Open Sources: A Visualization for Understanding Project Evolution and Communication Patterns in F/OSS Communities

Aaron Zinman*
MIT Media Lab

Judith Donath†
MIT Media Lab

ABSTRACT

Free software projects are a growing and important source of software. They range in size from one developer to hundreds. Unlike traditional models of software development, they use the Internet to collaborate in a decentralized, self-organizing fashion. It is important to understand how such communities can prosper without traditional centralized control. Open Sources is a visualization that operates on existing code bases and mailing list archives in order to understand the relationship between the community’s communication patterns and the code. It attempts to reveal characteristic signatures that may result in successful projects, as well as the locally hidden history behind code and mailing list archives. A proof of concept is demonstrated with an informal evaluation.

CR Categories and Subject Descriptors: D.2.2 [Software Engineering] Tools and Techniques – User interfaces; H.5.1 [Information Interfaces and Presentation] Multimedia Information Systems; H.5.3 [Group and Organizational Interfaces]; Additional Keywords: open source, communication, source code, collaboration

1 INTRODUCTION

The Free and Open Source Software (F/OSS) community is a powerful new force within the world of computer science. Creativity and fun drive developers to come from both industry and hobby [11]. They are creating enterprise quality, large-scale systems even though they usually not have met in person. It is a complete departure from typical methods of project management and resource allocation, using only the Internet to orchestrate collaborative behavior within their decentralized architecture. This self-organization takes on a range of different forms. Linux, an exceptionally large project of hundreds, uses kernel maintainers to ensure quality and direction from the abundance of patches. Most projects, often consisting of a small number of developers [11], willingly grant individuals write access to the public read-only repository. As Free Software becomes an increasingly powerful force, it is important to understand how successful projects, large and small, employ organizational and collaborative practices. This leads to questions such as does the method of organization matter? Is there a winning strategy for code management and ownership, and can well-structured communication extend its form?

Open Sources attempts to expose the dynamics of group collaboration and community evolution through affording interactive examination of the instances of communication through mailing lists and commitment of code to repositories. The tool can also be used to compare various projects to understand how they have individually changed over time, and through comparison create a vocabulary of development practices that successful projects share.

In order to understand how these communities work, it is important to understand the relationship and patterns between code and communication. The imposed architecture on distant-participants creates a more constrained collaborative environment than for those who are co-located. This visualization attempts to help reveal how developers use communication in their coding practices. Such information might expose how knowledge transfer takes place.

Open Sources is a first step towards creating tools to understand and help facilitate decentralized collaboration by focusing on communication.

2 MOTIVATION

One set of questions we’d like the tool to answer relate to the activity patterns and associated consequences. Are there patterns that are harbingers of success? Would that be a characteristic signature of communication patterns from the right people, or is it about control of the code? Are there voids of communication before large commits, or does collaboration occur to the end? Do the individuals who control the code collaborate less and develop more independently, or are they productive because they draw on a lot of feedback?

Another set of questions is oriented towards developers. Without a centralized company, there is no orientation when joining a project. It is useful to know who used to be involved and who still is. Project leaders might want to monitor individuals and compare contributions to both code and communication.

Since email is the prevailing method of communication, our efforts are focused on using public mailing lists. They provide enough richness in their metadata to give us a high-level idea of the activity space without needing to look at their contents. This depends on all messages being public; however often discussion can move into the private reply space [15]. Thus our data might not be a complete record of all collaboration, but we believe the mailing lists represent enough of the supporting structure to be useful.

*e-mail: azinman@media.mit.edu
†e-mail: judith@media.mit.edu
Augur [4, 5] is a project developed by Jon Froehlich and Paul Dourish to help visualize and understand the structure of code in a repository. The target uses are to monitor the activity as an aid for distributed developing environments and to understand a project’s history. Thus it was created to help programmers within a project from a code-centric perspective. All data is polled from central versioning repositories such as CVS or Subversion. The data includes the actual code, the author, time of commit, and an annotation on the commit. The data is visualized using Seesoft-like views where files are zoomed out from their source view and placed adjacent to other files [2]. Seesoft views often use color to visually filter the data. No communication is monitored outside of the annotation, so the view only helps the user understand how segments of modules changed overtime.

Augur cannot allow us to understand the dynamic nature of a group through its interaction outside of commits. For example, Augur has a view to create a social network amongst developers, however it only works under the assumption that collaboration is signified by temporally neighboring commits. While this may be true some of the time, it certainly does not deal with coincidences nor demonstrate the level and type of collaboration.

History flow [16], developed by Fernanda Viégas and Martin Wattenberg at IBM, was a major inspiration for the design of Open Sources’ code visualizing algorithm. History flow is a tool to visualize the content change over time of the Wikis. It allows users to understand how pages are created, structured, restructured, vandalized, and occurrences of conflict. Color-coded segments of text move across time in the visualization to their respective locations on the topic page. It allows the user to estimate at a glance the collaborative activities and find interesting moments. Since the topics are not discussed in a typical public forum as open source projects, collaborative messages are built into the content and commit annotations, forcing exploration to understand the rational behind the changes. Thus the tool gives users a powerful perspective to look at the changes throughout time, but cannot provide more information on the communication at higher-level glance.

Ben Fry’s revisionist [6] also provided design inspiration. It is a visualization of how a source code file’s evolves over time. Each version is depicted vertically as raw text in a zoomed out view. Each revision uses smooth lines to connect it to the next version to demonstrate changes over time. It works well in giving an impression about code retention and size are manipulated over time. However, it does not provide any context as to the motives behind the changes nor the responsible actors, nor does it provide global (multi-file) views.

StatCVS [14] is an open source project that generates multiple kinds of visualizations to explore CVS archives. Those involved are more concerned with contribution instances and less with who owns the code at any one point. For this reason, StatCVS only tracks additions from a user rather than replacement or deletion of code. It is meant more as a tool for community members to understand who have been key players and which modules have undergone the most development, rather than looking at how collaboration is created and reshaped over time. StatCVS is one of an increasingly many different projects that look at the history within CVS repositories. Such growth demonstrates the need for developers to access better tools.
4 DESIGN AND IMPLEMENTATION

4.1 Design

Strategies that use Seesoft-derived views localize the search space onto the code [2, 4, 5, 8, 9, 13], making global observations difficult to see and filter against. These views suit applications where spatial relative positioning can help reveal insight into the data, especially when particular modules or files are the point of interest. Open Sources is concerned with higher-level changes across time, such as fluctuations of activity. This would allow us to understand the trends and emergent behavior that remain hidden in local views. Using individual representations of the files is not suited for this activity as it makes the information difficult to compare. When spatial icons are disconnected and spread out, users are required to mentally select, estimate, and add features from the dataset to understand global trends. This sporadic volume estimation is significantly more difficult cognitively than length estimation [10]. Thus, we choose a technique that affords easier referencing when contrasting individual data sets by not separating out the individual files and instead focus on the developers.

We chose a layered approach to represent global contribution of code. This approach was chosen instead of a similar view from StatCVS because of the differing goals of Open Sources (figure 4). StatCVS’s view would be appropriate for comparing individual developers to each other only in terms of code contribution. It is difficult to overload the graph with addition data like communication since it only uses lines that overlap. Furthermore, because StatCVS’s visualization is not stacked, estimating an individual against a project of many developers becomes much more difficult. By taking a stacked approach we can be sure that each developer has a chance to be legible (depending on the scale) and will not be obstructed.

One problem of stacking users is that one contribution often reads as many when the above developers are also pushed up. We present an option that the user can toggle to emphasize the true contributor with two gestalt effects. The first is a more subtle effect that where we create a colored shadow for the originating curve. Red shadows represent code loss while green represents net gain. The second optional technique is to create thick white line over the curve in addition to the shadow. The resulting image adds legibility when exploring activity trends globally over time.

4.2 Implementation

The main interface shows a series of stacked colored horizontal layers across time (x-axis). Each bar represents a developer by a color that is locally unique. The height of a layer corresponds to the amount of a contributor’s code in the repository at that time. An individual’s cardinality in the stack refers to the order in which they submitted code relative to the group. Each bar is assigned a color from a small repeating palette artificially to help differentiate between individuals stacked together. Each color in the palette was chosen to have the same perceived luminosity as the rest in order to keep individuals from appearing to be more or less important for artificial reasons.

On the left of the interface is a listing of the developers to further explore the dataset. Below the user list are a set of buttons that allow the user to filter the dataset by selected individuals. One mode allows selected users to be highlighted by creating empty space between themselves and surrounding layers (Figure 7). Another mode only draws the layers of the selected individuals (Figure 8). Such a mode is useful for comparing individuals.

Communication events are depicted as small circles. Because we are looking at emails to a mailing list as activity indicators, further metadata is not necessary. They are first shown in the middle of the layer that corresponds to the sender.
Only showing the communication data in the authors layer was not sufficient to our goals. First, it unnecessarily increases cognitive complexity when trying to mentally create a global reference if there is a lot of activity. Second, depending on the scale, a participant with little code might not have visible messages even though they may support a lot of collaboration. Third, individuals who communicated before committing any code would be artificially filtered.

We decided to include a complementary area above the graph where all communication incidents are also represented. This technique gives the user a linear representation across time that can be visually filtered by manipulating colors to highlight particular individuals. If a short period of time has multiple messages, we avoid overlap by stacking the quantized messages vertically. When in a mode that highlights selected individuals, the non-selected author’s communications are made transparent to preserve the global reference but give less emphasis (see figure x). Should the stack reach the edges of its container, all the dots scale down to ensure a proper fit. Thus by looking at the height and density of a vertical strip users can easily spot and estimate communicative activity.

5 EXAMPLES

We present an informal analysis of several different open source projects to give an idea of what Open Sources is capable of revealing and how users can interact with the dataset. Each project examined has existed for many years and respectively serve different target audiences and functions.

Project A is a stable project that has faced widespread adaptation amongst the open source community (Figure 9). It should be noted that the first commit is large because it took an existing code base, written by both the committer as well as others, and moved it to a public CVS repository when becoming a public project. It is interesting to note that even though not all of the initial code (in purple) was written by the committer, the author continued to develop to almost doubling his contribution two years later. Furthermore, the frequency of activity dramatically increases with each addition of a new person, even though the contributions from the newcomers are dwarfed by comparison in size. This is especially highlighted at the end of the timeframe with the addition of even more developers.

Project B has been active for approximately a decade. This project makes heavy use of mailing lists which can be seen from two years worth of data in Open Sources (Figure 10). The visualization illustrates that an individual’s level of discussion is not globally proportional to their amount of code. On the bottom layer in purple, we see very infrequent (several per month) communication instances that seem to be clustered. This developer maintains a roughly constant amount despite all other additions and changes, indicating a key role. The next two bottom layers, by contrast, have significantly more communication than code. The next layer above, the second largest contributor later in time, lacks communication markers because we could not find a correlation between email and CVS name. Perhaps the smaller contributors need to communicate to the group more because of their distance from the knowledge behind the code base. What is puzzling is the surge of development and new users given the even larger development of competing projects. This might have been caused by a reaction by users to become developers to ensure compatibility and matching of features.

Project C portrays a very different developmental practice than the projects A and B (Figure 11). This might be attributed to the types of potential developers. Projects A and B are more general desktop type applications whereas project C is a programmable...
scientific tool. Because users need to know how to program the tool to use it, they are more likely to contribute their work back to the community. Note that communication archives were not available until later dates. However, within the available range the majority of communications come from late-starting developers with small contributions. This could indicate that the bottom two developers made their initial contribution and then left the project, which would explain their relative stability and minor activity.

6 DISCUSSION

Open Sources provides a view of the junction between content and communication in a simple fashion. Code plays too big a role in open source communities to not give it enough weight when analyzing computer-mediated collaboration. Garton notes, “People linked to people and groups are not the only sources of network data”; by employing other types of structural arrangement the “data reveals cognitive maps and identifies people who hold similar conceptual orientations” [7]. Thus the cognitive map of the open source developer deals with what’s relevant to them: people and code that directly affects their code. The social ties they need to retain and be aware of can be a sub-set of the larger projects. The tool’s high-level perspective allows a participant to reflect on the actual interaction space versus what is subjectively perceived, giving a better sense of roles within the group.

An informal evaluation was conducted over email with an existing open source community. They noted that Open Sources succeeded in finding a story that was otherwise hidden: “He was very active when the project started, but then stopped contributing, and his share of the code base gradually declined as parts were rewritten and changed. But still, there seems to be surprisingly much of his code left."

Many users suggested that Open Sources might benefit from a 3D perspective. One example is sv3D, a visualization that puts Seesoft-like views into a 3D space [13]. They attempt to use filters to hide most occluding data, but can never achieve their goal when the non-filtered data can occlude itself. This is inherent in most 3D visualizations, since they rely on the user’s ability to move around the space. However, the likelihood of finding the canonical image that best represents the data is diminished. One obvious way of extended Open Sources also incurs this penalty. It would be to rotate the graph 90 degrees into the monitor and thicken the separating lines, such that layers would appear almost like roller coasters. Aside from occlusion, the work to use global references is significantly increased, as the user now needs to mentally store and process each individual layer instead of chunking it into one.

With this technique, the movement of unrelated layers from activity underneath can be a drawback. We have employed techniques such as shading and curve emphasis to help draw the eye to the sources of change. However, additional methods can help combat this problem while maintaining the advantages of the stacked view. It was suggested that by altering the color of a layer we could show age, as in History Flow. One such way is to fade a layer towards transparent after each commit, regaining full opacity upon the next commit. Thus the most active layers would be the most visible while still retaining the global reference. Another method might be to use transparency to show age of individual lines of code. By locking each horizontal line of a layer to the code it represents, we could see where activity is strongest, the lifetime between developers, and fully realize commits that net small amounts but gain and lose a lot.

Open Sources currently looks at all the code within a repository. Future versions will look at using other methods, similar to Augur, to collapse the information space to aid more targeted searching.

7 FUTURE WORK

We created a method of zooming to preserve our extra markers, but more can be done. Within the zoomed view, further options might allow the user to automatically filter out people whose neither code nor communications appear, helping reduce unnecessary clutter. Such filter might also be applied by selecting regions in time rather than rectangles from the rendered view.

To enable contextual relevance, additional views could highlight communication between individuals of choice in a Loom2 [1] style fashion. Such a view would move into the more traditional roles of social network analysis, such as examining how ties and relations change overtime given their communications environment [3, 7].

By augmenting this visualization with other interactive tools, users could explore the giant data space more manageable and effectively. Open Sources is not intended to be the only tool, but a foundational visualization from which to bring in other filters
and views. For example, we currently only show instances of communication, but not the details such messages contain. By using other tools, we could allow users to explore a set of email and their links to code after selecting the interesting region in Open Sources. Another tool might integrate into an IDE, providing access to relevant discussion surrounding highlighted code. We would like to use Open Sources to actively investigate several open source communities. In particular, we wish to look at the differences between successful projects, failed projects, and projects that started successfully but then failed. We hypothesize that consequential events will leave fingerprints of activity that can be used to predict future success.

Finally, we would like to investigate new communicative interfaces and methods that help annotate, negotiate, summarize, and inform their information that is created with Computer-Mediated Communication. We encourage the active development of any tool that helps decentralized environments to lower the costs of communication and knowledge transfer.

8 CONCLUSION

Open Sources is a tool that allows researchers, developers, and managers to visualize the relationship between code and communication. The visualization affords the deduction of key players, activity changes over time, key players, and the communication space. Novel techniques to isolate activity within an existing visual framework allow users to quickly discriminate regions of interest. An informal evaluation has shown the tool to be useful in understanding a community’s hidden history, while providing suggestions for further enhancement.

9 ACKNOWLEDGMENTS

We would like to thank the MIT Media Lab for their wonderful resources, the Sociable Media Group for tremendous insight, and StatCVS’s Richard Cyganiak for his valuable time and criticism.

REFERENCES

[16] Fernanda Viégas, Martin Wattenberg, and Kushal Dave, Studying Cooperation and Conflict between Authors with history flow Visualizations, CHI 2004. Vienna, Austria. 24-29, April, 2004