Employee effort and productivity in organizations: A network data science approach

**Nan Wang; Evangelos Katsamakas**

# Abstract

Estimating the performance of employees is an important consideration in all organizations. This paper proposes a network data science approach to the estimation and visualization of employee effort, productivity and collaboration patterns. Using data from a software development organization, a bipartite weighted network is firstly constructed for developers’ commitment to project repositories. It is afterwards projected into two weighted one-mode networks: developer-to-developer and repository-to-repositories. For the former graph, two developers are defined to be connected if they once shared projects and for the latter, two repositories relate with each other if sharing developers. Techniques applied include graph theoretic metrics, power-law estimation, and community detection algorithms. We validate the existence of power-law relationships on project sizes (number of developers). We discuss implications for managers and future research directions. As a methodological contribution, the paper demonstrates how network data science can be used to derive a broad spectrum of insights about employee effort in organizations.

**Keywords:** networks, data science, network analysis, power laws, employee effort, productivity, collaboration

# Introduction

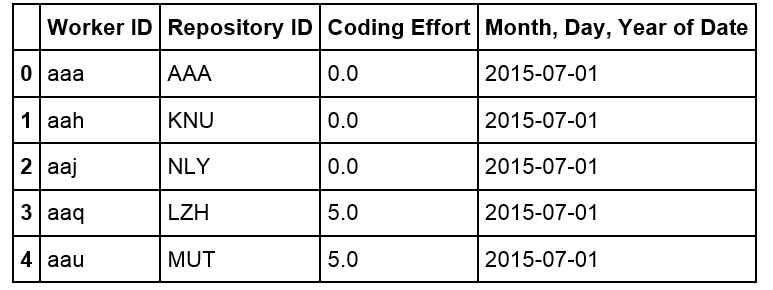
Software application development has turned into an enormously profitable business, with revenue from mobile application purely expected to exceed fifty billion USD by 2016 [1]. Accurate evaluation of developers’ achievement and contribution has been proved to be critical for the long-term development of companies as it is directly related to employee morale, overall productivity and creativity. Poor capability to discover and monitor coding performance prevents technical companies from transforming information on network activity and infrastructural capabilities into strategic knowledge [2, 3].

Although extensive historical study has applied social network analysis as a powerful solution to help understand organizational network performance and service interaction [4], most of them were designed in the context of binary networks. However, many real-life networks are widely recognized to be intrinsically weighted. Characterizing features and structures in weighted networks, therefore, has more important practical significance [5]. Bearing that in mind, this paper investigates a weighted bipartite network where developer is one mode and repository is another.

Question attempted to be answered, at either individual or corporate levels, include identification of developers’ role, contribution, motivation, functionality and working preference, as well as in a broader sense, project membership, corporate cohesion and software development.

# Dataset and data preparation

The dataset used in this paper is the coding effort of 2621 developers to 1705 repositories in a real-life company in a period of 92 days, from July 1, 2015 to September 30, 2015. This panel data has four variables, namely date, Worker ID, coding effort, Repository ID. There are 172,354 records in total, where “Worker ID” and “Repository ID” identify each unique developer and project respectively. Data is gathered on past software development activity via source code repositories like Subversion and Git, and task tracking systems such as Jira. This dataset is kindly provided by the company BlueOptima. Figure 1 shows a subset from the dataset.



**Figure 1.** Sample of original dataset. Coding effort is a proprietary software measurement and is calculated through evaluating every change that software developers contribute to projects in terms of a series of metrics, such as volume, complexity and interrelatedness of codes [6].

There are many records with zero coding effort in the data. It is resulted from the fact that if a developer is involved in a project, coding effort will be recorded regardless of the absence of contribution. Therefore, to keep the information of developers’ involvement, we remained all data records.

# Modeling Contribution: Developers-projects graph

When relations are modeled between two different classes of objects, bipartite graphs arise naturally [7, 8]. A graph of customers and products, with an edge between a customer and a product, if the customer has bought that product, is a simple example of a bipartite graph.

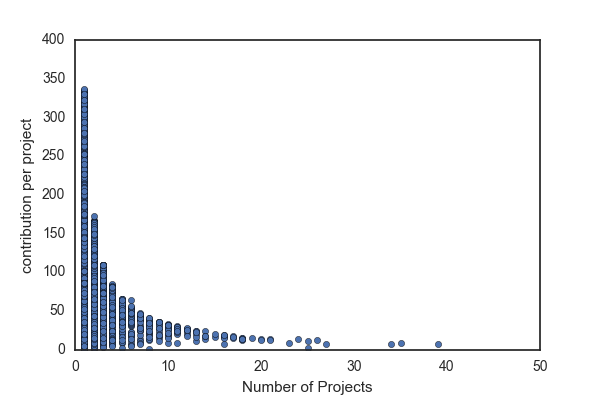
Before constructing the graph, we aggregated each developer’s contribution to every repository. The graph therefore stands for a static developer-to-repository network, where developers belong to one mode and repositories another. Edges represent developers’ contribution to projects and are weighted by coding effort.

* 1. **Graph Properties**

To understand the properties of this network, we applied a measurement called degree centrality to the graph.

Degree is the number of ties that a node has. In the developer-repository situation, nodes with degree of one can be interpreted as “focused developers” who worked on only one project, or individual project which contained only one developer. It is found that during the entire period in concern, over 50% developers (1409 out of 2459) devoted to only one project and above 40% projects (610 out of 1496) were individual projects. In addition, the maximum degree is 39 for all developer nodes and 96 for repository nodes, meaning that the most multiple-tasking developer (Developer “ouj”) contributed to 39 projects and the largest project (Project “KGD”) were contributed by 96 developers. Also, the average degree is 2.3 for worker nodes and 3.8 for repository, meaning that, overall, each worker worked on two projects and each repository contained four workers.

With most projects being small and a few large, it is reasonable to conclude that sizes of projects were polarized, indicating that the company’s software development business was a combination of a few principal projects and plenty of small projects. Moreover, if developers’ roles can be implied by the number of projects they worked on and the contribution they made, for example, managers tend to distribute their contribution in a wider range of projects, we are probably able to see the workforce allocation. As it shown in Figure 2, developer nodes with large degree (many projects) but small-weighted edges (small contribution per project) can possibly be interpreted as project managers, while nodes with small degree (small project volume) and small-weighted edges (small contribution per project) can be considered as unproductive workers.



**Senior manager**

**Manager**

**Unproductive**

**Figure 2.** Roles and productivity based on node degree and edge weight

* 1. **A Graph Recommender System**

**[https://www.kernix.com/blog/an-efficient-recommender-system-based-on-graph-database\_p9]**

Recommender systems have been created for various data science applications in a variety of areas. For example, Facebook utilizes recommender systems to suggest friends to users, iTunes and YouTube utilize similar machine learning and recommendation algorithms to suggest songs, videos and movies. Given this general theme, we attempted to create a recommender system to suggest the assignment of developers to projects.

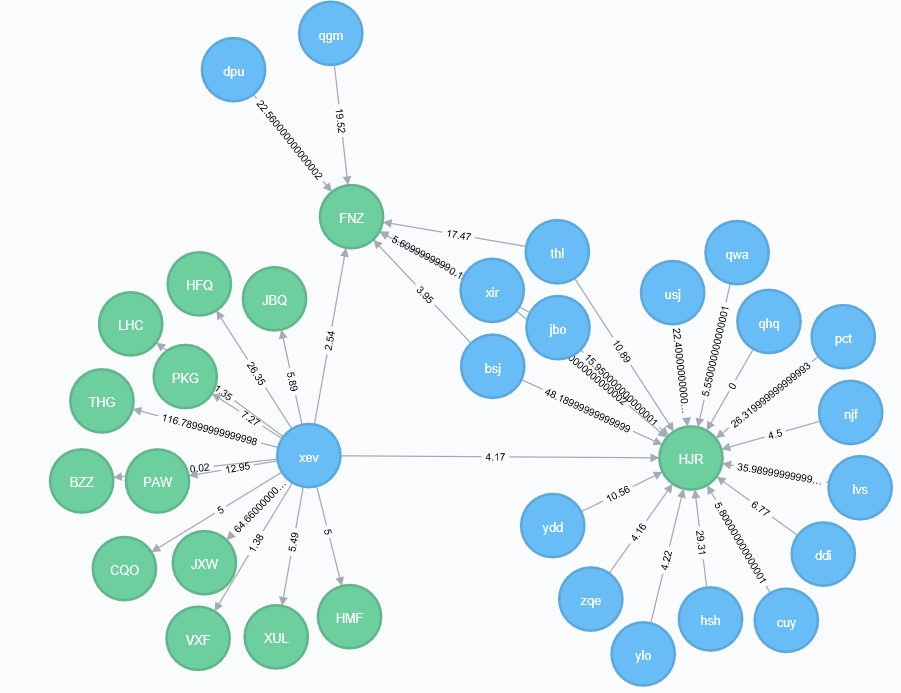
Collaborative filtering (CF), one of the most popular ways to implement recommender system, is a method of making [predictions](https://en.wikipedia.org/wiki/Prediction) or recommendations (filtering) about the interests of a user by collecting preferences or [taste](https://en.wikipedia.org/wiki/Taste_(sociology)) information from [many users](https://en.wikipedia.org/wiki/Crowdsourcing) (collaborating). There are three major types of collaborative filtering algorithms in the recommender system literature, namely memory based approaches (uses user rating data to compute the similarity between users or items), model based Approaches (uses [data mining](https://en.wikipedia.org/wiki/Data_mining), [machine learning](https://en.wikipedia.org/wiki/Machine_learning) algorithms to predict users' rating of unrated items), as well as hybrid approaches (combines the memory-based and the model-based algorithms). Each of those approaches has its own advantages and limitations. [N]

In the situation of developers’ contributing to projects, developers can be modeled as users in the recommender systems. Similarly, projects can be modeled as items and coding effort are ratings. The choice of modeling methods is usually affected by the data available and the purpose of recommender systems. Considering that our recommender system is to make recommendations rather than predictions, and content data (data describing users or items) is unavailable, we’ve chosen to apply memory based approaches.

Memory based approaches has advantages including content-independence, easy implementation and intuitive interpretation. [N] However, it is sensitive and vulnerable to large sparsity data. Given the fact that in our case, there are 2621 users and 1705 items, and over half of users rated only one item, the data is expected to be sparse.

One of the common ways to address sparsity is to apply hybrid approaches. For example, use principle component analysis to compress a high dimensional user-item matrix containing abundant number of missing values into a much smaller matrix in lower-dimensional space. While overcoming the problems of sparsity and loss of information, this method is complex and expensive to implement. [N]

Therefore, instead of hybrid approaches, we turned to graph to address the problem of data sparsity. We firstly defined that two users were neighbors if they shared at least one item. In other words, two developers were neighbors if they worked on at least one project. Instead of computing similarity of every two users, we only did calculations between neighbors.



**Figure 3.** ‘XEV’ was neighbored with both ‘dpu’ and ‘bsj’, by sharing the one project with the former and two with the latter.

To define the similarity of ui to u1, the following function was applied:

Number of Common Projects between u1, ui

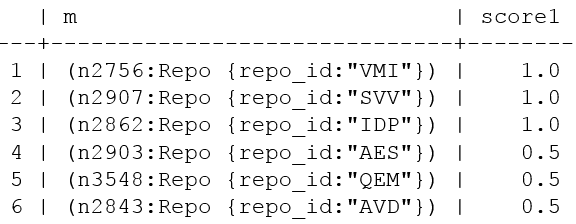
Sim(u1, ui) =

Number of Projects of ui

The similarity of ui to u1 is the ratio of the number of projects they have in common over the number of projects ui works on. We further set a threshold of 0.5 for similarity, which means that developer ui is considered to be similar to developer u1 only if u1 work on at least half of ui’s projects.

We can afterwards make recommendations by answering either the question of what projects popularly shared by similar neighbors, or what projects actively contributed by similar neighbors.

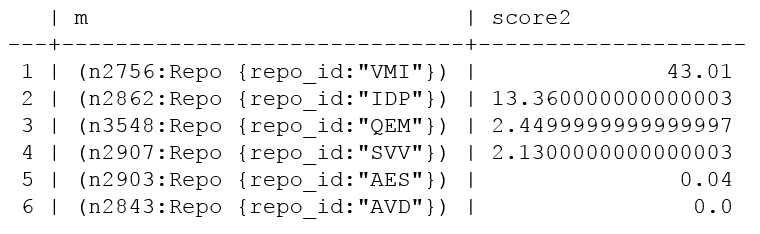
Figure 4 is a list of projects recommended by the system to the developer ‘xev’:



**Figure 4.** Recommender system with similarity threshold being 0.5

The scoring function calculates the ratio of similar neighbors who contribute to the recommended project over those who don’t. Assuming a manager is about to assign the developer ‘xev’ a familiar project, ‘VMI’, ‘SVV’ and ‘IDP’ would be worthy of consideration, given the fact that all xev’s nearest neighbors have worked on them.

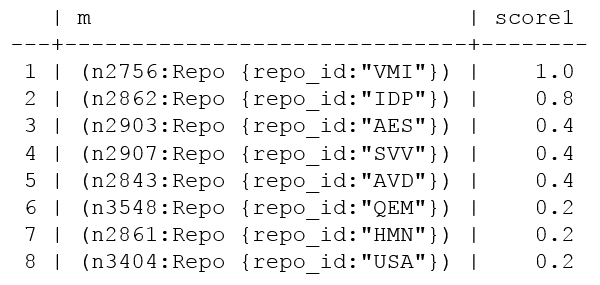
Figure 5 is another list of projects recommended by the system to the developer ‘xev’. Different from the previous system, the scoring function has changed into the sum of similar neighbors’ contribution. Therefore, the system recommends new projects to the developer ‘xev’ in an order from ‘contributed the hardest by similar neighbors’ to ‘contributed the least by similar neighbors’. If a manager wants to ensure developers’ contribution when assigning new projects, he probably prefers this recommendation scheme based on the assumption that a developer will contribute to a project in a comparable way as his similar developers. If nearest neighbors unanimously work hard on a project, the developer is likely to devote too.



**Figure 5.** AnotherRecommender system with similarity threshold being 0.5

More insights can be retrieved if the results of both recommender systems are combined. For example, although both project ‘VMI’ and ‘SVV’ were participated by all the nearest neighbors of developer ‘xev’, the former received massive contribution while the latter had little. Also, although half of nearest neighbors participated in the project ‘AVD’, none of them contributed anything in the concerning period.

Furthermore, we’ve found that different similarity threshold would affect recommendation results largely. For example, if we change the similarity threshold to 0.4, as shown in figure 6, the results would be as follows and ‘VMI’ was the only project shared by all the developer’s nearest neighbors.



**Figure 6.** AnotherRecommender system with similarity threshold being 0.4

An optimized graph-based recommendation system is beyond scope of this paper. Here we just implemented a general-purpose way to demonstrate the simplicity and scalability of a graph-based approach. It doesn’t necessarily require user features or large volume of data for training. Moreover, it helps solve the data sparsity and computation problem of memory-based approaches. [https://arxiv.org/ftp/arxiv/papers/1604/1604.03147.pdf]

* 1. Conclusion

With the weighted bipartite developer-project graph, we gained an overview of the structure and workforce allocation of the company’s software development business, as well as built an efficient graph-based recommender system which facilitates the project assignments of different purposes.

# Modeling Collaboration: Developers-developers graph

When relations among only one class of objects in a bipartite graph are needed, projection is usually applied to transform the graph from two-mode into one-mode. The process works by selecting one set of nodes, and linking two nodes if they are connected to the same node of the other set. However, since one-mode projection is always less informative than the bipartite representation and the redistribution of weights would strongly affect the community structure [17], a proper weighting method is required to better retain the original information.

In this paper, Newman’s weighted projection was applied, which adopts the weighting scheme presented by Newman. This method creates a one-mode network in which the out-strength of a node is equal to the sum of the weights attached to the ties in the two-mode network that originated from that node.

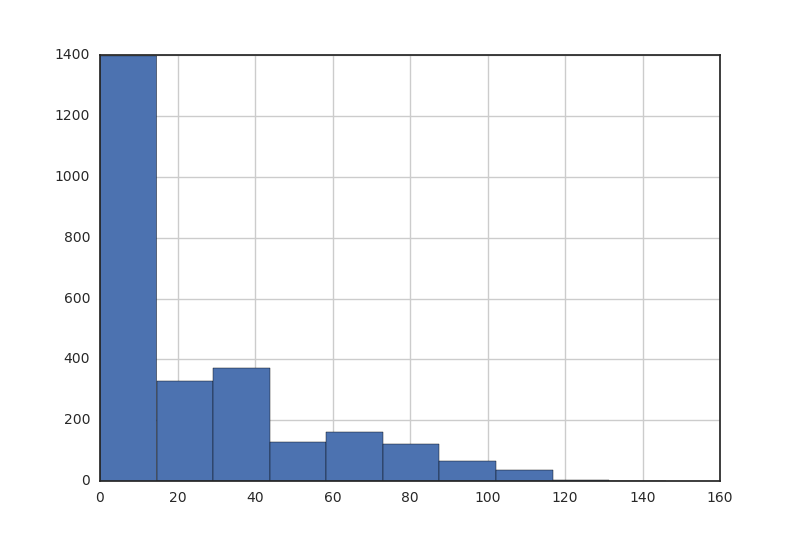
Accordingly, the developer-to-repository graph is projected into two separate one-mode graphs. One is a developer-to-developer graph with 2459 nodes and 61378 edges where each developer is a node and an edge exists between two nodes if both developers are on the same project, and another is a repository-to-repository graph with 1496 nodes and 16764 edges where repositories are nodes and edges represent shared developers.

* 1. **Graph Properties**

With the one-mode developer-developer graph, we can then elaborate on developers and their connections through studying properties of the developer-developer graph.

* + 1. **Degree Centrality**

We firstly applied degree centrality and plotted the distribution of degrees. Figure 7 shows that the degree distribution appears to follow a power-law distribution. In the network theory, a network is named scale-free if its degree distribution follows a mathematical function called a power law [22]. Unlike a random graph (i.e., new nodes attach to existing nodes with uniform probabilities), a scale-free graph displays preferential attachment of new nodes (i.e., some nodes have higher probability of attachment than others) [25]. In our developer-project case, it is intuitive that the network was not randomly formed. It is also in accordance with expectation that a developer, if having diversified skillsets or senior experience, was likely to collaborate with more developers.



**Figure 7.** Node degree distribution for developer graph

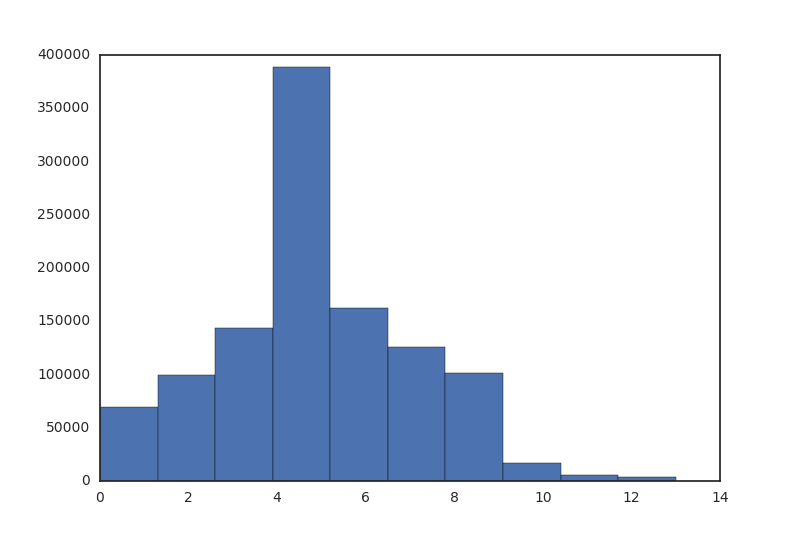
To validate our assumption that nodes in the graph have preferential attachment, we applied a Statistical analysis called Kolmogorov-Smirnov test. [Hazewinkel, Michiel, ed. (2001), ["Kolmogorov–Smirnov test"](https://www.encyclopediaofmath.org/index.php?title=p/k055740), [Encyclopedia of Mathematics](https://en.wikipedia.org/wiki/Encyclopedia_of_Mathematics), Springer, [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-1-55608-010-4](https://en.wikipedia.org/wiki/Special:BookSources/978-1-55608-010-4)] Test results are as follows. The combination of small test statistic (KS.stat) and bigger p-value (KS.p) together indicates a good fit of power-law distribution [23, 24]. As it shown in Table 1, KS.stat is 0.074 and KS.p is 0.99. It therefore proves that the developer-to-developer network has the property of scale-free.

|  |  |  |
| --- | --- | --- |
| **Measurement** | **Definition** | **Value** |
| Alpha | The exponent of the fitted power-law distribution. | 18.67 |
| xmin | The minimum value from which the power-law distribution was fitted. | 206 |
| logLik | The log-likelihood of the fitted parameters. | -45.63 |
| KS.stat | The test statistic of a Kolmogorov-Smirnov test that compares the fitted distribution with the input vector. **Smaller scores denote better fit.** | 0.074 |
| KS.p | The p-value of the Kolmogorov-Smirnov test. Small p-values (less than 0.05) indicate that the test rejected the hypothesis that the original data could have been drawn from the fitted power-law distribution. | 0.99 |

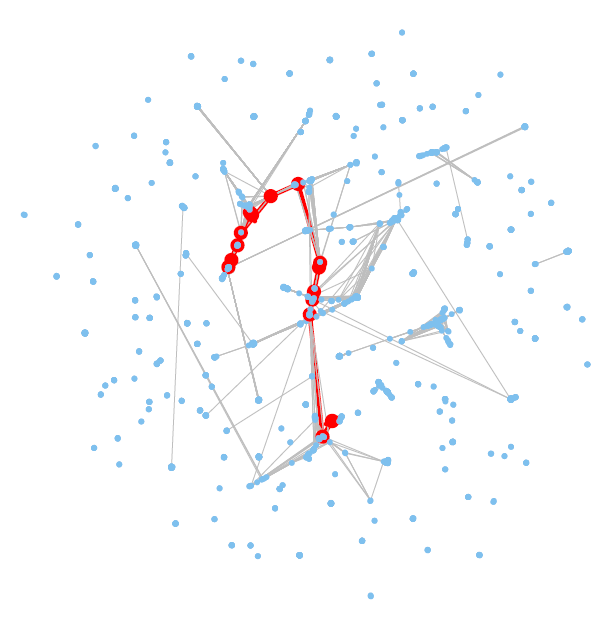
**Table 1.** Statistical tests for power-laws

* + 1. **Shortest Path Length and Closeness Centrality**

Shortest path length is the number of steps along the shortest paths for all possible pairs of network [nodes](https://en.wikipedia.org/wiki/Node_(networking)). Figure 8 shows the distribution of shortest path length in the graph. Averaged at 4.86, length of the shortest path ranges from one to thirteen, meaning that a developer might expect to reach a randomly-selected developer in a typical distance of five, and the distance could be as small as one and as large as thirteen in the company of 2,459 developers.



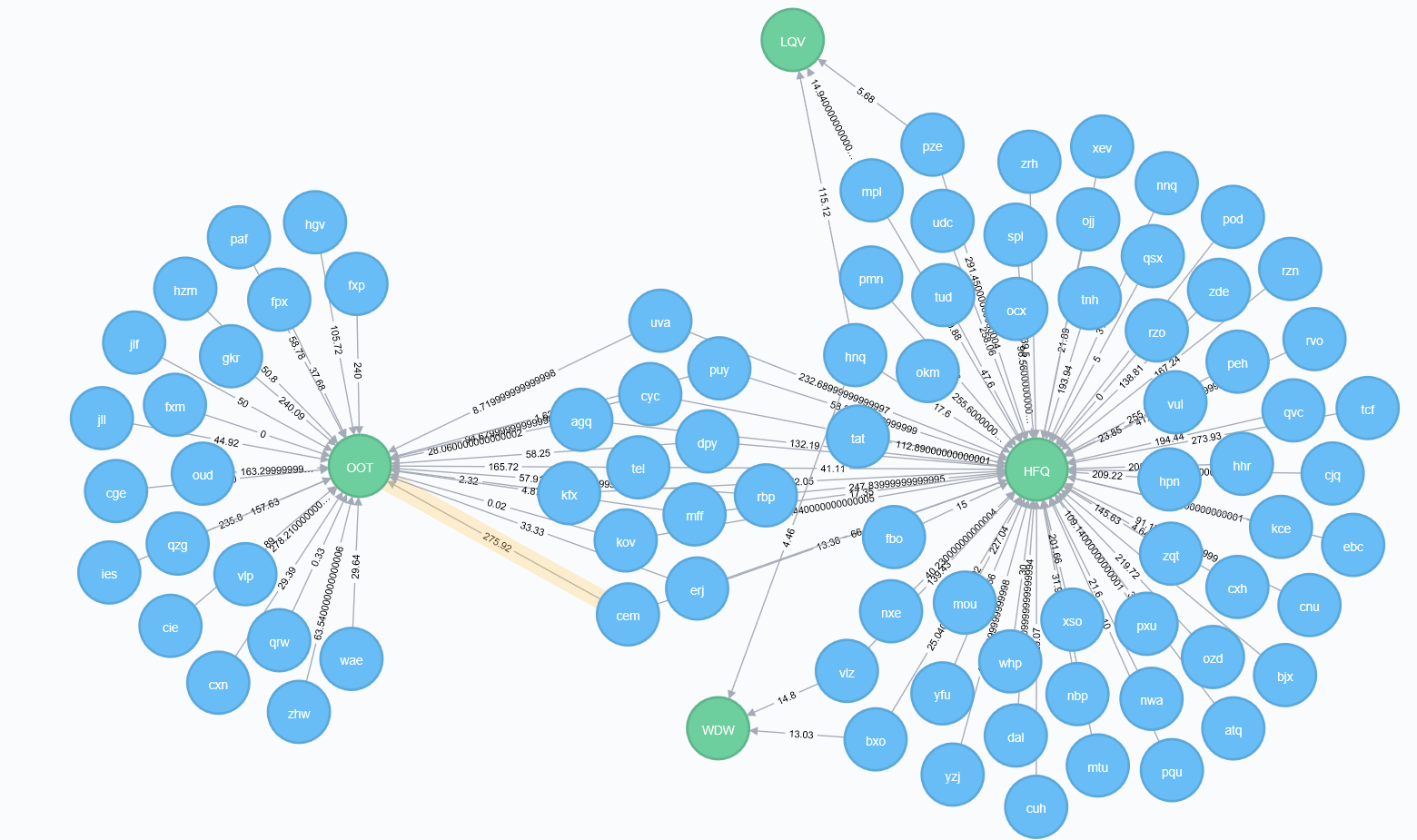
**Figure 8.** Distribution of Shortest Paths in the developers’ network



**Diameter: 13**

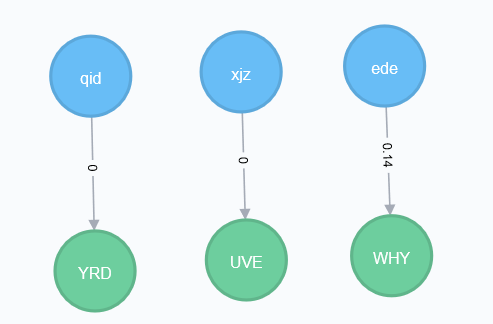
**Figure 9.** Visualization of developer network diameter

Closeness centrality, which uses the reciprocal of the average shortest distance to other nodes, is a measure of the degree to which an individual is near all other individuals in a network. High closeness centrality therefore indicates close access to the resources in the network. In the developer-project network, developer 'rbp', 'kfx', 'agq', 'tel', 'erj', 'kov', 'cyc', 'uva', 'dpy', 'mff', 'puy' and 'cem' were top twelve individuals having closest access. Furthermore, we found that all of them worked on project ‘OOT’ and ‘HFQ’. It can be therefore implied that project ‘OOT’ and ‘HFQ’ were projects gathering massive resource and by working in both, developers were able to get closer access to recourse.



**Figure 9.** Project ‘OOT’ and ‘HFQ’ were shared by developers who had the highest closeness centrality.

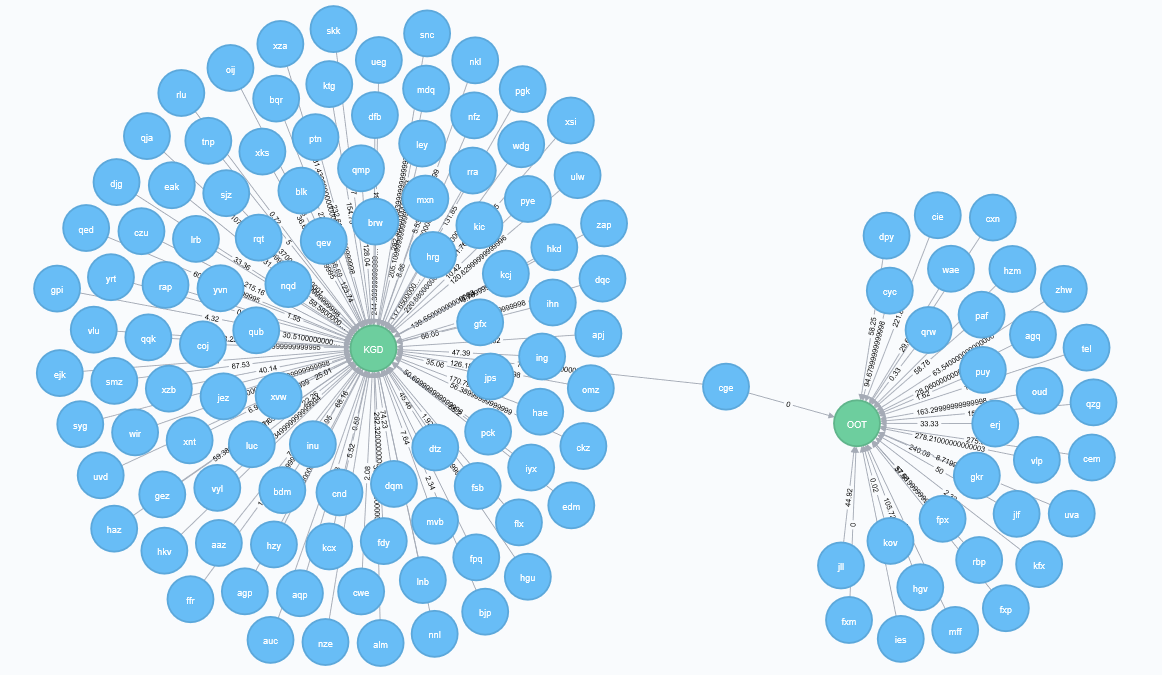
In terms of nodes with smallest closeness centrality, they all exclusively engaged in one individual project.



**Figure 10.** Developers with lowest closeness centrality and their relationship

* + 1. **Betweenness Centrality**

Betweenness centrality for each [node](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) is the number of these shortest paths that pass through the node. Nodes with higher betweenness scores therefore have more control of other nodes in terms of communication access. In the develop-developer network, developer ‘cge’ has the highest betweenness centrality. As shown in the figure 11, ‘cge’ was the only person working in both ‘OOT’ and ‘KGD’, two principal projects in the company. Thus, it is reasonable to infer that developer ‘cge’ played a significant role for the information sharing in between.

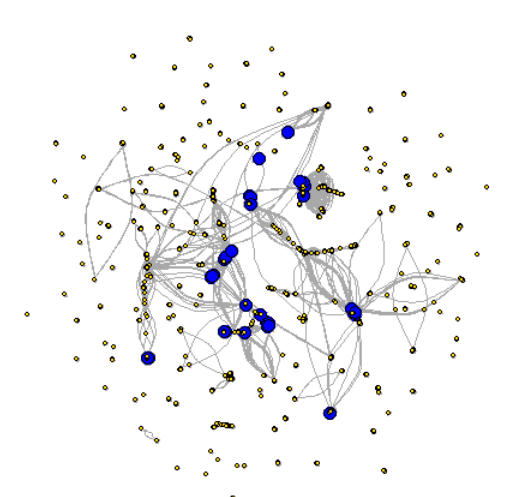


**Figure 10.** Developer ‘cge’ and corresponding connections

* + 1. **Eigenvector Centrality**

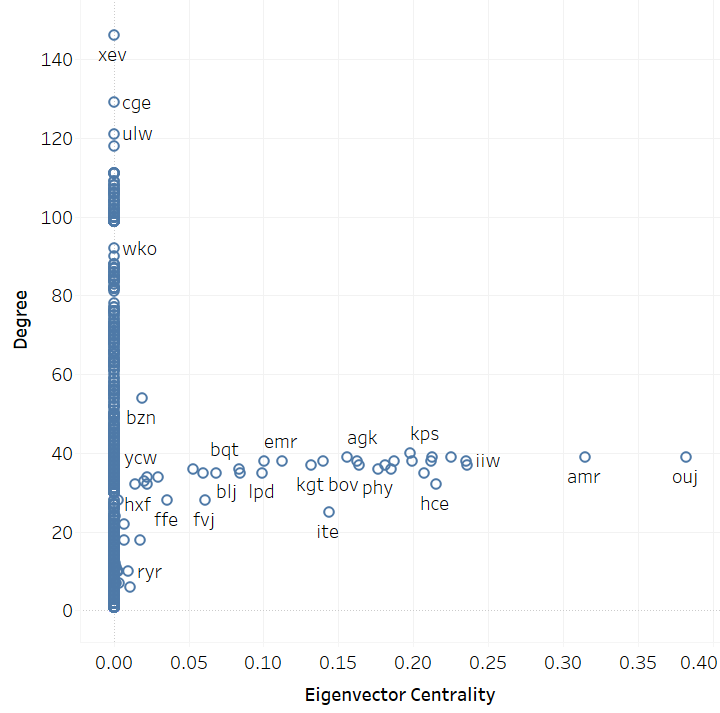
Eigenvector Centrality is a measurement revealing both neighbors’ quantity and quality. In other words, a high eigenvector score is resulted from a large number of high-quality neighbors. High-quality developers can be described as those involving in one or several principal projects, or participating in diversified projects, by which they interact with many people in the network. Correspondingly, low-quality developers are those engaging in unitary or individual projects, by which they collaborate with few others in the network.

As shown in Figure 11, nodes colored in blue are those with highest Eigenvector. They are widely positioned among different subcomponents in the graph and are connected with different sets of nodes.



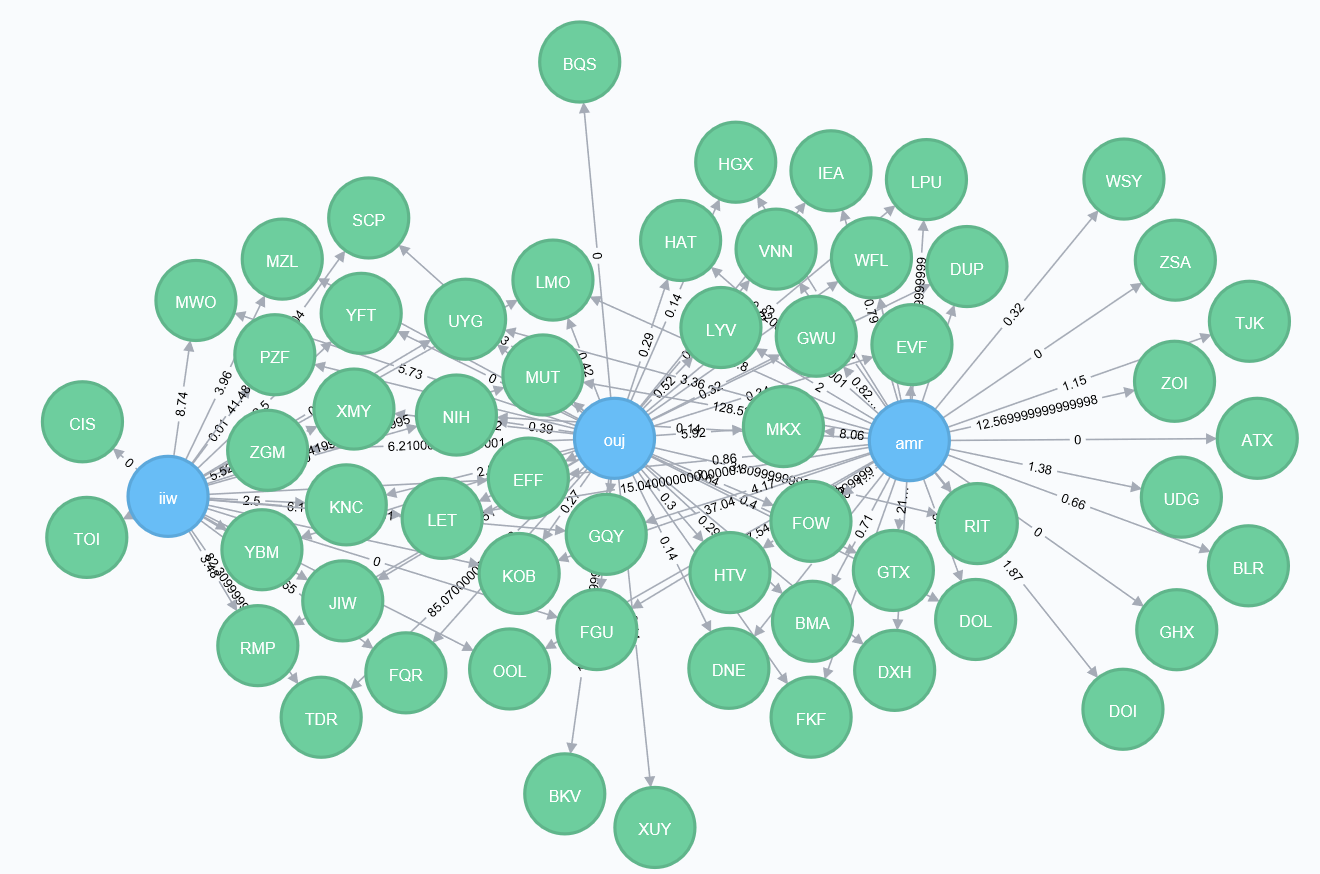
**Figure 11.** Visualization of developers with high eigenvector scores

Figure 12 is the scatter plot showing eigenvector centrality versus degree. We can see that most nodes has an eigenvector centrality of about zero. Also, although that ‘xev’ has more than three times degree than ‘ouj’, ‘ouj’ has the largest eigenvector centrality while ‘xev’ the least.

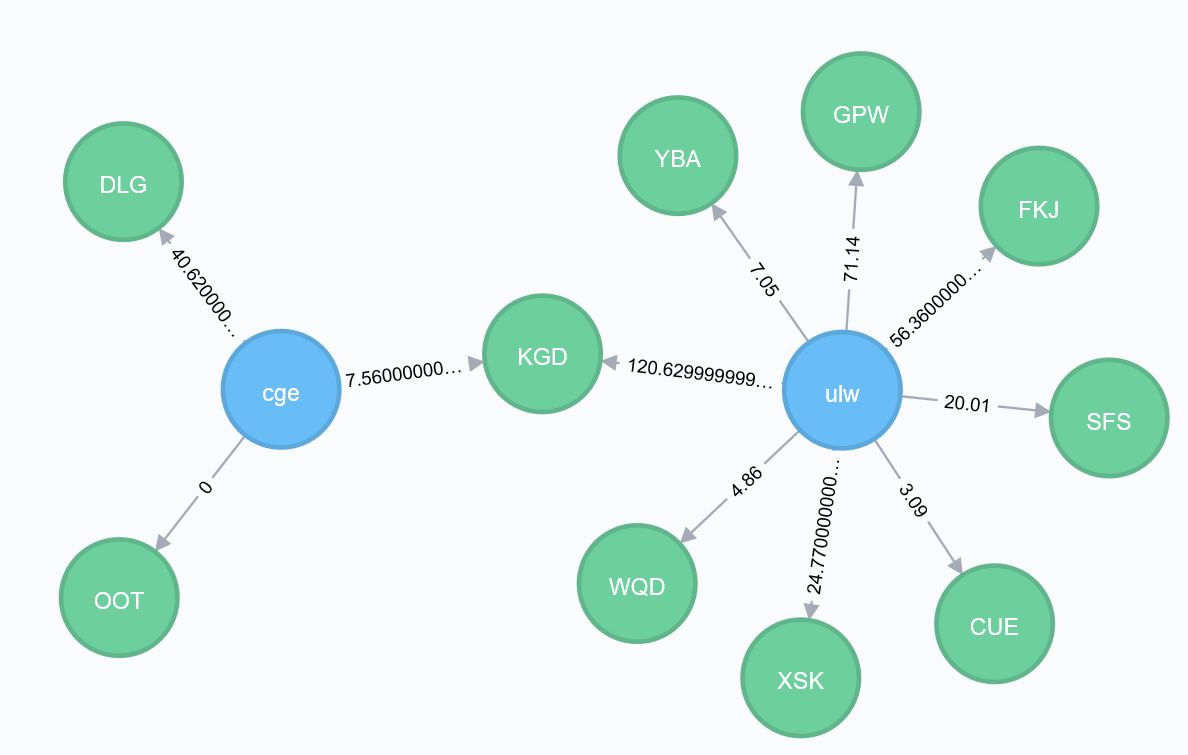


**Figure 12.** scatter plot of degree and eigenvector centrality

In addition, other than developer ‘ouj’,’amr’ and ‘iiw’ are another two having high eigenvector centrality scores, while developer ‘xev, ‘cge’ and ‘ulw’ have the lowest. Furthermore, as it shown in figure 13, ‘ouj’,’amr’ and ‘iiw’ were intensively sharing projects. It can be therefore implied that those shared projects were worked by a lot of high-quality developers, bringing those three developers high-quality neighbors and thus high eigenvector centrality. On the contrary, as it shown in figure 14, ‘cge’ and ‘ulw’ shared only one project, ‘KGD’. Similarly, it suggests that ‘KGD’ was a project worked by plenty of low-quality developers, giving both ‘cge’ and ‘ulw’ low-quality neighbors and thus low eigenvector centrality.



**Figure 13.** Developer ‘ouj’,’amr’ and ‘iiw’ and their connections



**Figure 14.** Developer ‘cge’ and ‘ulw’ and their connections

* 1. **Community Detection**

Communities in a network are groups of nodes internally connected or nodes sharing attributes. Detecting communities provides insights regarding the overall network structure, behavioral patterns of nodes and their relations. For the developer-developer network, it supports findings about developers’ roles, collaboration preference as well as corporate cohesion.

This paper chooses *R* package *igraph* [32] as a main tool to conduct community detection.Applicable algorithms including Edge-Betweenness, Leading Eigenvector, Fast-Greedy, Walktrap, Label Propagation and Infomap were considered. Summary and comparison of those algorithms in the package are shown in Table 2.

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Directed Edges** | **Weighted Edges** | **Multiple Components** |
| *Edge-Betweenness* | TRUE | TRUE | TRUE |
| *Leading Eigenvector* | FA LSE | FALSE | TRUE |
| *Fast-Greedy* | FALSE | TRUE | TRUE |
| *Multi-Level* | FALSE | TRUE | TRUE |
| *Walktrap* | FALSE | TRUE | FALSE |
| *Label Propagation* | FALSE | TRUE | FALSE |
| *Infomap* | TRUE | TRUE | FALSE |

**Table 2.** Community detection algorithms considered and their properties

Considering that the developer-developer network is a weighted undirected graph, four out of seven algorithms above were implemented, namely Multi-Level, Walktrap, Label Propagation and Infomap.

Multilevel models are [statistical models](https://en.wikipedia.org/wiki/Statistical_model) of [parameters](https://en.wikipedia.org/wiki/Parameter) that vary at more than one level. The multi-level modularity optimization algorithm for finding community structure is based on the modularity measure and a hierarchical approach [https://arxiv.org/abs/0803.0476]. It works according to the following steps: firstly, each node is assigned to a community independently. Secondly, each node is moved to the community in a local, greedy way, where it achieves the highest contribution to modularity. Thirdly, when no node can be reassigned, each community is considered as a node on its own, and repeat the second step. The process stops either when there is only one node left or when the modularity cannot be increased any more.

Walktrap is a method based on random walks. With short random walks having the tendency of staying in the same community, densely connected subgraphs, or communities, can be found. [ ]

Label propagation works in line with the following methodology: firstly, each node is assigned to a unique label. Secondly, each node changes its label to a dominant label in its neighborhood in each iteration. Both breaking edges and updating nodes happen in a random way before every iteration. The process stops when nodes reach a consent. [ ]

Infomap model tries to build a grouping which provides the minimum description length for a random walk on the graph. The minimum description length (MDL) corresponds to an implementation of [Occam’s razor](https://en.wikipedia.org/wiki/Occam%27s_razor), where the simplest model is selected, among all possibilities with the same explanatory power. Therefore, it can avoid the problem of overfitting. [ ]

To measure the performance of those algorithms, we used modularity, a common measurement of dividing a network into communities [33]. Networks with high modularity have dense connections between the nodes within modules but sparse connections in different modules.

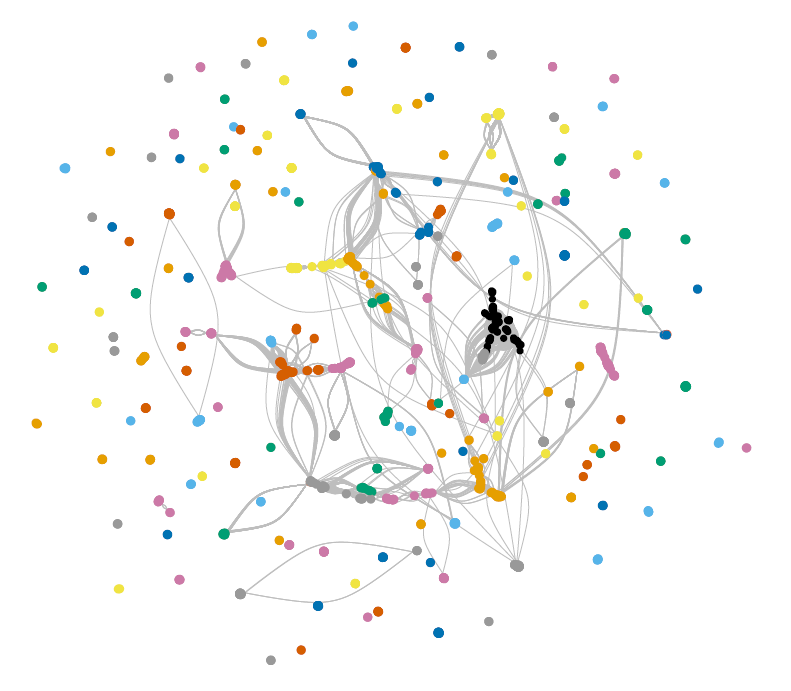
Performance for each algorithm is shown in Table 3 below. Considering both of modularity and group number, we concluded that Multi-Level algorithm performed the best, followed by the Walktrap algorithm.

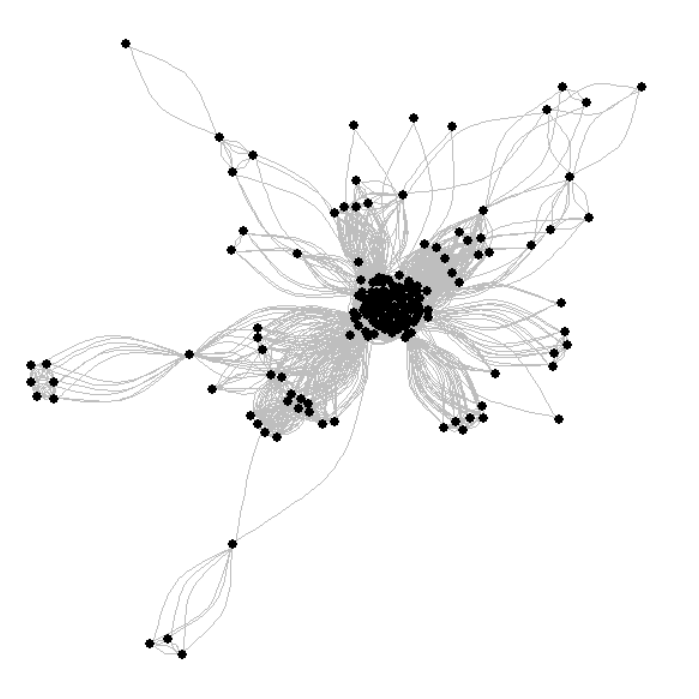
|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Modularity** | **Groups** |
| Multi-Level | 0.97 | 172 |
| Walktrap | 0.96 | 193 |
| Label Propagation | 0.89 | 288 |
| Infomap | 0.16 | 1119 |

**Table 3.** Performance of algorithms used for community detection

In addition, according to the Multi-Level algorithm, the largest community in this company contained 173 developers while the smallest (six communities) consisted of just two developers. On average, communities contained 98 developers. Figure 15 displays 172 groups with different colors, among which the largest group is colored in black.

A comprehensive and optimized way of inferring network structure is beyond the scope of this paper. Here we applied some easy-accessible algorithms, such as hierarchical clustering model (Multilevel model), stochastic model(Walktrap) and non-parametric model (Label propagation), to demonstrate that detecting communities is a convenient way to simplify and characterize the structure of a complex network.





**Figure 15.** Visualization of community detection in the developer’s network

## Summary

To understand the interaction among developers, we created a weighted developer-developer graph by projecting the weighted bipartite developer-project graph. Four graph properties, namely degree, closeness, betweenness and eigenvector centrality, were studied, from which we achieved implications about developers’ functionality, roles and relationships. In addition, we compared and implemented different community detection algorithms, including popular modularity maximization approach, to simplify and capture the network structure.

# Modeling Dynamics: Evolving Network

Analyzing dynamic network has gained increase popularity and significance as longitudinal network data becomes more available. Organizations study corporate network in a dynamic way in order to discover the decision cycle of major events, track the evolutions of corporate finances, operations and culture, as well as detect changes in the organizational behavior. [ ] Being able to identify a change in network dynamics can enable managers to better prepare for the coming change and prevent potential problems. For example, before the introduction of e-mail and cell phones into the workplace, workers had limited peers available that they could ask about problems before they had to seek guidance from senior management. However, with growing on-line communities, the available peers to consult are no longer limited to acquaintances. Individual network is becoming larger. While this is good that workers are able to resolve problems at a lower level, senior managers are unable to influence decisions with their senior

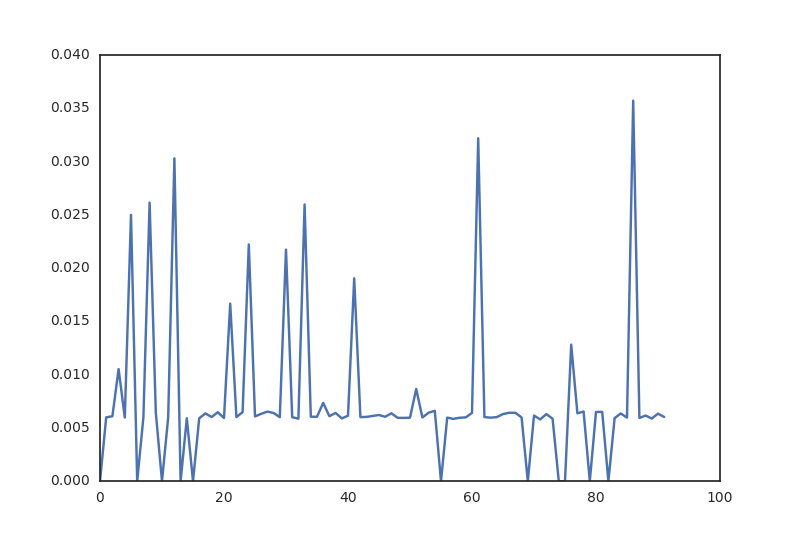
judgment and experience. Dynamic network analysis therefore can provide those managers with a tool to prevent potential problems in their organization by tracking every change in the social network of employees. [http://www.casos.cs.cmu.edu/publications/papers/CMU-ISR-09-118.pdf]

Dynamic network analysis (DNA) brings together traditional social network analysis, link analysis, social simulation and multi-agent systems within network science and network theory. There are two aspects of this field. The first is the statistical analysis of DNA data. The second is the utilization of simulation to address issues of network dynamics. [https://en.wikipedia.org/wiki/Dynamic\_network\_analysis]. Dominant modeling methods include Markov chain models, multi-agent simulation models, and statistical models. [http://www.casos.cs.cmu.edu/publications/papers/CMU-ISR-09-118.pdf]

A complete application of dynamic network modeling is beyond the scope of this paper. Considering the nature of the given problem, we choose statistical models as our major measure. In other words, we compare the properties of networks at different points in time.

[figures of graph at different time points]

[density trend (daily), waiting to be modified, expecting zero on weekends…]



# Discussion and concluding remarks

# A study of the proprietary software development was started with constructing weighted graphs. Afterwards, initial insights about developers’ role (managers or engineers), productivity and individual influence were achieved through analyzing network properties. Finally, implications about software development cohesion and culture were developed through implementation of cluster analysis as well as a comparison between constructed networks and complex systems.

# Several assumptions and limitations are present in the study. For example, the analysis only looked at a snapshot or aggregated version of the software development network at this particular company. A longitudinal and dynamic analysis in the future may provide better understanding of how node attach and detach from the network (developers dropping off or initiating projects), and thus lead to a more thorough understanding of developers’ working pattern.

# References

[1] Statista. (2014). Worldwide mobile app revenues from 2011 to 2017 (in billion U.S. dollars). Retrieved from <http://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/>

[2] McGarry, John (2001) Practical Software Measurement, p19. Addison-Wesley.

[3] UKESSAYS (19) Importance of Software Measurement and Metrics. Retrieved from <https://www.ukessays.com/essays/information-technology/importance-of-software-measurement-and-metrics-information-technology-essay.php>

[4] Wu, M.W., Lin, Y. D. (2001). "Open Source Development: An Overview," IEEE Computer, pp. 33-38.

[5] S.Z. Yang, S.W. Luo. (2009). Modern Physics Letters B 23, 3209.

[6] Newswire (2013). BlueOptima Coding Effort Analytics Announces Support for Git. Retrieved from <https://www.newswire.com/blueoptima-coding-effort-analytics/252989>.

[7] Asratian, Armen S., Denley, Tristan M. J., Häggkvist, Roland. (1998). Bipartite Graphs and their Applications, Cambridge Tracts in Mathematics, 131, Cambridge University Press, ISBN 9780521593458.

[8] Guillaume, J., Latapy, M. (2004). Bipartite structure of all complex networks. Information Processing Letters, 90(5):215-221.

[9] Newman, Mark, Albert-László Barabási, Watts, Duncan J. (2006). The Structure and Dynamics of Networks, The Princeton Press, ISBN 0-691-11357-2

[10] Newman, M.E.J. (2010). “Networks: An Introduction.” Oxford, UK: Oxford University Press.

[11] Bonacich, Phillip (1987). "Power and Centrality: A Family of Measures". American Journal of Sociology. University of Chicago Press. 92: 1170–1182. doi:10.1086/228631.

[12] Borgatti, Stephen P. (2005). "Centrality and Network Flow". Social Networks. Elsevier. 27: 55–71. doi:10.1016/j.socnet.2004.11.008.

[13] Borgatti, Stephen P.; Everett, Martin G. (2006). "A Graph-Theoretic Perspective on Centrality". Social Networks. Elsevier. 28: 466–484. doi:10.1016/j.socnet.2005.11.005.

[14] Benzi, Michele; Klymko, Christine (2013). "A matrix analysis of different centrality measures". arXiv:1312.6722Freely accessible.

[15] Kamada, Tomihisa, Kawai, Satoru (1989), "An algorithm for drawing general undirected graphs", Information Processing Letters, Elsevier, 31 (1): 7–15, doi:10.1016/0020-0190(89)90102-6

[16] Zhou, Tao, Ren, Jie, Matúš Medo, Zhang, Yi-Cheng. (2007). "Bipartite network projection and personal recommendation". PHYSICAL REVIEW E 76(4): 046115.

[17] Fan, Ying, Li, Menghui, Zhang, Peng, Wu, Jinshan, Di, Zengru. (2007). "The effect of weight on community structure of networks". PHYSICA A 378. 583–590.

[18] Klein, David R. (2012) "Chapter 4 Section 6: Drawing Newman Projections." Organic Chemistry. 2nd ed. N.p. Wiley. 158-59. Print.

[19] Ladyman, James; Lambert, James; Wiesner, Karoline (2013). "What is a Complex System?". European Journal for Philosophy of Science. 3: 33–67. doi:10.1007/s13194-012-0056-8.

[20] Simon, Herbert A. (1991). "The architecture of complexity". In George J. Klir. Facets of Systems Science. Springer US. pp. 457–476. doi:10.1007/978-1-4899-0718-9\_31.

[21] Albert R.; Barabási A.-L. (2002). "Statistical mechanics of complex networks". Rev. Mod. Phys. 74: 47–97. arXiv:cond-mat/0106096Freely accessible. Bibcode:2002RvMP...74...47A. doi:10.1103/RevModPhys.74.47.

[22] Clauset, A.; Shalizi, C. R., Newman, M. E. J. (2009). "Power-Law Distributions in Empirical Data". SIAM Review. 51 (4): 661–703. arXiv:0706.1062Freely accessible. Bibcode:2009 SIAMR..51..661C. doi:10.1137/070710111.

[23] Stephens, M. A. (1974). "EDF Statistics for Goodness of Fit and Some Comparisons". Journal of the American Statistical Association. American Statistical Association. 69 (347): 730–737. doi:10.2307/2286009. JSTOR 2286009.

[24] Marsaglia G, Tsang WW, Wang J (2003). "Evaluating Kolmogorov's Distribution". Journal of Statistical Software. 8 (18): 1–4.

[25] Béla Bollobás, Random Graphs, 2nd Edition, 2001, Cambridge University Press

[26] Barabási, Albert-László (2004). Linked: How Everything is Connected to Everything Else. ISBN 0-452-28439-2.

[27] Holland, P. W., Leinhardt S. (1971). "Transitivity in structural models of small groups". Comparative Group Studies. 2: 107–124.

[28] Opsahl, Tore, Panzarasa, Pietro. (2009). "Clustering in Weighted Networks". Social Networks. 31 (2): 155–163.

[29] Kaiser, Marcus. (2008). "Mean clustering coefficients: the role of isolated nodes and leafs on clustering measures for small-world networks". New Journal of Physics. 10 (8): 083042. arXiv:0802.2512Freely accessible.

[30] Watts, D. J., S. H. Strogatz, (1998). Nature (London) 393,440

[31] Girvan, M., Newman, M. E. J. (2002). "Community structure in social and biological networks". Proc. Natl. Acad. Sci. USA. 99 (12): 7821–7826. doi:10.1073/pnas.122653799. PMC 122977Freely accessible. PMID 12060727

[32] Gábor Csárdi, Tamás Nepusz. (2006). “The igraph software package for complex network research. InterJournal Complex Systems.” necsi.edu.

[33] Baldwin, C. Y., Clark, K. B. (2000). “Design rules, Volume 1: The power of modularity.” Cambridge, MA: MIT Press.