Documentation and Usability

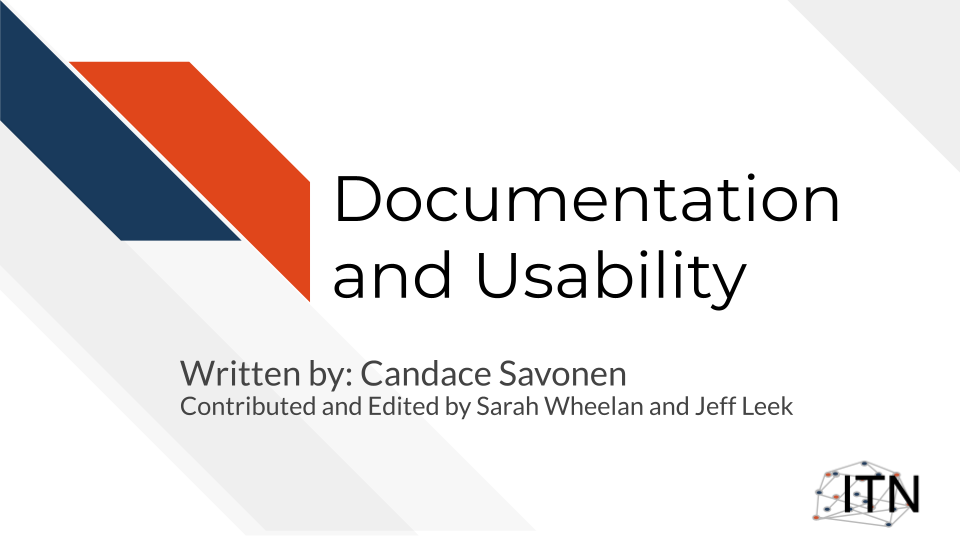
2022-03-04

Table of Contents

# About this course

This course is part of a series of courses for the [Informatics Technology for Cancer Research (ITCR)](https://itcr.cancer.gov/) called the Informatics Technology for Cancer Research Education Resource. This material was created by the ITCR Training Network (ITN) which is a collaborative effort of researchers around the United States to support cancer informatics and data science training through resources, technology, and events. This initiative is funded by the following grant: [National Cancer Institute (NCI)](https://www.cancer.gov/) UE5 CA254170. Our courses feature tools developed by ITCR Investigators and make it easier for principal investigators, scientists, and analysts to integrate cancer informatics into their workflows. Please see our website at <www.itcrtraining.org> for more information.Except where otherwise indicated, the contents of this course are available for use under the Creative Commons Attribution 4.0 license. You are free to adapt and share the work, but you must give appropriate credit, provide a link to the license, and indicate if changes were made. Sample attribution: Leadership for Cancer Informatics Research by Johns Hopkins Data Science Lab (CC-BY 4.0). You can download the illustrations by clicking [here](https://docs.google.com/presentation/d/1cd434bkLer_CJ04GzpsZwzeEA9gjc5Ho6QimiHPbyEg/edit?usp=sharing).

# 1 Introduction



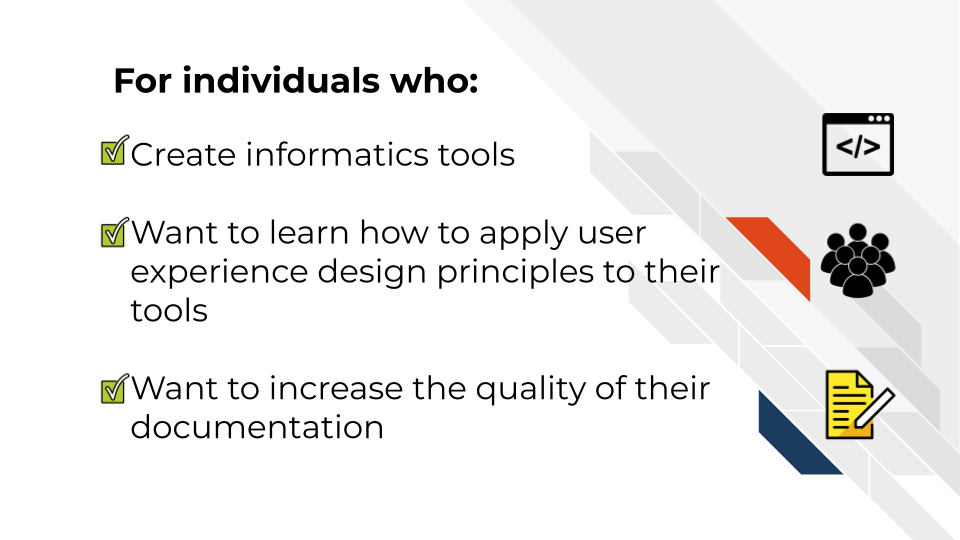
## 1.1 Motivation

Cancer datasets are plentiful, complicated, and hold information that may be critical for the next research advancements. In order to use these data to their full potential, researchers are dependent on the specialized data tools that are continually being published and developed. Bioinformatics tools can often be unfriendly to their users, who often have little to no background in programming ([Bolchini et al. 2008](#ref-Bolchini2008)). The usability and quality of the documentation of a tool can be a major factor in how efficiently a researcher is able to obtain useful findings for the next steps of their research.

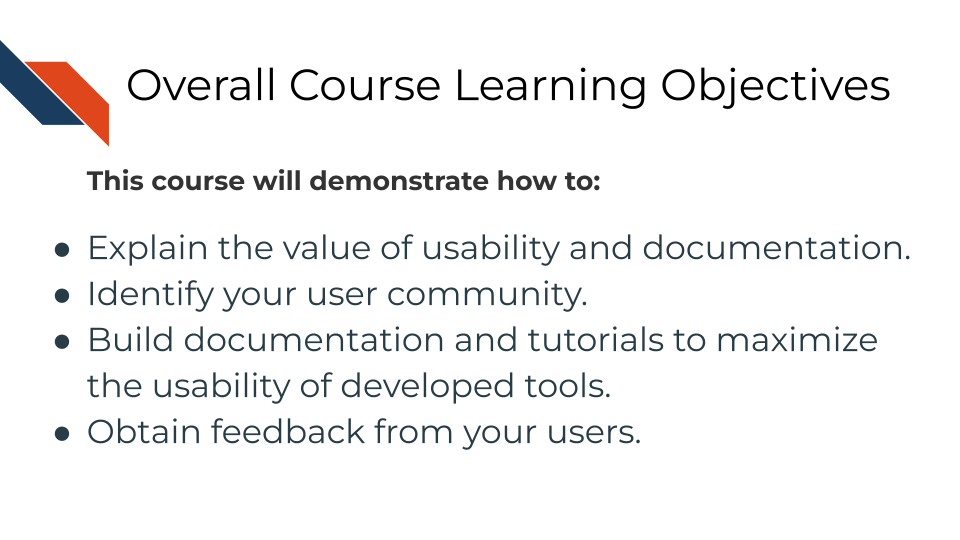
Increasing the usability and quality of documentation for a tool is not only helpful for the researcher users, but also for the developers themselves – the many hours of work put into the product will have a higher impact if the tool is usable by the target user community. 70% of bioinformatics tools surveyed by Duck et al. ([2016](#ref-Duck2016)) were not reused beyond their introductory publication. Even the most well-programmed tool will be overlooked by the user community if there is little to no user-friendly documentation or if they were not designed with the user in mind.

## 1.2 Target Audience

The course is intended for cancer informatics tool developers, particularly those creating tools as a part of the [Informatics Technology Cancer Research](https://itcr.cancer.gov/informatics-tools).

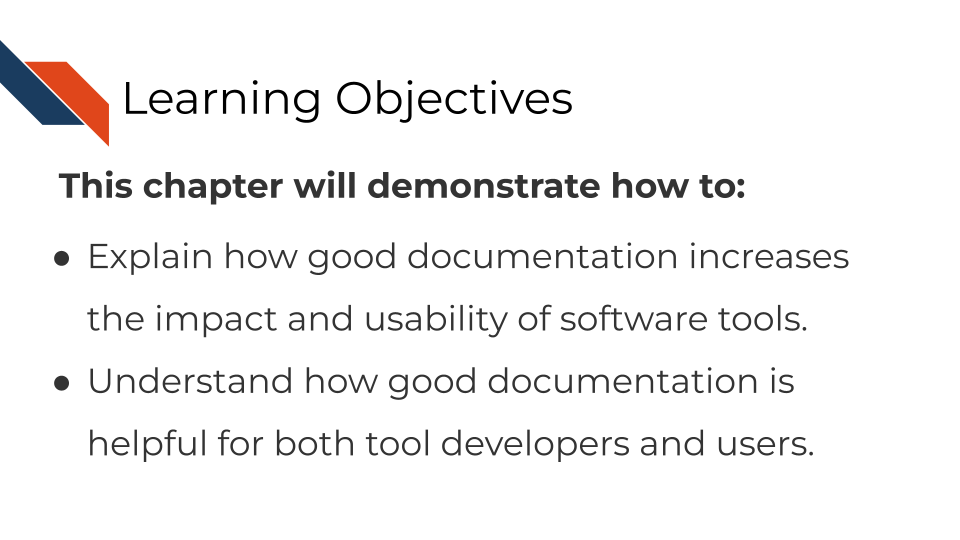


## 1.3 Curriculum



The course includes a hands-on exercises with templates for building documentation and tutorials for cancer informatics tools. Individuals who take this course are encouraged to use these templates as they follow along with the course material to help increase the usability of their informatics tool.

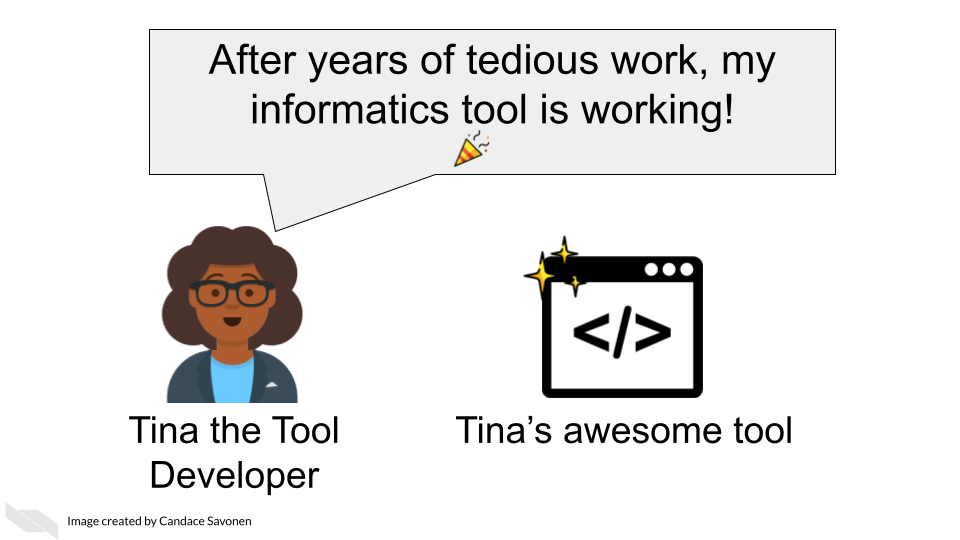
# 2 Documentation: Why it’s worth the effort!



## 2.1 The context of bioinformatics tool development

Tool development is an exciting but long process – filled with lots of careful programming, tedious troubleshooting, but also ‘Aha’ moments that ultimately can result in an amazing product that you should be proud of!

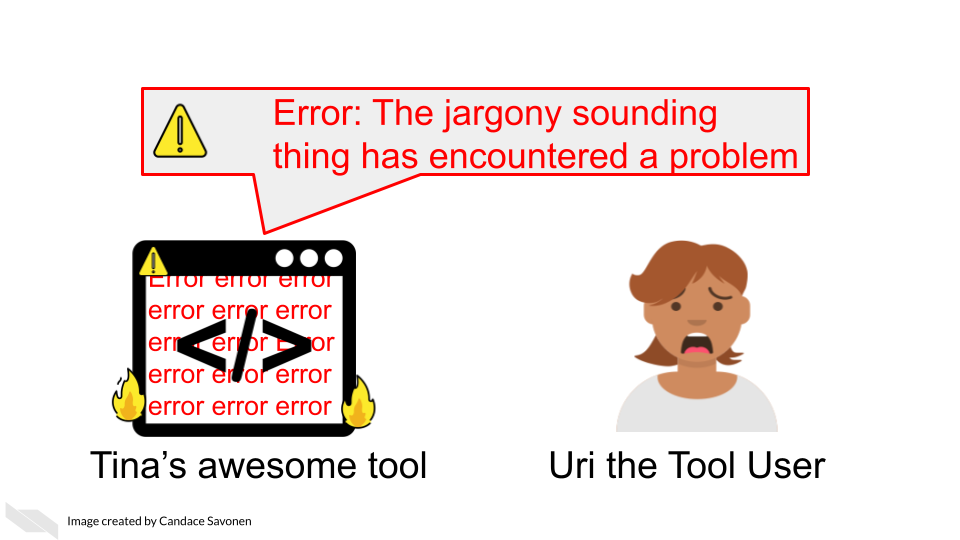
Tina the Tool developer, perhaps like you, has just gotten her product working well and many of the bugs have been sorted out. Tina’s awesome tool is working exactly as designed and Tina is excited to get her tool out there to be used by the community!

 [[1]](#footnote-35)

This is indeed cause for celebration! Perhaps researchers like Uri the Tool User will come across Tina’s awesome tool and share in Tina’s enthusiasm for the project! Tina’s bioinformatics tool may be just what they were needing for their research project!



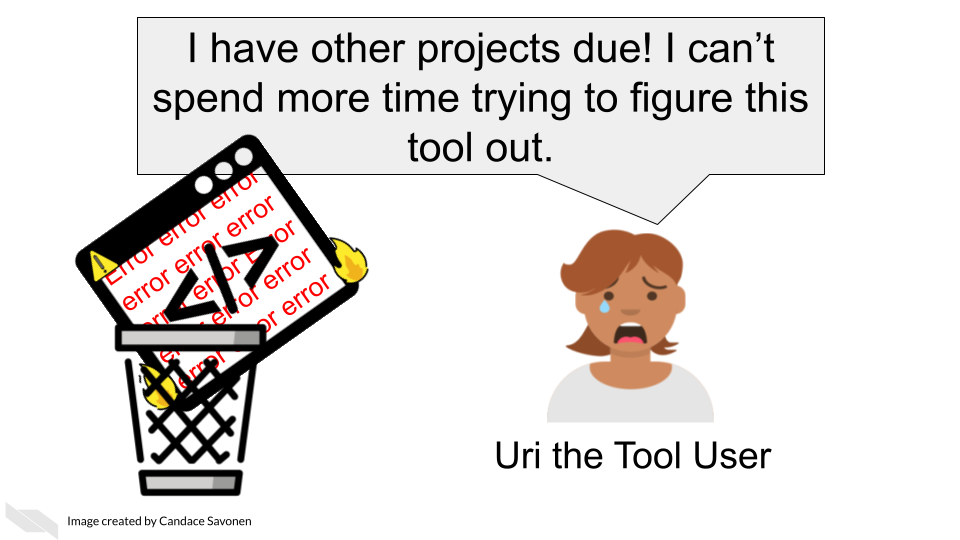
Uri the Tool User can’t wait to apply Tina’s awesome tool to their project! But, it may not be long before Uri encounters errors, or questions about Tina’s awesome tool, no matter how high quality Tina’s programming of the tool is.



Often users like Uri, particularly in the biology and cancer fields, have little to no programming experience. Even if a user does have programming experience, they are still unfamiliar with how Tina has set up tool. The tool may even be working exactly according to Tina’s vision but if users like Uri do not understand Tina’s vision or basic programming principles that Tina might take for granted, it can lead to a lot of frustration and time inefficiently spent.

If the tool’s documentation is non-existent, scarce, out-of-date, or filled with too much jargon, the chances that Uri will be able to successfully and efficiently create a product with the tool is drastically diminished.

Lack of usability often leads users to ditch even the most well-programmed of tools.



This is the unfortunate and all-too-common result of many bioinformatics tools.

## 2.2 Bioinformatics and usability

The lack of focus and education on usability in the bioinformatics tool development is not only a disservice to progress of cancer research, but also to the tool developers themselves who have equally spent uncountable work hours and effort on the development of cancer tools.

We know that bioinformatics tool development doesn’t occur in a vaccuum. User designers in the field of bioinformatics have commented on reasons why documentation and usability sometimes suffer for bioinformatics tools:

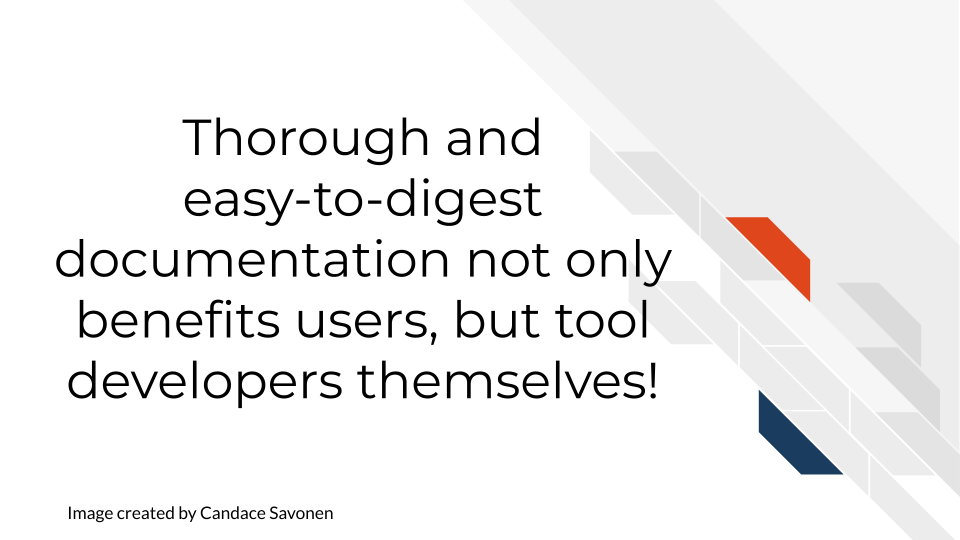
* Problem 1) Tools developed in academia are often left to deprecate after publication because novelty is often prioritized over long-term maintenance and usability ([Mangul, Martin, et al. 2019](#ref-Mangul2019)).
* Problem 2) Bioinformatics tool development teams generally don’t have the resources to hire user-centered design experts and the small and specialized user communities are often overbooked and not incentivized to give feedback ([Pavelin et al. 2012](#ref-Pavelin2012)).
* Problem 3) There is a lack of resources/education about usability specific to bioinformatics tool developing communities ([Pavelin et al. 2012](#ref-Pavelin2012)).

Unfortunately this specific course cannot address issues 1 and 2, but will attempt to help with problem 3.

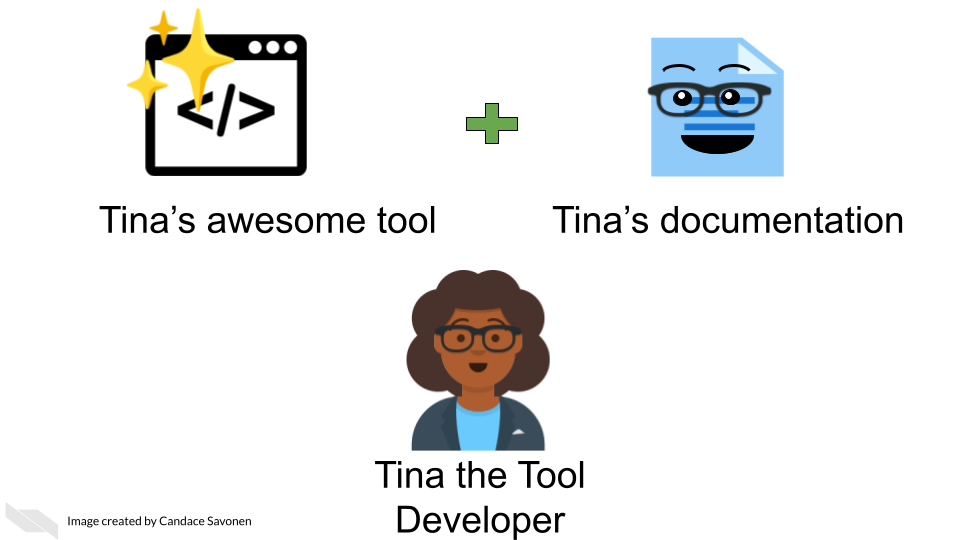
## 2.3 Why documentation is worth the time

We realize many tool developers feel unenthused about the process of creating documentation. The documentation process requires a different skill set from the tool development itself; meaning many developers were likely not attracted to tool development because of documentation and may not be sure how to craft good documentation ([Wolf 2016](#ref-Wolf2016)). They may know its good for their tool, but they just aren’t enthused about it.

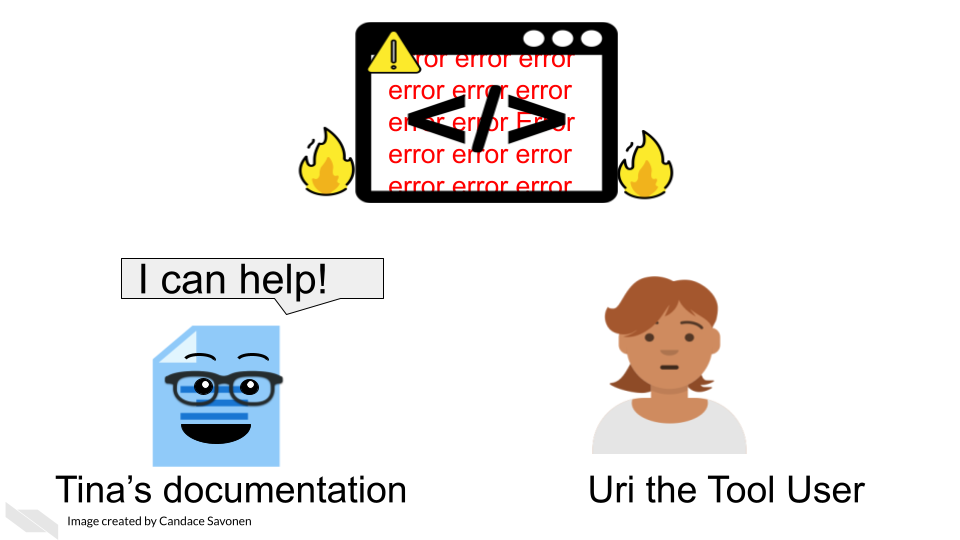
We’d like to assure you that the effort for creating documentation has a high return payoff for the continued success of your tool as a whole!



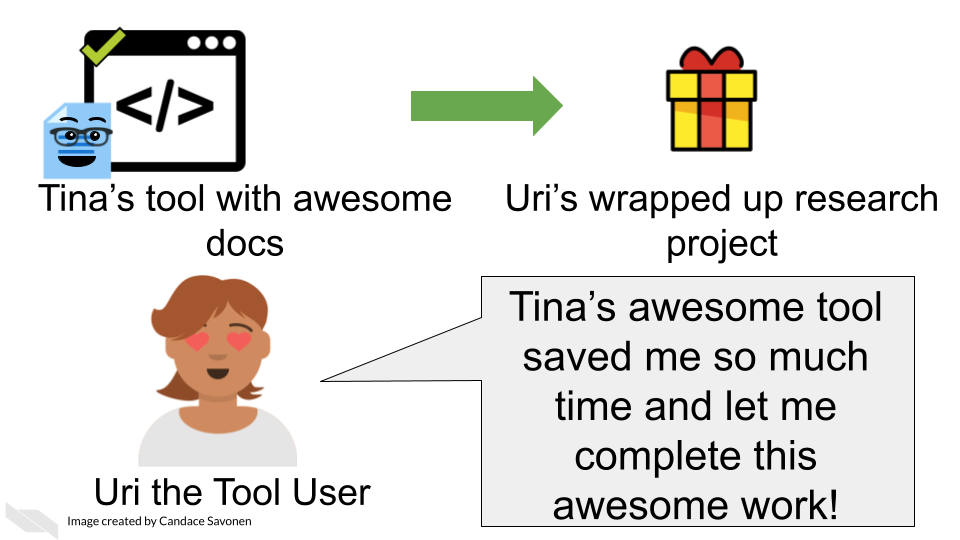
Returning to our cast of characters, let’s say that Tina the Tool Developer, had the time and knowledge to create awesome documentation for her tool.



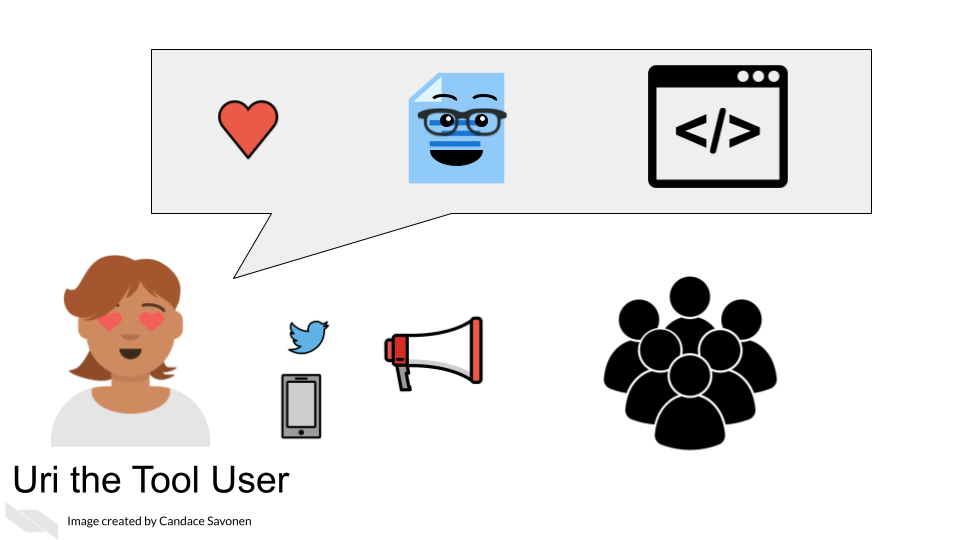
Uri the tool User is still likely to encounter errors and problems, but with thorough and easy-to-digest documentation, Uri is better equipped to troubleshoot these problems! They may also learn more about the features and limitations of the tool that will better guide Uri’s next steps!



Being equipped with user-centered documentation, Uri is more likely to be able to reach the next steps of their research and potentially share a publishable result! Tina’s tool is now more likely to be cited in publications, or other forms of media.

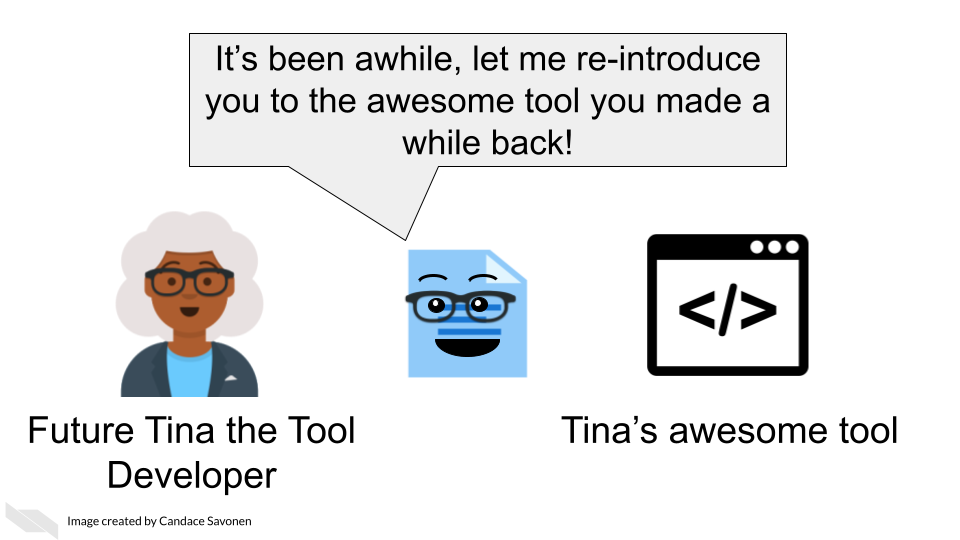


This rewards Uri for having used Tina’s tool, making Uri not only likely to continue to use the tool for their next projects, but Uri may also help spread the word about how great their experience with Tina’s tool was.

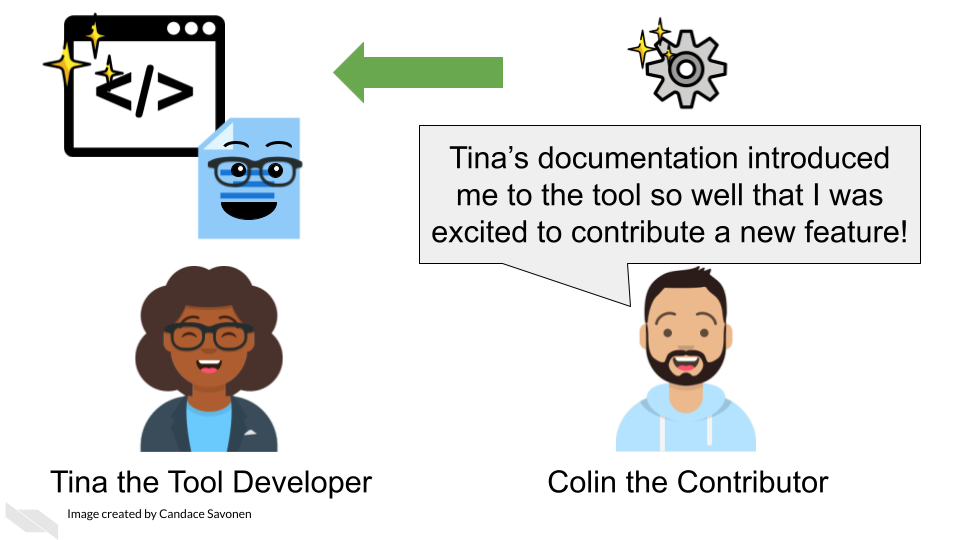


This means that Tina may have a larger user base for her tool and will help Tina with future funding opportunities and making connections that will help her create more awesome tools!

Well-documented tools help developers better maintain their code in the future because they may forget the mechanics of their tool over time. If future Tina has to divert her time and effort to another project but then returns to do tool maintenance, documentation may help jog her memory!

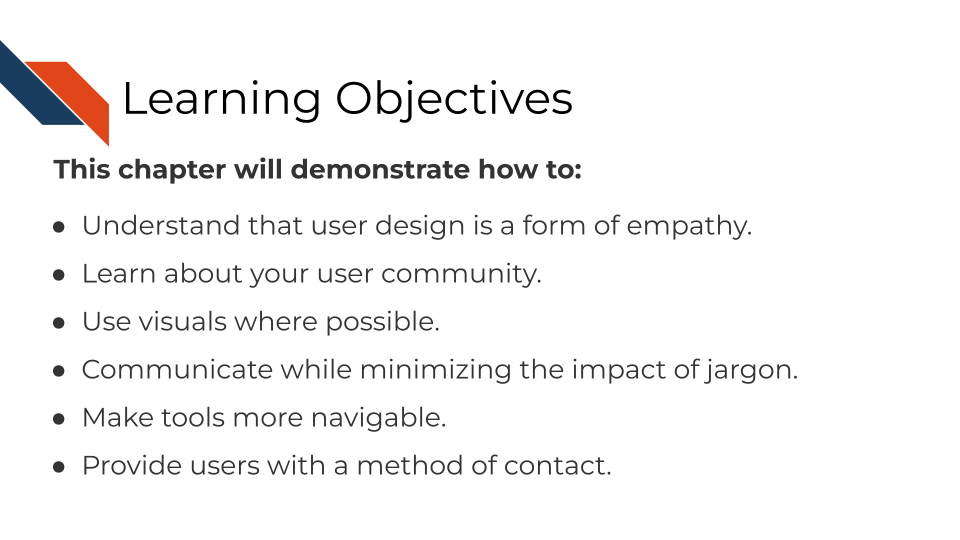


Thorough and easy-to-digest documentation may also help other tool developers contribute features or fix bugs in Tina’s tool. Here Colin the Contributor was able to read Tina’s awesome documentation. It not only got him excited about the tool, but allowed him to program a new feature which he sent to Tina.



Now that you are hopefully energized and ready for creating documentation for your tool, let’s talk about a bit user-centered design concepts!

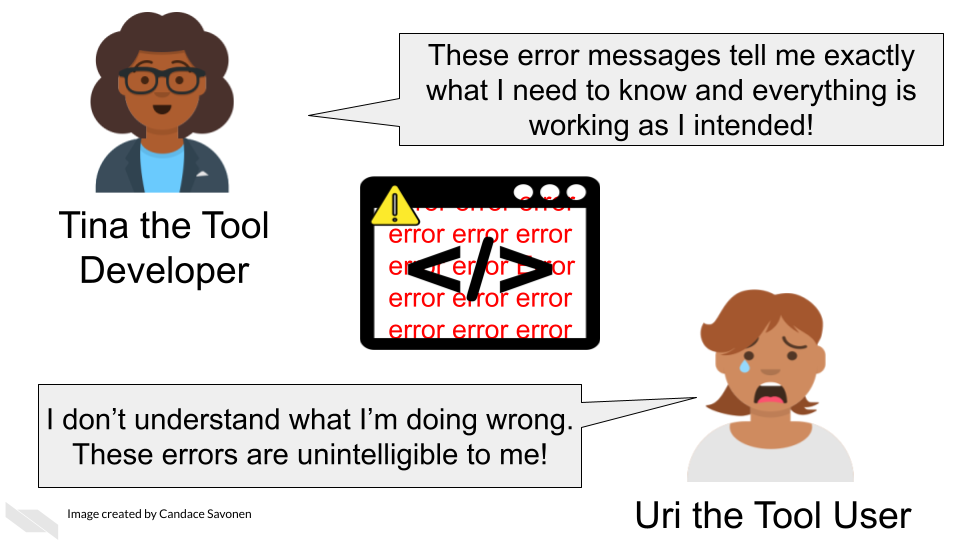
# 3 Lessons we should borrow from user designers



## 3.1 Thinking about user-centered development

Creating tools that are easy to use starts with thinking about your user’s perspective. In other words, user-centered design is an exercise in applied empathy ([Matos et al. 2013](#ref-deMatos2013)).

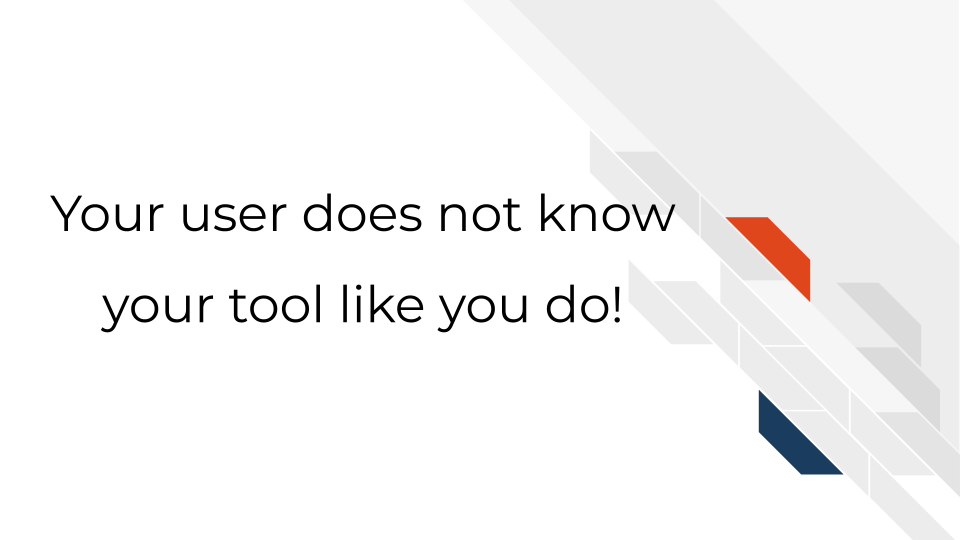
This is why a common saying in user-centered design is “You are not your user” ([Alexakis, n.d.](#ref-Alexakis2017)). Although it may be true that you may have a lot in common with your user, this saying is based in the idea that you should not assume your user knows what you know or thinks like you do. For example, a warning message that may seem perfectly clear to you as a developer, may be a foreign language to your user.

 [[2]](#footnote-54)

As compared to yourself, your typical user may likely have a different:

* Educational background.
* Programming experience level.
* Set of motivations for using your tool.

And most importantly *your user does not know your tool like you do*! You have spent many, many hours developing this tool and its unrealistic and impractical for them to spend the same number of hours with your tool that you have.



Also keep in mind users are humans in a context. Humans have demands in their life, have been working long days, and are tired/frustrated/distracted/etc. Making your tool as easy as possible to use increases the likelihood of your user continuing to stick with your tool and even becoming an advocate for your tool to their colleagues!

### 3.1.1 Generalities about bioinformatics tool users

On a general level, there are some characteristics we know about bioinformatic tool user communities.

The typical user of bioinformatics tools are generally:

* Highly educated in their specific research topic.
* Skilled at picking up new information.
* Unlikely to have extensive experience in programming; may be uncomfortable with using command line.
* If they do have experience programming, it’s likely been from informal and self-taught learning.
* Like many professionals, likely to feel time pressure from deadlines.

However, the bioinformatics user community also includes a variety of individuals with different roles and experiences. Mulder et al. ([2018](#ref-Mulder2018)) described 10 user personas for bioinformatics software, all with their own skills and competencies:

* Physician
* Lab technician
* Ethicist
* Biocurator
* Discovery biologist/academic life science researcher
* Molecular life science educator
* Academic bioinformatics researcher
* Core facility scientist
* Bioinformatician in an academic or research infrastructure support role
* Bioinformatics software developer/software engineer

Additionally, users may be at various stages in their education (undergraduates, graduates, postdocs, etc) and may have varying experience and time constraints.

## 3.2 Exercise: Think about the user community for your tool

Write down what you know (or think you know) and try to identify any knowledge gaps you might have about your user community.

* Who is your user community?
* What need of theirs is addressed by your tool?
* What kinds of skills does your tool assume your user has (does your user community have those skills)?

Keep the questions about your user community in mind and in a [later chapter](https://jhudatascience.org/Documentation_and_Usability/obtaining-user-feedback.html), we’ll go into more detail about conducting user research to address any knowledge gaps you may have about your user community.

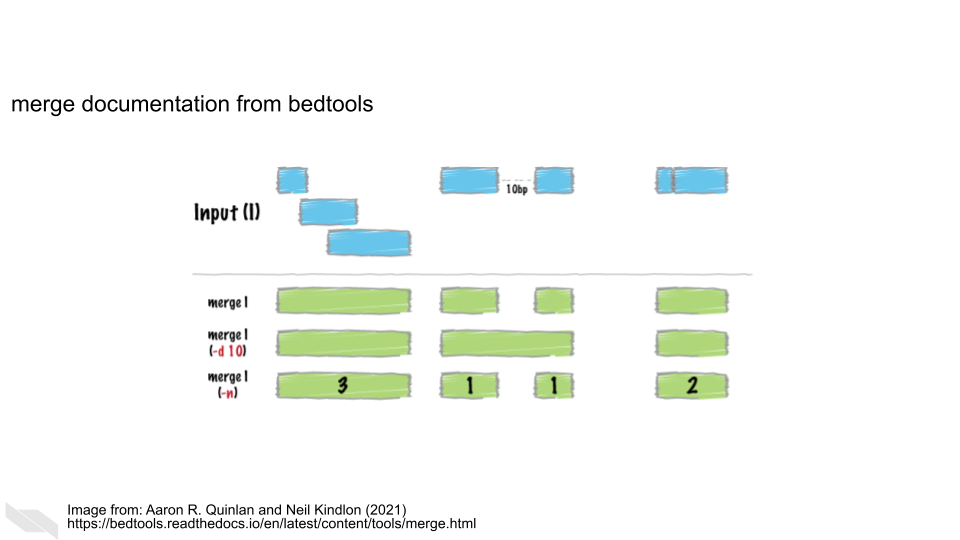
## 3.3 General principles about user-friendliness

While finding out about your user community is critical, there’s also principles we can discuss that are common to all users/humans.

#### 3.3.0.1 Humans like visuals

Humans are drawn to intuitive visuals. Visuals are efficient means of communication and help users absorb information better than long-winded paragraphs (though visuals need an accompanying explanation too).

Sometimes this is particularly helpful for complicated concepts. For example, BEDtools (Quinlan and Hall ([2010](#ref-Quinlan2010))) allows for the manipulation of genomic sequences in BED files. Some of these principles can be complicated to visualize, but the authors of BEDtools do a great job of using visuals to explain each function:



Here, this figure explains how the merge function works given a particular set of ranges.

#### 3.3.0.2 Stay away from jargon

What someone considers jargon is a very relative to their own experiences. Terms that may seem like common knowledge to you may be foreign to your user.

For example, something seemingly commonplace to you like TSV may be something a user does not understands what it means. To help smooth over jargon-related barriers, spell out abbreviations the first time you mention them (e.g. ‘tab separated values (TSV)’). This doesn’t mean you have to have long winded explanations of every term, instead you can quickly link out to an article or website with information about a term you’ve used (e.g. [tab separated values file (TSV)](https://en.wikipedia.org/wiki/Tab-separated_values)). This has the benefit of saving you and your user’s time without making your explanations too long winded.

#### 3.3.0.3 Navigability: Make help easy to find

When your user has a question in mind, it can make their experience a lot more stress-free if they can find their answer with ease. Its not always obvious to a user where you have put an answer to something, particularly if user is already frustrated; deep in the troubleshooting trenches! For these reasons, navigability is important for your tool and documentation.

*Quick tips for increasing your tools’ navigability*:

* Minimize the number of clicks for finding help!
* The more important a piece of information is to a users’ ability to use your tool, the more it should be in the forefront of your tool!
* Patterns are super helpful! Be consistent about where to find things. Once a user has caught on to your pattern of how you store things in your tool, don’t switch it.
* Use visual cues where possible!

##### 3.3.0.3.1 Use reading patterns to your advantage:

Even if you try to make your text succinct and lean on visuals where possible, there will still be instances where you may inevitably require large chunks of text.

Large chunks of texts are hard on your user, especially if they are trying to scan the information for something in particular.

To help these chunks of text be less cumbersome, utilize reading patterns:

Described in this (useful article [Soegaard, n.d.](#ref-Soegaard2020))(<https://www.interaction-design.org/literature/article/visual-hierarchy-organizing-content-to-follow-natural-eye-movement-patterns>), humans’ eyes are drawn to items based on:

* Size: Larger elements will dominate and catch eyes first.
* Color: Bright colors catch eyes ahead of muted, drab ones.
* Contrast: Stark differences between elements draw eyes to the brighter one.
* Alignment: Users expect to find certain elements in the same place.
* Repetition: A repeated quality (e.g.,colored parts of text) draws the user’s eye.
* Proximity: Putting related elements (e.g., header with associated text) close together means these are related.
* Whitespace: Including whitespace around elements singles them out as separate groups of information.
* Texture and Style: Using distinct textures/styles (e.g., chunky, military-style buttons) draws the eye while setting the theme.

Based on this, human eyes will scan a website’s page in Z or F pattern ([Soegaard, n.d.](#ref-Soegaard2020)).

The Z Pattern — In designs without much text, our eye starts scanning from top left to top right, then diagonally down to bottom left, stopping at the bottom right.

The F Pattern — In designs with more text, we scan across the top, from left to right, then down the left, searching for clues to what we want to know. On finding one, we’ll scan across to the right.

For more details on these patterns and how to implement them with your own content, read more: - [About natural eye movements](https://www.interaction-design.org/literature/article/visual-hierarchy-organizing-content-to-follow-natural-eye-movement-patterns) by Soegaard ([n.d.](#ref-Soegaard2020)). - [About “F” patterns](https://uxplanet.org/f-shaped-pattern-for-reading-content-80af79cd3394) Babich ([n.d.](#ref-Babich2017)).

More about user-friendly principles in general:

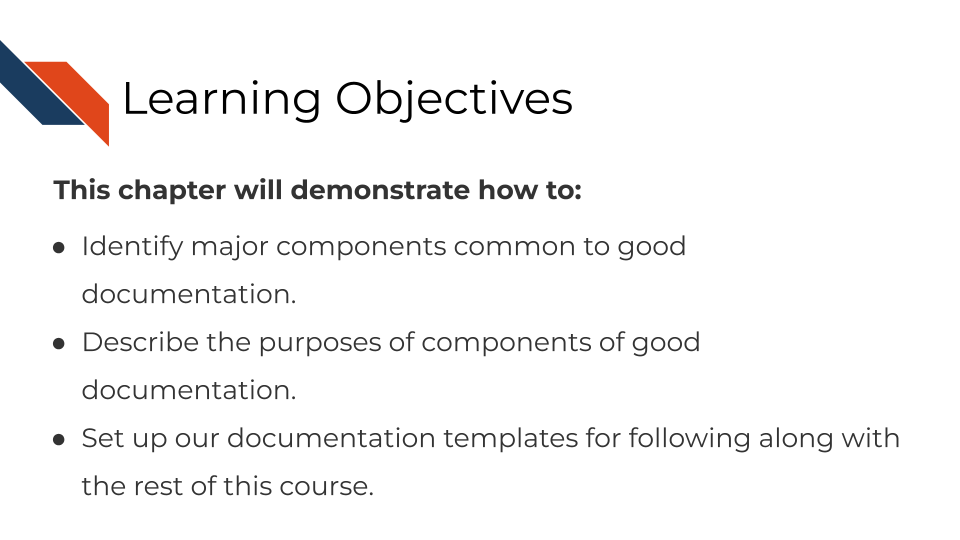
* [Principles of User Interface Design](http://bokardo.com/principles-of-user-interface-design/) by Porter ([n.d.](#ref-Porter)).
* [Golden Rules of UI Design](https://xd.adobe.com/ideas/process/ui-design/4-golden-rules-ui-design/) by Babich ([2019](#ref-Babich)).
* [About intuitive layouts](https://www.oreilly.com/library/view/designing-interfaces/0596008031/ch04.html) by Tidwell ([2021](#ref-Tidwell)).

#### 3.3.0.4 Provide a method for users to file concerns

No matter how much you have perfected your tool, it will never be perfect, especially since software deprecates over time. Because of this or other [unknown unknowns](https://uxdesign.cc/the-knowns-and-unknowns-framework-for-design-thinking-6537787de2c5), a usable tool still has a way for a user to let the developers know when something isn’t working. This might be a direct and obvious break like a bug or broken link, but it could also be something more subtle that also requires your attention. We’ll discuss this in more detail in a [later chapter](#obtaining-user-feedback).

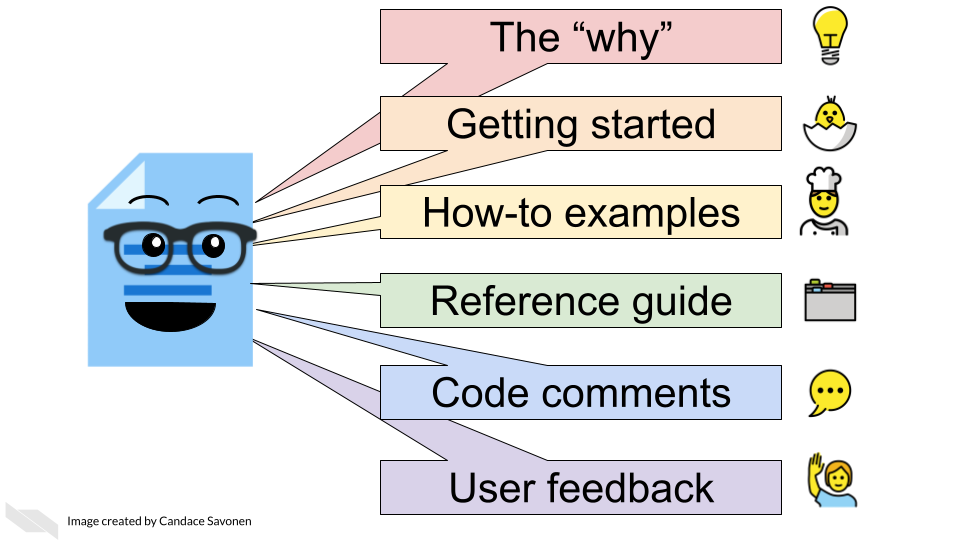
Now that we’ve discussed some major principles about users and design, let’s dive in to talking about how documentation can help!

# 4 What does good documentation look like?



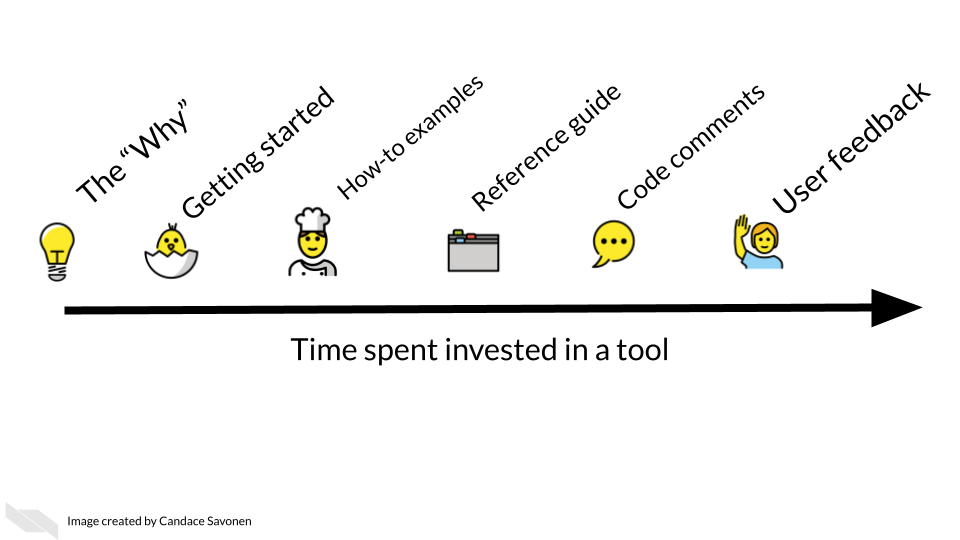
## 4.1 Major components of good documentation

In this chapter we are going to cover the major aspects of a well-documented tool. In the subsequent chapters, we will talk about each of these components in more detail, providing relevant examples and tools.



[[3]](#footnote-77)

We can think of these components of documentation in terms of how much time a user has invested in learning the tool:



This isn’t always a perfect linear timeline like this, users may use these items in a different order than we expect, but this demonstrates the intent of how users would progress through the documentation.

### 4.1.1 The ‘Why’

Before we get into the technical information in your documentation, the first thing that should be obvious to your user is why they should want to use your tool! What need of your users does your tool fulfill? If this is not glaringly obvious, users will move on without realizing how valuable your tool could be for their research!

#### 4.1.1.1 Exercise: Write the why of your tool!

This should be the first thing your user sees on the main page of your tool. If it currently is not clear, take this time to workshop one or two sentences that explain the ‘why’ of your tool. As you craft this sentence, think about the needs of your user and how to summarize your tool’s purpose in a brief, punchy way. Stay away from jargon unless perhaps its jargon that you know your user will understand.

*Examples of tools with their why’s well-stated prominently on their web page:*

Gene Set Enrichment Analysis (GSEA) is a computational method that determines whether an a priori defined set of genes shows statistically significant, concordant differences between two biological states (e.g. phenotypes).

Here, [GSEA](http://www.gsea-msigdb.org/gsea/index.jsp) tells us the exact kind of question we can ask and what input is needed.

Salmon is a tool for wicked-fast transcript quantification from RNA-seq data. It requires a set of target transcripts (either from a reference or de-novo assembly) to quantify.

[Salmon](https://salmon.readthedocs.io/en/latest/salmon.html#using-salmon) tells us its for transcript quantification but that its particular strength is it’s “wicked-fast.”

### 4.1.2 Getting started

A getting started section will be a new user’s first introduction to your tool. It will tell them specific steps they need to “get started” (hence why we call it this) – no long-winded explanations, just the quickest way to reach some sort of output.

In the most clear steps as possible, a getting started section will tell the user how to:

* Install the tool and any of its dependencies.
* Run the most basic analysis with the tool.
* Obtain some quick result.

It’s crucial that the steps here are simple and easy to follow or you risk losing new users before they even get going.

### 4.1.3 How-to examples

A set of how-to examples are like a cookbook of recipes that will demonstrate step-by-step the most common uses for your tool after they have completed the getting started steps.

* The most common use cases of your tool are shown step-by-step
* Each step is clearly provided and explained.
* Any data needed to run the example is provided.
* Example code is written with teaching in mind.

How-to examples give users next steps to get further comfortable with your tool and shows them what an analysis might look like.

### 4.1.4 Reference guides

A reference guide is like a dictionary that a user can look up items as questions arise. A user will likely have a specific question about a function, parameter, data type, or option and will want to be able to navigate to information about that item. Users who end up looking through your reference guide are likely a bit invested and may have already gone through the getting started section and/or how-to examples.

* All items are described in clear, non-jargon language.
* The items are searchable or at least can be easily visually scanned.
* All options and defaults are explained and linked to more information where appropriate.

### 4.1.5 Code comments

Comments in your source code are also a part of documentation – and likely the first part of the documentation you worked on! In most instances, if your tool is functioning fine, code will probably only be looked at by advanced and/or the most invested users. But also recall that documentation is not only for your user but for future you and existing or potential collaborators.

### 4.1.6 User feedback mechanism

Try as you might, you will not be able to predict every scenario that a user may engage with your tool. Your user may encounter errors or quirks that you did not foresee but that would be helpful for you to know about. Your documentation should direct your users where they should send any comments or concerns. You should also make sure that this feedback method is something you can check up on and respond to regularly. We’ll also discuss how to conduct usability research to get the most informative feedback on your tool.

## 4.2 The documentation templates for this course

We encourage you to create these pieces of documentation we will further discuss in the rest of this course! We have template documentation you can use to use as a starting point.

*Still in progress: If you are creating documentation to accompany a package you are submitting to Bioconductor or Galaxy we are working on constructing specific templates and recommendations for those instances.*

### 4.2.1 Well-documented checklist

[This document is a checklist](https://github.com/jhudsl/template-documentation/blob/master/docs/well_documented_checklist.md) that summarizes the major aspects that should be included in a tools’ documentation. We recommend using it to evaluate the documentation for an existing tool and identify any gaps you may need to address, or as a to-do list for creating new documentation that you can check off as you follow along with this course.

For more inspiration, here are more articles about good documentation:

* [Benefits of Software Documentation - SDLC Best Practices](https://systemsplusgroup.blogspot.com/2014/05/benefits-of-software-documentation-sdlc.html)
* [5 Examples of What Great Documentation Looks Like for Developers](https://codeburst.io/5-examples-of-what-great-documentation-looks-like-for-developers-162bdec11020)
* [Software Documentation Types and Best Practices](https://blog.prototypr.io/software-documentation-types-and-best-practices-1726ca595c7f)

## 4.3 Exercise: setting up our templates for your own use

There are two options we suggest for creating documentation as you follow along with this course.

**Option 1)** Use these template markdowns essentially as they are (after you fill them in) and add them to an existing repository.  
*Pros*: Is easier and quicker.  
*Cons*: Is not as user-friendly as option 2.

**Option 2)** Clone a repository with these templates and set up a MkDocs github pages site. Slightly more work, but a very nice end result; [see demo here](https://jhudatascience.org/template-documentation/).  
*Pros*: This format of documentation sites are easy to navigate and likely familiar to your user.  
*Cons*: Will require you to use mkdocs package to get this set up.

### 4.3.1 Steps for Option 1) Adding completed template markdowns to an existing GitHub repository

* [Download the folder of templates using this link](https://github.com/jhudsl/template-documentation/tree/master/).
* Complete each markdown template, filling in the blanks as you go along with the course.
* File a pull request to your repository to add these files.
* (Optional but encouraged) [Publish your repository to Github Pages](https://guides.github.com/features/pages/).

Here’s what’s in that folder:

templates/  
├── well\_documented\_checklist.md  
├── getting\_started\_template.md  
├── how\_to\_examples.md  
├── reference\_guide\_template.md  
├── bioconductor-guides/  
│ ├── bioconductor\_example\_script.R  
│ ├── bioconductor\_vignette\_template.Rmd  
│ └── README.md  
└── galaxy-guides/

### 4.3.2 Steps for Option 2) Creating a MkDocs site with Github pages

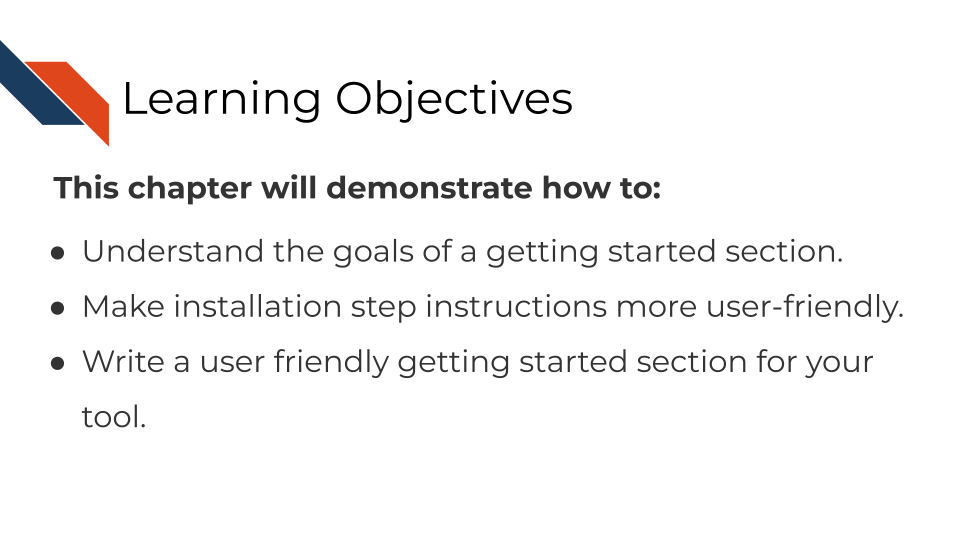
* Go to our [template documentation repository](https://github.com/jhudsl/template-documentation).
* Click Use this Template to get started.
* Name your repository something that relates to your tool.

*On your computer’s command line*:

* [git clone your new repository](https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/cloning-a-repository) you made from our template.
* Navigate to your the root of your local version of this repository.
* [Install mkdocs following their instructions](https://www.mkdocs.org/#installation).
* Run mkdocs serve to see a preview of your docs.
* Edit the template pages we’ve provided in the docs/ folder.
* As you make changes you can run mkdocs build and then mkdocs serve to see a preview.
* When you are ready to publish your documentation to its own website, run [mkdocs gh-deploy](https://www.mkdocs.org/user-guide/deploying-your-docs/), it will return the web address of your new site
* Go to that address and bask in the success of your newly made documentation!

Now that we have a basic structure and plan for our documentation, let’s discuss each each section of this documentation in more detail!

# 5 Creating a smooth getting started section



## 5.1 The goal of a getting started section

A getting started section is new users’ first experience with your tool. It is also can be the most frustrating experience for your user if installation doesn’t happen smoothly.

 [[4]](#footnote-108)

Our goal for our getting started section is to guide our new users through the installation steps as quickly and smoothly as possible then send them to a brief tutorial to show off the awesomeness of your tool!

## 5.2 Aspects of a smooth getting started section

### 5.2.1 Easy to find!

It’s hard to get started if you don’t know where to begin! Your getting started section should be the easiest page to find. Have your link to your getting started page prominently feature on your landing page. If you have a navigation bar, include a link to your getting started page.

### 5.2.2 Introduce your user to the basics

Provide users with the introductory concepts of the tool; briefly expand a bit more on The Why about your tool they already saw.

* In what context would someone be using your tool?
* What scientific questions or other needs might this tool assist with?

If your users’ needs fit your description, this will fuel them with the motivation to get through the first big hurdle: installation.

Before getting to the install steps, a special consideration: Does your tool have multiple ways to run it, for example can it be ran either through a GUI or command line? Describe this to your users so they get shuttled to the method of running your tool that is right for them.

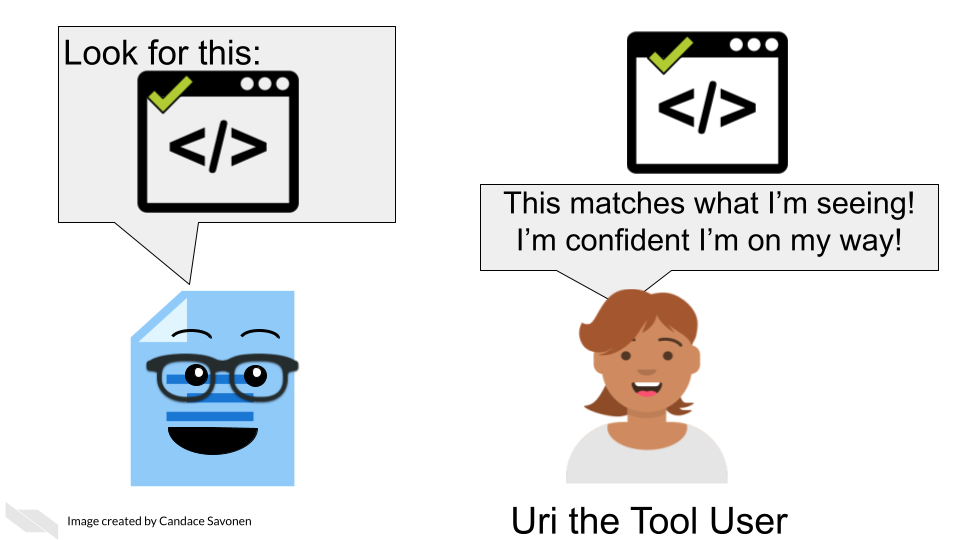
### 5.2.3 Installation steps: the first big hurdle

Installation is the first and perhaps biggest hurdle your user will encounter with your tool. The **clearer and more specific these steps the better**. Mangul, Mosqueiro, et al. ([2019](#ref-Mangul2019b)) found that tools that required more installation steps (but didn’t describe these steps adequately) were less likely to be installed successfully, and tools that were less likely to be installed successfully had significantly less citations!

If installation happens through command line, provide **copy-and-paste or directly runnable commands** that your user can use as-is. In these commands, if parts of it need to be tailored, call attention where the tailoring needs to happen and how your user can determine what they need to put there. Fill-in-the-blank cues can be handy for these scenarios.

**Tell your users what to expect**. Do some steps take more time than others? Warn them about that. Are there output prompts that may not be intuitive but are to be expected? For example, sometimes a regular red text installation message may indicate things are working fine, but if a user doesn’t know what the text means, sometimes they will try to interpret red text as meaning something bad has occurred.

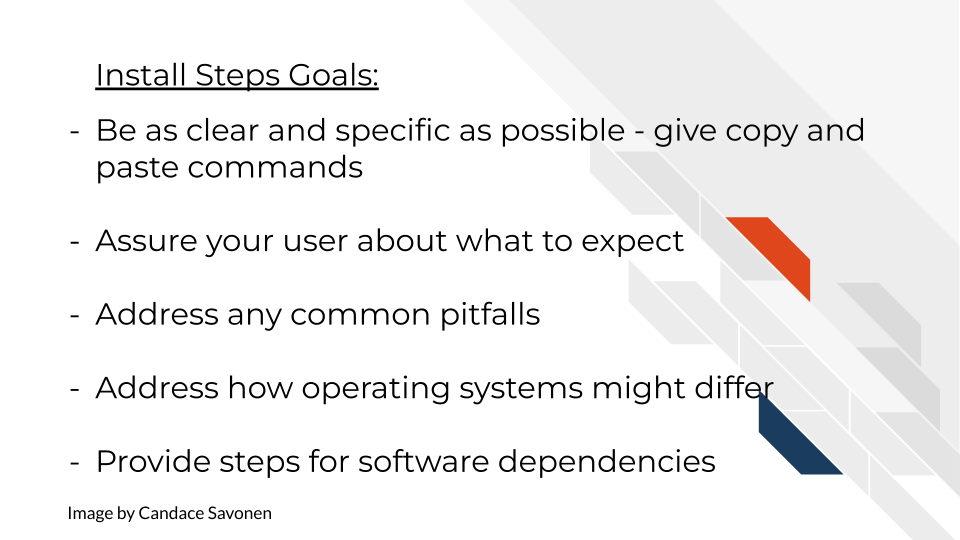
Where it makes sense, you **use screenshots as assurances** to the user that they are on the right track. Being able to see that your users’ screen matches what is shown in your screenshots reassures them that things are progressing correctly. Conversely, if something does not match, it can help them narrow in on a problem. *Keep in mind out-of-date screenshots can add to the confusion! – more on tips about how to keep things up to date in a later chapter.*



Install steps should also try to address any common pitfalls, particularly **how different operating systems might require different steps**. You may consider having separate sections or pages to describe install steps on different operating systems.

What dependencies does installing your tool require? Will these be installed automatically by the steps you describe or **does your user need to install other software** before being able to install your tool? This can be a big roadblock to users if dependencies and how to install them are not addressed.

**To recap:**



Installation steps can be tricky – and admittedly hard to give guidance on when individual computer’ set ups can differ so much, but the more you are able to workshop your guidance to your users here, the more they will appreciate it and stick with your tool!

### 5.2.4 Cover the basics

Your getting started section should give your users the basic concepts they need for running your tool – a knowledge foundation that they can build upon as they continue to explore and follow your how-to examples.

* What basic terms should your users know?
* What do your users need to get started - what kind of data?

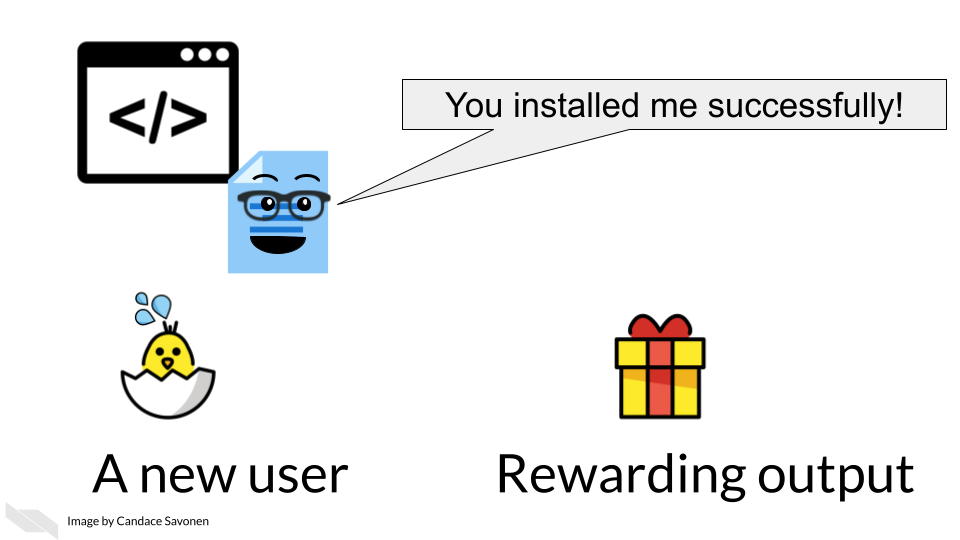
Because you know all the ins and outs of your tool, you may be tempted to discuss the intricacies of your tool as you are writing the getting started information, but try to stick to [“just in time”](https://en.wikipedia.org/wiki/Just-in-time_learning) information – only bring up concepts exactly when your users will need them, not before. Users will almost always feel more information-overload than you suspect.

### 5.2.5 Reward your user with a short tutorial!

Installation steps are not fun so the later part of your getting started section should lead your user into a quick tutorial that will reward your user for making it through the hard part!

Give your users the fewest steps needed to produce a rewarding result that will excite them about continuing to use your tool! Use this opportunity to show off your the simplicity and awesomeness of your tool!

This rewarding result might be a cool visual or a plot – but also should demonstrate the most popular thing your users would like to see.

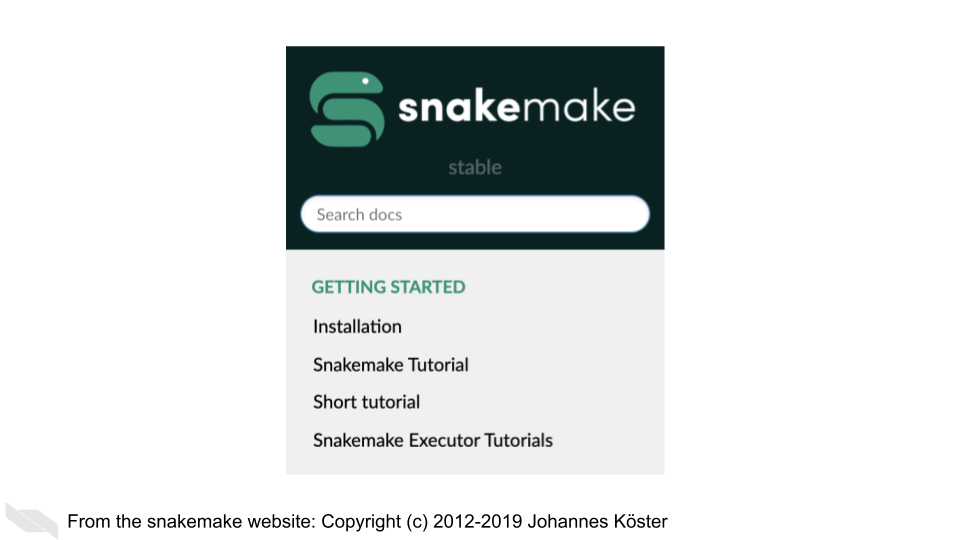


### 5.2.6 Directs the user to the how-to examples section

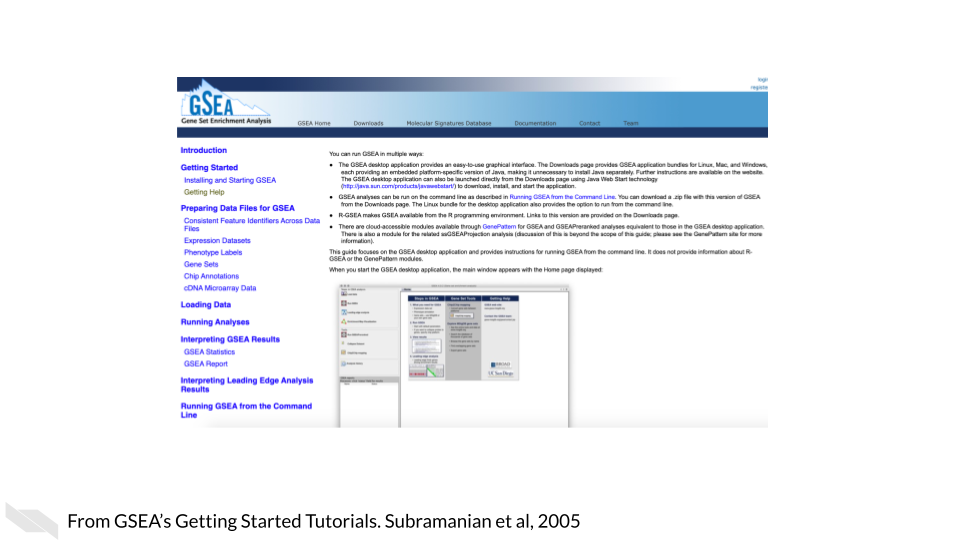
Now that your user has successfully installed your tool and understands the basic idea, let them know where they can find more examples to keep the learning train going! Have a link at the end of your getting started section that sends them to a place they can browse your how-to examples (which we will discuss in the next section!).

## 5.3 Good examples of getting started sections

[Snakemake has a great getting started section](https://snakemake.readthedocs.io/en/stable/getting_started/installation.html) ([Molder et al. 2021](#ref-Molder2021)). The makers of Snakemake tell their users how to install Snakemake using different situations and keeping dependencies in mind, right after which they have a short tutorial to entice their users!



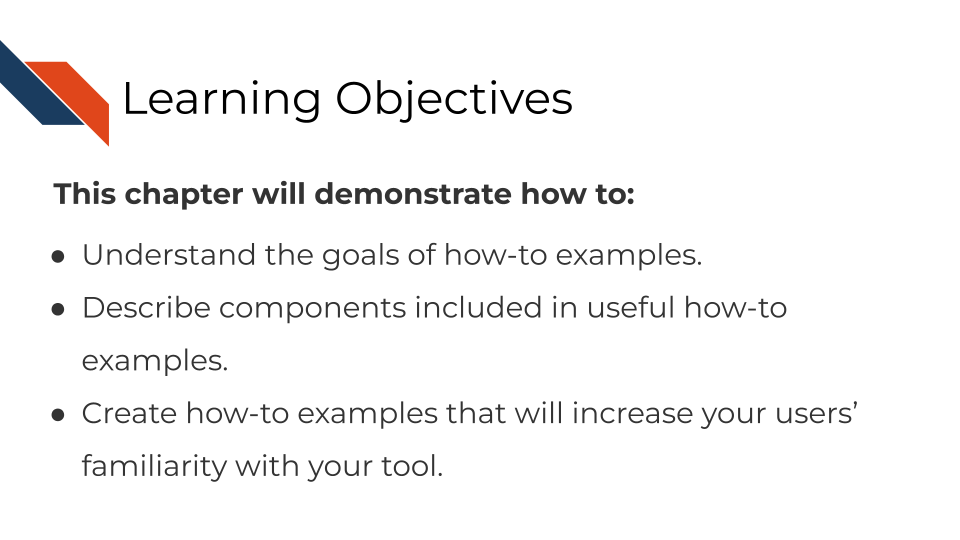
[GSEA](https://www.gsea-msigdb.org/gsea/doc/GSEAUserGuideFrame.html) introduces their users to multiple options of how they can run the tool and nicely use reassuring screenshots throughout to let their users know if they are on the right track ([Subramanian et al. 2005](#ref-Subramanian2005))!



## 5.4 Exercise: Create your own getting started section!

Use [the template getting started document](https://raw.githubusercontent.com/jhudsl/template-documentation/master/docs/getting_started_template.md) to start your own getting started section either by using the markdown template directly, or navigating to the MkDocs repository (or other set of documentation) you set up in the previous chapter.

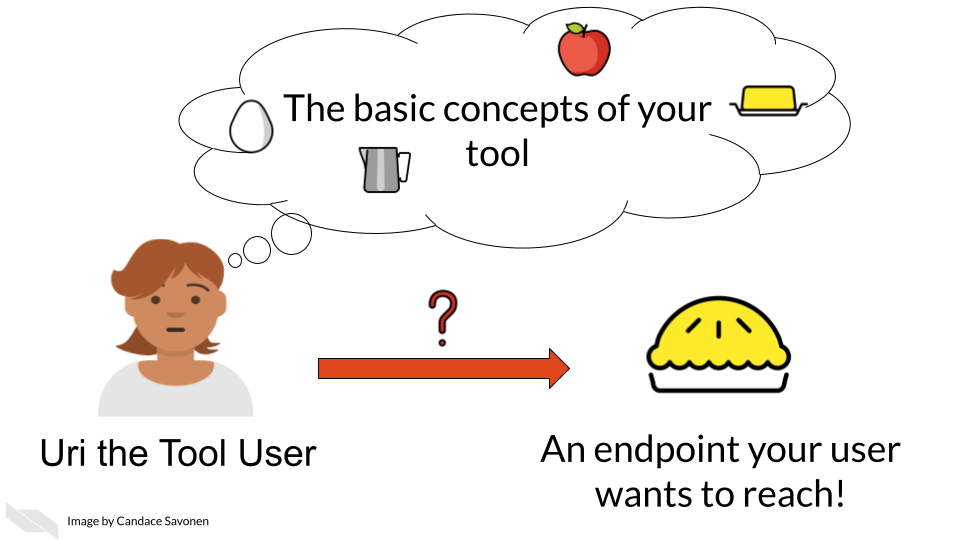
# 6 Creating helpful how-to examples



## 6.1 The goal of how-to examples

While getting started sections are geared toward brand-new users, how-to examples are geared toward intermediate users that have successfully installed your tool and now want to know more about what they can do with it. How-to examples can turn these moderately interested users into enthusiastic and invested fans of your tool!

How-to examples are like recipes in a cookbook. We can generally assume your user has found the kitchen, now give them sets of steps to create something awesome!

 [[5]](#footnote-131)

Our goal for our how-to examples is to show off the best and most exciting use cases of your tool!

Note – In some contexts, like Bioconductor, how-to examples are called vignettes – we consider these to be the same.

## 6.2 Characteristics of useful how-to examples

### 6.2.1 Should cover what your user wants to do!

Users won’t go through examples that demonstrate analyses they aren’t interested in. So although you may have your favorite pet functions of your tool, it doesn’t necessarily mean that those are the prime interests of your user (though perhaps after you create a good set of examples you can return to make examples of your favorites). You may want to do some asking around, or conduct some user research to find out what your users are most interested in. You also are not restricted to just one example, users love having a full library of how-to examples to choose from! That being said, for your own time and planning, you may want to start with the most common use cases to create examples for and then you can move to more fringe cases.

### 6.2.2 Every step is specifically given

Examples should explicitly give every step needed to reproduce your result. Keep a special look out for steps you complete that you may take for granted that your user would know. For example, **if your user needs to change to a specific directory to run a command, don’t assume that they will know that for sure.**

For command line based tools, provide the exact code your user needs to run. Ideally this example can be provided as a notebook or script so your user can run it directly.

In the case of a GUI, provide screenshots or a video tutorial.

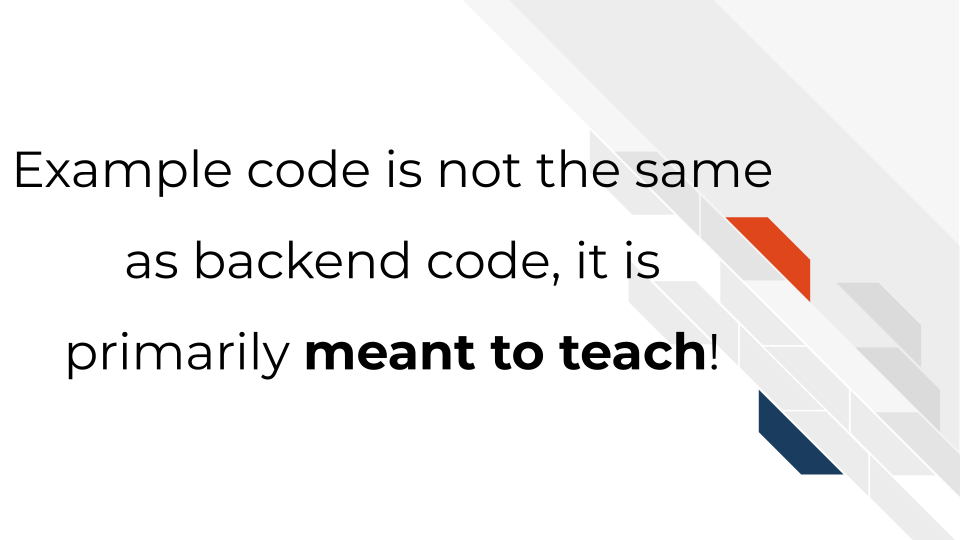
### 6.2.3 Any data needed to run the example is provided and explained

Getting data formatted correctly is another huge hurdle of users and although you should give guidance on how data should be formatted for your tool, your examples should not depend on your user’s data. Instead, provide your users with example data that your example code directly downloads (or is available through your GUI). This has the added benefit of being a positive control for when users are troubleshooting the formatting for their own data later on, but doesn’t force them to face that battle before they can follow your example.

Make sure your example adequately introduces the example: what are the measurements from, what was the goal of this dataset. And of course, link to the source of the data and cite it!

### 6.2.4 Example code is about the lesson not the output per se

Example code is not the same as backend code. Although example code should also be functional and work, its primarily meant to teach. Even more so than usual, code in examples should always prioritize clarity over cleverness or even brevity.



This means your examples should include the most easily readable code you can muster – this often means extra workshopping to reach peak clarity. Give commentary at each and every step – don’t assume your users understand your typical conventions. Also in the interest of being as readable as possible, try to stick to a styling conventions – s p a c i n g matters!

*Related to this, your examples’ code should model best practices.*

Pretend you are the model user of your tool – how should your users interact with your tool? This means keeping in mind the important basics:

* Emphasize reproducibility
* Include code comments
* Stick to a code style
* Be consistent with your conventions
* Pick good object names

For more tips on reproducible code:

* [How to Write a Reproducible Example - Hadley Wickham](https://gist.github.com/hadley/270442#how-to-write-a-reproducible-example)
* [Making your code reproducible](https://methodsblog.com/2017/12/06/making-your-code-reproducible/)
* [How to Make a Great R Reproducible Example](https://stackoverflow.com/questions/5963269/how-to-make-a-great-r-reproducible-example)

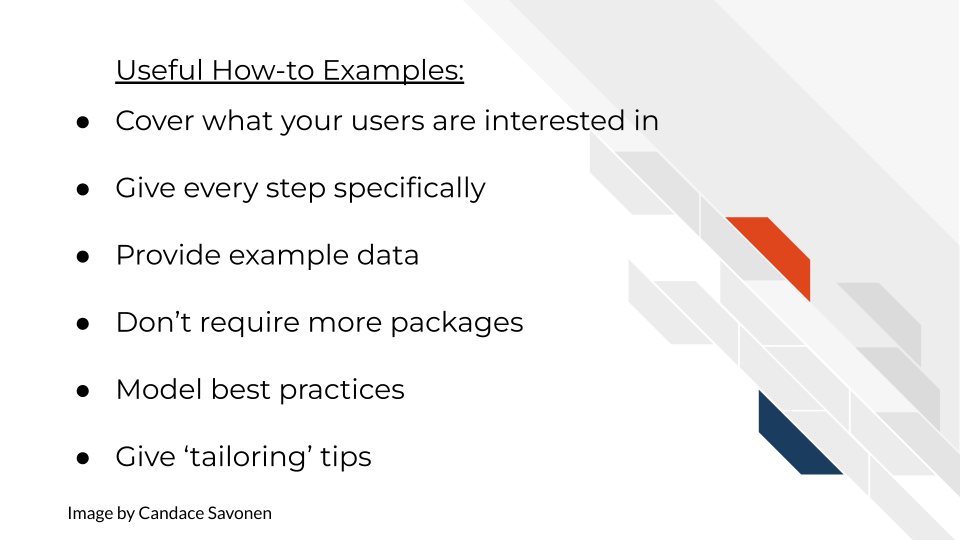
### 6.2.5 Try to stick to your own package and its dependencies

Your user already made it through the installation process, try not to make them add more installation steps to follow your examples unless absolutely necessary. If it is absolutely necessary, should a package be added as a dependency – can you have it automatically installed for the user if it is critical to common uses?

### 6.2.6 Give tips on how to tailor your examples

If your users follow along with your examples successfully, next they will probably need to tailor your examples to their own questions and data. Whether you intend it or not, your examples will probably be used as template framework for your user’s analysis. Knowing this, try to highlight places that users will absolutely need to change the code and other places where they might want to personalize it. Providing them with more resources about options and possibilities is always nice too.

**To recap:**



## 6.3 Good examples of How-to examples

[DESeq2 has excellent vignettes](http://www.bioconductor.org/packages/release/bioc/vignettes/DESeq2/inst/doc/DESeq2.html)! Love, Huber, and Anders ([2014](#ref-Love2014)) walk through the most common use cases of DESeq2, providing data and explaining the set up. They efficiently move into other scenarios, explaining common questions and areas of nuance along the way.

[QIIME2 also has an extensive set of examples](https://docs.qiime2.org/2021.2/tutorials/pd-mice/)! Bolyen et al. ([2019](#ref-Bolyen2019)) give a great set up and hypothesis to a question with a given dataset and walk through each step to answer that question. At the very end of the example they also provide the end result for comparison!

## 6.4 Exercise: Create your own how-to examples!

Use [the how-to example templates](https://raw.githubusercontent.com/jhudsl/template-documentation/master/docs/how_to_examples.md) to start your own how-to examples either by using the markdown template directly, or navigating to the MkDocs repository you set up in the previous chapter.

If your tool’s destination is Bioconductor or Galaxy, see our specific guidance on those repositories’ examples:

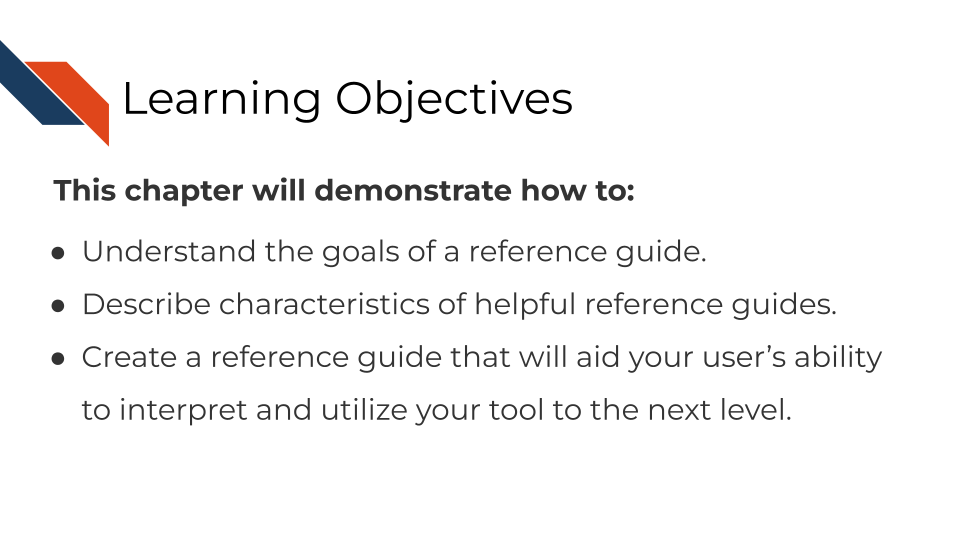
*For Bioconductor vignettes*:

* [Our Bioconductor specific how-to example templates](https://jhudatascience.org/template-documentation/bioconductor_guides/bioconductor_vignette_template.Rmd).
* [Bioconductor’s own vignette guidance](https://www.bioconductor.org/developers/package-guidelines/#Vignettes).
* [Guidance on writing RMarkdown Vignettes](https://bioconductor.org/packages/devel/bioc/vignettes/BiocStyle/inst/doc/AuthoringRmdVignettes.html) by Andrzej Oleś and Morgan ([2021](#ref-Oles2021)).

*For Galaxy vignettes*:

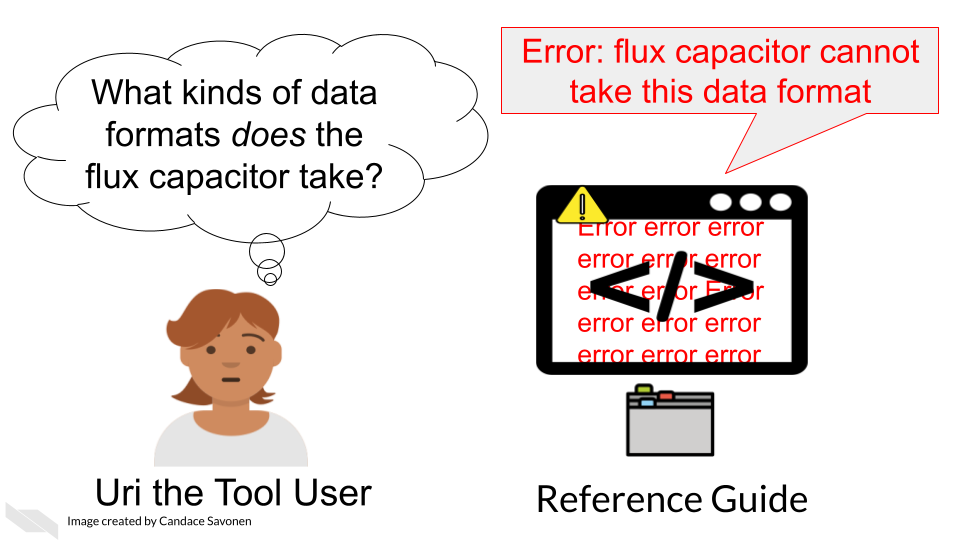
* [See this Galaxy tutorial for creating new tutorials!](https://training.galaxyproject.org/training-material/topics/contributing/tutorials/create-new-tutorial/tutorial.html)

# 7 Creating handy reference guides



## 7.1 The goal of a reference guide

Reference guides are the dictionary of your tool: they aren’t meant to be read front to back, but the best ones are easily searchable. Your user will have something in mind that they are trying to find information on – the quicker they can find it, the quicker their question can be answered.

 [[6]](#footnote-157)

Our goal for a reference guide is to be as comprehensive, navigable, and as always, as *clear* as possible.

## 7.2 Characteristics of handy reference guides

### 7.2.1 Is easy to find

As with our other documentation sections, no matter how well they are crafted, they are of no use if no one can find them. Make sure that a link to the reference guide is clear to find in a navigation bar and/or the main page.

If the reference guide is a part of a command line package, make sure that it is clear in your documentation what command you need to see it (e.g. --help).

### 7.2.2 Is searchable

Users will be digging into your reference guide looking for a specific entry. Making your reference guide alphabetical is a start. If you are able to make terms searchable that’s even better, but at the very least, if your reference guide is visually easy to be scan, that can serve a similar function.

### 7.2.3 Is comprehensive

All items are covered in the reference guide – every. single. thing. This includes all:

* Terms
* Functions
* Arguments
* Parameters
* Defaults
* Datasets or items included in the package
* Buttons (in the case of a GUI)

**The most useful entries in these reference guides…**

* Not only define the item, but tell how it relates to other items (and they have links where relevant).
* Discuss all the possible inputs that can be used as well as how to decide which input fits the user’s needs.
* Make any existing defaults very clear.
* Shows the usage of that item in context – some example lines of code go a long way.
* Tries to avoid the use of jargon, but if it is absolutely necessary to use a jargon-y term it links to information about the meaning of the term.

Take a look at [DESeq2’s reference guide](https://bioconductor.org/packages/release/bioc/manuals/DESeq2/man/DESeq2.pdf), Love, Huber, and Anders ([2014](#ref-Love2014)) do a great job at their entries – covering the right amounts of information and informing the user about how to decide what works for them.

### 7.2.4 Data formats are described

Perhaps after installation, getting data formatted correctly is one of the other very large hurdles users will need to deal with.

Ideally, your tool can use a data format that is common. But the more that your tool is particular about an odd data format, the more your documentation needs to be specific about what the odd data format looks like.

* Include example files for a positive control/example.
* Send your user to any tools they can use to convert their data format automatically.
* If its feasible, include an automatic data converter as a part of your tool.

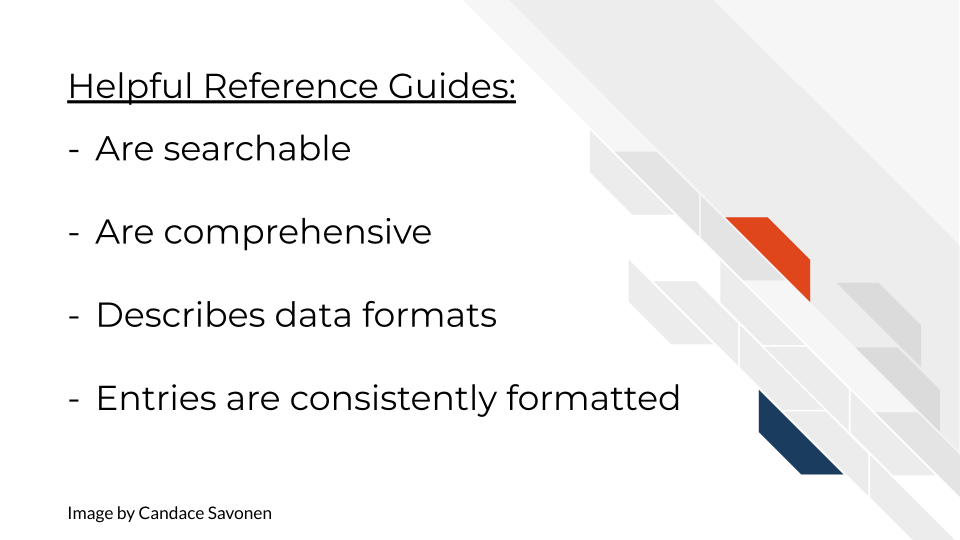
[GSEA has great descriptions of their data formats](https://www.gsea-msigdb.org/gsea/doc/GSEAUserGuideTEXT.htm#_Loading_Data) with examples of what the data formats look like.

### 7.2.5 Entries should have a consistent format

Consistency helps your users know what to expect and know where to find information! Each entry in the reference guide should have the same format and sections, in the same order.

Depending on the destination of your package, a consistent format may already be determined for you. Bioconductor packages have [specific guidance on these reference manuals](http://cran.fhcrc.org/doc/manuals/R-exts.html#Documenting-functions).

**To recap:**



## 7.3 Good examples of reference guides

[QIIME2 has a great reference guide](https://docs.qiime2.org/2021.2/glossary/)! Bolyen et al. ([2019](#ref-Bolyen2019)) cover all items and terms with lots of links to more information or related entries.

[Bioconductor packages](https://bioconductor.org/packages/3.12/bioc/) have a consistent reference guide format that the packages there adhere to which makes it easier for users to find what they are looking for once they are familiar with the format. A typical package’s reference guide [looks like this](https://bioconductor.org/packages/3.12/bioc/manuals/GenomicRanges/man/GenomicRanges.pdf).

## 7.4 Exercise: Create your own reference guide!

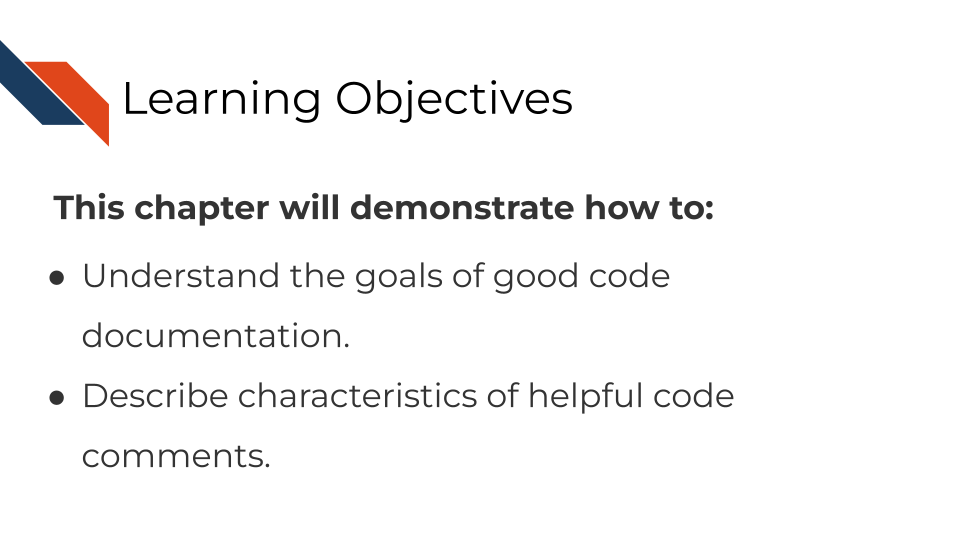
*For R package documentation:* Follow the advice from Hadley Wickham from the [R Packages book](https://r-pkgs.org/man.html) which includes using [roxygen2 package](https://cran.r-project.org/web/packages/roxygen2/vignettes/roxygen2.html) to automatically render those .Rd files!

If your tool’s destination is Bioconductor, see their specific guidance on [manual pages](https://bioconductor.org/developers/package-guidelines/#manpages).

*For Python package documentation:* Follow the docstrings guidance and instructions [here](https://realpython.com/documenting-python-code/).

For other general purposes, you can our [the reference guide template](https://raw.githubusercontent.com/jhudsl/template-documentation/master/docs/reference_guide_template.md) to start your own reference guide either by using the markdown template directly, or navigating to the MkDocs repository you set up in the previous chapter.

# 8 Creating clarifying code comments



## 8.1 The goal of a code documentation

Code documentation goes beyond your user. It’s a part of writing good code and helps your collaborators and ‘Future You’!

Most users will probably not look at your code directly – those who do are probably:

* Quite desperately troubleshooting something and were not able to find answers elsewhere.
* Are “power users” who are already pretty invested in your tool and want to get more out of it – may be helpful contributors too!

Good code comments are a part of writing good, readable code! Your code is more likely to stand the test of time for longer if others, including yourself in the future, can see what’s happening enough to trust it themselves. This will encourage others to use your code and help you maintain it!

‘Current You’ who is writing your code may know what is happening but ‘Future You’ will have no idea what ‘Current You’ was thinking ([Spielman, n.d.](#ref-Spielman)):

‘Future You’ comes into existence about one second after you write code, and has no idea what on earth Past You was thinking. Help out ‘Future You’ by adding lots of comments! ‘Future You’ next week thinks Today You is an idiot, and the only way you can convince ‘Future You’ that Today You is reasonably competent is by adding comments in your code explaining why Today You is actually not so bad.

Your code and your understanding of it will fade soon after you write it, leaving your hard work to deprecate. Code that works is a start, but **readable** AND working code is best!

Comments can help clarify at points where your code might need further explanation. The act of writing them can also help you think out your thought process and perhaps identify a better solution to the odd parts of your code.

## 8.2 Characteristics of clarifying code comments

What kinds of comments are most helpful for ‘Future You’ and others?

Spertus ([n.d.](#ref-Spertus2021)) lays out “Rules” for writing helpful code comments [in this article](https://stackoverflow.blog/2021/07/05/best-practices-for-writing-code-comments/) which we recommend you read.

Rule 1: Comments should not duplicate the code.

Rule 2: Good comments do not excuse unclear code.

Rule 3: If you can’t write a clear comment, there may be a problem with the code.

Rule 4: Comments should dispel confusion, not cause it.

Rule 5: Explain unidiomatic code in comments.

Rule 6: Provide links to the original source of copied code.

Rule 7: Include links to external references where they will be most helpful.

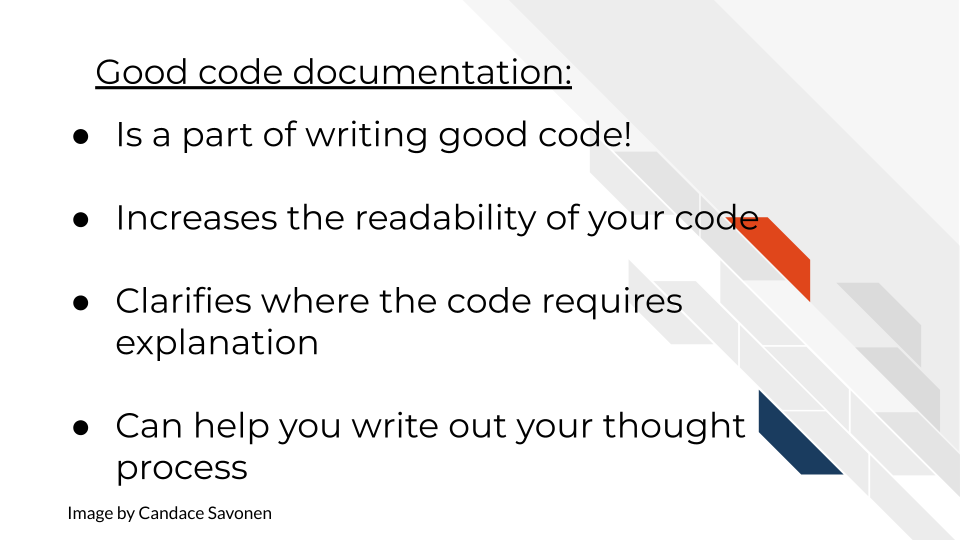
Rule 8: Add comments when fixing bugs.

Rule 9: Use comments to mark incomplete implementations.

Here are more resources about good code comments that have plenty of great discussions about how to up your code comment game.

* [Best Practices for Writing Code Comments](https://stackoverflow.blog/2021/07/05/best-practices-for-writing-code-comments/) by Spertus ([n.d.](#ref-Spertus2021)).
* [What Makes a Good Code Comment?](https://itnext.io/what-makes-a-good-code-comment-5267debd2c24) by Cronin ([2019](#ref-Cronin2019)).
* [The Value of Code Documentation](https://www.olioapps.com/blog/the-value-of-code-documentation/) by Meza ([2018](#ref-Meza2018)).
* [Some internet wisdom on R documentation](http://alyssafrazee.com/2014/04/20/rdocs.html) by Frazee ([2014](#ref-Frazee2014)).
* [How to Comment Your Code Like a Pro: Best Practices and Good Habits](https://www.elegantthemes.com/blog/wordpress/how-to-comment-your-code-like-a-pro-best-practices-and-good-habits) by Keeto ([2019](#ref-Keeton2019)).

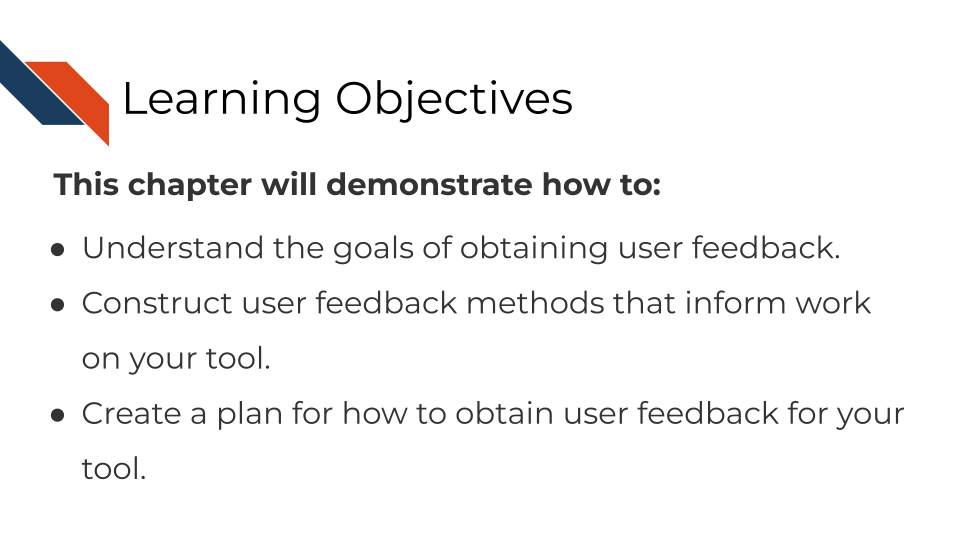
**In summary:**



## 8.3 Exercise: Evaluate your own code’s comments

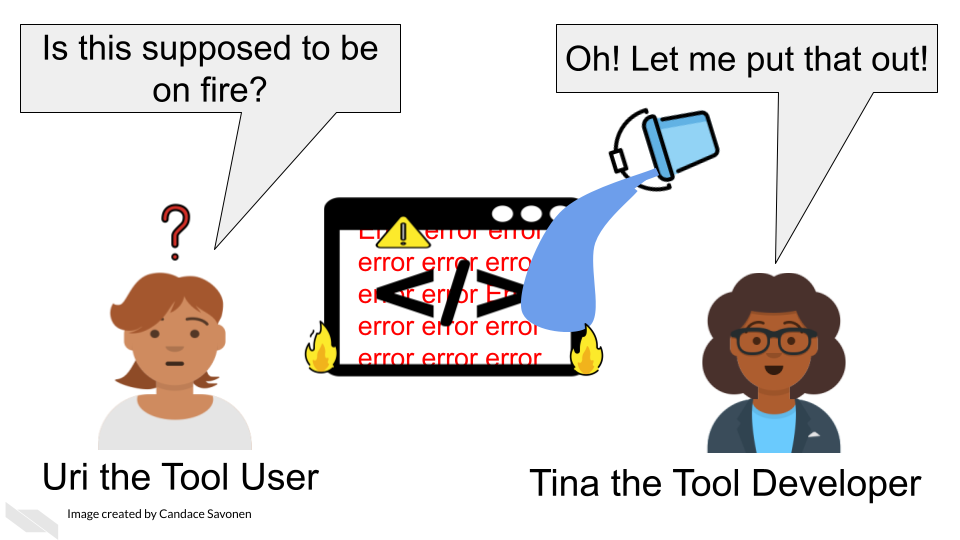
Take a look through some of your tool’s code and its comments. Are there places it could be clearer? Take some time to think about your coding process and how it could improve. [Code review](https://simpleprogrammer.com/why-code-reviews-make-better-code-teams/) is also an excellent way to improve code clarity and quality ([Bodner 2018](#ref-Bodner2018))!

# 9 Obtaining user feedback



## 9.1 The goal of user feedback

How do you know if your code is working? You test it and get feedback! Similarly, how do you know if your tool is working for your user? Ask them for feedback!

 [[7]](#footnote-194)

## 9.2 How to collect informative user feedback

### 9.2.1 User contact method

At the most basic level, you need to provide your users a way to alert you if something with your tool is not working appropriately.

It may sound disappointing that a user has found a problem but this is something to be happy about!

1. This means someone is interested and invested in using your tool!
2. They’ve identified a way to improve your tool to make it even better!

Providing a method of contact to your users doesn’t mean you need to give users your personal email. In fact, that is probably not the most practical way to keep user queries organized.

*Example contact method ideas*:

* Have a link to a form for users to submit ([Google forms](https://www.google.com/forms/about/) are free).
* Direct users to file a [GitHub issue](https://docs.github.com/en/github/managing-your-work-on-github/about-issues).
* Have a separate email inbox that you have a notification set up for.
* Have a [Slack channel](https://slack.com/) that you direct users to.

Whatever method of contact you provide your users, just make sure its something that works for you and your team to respond to.

### 9.2.2 Interviews and usability testing

In whatever contact method you settle on, give your users a way to indicate if they are willing to chat with you or do testing to provide even more feedback.

Depending on your time and resources you can do a lot with usability testing. [This excellent article](https://uxstudioteam.com/ux-blog/usability-testing/) by Csontos ([2019](#ref-Csontos2019)) takes us through how to conduct usability testing which we will echo the main points of here.

What to use usability testing for:

* Identifying main issues in the usability of a product
* Checking if users understand the steps to carry out a task and the navigation
* Observing how easily and quickly they accomplish tasks
* Validating the value proposition of an app or website – do your potential customers understand it?

*To summarize the steps laid out by Csontos (*[*2019*](#ref-Csontos2019)*) for conducting usability research*:

[**Step 1) Plan out your study**](https://uxstudioteam.com/ux-blog/usability-testing/#Step_1_Plan_your_study) - clarify what you want to learn and write a plan and script.

* What main questions about your tool do you have? – try to narrow down to one or two priority questions that you can design a test for.
* What tasks would a user participant perform that would best inform you about your questions?
* Write out a script that you would use to conduct testing.

*See more advice about writing usability testing scripts here:*

* [Steve Krug’s usability testing script template](http://sensible.com/downloads/test-script-web.pdf) from Krug ([2010](#ref-Krug2010)).
* [Gitlab’s advice about usability testing script](https://about.gitlab.com/handbook/engineering/ux/ux-research-training/writing-usability-testing-script/) from GitLab ([n.d.](#ref-Gitlab)).
* [Gitlab’s usability testing template script](https://docs.google.com/document/d/1_5Qu2JR9QE5LE6cK4eq9yJs-nXv2rlWWifcjacaiWdI/edit).

**Pilot your test with colleague(s)**

Before you use your test on a potential user, borrow a colleague or other internal person to run through your plan and script for a rehearsal. Though your colleagues are likely not the same as your typical user, running your testing plan on someone who works with you first will allow you to identify bugs in your testing plan. Run this pilot test with your colleague as if you are doing the real thing and try to be as prepared as possible!

[**Step 2) Prepare for the testing session**](https://uxstudioteam.com/ux-blog/usability-testing/#Step_3_Lead_the_test)

[Use this checklist](http://sensible.com/downloads/checklists.pdf) from Krug ([2010](#ref-Krug2010)) to check if you are prepared!

You can use [these prompts](https://sensible.com/downloads/things-a-therapist-would-say.pdf) from Krug ([2010](#ref-Krug2010)) to help you know what to say to help create a non-judgmental atmosphere for testing.

[**Step 3) Find user participants**](https://uxstudioteam.com/ux-blog/usability-testing/#Step_2_User_test_participants)

Who is interested in available for being your test subjects? You might be able recruit people from setting up a feedback form, but you also may use word of mouth and ask around. You could also try advertising your need for participants through Twitter, LinkedIn or some other social media platform.

Note: [Calendly](https://calendly.com/) is very handy tool for scheduling usability testing sessions!

**Step 4) Run the test!**

Now that you’ve rehearsed your test and scheduled a testing session with a user, you are ready to run the tests and collect information.

Make sure to check that they are okay with you recording the session (if that is what you want to do).

A very important point from Csontos ([2019](#ref-Csontos2019)):



[**Step 4) Analyze & Report**](https://uxstudioteam.com/ux-blog/usability-testing/#Step_4_Analyze_Report)

Sit down with your notes and recordings, look for patterns.

*More reading on usability testing:*

* [How to run moderated usability testing](https://www.hotjar.com/usability-testing/process-examples/) HotJar ([2019](#ref-Hotjar2020)).
* [Usability Testing 101](https://www.nngroup.com/articles/usability-testing-101/) Moran ([2019](#ref-Moran2019)).
* See Krug ([2010](#ref-Krug2010)) ’s list of [downloadable usability resources](https://sensible.com/download-files/).
* See this [case study](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002554) for a user designer conducting user research on a bioinformatics tool ([Pavelin et al. 2012](#ref-Pavelin2012)).
* For even more reading, you could buy (this book by [Krug 2010](#ref-Krug2010))(<https://www.amazon.com/Rocket-Surgery-Made-Easy-Yourself-ebook/dp/B002UXRGNO>).

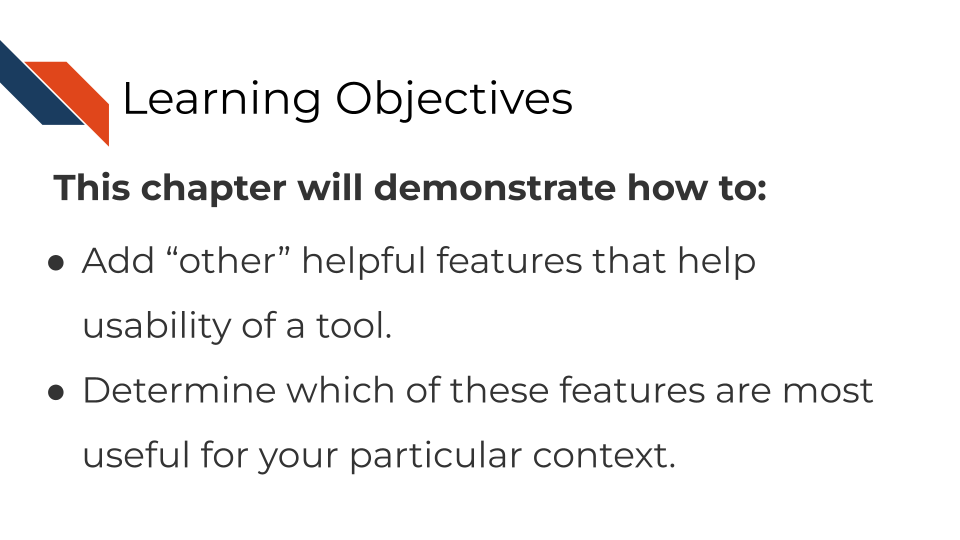
## 9.3 Exercise 1: Create a plan for user feedback for your tool

Set up a method of contact for your users. We have a mock feedback user form [set up here](https://docs.google.com/forms/d/1erbaH2k8cra0A2GB6W9Da0tqJCT41ZPlCmHXpKRcMLk/edit?usp=sharing). This form shouldn’t be used as is, but could be tailored to more specifically ask questions about your tool that you are looking to get information about.

## 9.4 Exercise 2: Determine a plan for usability testing

Go through [All You Need to Know to Run Successful Usability Testing](https://uxstudioteam.com/ux-blog/usability-testing/) and construct a plan for usability testing following the steps we discussed above from the article ([Csontos 2019](#ref-Csontos2019)). Think about what in your tool you want to test and write a plan for it.

# 10 Other helpful features



## 10.1 The goal of these “other features”

There are some features that can increase the usability of your tool, but would probably not work well as standalone documentation. The extent to which these other kinds of documentation are applicable to your software tool, is dependent on your own context and resources.

In this chapter, we will discuss these other features and when you might want to consider adding them to your set of documentation.

## 10.2 FAQs

As your user base grows and you start to get recurring questions, you may want to try to head off future users asking the same questions by incorporating clarifications into your documentation. Ideally you would make sure the most *frequently* asked questions would be addressed in the getting started section or perhaps your how-to examples.

But, there may be other recurring questions that you don’t have a logical place to describe the answer to, but would be helpful for users to know about. In this instance, a FAQ page may be what you need.

FAQ’S can easily get disorganized and out of hand as they grow. Try to keep similar questions underneath larger headings.

*More reading on FAQ pages:*

* [Making a killer FAQ page](https://www.socialmediatoday.com/content/10-tips-creating-killer-faq-page) ([Moon 2011](#ref-Moon2011)).
* [25 of the Best Examples of Effective FAQ Pages](https://www.searchenginejournal.com/best-faq-page-examples/267709/) ([Wilson 2020](#ref-Wilson2020)).

## 10.3 Cheatsheets

If your tool has a lot of items that are handy but tough to keep track of a cheatsheet may be especially helpful for your users.  
Cheatsheets are like reference guides but shorter (one or two pages) and more aesthetically pleasing. Users like cheatsheets for handy referencing! Ideally your cheatsheet won’t be identical to your reference guide but instead will highlight the most commonly used/most helpful items!

*More reading on cheatsheets:*

* For inspiration and examples of nice cheatsheets, take a look through [RStudio’s cheatsheets](https://www.rstudio.com/resources/cheatsheets/) ([RStudio Team 2020](#ref-RStudioTeam2020)).
* If you use Overleaf, there are [template cheatsheets you can use here](https://www.overleaf.com/gallery/tagged/cheat-sheet).

## 10.4 Videos

Particularly, if your tool has a graphics user interface, sometimes videos can be the best way to demonstrate getting started information. Videos are not scannable so in general, shouldn’t be depended on as the main source of information. Your users may have a quick question they want answered quickly and videos require time to watch. Users may be frustrated if the video didn’t answer their question after watching it, if there is no other form of documentation.

That being said, videos are very helpful for GUI’s in particular or for introducing basic concepts. And the majority of users who like visuals will appreciate that aspect of videos.

*More reading on making videos:*

* [Creating Instructional Videos](https://www.techsmith.com/blog/instructional-videos/) ([Simon 2020](#ref-Simon2020)).
* [7 Great Tools for Creating Your Own Instructional Videos](https://helpdeskgeek.com/free-tools-review/7-great-tools-for-creating-your-own-video-tutorials/) ([Trounce 2019](#ref-Trounce2019)).
* [See Galaxy’s tutorial videos as examples](https://training.galaxyproject.org/training-material/topics/introduction/) ([Afgan et al. 2018](#ref-Afgan2018)).

## 10.5 Forums/Knowledge Base

Forums and knowledge bases can be a handy way to keep public records of questions and answers and reduce the number of times your team has to respond to a repeated question. Forums are mostly useful if your tool grows a big user base (hooray!) and your team has trouble keeping up with inquiries. Advanced users may also be able to help answer questions of newer users which further frees up your team for other issues. Making this knowledge base be searchable can reduce the support load of your team.

If you don’t have the resources to host your own forum, a Slack channel could serve similar purposes or you can look into the options discussed in these articles:

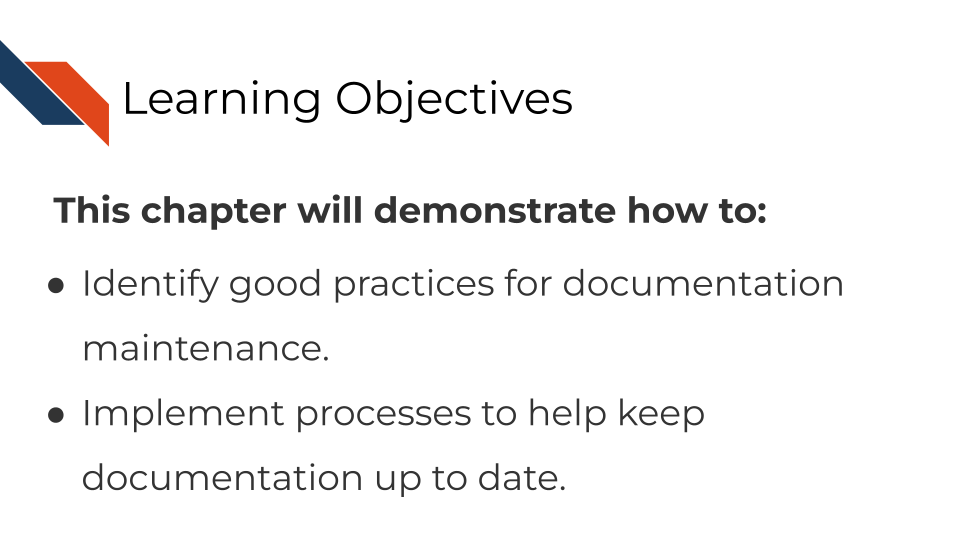
*More reading on knowledge bases and forums:*

* [Knowledge Base Options](https://herothemes.com/blog/best-knowledge-base-software/) ([Herothemes 2020](#ref-Herothemes)).
* [Building an Online Community](https://geekflare.com/online-community-software/)([Rehan 2019](#ref-Rehan2019)).
* [A Quick Guide for Building a Successful Bioinformatics Community](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003972) ([Budd et al. 2015](#ref-Budd2015)).

## 10.6 Exercise: Determine which (if any) of these features would work well for your tool

Now that we’ve discussed some of other features that are helpful for your users, think about which of these might be helpful for your own tool, keeping mind your current documentation set up, user base, and general format. The answer might be multiple or none of these features – only you and your team can answer this!

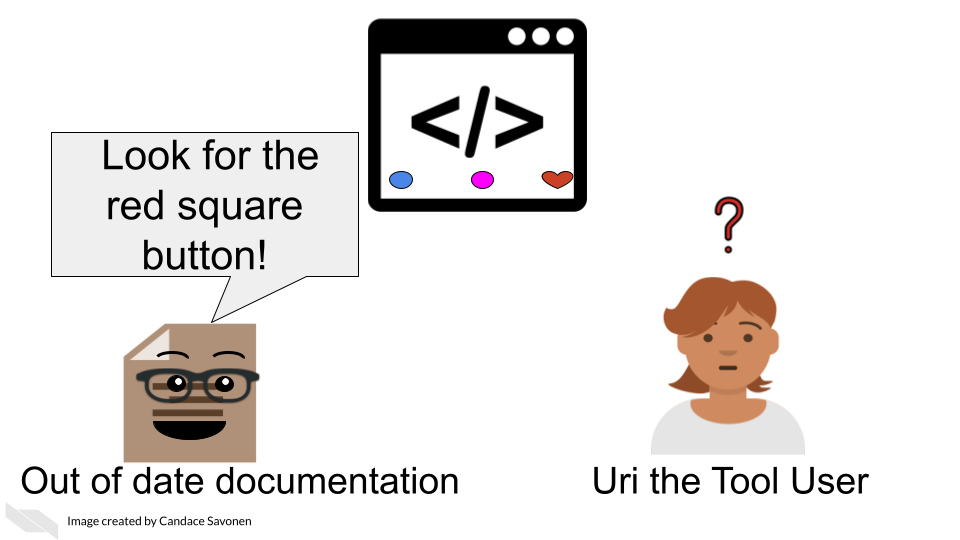
# 11 How to keep your documentation up to date



## 11.1 The goal of documentation maintenance

Perhaps you’ve been making improvements or otherwise updating your software tool. That’s excellent and you deserve a big kudos for continuing maintenance on your tool!

But your work is not done yet. For each (user-facing) update you make to the tool, you should also make a documentation update. As a user, the only thing worse than having a tool with no documentation at all is having a tool with documentation that is out of date or otherwise incorrect.



If documentation updates aren’t prioritized, your tool can easily get several versions ahead leaving the documentation you carefully crafted rather useless and misleading.

## 11.2 Keep your documentation in one, version-controlled place

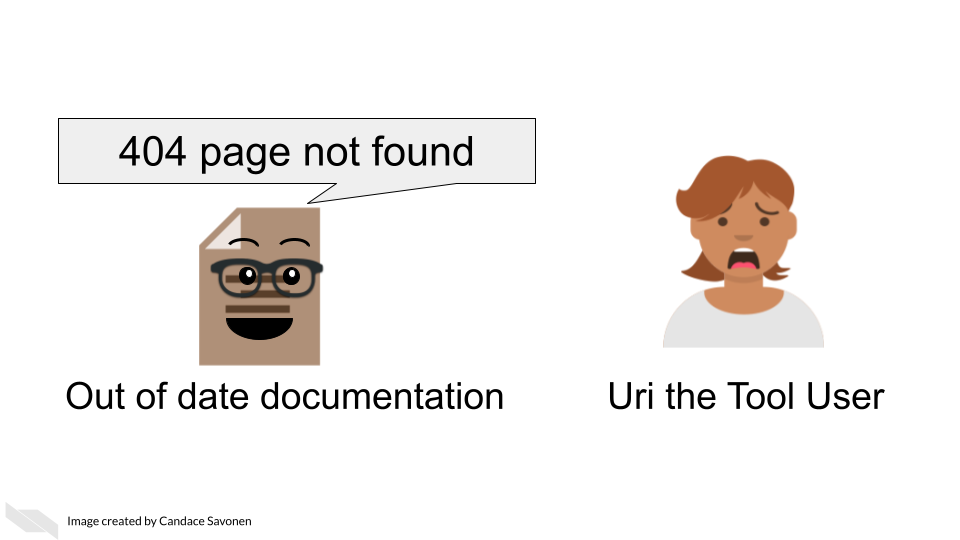
Presumably you have some sort of process for version controlling your tool updates (we assume GitHub but could be other services). Ideally, your documentation should be version controlled similarly and, if appropriate, in the same place. The easier you make it on yourself to update your documentation, the more likely future you will be at updating it successfully! It’s worth spending time thinking about your own development process and how you can make it easier on yourself and your team for longer-term better maintained documentation.

## 11.3 Do not consider a tool fix done before its relevant documentation update is also completed

However you track your tasks, also track your documentation issues and always pair a software fix with a documentation fix – or at least check if it affects anything user-facing. To help you remind you of this, you may want to use an [issue template](https://docs.github.com/en/communities/using-templates-to-encourage-useful-issues-and-pull-requests/configuring-issue-templates-for-your-repository) (if you use GitHub) and make sure that issue template includes a reminder to update documentation.

## 11.4 Make sure links work

A very simple but all too common problem with out of date documentation is broken links!



You can catch these broken links by manually clicking on all your links, but sometimes broken links will still slip through the cracks anyway! There are GitHub actions and other automated tools that can check your URLs for you. Take advantage of automation to do this for you so you can save your time an effort for other improvements to your tool and documentation!

Here’s some options for automated URL-checking:

* [GitHub action: urlchecker-action](https://github.com/marketplace/actions/urlchecker-action).
* [GitHub action: URL checker](https://github.com/marketplace/actions/url-checker).
* [6 Tools to Find Broken Links on Your Website](https://www.outlookstudios.com/tools-to-find-broken-links-on-your-website/) ([Studios 2020](#ref-OutlookStudios2020)).

## 11.5 Exercise 1: Add a reminder for documentation updates to your task manager

* If you use GitHub, add an [issue template](https://docs.github.com/en/communities/using-templates-to-encourage-useful-issues-and-pull-requests/configuring-issue-templates-for-your-repository) that has a reminder to update documentation.
* If you use something else for task management, look for some other way to remind yourself (and your fellow developers on the project) to keep documentation up-to-date for each change.

## 11.6 Exercise 2: Implement a URL checker

* If you use GitHub for your documentation, add a url checking GitHub action to your repository. We used a url-checker GitHub action for developing this course! You can see [ours here for an example](https://github.com/jhudsl/Documentation_and_Usability/blob/main/.github/workflows/url-checker.yml).
* If you use something else for version control, look into URL checkers that you can easily implement into your development process.

# About the Authors

These credits are based on our [course contributors table guidelines](https://github.com/jhudsl/OTTR_Template/wiki/How-to-give-credits).

***In*** [***memory***](https://galaxyproject.org/jxtx/) ***of*** [***James Taylor***](https://genomebiology.biomedcentral.com/articles/10.1186/s13059-020-02016-0)***, who was instrumental in initiating this project.***

| Credits | Names |
| --- | --- |
| Lead Content Instructor | [Candace Savonen](https://www.cansavvy.com/) |
| Content Editors/Reviewer | [Carrie Wright](https://carriewright11.github.io/), [Sarah Wheelan](https://www.hopkinsmedicine.org/profiles/details/sarah-wheelan) |
| Content Directors | [Jeff Leek](https://jtleek.com/), [Sarah Wheelan](https://www.hopkinsmedicine.org/profiles/details/sarah-wheelan) |
| Content Consultant | [Deepa Prasad](https://www.linkedin.com/in/dvprasad991) |
| Template Publishing Engineer | [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/) |
| **Production** |  |
| Content Publisher | [Ira Gooding](https://publichealth.jhu.edu/faculty/4130/ira-gooding) |
| Content Publishing Reviewers | [Ira Gooding](https://publichealth.jhu.edu/faculty/4130/ira-gooding), [Carrie Wright](https://carriewright11.github.io/) |
| **Technical** |  |
| Course Publishing Engineer | [Candace Savonen](https://www.cansavvy.com/) |
| Template Publishing Engineers | [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/) |
| Publishing Maintenance Engineer | [Candace Savonen](https://www.cansavvy.com/) |
| Technical Publishing Stylists | [Carrie Wright](https://carriewright11.github.io/), [Candace Savonen](https://www.cansavvy.com/) |
| Package Developers ([Leanbuild](https://github.com/jhudsl/ottrpal))[John Muschelli](https://johnmuschelli.com/), [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/) |  |
| **Art and Design** |  |
| Illustrator | [Candace Savonen](https://www.cansavvy.com/) |
| **Funding** |  |
| Funder | [National Cancer Institute (NCI)](https://www.cancer.gov/) UE5 CA254170 |
| Funding Staff | Emily Voeglein, Fallon Bachman |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value   
## version R version 4.0.2 (2020-06-22)  
## os Ubuntu 20.04.3 LTS   
## system x86\_64, linux-gnu   
## ui X11   
## language (EN)   
## collate en\_US.UTF-8   
## ctype en\_US.UTF-8   
## tz Etc/UTC   
## date 2022-03-04   
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date lib source   
## assertthat 0.2.1 2019-03-21 [1] RSPM (R 4.0.3)   
## bookdown 0.24 2022-02-15 [1] Github (rstudio/bookdown@88bc4ea)   
## callr 3.4.4 2020-09-07 [1] RSPM (R 4.0.2)   
## cli 2.0.2 2020-02-28 [1] RSPM (R 4.0.0)   
## crayon 1.3.4 2017-09-16 [1] RSPM (R 4.0.0)   
## desc 1.2.0 2018-05-01 [1] RSPM (R 4.0.3)   
## devtools 2.3.2 2020-09-18 [1] RSPM (R 4.0.3)   
## digest 0.6.25 2020-02-23 [1] RSPM (R 4.0.0)   
## ellipsis 0.3.1 2020-05-15 [1] RSPM (R 4.0.3)   
## evaluate 0.14 2019-05-28 [1] RSPM (R 4.0.3)   
## fansi 0.4.1 2020-01-08 [1] RSPM (R 4.0.0)   
## fs 1.5.0 2020-07-31 [1] RSPM (R 4.0.3)   
## glue 1.6.1 2022-01-22 [1] CRAN (R 4.0.2)   
## htmltools 0.5.0 2020-06-16 [1] RSPM (R 4.0.1)   
## knitr 1.33 2022-02-15 [1] Github (yihui/knitr@a1052d1)   
## lifecycle 1.0.0 2021-02-15 [1] CRAN (R 4.0.2)   
## magrittr 2.0.2 2022-01-26 [1] CRAN (R 4.0.2)   
## memoise 1.1.0 2017-04-21 [1] RSPM (R 4.0.0)   
## pkgbuild 1.1.0 2020-07-13 [1] RSPM (R 4.0.2)   
## pkgload 1.1.0 2020-05-29 [1] RSPM (R 4.0.3)   
## prettyunits 1.1.1 2020-01-24 [1] RSPM (R 4.0.3)   
## processx 3.4.4 2020-09-03 [1] RSPM (R 4.0.2)   
## ps 1.3.4 2020-08-11 [1] RSPM (R 4.0.2)   
## purrr 0.3.4 2020-04-17 [1] RSPM (R 4.0.3)   
## R6 2.4.1 2019-11-12 [1] RSPM (R 4.0.0)   
## remotes 2.2.0 2020-07-21 [1] RSPM (R 4.0.3)   
## rlang 0.4.10 2022-02-15 [1] Github (r-lib/rlang@f0c9be5)   
## rmarkdown 2.10 2022-02-15 [1] Github (rstudio/rmarkdown@02d3c25)  
## rprojroot 2.0.2 2020-11-15 [1] CRAN (R 4.0.2)   
## sessioninfo 1.1.1 2018-11-05 [1] RSPM (R 4.0.3)   
## stringi 1.5.3 2020-09-09 [1] RSPM (R 4.0.3)   
## stringr 1.4.0 2019-02-10 [1] RSPM (R 4.0.3)   
## testthat 3.0.1 2022-02-15 [1] Github (R-lib/testthat@e99155a)   
## usethis 2.1.5.9000 2022-02-15 [1] Github (r-lib/usethis@57b109a)   
## withr 2.3.0 2020-09-22 [1] RSPM (R 4.0.2)   
## xfun 0.26 2022-02-15 [1] Github (yihui/xfun@74c2a66)   
## yaml 2.2.1 2020-02-01 [1] RSPM (R 4.0.3)   
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library

# References

Afgan, Enis, Dannon Baker, Bérénice Batut, Marius van den Beek, Dave Bouvier, Martin Čech, John Chilton, et al. 2018. “The Galaxy Platform for Accessible, Reproducible and Collaborative Biomedical Analyses: 2018 Update.” *Nucleic Acids Research* 46 (W1): W537–44. <https://doi.org/10.1093/nar/gky379>.

Alexakis, Alex. n.d. “You’re Not Your User.” <https://blog.prototypr.io/dear-product-managers-youre-not-your-users-e5739627cbc8>.

Andrzej Oleś, Wolfgang Huber, and Martin Morgan. 2021. “Authoring r Markdown Vignettes.” <https://bioconductor.org/packages/devel/bioc/vignettes/BiocStyle/inst/doc/AuthoringRmdVignettes.html>.

Babich, Nick. 2019. “The 4 Golden Rules of UI Design.” <https://xd.adobe.com/ideas/process/ui-design/4-golden-rules-ui-design/>.

———. n.d. “F-Shaped Pattern for Reading Content.” <https://uxplanet.org/f-shaped-pattern-for-reading-content-80af79cd3394>.

Bodner, Herbet. 2018. “10 Reasons Why Code Reviews Make Better Code and Better Teams.” <https://simpleprogrammer.com/why-code-reviews-make-better-code-teams/>.

Bolchini, Davide, Anthony Finkelstein, Vito Perrone, and Sylvia Nagl. 2008. “Better Bioinformatics Through Usability Analysis.” *Bioinformatics* 25 (3): 406–12. <https://doi.org/10.1093/bioinformatics/btn633>.

Bolyen, Evan, Jai Ram Rideout, Matthew R. Dillon, Nicholas A. Bokulich, Christian C. Abnet, Gabriel A. Al-Ghalith, Harriet Alexander, et al. 2019. “Reproducible, Interactive, Scalable and Extensible Microbiome Data Science Using QIIME 2.” *Nature Biotechnology* 37 (8): 852–57. <https://doi.org/10.1038/s41587-019-0209-9>.

Budd, Aidan, Manuel Corpas, Michelle D. Brazas, Jonathan C. Fuller, Jeremy Goecks, Nicola J. Mulder, Magali Michaut, B. F. Francis Ouellette, Aleksandra Pawlik, and Niklas Blomberg. 2015. “A Quick Guide for Building a Successful Bioinformatics Community.” Edited by Amarda Shehu. *PLOS Computational Biology* 11 (2): e1003972. <https://doi.org/10.1371/journal.pcbi.1003972>.

Cronin, Mike. 2019. “What Makes a Good Code Comment?” <https://itnext.io/what-makes-a-good-code-comment-5267debd2c24>.

Csontos, Anna-Zsófia. 2019. “All You Need to Know to Run Successful Usability Testing.” <https://uxstudioteam.com/ux-blog/usability-testing/>.

Duck, Geraint, Goran Nenadic, Michele Filannino, Andy Brass, David L. Robertson, and Robert Stevens. 2016. “A Survey of Bioinformatics Database and Software Usage Through Mining the Literature.” Edited by Shoba Ranganathan. *PLOS ONE* 11 (6): e0157989. <https://doi.org/10.1371/journal.pone.0157989>.

Frazee, Alyssa. 2014. “Some Internet Wisdom on r Documentation.” <http://alyssafrazee.com/2014/04/20/rdocs.html>.

GitLab. n.d. “Writing a Usability Testing Script.” <https://about.gitlab.com/handbook/engineering/ux/ux-research-training/writing-usability-testing-script/>.

Herothemes. 2020. “8 Best Knowledge Base Software Compared (Pros & Cons).” <https://herothemes.com/blog/best-knowledge-base-software/>.

HotJar. 2019. “How to Run Moderated Usability Testing.” <https://www.hotjar.com/usability-testing/process-examples/>.

Keeto, B. J. 2019. “How to Comment Your Code Like a Pro: Best Practices and Good Habits.” <https://www.elegantthemes.com/blog/wordpress/how-to-comment-your-code-like-a-pro-best-practices-and-good-habits>.

Krug, Steve. 2010. “Rocket Surgery Made Easy:the Do-It-Yourself Guide to Finding and Fixing Usability Problems.” <https://sensible.com/downloads/things-a-therapist-would-say.pdf>.

Love, Michael I, Wolfgang Huber, and Simon Anders. 2014. “Moderated Estimation of Fold Change and Dispersion for RNA-Seq Data with DESeq2.” *Genome Biology* 15 (12). <https://doi.org/10.1186/s13059-014-0550-8>.

Mangul, Serghei, Lana S. Martin, Eleazar Eskin, and Ran Blekhman. 2019. “Improving the Usability and Archival Stability of Bioinformatics Software.” *Genome Biology* 20 (1). <https://doi.org/10.1186/s13059-019-1649-8>.

Mangul, Serghei, Thiago Mosqueiro, Richard J. Abdill, Dat Duong, Keith Mitchell, Varuni Sarwal, Brian Hill, et al. 2019. “Challenges and Recommendations to Improve the Installability and Archival Stability of Omics Computational Tools.” *PLOS Biology* 17 (6): e3000333. <https://doi.org/10.1371/journal.pbio.3000333>.

Matos, Paula de, Jennifer A Cham, Hong Cao, Rafael Alcántara, Francis Rowland, Rodrigo Lopez, and Christoph Steinbeck. 2013. “The Enzyme Portal: A Case Study in Applying User-Centred Design Methods in Bioinformatics.” *BMC Bioinformatics* 14 (1): 103. <https://doi.org/10.1186/1471-2105-14-103>.

Meza, Frank. 2018. “The Value of Code Documentation.” <https://www.olioapps.com/blog/the-value-of-code-documentation/>.

Molder, F, KP Jablonski, B Letcher, MB Hall, CH Tomkins-Tinch, V Sochat, J Forster, et al. 2021. “Sustainable Data Analysis with Snakemake [Version 1; Peer Review: 1 Approved, 1 Approved with Reservations].” *F1000Research* 10 (33). <https://doi.org/10.12688/f1000research.29032.1>.

Moon, Garrett. 2011. “10 Tips for Creating a Killer FAQ Page.” <https://www.socialmediatoday.com/content/10-tips-creating-killer-faq-page>.

Moran, Kate. 2019. “Usability Testing 101.” <https://www.nngroup.com/articles/usability-testing-101/>.

Mulder, Nicola, Russell Schwartz, Michelle D. Brazas, Cath Brooksbank, Bruno Gaeta, Sarah L. Morgan, Mark A. Pauley, et al. 2018. “The Development and Application of Bioinformatics Core Competencies to Improve Bioinformatics Training and Education.” Edited by Olga G. Troyanskaya. *PLOS Computational Biology* 14 (2): e1005772. <https://doi.org/10.1371/journal.pcbi.1005772>.

Pavelin, Katrina, Jennifer A. Cham, Paula de Matos, Cath Brooksbank, Graham Cameron, and Christoph Steinbeck. 2012. “Bioinformatics Meets User-Centred Design: A Perspective.” Edited by Philip E. Bourne. *PLoS Computational Biology* 8 (7): e1002554. <https://doi.org/10.1371/journal.pcbi.1002554>.

Porter, Joshua. n.d. “Principles of User Interface Design.” <http://bokardo.com/principles-of-user-interface-design/>.

Quinlan, Aaron R., and Ira M. Hall. 2010. “BEDTools: A Flexible Suite of Utilities for Comparing Genomic Features.” *Bioinformatics* 26 (6): 841–42. <https://doi.org/10.1093/bioinformatics/btq033>.

Rehan, Adnan. 2019. “Build Online Community and Forum with These 8 Best Software.” <https://geekflare.com/online-community-software/>.

RStudio Team. 2020. *RStudio: Integrated Development Environment for r*. Boston, MA: RStudio, PBC. <http://www.rstudio.com/>.

Simon, Justin. 2020. “The Ultimate Guide to Easily Make Instructional Videos.” <https://www.techsmith.com/blog/instructional-videos/>.

Soegaard, Mads. n.d. “Visual Hierarchy: Organizing Content to Follow Natural Eye Movement Patterns.” <https://www.interaction-design.org/literature/article/visual-hierarchy-organizing-content-to-follow-natural-eye-movement-patterns>.

Spertus, Ellen. n.d. “Best Practices for Writing Code Comments.” <https://stackoverflow.blog/2021/07/05/best-practices-for-writing-code-comments/>.

Spielman, Stephanie. n.d. “Introduction to r - Cb2r Data Science Workshop, Summer 2020.” <https://github.com/sjspielman/cb2r-ds-summer2020/blob/71cb11277e7383292bf727841ab5fa4ed43cfcbe/resources/introduction_to_R.Rmd#L92>.

Studios, Outlook. 2020. “6 Tools to Find Broken Links on Your Website.” <https://www.outlookstudios.com/tools-to-find-broken-links-on-your-website/>.

Subramanian, Aravind, Pablo Tamayo, Vamsi K. Mootha, Sayan Mukherjee, Benjamin L. Ebert, Michael A. Gillette, Amanda Paulovich, et al. 2005. “Gene Set Enrichment Analysis: A Knowledge-Based Approach for Interpreting Genome-Wide Expression Profiles.” *Proceedings of the National Academy of Sciences* 102 (43): 15545–50. <https://doi.org/10.1073/pnas.0506580102>.

Tidwell, Jeniger. 2021. “Organizing the Page:layout of Page Elements.” <https://www.oreilly.com/library/view/designing-interfaces/0596008031/ch04.html>.

Trounce, David. 2019. “7 Great Tools for Creating Your Own Video Tutorials.” <https://helpdeskgeek.com/free-tools-review/7-great-tools-for-creating-your-own-video-tutorials/>.

Wilson, Lee. 2020. “25 of the Best Examples of Effective FAQ Pages.” <https://www.searchenginejournal.com/best-faq-page-examples/267709/>.

Wolf, Damian. 2016. “Why Developers Write Horrible Documentation and How to Solve It.” <https://dzone.com/articles/why-developers-write-horrible-documentation-and-ho>.

1. For all cartoons:  
   Avataars by <https://getavataaars.com/>.  
   Icons by <https://thenounproject.com/> License CC BY-NC-ND 2.0.  
   Emojis by OpenMoji License: CC BY-SA 4.0. [↑](#footnote-ref-35)
2. For all cartoons:  
   Avataars by <https://getavataaars.com/>.  
   Icons by <https://thenounproject.com/> License CC BY-NC-ND 2.0.  
   Emojis by OpenMoji License: CC BY-SA 4.0. [↑](#footnote-ref-54)
3. For all cartoons:  
   Avataars by <https://getavataaars.com/>.  
   Icons by <https://thenounproject.com/> License CC BY-NC-ND 2.0.  
   Emojis by OpenMoji License: CC BY-SA 4.0. [↑](#footnote-ref-77)
4. For all cartoons:  
   Avataars by <https://getavataaars.com/>.  
   Icons by <https://thenounproject.com/> License CC BY-NC-ND 2.0.  
   Emojis by OpenMoji License: CC BY-SA 4.0. [↑](#footnote-ref-108)
5. For all cartoons:  
   Avataars by <https://getavataaars.com/>.  
   Icons by <https://thenounproject.com/> License CC BY-NC-ND 2.0.  
   Emojis by OpenMoji License: CC BY-SA 4.0. [↑](#footnote-ref-131)
6. For all cartoons:  
   Avataars by <https://getavataaars.com/>.  
   Icons by <https://thenounproject.com/> License CC BY-NC-ND 2.0.  
   Emojis by OpenMoji License: CC BY-SA 4.0. [↑](#footnote-ref-157)
7. For all cartoons:  
   Avataars by <https://getavataaars.com/>.  
   Icons by <https://thenounproject.com/> License CC BY-NC-ND 2.0.  
   Emojis by OpenMoji License: CC BY-SA 4.0. [↑](#footnote-ref-194)