Note: The most recent version of this guide can be found at opensource-events.com. Feel free to share widely. Have feedback? Please leave a comment.

The Open Source Project Guide: Hackathon/Sprint version

by Shauna Gordon-McKeon of OpenHatch

In order to get the most out of a workshop, hackathon or sprint, it's important to plan ahead and identify goals and tasks for helpers, as well as possible stumbling blocks. We've created the following guide to help you plan. We've used our own project - OpenHatch - as an example below.

The checklist version of this guide can be found at the very end of the document.

Contents

1. Defining goals

You want to be able to state clearly your goals for the event, as this gives your group something to work towards. You can start by asking:

- **What is the overall goal of your project?**

  You want a short (1 paragraph or less) answer to this question which you can use to

entice potential contributors to your project. Details are great, but at this point, you shouldn't need to be too technical.

For example:

> *OpenHatch's goal is to make the free software/open source community more welcoming to newcomers. To do this, we provide curricula and logistical support for running "Intro to Open Source" workshops, a website with open source tools, "training missions" and a volunteer opportunity finder, and several other projects in progress.*

- **What do you want to accomplish at this event?**

Think about what, specifically, you'd like to get done at this event. You can break these down by elements of your project, if you have more than one. It should be clear how these event goals contribute to the overall goal of your project. At the same time, these are not "tasks" - it should be necessary to break these goals down further in order to accomplish them.

It's useful to phrase these in terms of "Base" and "Reach" goals. Having modest base goals gives you something to celebrate at the end of the event, while adding reach goals lets you plan for the exciting scenario of having a large and/or effective team that's able to accomplish a ton.

In general, it's better to have too many goals than too few, but make sure you prioritize them. When you get to the task-breakdown part of this guide, focus on doing a thorough job with each individual goal before moving on to the next one.

For example:

1. *Make a new training mission:*
    a. *Base goal: Pick a skill to create a new training mission around, and design what the mission will look like. Create a mock-up of the mission.*
    b. *Reach goal: Implement the mock-up, and user test it on volunteers from the event.*
2. *Clean out issue tracker:*
    a. *Base goal: Go through tracker and label issues by what type of "cleaning" they need (Does a bug need to be verified? Does a patch need to be tested? Does the feature need to be attached to a milestone?)*
    b. *Reach goal: Address the added labels. (Verify bugs, test patches, assign features, etc.)*

2. Project setup

In our experience, project setup is the single biggest barrier to participation. We've seen (and run!) events where the majority of time spent by participants was on setting up their development environment and becoming acquainted with the project. Documenting and improving the process beforehand can save everyone a lot of time and energy. If you know that a part of your project will inevitably be time-consuming, make sure participants know to expect that. Consider contacting participants ahead of time and trying to do some of the set-up asynchronously before the event.

(If you're a very new project, read the rest of this section with an eye for what you'll want to document as you build.)

All of the information below should be documented in a README at the top level of your source repository. Other places put the info include a "Want to contribute?" section of your project website, and/or you can include a link to the README in the signature of your mailing list or in the status bar of your IRC channel.

- **How to find the project's community/maintainers**

  Contact information should be displayed prominently, as you may have remote contributors, or contributors who want to start before the event. Make clear how to get in touch with you in case they run into trouble with any subsequent steps. Types of contact information can include:

  - A link to your mailing list.
  - Your IRC channel name and server (including link to IRC installation guide and link to webchat version).
  - Social media accounts such as Identica, Twitter, Facebook, if your project has them.
  - Maintainers' personal contact information, if you feel comfortable giving it out.

  For example:

  > *OpenHatch has two places for contact info, which we try to keep updated and consistent with each other. There's our contact info in the documentation, primarily linked to from our source code repository, and our contact info in the wiki, primarily linked to from the website's main page.*

- **The project's structure**

  Describe the basic structure of your project. What are the biggest pieces and where are they located? How do those pieces interact? Then break each piece down.

  You don't need to talk about every file or subdirectory of your project, but you don't want

to assume that what a script does, or how the files in a directory interact, or what language a part of your project is in is obvious to a newcomer. Making those assumptions turns getting access to *you* into the bottleneck resource for working on your project.

Depending on the size and complexity of your project, this can be a pretty big undertaking. At OpenHatch, we're still working on getting the full structure completely documented. We recommend doing a "top level" explanation of your project's structure, and then going into detail about areas that people commonly work on (or are likely to work on at sprints or hackathons.)

For example:

> *A description of the top-level structure of the OpenHatch project can be found at Project Overview. A description of the structure of OH-Mainline (the repository that runs our website) can be found here.*

- **How to set up a local ("development") environment**

  In order to contribute to your project, people will usually need to set up a local version of the project where they can make and test changes. The more detailed and clearer your installation/development guide, the better.

  Installation will often differ depending on the operating system of the contributor. You will probably need to create separate instructions in various parts of your guide for Windows, Mac and Linux users. (If you only want to support development on a single operating system, make sure that is clear to users, ideally in the top-level documentation.)

  Here are common elements of setting up a development environment you'll want your guide to address:

  - Preparing their computer
    - Make sure they're familiar with their operating system's tools, such as the terminal/command prompt.
    - If contributors need to set up a virtual environment, access a virtual machine, or download a specific development kit, give them instructions on how to do so.
    - List any dependencies needed to run your project, and how to install them.
  - Downloading the source
    - Give detailed instructions on how to download the source of the project, including common missteps or obstacles.
    - If there are multiple versions of the project, make clear which version they

should download.
- How to view/test changes
    - Give instructions on how to view and test the changes they've made. This may vary depending on what they've changed, but do your best to cover common changes. This can be as simple as viewing an html document in a browser, but may be more complicated.

For example:

> *You can see OpenHatch's version of this information in our Installation Guide. Instructions on how to contribute changes can be found in handling patches, which is linked to in the installation guide. Instructions for testing changes can be found in the documentation for different changes one might make (for instance, Documentation changes.)*

- **Contributing changes and feedback**

  How do contributors contribute their changes to the project? Do they submit a pull request via Github? Do they generate a patch and attach it to an issue in an issue tracker? Make sure this information is explicitly provided.

  For example:

  > *OpenHatch's guide to submitting changes can be found here.*

  It's also useful for people to know how they can give feedback/report bugs to the project. If your project doesn't have an issue tracker, consider creating one.

  For example:

  > *Issues with the Open Source Comes to Campus project can be reported here. Most other issues with OpenHatch can be reported here.*

- **Verify that this documentation is complete/effective by testing on individuals who haven't used or contributed to your project before.**

  Find at least one person for each operating system to read your documentation and attempt to install, make and test changes, and contribute the changes to the project. (These can be simple, fake changes or, if your tester is willing, actual tasks.)

  Make sure that any problems which arise during verification are added to the documentation. Once the documentation has been verified, and a line to the top of your guide which states what was verified and when.

For example:

> *Development environment instructions tested successfully on Ubuntu 12.04 (on 2013-10-03), Mac OS X 10.8 (on 2013-10-01) and Windows XP (in Jan 2005).*
>
> *You can see OpenHatch's version of this here.*

3. Defining tasks

Let's return to the event goals we talked about in the first section. For each of these goals, we should be able to break down the steps that need to be taken to reach them into discrete tasks. These tasks should include a "plain english" summary as well as information about where to make the changes (for instance, which files or functions to alter). We recommend including a list of needed skills (e.g. "design skills", "basic Python", "English fluency", "familiarity with the command line") and tools (e.g. "Mac development environment"). It's also useful to include an estimate of how much time the task will take, to label some tasks as higher or lower priority, and to mark where one task is dependent on another.

We recommend using a wiki or similar planning document to keep track of tasks. OpenHatch has a task-tracker that we use for our events - you are welcome to fork it and customize it for your project/event, although you might want to wait as we'll be making some big improvements soon. Something as simple as an etherpad should also be just fine.

For example:

> *Reach goal: Address the added labels. (Verify bugs, test patches, assign features, etc.)*
> *Task 1: Verify Bugs*
> - *Skills/tools needed: Strong English language skills, ideally familiarity with virtual machines to test on multiple OSs.*
> - *Estimated time: ~15 minutes set up, ~20 min per bug (high variance)*
> - *Get started: <Download the development environment> and make sure you can run the project. Make sure you have an account on <the issue tracker> and are familiar with how to add comments or change labels.*
> - *For each bug: Try to reproduce the bug. Record the results in a comment, including your operating system type and version #. If possible, test on multiple browsers. If there are recent comments covering all three major OSs, add label to bug "ready_for_maintainer_review".*

Once you've created this list you can use it when recruiting/assigning participants to your project. (If event organizers have the bandwidth to recommend participants to projects based on needed skills, they will be very glad for this information.)

4. Follow-up

Contributors may not be able to finish the tasks they are working on during the event. Or they may want to continue participating in the project by working on other tasks. Thinking ahead about how you will follow up on the event makes it easier to exchange information with participants and plan the direction of your project.

We recommend asking each participant to answer the following questions about the tasks they worked on. Giving them this list at the start of the event will help them document what they're doing as they go along.

For each task you worked on, please answer:

- What task did you work on?
- Please briefly document your workflow. What steps did you take, in what order, and why?
- Where can I find the work you did at the event? This includes code, documentation, mock ups, and other materials.
- If you created any accounts for the project, please list the site, account name, and password.
- What obstacles did you encounter when working on this task? Do you have any feedback for me to make the process better for future contributors?
- Would you like to stay involved in this project? If so, in what capacity?

If there is strong enthusiasm for continuing to work, we recommend planning a follow up meeting at the event. If you're all local, try setting a date 2-3 weeks after the event for you and your team to meet at a local coffee shop, coworking space, or project night. If you're remote, set a date to meet on IRC or a google hangout. At the very least, get email addresses and/or other contact info from anyone interested in following up, and contact them within 48 hours thanking them for their help at the event.

5. Checklists

That's a lot of advice! To help you keep track of each step, we've created two checklists for you. The detailed version includes all of the advice above. The quick and dirty checklist includes the elements of the above document which we think are most important. These represent the minimum needed for a project to be included at an OpenHatch-run event. We highly recommend you follow the full checklist.

DETAILED CHECKLIST

Defining Goals

- ❏ Write one paragraph description of overall project goal.
- ❏ Define “base” and “reach” goals for this particular event.

Project Setup

- ❏ In README, document:
  - ❏ How to contact maintainers/project community
  - ❏ Basic project structure
  - ❏ How to set up the development environment
    - ❏ Preparing the computer/installing dependencies
    - ❏ Downloading the project
    - ❏ Viewing/testing changes
  - ❏ How to contribute changes
- ❏ Verify that your documentation is clear, ideally for Mac, Windows & Linux individually

Defining Tasks

- ❏ Define tasks for contributors to work on, including for each:
  - ❏ a brief summary
  - ❏ where to make changes
  - ❏ skills and tools needed
  - ❏ optionally: estimate of time the task will take
  - ❏ optionally: priority of task
  - ❏ optionally: dependencies
- ❏ Create a resource such as a wiki for displaying and tracking the tasks

Follow Up

- ❏ Create/modify a list of questions to ask participants to capture their knowledge/feedback
- ❏ Gather contact information from participants interested in following up

QUICK AND DIRTY CHECKLIST

Defining Goals

- ❏ Write one paragraph description of overall project goal.

Project Setup

- ❏ In README, document:
  - ❏ How to contact maintainers/project community
  - ❏ Basic project structure
  - ❏ How to set up the development environment
    - ❏ Preparing the computer/installing dependencies
    - ❏ Downloading the project
    - ❏ Viewing/testing changes
  - ❏ How to contribute changes
- ❏ Verify that your documentation is clear for at least one of: Mac, Windows & Linux

Defining Tasks

- ❏ Define tasks for contributors to work on, including for each:
  - ❏ a brief summary
  - ❏ where to make changes
  - ❏ skills and tools needed

Follow Up

- ❏ Provide your contact information for participants interested in following up