申请上海交通大学工程硕士学位论文

基于社会软件工程的软件缺陷预测

|  |  |
| --- | --- |
| 学校代码： | 10248 |
| 作者姓名： |  |
| 学 号： |  |
| 导 师： |  |
| 学科专业： | 软件工程 |
| 答辩日期： | 2015年 1月 14日 |

上海交通大学软件学院

2014年 12月

A Dissertation Submitted to Shanghai Jiao Tong University

for Master Degree of Engineering

Software Defect Prediction Based on Social Software Engineering

|  |  |
| --- | --- |
| University Code： | 10248 |
| Author： |  |
| Student ID： |  |
| Mentor： |  |
| Field： | Software Engineering |
| Date of Oral Defense： | 2015.1.14 |

School of Software

Shanghai Jiaotong University

Dec., 2014

上海交通大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期： 年 月 日

上海交通大学

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

**保密**□，在 年解密后适用本授权书。

本学位论文属于

**不保密**☑。

（请在以上方框内打“√”）

学位论文作者签名： 指导教师签名：

日期： 年 月 日 日期： 年 月 日

基于社会软件工程的软件缺陷预测

# 摘 要

作为当前软件工程的研究热点，社会软件工程已取得了不少研究成果，并被应用于软件开发的多个方面，例如专家发现、缺陷分类、开发协同模式等。与此同时，软件缺陷预测在软件质量的提高和保证、平衡软件成本方面起着重要的作用。将两者结合，基于社会软件工程来进行缺陷预测具有良好的研究价值和应用价值。但相关研究刚刚起步，现有研究主要从传统社会网络分析方法提取特征进行缺陷预测，所采用的社会网络图的构建方式存在数据单一和关联性差的不足，同时缺乏对非结构化的交流信息的分析与研究。

本文从开发者邮件列表和社会网络分析两个角度分别对社会软件工程中开发者的交流与协作进行探索，进而提取出新的缺陷预测特征：

1）将缺陷跟踪信息与开发者邮件进行关联，从邮件的结构、正负面情感和主题三个方面设计相应的分析方法开展探索实验，根据探索发现提取出和代码潜在缺陷相关的邮件交流特征。

2）改进现有社会网络图的构建方法，将软件仓库中多种数据源相结合，使得所构建的网络图包含文件依赖、开发者贡献和开发者协作等信息，通过探索实验，抽取出与缺陷较强的关联的度中心性、中介中心性等特征。然后对社会网络进行扩展，从开发者地位与影响、项目组织结构等角度研究，通过探索实验，抽取出核心开发者、工作组织流程等新特征。

在此基础上，本文设计出多项社会软件工程新特征，包括邮件的内容结构特征、网络结构特征、情感特征与主题特征，和网络中心性特征、核心开发者特征和组织结构特征，实现基于新特征的缺陷预测，并采用eclipse开源项目进行了特征相关性实验和对比分析实验。实验结果表明，研究所提出的社会软件工程新特性与缺陷具有0.389~0.473的较高相关性；新特征的预测准确率为83.1%，覆盖率为81.2%，均优于现有社会化特征；同时在现有特征上加入社会化预测特征，能明显提高预测效果。

关键词：社会软件工程，缺陷预测，邮件列表，社会网络分析，软件仓库挖掘

DEFECT PREDICTION BASED ON SOCIAL SOFTWARE ENGINEERING

# ABSTRACT

As current software engineering research focus, social software engineering has been studied in-depth and a lot of achievements have been gained. Achievements are applied in many fields, such as expert finding, bug categorizing and the development of [coordination](javascript:void(0);) [pattern](javascript:void(0);). Besides, software defect prediction plays an important role in improving software quality and balancing software costs. Their combination, which means predicting defects based on social software engineering should have great research and application value. However, this is just been proceeded and the existing research is mainly extracting features from the traditional social network for defect prediction while the method of building social network is simple and monotonous, there also exists a lack of analysis and research on the unstructured communication information.

This paper explores the communication and collaboration information of social software engineering separately based on the developer mailing list and social network analysis, and then extracts new defect prediction features:

1) Link bug tracking databases to the mailing list archives. Then perform experiments through three aspects including the structure of message, both positive and negative emotions and the topic of mailing list with different approaches. Based on the exploration, mailing list features that may be correlated with source code defects are extracted.

2) Improve social network from the perspective of construction method. The combination of multiple data sources in software repository enables the new network to contain more information about file dependency, developer contribution and collaboration. Through exploration, several features such as degree centrality and betweenness centrality which are of strong correlations with the software defects have been extracted. Next social network is expanded for researching the status and influence of developers as well as the organization structure of projects. Then new defect prediction features are extracted as core developer workflow in organization.

On this basis, this paper makes use of new features including mail content feature, mail network feature, emotion feature, topic-based feature, network centrality feature, core developer feature and organization structure feature based on social software engineering to conduct defect prediction. Then correlation experiment and contrastive analysis experiment are conducted with eclipse project as the dataset. Results show that features based on mailing list and social network analysis are correlated with defects between 0.389 and 0.473 and the precision and recall of new features are 83.1% and 81.2% respectively, which are better than existing social features. Besides, by combining social features into the state-of-the-art features, the effectiveness of the prediction model is obviously improved.

**Keywords** Social Software Engineering, Defect prediction, Mailing list, Social Network Analysis, Software repository mining

**目 录**

[1 绪 论 1](#_Toc407483762)

[1.1 研究背景 1](#_Toc407483763)

[1.2 研究目的和意义 1](#_Toc407483764)

[1.3 研究目标和内容 2](#_Toc407483765)

[1.4 论文结构 3](#_Toc407483766)

[2 国内外研究现状及分析 4](#_Toc407483767)

[2.1 社会软件工程 4](#_Toc407483768)

[2.1.1 社会软件工程的概念和研究现状 4](#_Toc407483769)

[2.1.2 交流与邮件 5](#_Toc407483770)

[2.1.3 社会网络分析在软件工程中的应用 6](#_Toc407483771)

[2.2 软件缺陷预测 7](#_Toc407483772)

[2.2.1 缺陷预测的概念和基本流程 7](#_Toc407483773)

[2.2.2 缺陷预测特征 8](#_Toc407483774)

[2.3 软件仓库挖掘技术 9](#_Toc407483775)

[2.4 本章小结 10](#_Toc407483776)

[3 基于开发者邮件列表的缺陷预测新特征研究 11](#_Toc407483777)

[3.1 研究方法 11](#_Toc407483778)

[3.2 开发者邮件列表的数据分析与挖掘 12](#_Toc407483779)

[3.2.1 关联邮件与源代码 12](#_Toc407483780)

[3.2.2 挖掘邮件列表结构 13](#_Toc407483781)

[3.2.3 LIWC 分析 15](#_Toc407483782)

[3.2.4 主题分析 16](#_Toc407483783)

[3.3 探索实验及发现 18](#_Toc407483784)

[3.3.1 实验数据 18](#_Toc407483785)

[3.3.2 结构分析实验及发现 20](#_Toc407483786)

[3.3.3 情感分析实验及发现 22](#_Toc407483787)

[3.3.4 主题分析实验及发现 23](#_Toc407483788)

[3.4 基于邮件列表的新特征 24](#_Toc407483789)

[3.5 本章小结 25](#_Toc407483790)

[4 基于社会网络分析的缺陷预测新特征研究 26](#_Toc407483791)

[4.1 研究方法 26](#_Toc407483792)

[4.2 社会网络图设计与分析 26](#_Toc407483793)

[4.2.1 社会网络分析方法 26](#_Toc407483794)

[4.2.2 社会网络图构建与度量 27](#_Toc407483800)

[4.3 开发者地位与影响探索 32](#_Toc407483801)

[4.4 项目组织结构研究 34](#_Toc407483802)

[4.4.1 项目组织结构特征提取 34](#_Toc407483803)

[4.4.2 开源项目组织结构案例研究 37](#_Toc407483804)

[4.5 探索实验及发现 39](#_Toc407483805)

[4.5.1 网络图研究发现 39](#_Toc407483806)

[4.5.2 开发者地位研究发现 40](#_Toc407483807)

[4.5.3 组织结构研究发现 41](#_Toc407483808)

[4.6 基于社会网络分析的新特征 42](#_Toc407483809)

[4.7 本章小结 42](#_Toc407483810)

[5 基于社会软件工程新特征的缺陷预测 43](#_Toc407483811)

[5.1 缺陷预测模型 43](#_Toc407483812)

[5.2 预测特征 44](#_Toc407483813)

[5.3 预测评估指标 44](#_Toc407483814)

[5.4 缺陷预测实验 45](#_Toc407483815)

[5.4.1 实验数据和实验过程 45](#_Toc407483816)

[5.4.2 实验结果 45](#_Toc407483817)

[5.4.3 实验对比分析 46](#_Toc407483818)

[5.5 本章小结 48](#_Toc407483819)

[6 总结与展望 49](#_Toc407483820)

[6.1 本文工作小结 49](#_Toc407483821)

[6.2 展望 49](#_Toc407483822)

[参考文献 53](#_Toc407483823)

[攻读学位期间发表的学术论文 60](#_Toc407483825)

# 绪 论

## 研究背景

社会软件工程（Social Software Engineering，简称SSE）是软件工程的一个分支，主要关注软件开发和开发软件的社会方面或社会性。

早在1971年，将软件开发作为一种社会活动的首批推动者之一的Gerald M. Weinberg[1] 认为软件开发是一种以人为中心的活动，开发者之间的相互关系发挥着关键作用。Lui和Chan[2] 同样对软件工程有以人为中心的认识，他们提出多种方法来处理协同编程，并且展示出如何将这些实践结合与协调。Yourdon[3] 对一系列软件项目标识为“death march”，表明当前的组织、管理方式与项目成功之间的重要关系。可见，社会软件工程能够有助于更有效地进行软件开发。

另一方面，软件缺陷预测是一种通过挖掘软件仓库，获取软件的特征，从而对软件的缺陷数量和缺陷类型做出预测的技术，在软件质量的提高和保证、平衡软件成本方面起着重要的作用。缺陷预测技术可以帮助开发者对软件的质量状态进行清楚并客观地评估，既能在软件发布前判断软件是否可以交付使用，又能预测软件使用过程中的失效模式，同时准确地预测软件缺陷的分布情况，对软件测试工作也有着重要的指导意义。

近来，研究者开始了解到社会网络、工作依赖、日常事务处理等活动的错综复杂的性质，进而推断其与软件产品质量的相关性。对这些活动的跟踪可以从开发者每日使用的软件仓库中获得信息，如版本存档、事件追踪系统、邮件通讯存档。在此基础上，探究开发者的交流特征、社会化结构等方面与软件质量的关系，进行基于社会软件工程的缺陷预测研究将具有良好的研究价值和应用价值。

## 研究目的和意义

现在已经普遍认同软件开发是一项以人为中心的社会活动。软件开发中的协作交流、开发者之间的组织结构乃至人员能力、专家分布等等都会对软件质量产生影响。在协同开发软件的过程中，交流结构的不合理会对开发者产生约束，进而造成软件缺陷。Timo Wolf[5] 等人对成功合作的项目特征与开发者交流结构之间的关系进行探索，得出交流对软件质量有较大影响的结论。关于协作活动的信息可以通过跟踪交流内容来获取，这些交流内容被记录在软件开发中的软件仓库[4]。通过对开源软件项目进行广泛的案例研究，证实出涉众之间的社会关系对软件系统开发的重要影响。

因此，研究社会软件工程，分析开发者之间的社会关系信息对软件质量的影响能够帮助开发者了解交流协作方式中存在的问题，进而进行调整，这对提高开发效率有重要意义。另外，现有的缺陷预测研究工作广泛使用产品与过程度量手段，从软件源代码以及其变更与缺陷历史中获取特征。然而，源代码是软件开发者通过一系列协作活动得到的成果，如何使用作为主体的开发者之间的社会关系信息来预测软件缺陷，研究其对软件质量的影响，进而利用社会软件工程的特征对软件缺陷进行预测，发现项目潜在缺陷，应用于软件工程的维护与测试中，同样具有重要意义。

## 研究目标和内容

本文的研究目标是基于社会软件工程的特性，提取社会性相关的软件特征，建立预测模型，进行软件缺陷的预测。

具体研究内容包括以下几个部分：

1. 收集数据，构建软件仓库，提取社会化信息。从版本控制系统（如CVS、SVN）、缺陷跟踪库（如Bugzilla）以及邮件列表等收集软件开发过程中协作交流、组织结构等相关数据，建立软件仓库，进行分析，从中提取出社会化的信息，并进行合理存储。
2. 研究与提取基于开发者邮件列表的交流特征。通过挖掘软件开发者之间的邮件列表来提取缺陷预测特征，将缺陷跟踪信息与开发者邮件进行关联，采用社会网络分析和文本分析技术挖掘邮件中的结构化与非结构化信息。从邮件的结构、情感特征和主题三个方面开展探索实验。
3. 研究与提取社会网络分析与特征。一方面通过软件仓库中得到的社会化数据信息建立社会网络图并进行改进，基于社会网络分析方法从中提取预测特征；另一方面对社会网络进行扩展，将开发者地位与项目组织结构归结为社会网络分析问题，提取出新的缺陷预测特征。
4. 建立基于社会软件工程的缺陷预测模型，进行预测实验。对各项社会软件工程特征进行分析，建立缺陷预测模型，预测实际开源项目中的缺陷情况。通过实验和对比分析，对预测模型进行调整，改进预测效果，最后得出结论。

## 论文结构

论文共分六章，第一章主要就本文研究背景、目的与意义以及研究目标与内容进行整体介绍，对本文将要讨论的内容进行总揽；第二章将从社会软件工程和软件缺陷预测等方面总结国内外研究现状，并分析现有研究的不足；第三章将主要针对邮件列表对社会软件工程中的交流与质量的联系进行探索，提取特征；第四章采用社会网络分析方法，对网络图进行改进，探究项目开发与组织中的社会化特征；第五章基于社会软件工程的新特征，建立预测模型，进行缺陷预测；第六章进行总结与展望。

# 国内外研究现状及分析

## 社会软件工程

本节就社会软件工程的概念以及本文中涉及到的社会网络与邮件交流等研究进行综述。

### 社会软件工程的概念和研究现状

社会软件工程能简单定义为应用过程、方法和工具于在线环境下进行社区驱动的软件创建、管理、部署和使用。虽然很难对社会软件工程给出确切的定义和划定严格的界限，但至少应关注两个软件开发的中心方面：一是轻载的以社区为中心的合作；再是用在线环境共享一个软件产品的制品和知识。总结起来，社会软件工程有以下特征描述：

1. 以社区为中心（Community-centered）：软件的生产和/或消费是就社区而言，而不是专注于个人。
2. 协作/集体主义（Collaboration/collectivity）：利用人类协作和行动的集体能力。
3. 同伴/关系（Companionship/relationship）：明确人与人之间的各种关联。
4. 人力/社会活动（Human/social activities）：软件有意地设计为支持人类活动以及解决社会问题。
5. 社会融合（Social inclusion）：软件开发应该发挥社会包容性增强社区的联系和信任。

社会软件工程目前是软件工程一个新的研究热点，其研究内容涉及到三个方面：软件工程中社会活动的研究、社会软件工程相关数学方法的研究、社交应用（social application）软件的开发。

1. 软件工程中社会活动的研究既包括对软件开发中涉众间的交流与协作进行研究，也指将社会学、心理学等学科集成到软件开发过程中来帮助开发团队提高生产效率。其中关于交流以及邮件的研究会在2.1.2节概述。
2. 社会软件工程相关数学方法的研究，包括关系数据的处理，特别指利用图论建立一系列用于评价社会软件工程模型。而所有包含的技术中最实用最相关的就是社会网络分析方法，关于其研究与应用将于2.1.3节总结。
3. 社交应用是指通过提取社区中成员之间的关系加以考虑，以此来更好地支持社会活动，为社区带来利益的软件。根据这个定义，所有支持用户间交流与协作的软件都能被认为是社交应用。这种支持具有一般性与领域独立性，例如电子邮件和即时消息或者为特定领域和相关社区定制的仅对领域内虚拟交流环境建立模型的工具。社会性网络服务（Social Network Services，简称SNSs）[9] 代表社交应用的一个子集得到广泛关注。除了通常的协作功能，SNSs也明确保持人与人之间的关系。这些成员之间的关系通过图的形式建立模型，能够用来分析并提取有用的信息来优化社区。Dabbish[10] 等人提出工具Transparency，帮助GitHub中的开发者管理项目，更有效地处理依赖关系，减少通讯需求，发现需要注意之处。Treude[11] 等人对普遍应用于社交应用领域的轻量级的非正式协同工具进行探究，以轻量级社交计算机制“标记（tagging）”为例分析其对工作项目管理过程中的技术与社会方面的协调优化，证明此类工具在基于团队的软件开发实践中发挥着重要作用。

### 交流与邮件

越来越多的研究人员开始关注软件开发中交流的重要性与影响力，而对开发者邮件列表的挖掘是研究交流的主要手段。

Wolf[6] 等人采用IBM Jazz 平台的项目数据，研究了具有高度协同需求的开发团队进行交流的结构模式，并阐明了交流的重要性。但其主要的不足在于研究的平台基于Jazz，对更加广泛的研究有所限制。开源软件开发团队常常使用一般电子手段（电子邮件，即时消息，论坛）[42] 来展开公开、公众的讨论，在这些手段中邮件已成为开源项目的主要交流渠道之一，研究者将邮件列表视作项目交流的中心进行研究。电子邮件涉及到系统开发中的各种问题，形成一种重要的信息资源，主要关注高层次的设计决策和低层次的实现细节，还包括开发者之间的社会结构[44]。

目前已有不少研究者基于邮件列表进行了实验探究，包括社会网络关系分析，补丁（patch）的处理，关注点的可追溯性分析，用户行为分析等。例如，Christian Bird[12] 等人通过挖掘开源项目电子邮件存档将构建出邮件参与者的社会网络图，并且探索出邮件相关活动与提交活动之间的正相关性关系。Wagstrom[13] 等人将多种社会性网络数据，包括博客、邮件列表和社交网站结合起来，构建出了社会行为模型；并基于这些数据建立了一个仿真模型描述用户加入与离开项目的过程。这些研究通过对邮件列表的挖掘发现了多项规律并加以利用，有效协助了开发。Bacchelli[16] 在此基础上从更广泛的视角对开发者所进行的交流进行研究，目的是增进对开发相关的邮件列表交流的理解。通过对邮件样本的定性与定量分析揭示出实现细节相关邮件的讨论度；核心开发人员的参与度乃至开发相关的邮件列表在开源软件项目交流中的地位。

为了更好地获取这些信息，Bacchelli试图建立电子邮件与其中所讨论的软件产品之间的链接，这是一项有意义但是困难的工作，由与人类交流中内在的非正式的特性，另外这项追溯性工作虽然有多种方法解决，但缺乏一个公认的基准来进行评判。Bacchelli[14] 等人研究了邮件与源代码之间的可追溯性连接生成问题，并在此基础上开发了Miler[15]，一个将邮件与源代码进行连接并提供统计上显著的基准数据的工具，这一工作为新加入者研究邮件列表提供了重要基础。结合基于人工检验的方式提出的基准，研究人员可以用来测量各种连接方法的有效性，包括成熟的信息检索方法与轻量级的基于正则表达式的方法，实验结论也为电子邮件信息的处理提供了参照。

### 社会网络分析在软件工程中的应用

通过构建网络的方法对社会软件工程进行研究已经被广泛应用。根据社会网络分析能够进一步了解社会软件开发中潜在的组织结构与协作模式等信息，这些信息能够协助相关人员进行更加深入的操作，例如缺陷预测[51]，缺陷分类，软件测试[17]，专家发现[54]等工作。

Pinzger[18] 等人主要通过构建贡献网络（Contribution Network），使用中心性测量的方法分析其与出错率的关系，采用windows vista进行实证研究，得出结论网络中心性是能够预测潜在错误的。另外还有更多针对开源软件的社会软件工程研究。Wolf[5] 等人对成功合作的结果与开发者交流结构之间的关系进行探索。合作结果采用代码集成，构建进程等进行判定。交流结构（Communication Structure）通过社会网络的方法进行构建。得出结论，交流对软件质量有较大影响，并提出将其加入已有预测模型进行预测的建议。Bettenburg[19] 等人通过分析软件开发中的社会结构来进行软件缺陷预测，与传统的静态模型分析比较，在传统方法上考虑开发者活动与软件质量的关系，从事件跟踪信息与版本控制库中挖掘社会信息作为预测变量（源代码，社会结构，交流测量，工作流测量），以此建立模型对开源项目进行缺陷预测。得出结论，发现社会网络方法与传统方法预测效果相似，结合预测可获得更好的效果。Bhattacharya[20] 等人展示了如何用图的方法预测安全性、维护性、错误等潜在问题，他们从代码、模块层次和处理层次（开发者协同层次）分别构建图进行预测。处理层次模型构建来自两部分：基于缺陷的开发者协同，基于提交的开发者协同。可见，从社会软件工程角度，通过社会网络分析方法进行缺陷预测是可行的。Nagappan[4] 等人从商业项目组织结构的角度考虑，整理出若干特征，研究结果提供的经验证据表明，组织结构的特征与错误产生是相关的，并且能有效进行缺陷预测。

Zanetti[21] 等人提出通过社会网络进行缺陷分类（Categorizing）的方法，缺陷分类是成功的协同软件工程的前提，其目的是发现缺陷报告中的错误。与Pinzger[18] 类似采用中心性测量方法，另外附加时间因素的限定以提高准确性。最后得出缺陷报告者在网络中的中心性与缺陷报告有效性间的相互关系，并以此提出缺陷报告有效性的预测机制。Kevic[22] 等人在原有基础上加入协同处理方法进行缺陷归类（Triaging），与缺陷分类[21]不同，缺陷归类是将发现的缺陷交付给对此缺陷最为熟悉的开发者，让其处理。

## 软件缺陷预测

软件缺陷预测是一种通过挖掘软件仓库，获取软件的特征，从而对软件的缺陷数量和缺陷类型做出预测的技术。

### 缺陷预测的概念和基本流程

软件缺陷预测主要会涉及到以下几个概念：

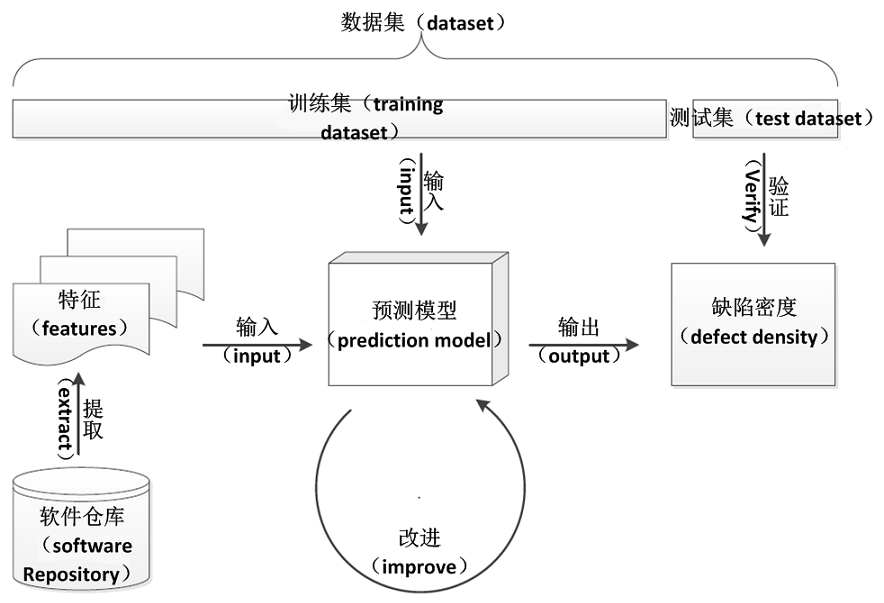
1. 软件缺陷：软件缺陷（Software Defect）在ISO 9000 中定义为“未能满足与预期或者规定用途有关的要求”，它包括检测缺陷和残留缺陷。软件缺陷的属性包括标识、类型、严重程度、优先级、起源、来源和根源。
2. 软件仓库：软件仓库（Software Repository）是用来存放软件开发过程中所有数据的储存介质，通常有软件配置管理系统、软件缺陷跟踪系统和项目管理软件系统三类。
3. 软件特征：软件特征（Software Features）是反映软件特征和质量的一系列参数，通常可以分为源代码特征、变更特征、缺陷历史特征等。
4. 预测模型：缺陷预测模型（Defect Prediction Model）是指以各种软件特征作为输入，以将来版本中缺陷数量作为输出的一套算法。

目前，缺陷预测模型分为机器学习和统计方法两大类[50]。基于软件仓库的缺陷预测过程一般分为三个主要步骤，如图2-1所示：

第一步、提取特征，从软件仓库中提取出用于预测的特征，包括源代码特征、变更特征、缺陷历史特征等；

第二步、预测缺陷，即将挖掘出来的特征输入预测模型，预测模型输出缺陷密度预测；

第三步，验证结果，为了验证预测结果的有效性，一般将数据集分为训练数据集和测试数据集，分别用于预测模型学习和验证预测结果。



1. 缺陷预测基本流程
2. The Basic Flow of Defect Prediction

### 缺陷预测特征

目前预测模型和算法的研究已经相对成熟，预测特征的选择和计算成为提高缺陷预测效果的关键。研究人员研究了软件系统中多种质量影响属性作为预测特征，包括代码复杂性，变更/处理过程复杂性[53]，软件模块间的语义依赖等等[26]。常用的特征可分为以下四类[23, 43]：

1. 变更特征（Change）- 源代码变更导致缺陷的产生。
2. 历史缺陷（Previous defects）- 过去的缺陷能有效地预测将来的缺陷。
3. 源代码特征（Source code）- 复杂的模块更难适应变更，也更容易产生缺陷。
4. 变更熵（Entropy of changes）- 复杂的变更更易产生缺陷。

此外，还有一些新颖的源代码特征，例如代码扰乱度（Churn）、源代码熵（Entropy）、源代码的主题[24] 等。

虽然随着软件规模的日益增大，软件开发越来越呈现出社会化特征，但在软件预测方面，社会化特征的研究相对较少。Wolf[6] 提出了对开发者交流进行社会网络分析的方法提取社会网络特征，有效预测了构建与集成的缺陷。Martin[18] 利用开发者与软件模块的关联网络中找出了预测缺陷的关键特征。Nicolas[19] 和Zanetti[25] 从开发社区组织的中提取出社会化结构，并实验验证了其与缺陷的相关性。

综上所述，目前现有的缺陷预测研究工作广泛使用产品与过程度量手段，从软件源代码以及其变更与缺陷历史中获取特征。然而，源代码是软件开发者通过一系列协作活动得到的成果，如何使用作为主体的开发者之间的社会关系信息来预测软件缺陷，研究其对软件质量的影响，具有重要意义。

近来，研究者开始了解到社会网络、工作依赖、日常事务处理等活动的错综复杂的性质，从而推断其与软件产品质量的相关性。对这些活动的跟踪可以从开发者每日使用的软件仓库中获得信息，如版本存档、事件追踪系统、邮件通讯存档。基于社会软件工程的缺陷预测的研究涉及开发者与用户的社会化结构与软件质量的关系，以及开发者与用户的交流内容与软件质量的关系。基于以上数据与猜想，本文利用社会网络分析等社会软件工程研究方法，建立模型，分析基于社会软件工程的测量特征与软件缺陷依赖关系，以达到研究目的。

## 软件仓库挖掘技术

数据挖掘，又称为数据库中知识发现（Knowledge Discovery in Database， KDD），是一个从大量数据中抽取出多种潜在信息（如知识规则、约束、规律等）的复杂过程[27]。这里软件仓库是软件开发过程中所有相关数据的综合，可以提供海量信息，并能直观的反映出软件系统开发过程中所经历的各条演化路径。软件仓库通常包括历史数据仓库，如数据控制仓库、缺陷仓库；运行记录数据仓库，如部署日志；代码仓库如软件源代码管理仓库等。软件仓库挖掘可用于软件开发的规律发现、过程改进、估量与预测、项目监控等[28]，这里主要阐述与软件缺陷预测相关的软件仓库挖掘技术。

软件仓库的常见数据源包括[40]：

1. 版本控制系统：即CVS、SVN等源代码仓库。
2. 缺陷跟踪系统：主要对缺陷报告进行记录与状态管理，如Bugzilla。
3. 项目管理系统：主要对项目的进度、资源以及开发过程进行管理，例如微软的Project，开源产品Redmine平台、JIRA平台等。
4. 其它：项目的邮件列表，即时消息，开发者论坛等。

软件仓库是软件开发过程中所有相关数据的总和，它包括软件系统的源代码、文档、缺陷记录及软件开发实施过程数据等。采用适当的数据挖掘的方法对其中各项数据进行挖掘可以发现其关联关系，预测缺陷，了解软件复用模式以及演化过程等等。从社会化角度进行挖掘，软件仓库也能反映出开发者交流与协作的行为模式。充分利用挖掘所得到的各项信息能对软件开发与维护提供有效协助。

## 本章小结

本章对本文的两个研究主题――社会软件工程与软件缺陷预测进行了详细综述，对其概念进行说明，并对当前国内外研究现状进行了总结。从缺陷预测技术的特征提取步骤的分析中提出了引入社会软件工程特征的观点，对其可行性与意义进行了分析，进而奠定了基于社会软件工程的软件缺陷预测的研究思路。最后对软件仓库挖掘技术进行简要综述，这是研究数据的来源，其技术细节会在下文研究过程中根据具体的软件仓库进一步涉及。

# 基于开发者邮件列表的缺陷预测新特征研究

目前软件缺陷预测模型和算法的研究已经相对成熟，预测特征的选择和计算成为提高缺陷预测效果的关键。本文第三章和第四章将从社会软件工程的角度挖掘与软件质量相关的新特征，第五章将提出预测模型，基于新特征进行缺陷预测。

为提取新的预测特征，本章以邮件列表作为研究对象。利用软件开发相关人员之间的交流信息进行软件缺陷预测已经被研究与验证，然而研究中对软件质量与开发者邮件列表之间关联的探究较少。本章将针对如何通过挖掘软件开发者之间的邮件列表来预测缺陷进行研究，将缺陷跟踪信息与开发者邮件进行关联，采用社会网络分析和文本分析技术挖掘邮件中的结构化与非结构化信息，从邮件的结构、正负面情感和主题三个方面开展探索实验，发现和代码潜在缺陷有较强相关性的交流特征。

## 研究方法

为了探索邮件列表交流信息对软件质量的影响并利用其进行软件缺陷预测，本文分为以下五个步骤开展深入的研究：

第一步，将邮件与源代码进行关联，定义软件数据之间的连接模式，作为分析与验证的基础，参见3.2.1节。

第二步，探索邮件列表与软件缺陷之间的关系。文中尝试回答以下三个研究问题来协助分析：

RQ1.具有特定结构的邮件列表是否与软件缺陷有所关联？其研究探索参见3.2.2节和3.3.2节。

RQ2.开发者情感是否与软件质量相关？其研究探索参见3.2.3节和3.3.3节。

RQ3.具有什么样的关注点的邮件更可能与缺陷关联？其研究探索参见3.2.4节和3.3.4节。

第三步，基于上述三个研究问题的探索和实验发现，提炼出关于邮件背后的交流行为的新特征，参见3.4节。

第四步，基于交流新特征，提出基于社会软件工程的软件缺陷预测新模型，参见5.1节和5.2节。

第五步，开展一系列缺陷预测实验，与现有预测模型进行对比并不断改进，经过特征结合与方法优化，提高新模型的预测效果，参见5.4节。

## 开发者邮件列表的数据分析与挖掘

本节将从多角度对邮件列表进行分析与挖掘，对邮件与源代码进行关联，然后就RQ1、RQ2、RQ3分别进行内容与网络组织结构、正反面情感和主题模型分析，并展开实验探究，挖掘其中的规律，为特征提取以及缺陷预测做准备。

### 关联邮件与源代码

邮件列表等非代码软件工件中蕴含着大量软件开发信息，将邮件列表显式地关联到源代码，能更好地进行社会软件工程分析。软件开发过程中和缺陷相关的三个核心工件――缺陷跟踪信息（如Bugzilla中的条目）、版本控制信息（如svn源码库）、以及开发者交流信息（如newsgroups和邮件列表）的关联数据图如图3-1所示，其中缺陷跟踪库中的实体能够作为原点，因为它们代表项目中工作的逻辑单位；作为对实体的响应，代码产生并被检入到源码版本库；而交流信息通常包含邮件条目，或与已有实体相关，或导致新实体的产生。



1. 工件关联模式
2. Artifact Linkage Schema

在这些工件的关联关系中，源代码与邮件消息之间显性连接是最难的，现有两种抽取方法，基于正则表达式的轻量级方法和采用信息检索技术的重量级方法[29]。后者具体可采用向量空间模型（Vector Space Model，VSM）和隐式语义索引（Latent Semantic Indexing，LSI）来实现。

向量空间模型（或词组向量模型）由Salton等人提出，并成功地应用于SMART文本检索系统。模型基本思想为将对文本内容的处理与分析抽象为[向量空间](http://baike.baidu.com/view/327493.htm)中的向量运算，把语义的相似度转化为向量空间上的相似度。文本处理中最常用的[相似性度量](http://baike.baidu.com/view/3839126.htm)方式是余弦距离，余弦可通过向量内积/各个向量的模的乘积进行计算。将邮件与源文件作为文档被表示为空间向量，就可以通过计算向量之间的相似性来评估邮件与源文件相关性。将文档（邮件）视为索引词（关键词）形成多元向量空间，输入检索词对其进行搜寻。然后通过比较每个文档（向量）和检索词（向量）的夹角偏差程度来得到文档和搜索词的相关程度。

隐式语义索引LSI提出的目的基于一种假设：即在千变万化的各种词的组合下隐藏着语义结构，可以通过某种方式来估计这种结构并去除噪音。噪音的来源是自然语言中大部分词存在一词多义的特点，机器算法无法确定在何环境下使用何种词义。考虑到自然语言理解的复杂性，隐式语义索引选择以大数量样本的统计分析找出不同的词（词组、短语）间的相关性，将这种相关性作为特定语义结构应用于检索。

相较于这两种复杂方法，由于轻量级方法易于实现而且精确度也相对够高，并且针对邮件列表与源文件的关联，复杂方法的优势不能充分发挥，所以本文通过正则表达式从实体名称、大小写以及标点等方面来对邮件内容中的源代码信息进行筛选，进而确定其关联。例如，以下表达式可用来简单分析java语言的项目中邮件与文件或类的连接：

Eg. (.\*) (\s|\.|\\|/) <packageTail> (\.|\\|/) < EntityName > ((\.(java|class))|(\s))+ (.\*)

以下三小节将在此关联关系的基础上分别针对RQ1、RQ2和RQ3三个问题对邮件列表进行分析。

### 挖掘邮件列表结构

针对RQ1，本文定义了两种邮件列表的结构：“内容结构”与“链结构”。前者表示邮件正文的内部组成结构，后者表示邮件外在的结构关系。

现举例说明，图3-2是一条典型的邮件消息的示例。A、B、C区域表示三封邮件，其中C为原邮件，B是对C的回复邮件，A是对B的回复，形成简单的回复链。C邮件（被框出）的邮件头邮件体在D区被展示出。



1. 消息示例
2. Message Sample

邮件正文通常有多种内容混合，这些内容可以被归为4类（自然语言、代码、堆栈跟踪、其他文本）[30]，在图中用不同的图标表示。这些不同类型内容的组合形成了邮件正文的内容结构。图3-2中D区展示的邮件正文的内容结构即为：自然语言\*6、代码\*16、堆栈跟踪\*7、其他文本\*3，其数字是该类内容所占的物理行数。

消息往往不是独立的，每封邮件都具有发送者，邮件之间也存在回复关系。发送者与邮件的关联以及邮件之间的回复关系就形成了邮件列表的链结构，如图3-3所示。本文选择采用网络图来描述邮件列表的链结构特征。每封邮件都被其回复以有向边指向，邮件节点也由无向边链接到其发送者，这样各条邮件回复链之间的关系也被表示出来。邮件链结构的特征即为其网络图特征，例如度中心性、紧密度中心性、中介中心性等。



1. 消息链结构的网络图
2. Message Thread Structure Network

本文将对邮件的内容结构和链结构的特征进行实验考查（参见3.3.2），探索它们与缺陷的联系与规律，并确定其能否成为缺陷的预测特征。

### LIWC 分析

本文采用语言获得和词汇计数（LIWC）工具来协助分析邮件文本，得到邮件中的开发者情感特征，研究其与缺陷的关系，进而回答RQ2。

LIWC是由James W. Pennebaker等人开发的一款文本分析软件，能够在广泛的文本范围中计算人们对各种类别的单词的使用程度。LIWC 使用字典的方式将单词划分成一些基本的语言学以及心理学的维度，通过计数字典中所包含的单词数，将文本中每个单词归为其中一个或者多个维度中。其中，LIWC所使用的基于心理测量学的字典，被Pennebaker 等人应用于多种实验，其效果已经得到验证[31]。本文采用LIWC工具对邮件进行心理特征分析，表3-1作为样例展示了LIWC工具针对一封邮件的正文的分析结果。其中有关于文本积极情绪与消极情绪的分析结果，将被记录并作进一步研究。

1. LIWC分析结果示例
2. LIWC Analysis Sample

|  |  |  |  |
| --- | --- | --- | --- |
| LIWC 维度(Dimension) | 分析结果 | 私人文本 | 正式文本 |
| 自我认指（Self-references: I, me, my） | 0.00 | 11.4 | 4.2 |
| 社交词汇（Social words） | 5.04 | 9.5 | 8.0 |
| 积极情感（Positive emotions） | 4.20 | 2.7 | 2.6 |
| 消极情感（Negative emotions） | 0.00 | 2.6 | 1.6 |
| 冠词（Articles: a, an, the） | 9.24 | 5.0 | 7.2 |
| 长单词（Big words: > 6 letters） | 38.66 | 13.1 | 19.6 |

LIWC 是基于单词或微粒的计数功能。但是，如果只是对单词计数，则文本中的情感将会丢失，因为单词计数程序不会捕捉讽刺、反语甚至单词在上下文语境中的意义。LIWC 则使用了一个基于心理测量学的字典，字典已经被Pennebaker 等人用于多种实验，其效果已经得到验证，其中包含70个维度，超过2300个单词与词干。

这一过程主要目标将LIWC 工具作为预报器和分类器来理解出软件开发中的纷繁复杂之处。LIWC 工具以如下的方式工作：像maybe 这样的词会与迟疑性相关联，而像important 这样的词会与确定性相关联，等等。LIWC 工具只是简单地计数字典中所包含的单词数。字典被划分成一些基本的语言学以及心理学的维度，使得每个单词都会被归为其中一个或者多个维度中。本文只使用LIWC 中关于心理学的部分功能来分析eclipse 开发者邮件列表，将邮件文本作为输入，输出则是文本中所包含的情感。

本文将从情感的角度出发，计算邮件内容的积极情绪特征和消极情绪特征值，进而进行实验考查（参见3.3.3），探索情感与缺陷的联系与规律，并确定这一属性能否成为缺陷的预测特征。

### 主题分析

关于RQ3，本文采用基于主题的方法分析邮件的关注点。从机器学习领域进行文档主题提取的自动化文本挖掘方法已经得到广泛研究[32]，其基础思想是通过识别单词在语料库中的共同出现频率进而发掘文档潜在结构。开发人员的邮件列表是在开发不同模块时留下的，邮件中势必会存在构建相关的关注点，而某些话题关注点被发现具有缺陷倾向。举例来说，当开发者们讨论某一构件中潜在的缺陷时，相关邮件的主题模型通常会具有相似性。

概率主题模型[33]是一系列旨在发现隐藏在大规模文档（document）中的主题结构的算法。主题建模算法是一种统计方法，它通过分析原文本中的词语（word）以发现蕴藏于其中的主题（topic），主题间的联系，以及主题随时间的演变。

判断两个文档相似性的传统方法是通过计算两个文档共同出现的单词的多少，如TF-IDF等。但是这种方法无法分析出文字背后的语义关联，因为只自然语言中，不能单凭词语相似度确定语义的异同。而主题模型可以做到语义分析，在主题模型中，主题代表一个概念或一个方面，用系列相关的单词来表示。主题模型认为在文档生成的过程中，每个词都是通过“文档以一定概率选择了某个主题，并从这个主题中以一定概率选择某个词语”这样一个过程得到。其中每个词语出现的概率为：

…… (3-1)

这个概率公式(3-1) 表示这些单词的条件概率，可以用“文档 - 词语” （document-word） 矩阵表示：

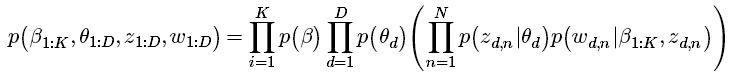


其中“文档—词语” 矩阵表示每个文档中每个单词的出现概率；“主题—词语” 矩阵表示每个主题中每个单词的出现概率；“文档—主题” 矩阵表示每个文档中每个主题的出现概率。利用主题模型从非结构邮件列表中挖掘潜在信息的一个主要优势是在模型构建过程中不需要对数据进行训练，另一方面主题模型也省去了复杂的人力标记过程。

本文采用潜在狄立克雷分配（Latent Dirichlet Allocation, LDA）[34] 主题模型来识别邮件文本中的主题。基于所识别的邮件主题，本文将通过实验来探索其与缺陷的联系与规律（参见3.3.4），并确定这一属性能否成为缺陷的预测特征。LDA 用来模拟同一语料库中的文档通过部分或全部主题以及各主题下分布的词语的生成。假设所有的主题在文档生成之前就已经产生且指定，并将主题定义为固定词语的概率分布，则可通过如下过程生成文档：

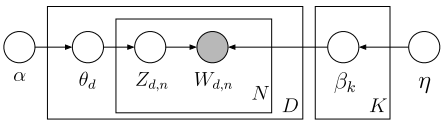
1. 随机产生一个主题分布；
2. 对文档中的每个词，(a)从第一步产生的分布里随机选择一个主题，(b) 从主题对应的词语的概率分布中随机选择一个词。

从文档的生成过程来看，第1) 步的目的是将文档由不同主题以不同比例组成，所产生的主题分布作为狄立克雷分布，作用是将文档中的词分配给不同的主题。第2) 步使各文档中每个词是从分布里的一个随机选择的主题中所得到。对这一过程形式化地定义，得出LDA的生成过程对应的观测变量和隐藏变量的联合分布如下[34]：

…… (3-2)

式中β1:K表示全部主题，βk表示第k个主题的词的分布；第d个文档中主题所占的比例为θd，θd,k表示第k个主题在第d个文档中的比例；zd代表第d个文档的主题的全体，zd,n表示第d个文档（wd）中第n个词（wd,n）的主题，其中每个词都是固定的词汇表中的元素。

基于生成过程的统计假设，形成联合分布的特定数学形式，以此表示出作为LDA本质的变量之间的相互依赖关系。变量分布也可表示为LDA的概率图模型（如图3-4所示）。



1. LDA的图模型
2. Graph Model of LDA

## 探索实验及发现

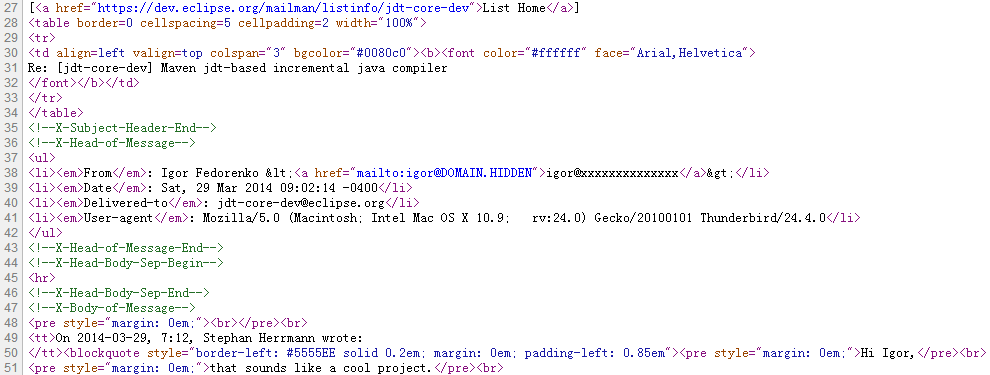
### 实验数据

本文中研究用到的数据集是从eclipse 的多个模块中得到的，其邮件列表由Mailman 管理。Mailman 提供了一种网页视图，本文采用Html parser 的java 库来解析网页的html 代码，以此来从网页中获取邮件内容。所获得的邮件分别记录其对应的作者、日期、所属模块等信息。然后通过轻量级的正则表达式解析方式把这些邮件连接到相应的程序文件或类，并过滤掉没有连接到的邮件。表3-2为通过以上预处理的数据集及其相关信息。表中和源代码关联的邮件比例为13.9%~19.8%，并对此结果进行了人工取样分析，误联率为0%，漏联率低于5%，自动关联的正确率达到要求。

1. 数据集信息
2. Information of Data Sets

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 构件 | 邮件数 | 连接邮件 | 连接率 | 描述 |
| jdt-dev | 751 | 117 | 15.6% | Eclipse JDT general developers list |
| jwt-dev | 1644 | 231 | 13.9% | Java Workflow Tool developers list |
| eclipse-dev | 9807 | 1944 | 19.8% | General development mailing list of Eclipse project |

获取邮件内容的方法如下，通过查看Mailman邮件网页视图html源码，如图3-5，找到代表发件人，发送日期，收件人，邮件正文等信息的元素，分析其上下文找出与之相关联的标签，根据标签信息，将邮件URL输入程序，即可通过HTML Parser提供的解析方法提取出所需信息，进一步可将其进行存储作为后续研究基础。



1. 邮件网页视图html源码
2. HTML Source Code of Email Webpage View

而想要做到批量处理邮件列表，则可采用网络爬虫工具实现将邮件列表中各邮件的URL获取，本文选取SoukeyMiner工具爬取邮件列表URL。相关代码如下：

**public** **void** handleSimpleTag(HTML.Tag tag, MutableAttributeSet attribs, **int** pos) {

**if**(from == **true** && date == **true** && mailText == **false** && tag == HTML.Tag.*HR*) {

mailText = **true**;

} **else** **if**(tag == HTML.Tag.*HR* && mailText == **true**) {

from = **false**;

date = **false**;

mailText = **false**;

}

}

**public** **void** handleText(**char**[] text, **int** pos) {

**if**(from == **true** && date == **false**) {

String tempParagraphText = **new** String(text);

//System.out.println("FROM - " + tempParagraphText + "\n");

sb.append(tempParagraphText + "###");

} **else** **if**(start == **true** && from == **true** && date == **true** && mailText == **false**) {

String tempParagraphText = **new** String(text);

//System.out.println("DATE - " + tempParagraphText + "\n");

sb.append(tempParagraphText + "###");

} **else** **if**(mailText == **true**) {

String tempParagraphText = **new** String(text);

//System.out.println("TEXT - " + tempParagraphText + "\n");

sb.append(tempParagraphText);

}

}

### 结构分析实验及发现

实验中针对邮件内容结构选取两类文本类型进行实验，即代码文本与堆栈跟踪文本。这两种文本都可能与缺陷相关，因为在多数邮件中，作者加入代码和堆栈跟踪的文本的目的是描述项目中遇到的问题，而这些问题在不久之后就可能引发新的缺陷。

表3-3和表3-4分别展示了两种内容结构与缺陷的相关性的实验结果，通过内容所占比率与文件缺陷率体现出来。表中，每种内容的比率是单纯通过计算文本行数得出的。其中所占比率排名前10的邮件被分别排序得出并分析。结果显示出代码行数所占比率较高的邮件，其关联文件缺陷率也较高（高于表格最后一栏的平均水平），堆栈行数文本比率高的邮件情况也如此。

1. Top10代码率邮件关联文件缺陷率
2. Defective Rate of Top Ten Code Rate Mails

|  |  |  |  |
| --- | --- | --- | --- |
| 构件 | 代码文本比率 | 文件缺陷率 | 平均缺陷率 |
| jdt-dev | 71.3% | 70% | 35.0% |
| jwt-dev | 68.2% | 60% | 24.9% |
| eclipse-dev | 64.7% | 70% | 21.5% |

1. Top10堆栈跟踪率邮件关联文件缺陷率
2. Defective Rate of Top Ten Stack Tracing Rate Mails

|  |  |  |  |
| --- | --- | --- | --- |
| 构件 | 堆栈跟踪文本比率 | 文件缺陷率 | 平均缺陷率 |
| jdt-dev | 47.1% | 80% | 35.0% |
| jwt-dev | 40.5% | 60% | 24.9% |
| eclipse-dev | 44.7% | 70% | 21.5% |

接下来采用网络图分析方法研究邮件链结构。针对每个构建都将收集一定时期内的邮件建立网络图。经过一系列评估之后，选择采用中介中心性特征（betweenness centrality），任一点的中介中心性相当于网络图中各顶点到其他各点的最短路径之中经过此点的数量，计算公式如下：

…………………… (3-3)

其中指从节点s 到节点t 的最短路径总数，指这些路径中经过节点的数量。所有节点的中介中心性都将被计算出，不过其中的发送者节点的中心性将不被考虑，因为只有邮件节点会被连接到源文件。邮件会通过其中心性进行排序，然后选出前10位进行分析（见表3-5）。通过对这些邮件所连接到的文件的缺陷率进行计算，最终发现在多数情况下，网络图中有较高中介中心性的邮件节点，也有较高概率对应于缺陷文件。

1. Top10中心性邮件关联文件缺陷率
2. Defective Rate of Top Ten Centrality Mails

|  |  |  |  |
| --- | --- | --- | --- |
| 构件 | 平均中介中心性 | 文件缺陷率 | 平均缺陷率 |
| jdt-dev | 0.47 | 60% | 35.0% |
| jwt-dev | 0.52 | 50% | 24.9% |
| eclipse-dev | 0.62 | 70% | 21.5% |

从实验结果看来RQ1的答案是肯定的。缺陷与邮件列表的特定结构是相关的，其中分别包括内容结构和邮件链结构。就前者来说，开发者通过不同的邮件内容表达不同的目的，正如代码文本与堆栈跟踪文本，尤其堆栈跟踪文本，通常与程序缺陷或问题相关。就后者来说，中介中心性代表节点被其他任意两点间的最短路径经过的可能性。由此可以认为节点的中心性代表其重要程度。这些关键节点之所以与缺陷相关是由于其对整个系统具有更强的影响；从另一方面考虑，越严重的问题也将会被更多的开发者所关注，提高其中心性。综上所述，邮件列表的结构特征反映了邮件本身的性质，特定的结构确实会预示缺陷。

### 情感分析实验及发现

本实验采用LIWC工具对各邮件正文依次分析，并利用Selenium IDE实现批处理。LIWC分析结果中的积极情绪“Positive emotions”与消极情绪“Negative emotions”将被着重考量。经过大量取样分析，设置了一个阈值来界定文本中的情感是相对积极还是相对消极。

1. 不同情绪邮件关联文件的缺陷率
2. Defective Rate of Emails with Different Emotions

|  |  |  |
| --- | --- | --- |
| 构件 | 低于阈值缺陷率 | 高于阈值缺陷率 |
| jdt-dev | 70% | 20% |
| jwt-dev | 70% | 10% |
| eclipse-dev | 60% | 20% |

表3-6显示的是针对RQ2的实验研究结果。实验采用“Positive emotions”值与“Negative emotions”值的差值作为阈值（设置为2）。从结果中可以看出，情绪差值高于阈值的邮件一般连接到的文件缺陷率较低，而情绪差值低于阈值的邮件会连接到较多的缺陷文件。

此时RQ2已经可以被回答，开发者情感与软件质量之间确实存在关联。不过，这一规律并不总是成立，在实验中存在一定的不确定性。这其中的原因是多样的：首先自然语言的语义分析较为复杂，而且开发者的感情表达也将比想象得更加复杂；其次一些关键词或者变量名称例如“if”、“new”等将会对结果有所干扰；同时应考虑分析工具所限。当然从整体结果看来情感与缺陷仍是相关的。

### 主题分析实验及发现

本文采用主题模型来分析邮件的关注点，并以此对此问题进行研究。基于LDA 模型，主题是通过语料库中单词的共现频率来收集产生的。在表3-7中展示的便是从eclipse JDT 构建的邮件列表中提取出的5个主题。每一列表示一个主题，共现频率最高的前10个单词也一并列出，每一主题都有其自身的关注点。

1. Eclipse JDT构件邮件列表中提取的5个主题
2. Five topics Extracted From Eclipse JDT Mailing List

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Rank* | *Topic 1* | *Topic 2* | *Topic 3* | *Topic 4* | *Topic 5* |
| 1 | java | mailing | jdt | eclipse | java |
| 2 | working | class | compiler | completion | code |
| 3 | problem | version | method | point | line |
| 4 | annotation | find | public | work | project |
| 5 | extends | jdt-core-  dev | change | type | files |
| 6 | variable | default | bug | create | jdt-core-  dev |
| 7 | time | context | source | build | extension |
| 8 | list | classpath | error | void | file |
| 9 | functionality | support | code | way | core |
| 10 | args | respond | default | static | found |

预处理阶段，对每封邮件中使用不标准的单词应首先被拆分或者删除。此时每封邮件可被看作是单词的集合，而整个系统则是邮件的集合。然后对整个集合采用LDA 技术进行分析，其中设置主题数K=5，迭代次数R=50，以及两个超参数（hyper parameters）的值（α= 0.5，β= 0.1）。通过以此发现的整体主题结构，可将各邮件关联到其所属的主题[24]。分析中也会得到各邮件中被分配到每一主题的单词数量，这将被用来作为基于主题的特征。

接下来计算并分析基于主题的特征与邮件相关程序文件的缺陷之间的相关性。如表3-8所示，T1~T5代表5类主题，可以发现基于主题的不同特征与缺陷数量之间的相关程度有不同级别，表现为缺陷率越高，相关性越强。由此看来，T3有相对较高的相关性而T1相对较低。这表示主题3较主题1来说与缺陷发生相关性更强，即关注主题3的邮件，其关联的程序文件更有可能产生缺陷。从表3-7的主题3中可以发现其关注点为本构件中的源代码以及缺陷等。因此可以设想在对Eclipse JDT 的讨论中，缺陷讨论主题（T3）更加具有缺陷倾向。本文的第三个研究问题也以此得到回答。

1. 基于主题的特征与缺陷的相关性
2. Correlation of Topic-based Features and Defects

|  |  |  |
| --- | --- | --- |
| 主题特征 | 缺陷文件数 | 文件缺陷率 |
| T1 | 4 | 0.10 |
| T2 | 8 | 0.20 |
| T3 | 13 | 0.32 |
| T4 | 7 | 0.17 |
| T5 | 9 | 0.22 |

实验结果证实了从邮件获得的特定构件的关注点从不同程度上与缺陷有关。然而，利用主题相关的特征预测缺陷的有效方法仍不明确。尽管能够从jdt 邮件列表中提取出5类主题，但具体每个主题的关注点难以表述清除。就结果来说，表3-8中的T2，T4，T5的内容有较强的相似性，换句话说，主题之间的区分程度并不高。造成这一问题的一个可能原因是在将邮件连接到文件之后，所留下进行分析的邮件数量较少，并具有一些共同之处。因此，在以后研究中，需要一个更加敏感的主题模型分析方法。

## 基于邮件列表的新特征

根据对邮件列表的分析和研究的发现，本文从邮件内容组成、网络组织中心性、情感特征以及主题结构角度提出具有社会化性质的新特征：

1. 邮件内容特征（Mail Content Features, MCF）：特定的邮件内容类型隐藏着潜在的缺陷。
2. 邮件网络特征（Mail Network Features, MNF）：核心开发者间的关键交流邮件会包含更多缺陷。
3. 情感特征（Emotion Features, EF）：邮件中的消极情感预示着缺陷产生。
4. 基于主题的特征（Topic-based Features, TF）：邮件中的特定主题预示了缺陷产生。

其中MCF 和MNF 由RQ1而设计，EF 和TF 分别由RQ2与RQ3而设计。以上4类新特征被细化成若干可计算的子特征，定义于表3-9中。

LOCF 和LOSTF 可通过计算相关内容行数得到。ROCF 和ROSTF 通过计算内容行数所占总行数比例得到。

DCNF 表示节点的度中心性，对给定图G:=(V, E)， 定点数|V|， 边数|E|， DCNF 等于deg (v)值。BCNF 计算参见3.3.2。

ESF 和EDF 分别代表Positive emotions 和 Negative emotions 值的和与差。之所以不单独使用Positive emotions 和Negative emotions 值，是因为其数值在不同邮件中差异较大。

TF1~TF5 表示各邮件属于T1~T5的概率，T1~T5是从JDT 邮件列表中提取的五个主题。

1. 特征定义及类型
2. Definitions and Types of Features

|  |  |  |
| --- | --- | --- |
| 特征 | 定义 | 类型 |
| LOCF | 邮件中代码文本行数(Line of Code) | Mail Content Features (MCF) |
| LOSTF | 邮件中堆栈跟踪文本行数(Line of Stack Trace) |
| ROCF | 邮件中代码文本比率(Rate of Code) |
| ROSTF | 邮件中堆栈跟踪文本比率(Rate of Stack Trace) |
| DCNF | 邮件网络中的度中心性  (Degree Centrality of Mail Network) | Mail Network Features (MNF) |
| BCNF | 邮件网络中中介中心性  (Betweenness centrality of Mail Network) |
| ESF | 邮件中情绪和值(Emotion Sum) | Emotion Features (EF) |
| EDF | 邮件中情绪差值(Emotion Difference) |
| TF1~TF5 | 前五种基于主题的特征  (Top Five Topic-based Features) | Topic-based Features (TF) |

## 本章小结

本章将针对如何通过挖掘软件开发者之间的邮件列表来预测缺陷进行研究，将缺陷跟踪信息与开发者邮件进行关联，采用社会网络分析和文本分析技术挖掘邮件中的结构化与非结构化信息。从邮件的结构、正负面情感和主题三个方面开展探索实验，发现和代码潜在缺陷有较强相关性的交流特征，为改进软件缺陷预测模型，提高预测的准确性做准备。

# 基于社会网络分析的缺陷预测新特征研究

本章将从社会网络分析角度探索社会软件工程与软件质量的关系。为了从传统的分析方式中得到新的缺陷预测特征，本章首先从社会网络图的构建角度进行改进，然后对改进后的网络图进行分析；其次对社会网络进行扩展，从开发者地位与影响，项目组织结构等角度研究社会软件工程，并将其归结为社会网络分析问题，提取出新的缺陷预测特征。

## 研究方法

本文将采取以下五个步骤基于社会网络分析方法探究社会软件工程中缺陷预测的特征并应用于缺陷预测：

第一步，根据社会网络分析方法，提出社会网络度量指标，构建社会网络，然后对网络图进行改进，参见4.2节。

第二步，探索开发者地位与影响，项目组织结构，并通过社会网络分析方法进行研究，参见4.3节和4.4节。

第三步，基于上述问题的探索和实验发现，提炼出基于社会网络的新特征，参见4.5节和4.6节。

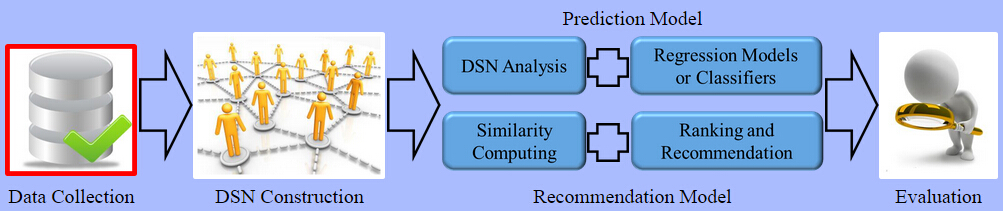
第四步，基于社会网络新特征，提出基于社会软件工程的软件缺陷预测新模型，参见5.1节和5.2节。

第五步，开展一系列缺陷预测实验，与现有预测模型进行对比并不断改进，经过特征结合与方法优化，提高新模型的预测效果，参见5.4节。

## 社会网络图设计与分析

### 社会网络分析方法

社会网络分析（Social Network analysis）是对社会网络的系统化的分析。社会网络分析从网络理论的角度观察社会关系，网络由节点（代表网络中的单独个体）和连结（代表个体之间的关系）组成。这些网络通常能够以社会网络图的形式描绘，其中节点和连结以点和线的形式表示出来[47]。



1. 社会网络分析的一般过程
2. General process of social network analysis

社会网络分析的一般过程如图4-1所示，各个步骤主要关注的问题如下：

1. 数据收集（Data Collection）：主要关注实验数据的来源，收集什么类型的数据集以及收集什么项目的数据。
2. 开发者社会网络构建（DSN Construction）：主要考虑哪些开发者社会网络类型要被构建，应用于哪些实验项目，相应的构建方法是什么。
3. 开发者社会网络分析（DSN Analysis）：主要考虑选取哪些社会网络分析特征来进行分析，针对特定的开发者社会网络分析特征采取的什么社会网络构建方法。
4. 开发者社会网络应用（DSN Application）：主要考虑开发者社会网络将被采用怎样的应用，如果被应用于缺陷预测时什么样的回归模型或者分类方式会被使用，如果被应用于推荐模型时将采取怎样的算法来计算相似度。
5. 评估（Evaluation）：主要考虑选取怎样的标准对开发者社会网络应用进行评估。

### 社会网络图构建与度量

1. 数据收集

为构建网络图，首先要进行数据收集。作为分析的基础，数据收集阶段主要从软件仓库中获取各项数据，包括版本管理系统（如SVN、Git等）中的日志信息；缺陷跟踪系统（如Bugzilla等）中的缺陷提出和修复信息；开发者交流信息等。关于交流信息，主要以邮件列表数据为主，前文已经涉及到，下面就其他信息的获取方法进行简单说明。

以eclipse的jdt为例，首先利用git的clone命令git clone （<git://git.eclipse.org/gitroot/jdt/eclipse.jdt.core.git>）将其clone至本地以方便分析。然后可利用log命令得到其提交过程中的日志信息，git log <file> # 查看该文件每次提交记录；git log --stat #查看提交统计信息。具体如下：

commit c660a0e6a2551567fa3f92ffe31ef9b52fdec6cc

Author: Harry Terkelsen <het@google.com>

Date: Tue Dec 10 11:50:16 2013 -0800

Fixes bug 423254 - There is no way to tell if a project's factory path

is different from the workspace default

Change-Id: Ia28cb895a17c327299920d56c65f1db6410218d7

Signed-off-by: Harry Terkelsen <het@google.com>

.../jdt/apt/core/internal/util/FactoryPath.java | 22 +++++++++++--

.../org/eclipse/jdt/apt/core/util/AptConfig.java | 14 ++++----

.../org/eclipse/jdt/apt/tests/RegressionTests.java | 38 ++++++++++++++++++++--

3 files changed, 64 insertions(+), 10 deletions(-)

然后可根据需要在此结构上提取提交时间、提交者、涉及文件、提交说明日志等信息。

关于缺陷相关数据，可利用开源工具从bugzilla仓库中获取。例如可通过引入b4j.jar，进行配置后，利用其提供的API获取bugzilla库中所需的缺陷信息。

**<?xml** version="1.0" encoding="ISO-8859-1"**?>**

**<bugzilla-session** class="b4j.core.session.BugzillaHttpSession"**>**

**<bugzilla-home>**http://your-bugzilla.your-domain.com/**<bugzilla-home>**

**<AuthorizationCallback>**

**<login>**your-name**</login>**

**<password>**your-password**</password>**

**</AuthorizationCallback>**

**<Issue** class="b4j.core.DefaultIssue"**</Issue>**

**<bugzilla-session>**

1. 社会网络图构建

基于从各数据源提取的信息，可以构建出多种网络图形。例如从程序分析角度可以构建各种源代码结构图，包括函数调用图和模块依赖/协作图等。而本文主要研究社会网络图，即将开发者社会关系考虑加入网络图的构建中，例如开发者交流图和开发者协作图等。关于开发者交流图，3.2.2节中邮件列表结构图可作为实例；开发者协作图则可根据对代码提交和对缺陷的修复行为进一步细分，通过基于提交的开发者协作图（Commit-based developer collaboration network）和基于缺陷的开发者协作图（Bug-based developer collaboration network）来实现。

基于提交的开发者协作图用于追溯开发者在开发事件中的合作方式。通过分析版本控制系统中的提交日志，本文采用在共同对同一文件进行提交的两个开发者D1与D2之间建立一条无向边的方式对协作图进行构建。

基于缺陷的开发者协作图用于追溯开发者在缺陷修复事件中的合作方式。根据缺陷的处理周期，考虑到当一个缺陷被分派给开发者D1时，D1没有能力解决或者由于其他原因将缺陷进一步分派给开发者D2。在这一情况下，本文通过在图中建立一条D1到D2的有向边的方式构建协作图。



1. 社会网络图演化关系（开发者协作图—演化—开发者贡献图）
2. Social Network Evolution (Developer Collaboration Network -Evolution- Developer Contribution Network)

从传统的源代码结构角度出发，通常采用函数调用图、模块依赖图等进行程序分析。而从社会软件工程角度出发，考虑到开发者的协作关系，将进行协作的开发者联系起来，形成基于缺陷的开发者协作图、基于提交的开发者协作图等。开发者贡献图则将开发者协作图与对应文件的依赖情况结合起来，如图4-2，这样既能够分析开发者协作情况，又可以发现协作情况受对应文件相互关系的影响。在开发者贡献图（图4-3）中，包含两种节点，开发者节点和文件节点；同时包含三种关系，开发者间的协作关系，文件间的依赖关系和开发者对文件的提交关系。这些关系作为网络图中的边，其权重可根据实验需求进行设置。



1. 开发者贡献图
2. Developer Contribution Network
3. 社会网络图度量

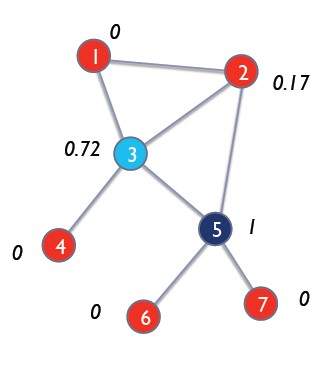
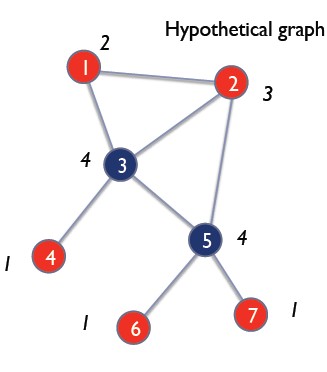
本文将主要采用开发者贡献图配合其他网络图研究缺陷预测特征。而为了对构建好的网络图进行分析，需要设计一些度量指标来反应网络图中所隐含的各项特征，相关的网络测量（Network measures）指标包括：

* 1. 网桥（Bridge）：一个个体自身的弱连接填充了结构洞（structural hole），成为两个个体或者集群之间的唯一的连接。它也包括由于消息失真或者交收失误的高风险性造成长路径的不可行时的最短路径。
  2. 中心性（Centrality）：中心性指的是通过一组指标来量化网络中的特定节点（或者群体）的“重要性”或者“影响力”，比较普遍的测量中心性的方法包括中介中心性（betweenness centrality），紧密度中心性（closeness centrality），特征向量中心性（eigenvector centrality），阿尔法中心性（alpha centrality）和度中心性（degree centrality）。

中心性度量应用较为普遍，Ehrlich[35] 等人通过中心性度量研究每个开发者的表现受通讯模式的影响，结果显示个人中心性对所在开发小组有益，对项目整体有害，团队闭包性对开发团队与项目整体均有益。中心性测量方法在表4-1中定义，图4-4为各项测量的示例。

1. 中心性测量方法
2. Measurement Method of Centrality

|  |  |
| --- | --- |
| **中心性测量** | **在社会网络中的解释** |
| 度中心性（Degree Centrality） | 这个人有多少人能直接到达？ |
| 中介中心性（Betweenness Centrality） | 这个人作为连接另外两个人的最短路径经过的节点的可能性有多大？ |
| 紧密度中心性（Closeness Centrality） | 这个人连接到达网络中每个人的速度有多快？ |
| 特征向量中心性（Eigenvector Centrality） | 这个人与其他连接性好的人之间的连接性如何？ |

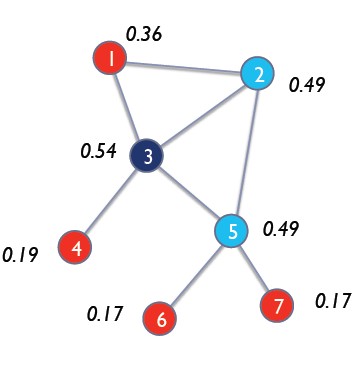
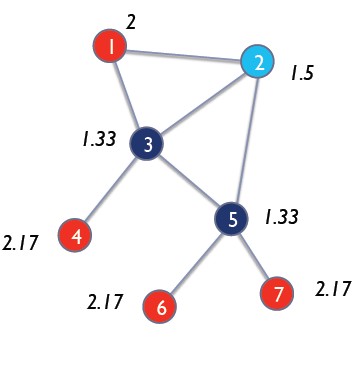


Eigenvector Centrality

Degree Centrality

Closeness Centrality

Betweenness Centrality



1. 中心性测量举例
2. Example of Centrality Measures
   1. 密度（Density）：网络中直接连结相对于可能的连结总数的比例。
   2. 距离（Distance）：连结两个特定个体所需的最小连结数量，由于Stanley Milgram的“小世界实验”和“六度分割理论”而普及。
   3. 结构洞（structural hole）：网络两部分之间连结的缺乏。发现并且利用结构洞能够帮助相关人员获得竞争优势，这一观念是被社会学家Ronald Burt开发，有时被称为是社会资产的替代概念。
   4. 连结强度（Tie Strength）：通过时间、情感强度、亲密度和相互作用的线性组合定义。强连接与同质性、邻近性、传递性相关，弱连接则与网桥相关。

## 开发者地位与影响探索

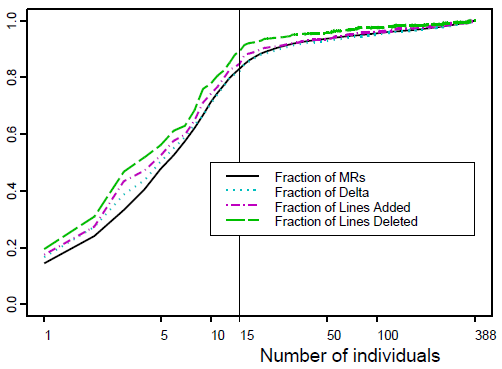
根据Omar Alonso[36] 等的定义，所谓“专家”，是指一段时间内对项目有大量贡献的开发人员。通过代码自身的关联性对代码进行模块划分，从而可以识别出各个模块中具有很高代码提交次数的开发人员，即为该模块的专家；同时对多个模块均有较高提交次数的开发人员被认定为团队中的通才，即核心成员。同时伴随着项目的进行，各模块的“专家”在不断发生变化，每个“专家”都有其特有的生命曲线[46]。

基于开发者贡献图，可以从开发者与文件连接的边的权值得出开发者对这一文件提交的行数，反映了贡献的多少。另一方面，由于开发者与其提交过的文件之间有边相连，故对某一开发者，连接到文件的边越多，说明其提交过的文件数量越多，也是贡献的反映；同时由于共同提交过同一文件的开发者之间会有边连接，所以提交过多个文件的开发者也往往会与多个其他开发者相连接。对于提交文件数量多，范围广的开发者可以看作核心开发者。反映到图中，可以通过中心性度量方法计算出中心性较强的开发者，这也可以作为核心开发者的判断依据。

为对核心开发者与非核心开发者的贡献和影响进行定量的研究，本文根据Apache的提交日志进行统计与分析，绘制了开发者贡献的分布图。

图4-5中绘制了累积的代码变更比例（纵坐标）相对于N个代码开发者（横坐标）的关系。从代码行数的增删（Lines added/deleted）和Delta（Unix diff算法计算出的代码变动）都可以反映出变更趋势。这里采用修改请求（Modification Request，MR）代表代码变更，CVS的提交事务代表一次基本的修改，这与商业开发环境中的修改请求类似。另外，每次提交都会自动的生成一封电子邮件信息，存储在apache-cvs档案中，这可以帮助重建CVS数据。从图中可看出前15个开发者提交了超过83%的MR和Delta，产生大约88%的代码行数添加与91%的行数删除。

非核心开发者贡献了非常少量的代码，以及相应的较小的工作量（为简单起见，可以将图中除前15个开发者之外的看作非核心开发者）。核心开发者生成的修改请求远大于非核心的团体。这其中的差异是在统计学上显著的。通过Kolmogorov-Smirnov 测试，修改请求的分布显著小于增加行数的分布（p<0.01），Kolmogorov-Smirnov 测试是采用经验分布函数的非参数检验。

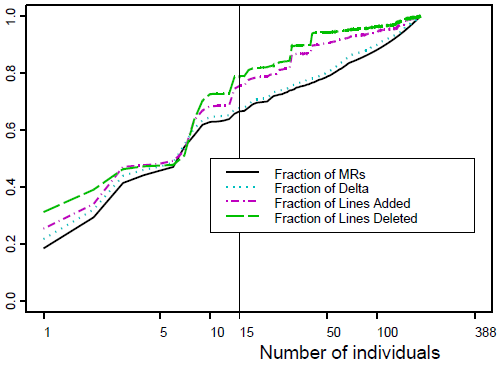


开发者数量

1. 代码贡献累计分布
2. Cumulative Distribution of Contributions to the Code Base

接下来单独从缺陷修复方面考虑，其贡献分布与代码提交存在较大的差异。通过对有182次修复样例相对于249次代码提交样例进行分析，采用图4-5中相同的开发者顺序，构建图4-6。通过与图4-5对比可发现代码提交贡献较多的开发者对于缺陷修复的贡献相对较少，但是在全体开发者中仍处于主要地位。

可见，核心开发者所参与提交的文件占有相当大比例，同时对于核心开发者所连接的文件，其作用往往比较关键，尤其是核心开发者提交的部分，而也因此可能会引发较严重的错误。对于此类文件，开发过程中也需要较多的协作，这也会引起理解或者集成的错误。故猜测对于修改人数较多而且其中包括核心开发者的文件，其产生错误的概率会较高。4.5节通过实验进一步探究。



开发者数量

1. 缺陷修复代码累计分布
2. Cumulative Distribution of Fixes

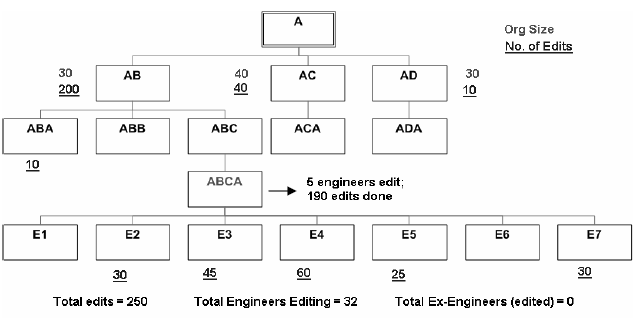
## 项目组织结构研究

关于项目组织结构，Fred Brooks[37] 在他的经典著作《人月神话》中描述了一个工程失败的例子，其失败的主要原因是因为交流以及随之导致的组织问题。Brooks进一步解释说，在软件系统中存在调度灾难，功能不适以及系统缺陷都是从不同团队之间交流的缺乏造成的。引用Brooks的话“组织的目的是减少必要的交流与协作总量，因此组织是对交流问题的根本的回击……”。同样，在软件产业中组织结构也十分重要，以至于可以认为软件模块的开发不仅仅是开发一个子程序而是对职责的一种分配。

### 项目组织结构特征提取

相对与开源项目社区，一些商业项目具有较为严格的管理模式与等级结构，本节在此背景下对项目组织结构相关特征进行研究与提取。

图4-7[4] 是某公司一个软件开发项目的组织结构及特征。图中表明，从开发之前到产品发布的整个过程，对项目的提交总数为250。其中A是公司的总项目经理管理着100人的组织。AB管理30人的部门，AC管理着40人的部门，AD管理30人的部门。图中也表示出了其他的各个副经理以及开发工程师。



1. 项目开发组织结构示例[4]
2. Example of an Organization Structure[4]

本文定义了一组组织特征，从软件开发的角度量化出项目组织结构的复杂性，并以图4-7为例进行说明。

1. 开发者数量（Number of Developers，NOD）：表示那些修改过项目的所有开发者的绝对数量，并且这些开发者目前仍然参与到项目中。

越多的人接触代码，代码产生缺陷的概率越高，因为这会造成开发者之间更高的协调的需要。Brooks[37] 称如果某一段代码有N个开发者会接触，那么对这N个开发者来说他们之间理论上会有(N\*(N-1))/2条交流路径。在本节的案例中如果有大量的开发者对特定代码进行了修改，就将会存在开发者之间信息的错误传达，引起设计上的不匹配，或者破坏了其他开发者的代码（build breaks），甚至造成设计基本原理的理解问题。由图4-7可知，本例中开发者数量为32。

1. 离职开发者数量（Number of Ex-Developers，NOED）：表示那些修改过项目的所有开发者中在项目某版本发布的时候已经离开团队的开发者数量。

这一项评估方式考虑到了知识的迁移。如果一个或多个曾从事于代码修改的开发者离开了团队，则后面将有新人来接管，很有可能不太熟悉设计原理，或者缺陷修复相关细节，也不了解其他负责相关工作的人员。这一特征也可从版本管理系统中提取，本例中离职开发者数量为0。

1. 编辑频率（Edit Frequency，EF）：这一特征表示项目中代码被修改的次数，一次编辑代表开发者将代码检出版本控制系统，修改之后再提交回去的过程，与代码行数没有特别的相互关系。

选取这一特征主要有两个目的，一个是如果项目中存在过多的编辑次数，这可能预示着代码缺乏稳定性，或者从其他角度来说存在可靠性以及性能等问题。第二，通过与开发者结合，这一特征提供了一个对编辑情况更加全面的分布视图，可知是否有一个开发者进行了大多数编辑。另外如果一个进行了较多编辑的开发者离开了项目这就可能造成前面所说的知识迁移问题。本例中编辑频率为250，同样可从版本控制系统中提取。

1. 主要所有权级别（Depth of Master Ownership，DMO）：这一特征根据编辑次数来确定对项目所有权的级别，对于组织级别的人来说他负责的开发者如果累计完成了超过特定比例的编辑工作（例如可选75%作为阈值），该组织者被认为具有主要所有权[49, 52]。

这一特征通过对项目的编辑活动来确定项目的所有人。在组织结构中所有权越深，所进行的开发活动，交流等越集中。更深的所有权级别意味着更少的讨论，单点的控制将使管理更加清晰。如果项目没有一个明确的所有人，则面对一些有风险的缺陷修复可能产生决策制定问题，或者对于代码交叉依赖的理解上缺乏相应的处理方案。本例中有超过75%的累计编辑次数可归于ABCA（199/250）。因此DMO的值是2（AB，AC，AD是level 0，ABA，ADA是level 1。A并没有参与到日常开发活动中不作考虑，而AB则是整体的组织负责人）。

1. 组织贡献百分比（Percentage of Org Contributing to Development，PO）：表示主要所有权级别下的开发者数量与主要负责人在组织规模的比率。

这一比率越低则表示所有权更加集中，项目开发中会对组织形成相对较低的协调与交流开销，同时也提高了人员之间的同步性，更利于控制。这一特征使得不平衡的组织的影响最小化。本例中特征值是(7/30)\*100%。ABCA负责7名开发者，而ABCA所述的组织规模是30人。

1. 组织代码所有权级别（Level of Organizational Code Ownership，OCO）：代码修改次数中由项目负责人所在组织进行修改的百分比，或者不存在项目负责人的情况下由进行了大多数修改的组织所占的百分比代替。

开发贡献越集中于同一个组织，他们越能形成一个共同的文化，集中性以及凝聚力，越多不同类型的开发者对代码进行贡献，则有越高的几率引入缺陷代码，例如同步问题，不匹配或者构建错误。本例中比率是200/ (200+40+10)。200是AB管理的组织下完成的修改次数，由此计算其相对三个部门总修改次数200+40+10的比率。

1. 整体组织所有权（Overall Organization Ownership，OOW）：这代表DMO中对项目修改的开发者相对于所有修改项目的开发者的比率，较高的数值较为理想。

正如之前的所有权度量指标，在同一组织中进行更多的活动，则他们更可能具有共同的文化和关注点。另外，组织间距离越远，就可能有更多的机会产生错误的传达与理解。这一特征可用于弥补OCO和PO特征中“super”开发者的现象，他们通过较少人数贡献了大量代码。为了不使这样的开发者影响度量又不会忽略他们，PO，OCO 和OOW相互辅助，进行处理。本例中可以看到有5个开发者由ABCA负责进行了代码提交，另外在整个组织中一共有32个开发者进行代码提交，因此该特征值为5/32。

1. 组织交叉系数（Organization Intersection Factor，OIF）：这一特征表示在整体的水平下，修改次数超过某一数量（如10%）的组织的数量。

这一系数的值越大，表示项目的分散度越高。这意味着缺乏来自一个特定组织的强制的所有权。这一特征适用于当一个项目在整体组织下所有权相对分散，没有明确所有者的情况。在本例中，总共有250次修改。如果设定阈值为10%，则总负责人下的两个组织AB与AC都贡献了超过25次的修改。因此OIF值为2，理想情况下较低的值意味着该组织结构较好。

### 开源项目组织结构案例研究

开源项目一般组织松散，不像商业公司中有严格的等级划分，本节主要研究开源项目的管理方式，开发者参与方式以及不同开发者级别划分。

开源软件开发具有社会性，对开源软件进行研究有助于更好地构建开源社区。Bird [38] 等人指出，开源软件项目得益于其缺乏组织性的特点，这使得开源软件项目更加灵活，更具创新性。但是他们得到证据证明开源软件项目中有较强的社区组织供开发者进行交流；开源软件项目中有潜在的自我组织的结构存在。Zanetti[25] 等人对开源软件的社会化组织进行多方面定量分析。通过一些测量发现开源社区成员协作结构特征的演变，并加入了一些复杂的测量方法突出开源软件工程社会结构中的深入差别。北大周明辉 [39] 等人研究在开源社区中什么原因会使新加入者成为长期开发者（LTC），作者将开发者自身因素与环境因素综合考虑，归纳成若干特征，对新加入者能否成为LTC建立预测模型。

接下来通过两个开源项目实例对开源项目组织进行研究：

1. Apache

总体来说，对Apache进行开发的参与者分布相当广泛，由大约400人贡献代码使之成为一个相对规模较小的产品。这里以Apache为对象研究开源项目参与者的角色及关系。

为了了解有多少人对新功能贡献代码而又有多少人进行缺陷修复，这里将变更区分开来，根据其目的是否是针对项目缺陷报告的处理。若是，则本次变更为缺陷修复，否则认为是代码提交。统计发现大约182人处理了695个修复任务，而249人贡献了6092次代码提交。

然后本文通过检查缺陷库来确定是谁来提交问题报告。问题报告来自相当广泛的参与者团体。实际上，有大约3000多人提交了总共3975个问题报告，其中有458人提交的591份报告在随后导致Apache代码或者文档的变更，而其他的报告没有造成变更，这是因为他们没有提供足够的细节使得其中提到的缺陷能够重现，抑或缺陷已经被发现或者修复掉，当然也可能是一些配置错误乃至操作系统自身原因导致的误报或者就重要程度来说可以不被考虑的问题。从数据对比也可看出，相当大部分的问题报告没有造成代码的变更，甚至在问题报告者中也包含一些单纯的垃圾邮件发送者。

可见在Apache的开发中主要有代码贡献者、缺陷修复者以及缺陷报告者组成，其工作量依次减少，对项目的参与程度也依次下降。

1. Mozilla

为探究开源项目中更加细致的分工与管理方式，本文选取Mozilla项目进行分析。Mozilla目前由mozilla.org staff（12名成员组成）负责管理操控来领导并协调这一项目，他们提出开发进程，并且参与到一些代码开发中。其中只有4名核心成员会花费他们相对较多的时间投入到浏览器应用的开发中，而其他核心成员主要从事于社区QA，里程碑发布管理，网站以及Bugzilla等工具的维护之类的事情中，来协助开发者。外部开发者在近年来逐步增加，甚至有一些外部人员（例如来自Sun Microsystems公司的开发者）会全职开发，相对的会从中得到酬劳。

项目不同模块的决策权会委托给开发社区中对特定代码最了解的人员。一些代码质量优秀的开发者会被创建记录，并且能够获得对CVS版本库进行提交的权利。特定模块中的目录与文件能够被添加，修改只有在得到模块管理者的允许之后。而添加新模块需要得到mozilla.org的权限。对于软件仓库，则采用分布式的提交方法以模块所有的方式进行授权管理，而mozilla.org有最重要的决策权，同时保有删除模块管理者的权限，并且负责解决所有可能产生的冲突。

通过Apache与Mozilla项目的组织特征描述可以发现其中的一些异同，Apache的使用者主要是开发者，而Mozilla使用者多为普通用户，相对于Apache相对松散的组织与管理模式，Mozilla拥有更加系统的组织结构，相对明确的分工以及严格的权限分配。

## 探索实验及发现

本节将进一步分析网络图、开发者地位和组织结构，探究其与软件质量的关系。

### 网络图的探索实验及发现

首先尝试从社会网络图中提取有用特征。对于社会网络图的构建方法，将传统网络图进行结合，意味着将不同网络所蕴含的数据信息以及社会信息相结合，例如开发者贡献图。相对传统网络图，开发者贡献图将会蕴含更加丰富的信息，本节对开发者贡献图与软件质量的相关性进行探究。

实验中，开发者贡献图的度量指标（Degree Centrality、Betweenness Centrality、Closeness Centrality）将被用作缺陷预测特征，Spearman系数用来评估预测特征与缺陷之间的相关性，数据集采用Eclipse3.0版本的缺陷数据。

1. 开发者贡献图特征与缺陷的相关性
2. Correlation between Developer Contribution Network and Defects

|  |  |
| --- | --- |
| 特征 | Spearman |
| Degree centrality | 0.473 |
| Betweenness centrality | 0.439 |
| Closeness Centrality | 0.357 |
| Random meature | 0.108 |

这里选取度中心性、中介中心性与紧密度中心性代表网络度量指标，见表4-2，其与缺陷的相关程度分别为0.473、0.439和0.357，为探究其效果，另外采用随机的方法生成度量指标分析与缺陷的关联。可以看出，社会网络指标与缺陷关联明显，能够作为缺陷预测的特征。

### 开发者地位的探索实验及发现

接下来探究开发者地位与软件缺陷关系。开发者可以根据其贡献被划分为核心开发者以及非核心开发者，核心开发者的开发行为也相对稳定。从4.4.2中Apache与Mozilla的例子中也可以发现，项目的领导者与核心开发者是关系到项目成功的关键。所以本文通过分析核心开发者所关联到的文件的缺陷情况来进一步探究。

本实验以Eclipse-jdt项目为对象，实验结果如表4-3所示，验证了开发者地位与缺陷的相关性。实验从以往的开发中选出4位核心开发者，并从bugzilla中提取出与其相关的缺陷数量。在表中，项目开发时期，所产生的缺陷总量大约为10000，比较可见，核心开发者所涉及缺陷数量相对较高。当然，缺陷的引入不一定与其相关，但是核心开发者有较高几率接触到缺陷文件。

1. 开发者重要性与缺陷的相关性
2. Correlation between Developer Position and Defects

|  |  |  |
| --- | --- | --- |
| 项目&时期 | 开发者 | 相关缺陷数量 |
| Eclipse JDT  (2002/01/01-2009/01/01) | Olivier\_Thomann | 1242 |
| Jerome\_Lanneluc | 996 |
| Frederic\_Fusier | 663 |
| Philipe\_Mulet | 998 |

接下来采用CraftBukkit项目分析核心开发者相关文件的缺陷率，这里利用论文[40]的研究成果来定义核心开发人员。结果如表4-3，其中Dinner Bone是核心开发人员，其余两位为非核心开发人员。可见，核心开发人员往往涉及到更多的缺陷文件，而这些文件通常起着关键作用，被修改的次数也较多。因此，核心开发者可以作为一个缺陷预测特征，但考虑到核心开发者的确定方式中涉及到对大量文件进行提交，这会降低其对缺陷预测的效果。

1. 核心开发者所涉及文件缺陷率
2. Defect Rate of Core Developer Involved Files

|  |  |
| --- | --- |
| 开发者 | 所涉及文件的缺陷率 |
| Dinner Bone | 51.6% |
| Travis Watkins | 31.1% |
| Celtic Minstrel | 22.8% |

### 组织结构的探索实验及发现

1. 组织结构特征总结
2. Summary of Organization Structure Features

|  |  |
| --- | --- |
| **特征** | **意义** |
| NOD | 更多的人接触代码会降低质量。 |
| NOED | 团队成员大量流失会影响到知识的保留进而影响质量。 |
| EF | 对组件过多的编辑将增加其不稳定性并降低质量。 |
| DMO | 项目所有权等级越低，质量越高。 |
| PO | 代码贡献者（从组织角度）紧密度越高，项目质量越高。 |
| OCO | 代码贡献者（从提交修改角度）紧密度越高，项目质量越高。 |
| OOW | 开发者对项目贡献越分散，其质量越低。 |
| OIF | 分散度越高的不同组织进行代码提交，质量越低。 |

最后探究组织结构与软件缺陷的关系。根据4.4.1节中组织结构的研究，在表4-5中列出了一组组织结构特征，并提出各项特性对项目质量影响的假设。

对于开源项目，往往缺乏严格的上下级组织，本文主要从人数规模、人员变动、人员地位等方面开展实验，探索开源项目组织特征[48]。数据来源主要是eclipse-jdt的bugzilla库，选取其中部分缺陷项看作修改任务。对参与到任务中的开发者以及人物涉及到的文件进行分析，分别从开发者数量、变更、工作流（Workflow Activity）特征结合文件缺陷情况来考虑，然后将其加以关联，找出相关性。关于工作流特征，考虑bugzilla任务发布后，开发者对其进行的修改的流程。这一流程反映了项目中针对特定开发任务的组织方式，特征值通过流程次数和不同的开发者数量确定。表4-6中是对开源项目三种组织特征与缺陷相关性分析的实验结果，相对其他特征，相关性较低。

1. 组织结构特征与缺陷的相关性
2. Correlation between Organization Structure and Defects

|  |  |
| --- | --- |
| 组织特征 | Spearman相关性 |
| Developer Number | 0.386 |
| Developer Change | 0.178 |
| Workflow Activity | 0.413 |

## 基于社会网络分析的新特征

根据对社会网络分析和研究的发现，本文从社会网络度量、开发者地位、项目组织结构等方面提取出以下新的缺陷预测特征：

1. 网络度量特征（Network Measure Features, NMF）： 对新构建的社会网络图进行度量，其度量指标与缺陷存在潜在联系。本文将采用Degree centrality 和Betweenness centrality作为特征在第五章中用于缺陷预测。
2. 开发者地位特征（Developer Role Features, DRF）： 核心开发者间相比非核心开发者往往更易接触到缺陷文件。但考虑到4.5节核心开发者特征的定义与探索实验，可知该特征将会导致缺陷预测结果有较高覆盖率但准确率很低，暂不能作为有效特征用于预测实验。
3. 项目组织结构特征（Organization Structure Features, OSF）： 项目的组织方式会影响到软件的质量。由于商业项目组织开发相关数据无法获取，而对于开源项目，开发者数量与变更并不能明显代表开源项目组织结构，工作流特征由于样例较少无法批量提取，故暂不用于缺陷预测。

## 本章小结

本章首先研究了社会网络分析技术及其创建方法，提出了利用社会网络本身的特征对社会软件工程进行度量。然后从社会网络角度从其他方面对软件开发的社会特性建立模型，包括项目的组织结构，开发者的地位，利用实际项目作为案例对各项特征进行研究，进一步提取出可用于缺陷预测的软件特征。

# 基于社会软件工程新特征的缺陷预测

本章首先对预测模型进行设计，然后根据对研究问题探究的发现，提取出基于邮件列表和基于社会网络分析的缺陷预测新特征，将其加入预测模型进行实验。最后对特征进行筛选与结合，对实验结果进行分析与讨论。

## 缺陷预测模型

本文采用回归模型进行缺陷预测，用于预测的自变量是各个类或文件的特征集合，所要预测的因变量表示一个类或文件是否具有缺陷以及所具有的缺陷数量，主成分分析（Principal Component Analysis）将被用来避免自变量之间的多重共线性问题。

为了比较与评价，本文建立了线性回归模型[23] 和逻辑回归模型[41] 分别进行缺陷预测。其中线性回归模型通过公式

………… (5-1)

表示，函数表示缺陷情况。为了把线性回归函数的值域映射到概率值的域[0,1]，采用逻辑回归公式

………… (5-2)

进行计算。Ambros[23] 和Bird[41] 的研究结果将被作为预测因子以及预测模型的评估基准。

主成分分析用于避免独立变量之间的多重共线性问题。这一问题来源于这些变量之间内部的相互关联，可能会引起对因变量评估时产生放大的效果。所以在构建回归模型时会采用一些组件来代替实际的自变量（特征），这些组件是独立的，因此避免了多重共线性问题，而为此也需要足够多的样本方差来协助筛选。

本文将采用交叉验证方法进行研究，其中90%的邮件列表数据（训练集）用来搭建预测模型，而其余10%数据（预测集）用来评估模型的正确性。在每次验证中，会分别随机进行10次90%~10%的划分，并将每次验证情况综合评价得到最终结果。

## 预测特征

根据上述研究总结，从相关性、成熟度等方面筛选出若干特征，用于预测实验。见表5-1。其中上部为邮件列表相关特征，下部为社会网络分析相关特征。

1. 实验所用预测特征
2. Predict Features in Experiment

|  |  |  |
| --- | --- | --- |
| 特征 | 定义 | 类型 |
| ROCF | 邮件中代码文本比率(Rate of Code) | Mail Content Features (MCF) |
| ROSTF | 邮件中堆栈跟踪文本比率(Rate of Stack Trace) |
| BCNF | 邮件网络中中介中心性  (Betweenness centrality of Mail Network) | Mail Network Features (MNF) |
| ESF | 邮件中情绪和值(Emotion Sum) | Emotion Features (EF) |
| EDF | 邮件中情绪差值(Emotion Difference) |
| TF1~TF5 | 前五种基于主题的特征  (Top Five Topic-based Features) | Topic-based Features (TF) |
| DC | 度中心性（Degree Centrality） | Network Measure Features (NMF) |
| BC | 中介中心性（Betweenness Centrality） |

## 预测评估指标

本文使用了两种IR 度量来评估预测结果：准确率（precision）和覆盖率（recall）。度量标准由如下公式计算：

………… (5-3)

…………… (5-4)

TP（真阳性）代表成功预测出的带缺陷的程序文件；FN（假阴性）集合包含没有被预测出的带缺陷的程序文件；FP（假阳性）指的是被错误预测为缺陷文件的正常程序文件。

同时也使用Spearman 相关系数评价预测能力，即使在特征之间并非线性关联的时候Spearman也可以被用作评价指标[18]。Spearman 相关性系数由两个列表（以实际缺陷数量排列的类的列表和以被预测出的缺陷数量排列的类列表）来计算出，可以看作列表相似性的指示器。

Spearman系数与IR 度量分别被用于对缺陷数量以及文件是否具有缺陷的预测评估。

## 缺陷预测实验

### 实验数据和实验过程

本文选取了91个邻近的eclipse 版本来进行实验，按前文提出的方法进行数据的收集与预处理，按5.2节分别计算出各预测特征的值，之后通过回归模型进行预测，通过与实际缺陷情况的对比来评估预测效果，其中所用到的eclipse 的缺陷数据是由Zeller 所提供。

缺陷预测实验分为3阶段执行：

1. 首先将各类预测特征分别作为自变量，把各文件所对应的缺陷数量作为因变量输入线性回归预测模型进行训练与预测实验，以预测缺陷数量与实际缺陷数量间的spearman相关系数来评估并与对比实验[23] 中的预测结果进行初步对比。
2. 然后将各类预测特征分别作为自变量，把各文件是否存在缺陷的情况作为因变量输入逻辑回归预测模型进行训练与预测实验，统计各文件的预测缺陷情况与实际缺陷来计算出IR度量来进行评估并与对比实验[41] 中的预测结果进行进一步对比。
3. 最后尝试将预测特征共同作为自变量输入模型，筛选出关键特征。并将基于社会软件工程的新特征结合对比实验[23] 中的特征集共同作为预测模型的输入，找出最佳预测结果的组合，并将各阶段的预测结果进行对比分析。

### 实验结果

表5-2中是各项特征的缺陷预测结果，本阶段将各类特征分别考虑，作为模型输入预测缺陷，并通过三种评估指标对结果进行评估。对于邮件列表，本文选取了邮件内容（Mail Content Features），邮件网络（Mail Network Features），情感特征（Emotion Features），主题特征（Topic-based Features）等4类特征；而对于社会网络分析方法，本文选取了度中心性（Degree Centrality），中介中心性（Betweenness Centrality）作为代表特征，这里的中心性特征是基于所构建的开发者贡献网络所计算的。

1. 单一特征预测结果
2. Prediction Results of Single Feature

|  |  |  |  |
| --- | --- | --- | --- |
| 特征 | Spearman | 准确率(Precision) | 覆盖率(Recall) |
| Mail Content Features（MCF） | 0.432 | 0.804 | 0.781 |
| Mail Network Features（MNF） | 0.416 | 0.763 | 0.746 |
| Emotion Features（EF） | 0.428 | 0.775 | 0.762 |
| Topic-based Features（TF） | 0.389 | 0.713 | 0.748 |
| Degree Centrality（DC） | 0.473 | 0.773 | 0.782 |
| Betweenness Centrality（BC） | 0.439 | 0.810 | 0.791 |

从结果中，可以发现社会网络分析的预测效果高于邮件列表特征，而到底所提取的特征对缺陷预测模型的改进效果如何，这需要将特征结合起来进行分析，同样也需要与已有特征进行对比。

### 实验对比分析

1. 预测结果及对比 Part1
2. Prediction Results and Comparison Part1

|  |  |  |  |
| --- | --- | --- | --- |
| 特征 | Spearman | 特征 | Spearman |
| 变更特征（Change Features） | 0.381 | MCF | 0.432 |
| 缺陷特征（Bug Features） | 0.434 | MNF | 0.416 |
| 代码特征（Code Features） | 0.395 | EF | 0.428 |
| 历史复杂度特征（History of Complexity Features） | 0.416 | TF | 0.389 |
| 代码扰乱度（Churn of code Features） | 0.442 | DC/BC | 0.473/0.439 |
| 代码熵（Entropy of code Features） | 0.425 | 本文特征最优组合 | 0.437 |
| 传统特征最优组合 | 0.448 | 整体最优组合 | 0.526 |

实验通过两部分数据对基于邮件列表新特征与现有特征的预测效果进行对比，如表5-3和表5-4所示。表5-3中左侧通过缺陷的预测数量与实际数量之间的spearman 相关性表示对比实验[23] 中模型的预测能力，右侧表示采用本文中预测模型的预测结果。可以看出当单独使用表5-2中定义的各类预测特征时相关性较低。而且将本文中的预测特征组合使用的最优结果相较以前方法中的最优结果仍有差距。在最后一行中，当将新的预测特征与现有特征结合之后取得了最好结果。

另一方面，由于所对比实验[23] 中并不包含社会化特征，故另外选取其他数据[41] 作进一步比较，即表5-4中上半部分所展示结果。本文中的特征对应结果在下半部分展示。同样，组合特征效果更优。对于预测模型，根据其特点与适用性，在与常用特征对比中采用线性回归模型预测缺陷数量；与社会化特征对比中采用逻辑回归模型预测是否具有缺陷。两种模型在其适用范围下都获得较好预测效果，无优劣之分。

1. 预测结果及对比 Part2
2. Prediction Results and Comparison Part2

|  |  |  |  |
| --- | --- | --- | --- |
| 特征类型 | | 准确率(Precision) | 覆盖率(Recall) |
| 传统社会技术特征（Social Technical） | Dependency | 0.789 | 0.625 |
| Contribution | 0.764 | 0.688 |
| Combined | 0.754 | 0.708 |
| Socio-technical | 0.753 | 0.761 |
| 邮件列表与社会网络新特征（Mailing List & Social Network） | MCF | 0.804 | 0.781 |
| MNF | 0.763 | 0.746 |
| EF | 0.775 | 0.762 |
| TF | 0.713 | 0.748 |
| DC | 0.773 | 0.782 |
| BC | 0.810 | 0.791 |
| Combined | 0.831 | 0.812 |

综上，与一些目前已有的特征和模型相比，本文所提出的方法在特定条件下取得了与其相当的预测效果，而且特征的组合有助于获得更好的结果。这表明，邮件相关的特征对缺陷的预测有一定作用。同时，从实验结果中发现虽然仅仅使用邮件相关特征时预测能力并不是最优，但是与现有特征结合之后能够进一步提升预测能力。对于社会网络特征，本文构建的开发者贡献网络对缺陷预测有一定的改进效果。

值得注意的是，对于邮件列表特征，当采用不同电子邮件相关的特征时，预测结果也有所不同。考虑到本文在预测模型中仅使用了来自jdt-dev中的数据而忽略了jwt-dev和eclipse-dev中的数据，结果并不明显。但可发现邮件结构特征比情感、主题等特征能够提供更好的预测效果。对于社会网络图特征，其预测效果也优于邮件文本特征。非结构化文本是邮件的主要部分而且囊括了更多的信息，所以也期待从非结构化文本信息中获得更好的特征。将非结构化的文本纳入考虑中的想法是值得肯定的，不过分析方法需要改进。特别是主题模型分析方法，主题和被排名的词汇的数目需要慎重获取，主题特征定义也需要调整。这些都有助于缺陷预测的结果的进一步改善。

## 本章小结

本章将研究问题探究实验所发现的基于邮件列表的缺陷预测特征以及社会网络特征，加入预测模型进行实验，并对结果进行分析与讨论。实验结果证明各项特征都能够从不同程度上进行缺陷预测，并且通过与传统缺陷预测特征的结合，能够对预测效果有所改进，证明了利用社会软件工程特征进行缺陷预测能有效改善缺陷预测效果。

# 总结与展望

## 本文工作小结

本文对社会软件工程与软件质量之间关系进行了探索。首先从邮件列表中挖掘交流信息，将eclipse 社区中挖掘得到的邮件列表关联到相应的源代码，采用了三种方式分析邮件，获取了邮件内容结构、网络组织结构、情感特征以及主题结构等新特征，并通过探索实验发现这些交流新特征与软件质量具有相关性。然后又提出社会网络分析方法，充分利用eclipse数据仓库，改进社会网络图，提取社会网络特征，在此过程中还从项目组织结构，开发者地位等方面进一步分析社会网络，探究新的预测特征。

在此基础上，本文利用这些社会软件工程新特征基于回归模型进行了软件缺陷预测。实验表明，基于邮件列表挖掘的新特征确实能够预测软件缺陷，同样从所构建的社会网络图中提取的社会网络特征也能够用来预测软件缺陷，而在此基础上的结合能进一步提高预测效果，并且可以有效改进现有的预测模型。

和现有工作相比，本文的创新性表现在以下方面：

1. 对邮件列表进行全面分析，将抽象的交流行为具体归结为多项邮件列表特征，用于缺陷预测。通过严格的数据提取与连接步骤，确保邮件交流内容的实用性；又通过多角度的分析手段保证了特征提取的完整性，为缺陷预测实验提供保障。
2. 改进社会网络分析方法，丰富了网络图信息，不仅考虑开发工件间的内在依赖，更进一步考虑其中反映出的开发者协作关系。并进一步对社会化软件开发的特征进行深入挖掘，包括开发者地位的探究和组织结构的分析。
3. 基于缺陷预测新特征对预测模型进行改进。本文以社会软件工程为研究对象，将开发者交流协作的行为抽象为数据特征，并与传统特征相结合，使得缺陷预测效果得到了提高。

## 展望

将来的研究重点将放在邮件分析方法效力的提高，交流模式的探究、实验范围的扩展，将采用更多大规模软件系统进行实证研究，对所提出的预测方法进行评估与改进。本文准备从以下几个方面进行进一步的深入研究：

1. 邮件列表分析方法的研究

在设计中，本文只考虑了能够被邮件列表关联到的文件。由于能够链接到的邮件的数量只是所有邮件的少部分，这也使得关联方法存在不可避免的缺陷。值得注意的是链接过后只有少数邮件被保留下来，这表示在这一阶段中有一些缺陷文件会疏漏掉，而覆盖率也会因此而受到影响。另外，从预测结果可以发现基于邮件列表的预测特征中，部分特征的预测效果不够好。其中部分原因是对邮件正文的分析方法还不够深入。将来的工作重点一是对邮件关联方式进行改进，二是致力于邮件列表特征效力的提高，例如基于主题的预测方式。

1. 基于交流模式的缺陷预测新特征的研究

Igor Cavrak[7] 等人根据从14个分布式项目中收集到的数据进行分析，识别出一系列的协作模式，并给出模式形成的原因与影响。模式是对团队中协同连接进行分析的结果，能够更好地协助管理者组织项目。Ernst Oberortne[8] 等人通过总结研究中提出的相似的交流问题，提出一些模式帮助组织开发团队，尤其是分布式开发团队来增强交流。Oberortner提出的模式对组织同地团队协作，增强与远程团队交流以及专家发现都提供了支持。

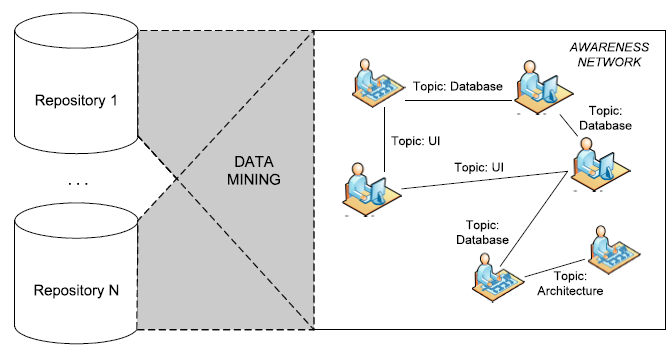
开发者进行交流会有怎样的特点，例如对于同一功能模块的开发者，其相互交流是否对等，每个人主动交流与被动交流的情况有没有差别，对于缺陷的发布与讨论情况如何，核心开发者的交流情况又有什么特点，以及开发计划的定制与开发进度的保证相关的消息的发布情况。这些都会形成一定的交流模式[55, 56]，而具体的模式可能会影响到交流的效果与质量。

1. 中间人模式

不同功能模块之间由于依赖等关系，往往也需要各自的开发者进行交流，跨功能模块的开发者交流同样会形成交流模式。例如，交流是通过具体开发者的亲自交流或者是通过各模块的“代表”作为接口进行交流，如图6-1。这一代表可看作中间人，负责将关键的变更，功能扩展等内容与其他团队中间人进行交流。这对于广泛交流中产生的错误传达、错误理解问题都有所改善，也能促进信息的分享。



1. 中间人模式
2. Broker Pattern
3. 有意识的连接模式



1. 有意识的连接模式
2. Awareness Link Pattern

有意识的网络连接Awareness Network主要为方便找到团队中的专家或者某些具有特定技术的开发人员而设计。

要构筑一个Awareness Networ，可从多个项目的软件仓库进行挖掘发现其中所反映出的开发团队成员间的交流内容，如图6-2。所形成的交流结构将被告知所有团队成员，使得他们能够找到所需的专门的开发人员。文献[21]研究如何从软件工程的交流中发现潜在的专家，这有助于避免无谓交流。具体方法是研究专家在email讨论中被包含的上下文，分析专家出现讨论的情况。

Awareness Networ不同于单纯以分配任务构筑的网络，因为无形的关系能被检测到。Awareness Networ能够加速分布式开发的团队中的交流。新加入的成员能够加快他们的生产力因为他能能够很快知道通过联系谁来解决遇到的问题。

1. 公告模式

邮件列表可根据不同特征分为若干类型：Developer mailing list, User mailing list, Announce mailing list等等。 其中Announce mailing list负责项目开发中各项事件的发布与公告。

交流模式的研究将作为以后的研究重点，可以尝试从各种交流模式中提取出缺陷预测特征，用于改进预测效果。

1. 研究范围的扩展

在缺陷预测实验中，文中采用eclipse 项目数据进行分析以此与相关工作进行对比。然而，实验的结果也只对eclipse 项目数据是有效地而缺乏对其他项目的普遍有效性。为更好验证结果，研究应涵盖更多项目，进一步的研究中将会采用更多大规模软件系统进行实证研究来对所提出的预测方法进行证明与评估。

参考文献

1. Navid Ahmadi, Mehdi Jazayeri, Francesco Lelli, et al. A Survey of Social Software Engineering[C]//Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering, L'Aquila, Italy, 2008: 1-12.
2. Kim Man Lui, Keith C. C. Chan. Software Development Rhythms: Harmonizing Agile Practices for Synergy[M]. Wiley-Interscience, 2008.
3. Edward Yourdon, Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible'[M]. Projects Prentice Hall PTR, 1999.
4. Nachiappan Nagappan, Brendan Murphy, Victor Basili. The influence of organization structure on software quality: an empirical case study[C]//Proceedings of the 30th International Conference on Software Engineering, Leipzig, Germany, 2008: 521-530.
5. Timo Wolf, Adrian Schroter, Daniela Damian, Thanh Nguyen. Predicting Build Failures using Social Network Analysis on Developer Communication[C]//Proceedings of the 31st International Conference on Software Engineering, Vancouver, Canada, 2009: 1-11.
6. T. Wolf, A. Schr¨oter, D. Damian, L. D. Panjer, and T. H. Nguyen. Mining task-based social networks to explore collaboration in software teams[J]. IEEE Computer Society, 2009, 26(1): 58–66.
7. Igor Čavrak, Marin Orlić, Ivica Crnković. Collaboration Patterns in Distributed Software Development Projects[C]//Proceedings of the 34th International Conference on Software Engineering, Zürich, Switzerland, 2012: 2-9.
8. E. Oberortner, I. Kwan, D. Damian. Towards Patterns to Enhance the Communication in Distributed Software Development Environments [C]//Proceedings of the 18th Conference on Pattern Languages of Programs, Portland, USA, 2011: No. 21.
9. Boyd, d. m., Ellison, N. B. Social network sites: Definition, history, and scholarship, Journal of Computer Mediated Communication[J]. Journal of Computer-Mediated Communication, 2007, 13(1): 210-230.
10. Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J. Leveraging Transparency[J]. IEEE Computer Society, 2013, 30(1): 37-43.
11. Christoph Treude, Margaret-Anne Storey. How Tagging helps bridge the Gap between Social and Technical Aspects in Software Development[C] //Proceedings of the 31st International Conference on Software Engineering, Vancouver, BC, 2009: 16-24.
12. Christian Bird, Alex Gourley, Prem Devanbu, et al. Mining email social networks[C]//Proceedings of the 2006 International Workshop on Mining Software Repositories, Shanghai, China, 2006: 137-143.
13. Patrick Wagstrom, Jim Herbsleb, Kathleen Carley. A social network approach to free/open source software simulation[C]//Proceedings of the First International Conference on Open Source Systems, Genova, Italy, 2005: 16-23.
14. Alberto Bacchelli, Michele Lanza, Romain Robbes. Linking e-mails and source code artifacts[C]//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, Firenze, Italy, 2010: 375-384.
15. Alberto Bacchelli, Michele Lanza, Marco D’Ambros. Miler: A Toolset for Exploring Email Data[C]//Proceedings of the 33rd International Conference on Software Engineering, Honolulu, Hawaii, USA, 2011: 1025-1027.
16. Anja Guzzi, Alberto Bacchelli, Michele Lanza, et al. Communication in Open Source Software Development Mailing List[C]//Proceedings of the 10th Working Conference on Mining Software Repositories, San Francisco, California, USA, 2013: 277-286.
17. Raphael Pham, Leif Singer, Olga Liskin, Fernando Figueira Filho, Kurt Schneider. Creating a Shared Understanding of Testing Culture on a Social Coding Site[C]//Proceedings of the 35th International Conference on Software Engineering, San Francisco, CA, USA, 2013: 112-121.
18. Martin Pinzger, Nachiappan Nagappan, Brendan Murphy. Can Developer-Module Networks Predict Failures?[C]//Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, Atlanta, Georgia, USA, 2008: 2-12.
19. Bettenburg, Nicolas, Hassan, et al. Studying the Impact of Social Structures[C]//Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension, Braga, Portugal, 2010: 124-133.
20. Pamela Bhattacharya, Marios Iliofotou, Iulian Neamtiu, et al. Graph-Based Analysis and Prediction for Software Evolution[C]//Proceedings of the 34th International Conference on Software Engineering, Zürich, Switzerland, 2012: 419-429.
21. M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer. Categorizing bugs with social networks: A case study on four open source software communities[C]//Proceedings of the 35th International Conference on Software Engineering, San Francisco, California, USA , 2013: 1032–1041.
22. Katja Kevic, Sebastian Müller, Thomas Fritz, Harald Gal, Collaborative Bug Triaging using Textual Similarities and Change Set Analysis[C]//Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering, San Francisco, CA, USA, 2013: 17-24.
23. M. D’Ambros, M. Lanza, R. Robbes, An extensive comparison of bug prediction approaches[C]//Proceedings of the 7th Working Conference on Mining Software Repositories, Cape Town, South Africa, 2010: 31-41.
24. Tung Thanh Nguyen, Tien N. Nguyen, Tu Minh Phuong. Topic-based Defect Prediction (NIER Track)[C]//Proceedings of the 33rd International Conference on Software Engineering, Honolulu, Hawaii, USA, 2011: 932-935.
25. Marcelo Serrano Zanetti, Emre Sarigol, Ingo Scholtes, et al. A Quantitative Study of Social Organization in Open Source Software Communities[C]// Proceedings of the 35th International Conference on Software Engineering, London, UK, 2012: 1032-1041.
26. Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction[C]//Proceedings of the 30th International Conference on Software Engineering, Leipzig, Germany, 2008: 181-190.
27. G.Piatetsky-Shapiro, W.J.Frawley, Christopher Matheus. Knowledge Discovery in Databases[J]. AI Magazine, 1991, 13(3): 57-70
28. 白洁, 李春平. 软件知识库的数据挖掘研究现状[J]. 计算机应用研究, 2008, 25卷1期.
29. Raffaele Branda, Anna Tolve, Licio Mazzeo, et al. Linking E-Mails and Source Code Using BM25F[C]//Proceedings of the 17th European Conference on Software Maintenance and Reengineering, Genova, Italy, 2013: 421-424.
30. Alberto Bacchelli, Tommaso Dal Sasso, Marco D’Ambros, et al. Content Classification of Development Emails[C]//Proceedings of the 34th International Conference on Software Engineering, Zürich, Switzerland, 2012: 375-385.
31. Peter C. Rigby, Ahmed E. Hassan. What Can OSS Mailing Lists Tell Us? A Preliminary Psychometric Text Analysis of the Apache Developer Mailing List[C]//Proceedings of the Fourth International Workshop on Mining Software Repositories, Minneapolis, USA, 2007: 23-30.
32. Xihao Xie, Wen Zhang, Ye Yang, et al. DRETOM: Developer Recommendation based on Topic Models for Bug Resolution[C]//Proceedings of the 8th International Conference on Predictive Models in Software Engineering, Lund University, Sweden, 2012: 19-28.
33. baiduforum. The mystery behind search - introduction to semantic computing[EB/OL]. http://blog.csdn.net/baiduforum/article/details/7007729, 2011.
34. D. Blei, A.Y. Ng, and M. Jordan. Latent Dirichlet Allocation[J]. Machine Learning Research, 2003, 3: 993-1022.
35. Kate Ehrlich, Marcelo Cataldo. All-for-One and One-for-All? A Multi-Level Analysis of Communication Networks and Individual Performance in Geographically Distributed Software Development[C]//Proceedings of the 15th ACM Conference on Computer Supported Cooperative Work, Seattle, Washington, USA, 2012: 945-954.
36. Omar Alonso, Premkumar T. Devanbu, Michael Gertz. Expertise Identification and Visualization from CVS[C]//Proceedings of 5th the International Working Conference on Mining Software Repositories, Leipzig, Germany, 2008: 125-128.
37. F. P. Brooks. The Mythical Man-Month, Anniversary[M]. Addison-Wesley Publishing Company, 1995.
38. C. Bird, D. Pattison, R. D’Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects[C]//Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, Atlanta, GA, USA, 2008: 24–35.
39. Minghui Zhou, Audris Mockus. What Make Long Term Contributors: Willingness and Opportunity in OSS Community[C]//Proceedings of the 34th International Conference on Software Engineering, Zürich, Switzerland, 2012: 518-528.
40. Weicheng Yang, Beijun Shen, Ben Xu. Mining GitHub: Why Commit Stops - Exploring the Relationship between Developer's Commit Pattern and File Version Evolution[C]//Proceedings of the 20th Asia-Pacific Software Engineering Conference, Bangkok, Thailand, 2013: 165-169.
41. Christian Bird, Nachiappan Nagappan, Harald Gall Brendan Murphy, et al. Putting It All Together: Using Socio-technical Networks to Predict Failures[C]//Proceedings of the 2009 20th International Symposium on Software Reliability Engineering, Mysuru, India, 2009: 109-119.
42. Leif Singer, Fernando Figueira Filho, Margaret-Anne Storey. Software engineering at the speed of light: how developers stay current using twitter[C]//Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 2014: 211-221.
43. Hideaki Hata, Osamu Mizuno, Tohru Kikuno. Bug prediction based on fine-grained module histories[C]//Proceedings of the 34th International Conference on Software Engineering, Zürich, Switzerland, 2012: 200-210.
44. Peter C. Rigby, Ahmed E. Hassan. What can OSS mailing lists tell us? A preliminary psychometric text analysis of the Apache developer mailing list[C]//Proceedings of the Fourth International Workshop on Mining Software Repositories, Minneapolis, MN, USA, 2007: 20-26.
45. Stacy K. Lukins, Nicholas A. Kraft, Letha H. Etzkorn. Source Code Retrieval for Bug Localization using Latent Dirichlet Allocation[C]//Proceedings of the 15th Working Conference on Reverse Engineering, Antwerp, Belgium, 2008: 155-164.
46. W. Zhang, Y. Yang, Q. Wang. An empirical study on identifying core developers using network analysis[C]//Proceedings of the 2nd international workshop on Evidential assessment of software technologies, Lund, Sweden, 2012: 43–48.
47. A. Kumar, A. Gupta. Evolution of developer social network and its impact on bug fixing process[C]//Proceedings of the 6th India Software Engineering Conference, New Delhi, 2013: 63–72.
48. J. Ell. Identifying failure inducing developer pairs within developer networks[C]//Proceedings of the 2013 International Conference on Software Engineering, San Francisco, California, USA, 2013: 1471–1473.
49. C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don’t touch my code!: examining the effects of ownership on software quality[C]//Proceedings of the 19th ACM SIGSOFT symposium and the 13th European Conference on Foundations of software engineering, Szeged, Hungary, 2011: 4–14.
50. Feng Zhang, Audris Mockus, Iman Keivanloo, Ying Zou. Towards building a universal defect prediction model[C]//Proceedings of the 11th Working Conference on Mining Software Repositories, Hyderabad, India, 2014: 182-191.
51. A. Meneely, L. Williams, W. Snipes, and J. Osborne. Predicting failures with developer networks and social network analysis[C]//Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, Atlanta, GA, USA, 2008: 13–23.
52. F. Rahman and P. Devanbu. Ownership, experience and defects: a fine-grained study of authorship[C]//Proceedings of the 33rd International Conference on Software Engineering, Honolulu, Hawaii, USA, 2011: 491–500.
53. B. Simpson. Changeset based developer communication to detect software failures[C]//Proceedings of the 35th International Conference on Software Engineering, San Francisco, California, USA, 2013: 1468–1470.
54. D. Surian, N. Liu, D. Lo, H. Tong, E.-P. Lim, and C. Faloutsos. Recommending people in developers’ collaboration network[C]//Proceedings of the 18th Working Conference on Reverse Engineering, Limerick, Ireland, 2011: 379-388.
55. Klaas-Jan Stol Lero, Brian Fitzgerald Lero. Two's company, three's a crowd: a case study of crowdsourcing software development[C]//Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 2014: 187-198.
56. Qi Xuan, Vladimir Filkov. Building it together: synchronous development in OSS[C]//Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 2014: 222-233.

# 攻读学位期间发表的学术论文

1. First Author. Mining Developer Mailing List to Predict Defects. The 21st Asia-Pacific Software Engineering Conference, December 1-4, 2014, Chejudo.