Oops, I made an open-source art tool!

A guide for tool-makers, by @galaxykate (Kate Compton)

So you made a tool for people to make art! Perhaps people have started making art with your tool. What do you do next?

This zine is intended to help people making open-source tools for art (OSTAs) figure out how to manage their projects and communities, in a way that works for them.

Tools for art are a covenant between the creator of the tool, the people making art with it, and the people contributing to the tool. Everyone has implicit responsibilities towards everyone else, but when we break or betray those responsibilities, feelings get hurt and work is lost. Likewise, a community can develop when everyone feels like their contributions are valued, and that they value the contibutions of the other members.

Open source art tools are different than some other forms of software development, because people are using it to create and run artwork that is deeply important to them. That can be exciting and rewarding, because they will show you all the wonderful things your tool can be used for. It is also terrifying, when you realize that you may have to think about whether their magnum opus can still run in 10 years! Your users spend their **time** and **love** to make art: it is up to you to respect both!

So, how do you run an open-source project that works for you, but also for artists, and contributors? Will you ask for lots of help building the source code, or do you only want people to make new art with your tool, while you control the code? Will you have a community?

Contributors

Who are my contributors? Open source has historically though of "contributors" as people who:

- modify or improve the core code of the project
- add extensions or features to the project

- write documention for the language

Two of these require coding skills, and one requires coding knowledge, so we often think of "contributors" as people who make code modifications to a central repo. But these aren't the only ways to contribute to open source art tools!

Recently, I compared notes with some other art tool makers; here some of the *many* things that contributors do in our communities:

- Translate tutorials, documentation or UI to other languages or needs/accessibilities
- Make new forms of tutorials: recipe books, comics, or zines
- Write tutorials and and run workshops for their own communities (Unity devs, activists, archaeologists, Arabic speakers, middle school teachers, Harry Potter fanfic writers, etc)
- Port it to other languages, or make tools to connect it to other platforms (Twincery)
- Create tools to help visualize or debug 'programs' (linters) or host them (CheapBotsDoneQuick)
- Make a platform to host and share projects (OpenProcessing, Glitch, Scratch, ShaderToy)
- Curate and signal boost projects made with that tool (botwiki, r/twinegames/)
- Keep old projects alive and running and accessible after many years (http://collection.eliterature.org/)
- Secure funding and organize people to do any of the above (The Processing Foundation)
- and not least: MAKE ART!

All of these folks contribute to the tools they work with. Sometimes they coordinate with the original toolmaker; often they don't (CBQD was made without my help or prior approval, but it's one of the best things about Tracery).

Not all open-source art tool work happens in the Github repo!

Communication

How do you communicate with users and contributors? There may be a number of different methods:

- **Github page:** good for people intending to use your tool, or potentially contribute.
  - Good: has basic community features & top-notch code and issue management.
  - Bad: may be intimidating to non-technical users, doesn't allow embedded examples

or other custom functionality

- **Home page:** (i.e., tracery.io) A good landing site for non-technical users. This is a good place to tell people, in general, about what your tool does and how its used (*think: if a journalist landed here, could they write an article about your tool?*)
    - You may have a gallery of work, or tutorials here.
    - You might even have an online editor, simulator, or artwork hosting.
- **Social media:** Should your tool have its own social media presence, or even just a hashtag? This can be useful to talk to users, to notify them of new releases, or to signal boost new art that they make. Can users find other users who use the tool?
- **Forums:** Whether a slack or a reddit or mastodon or whatever comes next, your users may want a place to talk informally about what they are making and share tips. Do you want to control and moderate this? Maybe not. Its ok for users to self-organize their own spaces.

Your front page

People will come to your Github or project page for two reasons: they want to *use* your software, or they want to *contribute* to it. Have two sections, a **contributor** and a **user** section. You can put either one first, but add a very visible link to get to the other.

User section

- Welcome!
- Statement of purpose and values (what problem do you solve? Or, what does this tool do?)

- Examples: a few examples of "normal" things you could do with this tool. The example here is not to be a virtuoso, but to show new users, through a couple of clear examples, what the tool is **for**.

- Documentation (see below for more about types of documentation):

    - Recipes
    - Artworks people have made
    - Version change history and previous version downloads
    - Resources
        - tutorials, videos, community-centric guides
        - ports, external libraries and plug-ins

    - Directory-style library documentation (list o' functions!)

- Project future
    - How often will you update? "Probably never" is an answer, too! It is better to be honest than optimistic
- Milestones: what features are you working on? What order will they come out in?
- Dreams: where do you see this project going in the future?
- Backwards compatibility:
    - should users update old artworks to the latest version? What risks are there?
    - Will you be commiting to full backwards compatibiility in the future? For all the tool, or just core functionality?
    - What features are at risk of being deprecated?

Contributor section

- Welcome!
- Statement of purpose and values (do you value long-term stability or new features? Ease-of-use or power-users?)
- Code of conduct, if you want one (what are your values as *a community of contributors?*)
- How to contribute:
    - **What kinds of contributions aren't you accepting**: core systems that you need to keep control over, things that go against your design principles, things that you tried and they made the experience worse
    - Have a **"jobs that need to be done"** section. This can help users pick out ways to contribute that are useful. These can be:
        - *general categories*: i.e, tracery always needs new modifiers
        - *specific*: self-contained features that you want but don't have time or ability to implement
        - *social/community tasks:* this is another place to remind contributors of all the ways (tutorials, workshops translations, etc) listed above that they can help your project.
            - if there are workflows for contributors to know about (e.g., how to add a new translation language), write a brief tutorial!
            - don't ask your contributors to do detective work to figure out how to contribute if you can help them out instead.

When a contributor starts working on your project, you need to build trust with each other. Do you trust them to do good work, that you agree with, and finish what they start? Do they trust that you will include and value their contributions? Some new contributors have also never done open-source work before, and they need to build trust in themselves that they can do things like pull requests and merges without breaking things. Git is scary!

New contributors: Is this a project where you want to encourage participation by contributors who have never made a github pull request?

Consider adding a "new to contributing" section with a "quest structure" to guide very novice contributors:

- **"Level 1 Quest:** Add your name to the plaintext list of contributors"
  - This teaches them about making a pull request.
  - It also proves to them (and you) that they can do so safely * (did they accidentally delete anyone else's name, did they follow the formatting?)
- **"Level 2 Quests":** non-code contributions that can be done *safely*
  - Make a tutorial and link it on the tutorials section (or link one that someone else made)
  - Create and add a recipe
  - Find and add links to new works using the tool

Types of documentation

There are many types of documentation that your language might need. (note: these are inspired by the talk **Humanizing Your Documentation** by Carolyn Stransky). Users and contributors come to your tool with lots of different needs. Different kinds of documentation help users solve different kinds of problems.

Recipes

What are the most common things that users want to do with your tool? Recipes are a way to tell the user quickly what they want to know, **and** tell them about features that solve common issues *(Are users not finding and using your best feature? Make a recipe!)*

One should be a **"hello world"** recipe: the minimum list of steps to get something to show up on screen (like which file to download and how to install it!). Other recipes might be for basic actions, or more advanced features.

For Three.JS, these might be:

- **HELLO WORLD:** place a camera, light, and an object in a scene, and set a rendering canvas
- animate an object
- add a texture to an object
- create dynamic geometry

For Tracery, it might be:

- **HELLO WORLD:** import the Tracery library in javascript, use it to create a grammar, add the modifiers library to it, and generate a piece of text
- use modifiers to do plurals and a/an prepending ("an assortment of zebras")
- push several sets of rules at the same time (i.e. to set they/them/their pronouns)

Some recipes are **tour guides** for people starting at a certain place. Processing.JS has a quickstart guide for Processing users *and a separate one* for Javascript users.

You can also make recipes for common errors! Recipes should cover whatever people google for. That will include common errors, (e.g: "RECIPE 5: Your console says `Failed to load resource: net::ERR_BLOCKED_BY_CLIENT` ")

Works in the wild

What have people used this tool for?

- This is a good place to celebrate users' work (and show users that you see and appreciate it)
    - Very often, creators like to be "featured", but it is best to ask permission before signal boosting them!
- You can also use it to showcase ways of using the tool that a new user might not have thought of. What would inspire a new user?
    - You may not know, so consider highlighting the **range** of uses, both common and unexpected

Resources

What other extensions, ports, tutorials, and translations have people made for your tool? You can use this section to highlight some of the communities that your tool has been useful to (e.g. The Programming Historian's twitterbot tutorial for historians:

https://programminghistorian.org/en/lessons/intro-to-twitterbots, or the guide to using Tracery in middle school: http://www.ohiofi.com/blog/how-to-program-twitterbots-as-an-intro-to-computer-science/)

Version history

Tools change over time. Language syntax evolves, tools get new features, and sometimes new updates break existing artworks. What do you need to tell users?

- a history of when major changes happened, especially if syntax changed
- what users need to know about, like if function names changed, or your Hello World recipes changed.
- historical releases. This is very important for people keeping old software running. It can be a great help if you
- There are Serious Thoughts on software versioning, you can read more about them here (https://robots.thoughtbot.com/maintaining-open-source-projects-versioning)

Traditional documentation

When people say "documentation", this is usually what it means: a big list of **all** the public functions and classes and features in your tool. Sometimes a user just needs to look up a function in a directory, or see a list of everything that is available to them!

I won't cover writing documentation here, but there are resources to help you figure out what's most important to include (e.g. https://www.writethedocs.org/guide/writing/beginners-guide-to-docs/)

A few last features to consider

Some things will make your life easier, by making the ives of your contirbutors and creators easier.

- **Keep all UI text in a separate file so that translators can edit a single place.**
    - bonus version: keep all UI text in a spreadsheet with columns for each language
- **Its safer and easier for contributors to edit non-code**, or well-structured data objects that can be linted (note: tell users where the linter is!) For example, if possible, keep contributor-modifiable files in formats like:

    - JSON
    - HTML
    - .txt or .md files
    - (a note: in the Sims, all object behavior was stored in LUA scripts. This allowed non-technical designers to design behavior during development. It *also* made it so that community members could easily mod objects, sparking a huge and long-lived modder community!)

- **Think about the future!**

    - In a year, 5 years, or 20 years, will these projects still run?
    - What dependencies are there (CDNs, API calls) that might go down in the future? Can you make code that degrades gracefully, losing functionality without fully breaking, even when dependencies disappear?
    - Using an open human-readable file format for user artworks (like JSON for Tracery grammars) can keep files readable for years to come, even if the main readers go down.
        - If the version of the tool used to interpret/run the art is included *in the source* of the art, this can help future curators revive and emulate the art in the future. (the Electronic Literature Organization does this for past works, will they be able to revive your tools artworks as well?)

- **Hey, can I get paid for this?**

    - There *are* ways of supporting yourself on open-source tool development
    - Some people give the library away free, but charge for consulting with power users
    - Some have a Patreon or other crowd-funded income that they link to their open-source work (I do)
    - Suprisingly, people inside of companies often *want* to pay to support the tools they use.
        - **But**, *they usually don't have a way to expense a chartiable "donation" to the developer*
        - **The hack:** some tools have a "golden pumpkin", a purchasable version of the software (sometimes with multiple price tiers). The dev gets money, and the user funds software development by "purchasing" software.
    - **Grants:** There may be grant money available for tool development. The NSF and the NEA (for USians) may have grants available, and other countries may have arts or science funidng available too!
        - All funding agencies like to see the "reach" their money gets. If you can track

users, downloads, number of artworks made, or number of people seeing the artwork, they *really like* that data.

Resources

A few references that were helpful to me in developing this:

- **A Code of Conduct for Open Source Projects** https://www.contributor-covenant.org/
- **Making Night in the Woods Better with Open Source**, Jon Manning (https://www.youtube.com/watch?v=Qsiu-zzDYww)
- **Humanizing Your Documentation**, Carolyn Stransky https://www.youtube.com/watch?v=FvPXMUuUCS4

Thanks: much gratitude to the many people who have helped me with ideas in this document, including but not limited to Golan Levin, Rebecca Fiebrink, Daniel Shiffman, Ricardo Cabello, and OpenProcessing members.

A Note: I, Kate Compton, have done none of these well. Tracery has succeeded due to many of the principles of this document. Many of the pronciples of this document are learned from the ways that Tracery succeeded, and some prinicples are learned from things I wish I'd done well. Every open source project is different, and limited in its own way. It is only on us to strive valiantly toward a more perfect project)