Code Insights

Summer 2017

Goals

- What are we trying to build?
- Will it be useful?
- What’s the competition?
- What are technical challenges?
- What’s the timeline?

What are we trying to build?

- Code Metrics tool *that analyzes individual's contributions to a codebase.*
- Will help identify:
    - For each developer:
        - What's the breakdown of their contributions between:
            - Actual contributions to product that's being shipped
            - Documentation
            - Testing
            - Other metrics to attempt to gauge quality and complexity of code

Will it be useful?

"Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs."- Bill Gates

- Not trying to answer questions about the codebase, but the team that's developing the codebase.
- Will be useful in bringing about large change to a repository. (Avoid a full rewrite).
    - We want to go from 0% -> >50% test coverage, but [Developer B] has still not written a single test case for his code.

What’s the competition?

- Upsource - https://www.jetbrains.com/upsource/features/codeinsight.html
    - Jetbrains makes amazing tools, loved by developers
- Clover - https://confluence.atlassian.com/clover/clover-for-idea-154633611.html
    - Atlassian is a well established and trusted by large companies
- Others - https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#Java
    - **None of them use version control to measure individual’s contributions.**
    - **Most of them are not open source.**
    - **None of them are web-based (ease of usage).**

Technical Challenges

Do this in a general way that so people can use it with

- All languages
- All testing frameworks
- All build systems
- Config file?
- Build system plugin?
- IDE plugin?

Technical Challenges

Things should be as simple as possible, **but** complex problems will likely have complex solutions.

- Smarter team members should probably be working on the more complex problems.
- They shouldn’t be punished because they’re code is simply more complex than everyone else’s.
- We’re trying to find people who are **writing complex code for simple problems**.
- @Complex documentation tag (something agreed upon by team)?

Technical Challenges

Noise reduction

- We want to incentivize “writing documentation” not “writing a lot of documentation”.
- How do we measure “the right amount” of documentation?
- Point system?
- Comparison to average?
- Preset value?

Technical Challenges

- Do we care about people trying to cheat the system?
- Heisenberg effect: Cannot measure something without changing it.
  - Management strategy: Make tool publicly available to team. Let people adapt their coding style, catch bad practices left over during code reviews.
  - Management strategy: Keep tool private, use it to find critical points of failure in team. Try to improve those points.
- Answer should become clear by simply trying to use the tool internally in different situations.

Timeline

Phase 1 (End of June): Be able to answer, **on an individual basis:**

- Unit Test Coverage (Who wrote most/least)
- Documentation Coverage (Who wrote most/least)
- Cyclomatic Complexity (Who wrote the simplest/most complex code)

Phase 2 (End of July): Start testing internally and tweaking.

Phase 3 (End of August): Start thinking about generalizing.

Phase 4 (Onwards): Start thinking about more ambitious features.