# 软件开发者推荐技术的研究现状

时间区间：2014年～至今

1. 传统开发者推荐
2. 基于众包平台开发者推荐
3. reviewer推荐

---

## 传统开发者推荐

### 1. A Novel Developer Ranking Algorithm for Automatic Bug Triage Using Topic Model and Developer Relations

- Authors: Tao Zhang, Geunseok Yang, Byungjeong Lee, Eng Keong Lua
- Institution: The Hong Kong Polytechnic University (香港理工大学, Tao Zhang), The University of Seoul (首尔大学, Geunseok Yang, Byungjeong Lee), NEC Laboratories (Japan, Eng Keong Lua)
- Conference: APSEC
- Year: 2014
- Objective: Recommend the most suitable fixer for software issues.
- Method: The proposed approaches combine topic model and developer relations (e.g., bug reporter and assignee) to capture developers' interest and experience on specific bug reports.
- Results: The experimental results reveal that our approach outperforms other recommendation methods for developers.

### 2. Utilizing a multi-developer network-based developer recommendation algorithm to fix bugs effectively

- Authors: Geunseok Yang, Tao Zhang, Byungjeong Lee
- Institution: University of Seoul, Seoul, Korea (首尔大学)

- Conference: SAC '14 Proceedings of the 29th Annual ACM Symposium on Applied Computing
- Year: 2014
- Method: We propose a novel algorithm for developer recommendation. We first introduce a component and a similar bug-based selection process to verify the candidate fixers, then by adopting the number of comments and commits, we construct a multi-developer network so that ranking these candidates for finding the most appropriate fixer to resolve the given bug.
- Results: The result shows that our approach performs the task of bug triage effectively.

### 3. TIPMerge: recommending developers for merging branches

- Authors: Catarina Costa, Jair Figueiredo, Anita Sarma, Leonardo Murta
- Institution: Federal University of Acre, Brazil (巴西联邦大学)
- Conference: FSE
- Year: 2016
- Why: Development in large projects often involves branches, where changes are performed in parallel and merged periodically. This merge process often combines two independent and long sequences of commits that may have been performed by multiple, different developers. It is nontrivial to identify the right developer to perform the merge, as the developer must have enough understanding of changes in both branches to ensure that the merged changes comply with the objective of both lines of work (branches), which may have been active for several months.
- Method: We designed and developed TIPMerge, a novel tool that recommends developers who are best suited to perform the merge between two given branches. TIPMerge does so by taking into consideration developers' past experience in the project, their

changes in the branches, and the dependencies among modified files in the branches.

---

## 基于众包平台开发者推荐

### 1. Developer recommendation for crowdsourced software development tasks

- Authors: Ke Mao, Ye Yang, Qing Wang, Yue Jia, Mark Harman
- Institution: University College London (Ke Mao, Yue Jia, Mark Harman), Stevens Institute of Technology (斯蒂文斯理工学院 USA, Ye Yang), Chinese Academy of Sciences (中国科学院, Qing Wang)
- Conference: SOSE
- Year: 2015
- Objective: Automatically match tasks and developers for crowdsourced software development.
- Method: The approach learns particular interests from registration history and mines winner history to favour appropriate developers.
- Results: The evaluation results show that promising accuracy and diversity are achievable. We also provide advice extracted from our results to guide the crowdsourcing platform in building a recommender system in practice.

### 2. A learning to rank framework for developer recommendation in software crowdsourcing

- Authors: Jiangang Zhu, Beijun Shen, Fanghuai Hu
- Institution: Shanghai Jiao Tong University, East China University (华东理工大学, Fanghuai Hu)
- Conference: Asia-Pacific Software Engineering Conference
- Year: 2015
- Why: crowdsourcing has been widely used in many tasks that

computers are not good at such as image recognition, entity resolution or some question answering tasks.People can deal with these tasks with common sense knowledge. However, different from crowdsourcing in a general domain, software crowdsourcing is more complex.

- Method: Task characteristics learned from their descriptions and developer characteristic distributions extracted from their historical tasks are fed into our learning to rank algorithms for developer recommendation together with some other features such as topic-based features.
- Results: The experimental results show that our approach is feasible and effective.

### 3. A developer recommendation framework in software crowdsourcing development

- Authors: Wei Shao, Xiaoning Wang, and Wenpin Jiao
- Institution: Peking University
- Conference: NASAC
- Year: 2016
- Method: We present a feature model to depict software crowdsourcing tasks and accordingly propose a recommendation framework to recommend developers in CSD by combining a neural network and a content-based method.
- Results: Our approach increases the accuracy more than two times besides having a pretty good extendibility.

---

## reviewer推荐

### 1. Reviewer recommendation for pull-requests in Github: What can we learn from code review and bug assignment

- Authors: Yue Yu, Huaimin Wang, Gang Yin, Tao Wang
- Institution: National University of Defense Technology (国防科技大学)
- Journal: Information and Software Technology
- Year: 2016
- Why:
  - With pull-requests becoming increasingly popular, the need for qualified reviewers also increases.
  - GitHub facilitates this, by enabling the crowd-sourcing of pull-request reviews to a larger community of coders than just the project's core team, as a part of their social coding philosophy.
  - However, having access to more potential reviewers does not necessarily mean that it's easier to find the right ones.
- Objective: This study aims to investigate whether and how previous approaches used in bug triaging and code review can be adapted to recommending reviewers for pull-requests, and how to improve the recommendation performance.
- Method:
  - We extend three typical approaches used in bug triaging and code review for the new challenge of assigning reviewers to pull-requests.
  - We analyze social relations between contributors and reviewers, and propose a novel approach by mining each project's comment networks (CNs).
  - We combine the CNs with traditional approaches, and evaluate the effectiveness of all these methods on 84 GitHub projects through both quantitative and qualitative analysis.
- Results: CN-based recommendation can achieve, by itself, similar performance as the traditional approaches. However, the mixed approaches can achieve significant improvements compared to using either of them independently.

### 2. Who should be review my code? A File Location-Based Code-Reviewer Recommendation Approach for Modern Code Review

- Authors: Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, Ken-ichi Matsumoto
- Institution: Nara Institute of Science and Technology (奈良先端科学技术大学, Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Hajimu Iida, Ken-ichi Matsumoto), Osaka University(大阪大学, Raula Gaikovina Kula), Nagoya University (名古屋大学, Norihiro Yoshida), Japan
- Conference: SANER
- Year: 2015
- Why: Little research is known the difficulty of finding code-reviewers in a distributed software development and its impact on reviewing time.
- Objective: Find appropriate code-reviewers.
- Method: We propose REVFINDER, a file location-based code-reviewer recommendation approach.
- Results: REVFINDER accurately recommended 79% of reviews with a top 10 recommendation. The overall ranking of REVFINDER is 3 times better than that of a baseline approach.

### 3. Automatically recommendation peer reviewers in modern code review

- Authors: Motahareh Bahrami Zanjani, Huzefa Kagdi, Christian Bird
- Institution: Wichita State University (US, 威奇塔州立大學), Microsoft Research (Christian Bird)
- Journal: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING
- Year: 2016
- Objective: Recommend reviewers who are best suited to participate in a given review

- Method: recommend reviewers who are best suited to participate in a given review, based on their historical contributions as demonstrated in their prior reviews.
- Results: We show that by leveraging the specific information in previously completed reviews (i.e.,quantification of review comments and their recency), we are able to improve dramatically on the performance of prior approaches, which (limitedly) operate on generic review information (i.e., reviewers of similar source code file and path names) or source coderepository data. We also present the insights into why our approach cHRev outperforms the existing approaches.