基于开发者协作关系和信息检索的需求跟踪系统①

李姣阳[1,2], 李 娟[1], 杨 达[1]

[1](中国科学院软件研究所 基础软件国家工程研究中心, 北京 100190)
[2](中国科学院大学, 北京 100190)

摘 要: 目前自动化需求跟踪的研究广泛使用文本信息检索技术. 然而信息检索会遗漏一些文本不相似但是实际相关的软件制品, 导致自动化跟踪的精度不高. 针对上述问题, 提出利用开发者协作关系来进行优化, 研发了基于开发者协作关系和信息检索的需求跟踪系统. 该系统在进行需求跟踪时, 首先用信息检索推荐与需求文本上相似的代码, 然后从代码提交日志中挖掘开发者协作关系, 根据开发者协作关系再推荐相关代码, 用户根据两次推荐的结果确定正确的需求代码跟踪关系. 试验结果表明该系统能够找到信息检索遗漏的需求跟踪关系, 能够提高自动化跟踪的准确性, 节省跟踪时间.

关键词: 需求跟踪; 软件需求; 开发者协作关系; 代码提交日志; 信息检索

Requirement-to-Code Traceability System Based on Developer Collaboration Relationship and Information Retrieval

LI Jiao-Yang[1,2], LI Juan[1], YANG Da[1]

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Science, Beijing 100190, China)

Abstract: Information retrieval (IR) is widely used in automatically discovering requirement traceability. However IR will miss some correct artifacts which have low text similarity with the requirement. There are accuracy issues in requirement traceability based on IR. To solve the problem, we propose an approach of using the developer collaborative relationship to improve the accuracy of the traceability links recovery between requirement and source code. Meanwhile, we develop a requirement-to-code traceability system. When the system is tracing, it retrieves the source code artifacts of the highest text similarity with the requirement and extracts the developer collaborative relationship from code commit logs. Then the system recommends some relevant code artifacts by developer collaboration relationship. Users can choose the correct code artifacts from the recommend result. The experiment shows that the requirement traceability system could improve the accuracy and the efficiency and reduce errors.

Key words: requirement traceability; software requirements; developer collaboration relationship; code commit log; information retrieval

1 引言

软件制品之间的跟踪关系对于软件开发和管理有很重要的作用, 跟踪关系可以将高层面的软件制品映射到低层面的软件制品, 从而能够进行有效的软件制品管理[1,2]. 高层面的软件制品主要指需求文档, 低层面的软件制品主要指代码. 需求跟踪的过程可以被看成是文档检索的过程, 将源制品(例如, 需求)视为“查询”, 将其他软件制品(例如, 源代码)视为文档空间, 可以通过信息检索工具来检索到与查询相关的文档[2-4]. 需求跟踪大多采用 IR(信息检索, Information Retrieval)技术或在 IR 基础上进行改进[2-9]. 当把需求和代码都看成静态文本时, 可以利用 IR 模型计算需

① 基金项目:国家自然科学基金(91218302,61073044,61003028,71101138,61100071);北京市自然科学基金(4122087);核心电子器件、高端通用芯片及基础软件产品项目(2012ZX01039-004)

收稿时间:2014-02-25;收到修改稿时间:2014-04-02

求文本与代码文本之间的相似度, 按照文本相似度的大小对检索结果进行排列, 通过设定阈值来筛选得到需求所对应的实现代码[10].

使用 IR 技术可以自动建立跟踪关系, 在一定程度上能有效解决维护困难、易出错、成本高等问题[11]. 但是基于 IR 技术的需求跟踪普遍存在精确度问题: 如果需求与代码的文本相似度低, 则即使它们之间是相互关联的, 却不能通过 IR 检索到, 导致需求跟踪的遗漏[11]. 总而言之, 所有基于 IR 的技术只捕捉了软件制品间关联关系的文本相似性, 会导致精确度低的问题.

为了解决上述问题, 本文提出在文本信息检索基础上利用开发者协作关系, 帮助用户发现更多正确的需求跟踪关系. 本文主要研究需求与代码之间的跟踪关系. 在软件开发和维护过程中, 开发人员是协作开发的. 如果开发人员 R1 和 R2 均开发或者修改过代码 C1, 同时他们也开发或者修改过代码 C2, 那么 C1 和 C2 可能实现同一条需求. 即如果 C1 与需求 R 相关, 那么 C2 也可能与 R 相关. 为了验证本文提出的假设, 本文设计和实现了一个基于开发者协作关系和信息检索的需求跟踪系统. 从代码提交信息中发现开发者协作关系, 对代码关联关系建模, 构建代码关联网络, 利用代码关联关系进行需求跟踪关系的推荐, 调整需求跟踪检索结果. 对需求跟踪系统的检索结果进行评估, 发现本文利用开发者协作关系与纯粹基于 IR 技术的需求跟踪比较, 需求跟踪的准确率和召回率有明显的提升.

2 相关工作

2.1 基于文本信息检索的需求跟踪

使用 IR 技术进行需求跟踪研究的核心思想是: 软件制品之间的文本相似性越高, 两者之间相互关联的可能性就越大[2]. Antoniol 等人使用概率模型和向量空间模型来跟踪源代码到软件文档[3]. Marcus 和 Maletic 使用潜在语义检索的方法来建立源代码和文档之间的跟踪关系[4]. Abadi 等人比较了维度约减方法(LSI)、概率模型和信息理论方法(JS), 以及标准的向量空间模型(VSM)等 IR 方法, 这些方法中实验结果最好的是 VSM 和 JS[7]. 最近, 有人提出使用 Latent Dirichlet Allocation(LDA)和相关主题模型(LDA 的扩展)与其他 IR 方法的组合来进行建立需求跟踪[11,15]. 有一些研究者致力于如何改进 IR 模型、提升软件制品的文本描述质量、改进软件制品标引词抽取、改进预处理技术、对 IR 模型进行组合, 有一些改进致力于从软件制品中抽取出标引词以及预处理技术[10,12,13]. 近年来, 也有研究者从软件开发中寻找其他信息来补充 IR 方法遗漏的跟踪联系. McMilla 等人使用软件源代码结构化信息来增强基于 IR 的跟踪方法[6,14,15]. Panichella 等人提出了只在 IR 方法获得的跟踪关系经过软件工程师确认以及被认定是正确的情况下, 才使用结构化信息[15].

2.2 基于开发者协作关系的相关研究

在软件工程其他领域, 有研究者利用开发者之间的关系进行其他方面的软件工程研究, 例如代码的缺陷预测[10,15-18]. 在这些代码缺陷相关研究表明, 开发者的协作和交流的关系与代码质量有很大关联. 开发者的交流间接表现了开发者的协作, 开发者共同修改代码是直接的协作关系[15].

Diaz 等人使用开发者对代码的拥有关系来改进基于信息检索的需求跟踪, 提出了在需求跟踪领域中利用开发者信息建立代码之间联系的方法[19]. 实验发现, 需求的主要贡献者所开发的代码往往与这个需求有关. 该方法的简要流程为: 在最初由需求和目标软件制品之间通过信息检索得到的代码列表中, 对于主要开发者所写的代码的相似度增加一个权重, 该权重值为实验经验值, 得到一个新的代码列表. 通过这种方法来得到的需求跟踪关系的评估结果比单独使用 IR 技术要好. 这是因为在文本信息基础上增加了开发者的信息, 在一定程度上补充了文本信息的不足.

在软件开发过程中, 开发者是代码的直接构建者. 稍具规模的软件开发都很难由一个开发者单独完成, 而软件开发中的一些经典模式(如模块化、层次化), 使多个开发者能高效独立地完成开发工作[10,16]. 在此过程中, 一个开发者通常关注一个或多个模块, 而出于代码质量保证的考虑, 一个模块也通常由多个开发者共同承担或更新. 由于相同模块内的代码之间的关联性很高, 开发过程中的协作关系直接体现了代码之间的关联性. 据我们所知, 目前尚没有基于开发者协作关系的需求跟踪方法的工作发表.

3 系统设计

本文设计和开发了一个基于开发者协作关系和信息检索的需求跟踪系统. 系统一共分为四个部分: 数据获取与处理、文本信息检索、开发者协作关系信息

挖掘和分析以及需求跟踪关系推荐. 该系统在进行需求跟踪时, 首先用信息检索推荐与需求文本上相似的代码, 然后从代码提交日志中挖掘开发者协作关系, 根据开发者协作关系再推荐相关代码, 用户根据两次推荐的结果确定正确的需求代码跟踪关系. 系统总体框图如图 1 所示.

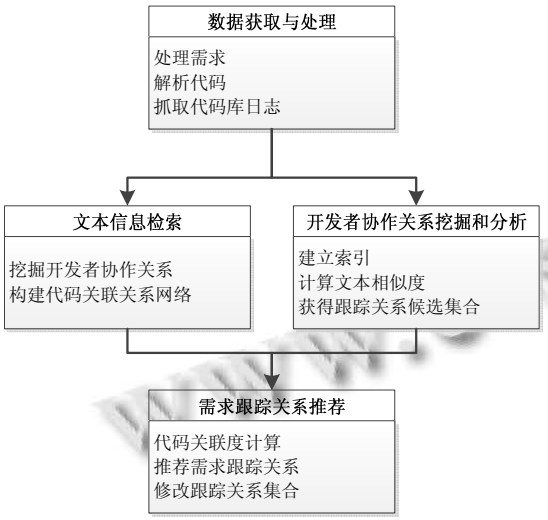

图 1 系统总体框图

基于 IR 的需求跟踪系统一般只包含数据处理与预处理模块和文本信息检索模块. 从图 1 中我们可以看出, 本系统在其基础上, 增加了开发者协作关系挖掘分析模块和需求跟踪关系推荐模块. 本文提出利用开发者协作关系改进基于 IR 的需求跟踪方法, 捕捉了开发者之间的协作关系, 在一定程度上补充了文本信息的不足, 有利于提高需求到代码的跟踪精度.

3.1 数据获取与处理

数据获取主要包括需求、源代码文件、代码库代码提交日志. 数据预处理过程主要是针对需求文本和代码进行预处理, 包括文本分词、停用词去除和标引词选择等过程.

需求预处理以英文需求文本为例. 将给定的需求当做字符序列, 拆分成一系列单词, 同时去掉特殊字符, 如标点符号等. 代码预处理以 Java 字节码文件为例. 选取 Java 字节码文件, 通过代码解析技术, 提取字节码文件中的类名、方法名、静态变量, 将多个词组成的标识符进行分割, 过滤掉无意义的字符.

停用词表, 主要采用英文通用的停用词表, 在其基础上根据项目的源代码和需求描述的特征增加一些停用词. 用停用词表过滤从需求和代码中抽取的到的词条.

标引词选择是在分词得到的词项集合的基础上过滤掉停用词, 得到的词项集合作为初始的标引词.

3.2 开发者协作关系信息挖掘和分析

3.2.1 开发者协作关系信息挖掘

现代的软件开发过程中, 开发者的代码协作关系直接体现在代码的提交日志中. 本文对开发者协作关系的定义是任意两个开发者共同开发或修改过某个代码文件, 这两个开发者在这个代码文件上有协作开发关系. 日志记录了开发者、代码名称、代码位置、提交时间等信息. 本文从日志中挖掘开发者协作关系.

3.2.2 构建代码关联网络

两个代码文件被开发者们协作开发过, 则可以确定它们之间存在关联关系, 它们被开发者协作开发过的次数决定了它们之间的关联度. 构建代码关系网络时, 代码节点之间的无向边, 表示代码之间存在关联关系, 边的权重是这两个代码共同被开发者协作开发过的次数. 代码关联关系网络如图 2 所示.

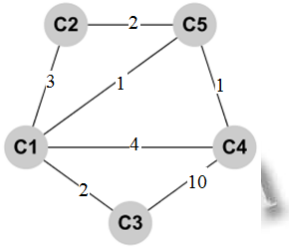

图 2 代码关联关系网络图

3.2.3 代码关联度计算

从上一步中得到代码之间的关系, 如果代码C1和代码 C2 之间有边, 则可以计算其关联度, 没有边的, 关联度默认为 0. 边的权重越大, 则代码之间的关联程度越高. 计算关联度大小, 是为了找到与某个代码最相关的某一个代码或某几个代码. 可以将边的权重进行归一化, 来表示代码之间的关联程度. 我们可以利用条件概率来计算. 例如, 计算 C2 与代码 C1 的关联程度, 用它们共同出现的概率来描述, 如公式(1)所示:

$$关联度(C2,C1)=\frac{C1,C2边权重}{C1边权重之和} \tag{1}$$

在图 2 所示网络中应用上述公式, 计算代码文件之间的关联程度(C1,C2) =3/(2+3+4+1)=0.3, 关联程度(C1,C2) =3/(3+2)=0.6.

3.3 文本信息检索

本文设计的需求跟踪关系检索系统是针对需求跟踪的应用领域, 在文本信息检索的基础上, 利用代码提交信息作为补充信息, 改进文本信息检索. 所以该系统的基础是建立一个需求到代码的文本信息检索模块, 这个模块包括数据预处理过程、构建 tf-idf 倒排索引表、文本相似度计算三个过程.

3.3.1 构建 tf-idf 倒排索引表

从需求和代码中抽取出标引词建立索引. 在对文本规范化预处理之后, 建立索引和语料.

建立索引过程的结果是生成一个 m*n 的矩阵, 标引词-文档矩阵, m 是所有软件制品中出现的标引词个数, n 是软件制品的数量.$w_{i,j}$ 表示第 i 个标引词在第 j 个文档中的权重. 一个广泛使用的权重模式是 tf-idf, 单词在一个文档中出现频次越高(高 tf), 出现过的文档数越少(高 idf), 权重越高[4]. 本文将需求和代码文档处理成 ARFF 格式的文件, 利用 Weka 工具提供的开源接口实现 tf-idf 的计算, 得到需求和代码的 tf-idf 向量空间集合, 作为下一步实验的输入.

3.3.2 计算文本相似度获得需求跟踪关系候选列表

建立需求文档和代码文档的标引词倒排索引表, 对需求文档向量和代码文档向量进行文本相似度计算. 将每一条需求视为一条查询, 将所有代码视为目标查询文档, 使用向量空间模型(VSM)算法, 对需求和代码的 tf-idf 向量空间计算余弦相似度. 文档和查询都用向量来表示. 在需求到代码跟踪检索系统中, 需求作为查询向量, 代码文件作为文档向量.

代码文档向量表示如公式(2)所示:

$$d_j = \left(w_{1,j}, w_{2,j}, \ldots, w_{t,j}\right) \tag{2}$$

查询向量表示如公式(3)所示:

$$q = \left(w_{1,q}, w_{2,q}, \ldots, w_{t,q}\right) \tag{3}$$

计算需求与代码之间的相似度如公式(4)所示:

$$sim\left(d_j, q\right) = \frac{d_j * q}{\| d_j \| * \| q \|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2}\sqrt{\sum_{i=1}^{N} w_{i,q}^2}} \tag{4}$$

将作为查询的需求向量与所有代码文档向量分别进行相似度的计算, 根据相似度计算结果对所有代码文档向量进行排序, 得到相似度由大到小的列表, 这就是根据文本相似度得到的需求和代码类之间的跟踪关系的候选列表.

3.4 跟踪关系推荐

本文将通过 IR 检索到的代码相似度由大到小的排序结果, 作为需求跟踪关系候选列表. 然后依据 3.3 得到的代码关联关系网络, 依次为候选列表中的代码, 查找最紧密关联的代码. 如果查找到的关联代码数量小于阈值 μ, 且文本相似度小于当前代码中, 就将它们推荐出来, 并将相似度调整为当前代码的文本相似度. 这样得到一个新的排序结果. 最终在新的排序结果上取 Top N 检验跟踪效果. 如果与某个代码关联度最大的代码超过了 μ, 则不对该代码进行关联代码推荐. 设定阈值 μ 是为了避免推荐过多无关的代码影响结果的准确性. μ 的取值由实验训练得出.

例如, 假设 μ=3, 需求 R1 的需求跟踪关系候选列表前 3 个代码及相似度为{C2:0.80, C4:0.75, C6:0.55}. 进行需求跟踪推荐时, 查找代码关联关系网络, 通过计算关联度, 筛选出与候选代码关联度最高的代码. 与 C2 最关联的代码是{C5:0.43}, 将会为 C2 推荐 C5, 且将 C5 的相似度调整为{C5:0.80}. 与 C4 最关联的代码是{C11,C2}, C2 相似度大于 C4, 则只为 C4 推荐 C11, 将 C11 的相似度调整为{C11:0.75}. 与 C6 最关联的代码是{C2,C4,C7,C11,C13 }, 由于 C6 最关联的代码数超过了阈值 μ 的大小, 将不对 C6 的进行推荐. 最后得到新的需求到代码的跟踪排序结果是{C2:0.80, C5:0.80, C4:0.75, C11:0.75, C6:0.55, …}.

4 系统应用与结果分析

4.1 实验数据

本文选取 ArgoUML 作为实验的分析对象, 目标是对 ArgoUML 的需求和代码建立需求跟踪系统, 并能够使该系统也能应用到其他开源项目中. ArgoUML 是一款 UML 建模的开源工具, 支持 UML 1.4 以及 UML 2.0 标准. 从 1999 年发展至今, ArgoUML 衍生出了若干软件产品. 在软件工程领域, 常有研究者使用 ArgoUML 的数据作为实验数据[8].

实验验证数据集采用威廉玛丽学院提供的公开标准数据集, 数据最初来自 10th Conference on Mining Software Repositories (MSR 2013), 数据下载地址: http://www.cs.wm.edu/semeru/data/msr13/. 实验选取 ArgoUML 的 0.22、0.24 和 0.26.2 三个版本中的新增功能 25 个作为需求(需求的描述语言为英文), 0.26.2 版本的 1701 个 Java 类作为检索目标, 这三个版本发布期间

的代码修改日志作为挖掘开发者协作关系的数据源. 需求跟踪粒度为 Java 类层面.

4.2 系统评价指标

采用上述标准数据集, 确定需求到代码的标准跟踪关系, 作为需求跟踪关系的验证数据. 对比需求跟踪关系文本信息检索的结果与机遇代码提交信息的需求跟踪关系检索的结果, 通过准确率、召回率和 F-值三个指标作为衡量结果好坏的标准.

准确率(Precision): 找到的正确关系数量与找到的关系数量的比值[10]. 计算方法如公式(6)所示: :

$$准确率 = \frac{检索到的相关文件}{检索到的文件总数} \tag{6}$$

召回率(Recall): 找到的正确关系数量与所有的正确关系数量的比值[10]. 计算方法如公式(7)所示:

$$召回率 = \frac{检索到的相关文件}{所有相关的文件总数} \tag{7}$$

F-值(F-measure)[10]计算方法如公式(8)所示. 在这值得一提的是, 对于需求跟踪关系检索, 查全率要比查准率重要, 因此的取值常大于 1[10]. 一般情况下, F 值取比较合适.

$$F_\beta = \left(1+\beta^2\right) * \frac{准确率 * 召回率}{\beta^2 * 准确率 + 召回率} \tag{8}$$

分别取文本信息检索前 N 个结果, 和基于代码提交信息的需求跟踪关系检索的前 N 个结果, 计算准确率、召回率和 F-值, 分别用 P@N、R@N 和 F@N 表示.

4.3 系统应用结果

在威廉玛丽学院提供的 ArgoUML 数据集上进行系统应用. 数据集包括 ArgoUML 的 0.22、0.24 和 0.26.2 三个版本的需求和代码数据以及正确的跟踪关系.

依照 3.1 介绍的数据处理方法, 对需求处理后得到 25 条需求向量, 平均长度为 11 个单词. 对 ArgoUML 的代码文件 jar 包进行代码解析, 得到 1701 个类, 抽取出每个类中的类名、方法名和变量名, 进行分词、去除停用词, 建立每个类的向量. 将需求和代码向量统一成一个向量空间, 以便利用 VSM 信息检索技术.

从 ArgoUML 官方网站上抓取代码修改日志记录, 共有 5581 条. 其中代码文件的个数为 1623 个, 开发者有 47 个. 从日志中挖掘开发者协作关系, 建立代码关联关系网络, 如图 3 所示. 图 3(a)是用 Gephi 工具绘制的整体网络图, 其中有 1701 个节点, 304110 条无向边, 78 个孤立节点, 最大的边权重为 152, 最小的边权重为 1, 网络的平均度为 55.382, 平均加权度为 83.9. 图 3(b) 是局部代码关联关系网络, 描绘的是以 SettingsTabNotation 为核心的权重大于 1 的边, 以及这些边所连接到节点的边. 与 SettingsTabNotation 关联的权重为 1 的边数量为 56, 由于不方便绘制, 因此在局部网络中忽略权重为 1 的边以及对应的节点.

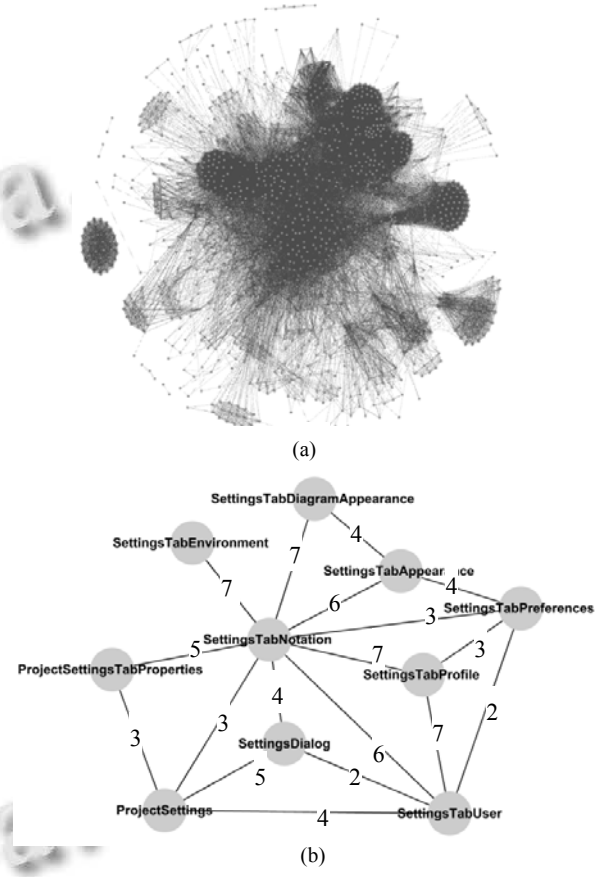

图 3 代码关联关系网络

对 ArgoUML 进行需求跟踪, 平均每条需求对应的代码个数为 6.52 个. 检索返回结果的前 10 个进行验证, 就可以有效地反映本文所提出的假设是否成立. 经过多次试验, 阈值 μ 取 3 时, 实验效果最好.

以输入需求“Better indication of default settings vs. project specific settings”为例, 分别取基于 IR 方法和基于 IR+开发者协作关系的方法的检索结果前 10 项, 结果如表 1 所示.

表 1　需求跟踪检索结果对比

IR	IR+开发者协作关系
ProjectSettingsTabProperties	ProjectSettingsTabProperties
SettingsTabNotation	SettingsTabNotation
ActionProjectSettings	SettingsTabDiagramAppearance
GUIProjectSettingsTabInterface	SettingsTabEnvironment
SettingsTabLayout	SettingsTabProfile
SettingsTabUser	SettingsTabUser
GUISettingsTabInterface	ActionProjectSettings
SettingsTabProfile	SettingsTabLayout
GUI	InitDiagramAppearanceUI
ProjectSettingsTabProfile	GUIProjectSettingsTabInterface

表中左列结果是基于 IR 方法得到的候选跟踪列表前 10 项. 右列是在这候选跟踪列表上应用 3.4 需求跟踪推荐方法调整后的排序结果前 10 项, 也就是基于“IR+开发者协作关系”方法得到的结果. 而根据标准数据集的正确跟踪关系, 该需求对应的 Java 代码类是{ SettingsTabNotation, SettingsTabProfile, BasicLinkButtonUI, JLinkButton, SettingsTabUser, SettingsTabDiagramAppearance}. 对比表中两列数据, 发现 IR 方法找到 3 个正确的类, “IR+开发者协作关系”方法找到 4 个类. 在这个例子中, “IR+开发者协作关系”方法结果优于 IR 结果. 具体分析, 根据图 3(b)网络局部图与公式(1)计算 SettingsTabNotation 与相关代码的相似度, 得到与 SettingsTabNotation 关联度最高的类是 SettingsTabProfile、SettingsTabEnvironment 和 SettingsTabDiagramAppearance. 当用 IR 方法检索到 SettingsTabNotation 后, 可以根据 3.4 描述的方法重新调整 SettingsTabDiagramAppearance 等的权重, 使得它们排名提前. 找到了文本相似度较低的 SettingsTabDiagramAppearance.

4.4 对比分析

对需求跟踪关系检索系统而言, 采用的需求数据和代码数据对于正确的需求跟踪关系的数量是有直接影响的. 本文实验中, 平均每条需求对应的代码个数为 6.52 个. 对需求到代码的检索返回结果的前 10 个进行验证, 就可以有效地反映本文所提出的假设是否成立. 我们对 25 条需求的检索结果, 采用系统评价指标准确率、召回率和 F-值($\beta=2$)来做评测, 分别取 P@10, R@10, F@10. 评测结果如图 4 所示.

图 4 中三个曲线图, 横轴代表 25 条需求的序号, 纵轴分别代表准确率、召回率和 F-值. 从图中可以看出, “IR+开发者协作关系”的值基本上都高于 IR 的值. 基于 IR 方法的需求跟踪结果准确率集中在 0.2~0.3 之间, 召回率集中在 0.38~0.48 之间, F-值($\beta=2$)集中在 0.27~0.37 之间. “IR+开发者协作关系”的需求跟踪结果准确率集中在 0.3~0.4 之间, Recall 集中在 0.50~0.60 之间, F-值($\beta=2$)集中在 0.35~0.45 之间. 几乎对于每一个需求到代码的跟踪关系检索上, “IR+开发者协作关系”的准确率、召回率、F-值都比单纯使用 IR 方法的评测值高. 这说明利用开发者协作关系进行需求跟踪关系的检索, 比只利用 IR 的效果要更好. 在这需要指出, 对于需求跟踪关系检索, 查全率要比查准率重要, 一般情况下, 取 $\beta=2$ 比较合适[10].

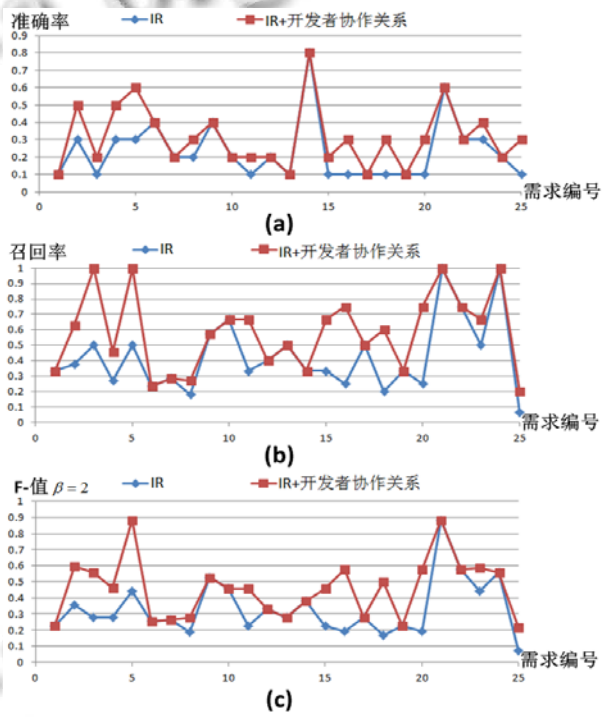

图 4　系统评价结果

平均值更能体现系统的整体效果. 对 IR 的方法和“IR+开发者协作关系”的方法, 比较前 N 个检索结果的平均准确率、平均召回率以及平均 F-值, 对比实验结果如表 2 所示. 表中 P 表示准确率, R 表示召回率, F 表示 F-值($\beta=2$).

表 2　对比实验 Top N 的各项指标

Top N	N = 5			N = 10		
指标	P	R	F	P	R	F
IR	0.35	0.35	0.35	0.30	0.42	0.36
IR+开发者协作关系	0.45	0.45	0.42	0.42	0.58	0.49
提升	0.10	0.10	0.07	0.12	0.16	0.13

从表 2 可以看出, 基于“IR+开发者协作关系”进行需求跟踪关系的推荐结果比基于 IR 的结果好, Top 10 的情况下, 准确率提高 0.12, 召回率提高 0.16, F-值也有相应的提高. 由此可见, 开发者协作关系确实可以用来辅助文本信息检索进行需求跟踪关系的检索, 且对检索结果的提升有明显的效果.

目前, 实际应用中的自动化需求跟踪系统在没有人工筛选的情况下, 召回率在 0.50-0.65 之间[18,20]. 本文开发的需求跟踪系统平均召回率为 0.58, 且自动化程度高. 用户只需要提供需求、代码和日志的数据, 系统将会进行需求到代码的自动跟踪. 本系统除数据收集以外的所有过程都是自动化的. 但是本系统的数据收集和预处理耗费的工作量较大, 未来需要规范数据格式, 进而减少工作量以及提高系统可拓展性. 另一方面, 利用开发者协作关系进行需求跟踪, 与只利用文本信息的需求跟踪相比, 整体运行效率没有显著差异, 运行时间均在 40s 至 60s 之间. 整体上, 本系统精确度较高、自动化程度高、运行稳定、效率较高、可拓展性较强, 能够满足实际工程应用情况的需求.

5 结语

本文针对需求到代码的跟踪关系检索方法, 提出改进方法, 研发需求跟踪系统: 从代码提交信息中发现开发者协作关系, 对代码关联关系建模, 构建代码关联网络, 利用代码关联关系进行需求跟踪关系的推荐, 调整需求跟踪检索结果. 系统应用和实验结果证明, 在文本信息基础上补充软件开发者协作信息, 能够找到一些文本信息检索找不到的代码, 有效提高自动化跟踪的准确率和召回率. 本文开发的需求跟踪系统能够满足实际工程应用情况的需求.

参考文献

1 Gotel OCZ, Finkelstein ACW. An analysis of the requirements traceability problem. Proc. of the First International Conference on Requirements Engineering, 1994. 94–101.

2 Baeza-Yates R, Ribeiro-Neto B. Modern information retrieval. New York: ACM press, 1999.

3 De Lucia A, Marcus A, Oliveto R, et al. Information retrieval methods for automated traceability recovery. Software and Systems Traceability, London, Springer. 2012. 71–98.

4 De Lucia A, Marcus A, Oliveto R, et al. Information retrieval methods for automated traceability recovery. Software and Systems Traceability, London, Springer. 2012. 71–98.

5 Abadi A, Nisenson M, Simionovici Y. A traceability technique for specifications. The 16th IEEE International Conference on Program Comprehension, ICPC 2008. 2008, 8: 103–112.

6 McMillan C, Poshyvanyk D, Revelle M. Combining textual and structural analysis of software artifacts for traceability link recovery. Traceability in Emerging Forms of Software Engineering, 2009: 41–48.

7 Asuncion HU, Asuncion AU, Taylor RN. Software traceability with topic modeling. Proc. of the 32nd ACM/IEEE Int. Conf. on Software Engineering. 2010. 95–104.

8 Gethers M, Oliveto R, Poshyvanyk D, et al. On integrating orthogonal information retrieval methods to improve traceability recovery. 27th IEEE International Conference on Software Maintenance (ICSM). 2011. 133–142.

9 Panichella A, McMillan C, Moritz E, et al. When and how using structural information to improve ir-based traceability recovery. 17th European Conference on Software Maintenance and Reengineering. 2013. 199–208.

10 Kong WK, Huffman Hayes J, Dekhtyar A, et al. How do we trace requirements: an initial study of analyst behavior in trace validation tasks. Proc. of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering. 2011. 32–39.

11 李引.需求变更影响分析模型及相关技术研究[学位论文]. 北京:中国科学院软件研究所,2010.

12 Cleland-Huang J, Czauderna A, Gibiec M, et al. A machine learning approach for tracing regulatory codes to product specific requirements. Proc. of the 32nd ACM/IEEE Int. Conf. on Software Engineering. 2010. 155–164.

13 Capobianco G, Lucia AD, Oliveto R, et al. Improving IR-based traceability recovery via noun-based indexing of software artifacts. Journal of Software: Evolution and Process, 2013, 25(7): 743–762.

14 Kagdi H, Maletic JI, Sharif B. Mining software repositories for traceability links. 15th IEEE International Conference on Program Comprehension, 2007: 145–154.

15 Meneely A, Williams L, Snipes W, et al. Predicting failures with developer networks and social network analysis. Proc. of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008: 13–23.

16 Wolf T, Schroter A, Damian D, et al. Predicting build failures using social network analysis on developer communication. Proc. of the 31st International Conference on Software Engineering. 2009. 1–11.

17 Anvik J, Murphy GC. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. ACM Trans. on Software Engineering and Methodology, 2011, 20(3): 10.

18 Duc Anh N, Cruzes D S, Conradi R, et al. Empirical validation of human factors in predicting issue lead time in open source projects. Proc. of the 7th Int. Conf. on Predictive Models in Software Engineering. 2011. 13.

19 Diaz D, Bavota G, Marcus A, et al. Using code ownership to improve IR-based Traceability Link Recovery. 21st Int. Conf. on Program Comprehension, 2013.