Mining Email Archives and Simulating the Dynamics of Open-Source Project Developer Networks

Liguo Yu[1], Srini Ramaswamy[2], and Chuanlei Zhang[2]

[1] Computer Science and Informatics, Indaian University South Bend, South Bend, IN, USA
ligyu@iusb.edu

[2] Computer Science Department, University of Arkansas at Little Rock, Little Rock, AR, USA
{sxramaswamy, cxzhang}@ualr.edu

Abstract. In distributed open-source software projects, participation of developers is largely by a voluntary basis. Programmers are not only free to join in or to leave the project, but they are also free to choose who they interact with, and how much they contribute to the project. In that sense, open-source project developers form a dynamic social network. This paper presents a measure to represent the interactions of distributed open-source software developers and utilizes data clustering techniques to mine their email archives to derive a representation of the associated social network. This method is applied on case studies of three social networks from two open-source projects, Linux and KDE. The dynamics of the three networks are then analyzed and simulated using agent-based modeling techniques. Our study shows that the three open-source developer networks evolved over time with some predictable patterns.

Keywords: Social network, open-source project, data mining, email archive, agent-based simulation.

1 Introduction

Software development process is not only the development work of developers, the interactions between them is also very important. Theses interactions which form a social network is critical for cooperation, issue resolution, and information sharing [9], [18]. Therefore, studying the organization of the social network is important to understanding and improving the software process [1], [8].

Open-source software consists of programs whose licenses give its users the freedom to run the program, to study and modify the source code, and to redistribute copies of the program [20]. The open-source software development process does not adhere to the traditional organizational structure and associated rationality found in the closed-source software development process. Instead, open-source software programs are built by an informal group of volunteers who work in a distributed environment. Communication and coordination between the developers are through emails and shared repositories, such as manuals, design documents, source code and bug reports. Therefore, open-source software development can be thought of as a complex web of socio-technical processes and development situations sustained within a global interaction network [19]. Studying such an open-source software

development network can provide an interesting perspective into how open-source software systems are constructed and evolve over time. It can also help us understand how such networks should be constructed and structured to improve the resulting development process.

One important difference between closed-source software organization and open-source software organization is the stability of the developer network. In closed-source projects, the developers are relative stable with respect to the activities they perform, other developers they interact with, and their respective roles; while in open-source projects, participation is largely voluntarily, programmers are not only free to join in or leave the project, but they are also free to choose who they interact with, and how much they contribute to the participating project. Accordingly, the open-source project developer network is highly dynamic and evolves continually over time.

Due to this loose management property, the information about open-source project organization, alliance formation, and communication network is not very well documented. However, open-source software projects contain email archives, which constitute an extensive on-line record of user feedbacks, issue resolutions, and problem-solving behaviors [21]. This data is publicly available and amenable to modern data mining techniques so that we can extract useful information on various development organizations and associated development processes [3], [4]. On the other hand, simulation is a powerful technique to model the behavior of social networks. Therefore, in this paper, we apply both data mining techniques and simulation techniques to study the dynamics and the evolution of open-source social networks.

2 Mailing List and Communication Network

In this study, we use communications in mailing lists to construct and analyze the social network. First, we introduce some terminologies and metrics to represent mailing lists and the interactions between developers.

A *message* is the smallest unit of information posted by one person at a time. A message can be either an initiating message that starts a new topic or a replied message that responds to other messages (either a new message or a replied message).

A *thread* is defined as a collection of messages that discuss the same topic. A thread contains one initiating message and zero or more replied messages.

A *poster* is a person posting a message on the list, who is either an initiator or a replier. An initiator is one who posts an initiating message on the mailing list. A replier is one who posts a replied message on the mailing list.

A mailing list is a forum for project managers, developers, bug-reporters, and users to exchange ideas, report problems, and find solutions. Any posted messages will be delivered to all the subscribers. Although messages are delivered to all the subscribers, most of the topics are not of interest to regular subscribers. Instead, a message thread might only be interesting to those subscribers who participate in the discussion in this thread.

Therefore, communications between two posters who posts the replied message and who receives the replied message (more accurately, the target audience of the replied message) form a channel. A channel could be either a one-way or a two-way channel. A one-way channel exists between two posters P1 and P2, in which P1 replies to the message posted by P2. A two-way channel exists between two posters P1 and P2, in which both P1 and P2 reply to the messages posted by each other.

A one-way channel (say P1→P2) represents the service relationship between P1 and P2, i.e., P1 answers or comments on P2's message. A two-way channel (say P1↔P2), represents the collaboration/coordination relationship between P1 and P2, i.e, P1 and P2 discuss some common interesting topics.

The organization of the communication network can be represented by two measures, bandwidth and interaction degree. The bandwidth of a channel is the measure of the communication frequency between two members: The bandwidth of a one-way channel P1→P2 is the number of messages posted by P1 that is a reply to the message from P2; The bandwidth of a two-way channel P1↔ P2 is the number of messages posted by P2 to reply a message from P1 plus the number of messages posted by P1 to reply a message from P2.

The interaction degree is the number of channels between one particular member and all other members in the network. Generally speaking, a poster with a larger interaction degree tends to play a more central role in the community, because s/he interacts with more members [21], [11].

3 Research Procedures

This study contains five steps. Each of these steps is identified and further elaborated below.

Step one: interaction representation

To represent the degree of interactions between developers, we adopt a terminology, *interaction frequency* (IF) [7], [6]. For two developers i and j, interaction frequency represents the degree of interactions between i and j based on one or more measures between them. It is represented as $IF_{i,j}$.

The measurement of interaction frequency is a context-based concept, which means different measures may result in different interaction frequency. In distributed open-source development, candidate measures for interaction frequency are the frequency of email correspondence, the frequency of co-editing, the frequency of task sharing, and so on. In this study, email correspondence between developer i and developer j is used as the measure of interaction frequency $IF_{i,j}$.

Interaction frequency gives the representation of the degree of interaction between two developers. What we are interested is a large project that contains many developers. Therefore, we define *interaction matrix* (IM). For a project that contains n developers, the degree of interactions between these n developers is represented as an $n \times n$ interaction matrix (*IM*), in which item at position (i, j) is the interaction frequency between developer i and developer j.

Step two: clustering

Clustering is a data mining technique to group items into clusters according to their similarities, differences, or distances [10]. In this research, we use single-linkage hierarchical clustering method [12] to group distributed developers according to the interaction frequencies between them.

Step three: network construction

In a distributed open-source development environment, developers take different roles. In our preliminary study [7], we found that developers that are in a cluster with greater interaction frequency (CIF) are more active and take more responsibility than those in a cluster with lower interaction frequency. Therefore, it is reasonable to claim that developers clustered earlier take more important roles than those clustered later. The entire developer network can be constructed according to the clustering result. Since some open-source projects involve over thousands of developers, to simplify the analysis and to illustrate our approach clearly, in this paper, we construct the network using the first 100 developers that are clustered and ignore the rest of the developers. We call this network central-100 network. The members in central-100 network are called central members. We assume that the behavior of the central-100 network represents the behavior of the entire network. Through analyzing the organization and the evolution of the central-100 network, we intend to understand the dynamics of the entire open-source developer network.

Step four: network dynamics analysis

To study the dynamics of the networks, we need to study the evolution of the organization of the social network, which includes the evolution of average bandwidth, the evolution of the average interaction degree, and the updating of the central members. Two metrics are used to measure the updating of the central members, annual updating rate and age distribution, which will be further explained later.

Step five: modeling and simulation

Agent-based simulation is a special type of discrete simulation, in which the individual entities (agents) are represented with an internal state and a set of rules which determine how the agent's state is updated from one time-step to the next [16]. Agent-based simulation has been widely used in constructing models of software organization and software process [8], [15], [22], [17], [5]. In this step, we build agent-based models for open-source project networks. The simulation results are used to evaluate the developer network evolution model.

4 Case Studies

In this research, three email archives of two open-source projects, Linux and KDE, are mined and used to study the open-source developer network.

The Linux-kernel mailing list (linux-kernel@vger.kernel.org) is maintained by vger.kernel.org to provide email list services for the Linux kernel developers [14].

Although there are several other mailing lists on specific subjects, such as linux-net@vger.kernel.org for networking users and netdev@vger.kernel.org for networking developers, the Linux-kernel mailing list (LKML) is the official and most heavily used communication platform for Linux kernel development. The earliest archived LKML message we found is in June 1995. Until now (February 2007), LKML has been used as the glue that holds the Linux kernel development community together.

In contrast to Linux, KDE has four development mailing lists [13]. KDE-development list is for application developers (both applications in central KDE packages and contributed applications). KDE-core list is used for discussions of KDE libraries development, SVN and other central development issues. KDE-quality list focus improving the general quality level of KDE applications. KDE-commits list carries automatic notifications for all changes made to KDE's source code repository. In these four lists, KDE-core list and KDE-development list contain the communication history of KDE developers are most suitable to study the developer network.

Therefore, Linux-kernel, KDE-core, and KDE-development mailing lists are chosen for studying the corresponding developer networks. In the remainder of this paper, they are referred to as Linux, KDE-core, and KDE-development respectively.

4.1 General results

A total of over 611k, 14k, and 37k messages over seven years (2000-2006) of communication were mined from the email archive of Linux, KDE-core, and KDE-development. Table 1 shows the total number of messages and total number of threads posted to the three mailing lists in these different years. As we can see, the number of messages and the number of threads of Linux have an increasing trend, while the number of messages and the number of threads of KDE-core and KDE-development give a decreasing trend. We can also find that Linux mailing list carries more messages than the KDE lists. For example, Linux mailing list had about 100 times of messages over KDE-core and KDE-development lists in year 2006. Therefore, the Linux and KDE mailing lists represent different scales and different evolutionary trends of developer social networks.

Table 1. The total number of messages and total number of threads in three mailing lists.

Year	Linux		KDE-core		KDE-development	
	Number of messages	Number of threads	Number of messages	Number of threads	Number of messages	Number of threads
2000	61423	16814	3602	1143	8030	3866
2001	69507	18198	2267	732	6269	2249
2002	82843	22789	2366	678	5286	1832
2003	87816	24491	1652	511	3282	1108
2004	94730	24387	1367	341	2201	729
2005	98870	25015	1227	304	1785	433
2006	116238	27787	1057	222	1221	298

Table 2. The number of posters and initiators in the three mailing lists.

Year	Linux		KDE-core		KDE-development	
	Number of posters	Number of initiators	Number of posters	Number of initiators	Number of posters	Number of initiators
2000	5543	4557	212	164	1435	1240
2001	6603	6143	226	157	1030	741
2002	5941	4926	307	200	1052	755
2003	6454	5401	282	186	844	562
2004	6493	5422	269	155	632	412
2005	5917	4864	265	143	489	258
2006	5925	4989	222	100	376	182

Table 2 shows the number of posters and the number of initiators in the three mailing lists during the time period of our investigations. A poster can post one or many messages and an initiator can post one or many initiating messages. The number of posters represents the number of developers in the social network. We can see that the size of the three networks evolves differently: Linux and KDE-core change a little bit while KDE-development decrease dramatically.

Next, we need to determine which metric in the mailing list can be used to represent the interaction frequency (IF) between two developers. Because in the clustering process, two members, P1 and P2, are considered equally important (active) regarding the channel bandwidth between P1 and P2, a two-way communication channel is superior to one-way channel. Therefore, we use the bandwidth of two-way communication channel to represent interaction frequency, to cluster the developers, and to construct the social network.

4.2 Social network

As stated in Section 3, in this paper, we study the evolution of central-100 network. Figure 1 shows part of the Linux central-100 developer network of 2006, in which different thickness of the link represents different bandwidth. The largest bandwidth has value 318 and exists between Andrew and Adrian, while the smallest bandwidth has value 73 and exists between Eric and Oleg.

Figure 1 shows that a few members, such as Andrew and Andi, have larger channel bandwidth and higher interaction degrees, while most others have smaller channel bandwidth and lower interaction degrees. This observation agrees with what Bird el al. found - that a few members account for the bulk activities of the network [3], [4]. This property is further illustrated in Figure 2 and Figure 3, which shows the distribution of channel bandwidths and interaction degrees respectively.

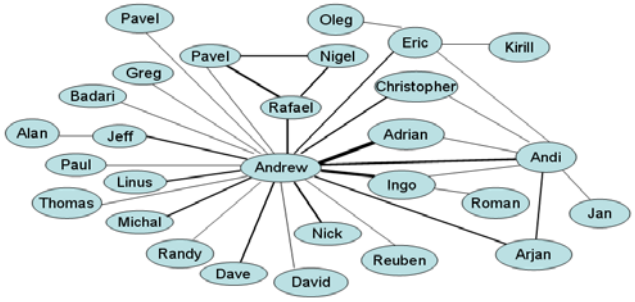


Fig. 1. Part of the central-100 network of Linux in 2006.

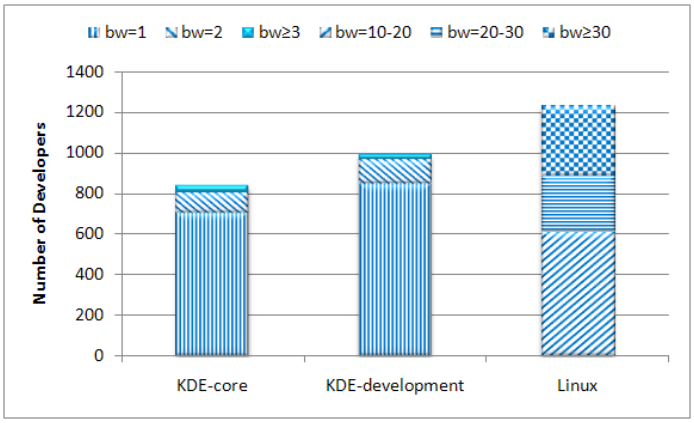


Fig. 2. The distribution of the channel bandwidth of the entire three networks in 2006.

Figure 2 shows the distribution of the channel bandwidth of three entire networks in 2006, in which *bw* represents channel bandwidth. This figure only shows the bandwidth greater or equal to 10 of Linux (the bandwidth that is smaller than 10 accounts for 96% channels in Linux; for the purpose of clear illustration, it is not shown in the figure), while KDE-core and KDE-development shows all bandwidth greater than or equal to 1.

It can be seen that the distribution of bandwidth is not uniformly distributed: more channels have smaller bandwidth while fewer channels have larger bandwidth. This is also found in the Linux central-100 network as shown in Figure 1.

Figure 3 shows the composition of the developers in central-100 network with different interaction degrees (ID) in 2006. Similar to bandwidth, Figure 3 shows that more developers have smaller interaction degree while fewer developers have large interaction degree, which can also be seen from Figure 1.

Another parameter to measure the network is the age of the central developers in central-100 network. If a developer joined in the central-100 network in 2006, he has age 1; if he joined the network in 2005, he has age 2; and so on. Figure 4 shows the distribution of different ages in three central-100 networks of 2006.

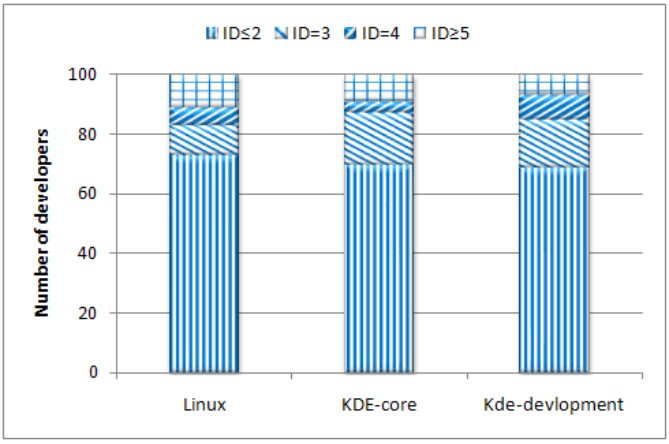


Fig. 3. The composition of the developers in central-100 network with different interaction degrees (ID) in 2006.

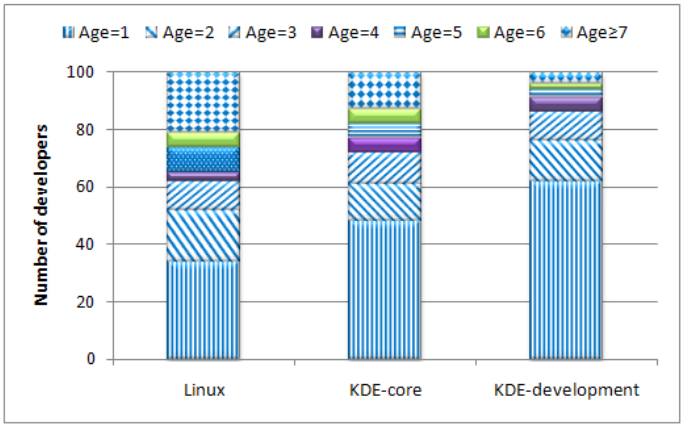


Fig. 4. The composition of different ages of developers in central-100 network of 2006.

The age of a member in the central-100 network represents the relative experience of the developer on this effort as well as the stability/evolution of the social network. It can be seen from the figure that different ages exist in the three networks, which means that the network developers are updating dynamically: every year, some developers leave the central-100 network and some members join the central-100 network.

4.3 Dynamics of the social network

The evolution of the dynamically changing open-source developer network can be measured using three metrics: the change in bandwidth, the change in interaction degree, and the change in membership of the central network.

We studied the evolution of the average channel bandwidth and the average interaction degree of the central-100 network, the results of which are shown in Figure 5 and Figure 6, respectively. The average interaction degree and the average bandwidth have other meanings in the social network: the former can represent the network structure and the latter can be used to represent the amount of activities in the network. As we can see in Figure 5, the activity of the Linux network have been increasing from 2000 to 2006, while the activity of KDE-core and KDE-development networks first decreased and then remained relative stable from 2000 to 2006. Figure 6 shows that the average interaction degree of three networks decreases from 2000 to 2006. Therefore, both the activity and the structure of the network evolve from year to year.

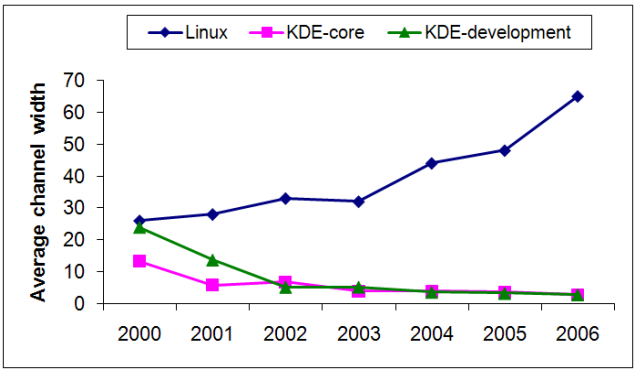


Fig. 5. The evolution of the average channel bandwidth of central-100 network.

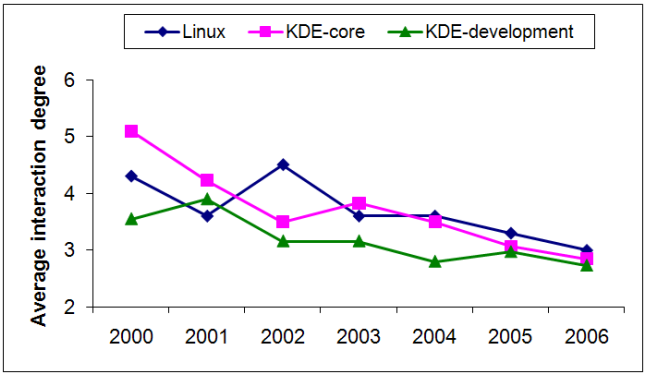


Fig. 6. The evolution of the average interaction degree of central-100 network.

Figure 7 illustrates the annual updating rate of the cluster-100 network. The annual updating rate is the percentage of developers in central-100 network that are changed compared to the previous year. For example, it shows that compared with 2000, about 50% of the Linux central-100 network developers are updated in 2001. Or in other words, it shows that in 2001, 50 developers in Linux central-100 network are new compared with the same network of 2000. Compared with the evolution of average bandwidth and average interaction degree shown in Figures 6 and 7, the annual updating rate of central-100 developers is relatively stable. Therefore, the annual updating rate of central-100 developers might be a stable parameter to represent the property of the network.

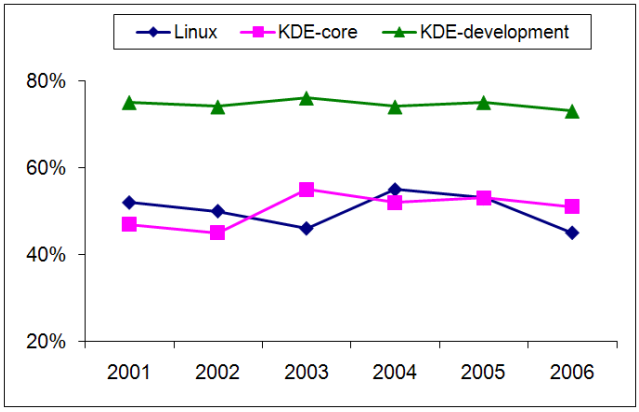


Fig.7. The annual updating rate of central-100 developers comparing to the previous year.

4.4 Modeling and simulation

As described before, channel bandwidth and interaction degrees changes from year to year for each project. They are not feasible to represent the property of the network. In contrast, updating rate of central-100 developers is relative stable for all networks. Therefore, it represents a unique feature of the network and is used to model and predict the behavior of the network.

In order to predict the evolution of open-source developer network, we modeled and simulated the entire developer networks. In the proposed simulation model, each developer is represented as an agent. The agent's activity is represented with a number in the range of [0, 10] and called activity score (AS). The larger the activity score an agent has, the agent is more active and plays a more important role in the network. The most active 100 agents (those have the largest 100 activity scores) form the central-100 network.

First, we assume that the activity score of an agent for a given year, i is most impacted by the activity score in year i-1. This assumption is represented as $AS_i \propto AS_{i+1}$.

A power decay function (base 2) is used to weight the most recent years' activity more than any of its predecessor years. This recency approach is adopted from our previous study on predicting open-source bugs [2].

$$f(x) = \frac{1}{2^x} \tag{1}$$

In Equation 1, x is the simulation step (year) and is equal to or greater than 1. In the proposed model, the activity score of each agent updates every year, and accordingly, the central-100 members are updated every year. The formula to calculate the activity score of a particular agent is shown below.

$$AS_i = \begin{cases} R & i = 1 \\ (1-\alpha)\sum_{x=1}^{i-1} f(x) AS_{i-x} + \alpha R & i > 1 \end{cases} \tag{2}$$

Where AS_i is the activity score at year i (i≥1); R is a random number that is uniformly distributed in range [0, 10]. Initially (i=1), the activity score of each agent is randomly generated. In the following steps, the activity score is determined by two parts, a recency factor based the agent's activity history and a random value αR, in which α is called the turbulence factor and is in the range of [0, 1]. In the proposed model, for each network, the value of turbulence factor α is same for all the agents. However, turbulence factor α might be different for different networks. The turbulence factor represents the stability of the network. The greater the turbulence factor the more unstable is the network. If α is 0, it means that the activity score of each agent is solely dependent on its activity history; if α is 1, it means that the activity of each agent is completely unpredictable and has no relation with his activity history.

The simulation setup is shown in Table 3. The number of agents is chosen as the total number of posters from 2000 to 2006 for each network. The number of cycles is the number of steps (years) the data is collected. In the observation, we have 7 years (2000 to 2006) data; therefore, the number of cycles is set to 7 for all three models. The number of pre-cycles is the number of cycles the model is run before the data is collected. Since the Linux project started in 1992, the number of pre-cycle is set as 8 (1992 to 1999); the KDE project started in 1996, the number of pre-cycle is set as 4 (1996 to 1999).

Table 3. The simulation model setup.

	Linux	KDE-core	KDE-development
Number of agents	7556	967	4731
Number of pre-cycles	8	4	4
Number of cycles	7	7	7
Turbulence factor (α)	0.08	0.23	0.37

Before the formal simulation, we run a series of pre-experiments with a range of turbulence factor α for Linux, KDE-core, and KDE-development. The turbulence factor is finally determined to use those that generate the best fit to the observed the annual updating rate, which is 0.50, 0.51, and 0.75 for Linux, KDE-core, and KDE-development respectively, as shown in Figure 7. Accordingly, the turbulence factor

for the formal simulation is 0.08, 0.23, and 0.37 for Linux, KDE-core, and KDE-development respectively.

It can be seen that KDE-core has a relative small turbulence factor than KDE-development, and Linux has a much smaller turbulence factor than both the KDEs. According to Equation 2, it is therefore reasonable to infer that the developers in the Linux project form a more stable group than those in KDE projects. Here, the stability is referring to the activity performed by each agent: if the agent's current activity is more dependent on his activity history and predicable, it is more stable; if his current activity is less dependent on his activity history and is unpredictable, it is unstable.

For the given model setup shown in Table 3, each network is formally simulated 100 times to study the changing of central members. The average annual updating rate of the central-100 active agents is shown in the boxplot of Figure 8, in which the mean of average annual updating rate of the 100 simulations is 0.50, 0.51, and 0.75, which match the observations shown in Figure 7. In the figure, the bold line within the box indicates the median. The box spans the central 50 percent of the data. The lines attached to the box denote the standard range. The circles indicate the data points that are out of the standard range.

To understand whether the proposed model can predict similar age distributions, the average value of the number of central-100 active agents with different ages is obtained in these 100 times of simulations. Figure 9 compares the age distribution of central-100 developers observed from the mailing list with the age distribution of the central-100 active agents obtained from the simulation in 2006. From the figure, it can be seen that KDE-development has a better match of age distribution than Linux and KDE-core.

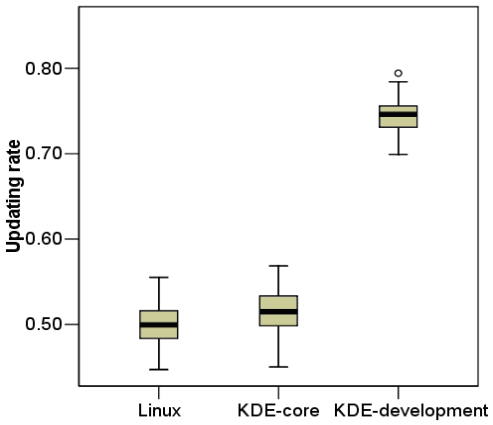
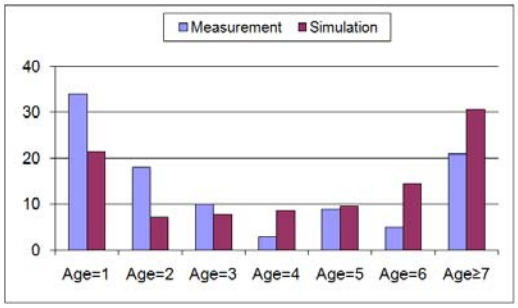


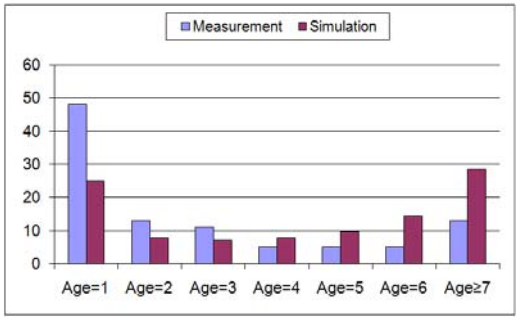
Fig. 8. The simulation results of the average updating rate of the central-100 active agents.

To quantitatively study the accuracy of the simulation, we calculate the average age of central-100 members from both the measurement and the simulation. The results are shown in Table 4. The difference is calculated using the formula $Difference = abs(simulation - measurement) / measurement$. We can see that the

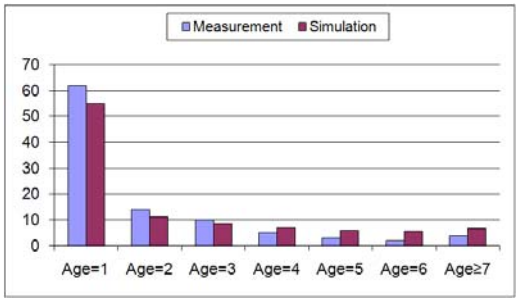
Linux model and the KDE-development model have higher accuracy in predicting the average age of central members.



(a)



(b)



(c)

Fig.9. The age distribution of the central-100 members in 2006 by the measurement and by the simulation: (a) Linux; (b) KDE-core; and (c) KDE-development.

Table 4. The average age of central-100 members of 2006.

	Measurement	Simulation	Difference
Linux	3.34	4.44	33%
KDE-core	2.73	4.27	56%
KDE-development	1.95	2.42	24%

In the proposed three agent-based simulation models, Linux has the smallest turbulence factor, which means that the activity of agent in the Linux is more stable than agents in KDE models. The stability of agents also represents the stability of the network. Accordingly, we can say that the Linux social network is more stable than KDE's. This is also reflected in both the measurement and the simulation of the average age of the central-100 members shown in Table 4.

5 Conclusions

In this paper, we presented the results of studying the evolution of open-source developer networks using data mining and simulation techniques. Case studies were performed on two open-source projects, Linux and KDE. Three developer networks were constructed and analyzed. The simulation of agent-based models successfully predicted the average age of the central-100 developers of three networks.

6 Acknowledgements

This work was based in part, upon research supported by the National Science Foundation (CNS-0619069, EPS-0701890 and OISE 0650939), Acxiom Corporation (# 281539) and NASA EPSCoR Arkansas Space Grant Consortium (# UALR 16804). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

References

1. Acuna, T.S. and Juristo, N.: Software Process Modeling. International Series in Software Engineering. Vol. 10. Springer: New York, NY (2005)
2. Joshi, H., Zhang, C., Ramaswamy, S., and Bayrak, C.: Local and Global Recency Weighting Approach to Bug Prediction. In: Proceedings of the 4th International Workshop on Mining Software Repositories, Minneapolis, IEEE Computer Society, Washington DC (2007)
3. Bird, C., Gourley, A., Devanbu, P., Gertz, M., and Swaminathan, A.: Mining Email Social Networks. In: Proceedings of the 3rd International Workshop on Mining Software Repositories, pp 137–143, ACM Press, New York (2006)
4. Bird, C., Gourley, A., Devanbu, P., Swaminathan, A., and Gertz, M.: Mining Email Social Networks in Postgres. In: Proceedings of the 3rd International Workshop on Mining Software Repositories, pp 185–186, ACM Press, New York (2006)
5. Cook, S., Harrison, R., and Wernick, P.: A Simulation Model of Self-Organising Evolvability in Software Systems. In: Proceedings of the 1st International Workshop on Software Evolvability, pp 17–22, IEEE Press (2005)

6. Yu, L. and Ramaswamy, S.: Verifying Design Modularity, Hierarchy, and Interaction Locality Using Data Clustering Techniques. In: Proceedings of the 45th ACM Southeast Conference, ACM Press (2007)
7. Yu, L. and Ramaswamy, S.: Mining CVS Repositories to Understand Open-Source Project Developer Roles. In: Proceedings of the 4th International Workshop on Mining Software Repositories, IEEE Computer Society, Washington DC (2007)
8. Yilmaz, L. and Phillips, J.: The Impact of Turbulence on the Effectiveness and Efficiency of Software Development Teams in Small Organizations. Software Process: Improvement and Practice 12(3): 247–265 (2007)
9. Hars, A. and Ou, S.: Working for free? Motivations for Participating in Open Source Projects. In: Proceedings of the 34th Annual Hawaii International Conference on System Sciences, Vol. 7. pp 7014–7023, Maui, Hawaii, January 2002.
10. Jain, A.K., Murty, M,N., and Flynn, P.J.: Data Clustering: A Review. ACM Computing Surveys 31(3): 264–323 (1999)
11. Wagstrom, P.A., Herbsleb, J.D., and Carley, K.: A Social Network Approach to Free/Open Source Software Simulation. In: Proceedings First International Conference on Open Source Systems, pp 16–23 (2005)
12. Johnson, S.C.: Hierarchical Clustering Schemes, Psychometrika 2: 241–254 (1967)
13. KDE Mailing List, http://www.kde.org/mailinglists/
14. Linux Kernel Mailing List, http://www.uwsg.iu.edu/hypermail/linux/kernel/index.html.
15. Smith, N., Capiluppi, A., and Fernández-Ramil, J.: Agent-Based Simulation of Open Source Evolution. Software Process: Improvement and Practice 11(4): 423–434 (2006)
16. Macal, C., North, M.: Tutorial on Agent-Based Modeling and Simulation. In: Proceedings of the 2005 Winter Simulation Conference, pp 2–15 (2005)
17. Madey, G.R., Freeh, V.W., and Tynan, R.O.: Agent-Based Modeling of Open Source Using Swarm. In: Proceedings of the 8th Americas Conference on Information Systems, pp 1472–1475 (2002)
18. Madey, G. R., Freeh, V.W., and Tynan, R.O.: Modeling the F/OSS Community: A Quantative Investigation. Free/Open Source Software Development, Koch S (ed.), pp. 203–221. Idea Group Publishing: Hershey, PA (2004)
19. Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K.: Understanding Free/Open Source Software Development Processes. Software Process: Improvement and Practice 11(2): 95–105 (2006)
20. Open Source Initiative, http://www.opensource.org/
21. Reis, C. and Fortes, R.: An Overview of the Software Engineering Process and Tools in the Mozilla Project. In: Proceedings of Workshop on Open Source Software Development, pp 155–175 (2002)
22. Robles, G., Merelo, J.J., and Gonzalez-Barahona, J.M.: Self-Organized Development in Libre Software Projects: A Model Based on the Stigmergy Concept. In: Proceedings of the 6th International Workshop on Software Process Simulation and Modeling, ACM Press, New York (2005)