# Network Structure of Issues in GitHub Project rust-lang/rust

Dorota Kapturkiewicz
MSc Web Science and Big Data Analytics
University College London

Academic year 2015-2016, Term 1

## Abstract

Managing a big and popular open source project is an extremely challenging task. It is very important to make clear what needs to be done and how problems are related. At this moment GitHub makes it possible by creating Issues with milestones, labels and comments.

In this paper I investigated a graph of issues in the project rust-lang/rust[1], since it's one of the biggest and most popular repository on GitHub. Nodes contained 30000 issues (almost all issues in project), and there was an edge between two issues if and only if one of them referenced to another. Applying PageRank algorithm on directed version of this graph successfully identified so called "metabugs". For undirected graph it came out that it shows incredibly strong community structure with modularity over 0.88. But surprisingly, it's not correlated to any of two methods of issue grouping (milestones and labels) that are currently used. In my opinion it is extremely interesting result, as it may suggest that there is something more about issue communities than currently known. Maybe, after further studies, it can be a background for developing new method of automatic issue management.

[1] https://GitHub.com/rust-lang/rust

# Contents

# 1 Introduction

Open source projects are extremely important for developers. I'm pretty sure almost all programmers use at least one of those projects on daily basis. And if project is big, has a lot of users and contributors from allover the world and from different backgrounds, it is incredibly important to be able to keep track of project's status and state clearly problems that need to be solved. GitHub, the biggest platform for collaborative programming, enables it by Issues. If there is a problem with the project (or a new idea as well), one can create an Issue. Then, one can assign a developer to solve it, comment on it, refer to it from other Issue and assign valid labels and milestones to it. But there is one problem: all of actions from above need to be done manually by an owner or privileged user. Because of it, there are plenty of issues without label or milestone. So it is really a lot of work to keep issue tracker in order, especially for big and popular repositories.

The most significant way of linking two issues is mentioning one of them in another. But it is also the most time consuming one, as user needs to be aware of existence of the related issue and include it somewhere in their comments. That's why I decided that it would be great to find some common features of manually linked issues, so that they could be linked automatically in the future. To achieve it, I decided to look at Issues as a graph and analyse it with complex networks tools.

I created two graphs of all Issues in the project: directed and undirected. In the directed graph there were edges from referencing (mentioning) issues to issues that were referenced. And in the undirected one issues were connected if and only if one of them was mentioned in another. Then I asked the following questions:

- **Q1:** ***What are the most important issues identified in directed graph by PageRank algorithm?***
  Directed graph of issues is quite similar to web in sense of determining node importance by who links to it. That's why I decided to run PageRank algorithm and analyse best results. It occurred that most of them are so called *metabugs* (Issues created to group others).

- **Q2:** ***How degree distribution and mixing pattern look like in directed graph?***
  This question was also strongly related to metabugs. I supposed that because of them both in- and out-degree distributions should resemble power-law like shape. Additionally I supposed that network should follow disassortative mix-

ing pattern. And these assumptions were correct.

- **Q3:** ***Is there significant community structure in undirected graph?*** The most important question of my research. I hoped that there is community structure and communities have something in common that can be used in future for automatic issues grouping. As the graph was big, I decided to use Louvain algorithm described in *Methodology* section. And in this case I was right: community structure is indeed extremely strong.

- **Q4:** ***Is community structure related to milestones, labels or time of Issue creation?*** Milestones, labels and time of Issue creation seemed to me natural candidates to be a common property within a community. But surprisingly they are not.

I strongly believe that my research questions made sense. I'm very happy that analysis of directed graph identified metabugs. It was something I wanted to achieve and it was exactly what I expected.

Results for communities where although quite surprising. With so strong community structure (modularity was larger than to 0.88!) these communities couldn't be random. I spent a lot of time trying to figure out some common properties within communities, but I've found nothing. So what could possible went wrong? Maybe there are more indirect properties that I didn't investigate, for example issues within one community are related to the same code. Additionally I suppose that there may be a problem in how people manage Issues. I think that developers are too sloppy with assigning properties and they like linking a lot of issues at once (when it's starting to look so chaotic that something really needs to be done) rather than doing it regularly. So the data set may be not really trustworthy.

# 2 Background

## 2.1 GitHub

GitHub[2] is one of the most popular social coding websites. It's a web-based platform with Git[3] repositories and social network of developers. Right now there are over 12.3 millions of users and 31.4 millions of repositories on GitHub[1]. It strongly supports open-source, as all open-source projects are hosted for free. It also offers

[2]www.github.com

[3]www.git-scm.com. Git is a source code management and distributed revision control tool.

paid closed repositories for companies and private users.

Developers can create their own projects or fork and contribute to repositories belonging to other users. In terms of contributing one can commit some code, send Pull Requests, manage Issues and so on. Users can also "observe" projects and users and get notifications while some kind of activity occurs. On user's page there are many interesting information like popular repositories, repositories contributed to, graph of amount of daily contributions (pull requests, issues opened, and commits) as well as location, organizations, followers, starred projects and many other statistics. Project pages contain commits, list of contributors, pull requests, issues, code and several statistics. All of these features make GitHub an extremely interesting data source for many unique complex networks.

## 2.2 Issue structure

Issues provide a method for tracking bugs and planning new features in GitHub projects. When something needs to be done, developer can create an Issue in a project and explain there what a problem is. When it's no longer actual (solved or outdated) Issue can be closed. All users can comment on Issues and reference to it from commits and other Issues. It's worth mentioning that Pull Requests (requests from contributors without write permissions to accept their code in a project) are also treated as Issues.

Additionally there are tools to make Issues more structural. First of all, one can assign someone to be responsible for an Issue. There are also labels and milestones. In a big project there can be a lot of different type of Issues, for example bugs in different parts, new feature proposal, problems with documentation and so on. Project collaborators can create as many labels as they need to to distinguish these different classes. Very often there are also labels representing difficulty and priority of an Issue. There are even so called *metabugs* - issues created just to group others.

Milestones group issues basing on project stage. For example, if something needs to be fixed for release of particular project version, it can be assign to a milestone representing this release. It helps user to decide what should they focus on and to determinate priorities.

# 3 Literature survey

To get familiar with existing research in GitHub and developer social networks in general, I've read following interesting articles:

1. Understanding a Developer Social Network and its Evolution [2]
2. Coding Together at Scale: GitHub as a Collaborative Social Network [3]
3. Network Structure of Social Coding in GitHub [4]
4. Combined Methods, Thick Descriptions: Languages of Collaboration on GitHub [5]

All of them have been written between 2011 and 2014 and seemed to be most important works analysing GitHub and other coding platforms from network point of view. A lot of graphs were created and authors of these articles asked and answered a lot of interesting questions. But nobody focused on issues in one project in a way I did in this coursework.

In sections below I present short summaries and discussions about these papers.

## 3.1 Understanding a Developer Social Network and its Evolution

Authors of this paper focused on comparing General Social Networks (GSN) such as Facebook, Twitter, Cyworld and the Amazon recommendation network with Developer Social Networks (DSN). They also asked questions how developer social network evolves over time and about its community structure.

To get a developer social network they decided to use Bugzilla, the bug tracking system created and used by Mozilla. Bug reports between years 2000 and 2009 were extracted and a social network was created as follows: nodes represented contributors and they were connected if and only if they commented on the same issue more than 2 times.[4] Nodes with no edge were removed.

Results of general and developer social networks comparison were very interesting. First of all, unlike GSN, DSN didn't have power low degree distribution, while still having most of nodes with small degree and not so many with high degree.

[4]It was an important optimization to get a network of really involved developers and remove people who just happen to report a few bugs.

Secondly, both general and developer social network have small world property, but degree of separation in DSN is smaller than in GSN. Paper authors speculate that it is because there is narrower choice of topics in DSN, so it's more likely that people share an interest. Regarding community structure, both types of networks have it (with modularity above 0.3), but size of communities in developer networks is smaller. Additionally, authors analysed DSNs over different time periods, and it occurred that 6-month DSN has almost the same properties as all developer networks created over larger time period.

Regarding developer social networks evolution, paper authors also did a very broad research. They found out that a number of active Mozilla developers stabilised after 2004 (when a release of Firefox 1.0 took place). Average path length between two developers increases slowly over time, but mostly it doesn't exceed 3. Also a modularity increases over time, so that communities (corresponding to different projects) become more tight-knit, but the size of communities remains small without significant changes.

In the last section of this paper authors decided to analyse evolution of communities. They found out that this evolution contains patterns like derivation, split, merge, extinct and emerges[5]. They also distinguished evolution paths corresponding to history of Mozilla projects and showed that communities at the beginning were very dynamic, but than they settled down into fixed groups.

In my opinion it was the best and most advance of all papers I read for this project. It contained a very wide, deep and thoughtful research. Results were also very carefully evaluated. For example, authors concluded that communities correspond to different projects in Mozilla (e.g. Firefox or Thunderbird). They evaluated it by analysing users attending different offline meetings in the Mozilla Summit 2010. When applying probabilistic Louvian algorithm for finding communities, they generated 50 data sets with different nodes ordering and presented all results.

Although it is quite old research (2011), I will definitely recommend it for all people interested in developer social networks. It's clear, interesting and carefully prepared.

[5]It's consistent with community evolution taxonomy proposed by Lin et al. Their work is referenced in original paper.

## 3.2 Coding Together at Scale: GitHub as a Collaborative Social Network

In the introduction the authors said: "In this paper, to the best of our knowledge, we present the first in-depth quantitative analysis of GitHub, as a unique example of large-scale real-world collaboration platform mainly used for software projects". And it is a really good characterization of this article. Authors focused on many different and interesting topics related to GitHub platform.

They analysed public events that occurred on GitHub between March 2012 and September 2013 and created four networks. First was directed followers graph, as on GitHub users can "follow" each other. Secondly paper authors created bipartite graph describing collaborations of users on repositories (users are "collaborators" in a project if they have write permissions). Similarly they obtained bipartite graph of all users contributing to repositories. They also created "stargazers" bipartite graph of users and repositories, as on GitHub users can assign "stars" to projects they like.

Researchers found out that all distributions of contributors, collaborators and stargazers per project are power-law. Additionally they observed that reciprocity in followers graph is very rare comparing to other social networks, since on GitHub people don't want to be distracted by notifications they are not interested in. The other interesting fact was that user's activity and a number of his/her followers are not correlated.

Paper authors also investigated impact of geographical locations on projects, and similarly to general social networks, users interact more with people who live relatively close. For me it was very surprising. I assumed that while choosing open source projects to contribute to, programmers don't look at locations. But maybe some projects are more popular in particular workplace or university backgrounds, so actually it makes sense.

This paper was also very rich and interesting. I think it is the most advanced work on GitHub itself. But I also have an impression that authors tried to include too much information (there were also small chapters about tree structure of forks, users rewards and other I didn't include in my summary), so it was a bit chaotic. But nevertheless, it's a really good piece of research.

## 3.3 Network Structure of Social Coding in GitHub

This paper was created before the previous one and its authors believed it was the first research investigating network structure of GitHub. They presented some network properties and identified influential project with PageRank algorithm.

Two weighted undirected graphs was constructed: graph of projects (two projects were connected if and only if there was at least one user contributing to both of them; weight was a number of such users) and graph of developers (two users were connected if and only if they contributed to at least one common project; weight was a number of such projects). It came out that while project network follows a power law degree distribution , it's not a case in developers network. That's because some projects have a really large number of contributors.

Using PageRank gave good results. In top five most popular projects it identified i.e. *homebrew, rails* or *node*, which indeed are extremely popular tools and libraries used and developed by thousands of programmers.

This paper was really short and simple, actually more like university homework than real scientific research. Still, it was interesting and provides a good introduction for people interested in analysing network properties of GitHub.

## 3.4 Combined Methods, Thick Descriptions: Languages of Collaboration on GitHub

This is very short paper, but it's still worth mentioning. That's because authors used here GitHub information ignored in other papers, that is users main programming language. They created a weighted graph of GitHub users with edges representing number of pull requests or forks of each others repositories. Than they narrowed it to users with Python as their main language choose one cluster and provided for it some measures like number of communities, modularity, averaged weighted degree etc. This was quite an interesting idea.

But this paper is really difficult to read. It lacks a lot of important information (for example, which algorithm they used for community detection) and is quite chaotic. In my opinion authors couldn't decide who is a target of their work and they focused sometimes on unimportant details while ignoring the necessary ones.

# 4 Methodology

## 4.1 Data and Networks

There is a GitHub API that allows downloading structured data from webpage. Unfortunately it doesn't provide queries for extracting which issues where mentioned in current one or in which this one was mentioned. That's why I decided to do classical web crawling and download all web pages containing issues that were created to that point (30000). Then, using regular expressions, I extracted from each issue its number, labels, milestones and issues that linked to this one. Using this data I created a directed graph. Each node contained one issue with additional parameters representing labels and milestones. Directed edges represented referencing from one issue to another. Undirected graph was created from the directed one by removing directions of edges (if it resulted in two edges between two nodes, redundant was removed as well).

## 4.2 Louvain algorithm for community detection

To find a community structure I used Louvain algorithm. It's much faster than Girvan-Newman algorithm we learnt during lectures, in which betweenness needs to be recalculated after each edge removal. And as my graph was quite big, using a more efficient algorithm seemed like a good idea.

This algorithm was presented in [6]. The main idea is to find communities "bottom-up" rather than "top-down" (like in Girvan-Newman) by optimizing modularity locally. Also, Louvain algorithm may give slightly different results for different ordering of nodes in graph. But it's incredibly fast and efficient, as on typical and sparse data its complexity seems to be linear.

At the beginning each node represents its own community. Louvain algorithm consists of two phases that are repeated iteratively. In the first phase it iterates through all nodes. For each node $n$ and for all its neighbours $m_1, ..., m_k$ algorithm tries to move $n$ to community of $m_i$ and calculates change of modularity. Then it chooses the best result, and if it is positive, it changes community of $n$. It is repeated till no more improvements are possible. Then it moves to second phase and creates new graph where nodes are best communities chosen in the previous phase and weighted edge between two communities represent sums of all edges between nodes in these communities. Then the first phase is applied to this new graph.

### 4.3 Tools

The whole project was created in **Python 2.7.11** with **networkx**[6] library. For drawing small diagrams I used **matplotlib**[7] library. In case of big network visualisation I used **pygraphviz**[8] (Python interface for Graphviz[9]) to create a .dot file and Linux command line tool **sfdp**[10] to draw .svg from created .dot. As networkx doesn't provide Louvain algorithm, I used for it third party implementation from *http://perso.crans.org/aynaud/communities/index.html*. Regarding all other algorithms including PageRank, connected components and assortativity coefficient I used implementation provided by networkx.

I run all analysis in **IPython Notebook**[11] as it's extremely useful and convenient tool. It enables user to run small parts of code (cells) in web browser making a beautiful and easy to read notebook full of code, diagrams and results. My notebook is attached as *extra.pdf*.

## 5 Results

### 5.1 The most important issues identified in directed graph by PageRank algorithm

Among 10 most important issues identified by PageRank 7 had label *metabug* (in the whole graph there are only 62 nodes with this label). It shows clearly that metabugs are special, as they are created to group issues rather than to represent a particular problem. For example, the most important node was Issue no. 29329 The Standard Library Documentation Checklist (Figure 1). It was created to make updating standard library documentation in more systematic way.

[6] http://networkx.github.io
[7] http://matplotlib.org
[8] https://github.com/pygraphviz/pygraphviz
[9] http://www.graphviz.org
[10] http://linux.die.net/man/1/sfdp
[11] http://ipython.org/notebook.html

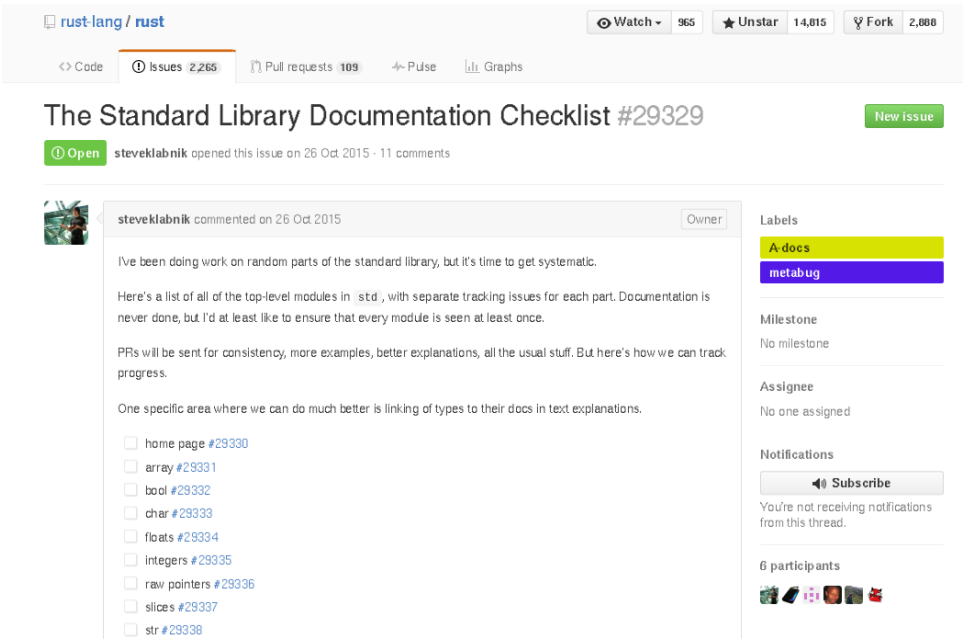
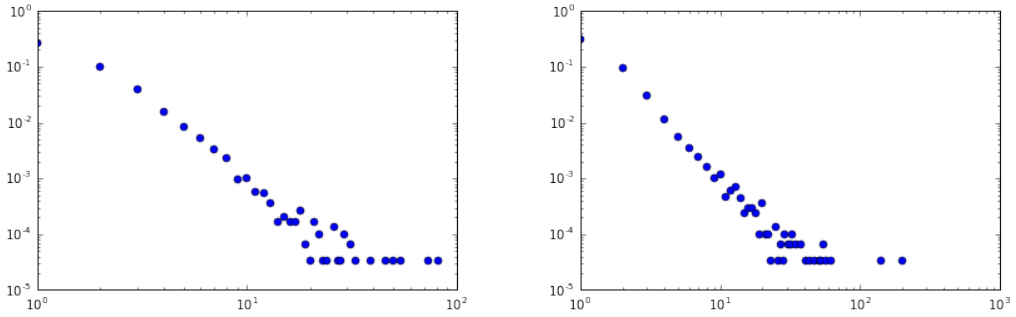



Figure 1: Screenshot of the most important issue identified by PageRank

## 5.2 Degree distribution and mixing pattern in directed graph

### 5.2.1 Degree distribution

The log-log plots of in- and out-degree distribution are presented on Figure 2. Similarly to many other networks, they also have power-law shape. For both in- and out-degrees I investigated Issues with the largest degree. It turned out that tails for both distributions represent mostly metabugs.



(a) in-degree distribution

(b) out-degree distribution

Figure 2: In- and out-degree distribution in directed graph of issues

### 5.2.2 Mixing pattern

Mixing pattern is slightly disassortative with assortative coefficient $\alpha = -0.01$

## 5.3 Community structure in undirected graph

### 5.3.1 Connectivity

Connectivity is strongly related with community structure. Communities make sense within connected components only. That's why I decided to investigate connectivity first. I assumed that there exists one large connected component (as it always does in "natural" networks) and I wanted to narrow my community detection to it.

Graph of issues is very sparse. Actually almost 10000 nodes (1/3 of all) are completely isolated. After removing all these nodes, there are still 2344 connected components. The largest connected component (Figure 3) consists of 14082 nodes and 19505 edges and indeed it is the significant one, as other have less than 25 nodes. But it's still worth to take a look at smaller components. The examples of them are presented in Figure 4. Colour of a node represent its milestone, where green means there is no milestone assigned to this node.

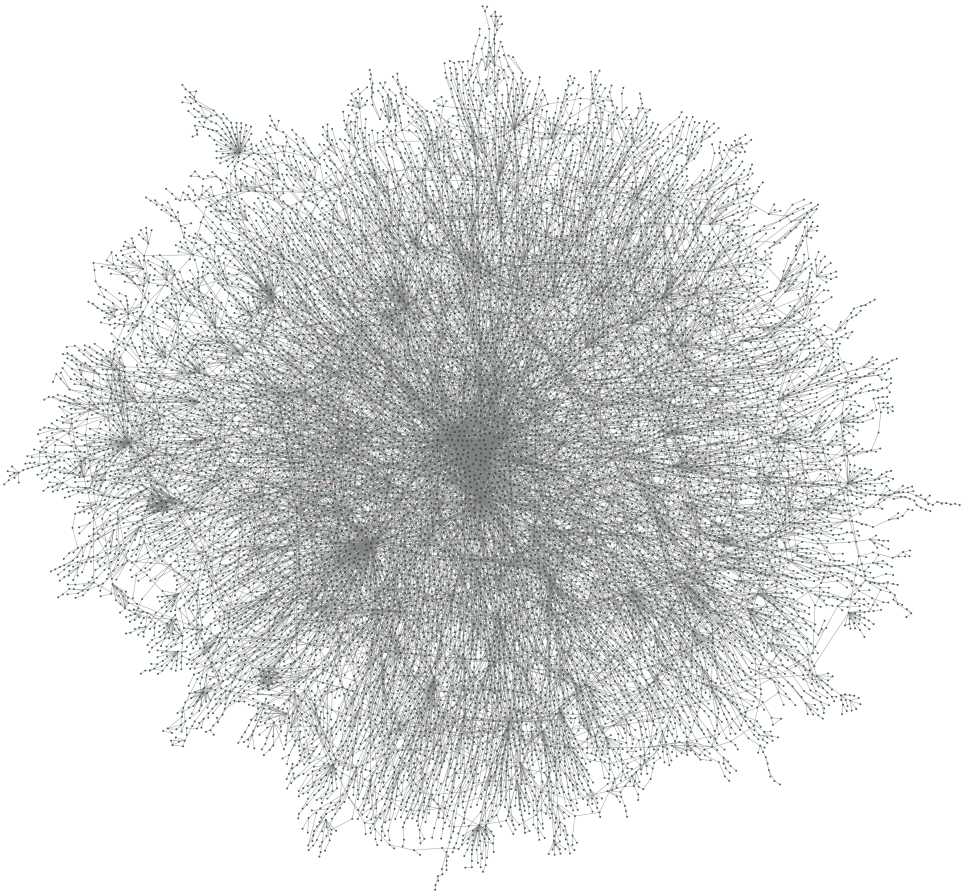
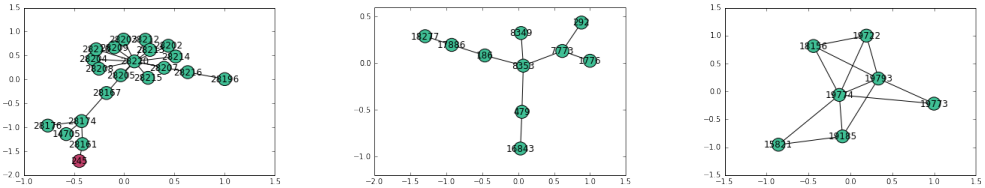


Figure 3: Largest connected component



(a) (b) (c)

Figure 4: Examples of small connected components

### 5.3.2 Community structure

Louvain algorithm detected extremely strong community structure. On the forth iteration (of both phases) it created 97 communities. Modularity was equal to 0.8885. I have to admit I've never seen so high modularity in a "natural" network. Sizes of communities were placed between 6 and 885 with majority between 40 and 200.

## 5.4 Correlation between community structure and other properties (milestones, labels and time of Issue creation)

I investigated the following properties of each community:

- Number of nodes
- Labels distribution
- Milestones distribution
- Degree distribution and assortative coefficient in community induced subgraph
- Oldest and newest issue
- Information if 40% of nodes are within range of 500[12]

It turned out that most of communities are very similar in terms of properties below.

Generally there is very few nodes with assigned milestone, so communities consist mostly of issues with no milestone assigned. I knew it before, but I hoped that maybe some communities are gathered around particular milestone. But it was not case. Actually there where no community with 50% or more nodes belonging to the same milestone.

I defined "leading label" as a label that occurs in more than 50% of nodes within a community. Unfortunately only two out of 97 communities have a leading label. Even when I decreased threshold to 30% of nodes, still over 85% of communities don't have it.

[12]Issues get consecutive natural numbers when created, so it's easy to say how "close" they are. 40% of nodes and distance of 500 were chosen as reasonable parameters after careful look at data and many experiments with other values.

Almost all community induced subgraphs followed power-law degree distribution and disassoratative mixing pattern.

When it came to "closeness" in time of nodes in community, the newest and the oldest one were often very far away. In almost all communities of size over 100 the number of the oldest one was smaller than 1000 and the number of the newest one was larger than 29000. Also only 24 out of 97 communities have 40% of nodes within range of 500 and they were almost only small communities.

# 6 Discussion

As I've already mentioned in the introduction, results of PageRank were not surprising. They confirmed my hypothesis that metbugs play a significant role in network structure of Issues. But metabugs seem to me quite unnatural, as the idea of Issue is to represent a real problem in a project, not to list other Issues. So their existence in a repository, and what is more important, their role in the network structure show that there is definitely some logic missing in Github issue tracker. Theoretically milestones should gather Issues related to stage of a project and labels should distinguish topics, but it's evidently not enough to make a transparent structure of Issues.

Regarding connectivity, I think that it's quite natural that there is a lot of Issues with degree equal to zero. I suppose that in a big project one can find many small bugs and problems that don't really link to another. But 1/3 is a really large number and I actually believe it is too many. I don't believe that there is so many problems completely unrelated to others. In my opinion this number confirms sloppiness of developers and that at this moment Issues don't show real structure of problems in a project.

Community structure of this network is simply mind-blowing. In a lecture we learnt that modularity between 0.3 and 0.7 means significant community structure. But what about 0.89? It's extremely high result. Well, one could expect something like this in some "special" graphs like for example almost disconnected cliques. But I ran community detection on a largest connected component of the "natural" network. Moreover, results were nontrivial and communities have sizes between 6 and 885 with majority between 40 and 300. So it doesn't look like a trivial solution or something like this.

What made this discovery even more unbelievable were statistics for labels, milestones and creation times of nodes. It seemed to me quite obvious that at least one of those properties should correspond somehow to community structure. After all, labels and milestones were designed exactly for purpose of grouping Issues. And when something is broken or new feature is being designed, there should be several Issues created in a similar time that represent this particular problem. But it turned out that it's also quite unrelated to communities.

Well, I can think of some explanations, but I'm not sure they are correct. First of all, graph of Issues is quite specific. When one thinks of social network, nodes can be in some kinds of relations, but actually each node decides for its own. And Issues are manged by people who can be very different and have various methods of managing problems. What I mean is that two quite unrelated issues that are managed by one person may have similar linking pattern specific for their owner. I strongly believe that it is something that should be investigated in the future and can produce some really exciting results.

The other property that is difficult to investigate but may be related to community structure are commits. Or, to be more specific, parts of code touched by commits for particular Issues. There may be cases that problems are really unrelated (for example code refactoring and small bug in a logic), but they happen to touch the same code. It may be natural to link them in this case, for example with comment "I'm going to fix it after user X will finish his refactoring (Issue #YYY) to avoid merge conflicts". So this is something I'd really like to investigate. And it can be done, but collecting data and defining what "the same code" means can be quite tricky.

To sum up, I strongly believe that my research is quite fascinating. One could find my unexpected results very inspiring. I hope someone may push it forward and find what's really going on with Issues and maybe develop some brilliant method to improve existing bug tracking systems.

# References


[1] "GitHub Press Info" retrieved on 6th Jan 2016.
https://github.com/about/press.

[2] Qiaona Hong, Sunghun Kim, S.C. Cheung, Christian Bird. Understanding a Developer Social Network and its Evolution. 2011 27th IEEE International Conference on Software Maintenance (ICSM).

[3] Antonio Lima, Luca Rossi, Mirco Musolesi. Coding Together at Scale: GitHub as a Collaborative Social Network. 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org).

[4] Ferdian Thung, Tegawende F. Bissyande, David Lo, Lingxiao Jiang. Network Structure of Social Coding in GitHub. 17th European Conference on Software Maintenance and Reengineering (CSMR 2013), Mar 2013, Genova, Italy. pp.1-4. hal-00790772

[5] Nicholas M. Weber. Combined Methods, Thick Descriptions: Languages of Collaboration on GitHub. ASIST 2012, October 28-31, 2012, Baltimore, MD, USA.

[6] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Renaud Lefebvre. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008(10), P10008 (12pp)