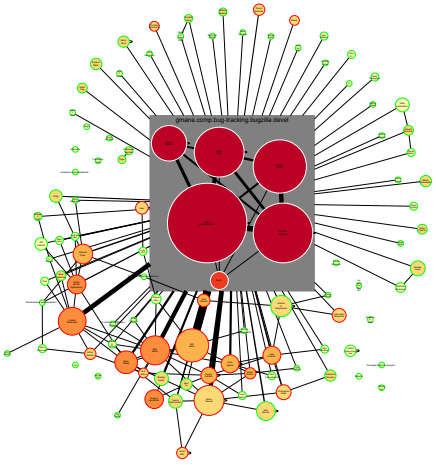
Freie Universität Berlin



The Onion has Cancer

Some Social Network Analysis Visualizations of Open Source Project Communication

Christopher Oezbek Lutz Prechelt Florian Thiel

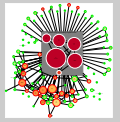
Freie Universität Berlin

ICSE Workshop on Emerging Trends in FLOSS Research and Development, 2010-05-08

2010-05-08

The Onion has Cancer

Freie Universität Berlin



The Onion has Cancer
Some Social Network Analysis Visualizations
of Open Source Project Communication

Christopher Oezbek Lutz Prechelt Florian Thiel
Freie Universität Berlin

ICSE Workshop on Emerging Trends in FLOSS Research and Development, 2010-05-08

I would like to welcome everybody back from Lunch to this talk, titled “The Onion has Cancer. Some Social Network Analysis Visualizations of Open Source Project Communication”.

My name is Christopher Oezbek and this is work done with Florian Thiel and Lutz Prechelt, who is in

the audience over there, at Freie Universität Berlin.

Background

- The software engineering group at Freie Universität Berlin investigates software development processes and means of improving them
- In the context of Open Source:
  - How can software engineering innovations be introduced into Open Source projects?
- Methods:
  - Action Research
  - Grounded Theory Methodology on mailing-list data
  - Using theories and models from social and organizational sciences:
    - Actor-Network Theory, Garbage Can Model, Path Dependence, Social Network Analysis

Background

- The software engineering group at Freie Universität Berlin investigates software development processes and means of improving them
- In the context of Open Source:
  - How can software engineering innovations be introduced into Open Source projects?
- Methods:
  - Action Research
  - Grounded Theory Methodology on mailing-list data
  - Using theories and models from social and organizational sciences:
    - Actor-Network Theory, Garbage Can Model, Path Dependence, Social Network Analysis

First, I want to give you some background about this work. This work is part of the software engineering group's effort at Freie Universität to analyze software processes and determine and implement improvements.
For instance, we have in recent years looked at pair programming as an important practice from XP and helped a company to achieve real-time collaboration in Eclipse between a European development center and one in india using our tool Saros.
In the domain of Open Source software development—which my PhD thesis work is about—we have concentrated on understanding how software engineering innovations can be introduced into Open Source projects.
This was driven by the desire to explain how volunteer organizations can achieve process change.
As methods to achieve this we first used Action research—directly interacting with Open Source projects in five case studies—then we used Grounded Theory methodology on mailing-list data—as a passive and qualitative method to extract insights from past innovation introductions—and last we used several theories and models from the organizational sciences to raise theoretical sensitivity.

In this last category was also Social Network Analysis, which provides the starting point for this paper.

Background

- The software engineering group at Freie Universität Berlin investigates software development processes and means of improving them
- In the context of Open Source:
  - How can software engineering innovations be introduced into Open Source projects?
- Methods:
  - Action Research
  - Grounded Theory Methodology on mailing-list data
  - Using theories and models from social and organizational sciences:
    - Actor-Network Theory, Garbage Can Model, Path Dependence, **Social Network Analysis**

SNA and Open Source

- Social networks created from e-mail communication data

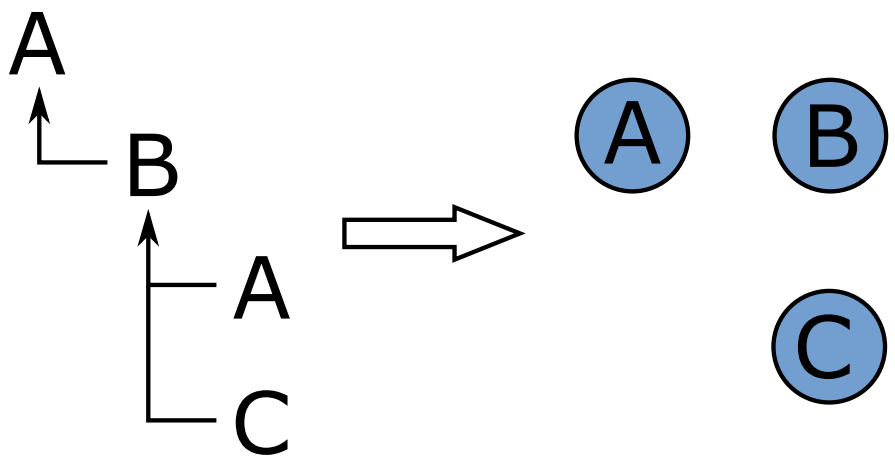
There are many possible ways to create a Social Network can be created.

- For instance Greg Madey et al. used SourceForge.data to created edges between two developers if those developers worked in the same project.
- James Howison and Kevin Crowston created edges if people made comments on the same entry in the bug-tracker.
- Fernandez et al. used commits to the same module in the source code repository of a project.

In this study we created the social network from the public mailing-list communication data and

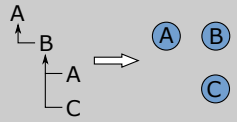
specifically the reply-to relationship to create edges between mailing-list participants

- Social networks created from e-mail communication data



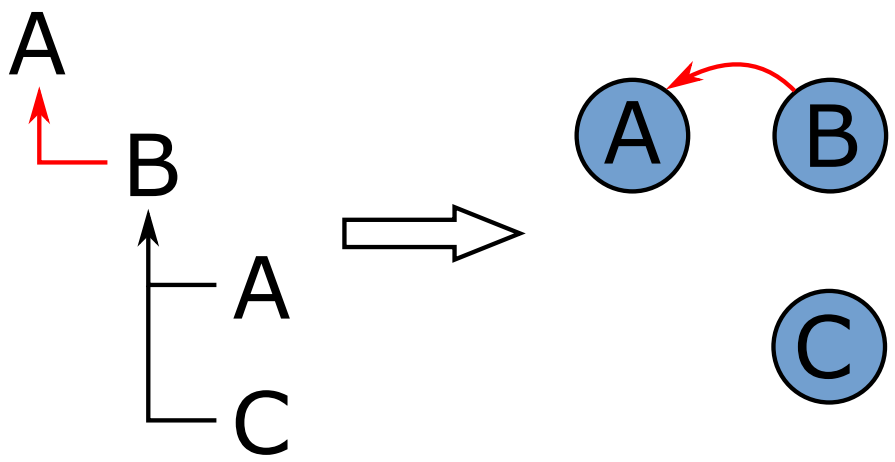
SNA and Open Source

- Social networks created from e-mail communication data

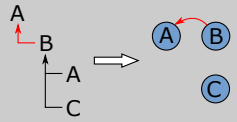


Given a mailing-list thread as shown below, where B replied to an email by A and A and C replied to this email by B we can construct the following example network by turning each author A, B, and C into a node in the social network

- Social networks created from e-mail communication data



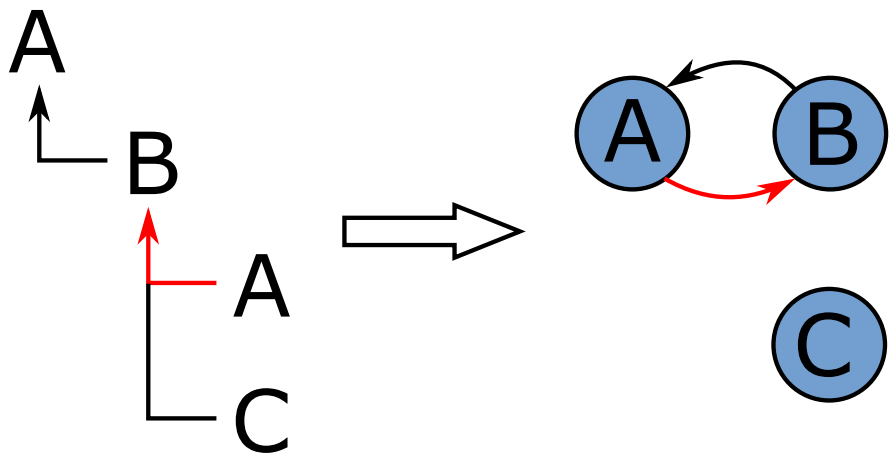
- Social networks created from e-mail communication data



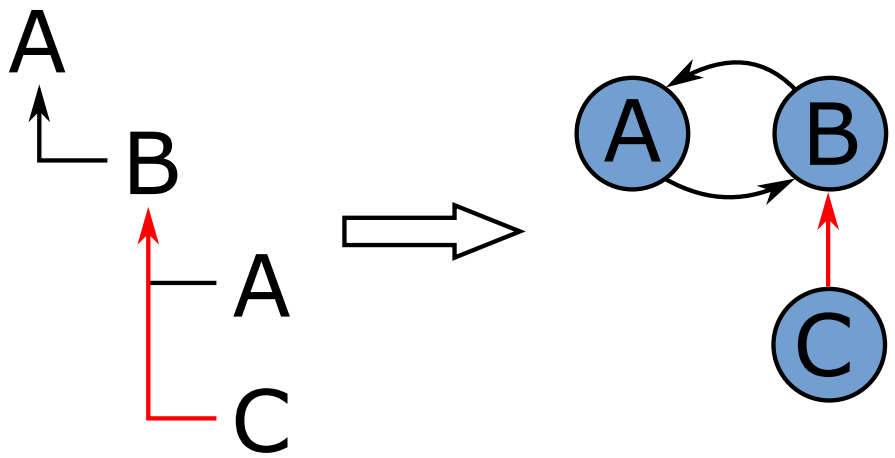
The reply by B to the e-mail by A is translated into an edge from B to A.

The others respectively.

- Social networks created from e-mail communication data



- Social networks created from e-mail communication data



Data set

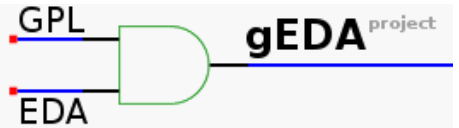
- 11 medium-sized Open source projects
- Observation period was the year 2007
- 14 000 e-mails

ArgoUML

Bugzilla

RequestTracker

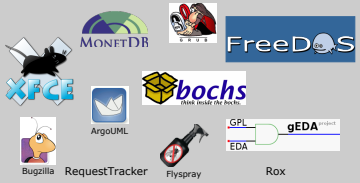
Flyspray



Rox

Data set

- 11 medium-sized Open source projects
- Observation period was the year 2007
- 14 000 e-mails

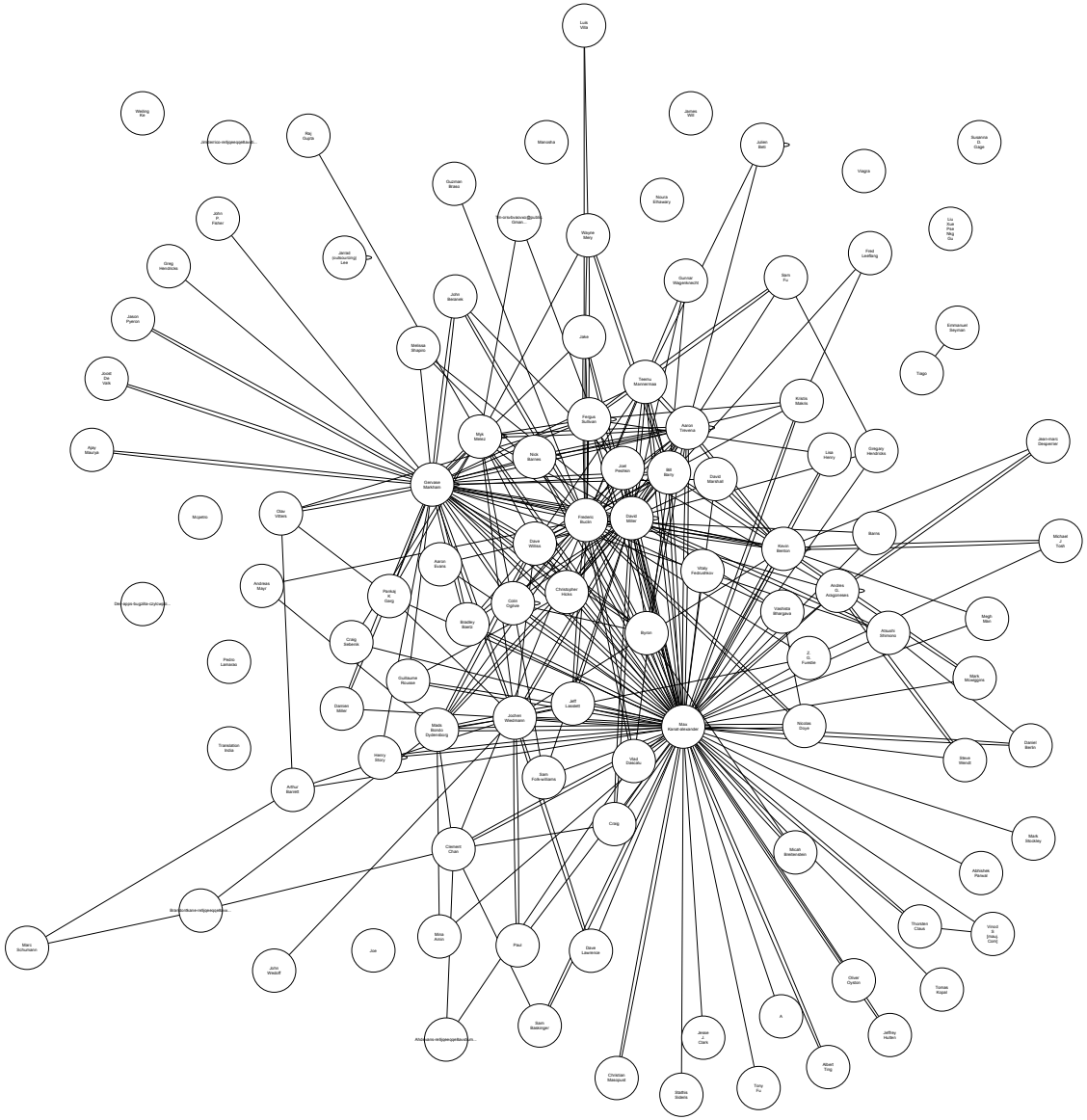


This basic idea of translating all mailing-list communication into a social network was done for 11 medium-sized projects.
Our sample included three work flow applications (Bugzilla, Flyspray, RequestTracker), two desktop environments (Rox, Xfce), two design tools (ArgoUML, a UML CASE tool, gEDA,a set of electronic design automation tools), one boot loader (Grub), one hardware emulator (Bochs), one operating system (FreeDOS), and one database management system (MonetDB)
So there is a healthy mix of applications for end-users, professional users such as IT staff, and software developers both on the desktop and on the server, and both rather close to the operating system and far away from it.

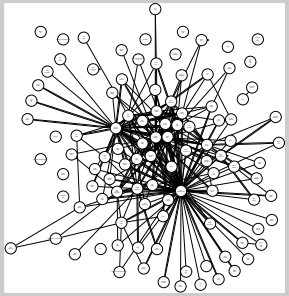
Lists had between 500 and 3000 e-mails in the observation period and were cleaned for unifying

authors with several e-mail addresses and to remove spam.

Plain graph

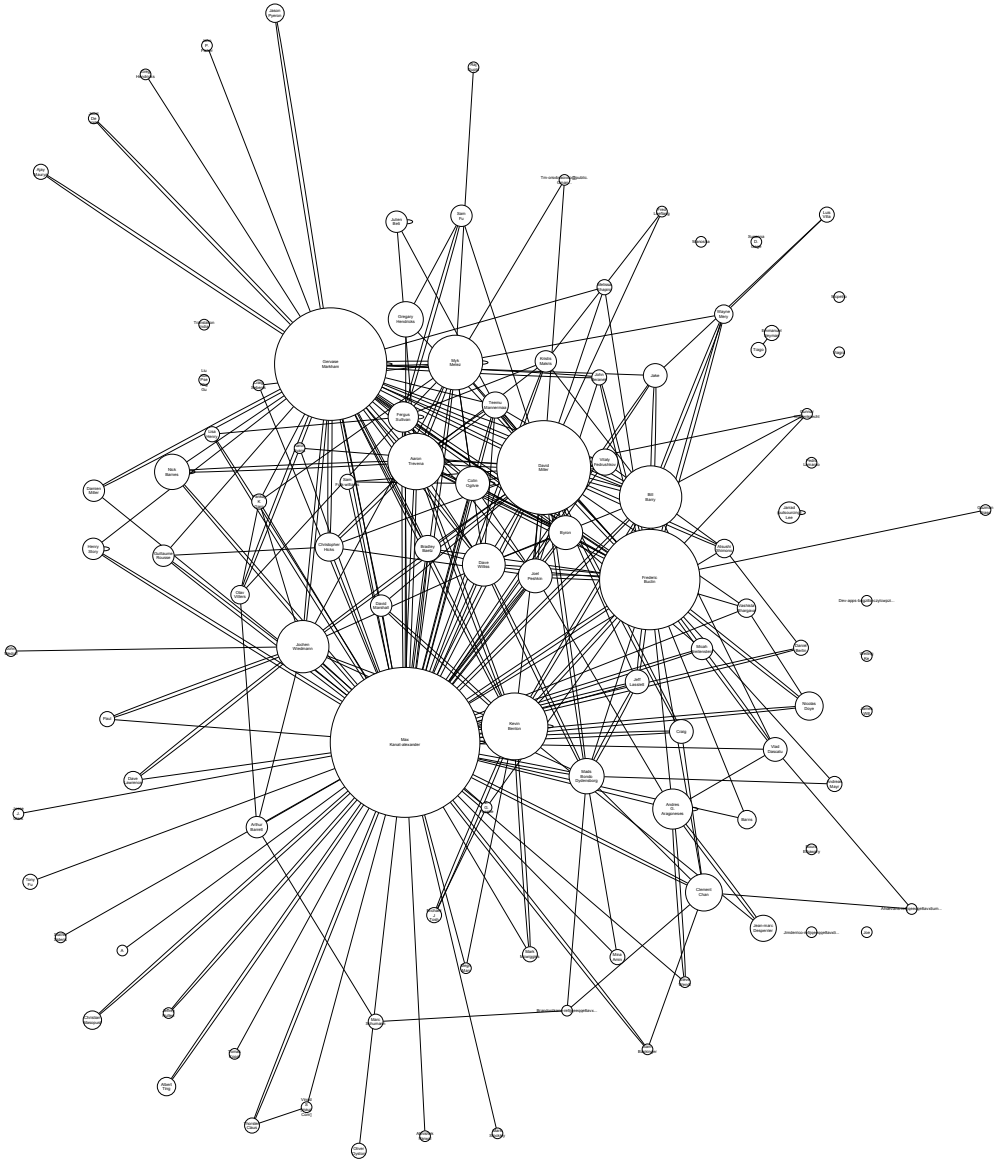


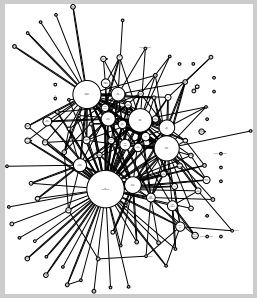
Plain graph



Using just the simple visualization results in the following, a rather uninformative graph–in this case for the project Bugzilla–which we hence enhanced in 5 steps

Scaled Nodes





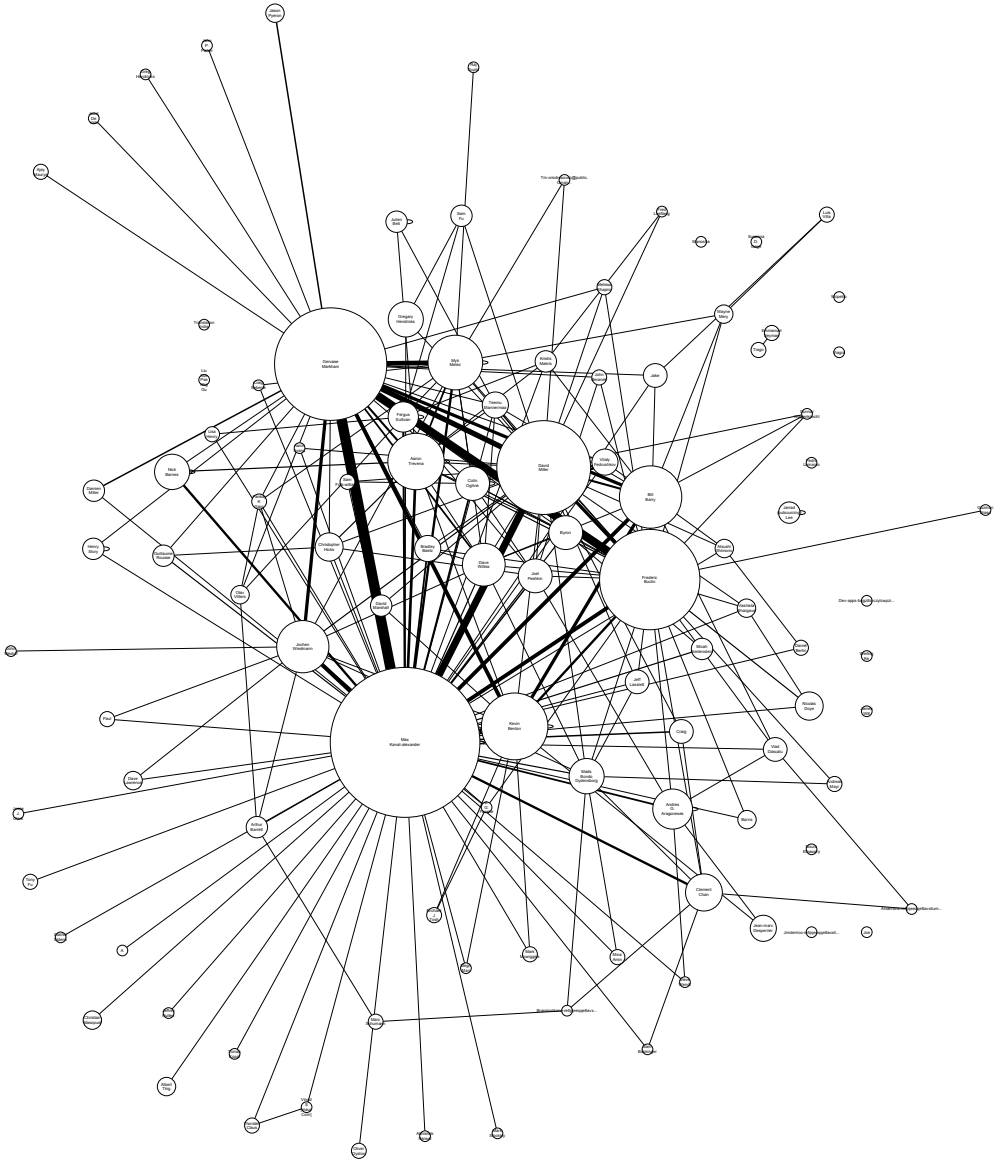
First, we scaled the size of the nodes so that their area was proportional to the number of e-mails written by the respective mailing-list participant
The Long-tail distribution in the communication is strongly noticeable when doing this:

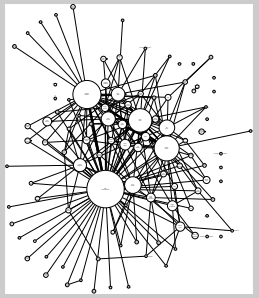
- Few mailing-list participants write most of the messages
- Large number of participants only write only one or two messages

In this example of the project Bugzilla there were 938 e-mails written by 106 people. 5 people wrote

50% of these e-mails, 22 80%

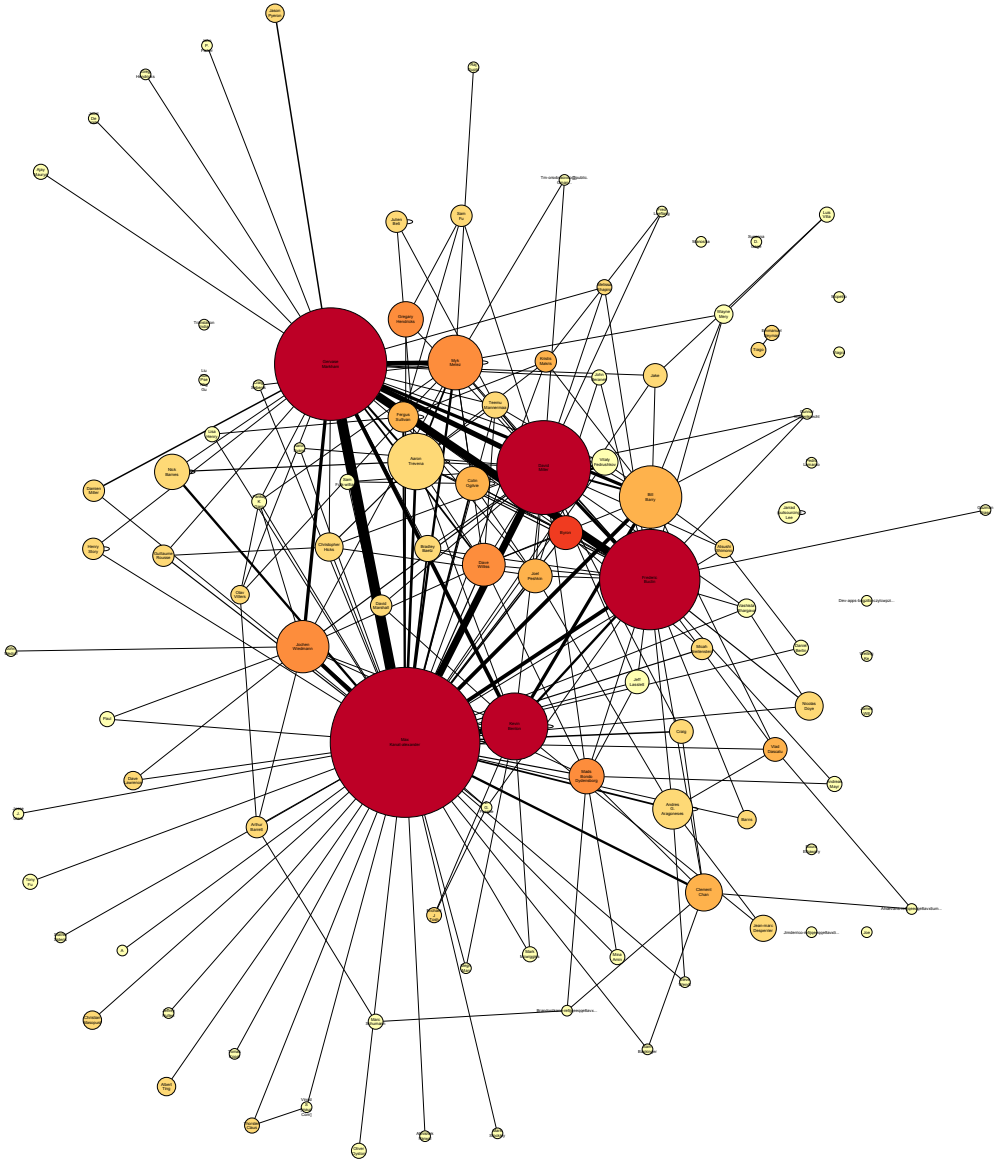
Make unidirectional and scale edges

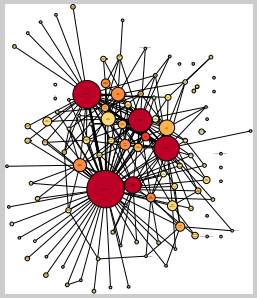




Secondly, we turned the graph unidirectional and scaled the width of the edges so that they were proportional to the number of e-mails exchanged between the respective mailing-list participants.

Color nodes

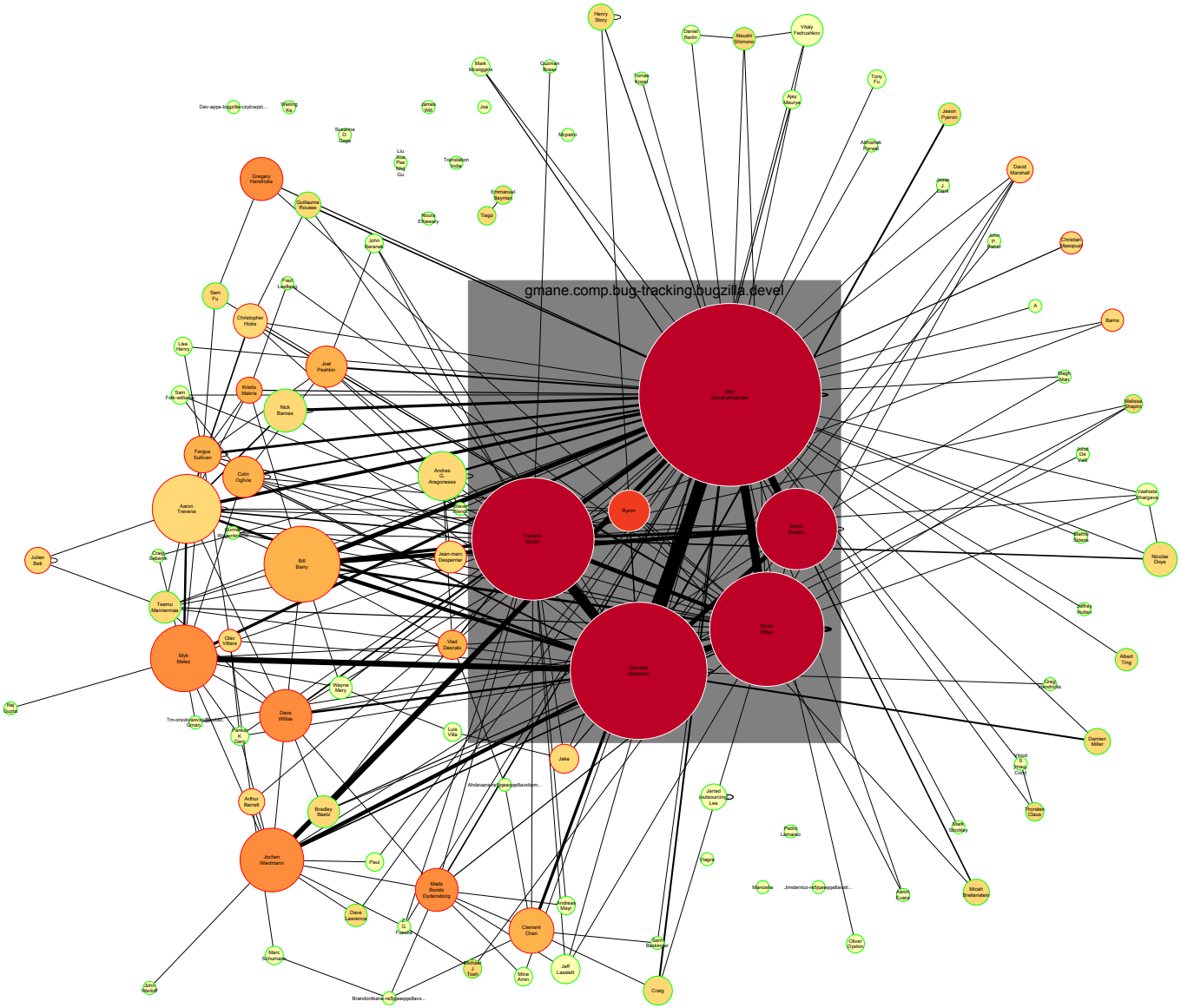




Thirdly, we used color to denote how many months of the year a person had been active on the list to assess the continuity of their engagement.

In this case we distinguish 2,4,6,8 and 10 months of activity

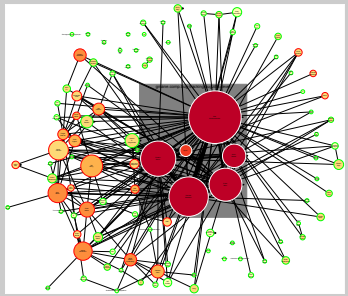
Layout project core separately



The Onion has Cancer

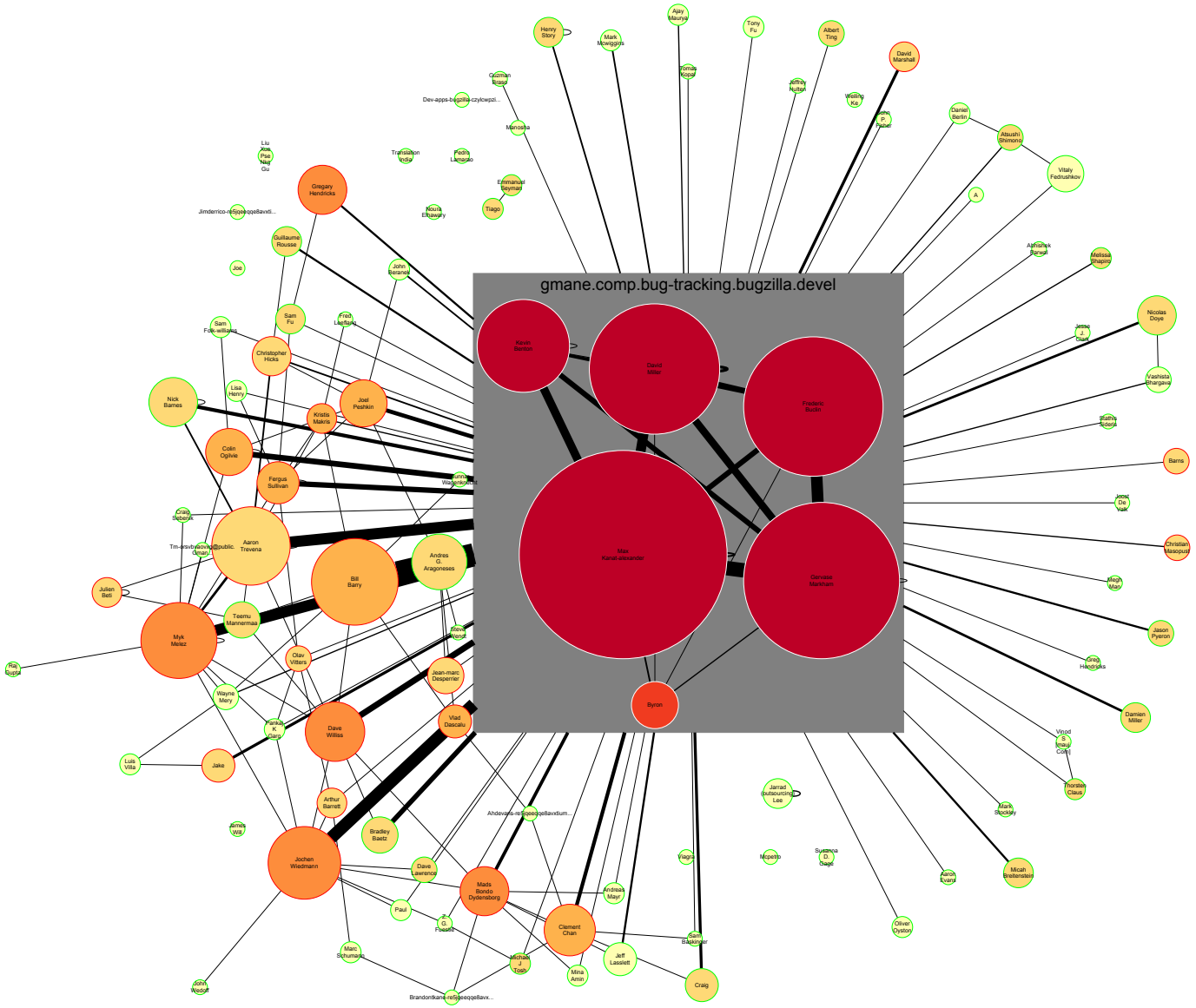
Main

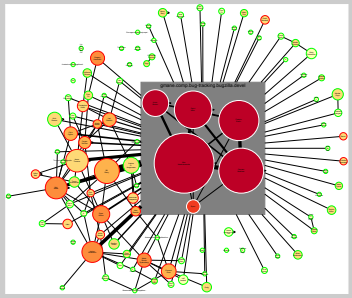
Layout project core separately



Fourthly, we identified the core-members of each project using a simple heuristic of 8 or month active per year and laid them out separately in the network, which is shown in the grey box.

Collapse communication with the core

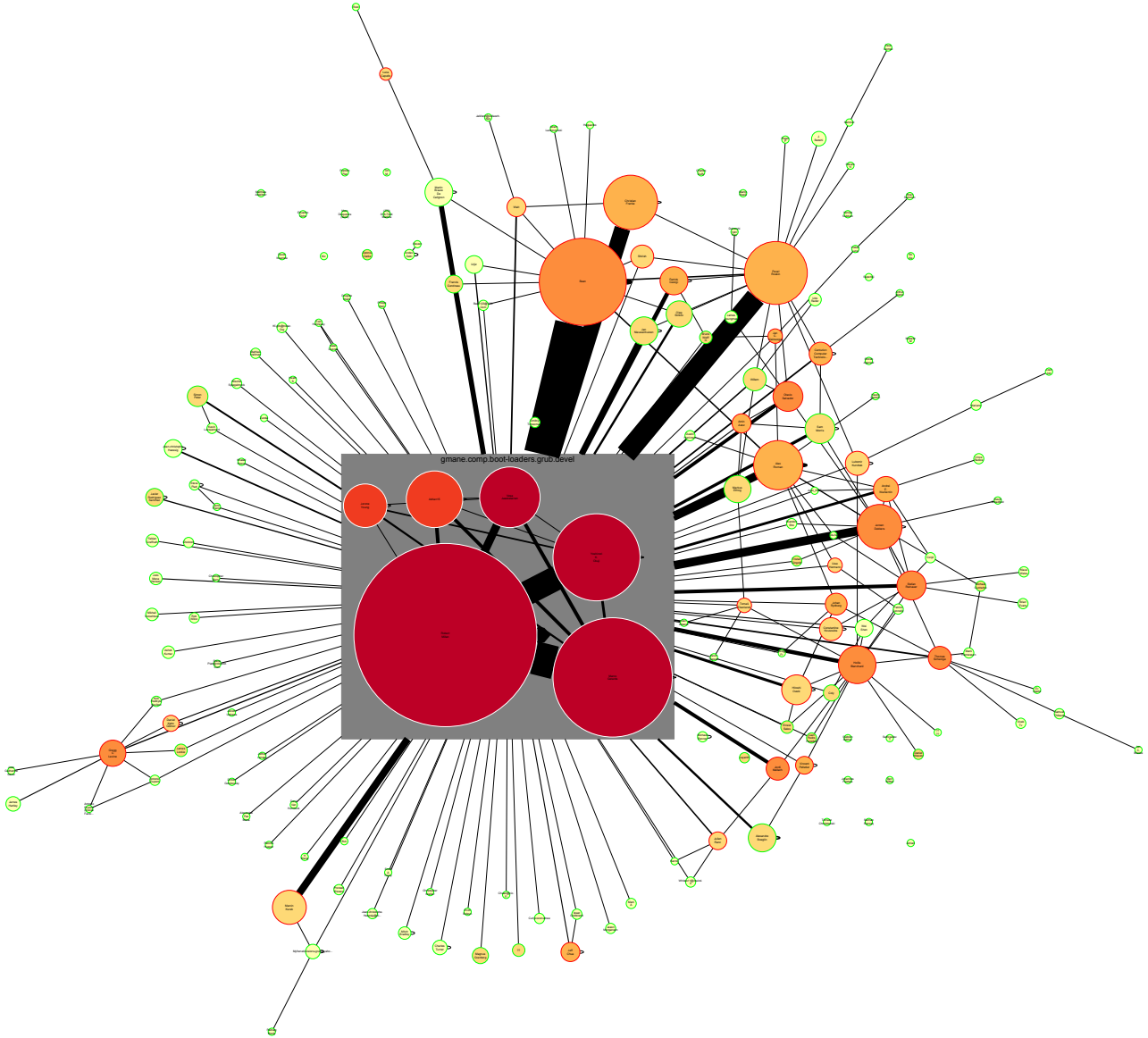


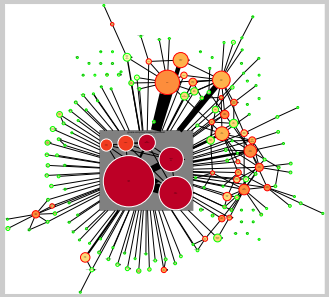


Last, we decided to collapse all communication from people outside the core to people inside the core to be focused on the project.
The result is a picture which is highly similar for all projects:

- In the center, shown in the grey box, is the project core, containing the predefined project-core
- around this project core, two sub-groups can be identified:
  - like a halo, there is the periphery of the project, which communicates almost exclusively with the core
  - then there is a set of co-developers which are attached strongly to the core—visible from the thick edges connecting them to the core— but also lightly integrated with each other.

GRUB

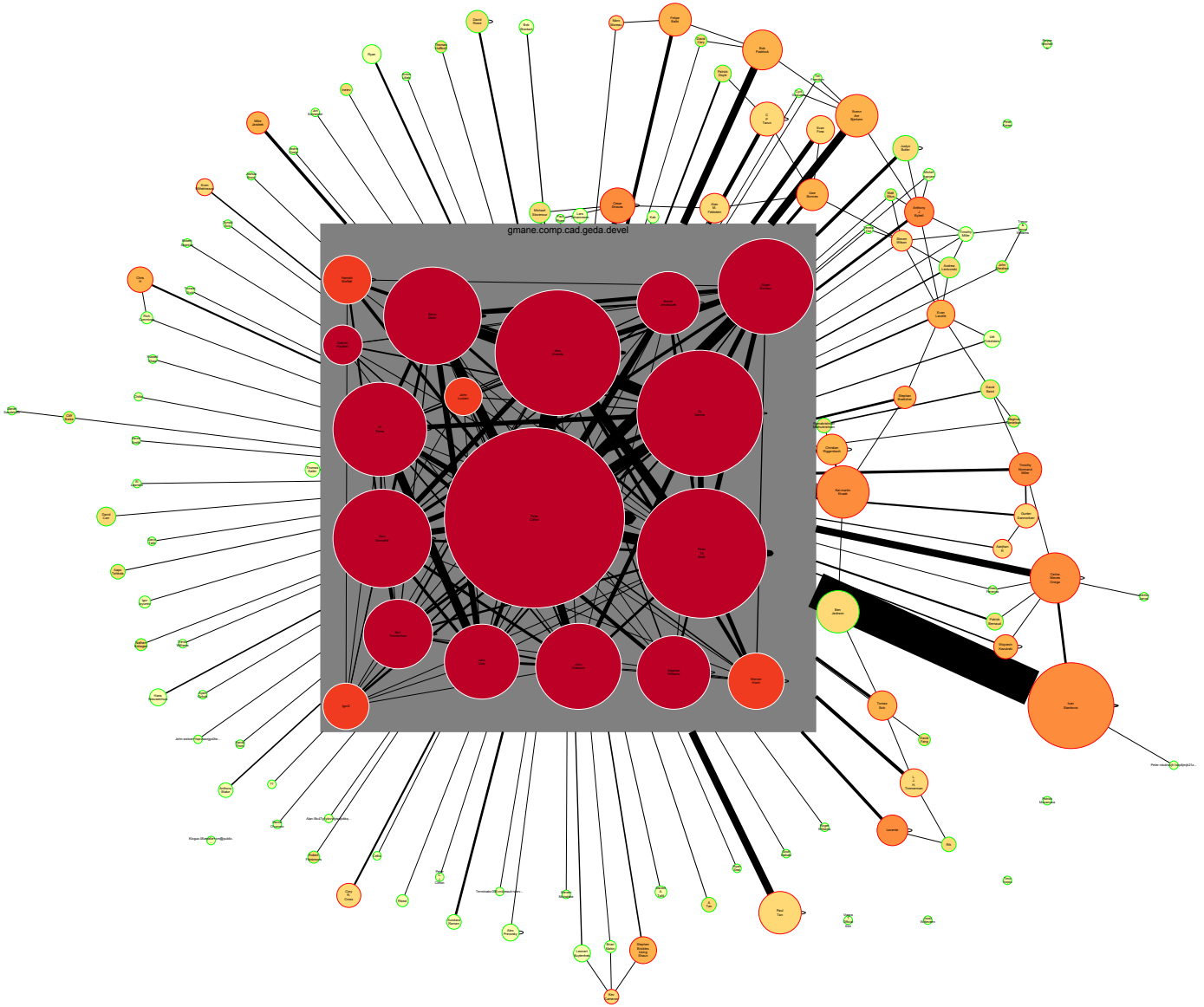




and this visualization looks very similar in other projects. Here for instance in the boot-loader GRUB.

a central core, a Periphery like stars or rays arranged around the core, and a crescent-shape periphery which seems to grow on the side of the core, inspiring the title.

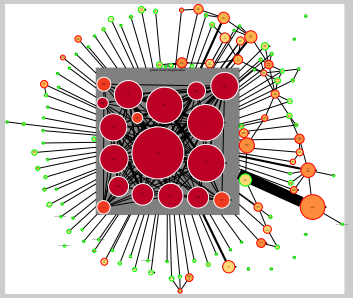
gEDA



The Onion has Cancer

Main

gEDA

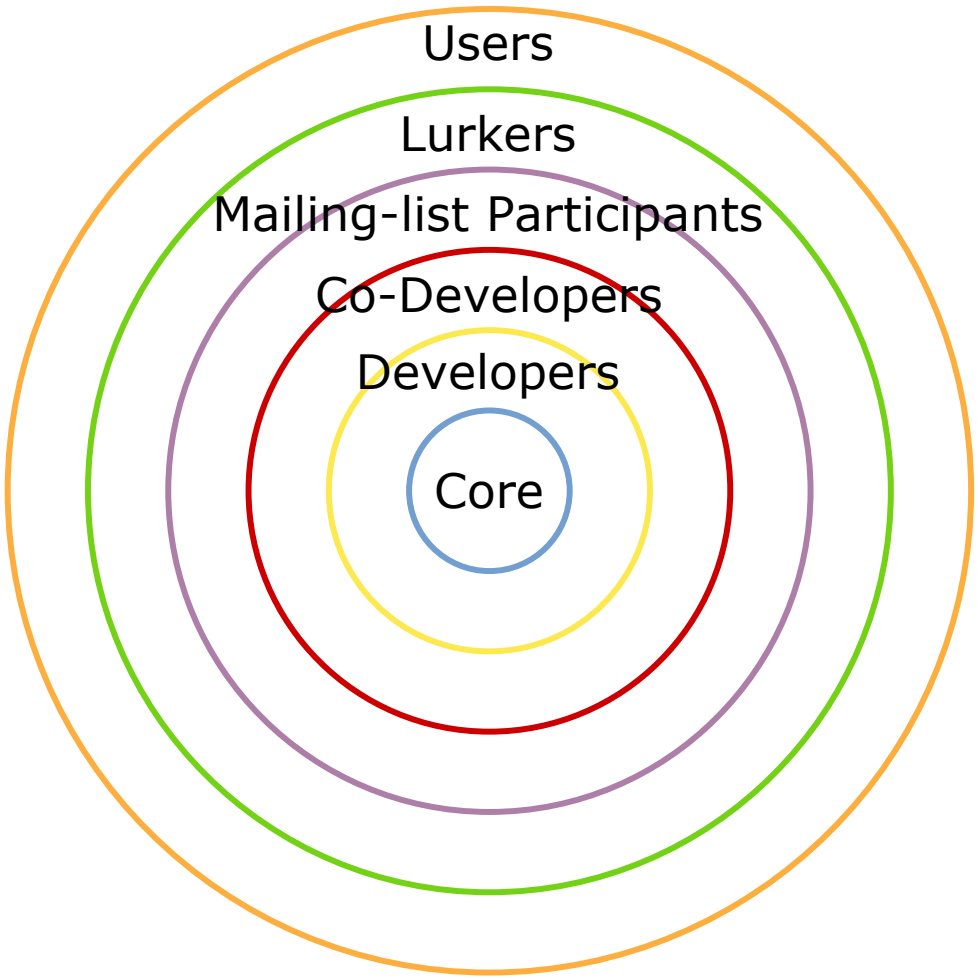


two questions came to mind when we saw this pattern over and over again:
First, how does this relate to the onion model

and Second, isn’t it surprising how strong everybody is oriented towards the core?

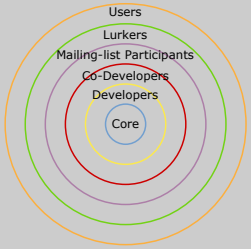
Relating to the onion model

- Onion model = metaphor for role advancement, activity and influence



Relating to the onion model

- Onion model = metaphor for role advancement, activity and influence



The onion model is a metaphor for role advancement, activity and influence. Transferring this model onto mailing-list communication behavior one might have assumed that new mailing-list participants attach to the periphery, increasingly build up connections to other peripheral developers thereby moving closer to the core, where they then can attach to co-developers and finally core developers. In a hierarchical organization that is what we would pretty much assume. Yet as the social networks show, this is not the case Attachment is strongly to the core, possibly even stronger than a uniform communication structure would indicate.

So let’s go on to check that...

Core-oriented communication

Freie Universität Berlin

- Used χ^2-test at $P < 0.01$ assuming uniform communication

We used a χ^2 test to see whether the communication of the periphery and the co-developers were really oriented statistically significantly more to the core or rather uniform. in other words, if the core was writing twice as many messages and thus would receive twice as many replies as the other groups, that would not be a very surprising result.

Core-oriented communication

Freie Universität Berlin

- Used χ^2-test at $P < 0.01$ assuming uniform communication
- In 8 of 11 cases $R0$ could be rejected
- 31% to 69% of expected communication between periphery and co-developers

Core-oriented communication

- Used χ^2-test at $P < 0.01$ assuming uniform communication
- In 8 of 11 cases $R0$ could be rejected
- 31% to 69% of expected communication between periphery and co-developers

1. We found that in all 11 cases the communication between periphery and co-developers to each other was less than expected. In 8 of the 11 cases, the t-test marked the differences as significant...
2. ... and communication was between 31% and 69% of what we would have expected from a uniform distribution

Core-oriented communication

Freie Universität Berlin

- Used χ^2-test at $P < 0.01$ assuming uniform communication
- In 8 of 11 cases $R0$ could be rejected
- 31% to 69% of expected communication between periphery and co-developers
- Other three projects: Core too small, or expectation almost met

- Used χ^2-test at $P < 0.01$ assuming uniform communication
- In 8 of 11 cases $R0$ could be rejected
- 31% to 69% of expected communication between periphery and co-developers
- Other three projects: Core too small, or expectation almost met

For the projects which did not show significant differences, we found that 2 of them had too small a core (Bochs and RT) based on our definition, which increases communication among the other groups of course, and one had almost meet our expectation (Xfce) at 95% of expected communication.

Conclusion

Freie Universität Berlin

- Implications:
  - Join scripts seem to rely on a good contact to the core
  - Little developer-to-developer assistance
- Open Question:
  - How to better identify core, co-developers and periphery?
  - Differences to commercially-dominated OSS projects?
  - Why does the core-orientation arise?

Conclusion

- Implications:
  - Join scripts seem to rely on a good contact to the core
  - Little developer-to-developer assistance
- Open Question:
  - How to better identify core, co-developers and periphery?
  - Differences to commercially-dominated OSS projects?
  - Why does the core-orientation arise?

1. To conclude, I want to give a couple of implications and open questions arising from this.
2. First, if we think about how people join an Open Source projects, we should focus in particular on the relationship between new developer and the core.
3. Second, as a person at the periphery, one should not expect much help from fellow peripheral people.
4. Obviously we used a very rough metric for splitting a project into the three groups, and future work should try to improve these metrics for instance by taking qualitative data or information from repositories into account.
5. Our sample unfortunately in particular excluded two commercially-dominated OSS-projects, which we had studied regarding innovation introduction, and it remains an open question, how our result would change. Minghui Zhou gave a good talk about several adjustments to the hypothesis from Mockus, Fielding, Herbsleb’s paper on the Apache and Mozilla project, which showed that this can have quite an impact.
6. Last, we need to qualitatively understand how the core-orientation arises, what communication is being made and how much importance to assign to this effect.

Freie Universität Berlin

Thank you

christopher.oezbek@fu-berlin.de

Thank you

christopher.oezbek@fu-berlin.de

1. Thank you all for listening!
   I’d be happy to take questions