

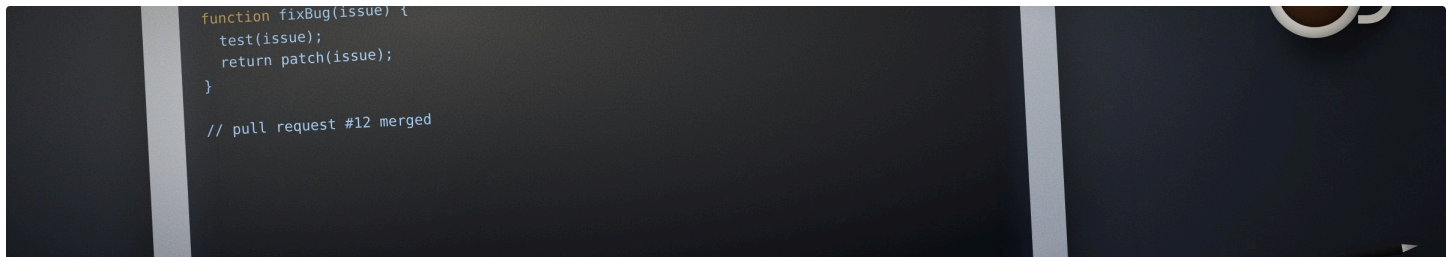
ADMISSIONS STRATEGY

The Art of the Open-Source Contribution: Proving Software Skills Without a Paid Internship



By **Alex Gomelskiy**
 Founder and Director of Admissions Strategy, Scholar Coach, Inc.

October 20, 2026



A student can build an application that works perfectly on their laptop and still have no experience maintaining software alongside other developers. They may have written thousands of lines of code, completed several programming courses, or created an impressive personal project. But how does anyone outside their immediate circle know whether the code is understandable, testable, maintainable, or useful to someone besides its creator?

Open-source software offers one answer. By contributing to an existing project, students can move beyond building something independently and learn to work within a codebase that other people depend on.

For high school students interested in computer science, this can be a meaningful alternative to pursuing a paid internship they may not yet qualify for. The objective, however, is not to accumulate GitHub activity or collect merged pull requests. It is to demonstrate that you can understand an existing problem, make a useful change, respond to technical feedback, and contribute responsibly to a shared project.

1. WHY CONTRIBUTING TO AN EXISTING PROJECT CHANGES THE CHALLENGE

A personal programming project gives students substantial freedom. They choose the features, organize the files, establish the coding conventions, and decide when the work is finished. That freedom is valuable: independent projects can demonstrate creativity, technical curiosity, and persistence.

Contributing to an established project introduces different constraints. The student must understand someone else's design decisions, follow existing conventions, identify the cause of a problem, and avoid breaking functionality that already works.

GitHub's official documentation describes the typical contribution process: understand a repository's guidelines, identify an appropriate issue, make changes in a separate branch, submit a pull request, and work with maintainers through review.¹

Each step presents a different opportunity to demonstrate competence. A student might write a test for a previously untested edge case, correct a reproducible bug, improve an installation guide, or implement a narrowly defined feature. The value comes from addressing a genuine need within the project's constraints.

This does not make open-source contributions inherently superior to coding clubs, hackathons, internships, or independent projects. Each can provide useful evidence of ability. The distinction is that a well-executed external contribution can show how a student works within an existing technical environment rather than only within a system they designed themselves.

For a selective college application, that distinction can make a computer science activity more concrete.

Want a plan built around **your** student?

Book a free consultation. The short intake takes about a minute and opens right away.

[BOOK NOW →](#)

A common mistake is to begin with the most famous repository a student recognizes. A large, widely used project may seem attractive on an application, but popularity does not guarantee that it welcomes beginners, has manageable issues, or can provide timely feedback.

2. FIND A PROJECT THAT NEEDS YOUR SKILLS—NOT ONE THAT LOOKS IMPRESSIVE

Students should instead look for three characteristics:

A real, understandable problem. The issue should describe something that needs fixing or improving, ideally with steps to reproduce the problem or a clear definition of the expected result.

An active, navigable project. Look for installation instructions, contribution guidelines, a license, and recent evidence that maintainers review and respond to contributions.

A manageable starting point. The task should fit the student's current skills while requiring enough thought to produce meaningful work.

GitHub recommends searching for issues labeled good first issue or help wanted and reading the project's contribution guidelines before beginning.² These labels can help identify potential entry points, although they do not guarantee that an issue remains available or that a pull request will be accepted.

Students can begin with projects related to programming languages they already know. A Python student might explore a scientific-computing library, a JavaScript student might investigate a developer tool, and a student interested in accessibility might look for a project where interface behavior or documentation needs improvement.

The best first project is not necessarily the one with the most stars. It is the one where the student can understand the problem, make a defensible contribution, and learn from the review process.

3. START SMALL, BUT SOLVE A REAL PROBLEM

Students sometimes dismiss documentation fixes, test improvements, or small bug fixes because they seem less impressive than building a major feature. That is a mistake.

A narrowly scoped change can demonstrate careful reasoning if it addresses a legitimate problem. A test that catches a previously missed failure, for example, may be

more useful than a large feature that duplicates existing functionality.

A practical first contribution might follow this sequence:

Read before coding. Study the README, contribution instructions, relevant source files, and existing tests. Understand what the software is intended to do.

Investigate the issue. Reproduce the problem if possible. Read related discussions and check whether another contributor is already working on it.

Confirm the scope. When the intended solution is unclear, ask a concise question in the project's preferred discussion channel before investing hours in an incompatible approach.

Make a focused change. Avoid unrelated refactoring or unnecessary features. Keep the proposed solution narrow enough for a reviewer to understand.

Test the result. Run the relevant tests and document any limitations or checks that could not be completed.

Explain the contribution. Submit a pull request describing the problem, the proposed fix, the reasoning behind it, and the verification performed.

The goal is not merely to submit code. It is to demonstrate a repeatable engineering process.

4. TREAT CODE REVIEW AS PART OF THE ACCOMPLISHMENT

A pull request is a proposal, not a certificate of excellence. Maintainers may ask for different naming conventions, additional tests, a simpler implementation, or a change in approach. They may also reject a technically functional solution because it does not fit the project's design or priorities.

These outcomes are part of collaborative software development. GitHub's guidance encourages contributors to respond professionally to feedback, revise the existing pull request when changes are requested, and seek clarification when necessary.¹

For students, this process can provide useful evidence of learning and collaboration. Consider a hypothetical example: a student submits a fix for a data-processing bug. A maintainer points out that the fix handles the reported example but fails when the input is empty. The student adds a regression test, revises the implementation, and

Want a plan built around **your** student?

Book a free consultation. The short intake takes about a minute and opens right away.

BOOK NOW →

explains why the revised behavior is correct.

The stronger story is not simply that the pull request was merged. It is that the student learned to account for an edge case, incorporated feedback, and improved the solution.

Not every contribution will be accepted. A project may change priorities, an issue may become obsolete, or a proposed approach may not fit the codebase. An unmerged contribution can still represent meaningful work, but the student should distinguish between work proposed, feedback received, and changes actually incorporated into the project.

Accuracy matters more than presenting every attempt as a success.

5. TURN GITHUB ACTIVITY INTO EVIDENCE A REVIEWER CAN UNDERSTAND

A public profile can provide supporting evidence, but the number of repositories, commits, stars, or pull requests does not establish software proficiency by itself. Activity counts measure different things, and some contributions require far more technical judgment than others.

Students should make their strongest work easy to evaluate. For one or two substantial contributions, preserve:

- A link to the original issue and the problem it described.
- The pull request and its final status.
- A concise explanation of the student's specific contribution.
- The tests or other checks used to validate the change.
- Relevant review comments and the revisions made in response.
- A clear account of any collaboration, mentor assistance, or AI tools used.

A merged pull request provides evidence that maintainers accepted a proposed change into a particular project. It does not prove that the student independently wrote every line, mastered the entire codebase, or worked at a professional engineer's level.

The repository's context matters, too. A small documentation correction and a complex performance improvement are different accomplishments, even if both

were merged. Students should describe the actual scope of their work rather than relying on the reputation of the project to imply more than they did.

If asked about the contribution in an interview or essay, the student should be able to explain the original problem, the relevant code, the alternatives considered, and the reason the final solution works. That explanation is a useful check on whether the experience reflects genuine understanding.

6. USE OPEN SOURCE TO DEEPEN AN EXISTING TECHNICAL INTEREST

Open source works best when it connects to something the student genuinely wants to understand. A student interested in machine learning might investigate a data-processing tool. Someone drawn to cybersecurity might study a defensive security project, respecting its authorization and responsible-disclosure rules. A student interested in language technology might contribute to text-processing software or a documentation project used by developers.

This connection can give the activity direction without requiring an elaborate personal brand. One carefully chosen contribution, followed by further learning or a second related contribution, may reveal more about a student's interests than a long list of disconnected repositories.

Students should also respect project licenses, codes of conduct, security policies, and privacy requirements. They should never submit changes that expose secrets, publish private information, or test systems without permission.

Finally, open source should not become another expensive admissions product. Public repositories and free contribution workflows make participation possible without a paid internship, although students may still need access to a computer, reliable internet, and time to learn. They should not pay a program simply to obtain a promised contribution or an impressive-sounding credential.

HOW SCHOLAR COACH HELPS STUDENTS BUILD A CREDIBLE TECHNICAL PORTFOLIO

At Scholar Coach, we help students connect technical work to a coherent academic profile without treating every activity as an admissions formula.

Want a plan built around **your** student?

Book a free consultation. The short intake takes about a minute and opens right away.

BOOK NOW →

Step 1: Identify the student's actual interests and skills. We determine which languages, tools, and subject areas the student understands well enough to contribute meaningfully.

Step 2: Evaluate potential projects. We help students assess whether a repository has clear documentation, an appropriate entry point, and a genuine problem that fits their current abilities.

Step 3: Define a realistic contribution goal. Rather than chasing a famous repository or an arbitrary number of pull requests, we establish a focused objective with a clear technical purpose.

Step 4: Document the work honestly. We help students organize issue links, pull requests, testing evidence, design decisions, and explanations of their individual contributions.

Step 5: Connect the experience to the broader application. We determine whether the contribution belongs in the activities list, a technical portfolio, or an essay where the

student's reasoning and development can be explained in greater depth.

We do not promise that a particular repository, contribution, or merged pull request will improve an admissions outcome. The aim is to help students produce real work they understand and can explain.

A paid internship is one way to gain technical experience, but it is not the only way. By contributing thoughtfully to software that other people maintain and use, a high school student can demonstrate initiative, technical reasoning, and a willingness to learn from criticism. The strongest evidence is not the prestige of the repository. It is the quality of the contribution—and the student's ability to explain why it matters.

SOURCES

¹ GitHub Docs — Contributing to a Project

² GitHub Docs — Contributing to Open Source

³ MIT Admissions — Extracurricular Activities

⁴ MIT Admissions — What We Look For

Want a plan built around **your** student?

Book a free consultation. The short intake takes about a minute and opens right away.

BOOK NOW →