中的其他推荐问题 [51–54]。

均值倒数排名（MRR）定义为所有查询的第一个真实迁移规则的排名的倒数均值：

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\min_{(q,x) \in R_t} rank(q,x)} \tag{16}$$

它的范围在 $[0,1]$ 之间，并且较高的值意味着用户可以为每个查询更快地看到第一个真实迁移规则。

假设 $R_k$ 是所有查询中排名前 $k$ 的候选规则，我们将前 $k$ 精度和前 $k$ 召回率定义为：

$$Precision@k = |R_k \cap R_t| \,/\, |R_k| \tag{17}$$

$$Recall@k = |R_k \cap R_t| \,/\, |R_t| \tag{18}$$

归一化折损累积增益（NDCG）衡量排名结果偏离理想结果的程度。对每个 $l \in Q$ 和其返回的前 $k$ 规则 $R_{lk}$，定义 $r(i) = 1$，如果第 $i$ 个结果是真实迁移规则，否则 $r(i) = 0$。通过定义折损累计增益（DCG）为 $DCG_l@k = \sum_{i=1}^{k} \frac{r(i)}{\log_2(i+1)}$，定义理想折损累计增益（IDCG）为 $IDCG_l@k = \sum_{i=1}^{|R_{lk} \cap R_t|} \frac{1}{\log_2(i+1)}$，我们定义前 $k$ 归一化折损累积增益（NDCG@k）为：

$$NDCG@k = \frac{1}{|Q|} \sum_{l \in Q} \frac{DCG_l@k}{IDCG_l@k} \tag{19}$$

范围为 $[0, 1]$，其中 1 表示与理想的前 $k$ 排名结果完美匹配，而 0 表示最差结果。

我们首先使用 GT2014，将我们的方法的性能与以下替代方法进行比较：

- **Teyton et al.** [12]：使用 $conf_T$（公式 6）来排序候选规则。
- **Teyton et al.'** [12]：将 $conf_T < 0.002$ 的规则过滤后，使用 $conf_T$ 排序余下的候选规则。
- **Teyton et al."** [12]：将 $conf_T < 0.015$ 的规则过滤后，使用 $conf_T$ 排序余下的候选规则。
- **Alrubaye et al.** [17]：过滤掉 $RS < 0.6$ 或 $AC = 0$ 的候选规则。
- **RS Only**：只使用 $RS$ 排序。
- **MS Only**：只使用 $MS$ 排序。
- **RS · MS**：使用 $RS \cdot MS$ 排序。
- **RS · MS · DS**：使用 $RS \cdot MS \cdot DS$ 排序。
- **RS · MS · AS**：使用 $RS \cdot MS \cdot AS$ 排序。

我们为 **Teyton et al.** [12] 选择三个阈值以展示其在不同条件下的性能，并 **Teyton et al.** [17] 调整参数以使其性能与论文报告的一致。由于空间限制，我们仅显示 MRR、前 1 精度、前 10 NDCG 和前 20 召回率，其中 MRR 和前 1 精度衡量准确性、前 10 NDCG 衡量理想性、前 20 召回率衡量完整性[4]。

性能比较结果显示在表3中。我们的方法可以在前 20 召回率方面胜过所有替代方案，同时保持高前 1 精度，MRR 和前 10 NDCG。尽管像 **Teyton et al."** 和 **Alrubaye et al.** 那样激进的过滤方法具有较高的精度、NDCG 和 MRR[5]，它们的召回率由于过滤而明显低于其他方法。与其他指标组合的比较还表明，要获得最佳性能，所有四个指标都是必需的。因此，与评估的替代方法相比，我们的方法可获得最佳的总体性能，并且是最适合用于库迁移推荐的方法。

为了证明我们方法的通用性，我们使用 GT2020 对 231 个查询进一步计算了质量指标。由于 GT2020 不完整，因此我们也展示了 **Teyton et al.'** 的性能用于比较。请注意，精度和 MRR 将低于其实际值，且前 20 个召回率是 1，因为我们只检查前 20 个结果。但是，我们仍然可以观察到，我们的方法在表 3中以相似的比例超越了 **Teyton et al.'** 的性能。例如，表 4中，我们的方法的 MRR 比 **Teyton et al.'** 高 13.1%，而表 3中的值比 **Teyton et al.'** 高 16.8%。因此，我们的方法可以在未知数据集中很好地推广。

[4] 可在我们的项目网站上找到更多结果（包括 F 量度）。

[5] 请注意，如果每个查询仅返回一个或很少的项，精度、NDCG 和 MRR 会是非常接近的值。

表 4: 在 GT2020 数据集中我们的方法达到的性能

| 方法 | MRR | Precision@1 | NDCG@10 | Recall@20 |
|---|---|---|---|---|
| Teyton et al.' | 0.6985 | 0.6035 | 0.6653 | 0.8020 |
| 我们的方法 | 0.7902 | 0.6870 | 0.7770 | 1.0000 |

表 5: `c3p0:c3p0` 的推荐结果

| 排名 | 是否正确 | 库 | 置信度 |
|---|---|---|---|
| 1 | 是 | com.mchange:c3p0 | 0.4083 |
| 2 | 是 | com.zaxxer:HikariCP | 0.0124 |
| 3 | 否 | org.jboss.jbossts.jta:narayana-jta | 0.0071 |
| 4 | 否 | org.springframework.boot:spring-boot-starter-test | 0.0050 |
| 5 | 是 | com.alibaba:druid | 0.0039 |
| 6 | 否 | org.jboss.spec.javax.servlet:jboss-servlet-api_3.0_spec | 0.0038 |
| 7 | 是 | org.hibernate:hibernate-c3p0 | 0.0037 |
| 8 | 否 | org.hibernate:hibernate-core | 0.0030 |
| 9 | 否 | org.springframework.boot:spring-boot-starter-web | 0.0029 |
| 10 | 是 | com.alibaba:druid-spring-boot-starter | 0.0026 |
| 11 | 否 | org.modeshape:modeshape-web-explorer | 0.0024 |
| 12 | 否 | org.springframework.boot:spring-boot-devtools | 0.0024 |
| 13 | 否 | org.springframework.boot:spring-boot-starter-thymeleaf | 0.0024 |
| 14 | 否 | org.springframework.boot:spring-boot-starter-aop | 0.0024 |
| 15 | 是 | commons-dbcp:commons-dbcp | 0.0012 |

# 6 推荐的例子

表5给出了我们的结果中的一个推荐示例：`c3p0:c3p0`。它是一个数据库连接池库，自 2007 年版本 `0.9.2` 起已重命名为 `com.mchange:c3p0`。从 2007 年开始，在不同的许可证或特定方案下，出现了许多其他的数据库池竞争者。我们选择 `c3p0:c3p0` 作为示例，不仅因为返回的迁移目标的多样性，而且还因为它很好地证明了我们方法的常见失败和特殊性。从表 5中我们可以看到，推荐的第 1 个迁移目标是 `com.mchange: c3p0`，其置信度比其他目标高得多。第 2 个结果是 Hikari CP，这是自 2013 年以来的数据库连接池，与其他库相比，它具有轻量和性能高的特点。第 5 和第 10 个结果是阿里巴巴 Druid，这是专门为监视目的设计的连接池。第 7 个结果是与 Hibernate 集成使用的 `c3p0`。第 15 个结果是 Apache Commons DBCP，这是 ASF 许可证下的连接池库。考虑到从 `c3p0:c3p0` 进行迁移的普遍性，开发人员应考虑将其放弃用于自己的项目，并在通过调查迁移提交和其他可用信息后，做出对其项目最合适的库迁移决策。

# 7 方法局限

## 7.1 失败情况分析

**冷启动**。我们方法中最明显的失败情况是，它只能推荐在已有数据中经常迁移的迁移目标。但是，可以通过使用海量项目数据库来减轻此局限，因为迁移的价值越大，数据库中的迁移就越常见。如 [15] 中所示，在 223 个 ASF 项目中，有 33 个经历了至少一次日志库迁移，主要是从普通的日志记录库（例如 `log4j`）转到日志抽象库（例如 `slf4j`）和日志统一库（例如 `logback`），我们的方法在这些流行的迁移中表现良好。开发人员还可以得出结论，即使某库提供的功能相似，但是如果我们的结果中没有这个库，那么它也不是一个值得迁移的目标。

**数据稀疏性**。其他失败情况通常是由我们数据中许多实际迁移的稀疏性引起的。如果对于一个查询只能识别一个或几个相关的迁移提交，则这些提交中的其他添加的库往往会作为误报与真正的迁移目标一起出现（例如，表5中的 `hibernate-core`）。$MS$ 和 $AS$ 可以处理这种情况，但是它们有时会失败，或者是因为开发人员没有编写信息丰富的提交消息，或者是因为迁移不需要进行代码更改，或者因为代码更改是在 `pom.xml` 之前/之后完成的。

**杂项**。如果一个库根本没有替代品，那么我们的方法将会失败，并且在这种情况下它不会发出任何警告。我们匹配提交消息的方法有时会错误地识别迁移，并且可以通过更复杂的 NLP 技术进行改进。我们的方法还经常在结果中返回 BOM（Bill of Materials），封装其他库的库或像 Spring 这样的 “完整解决方案” 框架。即使在标记过程中不将它们视为迁移目标，它们也可能对开发人员有用。

## 7.2 对有效性的威胁

### 7.2.1 建构效度

本文的方法主要是为了在提出的问题上实现高性能而设计的，可能不能很好地指示特定项目的特定迁移的适当性或最佳性。为了设计更好的推荐方法，需要进一步研究以了解驱动库迁移的因素。即使在性能方面，我们也不能保证设计的指标和选择的参数是最佳的。为了避免在单个数据集上过度拟合，我们为每个指标选择最简单的形式，然后在其他数据集上评估我们的方法。对评估而言，通常不可能在真实数据中计算所有可能的迁移目标的准确召回率 [12,14]。即使我们将评估限制在一部分库中，我们也不能保证所确定的迁移目标的完整性。为了缓解这种威胁，我们激进地从现有工作 [14] 扩展了最大的迁移规则集，并验证了所有扩展规则，以 “最大努力” 估算 RQ2 中 GT2014 的召回率。

### 7.2.2 外部效度

我们的方法可能不会推广到其他数据集（例如工业界项目）以及其他编程语言和库生态系统。我们通过收集大量开源 Java 项目和库并在许多库查询上进行评估来减轻这种威胁。同样，我们在第 3节中的方法没有做出任何特定于语言的假设，并且可以很容易地针对其他编程语言重新实现。其他人也可以参考第4节作为示例，参考如何为特定语言或库生态系统做出实现选择。

# 8 相关工作

在本节中，我们将总结之前未讨论过的相关工作，并陈述与之相比的关系，差异或改进。

Bartolomei 等人 [44, 45] 研究了使用 API 封装在两个 XML 库之间和在两个 Java GUI 库之间进行迁移的可行性、特殊性和设计模式。他们总结了封装库设计的特殊性，并提出了封装库的一组设计模式。但是，对于所有可能的迁移，实施封装通常很昂贵，而且为所有可能的库迁移实现封装是几乎不可能的。

为了向开发人员推荐库，Thung 等人 [51] 建议通过关联规则挖掘和协作过滤，基于项目已使用的库来推荐库。稍后研究人员针对此问题提出了更多方法，例如多目标优化 [54]、分层聚类 [55]、高级协作过滤 [56]、矩阵分解 [57] 等。但是，它们的主要目的是根据项目正在使用的库和项目的属性来建议错过的重用机会，而不是建议已经在使用的库的迁移机会。Chen 等人 [20] 通过训练在 Stack Overflow 标签上嵌入的单词来挖掘语义相似的库。给定一个库，他们的方法可以返回可能相似的库的列表，但是它不能提供有关该库与返回的库之间进行迁移的可行性和普遍性的证据。

Mora 等人 [21] 研究了若干通用指标在开发人员选择库中的作用。Alrubaye 等人 [18] 分析了库迁移前后的几种代码质量指标。他们都使用现有指标进行实证分析，而我们专门设计了新指标以准确挖掘和建议从现有软件数据进行迁移。

许多研究提出了用于挖掘两个相似库的 API 映射的方法 [13, 16, 19, 40, 43, 58] 或直接编辑代码以使用新的 API [59, 60]。Zheng 等人 [58] 从开发人员在论坛或博客文章中分享的知识中挖掘替代 API。Gokhale 等人 [40] 通过获取相似 API 的执行跟踪来构建映射。Teyton 等人 [13] 通过分析现有迁移中它们的同时出现频率来找到 API 映射。Alrubaye 等人使用信息检索技术 [16] 和机器学习模型 [19] 改进他们的方法。Chen 等人 [43] 提出了一种基于无监督深度学习的方法，该方法同时使用了 API 使用模式和 API 描述的嵌入向量。Xu 等人 [59] 提出了一种从现有项目中推断和应用迁移编辑模式的方法。Collie 等人 [60] 建议在没有先验知识的情况下对库 API 行为进行建模和综合，该行为可用于识别迁移映射和应用迁移更改。我们的方法的输出可以用作上述任何方法的输入，因为它们都需要手动指定两个库。我们新的迁移数据集也可能有助于促进对该领域的进一步研究。

最近的其他工作也旨在帮助依赖管理，不过是从其他不同的角度，例如表征库使用情况 [6]、版本管理行为 [61–63] 和升级行为 [37]、探索库选型的注意事项 [21, 23]、解决依赖树中的版本冲突 [64, 65]、多模块项目中的版本冲突 [66]，等等。

# 9 结论和未来工作

在本文中，我们提出了一种方法，该方法通过挖掘大量软件项目的依赖项变更序列来自动推荐库迁移目标。我们将此问题定义为挖掘和排名问题，并设计了四个排序指标。据我们所知，我们的方法在此问题领域中获得了最佳性能。我们还验证并建立了一个最新的迁移数据集以进行进一步研究。

将来，我们计划改进我们的方法以克服当前的局限性（第7节）。更重要的是，我们计划在更复杂的工业界环境中部署我们的工具，并收集开发者的使用反馈，同时以众包方式扩展迁移规则数据集。最后，我们计划使用收集的迁移数据集，系统地调查影响库迁移发生的因素。

# 参考文献

[1] W. C. Lim, “Effects of reuse on quality, productivity, and economics,” *IEEE Software*, vol. 11, no. 5, pp. 23–30, 1994.

[2] P. Mohagheghi and R. Conradi, "Quality, productivity and economic benefits of software reuse: a review of industrial studies," *Empirical Software Engineering*, vol. 12, no. 5, pp. 471–516, 2007.

[3] Github. [Online]. Available: https://github.com/

[4] Maven central repository. [Online]. Available: https://mvnrepository.com/repos/central

[5] Npm. [Online]. Available: https://www.npmjs.com/

[6] Y. Wang, B. Chen, K. Huang, B. Shi, C. Xu, X. Peng, Y. Liu, and Y. Wu, "An empirical study of usages, updates and risks of third-party libraries in java projects," *The 36th IEEE International Conference on Software Maintenance and Evolution (ICSME 2020)*, 2020.

[7] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, 2017, pp. 186–196.

[8] M. Valiev, B. Vasilescu, and J. D. Herbsleb, "Ecosystem-level determinants of sustained activity in open-source projects: a case study of the pypi ecosystem," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, 2018, pp. 644–655.

[9] D. German and M. Di Penta, "A method for open source license compliance of java applications," *IEEE Software*, vol. 29, no. 3, pp. 58–63, 2012.

[10] S. Van Der Burg, E. Dolstra, S. McIntosh, J. Davies, D. M. German, and A. Hemel, "Tracing software build processes to uncover license compliance inconsistencies," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, 2014, pp. 731–742.

[11] Maven repository: org.json:json. [Online]. Available: https://mvnrepository.com/artifact/org.json/json

[12] C. Teyton, J. Falleri, and X. Blanc, "Mining library migration graphs," in *19th Working Conference on Reverse Engineering, WCRE 2012, Kingston, ON, Canada, October 15-18, 2012*, 2012, pp. 289–298.

[13] ——, "Automatic discovery of function mappings between similar libraries," in *20th Working Conference on Reverse Engineering, WCRE 2013, Koblenz, Germany, October 14-17, 2013*, 2013, pp. 192–201.

[14] C. Teyton, J. Falleri, M. Palyart, and X. Blanc, "A study of library migrations in java," *Journal of Software: Evolution and Process*, vol. 26, no. 11, pp. 1030–1052, 2014.

[15] S. Kabinna, C. Bezemer, W. Shang, and A. E. Hassan, "Logging library migrations: a case study for the apache software foundation projects," in *Proceedings of the 13th International*

*Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, 2016, pp. 154–164.

[16] H. Alrubaye, M. W. Mkaouer, and A. Ouni, "On the use of information retrieval to automate the detection of third-party java library migration at the method level," in *Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019, Montreal, QC, Canada, May 25-31, 2019*, 2019, pp. 347–357.

[17] ——, "Migrationminer: An automated detection tool of third-party java library migration at the method level," in *2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019*. IEEE, 2019, pp. 414–417.

[18] H. Alrubaye, "How does API migration impact software quality and comprehension? an empirical study," *CoRR*, vol. abs/1907.07724, 2019.

[19] H. Alrubaye, M. W. Mkaouer, I. Khokhlov, L. Reznik, A. Ouni, and J. Mcgoff, "Learning to recommend third-party library migration opportunities at the API level," *Appl. Soft Comput.*, vol. 90, p. 106140, 2020.

[20] C. Chen, S. Gao, and Z. Xing, "Mining analogical libraries in q&a discussions - incorporating relational and categorical knowledge into word embedding," in *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*. IEEE Computer Society, 2016, pp. 338–348.

[21] F. L. de la Mora and S. Nadi, "An empirical study of metric-based comparisons of software libraries," in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2018, Oulu, Finland, October 10, 2018*, 2018, pp. 22–31.

[22] Alternativeto: Crowd-sourced software recommendations. [Online]. Available: https://alternativeto.net/

[23] E. Larios-Vargas, M. Aniche, C. Treude, M. Bruntink, and G. Gousios, "Selecting third-party libraries: The practitioners' perspective," *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020)*, 2020.

[24] Libraries.io open data. [Online]. Available: https://libraries.io/data

[25] N. Craswell, *Mean Reciprocal Rank*. Boston, MA: Springer US, 2009, pp. 1703–1703. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_488

[26] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, 2002.

[27] F. Fleurey, E. Breton, B. Baudry, A. Nicolas, and J. Jézéquel, "Model-driven engineering for software migration in a large industrial context," in *Model Driven Engineering Languages*

*and Systems, 10th International Conference, MoDELS 2007, Nashville, USA, September 30 - October 5, 2007, Proceedings*, ser. Lecture Notes in Computer Science, G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, Eds., vol. 4735. Springer, 2007, pp. 482–497.

[28] B. Verhaeghe, A. Etien, N. Anquetil, A. Seriai, L. Deruelle, S. Ducasse, and M. Derras, "GUI migration using MDE from GWT to angular 6: An industrial case," in *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019*, X. Wang, D. Lo, and E. Shihab, Eds. IEEE, 2019, pp. 579–583.

[29] A. T. Nguyen, T. T. Nguyen, and T. N. Nguyen, "Lexical statistical machine translation for language migration," in *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, 2013, pp. 651–654.

[30] A. T. Nguyen, H. A. Nguyen, T. T. Nguyen, and T. N. Nguyen, "Statistical learning approach for mining API usage mappings for code migration," in *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, 2014, pp. 457–468.

[31] A. T. Nguyen, T. T. Nguyen, and T. N. Nguyen, "Divide-and-conquer approach for multi-phase statistical migration for source code (T)," in *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, 2015, pp. 585–596.

[32] T. D. Nguyen, A. T. Nguyen, and T. N. Nguyen, "Mapping API elements for code migration with vector representations," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, 2016, pp. 756–758.

[33] A. T. Nguyen, Z. Tu, and T. N. Nguyen, "Do contexts help in phrase-based, statistical source code migration?" in *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*, 2016, pp. 155–165.

[34] B. Dorninger, M. Moser, and J. Pichler, "Multi-language re-documentation to support a COBOL to java migration project," in *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, M. Pinzger, G. Bavota, and A. Marcus, Eds. IEEE Computer Society, 2017, pp. 536–540.

[35] N. D. Q. Bui, Y. Yu, and L. Jiang, "SAR: learning cross-language API mappings with little knowledge," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, M. Dumas, D. Pfahl, S. Apel, and A. Russo, Eds. ACM, 2019, pp. 796–806.

[36] B. Cossette and R. J. Walker, "Seeking the ground truth: a retroactive study on the evolution and migration of software libraries," in *20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE'12, Cary, NC, USA - November 11 - 16, 2012*, 2012, p. 55.

[37] R. G. Kula, D. M. Germán, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies? - an empirical study on the impact of security advisories on library migration," *Empirical Software Engineering*, vol. 23, no. 1, pp. 384–417, 2018.

[38] M. Nita and D. Notkin, "Using twinning to adapt programs to alternative apis," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, Eds. ACM, 2010, pp. 205–214.

[39] P. Kapur, B. Cossette, and R. J. Walker, "Refactoring references for library migration," in *Proceedings of the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA*, 2010, pp. 726–738.

[40] A. Gokhale, V. Ganapathy, and Y. Padmanaban, "Inferring likely mappings between apis," in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, D. Notkin, B. H. C. Cheng, and K. Pohl, Eds. IEEE Computer Society, 2013, pp. 82–91.

[41] A. Santhiar, O. Pandita, and A. Kanade, "Mining unit tests for discovery and migration of math apis," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 1, pp. 4:1–4:33, 2014.

[42] A. C. Hora and M. T. Valente, "Apiwave: Keeping track of API popularity and migration," in *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, R. Koschke, J. Krinke, and M. P. Robillard, Eds. IEEE Computer Society, 2015, pp. 321–323.

[43] C. Chen, Z. Xing, Y. Liu, and K. L. X. Ong, "Mining likely analogical apis across third-party libraries via large-scale unsupervised api semantics embedding," *IEEE Transactions on Software Engineering*, 2019.

[44] T. T. Bartolomei, K. Czarnecki, R. Lämmel, and T. van der Storm, "Study of an API migration for two XML apis," in *Software Language Engineering, Second International Conference, SLE 2009, Denver, CO, USA, October 5-6, 2009, Revised Selected Papers*, 2009, pp. 42–61.

[45] T. T. Bartolomei, K. Czarnecki, and R. Lämmel, "Swing to SWT and back: Patterns for API migration by wrapping," in *26th IEEE International Conference on Software Maintenance (ICSM 2010), September 12-18, 2010, Timisoara, Romania*. IEEE Computer Society, 2010, pp. 1–10.

[46] Apache maven project. [Online]. Available: http://maven.apache.org/

[47] Y. Ma, C. Bogart, S. Amreen, R. Zaretzki, and A. Mockus, "World of code: an infrastructure for mining the universe of open source VCS data," in *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*, 2019, pp. 143–154.

[48] Mongodb: The database for modern applications. [Online]. Available: https://www.mongodb.com/

[49] Spring home. [Online]. Available: https://spring.io/

[50] Apache commons bcel™ - byte code engineering library. [Online]. Available: https://commons.apache.org/proper/commons-bcel/

[51] F. Thung, D. Lo, and J. L. Lawall, "Automated library recommendation," in *20th Working Conference on Reverse Engineering, WCRE 2013, Koblenz, Germany, October 14-17, 2013*, 2013, pp. 182–191.

[52] X. Ye, R. Bunescu, and C. Liu, "Learning to rank relevant files for bug reports using domain knowledge," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 689–699.

[53] H. Niu, I. Keivanloo, and Y. Zou, "Api usage pattern recommendation for software development," *Journal of Systems and Software*, vol. 129, pp. 127–139, 2017.

[54] A. Ouni, R. G. Kula, M. Kessentini, T. Ishio, D. M. Germán, and K. Inoue, "Search-based software library recommendation using multi-objective optimization," *Information & Software Technology*, vol. 83, pp. 55–75, 2017.

[55] M. A. Saied, A. Ouni, H. Sahraoui, R. G. Kula, K. Inoue, and D. Lo, "Improving reusability of software libraries through usage pattern mining," *Journal of Systems and Software*, vol. 145, pp. 164–179, 2018.

[56] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, and M. Di Penta, "Crossrec: Supporting software developers by recommending third-party libraries," *Journal of Systems and Software*, vol. 161, p. 110460, 2020.

[57] Q. He, B. Li, F. Chen, J. Grundy, X. Xia, and Y. Yang, "Diversified third-party library prediction for mobile app development," *IEEE Transactions on Software Engineering*, 2020.

[58] W. Zheng, Q. Zhang, and M. R. Lyu, "Cross-library API recommendation using web search engines," in *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*, T. Gyimóthy and A. Zeller, Eds. ACM, 2011, pp. 480–483.

[59] S. Xu, Z. Dong, and N. Meng, "Meditor: inference and application of api migration edits," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 335–346.

[60] B. Collie, P. Ginsbach, J. Woodruff, A. Rajan, and M. O'Boyle, "M3: Semantic api migrations," *The 35th IEEE/ACM International Conference on Automated Software Engineering (ASE 2020)*, 2020.

[61] A. Decan and T. Mens, "What do package dependencies tell us about semantic versioning?" *IEEE Transactions on Software Engineering*, 2019.

[62] J. Dietrich, D. J. Pearce, J. Stringer, A. Tahir, and K. Blincoe, "Dependency versioning in the wild," in *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*, 2019, pp. 349–359.

[63] C. Soto-Valero, A. Benelallam, N. Harrand, O. Barais, and B. Baudry, "The emergence of software diversity in maven central," in *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*, 2019, pp. 333–343.

[64] Y. Wang, M. Wen, Z. Liu, R. Wu, R. Wang, B. Yang, H. Yu, Z. Zhu, and S. Cheung, "Do the dependency conflicts in my project matter?" in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, 2018, pp. 319–330.

[65] Y. Wang, M. Wen, Y. Liu, Y. Wang, Z. Li, C. Wang, H. Yu, S.-C. Cheung, C. Xu, and Z. Zhu, "Watchman: Monitoring dependency conflicts for python library ecosystem," *The 42nd International Conference on Software Engineering (ICSE 2020)*, 2020.

[66] K. Huang, B. Chen, B. Shi, Y. Wang, C. Xu, and X. Peng, "Interactive, effort-aware library version harmonization," *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020)*, 2020.