0.1 Background

In my empirical analysis, I examine how changes in the cost of knowledge acquisition and communication across the hierarchy affect hierarchical structure and the work of OSS contributors.

1. GitHub Actions
2. Contributor Turnover
3. Issue & Pull Request Templates
4. Codeowners file (perhaps later?)

Focus first on what each of these does and then think about how it fits into the hierarchy. For each of the "predicted empirical effects" section highlight both my model's predictions and Garicano's predictions

0.2 GitHub Actions

0.2.1 Summary

GitHub Actions, released in November 2019 (October 2018 beta) is "a new way for developers to automate workflows directly from their repositories". Notably, it offers powerful CI/CD tools, which automate several aspects of the software development pipeline and help increase the rate of software release by packaging software improvements into smaller increments. Github Actions fall into a variety of categories; below, I list some example categories of GitHub Actions

- Utilities: Makes it easier to use various Github Actions in your pipeline
- CI & Deployment: Provides developers with pipieline for software deployment
- Code Quality & Code Review: Provides reports and information on aspects where the code should be improved
- Publishing: Helps automate part of the release cycle of software
- Open Source Management: Automates communication and management of issues

GitHub Actions has heterogeneous effects on the hierarchy, depending on the action category. Actions like code quality, code review, open source management which interact with lower ranked contributors reduce communication costs. For example, code

quality and review actions highlight improvements automatically for developers, which; this saves time because they no longer have to comb through the file line by line, checking for style. CI & Deployment, Publishing & Utilities reduce knowledge acquisition costs for higher ranked OSS contributors. For example, developers also don't have to learn about each project's custom software publishing/deployment processes because a developer will have integrated a GitHub Action that handles that for them into the pipeline.

0.2.2 Adoption and Treatment Notes

1. There are a few different initial adoption dates to consider and there may be underlying unobservable characteristics distinguishing projects that adopt GA during beta vs. at different points after release

2. One way to measure adoption is when the project first creates a '.yaml'/'.yml' file.

3. There is heterogeneity in the initial treatment value because different projects adopt different GitHub Actions to begin with, so there are many possible treatment values.

4. Projects add to their cache of GitHub Actions over times, which means that the treatment value changes over time.

5. One possible solution is to define treatment based off the categories of the GitHub Actions they've adopted.

6. The analysis is complicated because different categories have heterogeneous effects

7. Note also that the previous popular CI tool, Travis CI, discontinued open source support in 2020 (https://blog.travis-ci.com/2020-11-02-travis-ci-new-billing). This may have caused an increase in the number of Github Actions users and might be relevant for treatment timing.

Existing studies don't consider changes in the treatment value, defend the RDD assumption, explore the impact on individualized work as opposed to project-wide aggregations, or analyze the aggregate project-wide effects as opposed to separate effects for merged or unmerged PRs. Moreover, no study considers the impact of GitHub Actions in a hierarchical context. However, GitHub Actions are more recent which makes it easier to analyze and more relevant.

0.2.3 Predicted Empirical Effects

WIP

0.2.4 Literature

WIP

0.3 Misc

In all honesty, I'm not sure I'm that invested into analyzing GitHub Actions. The complexities of treatment values/adoption and the fact that it affects so many aspetcs of the hierarchy simultaneously makes it difficult. I think GitHub Actions is definitely interesting to analyze but maybe not under the hierarchies framework. We shall see though.

0.4 Knowledge Turnover

One of the major worries with OSS production is the reliance of projects on a few contributors, who, without a contractual obligation of a project, may disappear at a moments notice and bring with them the knowledge and skills necessary for key components of, or in some cases, the whole project, to function. One way to measure variation in the cost of knowledge acquisition in projects is to exploit OSS developer turnover. When a key contributor for an OSS project leaves, it is much harder for new contributors to acquire knowledge because a source of information is gone.

0.4.1 Adoption and Treatment

1. There's not really an "adoption" timeline.

2. We can consider variation in turnover by examining the "truck" factor of projects, which measures the number of key members. There are also variants to this measure, such as looking at the identity of the person who wrote the last line of code, as opposed to total lines written per person.

3. It will also be important to consider the presence of knowledge preservation tools. The impact of contributors leaving will be lessened by the presence of good documentation. For example, if a departing contributor wrote a lot of documentation count, that means that they probably did preserve their knowledge for future contributors so their departure will not impact knowledge acquisition by as much.

4. I can use departures as either an indicator for a decrease in knowledge acquisition costs or as the endogenous variable that proxies for an increase in knowledge acquisition costs.

0.4.2 Predited Empirical Effects

Do we observe increases in knowledge? Although knowledge acquisition costs and knowledge acquisition is unobserved, we can proxy for increases in knowledge by the % of problems write rank contributors need to solve/correct. Example statistics include

- % of PRs that are merged (higher knowledge = more merged PRs)
- % of comments/reviews per PR (higher knowledge = less review comments needed)
- After PR is opened, number of commits that are made (higher knowledge = less correction commits needed)
- Controlling for PR size, % of commits by PR reviewer (higher knowledge = less correction commits by reviewer needed)
- Controlling for PR size, time it takes to wrap up a PR. (higher knowledge = less time to wrap up a PR). Note that since my shock involves exploiting exogenous variation in reviewers, we'll obviously see higher review times regardless of changes in knowledge. Two ideas to account for this are to subtract "PR reviewer assignment time" or control for the # of active potential reviewers.
- Classify issues as bugs, enhancements and questions - then look at the % of linked PRs that are about enhancements (higher knowledge = more enhancements, as enhancements are more challenging than bugs to implement)

0.4.3 Literature

Rashid, Clarke, and O'Connor 2017 has a fairly good literature review on knowledge loss in OSS. Nassif and Robillard 2017 provides interesting comparisons about different statistics for measuring knowledge loss
Some papers that I'm hoping to read are

1. https://ieeexplore.ieee.org/abstract/document/8870181?casa_token=1rqBePn8XnoAAAAA:wCul
CNViaN8n-_20B3PaSKabeFN2vp5ZX

2. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4755634 (need to cite)

3. https://doras.dcu.ie/29119/1/RashidIJIMPaper-RevisionFollowingPeerReview.pdf (need to cite)

4. Managing knowledge assets for open innovation: a systematic literature review (need to cite)

5. Exploring the Foundations of Cumulative Innovation: Implications for Organization Science (need to cite)

0.5 Issue & Pull Request Templates

0.5.1 Summary

In February of 2016 (https://github.blog/?s=issue+templates), GitHub released issue and pull request templates. GitHub stated that "now project maintainers can add templates for Issues and Pull Requests to projects, helping contributors share the right details at the start of a thread". This is important because it ensures that OSS contributors have completed required checklist items before opening an issue/PR, and also makes it easier for reviewers to identify to find particular key pieces of information.

0.5.2 Adoption and Treatment Notes

I can measure initial adoption of PR and Issue templates by looking at when an ISSUE_TEMPLATE (optional ".md" extension?) file or PULL_REQUEST_TEMPLATE (optional ".md" extension?) was created. I can compare projects that adopted these templates during similar timeframes because while the particular timing of adoption might be adoption, adoption within a certain timerange tells us a lot about the attitudes of the OSS contributors towards improving their projects. In May 2018, users acquired the option to choose a template from (presumably) a variety of options when making a new issue (https://github.blog/changelog/2018-05-02-mutiple-template-choice/). However, while this implies that there may be multiple treatments, this is still easier to analyze because there are less possible combinations (less templates) and adoption is more spread out throughout time.

0.5.3 Predicted Empirical Effects

WIP

0.5.4 Literature

need to cite all and read An Empirical Analysis of Issue Templates Usage in Large-Scale Projects on GitHub Consistent or not? An investigation of using Pull Request Template in GitHub To Follow or Not to Follow: Understanding Issue/Pull-Request Templates on GitHub

barriers to contributing: how to contribute - see harvard pdf