



FLOSS
MANUALS

Table of Contents

What is Google Summer of Code?	1
Goals of the Program.....	1
A Brief History of Google Summer of Code.....	1
Participant Roles.....	1
Program Structure.....	2
Why GSoC Matters	5
About This Manual	7
How to Contribute to This Manual.....	7
About the Authors.....	7
Facilitation.....	9
What Makes a Good Mentor?	10
Be Prepared to Seek Help.....	10
Backup Mentors.....	10
What To Expect From Under-Mentoring.....	10
Defining a Project	12
Selecting a Student	14
Understanding Student Motivations.....	14
Selecting a Student.....	14
Finding a Match.....	15
Google's Selection Process.....	15
Starting at the Beginning.....	15
Community Basics	17
Set the Tone.....	17
Encourage Questions.....	17
Criticism Happens.....	17
Best Practices	18
Giving and Receiving Criticism.....	19
Warning Signs	21
Actions Speak Louder than Words.....	21
Open Source Culture	23
Openness and Sharing.....	23
Remote Communication.....	23
Abbreviations and Slang.....	24
Volunteerism and Gift Economies.....	24
Mind the Gap	26
Setting Expectations	27
Defining Success.....	27
Managing Output.....	27
Workflow	28

Table of Contents

Managing the Plan.....	29
Creating a Strategic Plan.....	29
Following the Plan.....	29
Delivering Feedback.....	30
Evaluations.....	31
They Won't Know Unless You Tell Them.....	31
When in Doubt, Fail the Student Early.....	32
But It's Not Impossible to Turn Things Around.....	32
Mentor, Heal Thyself.....	33
Upstream Integration.....	34
Recruit Committers Early.....	34
Get the Code into Your Repo.....	34
Dividing Up Patches.....	34
Patch Review.....	35
Building a Lifetime Contributor.....	36
The Quick Guide.....	38
Quick Tips on Effective Mentoring.....	38
The History of GSoC.....	39
Additional Resources.....	40
General Resources.....	40
Mailing Lists.....	40
Books.....	41
Associated Projects.....	41
What to Do When the Unexpected Happens?.....	41
Glossary.....	42
+1.....	42
-1.....	42
Committer.....	42
Community Bonding Period.....	42
DVCS.....	42
FLOSS.....	42
GSoC.....	42
IDE.....	42
IM.....	42
IRC.....	43
ISP.....	43
JFDI.....	43
Mentor.....	43
Organization.....	43
Organization Admin (Org Admin).....	43
Program Administrator.....	43
RTFM.....	43
Secondary Mentor.....	43
SMOP.....	43
Summer.....	43
TDD.....	44
Use Case.....	44

Table of Contents

Glossary	
Waterfall Model.....	44
License.....	45
Authors.....	46

What is Google Summer of Code?

Google Summer of Code (GSoC) is a program that offers college and university student developers stipends to write open source code. Each year Google works with many open source, free software and technology-related groups to identify and fund proposals for student open source projects.

GSoC pairs accepted student applicants with mentors from the participating projects. Accepted students gain exposure to real-world software development scenarios and the opportunity for employment in areas related to their academic pursuits. In turn, the participating organizations are able to more easily identify and bring in new developers. Best of all, more source code is created and released for the use and benefit of all; all code produced as part of the program is released under an Open Source Initiative approved license.

The program has brought together thousands of students and mentors from nearly 100 countries worldwide. At the time of writing, over 200 open source projects from areas as diverse as operating systems and community services have participated as mentoring organizations for the program. Successful students have widely reported that their participation in GSoC made them more attractive to potential employers and that the program has helped greatly when embarking on their technical careers.

Goals of the Program

The program has several goals:

- Get more open source code written and released for the benefit of all
- Inspire young developers to begin participating in open source development
- Help open source projects identify and bring in new developers
- Provide students the opportunity to do work related to their academic pursuits during the summer: "flip bits, not burgers".
- Give students more exposure to real-world software development scenarios (e.g., distributed development and version control, software licensing issues and mailing list etiquette).

A Brief History of Google Summer of Code

Google Summer of Code began in 2005 as a complex experiment with a simple goal: helping students find work related to their academic pursuits during their school holidays. The first year, 40 projects participated and 400 students began the experiment. In 2009, the fifth Google Summer of Code wrapped up to the best results yet. More than 85% of the 1,000 student participants in the program successfully completed their projects. Best of all, most of the organizations participating over the past five years reported that the program helped them find new community members and active committers.

If you are interested in a more extensive history of the program, please see the appendix.

Participant Roles

There are four roles in the Google Summer of Code program:

Program Administrator: Program administrators are employees of Google's Open Source Programs Office who run the program. These folks do a variety of tasks: select the participating open source projects each year, create and analyze the program evaluations, administer the program mailing lists, ensure that participants are paid and send out the all important program t-shirt. Program administrators do not select which student proposals are accepted into Google Summer of Code.

More broadly, program administrators can provide useful advice to both new and seasoned participants in a variety of areas due to their experience with the program and mentoring process. Not sure how to handle a disappearing student? Don't know which mailing list has the latest information on payments? Wondering how to best improve your organization's application for the program? Find a program administrator and ask away!

Organization Administrator: Org admins are the cat herders for an open source project involved in GSoC. These people submit the organization's application to participate in the program to Google, ensure that mentors fill out evaluations in a timely fashion and generally organize their project's participation in GSoC. Org admin acts as Google's go-to person in the event any issues arise. There are also some trivial administrative tasks in the online system for GSoC that can only be completed by organization administrators, all of which are enumerated in the system documentation. Some org admins also mentor students during GSoC, and that's perfectly fine; it is just highly recommended that folks know they have enough time to execute both roles simultaneously.

Org admins are the final authority about matters such as which student projects will be accepted, who will mentor whom. On the social side, should a mentor and student have difficulties communicating or making progress, an org admin will often step in as a neutral party to help the two work together more effectively. Org admins also help track down disappearing participants, whether mentors or students.

Mentor: Mentors are those people from the community who volunteer to work with a student. Mentors provide guidance such as pointers to useful documentation, code reviews, etc. In addition to providing students with feedback and pointers, a mentor acts as an ambassador to help student contributors integrate into their project's community. Some organizations choose to assign more than one mentor to each of their students. Many members of the community provide guidance to their project's GSoC students without mentoring in an "official" capacity, much as they would answer anyone's questions on the project mailing list or IRC channel.

Student: A student participant in GSoC is typically a college or university student, though the only academic requirement is that the accepted applicants be enrolled in an accredited academic institution. Students must also be at least 18 years of age in order to participate. Students come from a variety of academic backgrounds, and though most students are enrolled in a Computer Science program there is no requirement that they be studying CS or IT; past student participants in GSoC have come from disciplines as varied as Ecology, Medicine and Music.

Students submit project proposals to the various organizations participating in GSoC. The organizations select which student proposals they would like to see funded by Google. Successful student participants have often gone on to become committers to the organization they worked with in GSoC. Many students have also gone on to become mentors and even org admins for the program.

Program Structure

All of the program rules are enumerated in the GSoC FAQs each year. Provided all of the rules regarding eligibility for the program are followed, Google takes a fairly hands off approach to GSoC. Each organization will structure their participation in GSoC in whichever way makes the most sense for their technical and community needs.

Organization Applications: The GSoC program is announced each year on the Official Google Blog <http://googleblog.blogspot.com> among other places. Once the program is announced, open source projects know when they can submit applications. Each organization is expected to apply to participate. The questions asked in the organization application are published in advance and linked from the Program FAQ. Organizations usually have one week to apply for the program. Following receipt of applications, Google's program administrators will select which organizations will participate in that year's Google Summer of Code.

Student Applications: Students are encouraged to begin talking to the participating organizations as soon the list of accepted organizations is published. Prior to the opening of applications, taking some time to talk to would-be student applicants helps them refine their ideas so that mentoring organizations will have better quality proposals to sift through. Each organization is asked to provide a proposal template, but the best student applications go far beyond the template and an organization's ideas list. Students are given at least a week to complete their application.

Following the student application deadline, organizations begin reviewing the proposals they received. During the review phase, organizations maintain an open dialogue with their student applicants, asking them to refine their proposals or doing further additional interviewing to determine which students are most likely to be a good fit for the community and the work required. Over the course of several weeks, each organization whittles down their list of proposals to those they would most like to see funded. Google lets each organization know how many of its student proposals they will fund, and organizations select their top N proposals.

It is sometimes the case that a single student's proposals fall within the top N for more than one organization. Google leaves it to the organizations and the student to decide which organization the student will work with during the course of the program. Typically, this duplicate accepted state is resolved by discussions between the organization administrators and the student. Technically the organizations are not required to involve the student in the decision process, but it is good practice to take their preferences into account. Following any such resolutions, the list of students who are to be accepted is considered final and is then published.

The Community Bonding Period: Before students are expected to start working on their projects, there is a period of 6-8 weeks built into the program to allow them to get up to speed without the pressure to execute on their project proposals. During this time students are expected to get to know their project communities: join in IRC, introduce themselves on the mailing lists, etc. This time is also when students should begin setting up their development environments, understand how their project's source control works, refine their project plans, read up on any necessary documentation and otherwise prepare to complete their project proposals. Mentors should spend this time helping their students understand which resources will be most useful to them, introducing them to the members of the community who will be most helpful with their projects and generally acculturating them.

Start of Coding: Start of coding is the date when the program officially begins and students are expected to start executing on their project proposals. At start of coding Google provides an initial payment to the student, just over 10% of the overall stipend. At this point, students should begin regular check-ins and regular patch submissions.

Midterm Evaluations: Approximately half way through the program, Google requires that mentors submit evaluations of their students progress. Should the project not be proceeding effectively it does not continue: and the student is dropped from the program. Students who receive a successful evaluation from their mentors continue working on their projects and receive a second program payment, approximately 45% of the overall stipend. Google also asks students to submit an evaluation reviewing their project to date, their mentor and organization's performance and any obstacles to their progress. Google may also ask org admins and mentors without students assigned to submit a general evaluation about the program during this phase.

As software development is an iterative process, it is quite common that the original project plan will need to be reworked and new milestones set. Following evaluations, Google will help the organizations and their students make any necessary course corrections. Directly following midterm evaluations is the perfect time for mentors and students to have a formal check-in regarding progress to date and to reset goals for the project as needed.

Pencils Down: At the final deadline for coding, students are welcome and encouraged to continue working on their projects, but their work will only be evaluated based on work done before the pencils down date. Google suggests that work be done about a week earlier to give the student time for last minute improvements and

corrections, as well as preparing their work for delivery.

Final Evaluations: Final evaluations should be based only on work the student has completed during the program. Should the project goals not have been met to the satisfaction of the mentor, the student is dropped from the program and does not receive anything further from Google. As with midterm evaluations, students are asked to submit an evaluation of their overall success. Google will ask all participants from each organization to submit an evaluation of the overall success of GSoC.

Post Final Evaluations: Students who successfully complete their final evaluations are asked to submit a code sample to Google. These students then receive the final program stipend payment, a certificate of completion and a truly spiffy t-shirt. All program mentors and org admins also receive a t-shirt.

It's a goal of Google Summer of Code that the student participants stick around long after the program has ended and continue contributing to their project communities. Great mentors continue working with their students to encourage them to do so. It's also customary during this time for organizations to publish a post-GSoC wrap up report letting the whole world know how their participation in the program fared. Everyone, both mentors and students, take a well deserved break, but truly energetic organizations begin planning for the next GSoC during this time.

Why GSoC Matters

In addition to the great code produced, the GSoC experience is about building open source communities. By focusing on students, we are going to the source of all future open source efforts. The ultimate success of the GSoC program is thus measured by the quality of the student experience. Each organization and each mentor plays a crucial role in creating this experience through their project ideas, developer culture, and the guidance they provide. And they are likewise rewarded by not only adding lines of code to their base, but also adding to their ranks well-integrated members with a strong head start. But don't take our word for it! Here's what some of the students had to say :

â Overall, this is the most productive summer I ever had. It increased my confidence as a developer and as a person that I can actually pull off a project like this and interact with awesome people like you. I also become a part of a growing community and hope to help it grow further.â

Chetan Bansal

â There aren't many opportunities for computational biology enthusiasts to make a difference in the field while still in school. GSOC at [my org] was one such gem of an opportunity that illustrated exactly why writing software for biological research is benefited by a background in biology. The experience I have gained here is definitely irreplaceable. The fundamentals of open-source programming and the rhythm associated with regular coding and problem solving helped nourish an intellectual side of me that I will not forget in a hurry. This is definitely not the last time you will see me.â

Chinmoy Bhatiya

â This was the first time I took part into either GSoC or an Open Source project, and it was also one of the most exciting things I've ever done! This project gave me the chance to spend ~3 months learning lot of new things, having fun, doing something for the community and even getting pay for that! That's an amazing combination! â I will remain as an active participant of [my org's] community beyond the official end of GSoC 2009!â

Gerardo Huc

â It was definitely a very good learning experience for me. â !But I have some unfinished worksâ ! So Iâ ll continue this project after GSoC, and be help to drive [my org's] development.â

Kozo Nishida

â It was the first time that I worked with an open source community and it was really a great experience. I am very thankful to Google and [my org] for providing me this great opportunity. I would like to thank [my mentors] for their excellent support during the summer. Looking forward to working with you again.â

Srinivasarao Vundavalli

â It was a GREAT experience to work with [my org] during GSoC 2009!!! The administrators and mentors were very helpful during bounding and coding periods. Whenever I had problems they were always responsive and offered me help in time. Under their guidance, Iâ ve improved my programming and communication skill, and learned how to work within a group. I would like to express my gratitude to all my mentors in [my org]. If I can participate in GSoC next summer, I would like to work with [my org] again:)â

Xuemin (Helen) Liu

"Get over yourself. Understand there are people smarter than you or people better at some things than you. Just the same, you're still needed. JFDI (Just Fabulously Do It) and find your niche."

Justin Hunter

"This year's GSoC was my first contribution to the open source world and I will like to continue my contributions to open source whenever I get the opportunity. The best thing about open source is the huge opportunity it provides to everyone who wants to explore their career in the technical world and develop in their field of interest. The way people from different streams come together, exploit their talents, cooperate,

coordinate, focus their collective efforts towards a common goal that will help many more people in the end is the best part about open source."

Kanika Vats

"So, we've now gotten to the end of it all. It does feel a bit sad, I really did have a great time coding this summer and hopefully I can do it again. I don't think I've ever learned this much in a summer job and everybody working with me have been really fantastic."

Anna Granudd

About This Manual

This manual was written during a Book Sprint sponsored by Google and facilitated by FLOSS Manuals. The manual was written in two days but the maintenance of the manual is an ongoing process to which you may wish to contribute.

Since the manual may be updated at any time, you may wish to periodically check here for updated versions : <http://www.flossmanuals.net>

You can also find the epub (for use with Android ebook readers software, Sony Reader etc) of this book here : <http://objavi.flossmanuals.net/books/>

How to Contribute to This Manual

If you would like to contribute then follow these steps:

Register: Register at FLOSS Manuals:
<http://en.flossmanuals.net/register>

Contribute: Visit the manual on the following URL <http://en.flossmanuals.net/bin/view/GSocMentoring>. Press 'edit' and start work on a chapter.

Chat: It's a good idea to talk with us so we can help co-ordinate all contributions. We have a chat room embedded in the FLOSS Manuals website so you can use it in the browser.

If you know how to use IRC you can connect to the following:
server: [irc.freenode.net](irc://irc.freenode.net)
channel: #flossmanuals

Mailing List: For discussing all things about FLOSS Manuals join our mailing list:
<http://lists.flossmanuals.net/listinfo.cgi/discuss-flossmanuals.net>

For more information on using FLOSS Manuals you may also wish to read our manual:
<http://en.flossmanuals.net/FLOSSManuals>

About the Authors

This manual exists as a dynamic document on flossmanuals.net, and over time will have an ever-increasing pool of authors and contributors.

The following individuals were part of the 2009 GSoc Book Sprint. We thank them for their tireless efforts to create this first-of-its-kind volume.

Alexander Pico

Alex works at the Gladstone Institutes in San Francisco as a Bioinformatics Software Engineer. He holds a PhD in Molecular Neurobiology and Biophysics and has over 10 years of bioinformatics experience in data mining, analysis, visualization and integration. Over the past 5 years he has led the GenMAPP pathway analysis project, managing software development and coordinating research projects involving wet lab biologists and senior programmers. Alex is a member of the Cytoscape core development team, a creator of SNPLogic.org, and a founder of WikiPathways.org. He has also administered GenMAPP's participation in the Google Summer of Code program for the last 3 consecutive years.
<http://nextnucleus.org>

Bart Massey

Bart Massey graduated Reed College in 1987 and spent two years as a software engineer at Tektronix, Inc. He received his MSc in Computer Science from University of Oregon in 1992 and his PhD in 1999 for work with the Computational Intelligence Research Laboratory there. Since then, Bart has taught at Portland State University, where he is currently an Associate Professor, and for the Oregon Master of Software Engineering program. Bart is Secretary of the X.Org Foundation Board; his current research interests include open tech, software engineering, desktop interfaces and state space search.

<http://www.cs.pdx.edu/~bart>

Jonathan "Duke" Leto

Jonathan is an open source hacker who currently focuses on the Parrot Virtual Machine and Rakudo, Perl 6 on Parrot, as well as being the maintainer of many CPAN modules, many with a focus on scientific computing and cryptography. He first was a mentor for Math::GSL in GSoC 2008 and then became organization administrator for The Perl Foundation's involvement in GSoC 2009 as well as being the mentor for Math::Primality. Jonathan received a masters in mathematics from University of Central Florida and has published several papers in the field of differential equations. He enjoys discovering wheels within wheels.

<http://leto.net>

Jennifer Redman

Jennifer has over fifteen years of hands-on technical experience as a data center and information systems architect and administrator. Jennifer is currently the Associate Systems-Keeper for Systems, the oldest International online community of technical women. When not tinkering with her servers and installing new flavors of Unix or Open Source apps, Jennifer travels to interesting and sometimes "you went where?" sort of places. Previous careers included canvassing for Greenpeace and as a staff member on a national (and successful) presidential campaign. She also reads a lot of books.

<http://www.buunabet.org/>

Leslie Hawthorn

Leslie held various roles at Google before joining the Open Source Programs Office in March 2006. Her first project after joining the team was spinning up Google Summer of Code 2006 and she has managed the program ever since. She also conceived, launched and managed the Google Highly Open Participation Contest, an initiative inspired by GSoC that helps pre-university students get involved with all aspects of open source development. Mentoring in open source communities is one of her personal passions, along with humanitarian uses for open source software. She loves to cook, read and can occasionally be found pining for illuminated manuscripts. She also likes to think of herself as a superb filker.

<http://www.hawthornlandings.org>

Olly Betts

Olly is the lead developer of the Xapian search engine library. He's been working on Xapian for 10 years, and makes a living as a freelance developer and consultant on Xapian-related projects. In GSoC, he's represented SWIG as a mentor in 2008, and a mentor and co-admin in 2009. Olly is originally from the UK where he studied mathematics and then computer science at Cambridge University, but now lives near Wellington in New Zealand. He once broke a toe falling off a cliff in Majorca.

<http://survex.com/~olly/>

Selena Deckelmann

Selena works for End Point Corporation and is an enthusiastic open source advocate and PostgreSQL specialist. She is co-chair of Open Source Bridge conference, a conference for open source citizens. She currently leads PDXPUG, a PostgreSQL Users Group, and helps organize Code n Splode, a programming group whose goal is to get more women involved in open source. In her spare time, she likes to mix drinks for her local Perl and Postgres user groups, and fetch eggs from her chickens (when she has them).

<http://chesnok.com/daily>

Facilitation

The Book Sprint was facilitated by :

Adam Hyde

Adam is the founder of FLOSS Manuals. FLOSS Manuals is a community of 1500 (at the time of writing) volunteers that create quality free documentation about free software. FLOSS Manuals is pioneering the Book Sprint methodology that enables the development of well written manuals on free software in 2-5 days. Adam has facilitated over 15 Book Sprints on Free Software including Inkscape, OLPC, Sugar, CiviCRM, Firefox, Introduction to the Command Line, Digital Foundations (conversion to free software examples), Ogg Theora, How to Bypass Internet Censorship, Open Translation Tools, PureData, Video Subtitling and now the Google Summer of Code Mentors Guide. Adam is also the Project Manager for the development of 'Booki' - the forthcoming free software Collaborative Authoring Platform.

adam@flossmanuals.net

What Makes a Good Mentor?

Mentoring a student can be a very rewarding experience. However, being a good mentor is *not* just a matter of winding up the student and watching them go. There are specific skills that you need in order to be effective; even experienced mentors can improve these skills. This chapter highlights some of what it takes to be a good mentor.

Are you already part of the developer community? If you are not then you are not going to be effective at introducing a student to the local culture and practices. Similarly, you are not likely to be able to propose, guide and integrate successful projects relevant to the larger effort. On the other hand, if you are new to the community, working as a backup mentor on a project may be an excellent way to get involved.

Do you have a vested interest in potential GSoC projects? As a GSoC mentor you will be taking ownership of a project idea and seeing it through. If you are not passionate about the project, you will be wasting everyone's time. You are an integral part of the process from project proposal to integration. You should have project ideas that you are excited about and keen to see through the summer.

Are you willing to dedicate significant time? It is difficult to put a number on such a subtle art as mentoring. You should seriously consider your prior mentoring experiences and your available time before committing to this role. If you really don't want to or really won't be able to mentor, *then don't offer*.

Are you keenly interested in mentoring students? The main goal of GSoC is mentoring students. This is obviously important to the future of open source software, our immediate projects and the overall culture. Mentoring a student requires a combination of passion, responsibility and patience. A good mentor is willing to engage with students throughout their learning process.

Be Prepared to Seek Help

At all times don't forget that you have access to people, tools and resources that can make your job much easier *and* make you a better mentor. Make use of your org admin when you are not sure what is expected of you or have a difficult situation with your student. Make use of other mentors in your organization and the thousands of mentors on the mentor mailing list. Though it may be an annoying list at times (*don't feed the trolls!*), it is a valuable resource. The GSoC admins are another important resource. They set the tone and standards for the entire program. They have heard it all, so don't hesitate to contact them when a problem arises.

Backup Mentors

An organization may choose to assign backup mentors, which are recognized by GSoC, but not officially responsible for the project or evaluations. Backup mentors are useful during absences by the primary mentor. They may also have relevant expertise.

What To Expect From Under-Mentoring

The student's project is never properly defined. The project goals and deliverables are unclear, and the work schedule is not set. The consequences of this are serious and impact the project if left unchecked.

The mentor has little idea what the student is doing. The state of the project is unclear, and its progress is uncertain. Evaluation is impossible to do well.

The student gets stuck. The student seems to be engaged, and to be working hard, but no apparent progress is being made. Alternatively, the student's communications are infrequent and terse, and seem to always be on

the same issue or milestone.

The student disappears perhaps for days or weeks at a time. If the student is under-mentored, it may be difficult to determine when this period began, and thus to know when to panic. Insufficient information is available for evaluation, thus it becomes impossible to fairly evaluate the student.

Defining a Project

The process of defining ideal GSoC projects is not just a "summertime" activity. Such projects are generally useful to the overall open source software development effort. The same qualities that go into a good GSoC project go into entry projects for new developers and even projects that help to recruit new developers. The process of putting together a well-defined GSoC project also forces you to think about your project from a new point of view. This is a valuable exercise for defining the current scope of and potential new avenues for your work. In other words, it can be so much more than just getting help with your existing workload.

Your goal for GSoC is to generate a list of project ideas that capture the development needs of your organization, attract the interest of students, and help you get things done. This is often done as a community effort, involving as many potential mentors as possible; this helps to get buy-in from the mentors, and gives a broad range of perspectives on organization needs. It should also be part of an ongoing, long-term strategy rather than a rushed act to meet the application deadline. Many organizations maintain such project lists year-round.

There is an art to writing a project description that leads to good student applications. It is tempting to write a detailed project plan for the student to follow. However, students tend to just echo that plan back in their applications, making it difficult to evaluate their quality. It is better to briefly describe a general high-level need, and the motivation behind that need. Keeping the scope modest helps to encourage more applicants, while adding a stretch goal to the project description may encourage stronger students to take on the challenge of meeting it.

One strategy is to leave an opening for students to propose their own original project ideas. Some great ideas can come out of this process. Emphasize that the student who is proposing to do something original needs to engage with the community strongly before or during the application period, to get feedback and guidance to improve the proposal.

Note that the quality of the project descriptions on an organization's "ideas page" is one criterion for the organization's admittance into GSoC. It is worth spending some extra effort to ensure that the projects you propose are worthy of the GSoC banner.

There are many ways to define a good GSoC project, probably as many ways as there are student-mentor pairings. Here are just a few:

Low-hanging fruit: These are projects that require minimal familiarity with the code base and basic technical knowledge. They are relatively short, with a clear end.

Risky/Exploratory: These are projects that push the scope boundaries of your development effort. They might require expertise in an area not covered by your current development team. They might take advantage of a new technology. There is a reasonable chance that the project might be less successful, but the potential rewards make it worth the attempt.

Fun/Peripheral: These are projects that bring a fresh perspective to the effort. They might not be related to the current core development focus, but that create new innovations and new perspective for your project.

Core development: These projects derive from the ongoing work from the core of your development team. The list of features and bugs is never-ending, and help is always welcome.

Infrastructure/Automation: These projects are the code that your organization uses to get its development work done; for example, projects that improve the automation of releases, regression tests and automated builds. This is a category in which a GSoC student can be really helpful, doing work that the development team has been putting off while they focus on core development.

The project you propose will define the tone and scope of your organization's participation in GSoC. It is a key part of your organization's application. Further, it may be the first impression made on a potential student applicant.

Pro Tip: Maintain an "ideas page" with a running list of entry projects year-round. This can benefit your development effort throughout the year. It can also make your organization's GSoC application easier to put together the following summer.

Don't Be That Guy: Don't propose projects that neither you nor anyone else wants to mentor.

Selecting a Student

Successful participation in GSoC is based on a three-way fit between mentor, student, and project. As a mentor, your role in finding that fit is two-fold: help the organization identify and select strong students appropriate for your projects, and to find a pairing between yourself and a student that is productive and pleasant. Fortunately, these goals are quite compatible.

Understanding Student Motivations

A helpful starting point for finding, evaluating and selecting students is to look at the process from their point of view. Why do students apply to GSoC?

"I want to be rich." The stipend that students can earn for the summer is an important motivator for many. One of Google's explicit goals is to enable students to spend the summer coding "instead of flipping burgers".

"I want to be famous." Being a GSoC student carries a certain amount of prestige. However, the desire for fame is not a sustaining long-term motivation. Be aware of the difference between a student who wants to be "accepted" versus "successful".

"I want to learn." Students may want to learn various things as part of their GSoC experience, such as how to work in the organization's project, how to do open source development in general. But, it is important as a mentor that you are cognizant of the basic skill set required for the project.

"I want a t-shirt." Some motivations are simple, and easy to accommodate.

Students should want to participate because they have something to contribute to the organization's project. This is obviously an exciting and promising kind of GSoC applicant to receive. Assuming that the student has good technical skills and can interact well, a good result is almost inevitable.

Selecting a Student

There are some student qualifications that are pretty much a given for any successful GSoC experience. The student needs to be technically skilled, needs to have good communication skills, and needs to be a hard worker with sufficient worktime to succeed. Determining whether a student has these basic qualifications given a brief application document and some tiny amount of remote interaction is exceedingly difficult. Hopefully the organization has an application process in place that helps. However, as a mentor you will normally be expected to assist in the evaluation that will ultimately decide who gets accept to GSoC from your organization.

You have several techniques at your disposal for helping your organization evaluate students. First and foremost, your expertise is key in evaluating student proposals. Is a given proposal technically realistic? Is it useful to the organization? Does it meet the organization's overall goals for GSoC?

Some of the student application your organization receives will be obvious winners, about which little discussion is needed. Many more will be obvious losers that need little discussion at all. Applications that fail to conform to the organization submission rules, are extremely short, or are difficult to read or understand almost inevitably come from students who would fail miserably if accepted. The middle ground in student applications is where the action is. There are several techniques for assessing these promising but troubled applications:

Send an early query to the student asking for more information. Failure to respond well or in a timely fashion almost guarantees problems with the student later on.

Watch the student's community interaction. The best students interact with your organization's community during or even well before the application period. A mediocre application is much less concerning if it looks like the student is already moving forward.

Inquire about the student's GSoC history. A student who has participated in GSoC before may be easy to figure out. Past performance is usually an indicator of future GSoC success. This information could be asked for in your organization's student application template, or from a general web search.

Find out where else the student has applied. Does the student have other GSoC applications? Did they copy-paste the same application over and over? There are often opportunities for negotiation with other organizations around a student who has applied several places.

Look at the student's other summer plans. A student who at least claims to be solely focused on GSoC is more likely to be successful.

Finding a Match

As a mentor you want to do more than just help your organization select the best students. You also want to ensure that they select a student and project that you will enjoy working with.

This may involve more than just finding a bright student with the right area of expertise. It is worthwhile to look at the personality type and work style a student application reflects. If you are a methodical, organized person, for example, a loose and casual student style might not be an ideal fit for you. Mentoring a student geographically far from you can be a bit challenging, but also quite enlightening. Be aware of timezone differences that might require early morning or late night schedules for live meetings, which are critical for effective mentoring.

For those unfortunate students that don't make the cut for logistical reasons (e.g., not enough mentors or funded slots), consider providing feedback to let them know their applications were valued. This is a service both to the student and to the organization. These students will be more likely to stay engaged, possibly even contributing outside of the official GSoC program, and returning next year with an even stronger application.

Google's Selection Process

The Google Open Source Program Office (OSPO) has an internal process to select mentoring organizations and allocate GSoC positions; understanding it may help you make better student selections. Refer to the ample documentation updated yearly for guidance on its workings.

In brief, OSPO allocates each organization a number of "slots" based primarily (but not solely) on the number of student applicants. The organization is then responsible for ranking its student projects. The top-ranked projects equal to the number of allocated slots are then the accepted projects for your organization. You should also understand how student selection and mentoring can affect the eligibility of an organization. In particular, note that a poor job of mentoring that leads to poor outcomes makes it less likely that your organization will be selected in future years.

Starting at the Beginning

Like most things, a successful GSoC project is set up by a successful initiation. Finding the right three-way fit between mentor, student and organization can make success incredibly easy. Conversely, failing to find this fit makes it difficult or impossible for the project to succeed.

Once a fit is found, the project is ready to be elaborated. You and your student are prepared to embark upon a grand adventure. Excelsior!

Pro Tip: One temptation to be avoided is to give a promising student excessive help in rewriting their application. It is likely that the result will be an application stronger than the student it represents. Students' communication, organization and logical thinking skills rarely improve over the course of a summer.

Pro Tip: If in doubt about a student applicant leading up to the final ranking and allocation, err on the side rejecting. Limited program budget and mentor time can most certainly be better spent on another student in your or another organization. Yes, you can release slots to other organizations!

Don't Be That Guy: Don't even think about selecting a student with whom you've had no contact. You should establish an active back-and-forth prior to making a decision. If you or your student have failed to make this happen. Do not proceed.

Community Basics

Your community is the collection of individuals who work on or with your project. Helping your student become familiar with those individuals makes them a more effective contributor during GSoC and helps to make them feel part of your community, which encourages involvement after their GSoC project is completed.

Set the Tone

As a mentor, you can really help to set the tone for your student's initial experiences by facilitating community. Encourage your student to introduce themselves on mailing lists and IRC, and invite comments on their proposal. If your project maintains biographical information on contributors, ask your student to read through them, and spend some time talking with your student about the people with whom they may interact.

Once a student has a feeling for who's who in the community, they are more likely to communicate and seek advice from others. This increases their chances of getting issues resolved more quickly and effectively than if they relied on only their mentor for help.

Encourage Questions

Asking effective questions is a skill, not an innate talent. There are many resources for learning how to ask questions effectively, and Eric Raymond's classic *"How to ask good questions"* <http://catb.org/~esr/faqs/smart-questions.html> is a great place for students to start. Another useful resource is Simon Tatham's *"How to Report Bugs Effectively"* <http://www.chiark.greenend.org.uk/~sgtatham/bugs.html> which isn't explicitly about asking questions, but does cover aspects of effective communication about technical issues.

Before your student asks a question, it's often helpful for them to take a minute to struggle with it and attempt to find the answer on their own. That extra effort often helps your student to solve their own problem, and if not it sharpens the question they ultimately ask.

Criticism Happens

Discussion in open source communities can be very direct. People will often criticize the bad points of a patch or suggestion, but fail to praise what is good about it. Explain to the student that comments aren't meant to be critical of them as a person, but are aimed toward improving the patch or idea, and the project in general. Making potential new contributors feel welcome in your community is also important outside of GSoC.

Pro Tip: When you announce that you're taking part in GSoC to the community, make a point of explicitly asking people to make the students feel welcome.

Best Practices

Effective communication between mentor and student is absolutely vital to a successful and rewarding GSoC experience. Here are some tips for encouraging good communications:

Share contact details: Swap contact information with your student and org admin early on. If your contact information changes, be sure to tell people, and make sure your student does too. Be sure to keep the org admin informed about when you may be unavailable to your student, so that they are not caught unaware when your student contacts them.

You need to take care if you are planning a trip during the summer, especially if it is more than a few days, or near milestones in the GSoC timeline. If you coordinate with your student beforehand and give them sufficient work to chew on, it can go smoothly. What you don't want is for your student to be blocked on something while you are inaccessible and do nothing until you return. Make sure to give your student many different tasks to work on, so that she can work on something else if you are not available. This is a great time to utilize a secondary mentor to act as a backup in your absence.

Choose communication channels: Decide upfront how you are going to communicate with your student and what type of technology or medium you are going to use. Don't wait until mid-way through the GSoC to figure out that one of you can't get your microphone working on your desktop for voice chat.

It's good to make use of multiple means of communication, as different platforms can complement each other. Instant methods like IRC or IM are great for getting a quick answer or for interactive discussion, but require both parties to be available at once. Public communications are generally preferable to private ones.

Although many people prefer text-only communication, it is very helpful to talk on the phone or video-chat with your student at least once at the beginning of the program - hearing a voice or seeing a face can help people feel more connected, creating a sense of mutual respect and responsibility.

Decide on the frequency of reports: Discuss and decide the frequency and form of status reports upfront. Many organizations require students to deliver weekly status reports (e.g. posted to your development list or on their blog). Make sure you clearly delineate in what format the reports should be delivered and what information needs to be included. Providing an outline or template can be helpful.

Establish frequent chatter: You really want to be hearing from your student more often than once a week as this is a long time for a student to be stuck or heading up a blind alley. Be sure to encourage and instigate frequent communication for rapid question and answer and lightweight mentoring throughout the week.

Provide a safe environment: Create an environment in which your student feels comfortable enough to ask questions that she may believe to be "stupid". This will help to avoid them getting stuck, and fosters positive mentor-student and student-development community relationships, which is extremely important for GSoC success and fostering and encouraging long-term contributors. Some ways to help your student understand that his question is not stupid but an important part of the project's success:

Avoid gruff "RTFM" replies: It's likely the student will ask questions which are answered somewhere in your project's documentation, but do take the a few moments extra to politely point to the information, or you'll risk the student feeling reluctant to ask next time he has a question.

Ask some stupid questions yourself: Chances are your student has some area of knowledge that does not pertain to your current skill set, or maybe you've just forgotten the answer. Ask her. Or if you're asked a question which someone else in your project could answer better, put your student in touch with them.

Be inclusive: Invite your students to participate in your community events, not just the core development processes. This could include group retreats or related conferences that members of your community are

planning to attend.

Introducing your student to the community's communication style: Some groups have developed codes of conduct, such as Fedora IRC Helpers Code of Conduct (http://fedoraproject.org/wiki/IRC_helpers_code_of_conduct), which help initiate new members and guide the interactions of existing members. You are encouraged to read the code of conduct and think about which issues and solutions might apply to your group.

Mailing list etiquette: The GSoC mentors mailing list has more than 3000 subscribers. If you have a question for the Google team, mail ospoadmin@gmail.com instead of the list. Please take a moment to review list etiquette: <http://www.gweep.ca/~edmonds/usenet/ml-etiquette.html>

Addressing communication disconnects: Whenever working with people for the first time, a best practice is to assume that they do not mean any harm. If your student, for example, makes a comment via email that offends another member of the community, it's appropriate and constructive to speak up and address the issue. Assuming that the comment was the result of misunderstanding, rather a result of malice, allows you to ask questions and help your student adjust to your community's communication style.

After asking questions, you can then offer an explanation to a person or a suggestion on how they could behave or phrase their comments differently in the future. When offering individual coaching on how to behave, be mindful of embarrassing your student in public about what's happened, or demanding apologies. If possible, talk to your student 1:1 using an immediate communication medium like IM, IRC, the phone or in person is better than sending email. Remember that you're telling someone that they did something wrong, and keep in mind how you'd feel if someone was telling you that you'd made a mistake.

Effective Apologies: Making apologies is a fact of human life, and open source communities are no exception. In the event that you or a student finds themselves needing to make an apology, there are a few things that might help you apologize effectively:

- Make the apology as soon as possible
- Make it clear in the subject line that you're apologizing
- Make it an honest apology and not a defensive statement in disguise

Giving and Receiving Criticism

Mailing list culture and public code review can be a rude awakening for newcomers. Submitting a patch to a mailing list might be a student's first experience with public critique of their code.

Project policies vary widely on how contributions are treated. Some encourage committing early and often, hopefully maintaining a stable branch for bug-fixes. Others refuse to commit code to the core repository until a patch is fully discussed, tested and documented -- most projects lie somewhere in between.

All projects, however, engage in some form of code review and the inevitable criticism. Having people look at and ask questions about your code is a fact of working in the free and open source world. The directness around code review is one of the great strengths of open source code. People hone their coding skill and learn rapidly from others. Bugs are found quickly and fixed. And generally, the whole process is documented in source control repositories and mailing lists and made available online.

Here are some ways that you can help students prepare for code review:

- Provide example mailing list threads, or a list of the types of questions that are asked about code
- Direct students to review coding standards that apply to your project
- Go through an example code review on the student's first code sample submission that matches as closely as possible the process that your project normally goes through

People are more likely to respond positively to criticism from people that they trust and respect. Try introducing students to the people that are likely to comment on their code, and explaining the role that those people play in the group. Also, encourage students to both defend technical decisions, and be gracious in admitting mistakes.

Code, in addition to being mathematical and scientific, has been compared to poetry or creative writing. Most developers have a sense ownership and creative discovery associated with their code. You may have "gotten over" the bruised ego you received the first time that someone told you that a variable name, or an algorithm choice you made sucked. But a student may be experiencing this for the first time. Stay aware of the difficulty involved being told that you've made a mistake, and be encouraging about the fact that the students are participating in the first place. A simple "thank you" when students publish code, send email to a mailing list or make other contributions go a long way.

Make an effort to assist a student in their first steps into your community. Offer to proofread emails, blog posts, or patches. You don't want to do all the work for them, but you can help students feel confident in their communication, especially when they're just starting out.

Pro Tip: A good status report should include a few items that went well during the previous period, and a few problems that were encountered and how they were solved -- or not.

Don't Be That Guy: Don't reinforce your student's perception that by asking you a question, you automatically assume they are indeed stupid. Be nice. Remember you were also once a beginner.

Warning Signs

As a GSoC mentor, you are in the best position to identify "red flags" and warning signs early on. This is critical to being able to address problems before they get out of hand. Learn from the mistakes of others and learn how to identify and mitigate the following warning signs.

Not enough hours in the day: If your student has a full-time job or is attempting to defend a graduate thesis during the summer, that is probably going to not work. Even though your student thinks they will have enough extra time, don't believe them. They won't. If your student has every single minute of every day completely booked, then any unexpected event, such as getting sick or a family emergency, derails this plan beyond repair. If your student cannot commit to a specified time schedule, this is an immediate red flag that they need serious help with time management.

Where is my student: Missing a predefined meeting is a serious warning sign that your student is not taking the process seriously and this should be remedied as early as possible. If your student was in such a deep "coding zone" that they forget the meeting, and you have it later on, that might be acceptable. That brings us to the issue of "valid excuses."

My village was invaded by aliens: What is a valid excuse? Students have been known to come up with outlandish excuses as to why they are not meeting their milestones. *You did agree on milestones beforehand, right?* If you think that your student is lying to you, that is a huge warning that things are going sour. Make sure to remind them that the truth of "real life just got in the way, I will redouble my efforts next week" is always better than "my village is being invaded by aliens."

Bad students happen to good mentors: One thing to keep in mind is: *Sometimes bad students happen to good mentors.* Don't take it personally. If a mentor tries their hardest and their student fails, this does not reflect badly on the mentor.

Actions Speak Louder than Words

This bit of folk wisdom is proven year after year during the Google Summer of Code. A mentor will write to the mentors mailinglist asking for advice about a student that has not performed or gone missing for a period of time. Mid-term evaluations are coming up and the student has reappeared reinvigorated with no shortage of excuses and the mentor wants to know, "should I pass them?". What follows on the mailing list is story after story of prior experiences from other mentors who ignored the warning signs only to be disappointed later. The mentor considers the advice, but focuses on the unique aspects of their particular situation and passes the student... and their story is added to the list the following year.

Here are some specific scenarios to watch out for:

The Disappearing Student: A student submits a great proposal and is enthusiastic about discussing the project and getting started. You rank them high, they get accepted, and then they drop off the face of the earth. **Fail.**

Under Performance: The student sticks around for the community bonding period, but then it comes time to work. When it comes to actually writing code, they seriously under perform. They offer excuses when pressed, and offer scraps of code here and there. Then comes the mid-term evaluation and start committing like crazy. **Fail.**

Wrong Priorities: The student passes the mid-term and then goes on an unscheduled holiday for two weeks or starts a part-time job and the quality or quantity of work is seriously affected. You discuss this with them immediately and they promise to reprioritize, but the work is not produced. **Fail.**

Depending on your personality type, some of these might seem harsh. You might also shoulder some of the blame because you think that if you had been a better mentor, more on top of the situation, it would have been avoided. But even if you are partly to blame, so is the student. And it is up to the student to perform when expectations are communicated and agreed upon.

Pro Tip: In the event your student's boat is sinking, all mentors are equipped with emergency rocket-propelled jet skis.

Open Source Culture

When a student encounters an open source group for the first time, it may be a bewildering experience. Whether posting to a mailing list for the first time, blogging about the project they're taking on or hanging out on an IRC channel - the way people interact, and what they expect from each other is pretty different than a classroom, or their friends and families.

This chapter is about helping you identify things about open source culture that may need to be explained and made explicit to newcomers. This information is useful for students, but also is important for interacting with the wider world. One advantage that mentoring organizations have is they help define these aspects of their organization to the wider world, not just for students.

Openness and Sharing

Open source projects vary a lot. A core value they hold in common is that sharing code is good. Regardless of license, indentation style or language chosen, developers create, share and modify source code together.

Working on code together means a lot of things - transparency, directness and cooperation are words that are often mentioned by developers when describing the process. It can involve bug reports, re-factoring, implementing new features, documentation, project management and advocacy.

Amazingly, the ways in which people actually share code are as varied as the individuals involved. Orienting students to your project's practices will not necessarily prepare them for how other projects operate. Keep this in mind if you are integrating a new person that may have had previous open source experience -- you still need to take the time to show them around, and acquaint them with your particular brand of sharing.

One aspect of the "open culture" is that people are informal. People address each other by first name. They tend to speak directly to one another, regardless of social status, or formal title. Disagreements about code - whether as profound as which algorithm is most appropriate, or as seemingly mundane as how many spaces are used for indentation - are aired in public. This process is very intimidating to newcomers, who might be concerned about having their words immortalized on the internet, and worse, saying something wrong or embarrassing. The only way to get over this is to practice, and you as a mentor need to encourage students to participate publicly.

Remote Communication

Many projects involve individuals who are working not only in different cities, countries and continents, but collaborating across major cultural and language differences. And rather than having procedures or policies on how to interact with one another handed down from HR departments or other authorities, communication practices evolve between individuals over time.

Because there are few rules around working together on open source projects, there's a lot of freedom to share directly with one another. This freedom is at the core of what attracts many people to open source software. Multi-cultural projects offer an incredible opportunity to share knowledge between individuals directly. With sharing, however, comes a responsibility to inform new participants about expectations for communication and ways to solve misunderstandings to the benefit of all.

Be mindful of cultural assumptions about race, gender, sexual orientation and disability. People often make assumptions, have stereotypes or are biased in ways that no one can control. The best practice is to assume that people don't mean any harm, and when they're told respectfully that they've offended or hurt someone, that they'll stop whatever it is that they are doing that is harmful.

All that tolerant rhetoric aside, it is never productive to allow individuals who consistently try to cause harm to others to do so. If a person is causing undue problems, consider removing them from your project. It's often better to ask someone to leave, than to allow them to harm others and in turn, cause other productive members of your team to depart.

Abbreviations and Slang

Groups of humans come up with abbreviations and slang that are meaningful inside the group, but not necessarily to outsiders. Encourage newcomers to ask questions when they don't understand a term, a joke or some arcane bit of project lore. Remember that you were the newbie once too, and take a few minutes to explain. The best jokes are the ones that everyone is in on. Letting your students in on these jokes helps makes them feel more included in your community.

Here are a few useful resources for teasing out meaning from initialisms, acronyms and abbreviations:

- Urban Dictionary (<http://www.urbandictionary.com>)
- Webster's Online Dictionary (<http://www.websters-online-dictionary.org/>)
- Acronym Finder (<http://www.acronymfinder.com/>)
- A to Z word finder (<http://www.a2zwordfinder.com/>) -- *also great for cheating at Scrabble or other word games*

Volunteerism and Gift Economies

Donated and free time are the life blood of open source projects. Many individuals contribute their time and energy without any expectation of compensation or even a simple thank you in return.

This culture may be unfamiliar to students, and require explanation.

Contributions to projects are often self-directed, with developers having a personal itch to scratch in the form of a new feature, by correcting a bug that they've encountered or by implementing something from a TODO list. Projects operating in this way may be seen as chaotic by people only familiar with top-down management - where teachers, professors or bosses tell them what to do and when. Of course, some projects do have clearly defined project management, with milestones and tasks meted out carefully.

Regardless of your organization's style, it's best to just help students see that your group is productive the way that it is. Obviously, the group got far enough to be involved in the Google Summer of Code, so they must have done something right! Help students by introducing them to the right people and processes, and by answering their questions promptly.

Because many (or in some cases all) contributors are volunteering, methods of coercion available to businesses are not available. The best way to collaborate is to behave in a way that encourages others, and ultimately, makes people want to contribute. Some easy ways to encourage volunteerism include:

- Saying thank you
- Giving compliments when they are deserved, regularly and in public
- Publicizing cool hacks and features as they are implemented, in blog posts and on the mailing lists
- Promptly committing useful code
- Responding promptly to requests for information
- Clearly defining ways to contribute to your project (TODO lists are great!)

Overall, your goal should be to create an atmosphere around contribution that is enjoyable. What that means

will vary significantly depending on the project, but certainly the above points apply to any project.

Consider treating every patch like it is a gift. Someone put time and energy into creating it, and even if you don't ever use it, recognize that effort. Being grateful is good for both the giver and the receiver, and invigorates the cycle of virtuous giving.

Mind the Gap

The community bonding period is the 6-8 weeks between when accepted GSoC students are announced and the start of coding date. Here are some of the goals of this period:

- Prepare students to immediately start writing code at the official start of coding.
- Get students engaged socially in the project.
- Provide time for students to learn about the development practices of the organization.
- Ensure that students have a development environment set up. This includes getting set up with the project version control system, read up on necessary documentation.
- Further refine the strategic plan for project completion.
- Get required forms filled out, such as the tax forms required by Google, any contributor license agreements, or things that your project requires.

This period was added in 2007 to help students integrate with their development community and so encourage them to become lifetime contributors. New contributors to a project outside of GSoC often lurk in a project's IRC channel and/or mailing lists for weeks or months before submitting their first patch. The community bonding period is an attempt to improve that experience.

Successful completion of your student's GSoC project depends a lot on the bonding period. Make sure that you and your student make good use of this time and make significant progress on preliminary tasks.

The community bonding period is also a good chance for the students to start interacting with each other. Early connections can help the students support each other during coding.

Ideally students are ready to start writing code at the official start of coding, and are already engaged socially. During the community bonding period students are expected to learn about the development processes of their organization, ensure they have a development environment set up, get set up with the project version control system, read up on necessary documentation, and further refine the strategic plan for successful project completion.

Plan weekly activities for your student that only take an hour or two. The community bonding period occurs when students are likely to still be taking classes and have a full load of course work. It's unreasonable to expect that the student will be available full-time to work on GSoC at that time.

Pro Tip: Students are meant to be "in good standing with their community" to receive the initial payment at the start of coding. If you don't hear anything at all from your accepted student during community bonding, or the student explicitly drops out, tell Google. It may even be possible to select a replacement student.

Setting Expectations

Successful mentors set expectations at the start of their projects. This includes communication frequency, project goals, availability and ways of delivering feedback. While the mentor should take the lead in expectation setting, the process of creating and documenting the expectations must be collaborative. Students and mentors need to agree on what is expected, or success becomes quite difficult.

Defining Success

Performance measures make it easier to provide feedback, to help your student get back on track if she veers off-course. Clearly stated measures also help you make a fair determination that a student needs to be removed from the program.

Get student input: Make sure your student has input into the types of performance measures used to determine success or failure. It is very important that your student helps create the performance measures to determine project success and failures. Your relationship should be a highly collaborative one.

Set achievable goals: Help your student come up with manageable project goals. Rather than defining the project as one giant chunk, help your student break the project goals down into smaller pieces that allow a change in direction if necessary. It is sad to work the entire summer on one giant deliverable, only to find out in the last few weeks that the architecture or design is defective.

Anticipate time away: Make sure to set expectations for known or planned time away from the project, such as a known course work, a vacation trip or holiday time. Talk about how many hours or deliverables per week would be reachable goals and what amount would be a good stretch goal.

Managing Output

Decide in advance what happens when project goals aren't met. Remember to be flexible if your student has made good progress or has obviously worked hard but needs to re-scope the project at mid-term. Good project management is hard. Your performance measures will help you manage project rescope.

Plan for Slippage: Have a plan to deal with scope-creep and timeline slippage. What if something happens that prevents your student from working successfully for an extended period of time? At which point do you need to terminate the project? Have a plan in place for these scenarios.

Gather Feedback: Your student's wishes and desires for a successful project are as important as the project goals. Make sure that you solicit and incorporate her feedback when coming up with initial goals, performance measures and communication methods.

Overall, communicate and be reasonable when it comes to your students. Be ready to revise project plans if an unexpected requirement or bug occurs.

Pro Tip: Ask about the weather and local stability of public services. Is your student using the cafe down the street for Internet access? Are there seasonal weather conditions that may lead to flooding and the subsequent inability to turn-on one's computer? Work on a plan to address these types of environmental issues that can affect both communication and output.

Don't Be That Guy: No one likes dictators. Work with your student on the development of expectations, rather than barking out orders.

Workflow

Make sure that your student is familiar with the workflow of your community as early as possible. Learning the toolchain your project uses and along any other software like libraries, version control systems, bugtrackers is high priority. Workflow also encompasses code review, talking with other developers about which algorithm is best for the problem at hand and other meta-questions about how to best get from specification to implemented solution.

The best kind of workflows for the GSoC projects are iterative. That means that small, quantifiable goals are defined and then acted upon. For instance, an example of an iterative workflow is:

- Write a test that demonstrates what feature will be added.
- Run the test to verify that it fails in the way you think it should.
- If it fails in an unexpected way, your test may be wrong. This is a great time to ask your mentor for guidance.
- If it passes, you are done! You just added test coverage to an already existing feature, and that is great!
- Add the feature (also known as a Simple Matter of Programming a.k.a SMOP.)
- Run the test to verify that it passes
- Write documentation about your feature.
- Rejoice appropriately.

This iterative workflow is known as Test Driven Development (TDD). This workflow gets applied to each feature that will be implemented, so a TDD workflow will consist of many cycles of the above steps, each for a different feature. The polar opposite of this workflow looks something like this:

- Write code the first 1/3rd of the summer
- Write tests the next 1/3rd of the summer
- Write documentation for the last 1/3rd of the summer

This is almost always doomed to failure, since people are always optimistic about time estimates for completing something. Sometimes you must debug a weird issue. You can't predict how long it might take to resolve the issue. What usually happens in this workflow is that the code gets written, but takes longer, half the tests get written and no documentation is written because the students have run out of time.

Take an approach which produces usable code even if parts of your plan fail, or are never attempted. Be humble and flexible about your development model. The student may teach you something!

Pro Tip: Progressive milestones may allow code to be merged progressively.

Managing the Plan

The student and organization application process to GSoC helps all participants think about how to run the Google Summer of Code at a community level. One important tool is the creation of a strategic plan. The phrase "project plan" frequently elicits a Pavlovian response of groans, shrieks and malaise in techies. An even more severe reaction has been observed in open source communities. So let's talk about a "strategic plan" instead, and how it can help your student be successful.

Creating a Strategic Plan

The GSoC application process assists in developing a good preliminary outline of a strategic plan for the student's accepted project. To increase the probability of program success you should spend time with your student, and your development community, refining the plan developed in the application period. This can be done during the community bonding period. A good strategic plan includes:

A high-level design document: Think about the architecture and design of the new feature or enhancements that you are making to your community's project. Take some time to teach your student to think about usability by writing a few quick use cases, and consider the introduction of any new dependencies.

Progressive milestones that build on the previous work: At the completion of each milestone you and your student should take the opportunity to celebrate the accomplishment and reflect upon it. Using progressive milestones also gives you a good measure of how far you come, which can be very useful during periods of frustration. Additionally, if you don't reach all of the final goals of the project, you will have tangible achievements to point to when reviewing progress with your student. Reinforcement of a student's tangible accomplishments encourages them to stick around and helps to create life-time contributors.

Target completion dates for each milestone: In reality, completion dates are going to move. Nonetheless, a target date gives you a time frame for closure and helps control "scope creep". Coach your students in how to recognize that a milestone is going to be missed, and to notify the project participants before the dates passes.

Tasks associated with each milestone: Because your milestones are most likely going to be chunks of code, each milestone needs to include both testing and documentation around that particular "chunk." This approach helps guarantee that you and your student don't end up with a pile of code that hasn't been tested or documented at the end of GSoC.

Following the Plan

After you create a strategic plan, you actually need to follow it. Some ways you can use your strategic plan to stay on track are:

Collect regular status reports. Status reports are an important communication tool. They are also very important in making sure that time-line slippages and scope creep are addressed in a proactive manner. You do not want to find out two weeks after a milestone due date that your student is going to slip the date because they have been unable to solve a simple problem.

Check off associated milestone tasks. Find a way to keep track of tasks, and then indicate when they are completed. This can be as simple as keeping an informal to-do list in that is referenced in weekly status reports. You can also use the project management software that the rest of your organization uses. Do what works best within your community, but make sure you do something.

Set an expectation of prior notification of missed deadlines. Your student is going to miss project deadlines. Make sure that he understands that it is important to notify you of the missed deadline well in advance. Understanding how long something is going to take to complete is a valuable skill, but it is one that

is learned through ongoing evaluation.

If your student misses a deadline, make sure you discuss why. Helping your student understand where he become bogged down or stuck helps with future strategic plan development. Remember that you are cultivating a long-time contributor.

Be a good example. If you, the mentor, need to miss a deadline, make sure you communicate this to your student well in advance.

Delivering Feedback

Throughout the project you should be delivering effective feedback to your student about their code, communication, and documentation.

Deliver feedback early. Don't wait until several issues have come up, or until your student has impressed you multiple times with their efficiency. Let them know right away what you think about their work.

Make a point to give positive feedback. The open source community parcels out comments all too frugally. When a student completes a task on time, and especially when they exceed your expectations, let them know! Early praise is a far better motivator than late criticism.

Don't avoid critique, but don't be a jerk. Try to put yourself in your student's shoes, and consider how you might want to hear constructive criticism. Phrase suggestions positively. If criticism is personal in nature (i.e. tone of an email, timeliness or other non-code issues), deliver it in private rather than in a public forum. When in doubt, ask for advice from more experienced mentors or from your organization's administrator.

Pro Tip: Don't let the development of your design document become your GSoC project. It's useful to set a date for completion of your strategic plan and add a "Start of Coding" milestone.

Don't Be That Guy: Don't be overly-critical of date slippage. It happens. Fanatical adherence to dates does not lead to successful project completion, nor does it make your student feel excited to contribute to your community long-term.

Evaluations

Evaluations are a critical component of the GSoC program. Both at mid-term and at the end, evaluations serve to expose process flaws, assess performance, and precipitate pass/fail decisions. Taking time to evaluate the progress and workflow of the project provides an opportunity to correct course and address underlying issues. Most importantly, it provides a structure in which to give valuable criticism and praise to the students. After all, they are supposed to be actively learning (not just working) and effective learning requires evaluation.

Unlike code critique, which happens openly on mailing lists or other public forums, performance evaluations are personal in nature. Delivering the mentor-student evaluations should be done privately. And more generally, remember the maxim: criticize in private, praise in public.

There are three types of evaluations:

Student evaluations: Each student fills out a mid-term and final evaluation pertaining to their experience and their interaction with mentors.

Mentor evaluations: Each mentor fills out a mid-term and final evaluation covering their participation and their student's performance.

Admin evaluations: Each org admin fills out a mid-term and final evaluation about their role in the program.

Program evaluations: Every participant in GSoC answers questions about how the program is operating over all.

These are provided online by GSoC at specified times with deadlines. The deadlines are important for organizing the payments to students based on the pass/fail decisions by their mentors, so you should ensure that you complete your evaluations on time.

They Won't Know Unless You Tell Them

It is important to note that other than the pass/fail status, students do not have direct access to the evaluations of their performance. Mentors are welcome to make a PDF copy of their evaluation and share it with students directly. Students are welcome to do the same, but it is not required for either the mentor or student to share their evaluations with the other. Make sure to do a thorough check in over and above the topics covered by the evaluations regardless of your choice to share them with one another.

Mentors may not have access to student evaluations. Org admins have access to all evaluations submitted by their org's mentors and students. Mentors have access only to their own evaluations, not to the evaluations submitted by their student. Thus, if you want to give feedback to your student (and you should) you need to send this feedback directly to them; you cannot rely on Google to do it for you. While this may change in the future, it is important for mentors and org admins to coordinate in assessing the student and mentor evaluations and taking appropriate follow-up action.

Evaluations should result in a direct review of the student's progress and should be conducted using a real-time communication medium. This can be by phone, your favorite voice-over-IP service, or in person. The discussion should be frank and in the context of periodic review so that the student is prepared for criticism and to work with you to revise workflows, timelines, habits, etc. Take time to explain the value of learning how to take criticism and praise in a professional setting.

When delivering reviews of student performance, be specific about both positive and negative aspects of a student's performance. Make the suggestions for change or improvement relevant to what your student is currently working on, and provide specific examples. As you prepare for your student's review, you could

write it out as though you were sending an email to frame your thoughts and to help ensure that you are providing a balanced perspective. When you finally deliver the information, make the student aware ahead of time when the review will occur. If you provide a written copy of the review, schedule time to talk about it immediately after the student reads it.

When in Doubt, Fail the Student Early

This is harsh-sounding advice. However, Leslie Hawthorn reports based on a back-of-the-envelope calculation that more than 85% of the students who are reported as marginal at or before the midterm eventually fail GSoC. Whatever problems your student is currently having, they are likely to be worse than you currently appreciate, and to get worse rather than better over time. You are not doing your student, much less yourself, a service by prolonging the agony. Most GSoC students and their mentors have a great time and get a lot done. If you are having the other kind of experience, cut your losses and try again next year.

Note that GSoC gives mentoring orgs quite a bit of flexibility and cuts them a lot of slack. In particular, if a student fails during the community bonding period, there is likely an opportunity for the organization to substitute another student, or to give the slot to another organization to do so. Students need to understand that they are being evaluated from before they are accepted to the end of the program, and that you take the GSoC experience seriously and expect them to do likewise.

But It's Not Impossible to Turn Things Around

A true story: Once upon a time there was a student with limited English. The mentor considered the communication difficulty as a potential issue, but the student was enthusiastic, and hoped to use GSoC as an opportunity to improve his English.

The community bonding went well, but during the first half of coding, progress drifted off track. The student's mentor was much busier with work than was anticipated, and the student became stuck several times with various issues. The student failed to proactively ask for help, and the mentor didn't catch this, so the student became disheartened.

At the midterm, progress was disappointing, and the project came close to failing the student. But as the mentors looked at how they might try to rescue the situation, and after discussion with the student, the project came up with a concrete plan. The student had a job in the lab for about 8 hours a week, which he agreed to put on hold until the end of GSoC. Each day the student updated a Google document with what he was working on. This included any outstanding issues which might block progress, and he would connect to IRC during the hours he worked on his project.

Additionally, a second mentor was brought in so that a mentor would always be available when the student might be working. Progress was discussed daily. The organization also made it clear that if the plan didn't succeed, that the failure would reflect badly on both the organization and the student. All those involved were happy to work to address any additional issues that might come up.

By the final evaluation, the student was almost back on his original project plan, and had completed all the required goals.

So it is possible to rescue a failing student, but you need to really consider the issues, and come up with a plan to address them which everyone involved buys into. You also need to be prepared for the project to fail and be okay with the energy you have invested not resulting in the outcome you wanted.

Mentor, Heal Thyself

This is also an important time for self-evaluation. Are you managing your time adequately? Do you know where the project is at and where it is going? Are you enforcing the deadlines you set? Are you integrating your student into your community?

Also take the evaluation period as an opportunity to get feedback from students. Is there any way that you could have helped the student more? How does the student think you be more effective as a mentor? Ask your student directly for feedback.

Upstream Integration

Every project wants to get useful code. And once you get that code, you'd love to be able to commit it to your project and make use of it! The following are some helpful hints on making that process easier on both the students and the committers to your project.

Note: We use the term 'committer' throughout this section to mean the person or people responsible for adding or merging code to your organization's authoritative source code repository. Terminology and source control methods vary so widely, that it was difficult to choose a single term. Committer seemed to cover the widest variety of situations.

Recruit Committers Early

Typically, one or more committers to your project will be involved in GSoC. For very large projects, it can be helpful to alert all committers to upcoming student submissions. You can ask for them to be a second reviewer on patches from students, or simply keep them informed of your student's work schedule. This helps them know approximately when to expect code submissions.

Get the Code into Your Repo

It's best if you can get the code committed to your project's repo as early as possible. Whether you commit this to your project's equivalent of HEAD, to a development branch or a feature-specific branch, getting student's code into a publicly available and canonical location early is a good thing.

Even if your organization is unwilling to have student code drop directly into a released or releasable version of the org's project, it is a good idea to make sure that it is captured by the organization somehow. This ensures that you and your organization has a backup copy of the student's work. Getting the student's code into an unstable version or feature branch at the start of GSoC ensures that everything of value will be captured, decreases the eventual integration burden, and provides a better mechanism for community review of student code.

Dividing Up Patches

Many organizations take student code submissions in the form of patches. It is considered best practice to keep patches confined to a single feature at a time. Sometimes this is not possible, but encouraging students to submit their changes as they are working is always better than a massive "patch bomb" at the end of the program.

You can help your student by talking with them about how to chunk up their code into reasonable submissions to the project. This should include setting time-based milestones, grouping certain features and implementation details together and requiring that tests or specifications be written first. Using pseudocode, rapid prototyping and iterative design methodologies can be helpful in structuring your student's work and keeping you in the loop.

Be sure to set aside some time to teach your students how to use your revision control system, and especially its code merge tools, during the bonding period. This is both for their benefit and to save your time. If the student already knows how to test merging their changes before submitting them, it is far more likely that less time will be spent fixing patches which don't apply.

Patch Review

Make review of submitted patches an explicit task. By doing so, you make it easier to hand some or all of the review off to another member of your community. This kind of delegation makes the student more a part of your development community, and reduces the overall work that is required from you.

Ultimately, you'll be responsible for understanding and performing a final review of the code for the student's GSoC evaluation. However, there's nothing wrong with bringing other people into the process.

Some projects have a documented review process. If you don't you can find an example at the PostgreSQL Developers Wiki at http://wiki.postgresql.org/wiki/Reviewing_a_Patch, some of which was used as a reference for this section.

Many people feel that they're not qualified to do a full review of a patch. But review includes many different tasks, and even if you can't do all of them, a reviewer can help your organization by taking on some or all of pre-commit tasks.

If you can apply a patch and you can use the new feature, you're already qualified to start reviewing it.

A reviewer does not need to guarantee some level of quality, but they do need to report any problems they find. The review is done if you think the patch is ready for in-depth review from a committer. See this patch review at <http://archives.postgresql.org/pgsql-hackers/2009-07/msg01103.php> as one example of the output a thorough review might produce. Reviews for other patches might, of course, contain different sections or for that matter, look completely different.

Questions a patch reviewer might ask include:

- Does it include reasonable tests, necessary documentation, an overview of features, etc?
- Does the patch actually implement what the author intends and work as advertised?
- Does it follow a relevant specification, RFC or the community-agreed behavior?
- Are there corner cases or failure situations the author has failed to consider?
- Does the patch slow down simple tests or other features?
- Does it follow the project coding guidelines? For an example, see <http://developer.postgresql.org/pgdocs/postgres/source.html>
- Will it work on all supported operating system and hardware platforms?
- Are the comments sufficient and accurate? Are there any comments at all?
- Does it produce compiler warnings?
- Can you make it crash?
- Is everything done in a way that fits together coherently with other features/modules?

Building a Lifetime Contributor

"In just seven days...."

The work of transforming your summer students into lifetime contributors begins on the first day. Actually, it begins even earlier with the preparation of your development community for student participation and inclusion. In as many ways as possible, you should aim to make the students equal peers by the end of the program. Actively encourage students to attend community meetings and retreats. Highlight their work within your community. Part of being a mentor is being a cheerleader.

Say thank you often: People like to be thanked for their contributions. Publicly and privately. Creating an environment of appreciation makes students want to keep contributing.

Be gentle with criticism: Remember that you are cultivating a long-time contributor. Positive, constructive comments are the most useful as well as the most pleasant to receive.

Don't airbrush your community: If you want a student to stick around, you might as well make the experience as realistic as possible. Creating a bubble for the student takes a lot of effort. When it inevitably bursts, they will be disillusioned.

Credit where due: Open source has a reputation culture, sometimes known as a reputation economy. Create a CREDITS file in your code repository dedicated to identifying those people who have contributed to your project, and make sure that they are mentioned by name in the release notes. As these contributions are credited, mention this publicly on a mailing list or in a blog post. Remember to also acknowledge contributions in public: when your student completes important project milestones announce their successes to your development community.

Present talks at conferences: Recommend that students present their work at open source conferences. If you can, offer to co-present with them and help with designing their talks. If your organization can offer travel support for the student, this is quite helpful. If you know people who do work that your student is interested in, or that your student would be impressed to be introduced to, take the time to make introductions.

Professional development: If appropriate, offer written professional recommendations to students. Your help in career advice, job searches or referrals can be invaluable for a student.

Retreats and hackfests: If your organization is having a retreat or hosting a hackfest, invite your students along! Find ways of integrating them with your group the same way you would any other contributor.

Stay in contact: Open source organizations typically sustain themselves through the personal connections developers have to each other. Many contributors consider each other friends, and communicate non-technical information to each other. Remember though, that this is a student and GSoC is their job, so when in doubt err on the side of communicating professionally no matter how informal the tone of your overall community.

Communicate early and communicate often! Try sending periodic emails to check in, pass on an interesting link or share a photo. Social networks can be useful for maintaining contact without a lot of overhead. Examples include: Identi.ca, Flickr, and GitHub. Use what works for you, and don't feel pressured to get involved with a social network. The telephone is also a valuable resource; studies of open source communications have shown that VOIP technology is used with surprising frequency in open source projects, and it is a very personal and friendly form of communication.

Don't Be That Guy: Be socially and culturally sensitive. If you and your student don't share the same cultural background, ask respectful questions so you get to know about the similarities and differences a bit more. Also, make sure to maintain appropriate boundaries in your communications. For example, a student calling in the middle of the night to ask for relationship advice is not a pleasant situation for anyone; leave those

conversations for their college friends.

The Quick Guide

The process begins with proposing projects and selecting students over about a four week period. This is followed by six to eight weeks of community bonding. The coding phase runs for 12 weeks. A final two-week phase wraps up the project.

- During the student selection phase the mentor will be responsible for helping to try to find a fit between project, student proposal and mentor. This will include defining the proposal process, assisting in the evaluation of proposals, querying the students, providing them feedback about their proposals and ultimately finding a student proposal to mentor.
- During the community bonding phase, the mentor and student will further define the student's project and prepare for development. An important part of this phase, as the name implies, is to get the student connected with both the mentoring organization and the larger open source community.
- During the coding phase the student will be implementing the project. The mentor will be providing advice not just on the technical aspects of development, but on issues related to the interaction of the student's work with the organization. At midterm the mentor will evaluate the student's progress to determine whether the student will continue with the project and be issued a payment by Google.
- During the final phase the mentor will help the student submit a code sample to Google. The mentor will also perform an evaluation of the student's work to determine whether Google should issue a final payment.

Quick Tips on Effective Mentoring

The tips in this section are no substitute for reading this book. However, they may serve as an introduction to the material herein.

Project Definition: An ideas page is the starting point for project definition. You should include projects that you are interesting mentoring for the summer, that appeal to students and to your organization.

Selecting Students: Students come with many motivations; try to understand why your applicant is applying. It is better to give up a student slot than to select a student whose performance is likely to be poor.

Communication: Engage with your student as early as possible. Integrate the student into your community and its communication channels. Make sure that communication with your student is frequent and regular.

Collaboration: Start by working with your student to set solid expectations and goals for the project. Develop a detailed project plan, with deadlines and milestones. Be prepared for slippage, and be willing to revise the plan as necessary, especially at midterm. Student learning is a primary goal of GSoC, so try to provide a learning experience.

Evaluation: Keep your student evaluations objective; base them on your project plan. Keep in mind that it is better to fail a student earlier rather than later---data shows that students doing poorly at midterm rarely complete a GSoC project. Make sure your students hear your evaluations: deliver praise in public, and criticism in private.

If you follow the above advice, and the rest of the advice given in this book, you have a good chance to have a successful experience. Positive outcomes might include recruiting a permanent developer to your community, developing a lifelong relationship with your student, and helping the open source community grow and thrive.

The History of GSoC

Google Summer of Code began in 2005 as a complex experiment with a simple goal: helping students find work related to their academic pursuits during their school holidays. Larry Page, one of Google's co-founders, was pondering the age old problem of scholastic backsliding: students work hard and learn a great deal during the academic year, but without the right employment opportunities or other pursuits outside of school, technical skills atrophy rather than get honed and expanded. Larry wanted Google to help solve this problem.

The obvious solutions failed on geographical grounds. If a student wasn't in the optimal location, obtaining a useful internship could be difficult if not impossible. Finances were also a problem; many internships are low paying or entirely unpaid, making it difficult for students to take the right job while still paying pay the bills. Finally, even if a job was available in a technical field, it would not necessarily introduce a student to the broad set of skills required to do software development well. For example, creating a website for a local non-profit requires technical skill and would no doubt be personally gratifying, but wouldn't necessarily require using an IDE, checking into source control, or creating tests.

The perfect answer came from encouraging students to participate in open source projects. Open source development occurs online, both solving the geography problem and giving students the chance to work in a globally distributed team. Working on an open source project provides exposure to the entire software development process and toolchain. Students could enjoy the added benefit of having body of reference work available to future employers or committees. Even better, students would get the chance to work on a codebase under active development rather than a lab project or other single use assignment.

A cash stipend from Google allowed students to focus on their development work rather than getting a job unrelated to their academic pursuits. The final part of the puzzle was finding projects who were excited to find new contributors and to provide helpers to get these new folks up to speed both technically and socially. The first year 40 projects participated; 400 students began the experiment.

In 2009, the fifth Google Summer of Code wrapped up with the best results yet. More than 85% of the 1,000 student participants in the program successfully completed their projects. Best of all, most of the organizations participating over the past five years reported that the program helped them find new community members and active committers.

You can find more information about each year of Google Summer of Code on the program statistics page of the GSoC Knowledge Base: <http://code.google.com/p/google-summer-of-code/wiki/ProgramStatistics>

Additional Resources

We've collected our favorite and most useful resources specific to GSoC here.

General Resources

If your organization has participated in GSoC previously, chances are there are mailing lists already set up and useful information in their archives; take a moment to look through them, especially around the launch times (February to March), community bonding period (April) and evaluation times (mid-July and end of August), for more details. The archives of the program mailing lists, particularly the private mentors list (below), are also quite useful.

No matter what, you want to take a look at the Program Frequently Asked Questions each year to make sure you have a good idea of the rules for the program for both yourself and your students. There's a wealth of information included in the FAQs each year, even for experienced participants. You can always find the latest information, including a link to the FAQs, at <http://code.google.com/soc/>.

Additionally, these resources are quite helpful:

Program IRC Channel: Several knowledgeable folks hang out in #gsoc on Freenode and would be happy to give you a pointer in the right direction.

Blog Posts: You can find material related to GSoC on the Google Open Source Blog at <http://google-opensource.blogspot.com/search/label/gsoc>. Your project may have a blog or newsletter where GSoC information was published in the past, as well.

Knowledge Base: If you're looking for advice for mentors or students, program promotional materials, presentations about GSoC, etc., start with the knowledge base <http://code.google.com/p/google-summer-of-code/>, particularly the wiki <http://code.google.com/p/google-summer-of-code/wiki/WikiStart>.

Mentor Summit Wiki: Google has traditionally held an annual mentor summit after GSoC wraps up each year. During the summit, many great discussions are held on all sorts of topics regarding the program and open source overall. Check out the summit wiki for session notes and to do some further collaboration. Instructions for getting a login account on the wiki are available on the private mentors list, or ping the program administrators for help. Anyone can view the wiki's contents at <http://gsoc-wiki.osuosl.org/>.

List of Organizations: Each year, the community creates a list of categorized list of mentoring organizations. You can find it linked from the Knowledge Base: Advice for Students page. <http://code.google.com/p/google-summer-of-code/wiki/AdviceforStudents>

Mailing Lists

There are four program mailing lists.

Announcement Only List: For announcements from Google's program administrators only. Used infrequently. <http://groups.google.com/group/google-summer-of-code-announce>

Program Discussion List: Open subscription list for the program. General talk about the program, light traffic except during the launch phase of the program each year. Typically this list is not hugely relevant except just prior to the announcement of accepted organizations and accepted students, as neither organizations nor students have access to the private lists until acceptance unless they have previously

participated in the program. It is always excellent for you to stop by and encourage a newbie, though, so please don't totally ignore this list.

<http://groups.google.com/group/google-summer-of-code-discuss>

Students List: Private, invite-only list; students are subscribed to the list soon after they are accepted into GSoC. Successful student participants from previous years and students currently working on GSoC are subscribed to this list. Students who are dropped from the program are also removed from the list. While this list is supposed to give students a private place to discuss anything and everything so they aren't worried about looking silly elsewhere, more often than not the list traffic is mostly discussions of tracking numbers for shipments, tax forms and grumbles about t-shirt loss.

<http://groups.google.com/group/google-summer-of-code-students-list>

Mentors List: Private, invite-only list; mentors are subscribed to the list after their organization is accepted into the program and they register as mentors in the GSoC online system. This list is higher traffic at the beginning of the program and around the times of evaluations. Some great advice can be found on this list and in the archives, but it can also be noisy at times.

<http://groups.google.com/group/google-summer-of-code-mentors-list>

Books

Producing Open Source by Karl Fogel. Excellent guide to Open Source development. Its available free online.

<http://producingoss.com/>

Google Summer of Code Mentors Guide

<http://www.flossmanuals.net>

Associated Projects

Teaching Open Source

<http://teachingopensource.org>

irc : freenode #teachingopensource

What to Do When the Unexpected Happens?

Contact your Organization Administrator: He or she can help you figure out what to do next or contact Google for more help.

Talk to Google's Program Administrators: They have plenty of experience with all the corner cases and strange issues that can arise during GSoC. Email ospoadmin@gmail.com for help if you can't find one of the program admins in #gsoc on Freenode.

Glossary

+1

The shortest way in the geek world to say "I agree with this" or "This is a great idea". It is often used when others have already fleshed out the details and a consensus of how many agree/disagree with the sentiment. It is worth noting to your students if your project uses this as a voting signal so they do not accidentally comment on issues when, as newbies, they should be observing rather than commenting/voting.

-1

The opposite of +1. Often accompanied by an explanation why, if you are lucky.

Committer

An individual who has special rights in an open source project. While the scope of this term varies by project, the general idea is that this individual is able to check in source code to the project's main repository.

Community Bonding Period

The period of time between when accepted students are announced for a particular year of GSoC and the time these students are expected to start coding. This time is an excellent one to introduce students to the community, get them on the right mailing lists, etc. See the "Mind the Gap" section for more details.

DVCS

Distributed version control system. A version control system that does not require talking to a centralized server.

FLOSS

Free/Libre Open Source Software. Likely the most inclusive acronym to describe the software produced for GSoC.

GSoC

Google Summer of Code

IDE

Integrated Development Environment

IM

Instant Messenger

IRC

Internet Relay Chat

ISP

Internet Service Provider

JFDI

Just Fabulously Do It. Use your imagination. Ask for forgiveness, not for permission. :)

Mentor

Someone who helps a student with their project proposal. See the What is GSoC section for more details.

Organization

An open source, free software or technology-related project that mentors students for Google Summer of Code. Also known as a mentoring organization.

Organization Admin (Org Admin)

Cat herders for each open source project participating in the program. Often abbreviated to org admin. See the What is GSoC section for more details.

Program Administrator

Google employees who run the program. See the What is GSoC section for more details.

RTFM

Read The FLOSS Manual ;)

Secondary Mentor

A person who helps out a student's assigned mentor. At time of writing this manual, the GSoC online system only allows one mentor to be officially assigned to a student proposal, as one person must be responsible for submitting evaluations, etc. However, it is quite common to have multiple mentors for one student.

SMOP

Simple Matter of Programming

Summer

Not so much a season as a state of being. While the program is run during the Northern Hemisphere's Spring and Summer, the "Summer" in Google Summer of Code is actually a play on the "Summer of Love."

TDD

Test Driven Development

Use Case

A use case describes what a user can do with a particular software system.
http://en.wikipedia.org/wiki/Use_case

Waterfall Model

A sequential software development process.

License

Note : "Google Summer of Code" (GSoC) is a trademark of Google Inc.

All chapters copyright of the authors (see below). Unless otherwise stated all chapters in this manual licensed with **Creative Commons SA-BY 3.0**

Authors

THE QUICK GUIDE

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009

Alexander Pico 2009

Anne Gentle 2009

Bart Massey 2009

Jonathan Leto 2009

ABOUT THIS MANUAL

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009

Alexander Pico 2009

Anne Gentle 2009

Jennifer Redman 2009

Jonathan Leto 2009

Leslie Hawthorn 2009

Olly Betts 2009

selena deckelmann 2009

COMMUNITY BASICS

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009

Alexander Pico 2009

Jennifer Redman 2009

Jonathan Leto 2009

Olly Betts 2009

selena deckelmann 2009

CREDITS

© Google Inc And The Contributors 2006

Modifications:

adam hyde 2006, 2007, 2009

EVALUATIONS

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009

Alexander Pico 2009

Bart Massey 2009

Leslie Hawthorn 2009

Olly Betts 2009

selena deckelmann 2009

OPEN SOURCE CULTURE

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009

Alexander Pico 2009

Jennifer Redman 2009
Jonathan Leto 2009
Leslie Hawthorn 2009
Olly Betts 2009
selena deckelmann 2009

SELECTING A STUDENT

© Google Inc And The Contributors 2009
Modifications:
adam hyde 2009
Alexander Pico 2009
Bart Massey 2009
Leslie Hawthorn 2009
selena deckelmann 2009

GLOSSARY

© Google Inc And The Contributors 2009
Modifications:
adam hyde 2009
Jonathan Leto 2009
Leslie Hawthorn 2009

HISTORY OF GSoC

© Google Inc And The Contributors 2009
Modifications:
adam hyde 2009
Alexander Pico 2009
Jonathan Leto 2009
Leslie Hawthorn 2009

WHAT IS GSoC?

© Google Inc And The Contributors 2006
Modifications:
adam hyde 2006, 2007, 2009
Alexander Pico 2009
Anne Gentle 2009
Jonathan Leto 2009
Leslie Hawthorn 2009
Olly Betts 2009

BUILDING A LIFETIME CONTRIBUTOR

© Google Inc And The Contributors 2009
Modifications:
adam hyde 2009
Alexander Pico 2009
Bart Massey 2009
Jennifer Redman 2009
Leslie Hawthorn 2009
selena deckelmann 2009

MIND THE GAP

© Google Inc And The Contributors 2009
Modifications:
adam hyde 2009

Alexander Pico 2009
Jennifer Redman 2009
Jonathan Leto 2009
Leslie Hawthorn 2009
Olly Betts 2009
selena deckelmann 2009

MANAGING THE PLAN

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009
Alexander Pico 2009
Bart Massey 2009
Jennifer Redman 2009
Leslie Hawthorn 2009
selena deckelmann 2009

ADDITIONAL RESOURCES

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009
Alexander Pico 2009
Jennifer Redman 2009
Leslie Hawthorn 2009
selena deckelmann 2009

WHAT MAKES A GOOD MENTOR

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009
Alexander Pico 2009
Jonathan Leto 2009
Olly Betts 2009

DEFINING A PROJECT

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009
Alexander Pico 2009
Bart Massey 2009
Jennifer Redman 2009
selena deckelmann 2009

SETTING EXPECTATIONS

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009
Alexander Pico 2009
Anne Gentle 2009
Bart Massey 2009
Jennifer Redman 2009
Jonathan Leto 2009
Olly Betts 2009
selena deckelmann 2009

BEST PRACTICES

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009

Alexander Pico 2009

Jennifer Redman 2009

Olly Betts 2009

selena deckelmann 2009

WHY GSoC MATTERS

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009

Alexander Pico 2009

Jennifer Redman 2009

Jonathan Leto 2009

UPSTREAM INTEGRATION

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009

Alexander Pico 2009

Bart Massey 2009

Jonathan Leto 2009

Olly Betts 2009

selena deckelmann 2009

WARNING SIGNS

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009

Alexander Pico 2009

Jennifer Redman 2009

Jonathan Leto 2009

Olly Betts 2009

WORKFLOW

© Google Inc And The Contributors 2009

Modifications:

adam hyde 2009

Alexander Pico 2009

Jonathan Leto 2009

Olly Betts 2009

selena deckelmann 2009



Free manuals for free software