Open Source Software Development License Structures

*Poonacha Medappa, HEC Paris*

**Introduction:**

Open source software (OSS) has come to occupy a very important place in the Information Sciences space. Even though it appears as an emerging trend, we can trace the origin of open source projects to the 1960-1970’s, when key developments in the software industry occurred in a collaborative research and academic environment(Lerner, Tirole , 2002). Over time, two dramatic changes in the environment have taken OSS from research labs to corporations and our homes.

1. There have been major developments seen in the licensing, copyrights and legal aspects, which provided some formalization for the creation and distribution of open source software (eg. free, BSD, MIT, ASF, GNU GPL (General Public License) etc.).
2. The diffusion of Internet meant that anyone could be a collaborator and collaboration was not restricted to the academic or corporate environment. i.e. anyone, anywhere who had the knowledge and interest could now contribute and become a part of the open source community.

The unique feature of the open source industry is the fact that a loosely integrated community of developers comes together to build functional software with out any (or very little) direct monetary benefit. As OSS took a greater role in the software development industry, there emerged a need to protect “openness” of OSS software from the potential commercial hijack of the software developed (Stallman, 1999). In this light, new forms of licenses emerged that, unlike traditional licenses, were meant to protect the freedom of the software developed. In specific, there emerged two distinct license regimes, which came out of two different schools of thought. The first was termed as “copyleft” licenses and was meant to completely restrict the sublicensing and commercialization of the code. The way these licenses restricted the commercialization of the code was by introducing a “viral” feature for the license. This viral feature ensured that if the code with a “copyleft (restrictive)” license was used to build new software, then, the new software built should also have the same license. The second school of thought believed that the developer of the open source software should have the right to sublicense or commercialize the software developed. This led to creation of permissive licenses that had most of the features of the restrictive (copyleft) license but also gave the developer the liberty to commercialize and sell the software that she developed.

The emergence of two distinct license regimes and its impact on the nature of the artifact (software) produced and structure of collaboration is of concern in this article. In specific, this article tries to expand the work done by Howison and Crowston, 2014 on collaboration through open superposition.

In their field-based research, Howison and Crowston argue that collaboration through open superposition is at the core of the success of community-based OSS projects (Howison & Crowston, 2014). This article takes this result forward and tries to understand ***How the two license regimes differ in terms of superposition of tasks?.*** We use a network-based approach with individual tasks as nodes to test the propositions formulated. We hope that this article can be the starting point of an empirical research that can help developers and organizations make informed decisions on the type of license to be adopted.

This research paper is structured as follows; In Section 1, we highlight the two license regimes that exist in the case of OSS. In Section 2, we take a look at Howison and Crowston findings on collaboration through open superposition. Section 3, highlights the proposed expansion to their work by formulating a set of propositions. Finally in Section 4, we briefly discuss the potential implications of the results.

There are different terminologies that refer to Open Source Software, this includes, amongst others - FOSS (Free Open Source Software) and FLOSS (Free/Libre Open Source Software). Although these terminologies are meant to refer to slightly different policies that underlie open source development, for this article, we stick to using the acronym OSS to refer to Open Source Software development in general.

**Section 1: Open Source Software License Regime**

Traditional (proprietary) software developed by individuals or institutions are protected by traditional copyright laws, which make the software a private good. In this case, the individuals or institutions that created the software “own” the source code. Thus in the case of proprietary software, the source code is not made available to the user for reuse and modification.

The Free Software Foundation (FSF) created by Richard Stallman in 1983 tried to create a legal basis which would allow collaboratively developed code to not only remain free to use, but also ensure the free availability of the source code (Stallman, 1999). This movement gave rise to software that followed the new copyleft (restrictive) licenses. Software that followed restrictive licenses gave any user the right to copy and redistribute the software and also make modification to the source code (Stallman, 1999). In addition to the above, any enhancement to code and even any proprietary software that made use of the source code would be bound by the same license and would be forced to remain open. This “*vira*l” feature of the license made sure that any offshoots of the restrictive licensed software would also remain free or open. The main idea behind the viral feature of the restrictive licenses was to prevent the collaboratively developed code from being commercialized and eventually “closed”.

In an effort the relax the viral feature of the restrictive licenses, new “permissive” licenses emerged - like BSD, ASF and MIT licenses which provided most of the “open source” features of the earlier restrictive license but also gave the developer the freedom to change the license policy of the new software which was built. Following this line of thought, the Open Source Initiative (OSI) and its Open Source Definition (OSD) was created in 1997 to provide developers the freedom to use their code whichever way they choose (OpenSourceInitiative, 1999). The table below highlights the features of the restrictive and permissive license regimes.

|  |  |  |
| --- | --- | --- |
| **Features** | **Restrictive License (Copyleft) (Eg. GNU GPL V2.0)** | **Permissive License (Eg. MIT,)** |
| **Required** | * ***Disclose source - Source code must be made available while distributing software*** * Include a copy of license document in the copy of the code * Indicate significant changes made to the code | * Include a copy of license document in the copy of the code |
| **Permitted** | * ***This software may be used for commercial purposes*** * The software may be distributed at a price * Software may be modified * Express grant of patent rights from contributor to the recipient | * ***This software may be used for commercial purposes*** * The software may be distributed at a price * Software may be modified * Express grant of patent rights from contributor to the recipient * ***Sublicense – The authors CAN grant a sublicense by modifying any part of the existing license*** |
| **Forbidden** | * ***Sublicense – The authors cannot grant a sublicense by modifying any part of the existing license*** * Software author cannot be held liable | * Software author cannot be held liable |

Table1: Main features of restrictive and permissive licenses (*Source –www.Choosealicense.com)*

**SECTION 2: Collaboration through open superposition**

Howison and Crowston, in their field based research find that in OSS communities, (1) the overwhelming majority of work is accomplished with only a single programmer working on any one task, and (2) Tasks that appear too large for any one individual are more likely to be deferred until they are easier rather than being undertaken through structured teamwork (*Howison & Crowston, 2014*). To explain the reasoning behind the two findings, the authors resort to the well-researched area of motivation to contribution and task coordination.

1. Motivation to contribution: Why does OSS communities attract single programmer, single task related work? : To answer this question the authors begin with Ryan and Deci’s (2000) spectrum of motivation, which extends the two well-known categories of motivation, intrinsic motivation (driven by interests and enjoyment of the individual) and extrinsic motivation (driven by rewards). Ryan and Deci argue that extrinsic motivation should be further broken down according to the locus of regulation involved, from controlled (by others) to autonomous (self-directed) as one approaches intrinsic motivation. Ke and Zhang found that OSS participants with more autonomous regulation produced greater task effort and greater persistence and consistency. Based on this line of thought, motivation in the case of OSS projects tended to attract single programmer, single task related work.
2. Coordination of tasks: How does coordination of tasks occur when people work independently on single tasks? To answer this question the authors resort to coordination theory, which provides a modeling framework of actors performing tasks, where tasks might require or create resources of different kinds. A particular concern in OSS software development is that a development task (task B) often requires the outputs of some other task (task A) before it can be performed, thus creating a task–task dependency. In other words, the likelihood of completing a task depends on the completion of necessary prerequisite tasks (Howison and Crowston, 2014). If Task A already exists then the situation describes simple sequential layering of independently motivated tasks: ***collaboration through superposition***.

If Task A is not complete, the motivation for work on a task (Task B) may be in place, but the groundwork is not in place (Task A) and therefore its dependencies are not satisfied. In this scenario the authors contend that rather than engaging in co-work or undertaking both tasks together, the developer tends to defer the implementation of the Task B until work on Task A gets completed (wait untill collaboration through superposition is possible). The main reasoning that the authors provide to explain this observation is that - co-work creates the potential for a dependency problem: if one actor finishes task B but the other actor is unable to finish task A, then the first actor has wasted effort, as the success of task B depends on the completion of task A. The uncertainty involved in predicting the implementation of Task A tends to reduce the motivation for co-work. Similarly undertaking both Task A and B sequentially by the same developer tends to reduce the motivation; since any uncertainty about being able to finish both reduces the motivation to attempt either one.

In summary, the authors provide a fairly straightforward explanation for the observed working pattern: *superimposed individual work is the predominant organization of tasks in OSS development because this type of work has the fewest dependencies and the simplest motivational situation. In particular, the superposition of individual work is more likely to be well motivated because it increases autonomy and competence without eliminating relatedness. Work that cannot be completed in this manner might be undertaken through co-work, if the tasks seem likely to be completed and the loss of autonomy is balanced by the increase in relatedness. If those conditions seem unlikely, as is often the case, the work is deferred until other work renders it achievable through the superimposition of individual work.*

**Section 3: Collaboration Through Open Superposition in the two license regimes**

The main aim of this article is to create the foundation for expanding the work done by Howison and Crowston. In specific, we try to understand the implication of restrictive and permissive license regimes on the nature of collaboration through open superposition. We draw ideas from network theories with tasks as individual nodes and each OSS project representing the boundary of the network. Task can be defined as the sequence of actions contributing to a particular task outcome. Where task outcome is a change to the shared output of the project —A new feature, a fixed bug, updating documentation (Howison and Crowston, 2014). In this representation, superposition of tasks would imply a simple linear network structure as shown below.

Figure1: Linear network structure – superposition of tasks

In contrast, if superposition was not the preferred mode of collaboration, we would observe either a centralized with broker network representation or an interconnected network structure as shown below.

Figure2: Centralized with broker network structure

Figure3: Inter connected network structure

1. **Degree of superposition:**

To study the superposition across the two license regimes, we conceptualize “degree of superposition” as the ratio of the average length of the task chains in a network (OSS project) to the total number of network connections between tasks in the network. For eg: Figure 1 would have a degree of superposition = average length of task chain (4) / Number of network connections (4) = 1. Similarly, Figure 2 would have a degree of superposition = ¼.

1. **Superposition in the context of license regimes:**

To understand how the two license regimes differ in terms of degree of superposition, we go back to Ryan and Deci’s, 2000 motivation framework. When we look at restrictive licenses, we contend that intrinsic motivation is the predominant mechanism that drives contribution. This is because the developers building on restrictive license have limited monetary benefit (extrinsic motivation) since they cannot commercialize their code. In contrast, permissive license allow for some degree of extrinsic motivation as the developers can choose to commercialize their contribution. Ryan and Deci, suggests that the locus of control tends to shift from controlled by others to largely autonomous as the motivation shifts from extrinsic to intrinsic. Further, Howison and Crowston, contend that the collaboration through open superposition is largely due to the autonomous nature of work adopted in the case of OSS. Based on this premise, we formulate the following propositions:

1. *Proposition 1: OSS projects in general have higher (as compared to traditional projects ) intrinsic motivation and thus have higher degree of superposition of tasks. (This is a direct proof of Howison and Crowston, 2014 findings i.e. OSS are built by collaboration through open superposition)*
2. *Proposition 2: OSS projects with restrictive license have higher (as compared to permissive license) intrinsic motivation and thus have higher degree of superposition of tasks*
3. **Impact of maturity of the project on the degree of superposition**

Ryan and Deci’s motivation spectrum helps us understand why the locus of control tends to shift from controlled to self directed as the motivation shifts from extrinsic to intrinsic. While this theory can explain why a single programmer, single task work is dominant in the case of OSS projects, it cannot explain how the motivation sustains itself over time (von Krogh et al 2012). Based on the social philosophy of Alasdair MacIntyre, von Krogh et al attempts to answer the question “Why do developers sustain the social practice of OSS development”. According to the authors, *“OSS developers sustain the social practice of OSS development because social practice instills the motivation to uphold its standards of excellence over time.*” In other words, there is some inertia that is created through the creation of social practices associated with that particular OSS project. Based on this premise, we propose the following:

1. *Proposition 3: OSS projects following either of the license regimes should display fairly low variation in terms of degree of superposition with time.*
2. **Methodology:**

While this section of the article requires considerable thought, we make an initial attempt to identify a proposed methodology to hypothesize and test the propositions developed. The research setting is proposed as GitHub. GitHub is a platform developed by Google to host collaborative software development projects. GitHub hosts OSS projects that have both restrictive and permissive licenses. By controlling for project size (in terms of number of active developers), we believe the propositions can be tested with considerable reliability.

GitHub project data from 2011 is publically available and can be analyzed using BigQuery, which is a cloud based big data analytics platform. A key challenge will be to develop queries to extract the license data for each project and the citation which links one task to another. While we appreciate the difficulty of this task, we believe it should be possible with some study of the data available. Once the task citation and license data is extracted, we can build a task-task matrix that links one task to another based on its citation. Once we have this matrix, simple analysis can allow us to calculate the degree of superposition for each project, which can be averaged and compared across the two license regimes.

**Section 3: Discussion and conclusion**

This research is an initial attempt to understand how different license regimes invoke different motivation mechanisms, which in turn can result in fairly distinct task collaboration structures. While this research is fairly simplistic, we believe the result can have significant theoretical and practical implications. Firstly, the results (specifically proposition 1) can provide empirical support to Howison and Crowston theory of collaboration through open superposition. Secondly, the results can expand the current understanding of OSS software development, specifically that related to OSS licenses. With regards to practical implementations, we believe the results can provide some guidelines to the optimal choice of license based on the nature of software envisioned. The practical implications grows in significance as an outcome of increasing participation of organizations in the OSS development. For an organization looking to release its proprietary software to the OSS community, an important decision is regarding the choice of license to be adopted. We believe that the results from this research can help in this decision process.

**References**

1. Lerner, J., and Tirole , J., "Some Simple Economics of Open Source" *Journal of Industrial Economics,* 52: 197-234, 2002
2. Charles M. Schweik , 2006 **“***Free/Open-Source Software as a Framework for Establishing Commons in Science***”** Chapter 10 in, *Understanding Knowledge as Commons*, 277-320, MIT Press
3. OpenSourceInitiative,1999, *'Open Source Definition*' http://www.opensource.org/osd.html (accessedNovember14,1999).
4. Stallman, R. M. 1999. “The GNU Operating System and the Free Software Movement.” In C. DiBona, S. Ockman, and M. Stone, eds., *Open Sources: Voices from the Open Source Revolution.* Sebastopol, CA: O’Reilly and Associates.
5. Feller, J., and B. Fitzgerald. 2002. *Understanding Open Source Software Development.* London: Addison-Wesley.
6. *Richard Stallman; 1999 “Open Sources: Voices from the Open Source Revolution”; O’Reilly Publication*
7. *Howison, J., and Crowston, K. 2014. "Collaboration Through Open Superposition: A Theory of The Open Source Way." MIS Quarterly*
8. Ryan, R. M., and Deci, E. L. 2000. “Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions,” *Contemporary Educational Psychology* (25:1), pp. 54-67.
9. Ke, W., and Zhang, P. 2010. “The Effects of Extrinsic Motivations and Satisfaction in Open Source Software Development,” *Journal of the Association for Information Systems* (11:12), pp. 784-808.
10. Georg von Krogh, Stefan Haefliger, Sebastian Spaeth, and Martin W. Wallin; 2012 “Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development” MIS Quaterly, Vol. 36 No. 2 pp. 649-676

*I have noticed that as the size of the project increase the degree of superposition decresses !!!!*

*Repository\_pushed\_at gives the stable version of the code at one instance all events called are indicated as created\_at*

*Repository\_pushed\_at gives the time of the last pushed changes to the master !!*

*Creation of a branch is termed as a “Create Event” . Even when a branch is created the repository\_pushed\_at” time updates itself!!*

*I created a pull request. Strangely the created\_at and pushed\_at events are the same. IS this because I am the owner of the repo ?*

*Next km-poonacha makes a few changes to readme file:*

"type": "PushEvent",

"created\_at": "2015-09-07T09:04:56Z"

*"created\_at": "2015-09-07T08:47:08Z",*

*"updated\_at": "2015-09-07T08:47:08Z",*

*"pushed\_at": "2015-09-07T09:02:45Z",*

*Then km-poonacha creates a pullrequest*

"type": "PullRequestEvent",

"created\_at": "2015-09-07T09:08:36Z"

"pushed\_at": "2015-09-07T09:04:56Z"

Randomeshwar merges and closes the changes !

*"type": "PushEvent",*

*"created\_at": "2015-09-07T09:13:27Z"*

*"pushed\_at": "2015-09-07T09:04:56Z",*

*km-poonacha deletes the created branch*

"type": "DeleteEvent",

"created\_at": "2015-09-07T09:28:42Z"

*Now km-poonache makes changes that get rejected by randomeshwar*

"type": "PushEvent",

"name": "km-Poonacha/newtest",

created\_at": "2015-09-07T09:48:00Z"

"type": "PullRequestEvent",

"created\_at": "2015-09-07T09:49:45Z",

"full\_name": "km-Poonacha/newtest",

"created\_at": "2015-09-07T09:04:34Z",

"updated\_at": "2015-09-07T08:47:08Z",

"pushed\_at": "2015-09-07T09:47:59Z",

"full\_name": "randomeshwar/newtest",

"created\_at": "2015-09-07T08:47:08Z",

"updated\_at": "2015-09-07T08:47:08Z",

"pushed\_at": "2015-09-07T09:13:27Z",

Randomeshwar closes this requests with a comment.

"type": "PullRequestEvent",

"created\_at": "2015-09-07T09:49:45Z",

"updated\_at": "2015-09-07T09:53:08Z",

"closed\_at": "2015-09-07T09:53:08Z",

*"full\_name": "km-Poonacha/newtest",*

* *"created\_at": "2015-09-07T09:04:34Z",*
* *"updated\_at": "2015-09-07T08:47:08Z",*
* *"pushed\_at": "2015-09-07T09:47:59Z",*

"full\_name": "randomeshwar/newtest",

* *"created\_at": "2015-09-07T08:47:08Z",*
* *"updated\_at": "2015-09-07T08:47:08Z",*
* *"pushed\_at": "2015-09-07T09:49:45Z"*

*"type": "IssueCommentEvent",*

*"created\_at": "2015-09-07T09:53:08Z",*

*RANDOMESHWAR creates a new branch and sends a pull request:*

* "type": "CreateEvent"
* "name": "randomeshwar/newtest",
* "ref": "randomeshwar-patch-2",
* "created\_at": "2015-09-07T13:49:45Z"
* "type": "PullRequestEvent",
* "name": "randomeshwar/newtest"
* "created\_at": "2015-09-07T13:51:59Z",
* "full\_name": "randomeshwar/newtest"
* *"created\_at": "2015-09-07T08:47:08Z",*
* *"updated\_at": "2015-09-07T08:47:08Z",*
* *"pushed\_at": "2015-09-07T13:52:00Z",*
* *"type": "PushEvent",*
* *"name": "randomeshwar/newtest",*
* *"created\_at": "2015-09-07T13:55:19Z"*
* *"full\_name": "randomeshwar/newtest",*
* "*created\_at": "2015-09-07T08:47:08Z",*
* *"updated\_at": "2015-09-07T08:47:08Z",*
* *"pushed\_at": "2015-09-07T13:55:18Z",*

*km-poonacha adds a new file to the repo and commits directly*

* type": "PushEvent",
* "created\_at": "2015-09-07T14:02:25Z"
* Strangely the pushed at field of randomeshwar/newtest remains unchanged
* *"full\_name": "randomeshwar/newtest"*
* "created\_at": "2015-09-07T08:47:08Z",
* "updated\_at": "2015-09-07T08:47:08Z",
* "pushed\_at": "2015-09-07T13:55:19Z",

*The unique thing is that repository\_pushed\_at gives changes done to the corresponding files*

*In the new file created Km-poonacha makes changes and commits the changes to file directly. Note the repository associated in km-poonacha/newtest*

* "type": "PushEvent"
* "name": "km-Poonacha/newtest",
* "created\_at": "2015-09-07T14:13:25Z"

*Next we create a pull request for the for the randomeshwar/newtest master repo*

* "type": "PullRequestEvent",
* "name": "randomeshwar/newtest",
* "created\_at": "2015-09-07T14:26:37Z",
* "full\_name": "randomeshwar/newtest"
* *"created\_at": "2015-09-07T08:47:08Z",*
* *"updated\_at": "2015-09-07T08:47:08Z",*
* *"pushed\_at": "2015-09-07T13:55:19Z",*

*Randomeshwar pushes the created pull request: Note the pushed\_at time has changed after this push event not before*

* "created\_at": "2015-09-07T08:47:08Z",
* "updated\_at": "2015-09-07T08:47:08Z",
* "pushed\_at": "2015-09-07T14:26:57Z",

*NOTE:*

*For each PullRequest created bykm-poonacha, when the pull request is accepted by the randomeshwar, a new pull request and a new push request event is created .. the new pull request event only updates the time of merger of the repos. Note that the new pull request and the push event have the same time of creation (+- a few millisecs).*

*If a pullRequest is rejected. A new pullrequest event is created but no push event.*