A Nonprofit's Google Summer of Code Experience in the Age of AI-Assisted Contributions

Authors

William Gearty <wbgearty@syr.edu>, Open Source Program Office, Syracuse University, 0000-0003-0076-3262

Keywords

Open source software
Google Summer of Code
Code review
GitHub Actions
Software maintenance
AI-assisted code generation

Abstract

Learning Unlimited (LU), a national nonprofit empowering college students to deliver educational programs, maintains several open-source codebases (https://github.com/learning-unlimited), which I and three other volunteers maintain. Our limited nonprofit budget and unpaid volunteer workforce mean we have limited time and/or money to put towards further development of these codebases, despite issue reports and feature requests perpetually piling up. In an attempt to give our software a jumpstart, we participated in Google Summer of Code (GSoC), a global paid internship program that pairs beginner contributors with experienced mentors to work on real-world open-source projects. Our hope was to attract new contributors to our open-source program-management platform to squash bugs and work on larger new features that we didn't have the capacity to work on ourselves. Over roughly three months, the project received nearly 1,200 pull requests and 560 issues from 388 GSoC applicants competing for three internship slots. The lack of broader real-world context and characteristic AI errors made it evident that the vast majority of these contributions were generated with substantial AI assistance. This application period simultaneously became LU's most productive and most stressful: GSoC participants closed 140 backlogged issues, some over fourteen years old, while we were forced to triage an unprecedented volume of low-quality, AI-assisted work. Average weekly maintainer activity rose roughly thirtyfold during the contribution period, peaking at over 700 discrete actions in a single week, and more than half of the pull requests that were reviewed were closed without merging. To keep pace, we deployed a set of GitHub Actions automation to assist with assignment enforcement, contributor rate-limiting, triage gates, and activity tracking, reducing required maintainer activity by more than 60%. Further, to combat increasing time-to-review we began implementing AI-assisted code review via integrated GitHub Copilot and external Claude conversations for first-pass review. This appeared to keep some contributors engaged, while others became unresponsive to the AI reviews. Overall, this natural experiment reveals that AI has dramatically lowered the cost of producing code while increasing the human cost of properly evaluating it, with consequences for how RSE teams should structure contributor onboarding, code review, and sustainability practices. Our experience also amplifies concerns about maintainer burnout, especially when the marginal cost of unwanted contributions has approached zero. Finally, in addition to our three interns, a small number of unsuccessful applicants have remained active contributors past the application period, suggesting that the underlying value of programs like GSoC survives even under these new conditions, if maintainers can withstand the volume.

Connection to Mission, Goals, & Interests of US-RSE Community

The pressures documented here (high-volume AI-assisted contributions, the inversion of authoring versus reviewing effort, the need for automated triage to preserve maintainer capacity) apply directly to all research software projects that accept external contributions or operate within larger open communities. The talk shares quantitative evidence and concrete strategies that are immediately portable to RSE practice: automation that can reduce maintenance load, how to identify and characterize AI-generated contributions, and how to balance openness with sustainability when contribution volume is no longer a meaningful proxy for community engagement. The experience also raises questions of direct interest to the US-RSE community about how AI is changing the day-to-day work of research software development, particularly the relative effort balance between writing and reviewing code, and what institutional supports (workflow tooling, mentorship structures, recognition systems) may need to evolve in response.