GRA Continuity Guide

Cannell Research Team

Table of contents

1 Welcome and Overview

Welcome to the team!

We are excited to have you join us and want you to be successful. This guide is intended to help Graduate Research Assistants (GRAs) acclimate to the project, find resources, and gain a clear understanding of expectations.

2 Expectations and Advice

2.1 General Expectations

We believe that setting clear expectations upfront is kind and will help you to have a successful GRA experience. In that spirit, here are some of the general expectations we have for our team members:

2.1.0.1 Work Your Committed Hours

Nobody likes to be micromanaged, and we don't like micromanaging. In general, we prefer to have a relaxed work environment where people are free to work whenever and wherever they work best. However, at the end of the day, we are all getting paid to do a job, and the job needs to get done. If you committed to working 19.5 hours per week, the expectation is that the project will get a full 19.5 hours of your best effort each week if there is work that needs to be done.

2.1.0.2 Be Available

Try your best to be available when the team needs you. For example, be responsive to emails and attend team meetings. If you are sick, need to adjust your hours, or have a conflict, let the program manager know and we can arrange accommodation. We realize that emergencies, class commitments, and other situations arise.

2.1.0.3 Code Collaboratively through GitHub

All collaborative coding tasks will be tracked through GitHub. If you aren't familiar with GitHub, you can start by checking out the chapters in R4Epi on using Git and GitHub. When you complete your coding tasks, push them to GitHub so that the PI can review and approve them.

2.1.0.4 Be Flexible

Be ready for many kinds of work projects. Some weeks may be all coding, other weeks may have responsibilities that include working in iRIS, writing, or researching. All of these tasks have value and help move the project forward to success.

2.2 Advice for GRA Success

The most important trait you can master as a GRA is developing a habit of taking initiative. Of course, you will need some time at the beginning of your GRA experience to figure out what is going with the project you've been hired to work on and how each team member's role/personality should be best managed. However, after a few weeks to a month, you should start to take initiative on your own.

What does this mean? To us, it means:

2.2.0.1 Get the Big Picture

Make sure you have a solid understanding of the project's 'big picture.' What is the project trying to accomplish overall? What are some big milestones that need to be accomplished by the end of the year and/or the end of the project? If you don't know or aren't sure, ask!

2.2.0.2 Fill in the Details

Once you understand the 'big picture' goals for the project, really work on developing your ability to think through the individual steps that need to occur for those big picture goals to be achieved. As an example, let's say that you know that one of the project goals is to create a participant/patient recruiting dashboard so that we can track our recruitment progress. What steps need to happen? Well, we need data from somewhere, we typically need to clean that data, we usually need to do some exploratory analysis of the data, and then we need to start drafting useful metrics that we can add to our dashboard.

2.2.0.3 Make it Happen

Once you've thought through the steps that need to occur, don't wait for someone to assign you a task. This is where you take the initiative to complete a task that you know needs to be completed without someone telling you. Having said that, there's nothing wrong with running it by other team members first to make sure you aren't about to start on something that will be a waste of your time.

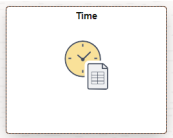
3 Onboarding

3.1 Getting Paid

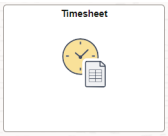
We all want to get paid! To make sure you do, please follow the steps outlined below:

1. If you haven't already, set up your UTHealth Virtual Private Network (VPN) following these instructions.
2. Log into the VPN using your UTHealth credentials.
3. Log into the Employee Self-Service site.

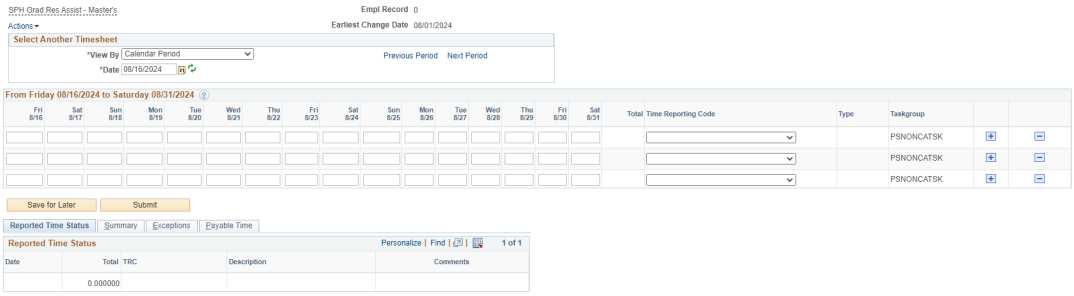
4. On the next screen, click on the `Time` button.



5. And then the `Timesheet` button.



6. Finally, fill in your electronic timesheet. After your hours are entered, the program manager will approve them in the system, and you will get paid.



To access W-2 forms, payroll calendars and more, click the `Payroll Resources` button.

If you have issues with any step in this process, you may reach out to the IT help desk at 713-486-4848.

3.2 IRB Requirements

IRB stands for Institutional Review Board, and operating the IRB is one of the most important functions of our school's Committee for the Protection of Human Subjects. As you might have guessed from the name, the IRB exists to review all research the school plans to conduct and make sure that it is in compliance with all federal, state, and ethical guidelines intended to protect human research participants.

One of the first things we will need to do when you join the team is add you to our IRB protocols. If you haven't already done so, we will need you to complete two steps before we can add you:

3.2.1 Complete CITI training

CITI is a program that provides human subjects protection training to many universities across the country. To complete your CITI training:

1. Follow the instructions on the UTHealth IRB website to log on to CITI and register.

2. After registering, complete the course called, "GCP – Social and Behavioral Research Best Practices for Clinical Research."
3. Download the Certificate of Completion to send to the study PI and program manager so that they may add you to the IRB.

3.2.2 Complete a Conflict of Interest (COI) Disclosure

Every UTHealth student, staff, and faculty member who participates in research needs to complete a COI annually. Please fill out your COI on the UTHealth COI website. You will also be able to look up frequently asked questions while you are there. Students very rarely have any financial conflicts of interest that they need to report, but feel free to reach out to Dr. Cannell if you have any concerns.

Part I

GRA Tasks

4 IRB Tasks

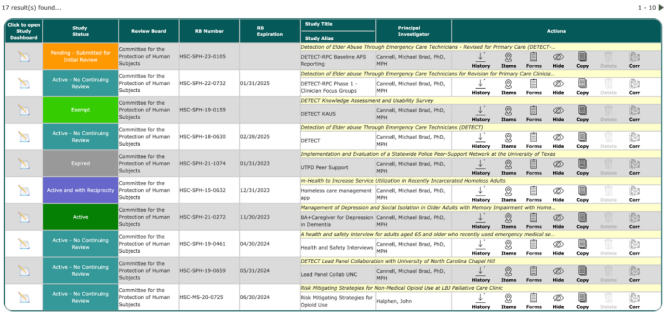
4.0.1 IRB Documents

Local copies of all IRB documents for the DETECT project are located in our SharePoint Document Library at Documents/General/IRB Private. You may need to read, copy, and/or update these files from time to time.

4.0.2 iRIS

Virtually all IRB documentation will be submitted and stored in UTHealth's Integrated Research Information Software (iRIS). If you have iRIS-related questions, you may contact the iRIS help line at 713-500-7960.

As part of your responsibilities as a GRA, you may or may not be asked to directly create or edit research protocols/documents in the iRIS system. After you sign in, you will land on the iRIS home screen. There can be multiple sections and studies listed on your iRIS home screen, but there should be a little pencil and paper icon on the left-hand side of every active study and study update that you have access to. Here is an example from Dr. Cannell's iRIS home screen with the pencil and paper icon highlighted in red.

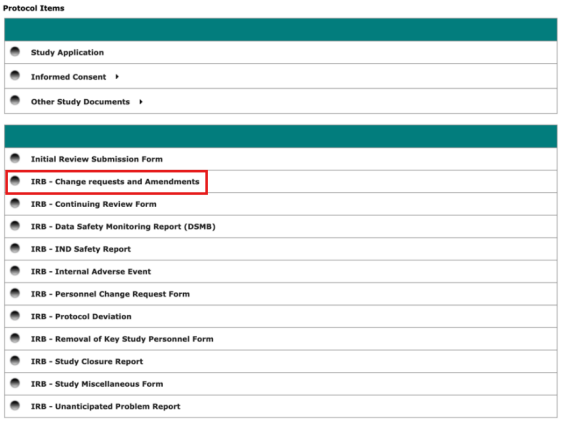


4.0.3 IRB Change Requests and Amendments

One task that we commonly ask GRA's to assist us with is submitting change requests and amendments. Change requests and amendments are typically used to let the IRB know that we intend to change something about the way we are conducting our study. For example, if

we originally told the IRB that we would pay research participants $20 and we later decide to pay them $25, then we will need to submit that change to the IRB for approval.

To access a change request form, click on the pencil and paper icon for the study you are working on. The next screen should look something like this.



One of the options will be `IRB - Change requests and Amendments` – click it. On the next page, click the `Add a New Form` button. Each protocol is somewhat unique, so it's difficult to provide instructions on precisely what to type into the form. However, here are some notes that previous GRAs found useful about the DETECT project specifically. Please keep in mind that we can change any part of the form before our final submission.

- Update number: If this is the first protocol update, then the update number will be 1. If it's the second, then 2, and so on.
- Short description. Just do your best to write something succinct and descriptive.
- Does the revision to this protocol require a formal change to the title: Typically, not.

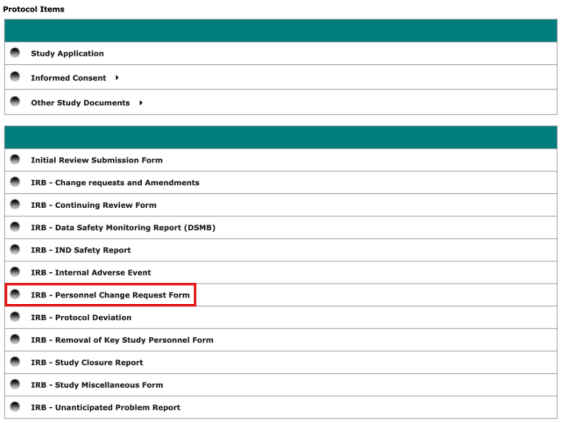
- Memorial Hermann location questions: Typically, the answer to all of these is "No."
- Type of revisions: Select those that seem applicable.
- Revision description and rationale: What changes do we want to make and why do we want to make them?
- Who is initiating this change: Typically, Investigator-initiated.
- Risk increase: Typically, the revision will not increase risk to participants.
- Designated Department Approval/Head: Bijal Bala.
- Funding: DETECT is federally funded by the National Institutes of Health (NIH) and the National Institute on Aging (NIA).

It can often be helpful to start by looking at other change requests that have been submitted and approved. At the end of the day, we ask that you just fill out the form(s) to the best of your ability. When you are done, please email the PI and program manager to let them know. They will review all of the forms and likely make edits before the final submission.

4.0.4 IRB Personnel Change Requests

Another common task we ask GRAs to help with is adding or removing personnel from the study.

To access a personnel change request form, click on the pencil and paper icon for the study you are working on. The next screen should look something like this.



One of the options will be `IRB - Personnel Change Request Form` – click it. On the next page, click the `Add a New Form` button. Here are some notes for filling out the form that previous GRAs found useful. Please keep in mind that we can change any part of the form before our final submission.

- Are you requesting a change to the principal investigator: No.
- Are you requesting the addition of a co-investigator: Maybe. Ask the PI or Program Manager if you are unsure.
- Are you requesting changes to key study personnel: This is the most common scenario.
- To add co-investigators or key personnel, we will need to have their CITI certificate and they will have to complete a COI. Both of these documents are explained above.
- Are you removing anyone from the study at this time: If we are, please just write their name and a brief explanation. For example, "Jane Smith. She graduated in May."
- Typically, we prefer not to add and remove personnel on the same form. So, if we need to add Jon Smith to the protocol and we need to remove Jane Smith from the protocol, we should do so using two separate forms.
- Significant financial interest: This is almost always "No." So, please select "No." The PI will change it to "Yes" before the final submission if necessary.
- Additional information: You can leave this blank.
- Designated Department Approval/Head: Bijal Bala.
- Funding: DETECT is federally funded by the National Institutes of Health (NIH) and the National Institute on Aging (NIA).

At the end of the day, we ask that you just fill out the form(s) to the best of your ability. When you are done, please email the PI and program manager to let them know. They will review all of the forms and likely make edits before the final submission.

5 Data Management and Analysis Tasks

Most, if not all, of our research projects will involve the management and analysis of data. This section of the SOP includes information intended to make it easier for us to find and use the files we need to access, manage, and analyze project data. These files broadly include project data files, programming code files, and general project documentation files. All three will be covered below. However, there are a couple of general topics that we should familiarize ourselves with first. Namely, protective health information (PHI), reproducible research, and GitHub.

5.0.1 Protected Health Information

We should already be familiar with PHI from our CITI training courses. However, this concept is so important that it bears repeating here. The UC Berkely Human Research Protection Program website summarizes PHI this way:

> *Protected Health Information (PHI) is any information in the medical record or designated record set that can be used to identify an individual and that was created, used, or disclosed in the course of providing a health care service such as diagnosis or treatment. HIPAA regulations allow researchers to access and use PHI when necessary to conduct research. However, HIPAA applies only to research that uses, creates, or discloses PHI that enters the medical record or is used for healthcare services, such as treatment, payment, or operations.*

It further goes on to list 18 identifiers that can be used to identify the individual associated with the health records. They are:

1. Names;
2. All geographical subdivisions smaller than a State, including street address, city, county, precinct, zip code, and their equivalent geocodes, except for the initial three digits of a zip code, if according to the current publicly available data from the Bureau of the Census: (1) The geographic unit formed by combining all zip codes with the same three initial digits contains more than 20,000 people; and (2) The initial three digits of a zip code for all such geographic units containing 20,000 or fewer people is changed to 000.

3. All elements of dates (except year) for dates directly related to an individual, including birth date, admission date, discharge date, date of death; and all ages over 89 and all elements of dates (including year) indicative of such age, except that such ages and elements may be aggregated into a single category of age 90 or older;
4. Phone numbers;
5. Fax numbers;
6. Electronic mail addresses;
7. Social Security numbers (SSN);
8. Medical record numbers (MRN);
9. Health plan beneficiary numbers;
10. Account numbers;
11. Certificate/license numbers;
12. Vehicle identifiers and serial numbers, including license plate numbers;
13. Device identifiers and serial numbers;
14. Web Universal Resource Locators (URLs);
15. Internet Protocol (IP) address numbers;
16. Biometric identifiers, including finger and voice prints;
17. Full face photographic images and any comparable images; and
18. Any other unique identifying number, characteristic, or code (note this does not mean the unique code assigned by the investigator to code the data)

5.0.2 Reproducible Research

We will strive to make our projects conform to best practices for promoting reproducible research. Many resources that describe what reproducible research is, and why it's important, are available on the internet, and we encourage you to look at some of them. Briefly, here is an excerpt from Wikipedia that is good enough for our purposes:

> *"The term reproducible research refers to the idea that scientific results should be documented in such a way that their deduction is fully transparent. This requires a detailed description of the methods used to obtain the data and making the full data set and the code to calculate the results easily accessible. This is the essential part of open science.*
>
> *To make any research project computationally reproducible, general practice involves all data and files being clearly separated, labelled, and documented. All operations should be fully documented and automated as much as practicable, avoiding manual intervention where feasible. The workflow should be designed as a sequence of smaller steps that are combined so that the intermediate outputs from one step directly feed as inputs into the next step. Version control should be used as it lets the history of the project be easily reviewed and allows for the documenting and tracking of changes in a transparent manner."*

Because the data we work with almost always includes (PHI), we will not typically be able to make our data freely available to the general public. However, we will do our best to conform to the remaining elements of reproducible research described above. Two of the most important tools we will use to make our research more reproducible are git and GitHub.

5.0.2.1 GitHub

GitHub is a website specifically designed to facilitate collaboratively creating programming code. In many ways, GitHub is a cloud-based file storage service like Dropbox, Google Drive, or OneDrive. However, GitHub contains some additional features that make it an exceptional tool for collaborating our research projects. Some of these features include:

- Repositories
- Projects
- Discussions
- Wikis

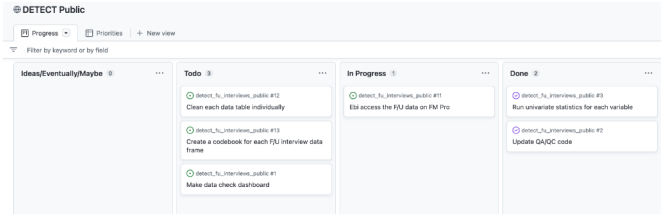
5.0.2.1.1 GitHub Repositories

We will make extensive use of GitHub repositories. If you aren't already familiar with git and GitHub, please start by reading the relevant chapters in R for Epidemiology. It's probably a good idea to read those chapters even if you are already familiar with git and GitHub because they describe how we will use our GitHub repositories. Here are some key summary points to keep in mind:

- Almost all programming code, code documentation, and even a large amount of general project documentation will flow through our projects' GitHub repositories.
- Almost all of our programming tasks will be tracked using GitHub projects.
- Please work with the project PI and/or program manager to make sure you are able to locate and access the GitHub repositories and GitHub project(s) associated with the research project(s) you are working on.

5.0.2.1.2 GitHub Projects

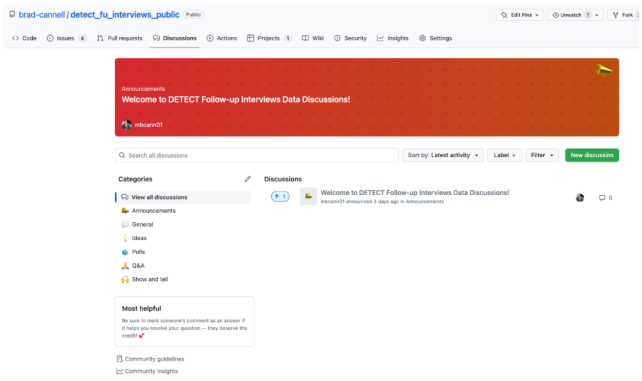
GitHub projects – we will sometimes refer to them as project boards – allow us to organize, track, and communicate about GitHub issues. The name "issues" sort of implies a problem, but that isn't how we will use them. We will use issues as tasks that need to be completed, and we will use project boards to track those tasks. In fact, for the rest of this continuity guide, I will refer to them as tasks instead of issues.



The example above is from the DETECT project and we can see examples of tasks that are yet to be started (i.e., Ideas/Eventually/Maybe and Todo), in progress, and already completed (i.e., Done). We can click on the tasks to open a detailed view where we can add a description, we can add sub tasks, and we can leave messages for each other about the task.

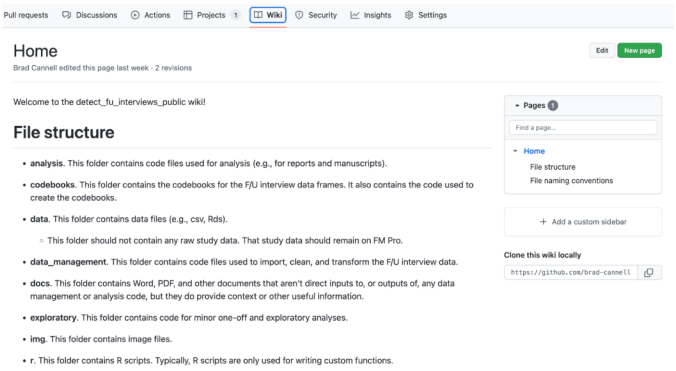
There are a couple of advantages to messaging each other about tasks in GitHub instead of sending emails. First, the messages stay with the tasks they are about. You don't have to go search for them later, and all the information/context you need is in one place. Second, when you graduate and leave UTHealth, your email account will be deleted. However, that doesn't necessarily mean you will want to quit working on a project. Fortunately, if we've been communicating in GitHub instead of through email, nothing important is lost.

5.0.2.1.3 GitHub Discussions



As the name indicates, GitHub discussions are a place where we can have discussions about the project. For example, we can exchange ideas and we can ask each other general questions that don't necessarily pertain to a specific task.

5.0.2.1.4 GitHub Wikis



We can generally think of GitHub wikis as an SOP for a specific repository. Wikis aren't about what to do – that's what tasks are for. Rather, wikis tend to contain information about how to navigate and use the repository and how to complete our tasks in a consistent way. For example, the screenshot above shows part of the DETECT follow-up interview repository's wiki. In this particular screenshot, we can see instructions for navigating and using the repository's file/folder structure.

5.0.2.2 Communicating About Research Projects

Over time, I have come to really appreciate the number and quality of tools that GitHub provides for collaboratively coding and working on research projects. It truly is an amazing tool in my opinion. Having said that, I understand how it can be pretty intimidating – or at least overwhelming – at first. So, here is a little cheat sheet to get us started.

Tool	Task	Example
Email	General communication that isn't about a specific project.	"What time are we meeting next week?"
Microsoft Planner	Assign and communicate about tasks, but not tasks that are specific to a single project.	Task: Update CITI training.
GitHub wikis	General information about a specific project, but not communication about a specific task.	"Data was collected between [date 1] and [date 2]."
GitHub discussions	Communication about a specific project, but not a specific task.	"Did we measure depression in this study?"
GitHub issues (tasks)	Communication about specific tasks.	[Assuming there is a task that directs us to perform a linear regression] "We decided to use X, Y, and Z as predictors in the model, right?"

Here are some additional notes about communication to keep in mind:

- Please follow the guidance in R4Epi when you have a coding question.
- When creating commits, please follow the guidance in R4Epi.
  - Scroll down to the paragraph that begins with, "The first line is called the commit message."

5.0.2.3 Separation of Data and Code

By design, public GitHub repositories are not secure – they are publicly available. So, why do we use them? As discussed above, we use them because we want our research to be as transparent and reproducible as possible. Having said that, we have an obligation to balance reproducibility with the protection of the participants who make our research possible. Our best effort at striking that balance will be to upload almost everything except data to our GitHub repositories and make it publicly available. Other people will still be able to use the data, but they will need to follow certain procedures.

We believe the reproducible research approach is the most ethical and productive way to conduct research; however, it does create some additional complications for us. Namely, we

can't store the data in our code repository, but we still want to use relative file paths in our code (Please read this chapter in R4Epi if you aren't familiar with the difference between relative and absolute file paths). In other words, the file paths we use in the code should work on team member's computer without having to make any alterations to it. There are at least a couple of different ways we can accomplish this.

1. Create a `data` folder in the repository, but make sure to gitignore all of the data files in the data folder. a. We will then exchange data to add to the folder using a process that is separate from cloning the repository. For example, we will exchange the data using a shared OneDrive folder, you will copy the data from the shared OneDrive folder into the `data` folder in your local repository.
2. Another option that is sometimes available is to store the data in a remote database (e.g., FileMaker Pro). Then, we connect to the data using ODBC. a. When using this method, we have to make sure we don't add our database username and password to the code that will be publicly available on GitHub. Additionally, we need to make sure that the code we use to access the remote database is identical on every team member's computer. A credential storage application like Keyring helps us meet both of these needs.

What if we do accidentally upload data, PHI, or passwords? Don't freak out! Just let the PI know and we will fix it together.

5.0.2.4 Writing Programming Code

When writing programming code we generally will follow the guidance given in the coding tools and best practices part of R4Epi. If you haven't already done so, please go ahead and read those chapters. In addition to reading about the coding style we will use, there are a couple of R packages that can help you style your code. They are lintr and styler. We highly recommend that you give them a try.

Here are some additional guidelines that may not jump out at you.

- Use Quarto documents.
  - Click here to read about Quarto files in R for Epidemiology.
  - We will almost always write our code in Quarto documents (as opposed to R scripts). However, we generally don't need to render them into HTML, Word, or PDF documents. The exception to this guideline includes:
    * We will typically write functions in R scripts when they will be used in more than one Quarto file.
    * We will write Shiny applications in R scripts.
    * If the project we are working on is an R package (as opposed to a research project), all functions will be written in R scripts.
- Don't add dates or your name to code files. That is what versioning is for.

  - **Do this:** `01_data_import`.
  - **Do not do this:** `01_data_import_mbc_edits`, `01_data_import_v2`, `01_data_import_2023_06_14`.

5.0.2.5 Getting Started Task List

Now that you have some basic information about our team process for working on data management and analysis, here is a quick task list to get you started. It should be applicable for just about any project you are working on.

- Make sure you've read the most applicable sections of R4Epi. Please pay special attention to:
  - The chapter on asking questions.
  - All of the chapters in the coding tools and best practices part of the book.
  - All of the chapters on using git and GitHub.
- Make sure you have already been added to the IRB protocol.
- Download all necessary software to your computer (e.g., RStudio, git, FM pro driver, GitKraken).
- Create all necessary subscriptions (e.g., GitHub).
- Fork the GitHub repository to your GitHub account.
- Clone the GitHub repository to your computer.
- Create a test pull request.
- Make sure you can access any data needed for the project.
- Start looking through the tasks on the GitHub project board.

Part II

Style Guide

6 Overview

The ultimate goal of a style guide is to reduce cognitive load. Doing so should improve the quality and consistency of our work product and make our work lives easier and more efficient. How does a style guide do this?

- First, it reduces the number of choices we need to make as we are authoring content. For example, we don't need to come up with our own answers to questions like, "where should I put this document?" or "what should I name this folder?"
- Second, having the predetermined choices written down for reference reduces the amount of information we need to store in our intentional memory. For example, "OK, remember to always start these file names with the date written as yyyy-mm-dd". Although we don't need to *intentionally* memorize them, these predetermined choices may eventually bleed over into our memory by accident through repetition.
- Third, having uniformly styled content makes it easier for others – *including future us* – to find what we are looking for and use it. We can focus our brain power on the content instead of the style and/or organization of the content.

7 Emphasizing Text

Use the following conventions to emphasize keywords, concepts, code snippets, and other words or phrases that need to stand out or be emphasized.

7.0.1 Application Names

Capitalize the names of applications.

- Do this: Microsoft Outlook, Outlook, Microsoft Planner, Planner.
- Do not do this: microsoft outlook, outlook, microsoft planner, planner.

7.0.2 Keywords

Underline a keyword or phrase if it is a keyword or phrase that we would want to define in the glossary of the document (whether the document actually has a glossary or not).

- **Do this:** A standard operating procedure (SOP) is…
- **Do not do this: A standard operating procedure (SOP)** is…, A standard operating procedure (SOP) is…

Bold a keyword or phrase that we want to call attention to, but it is not necessarily a keyword or phrase that we would want to define in the glossary of the document (whether the document actually has a glossary or not).

- **Do this:** **Step 1.**, ** Do this:**.
- **Do not do this:** Step 1., Do this.

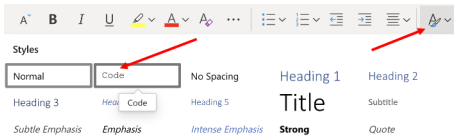
At times, we will also *Italicize* keywords or phrases that we want to call attention to, but are not necessarily keywords or phrases that we would want to define in the glossary of the document (whether the document actually has a glossary or not). In general, we will follow standard English grammar rules for using the italics typeface (pay special attention to bullet 14). Further, "avoid using italics with other stylized typefaces, such as bold and underline. Since all three are designed to make words stand out, only one at a time is necessary." (Ellis, 2022)

- **Do this:** We do not want to coerce participants into signing the consent form.

- **Do not do this:** We do NOT/**not**/*not*/***not***/***not*** want to coerce participants into signing the consent form.

7.0.3 Programming Code

Snippets of programming code should be written in the `Courier New` font with a light gray background. This matches the style convention used by many popular programming websites like GitHub and Stack Overflow. To make using this format easier, any document that was created by duplicating the New SOP Template will have a `code` option in the styles menu.



- **Do this:** `utcNow()`, `dplyr::select()`.
- **Do not do this:** utcNow(), *dplyr::select()*, **dplyr::select()**, "dplyr::select()"

7.0.4 Clickable Operations in Applications

In many of the applications we use, clickable steps or operations serve the same function as programming code. For example, in Microsoft word, we don't type "bold('Do this')". Instead, we highlight the phrase "Do this" with our mouse and then click the `B` button in the toolbar. Then, Microsoft Word takes care of the programming behind the scenes for us. Therefore, clickable operations that need to be performed in an application should be written in same style used to write code. To make using this format easier, any document that was created by duplicating the New SOP Template will have a `code` option in the styles menu.

- **Do this:** Click on the `Insert` tab...
- **Do not do this:** Click on the *Insert*/INSERT/**insert**/insert tab.

7.0.5 File Paths

All file paths should be written in italic text with a light gray background. To make using this format easier, any document that was created by duplicating the New SOP Template will have a `File Paths` option in the styles menu. Additionally, the root directory (i.e., starting point) for all SharePoint/Teams documents is the Documents folder.

- **Do this:** `Documents/General/SOPs/02 Style Guide.docx`.
- **Do not do this:** "Documents/General/SOPs/02 Style Guide.docx".

7.0.6 File and Folder Names

File and Folder names are essentially very short file paths. Therefore, file names and folder names should be written using the file paths guidelines from above. There is one exception to this rule. When creating a hyperlink to a file or folder, then the standard hyperlink style (i.e., link) should be used.

- **Do this:** The `SOP` folder.
- **Do not do this:** The "SOP" folder. The **SOP** folder.

8 Document Library Structure

8.0.1 Folder Structure

- Each folder in the document library should generally be associated with a specific theme or topic, which may have sub-themes and subtopics.
  - **Do this:** `IRB Documents` or `Budget Documents`.
  - **Do not do this:** `Budgets and IRB Documents`.
- Folders should not generally be associated with specific people or positions. There are at least two reasons why we don't want to associate folders with people or positions. First, people and positions change over time. Topics can change too, but they tend to be more stable. Second, and perhaps more importantly, this is a shared document library. We shouldn't think of anything we keep in this library as belonging to any individual. As such, the names and purposes of documents should be clear to others who may need to use them. For example, it's going to be much easier for most people to reason about what is contained in the `IRB` folder than in the `Brad` folder.
  - **Do this:** `IRB Documents` or `Budget Documents`.
  - **Do not do this:** `Brad's Documents` or `GRA Documents`.
- Folders that can be logically nested inside an existing folder, or two similar folders that can logically be combined under a single parent folder, should be.
  - For example, a folder containing CITI training certificates should probably be nested in the `IRB Documents folder`.
  - A folder containing GRA applicant resumes, a folder containing GRA work schedules, and a folder containing job announcements should probably all be nested in a `Hiring` folder, which should probably be nested in a `HR` folder.

8.0.2 Folder Names

- Folders for programming code repositories should be written in snake case. This is to help ensure that the folder name works easily with R, Git, Bash, and other software used in the data analysis pipeline.
  - **Do this:** `detect_public_repo`.

  - **Do not do this:** `DETECT Public Repo`.

- All other folders should be written in title case.
  - **Do this:** `Budget Documents`.
  - **Do not do this:** `budget documents`.

- Folder names should be descriptive enough for most people to reasonably be able to figure out what the folder contains from the name.
  - **Do this:** `Budget Documents`.
  - **Do not do this:** `Brad's Stuff` or `MU-334-011`.

- Folder names should be succinct.
  - **Do this:** `Pre-award`.
  - **Do not do this:** `Narrative - Spec Aims - Budgets - and Other Documents Submitted to NIA in the Original Proposal`.

- Folder names should only include letters, numbers, underscores, or dashes. Other characters can cause syncing failures and/or issues with other file systems.
  - **Do this:** `Materials and Operations`.
  - **Do not do this:** `Materials & Operations`.

- Folders that can be naturally arranged by dates should be. The folder name should begin with the date written in the **yyyy-mm-dd** format. This format is important because dates written in this format will naturally be arranged in correct chronological order across years. Some examples of folder topics that may have a natural chronological order include IRB documents, budgets, and meeting minutes.
  - **Do this:** `2023-02-17 Budget`.
  - **Do not do this:** `February Budget or 02-17-2023 Budget`.

- Folders that can be naturally arranged in a sequential order should be. When doing so, the folder names should be sequentially numbered. Single digits should be prefixed with a zero (0). This format is important because single digits written in this format will naturally be arranged in correct sequential order when the number of folders in the sequence exceeds 9. Data analysis projects are an example where folder topics may have a natural sequential order.
  - **Do this:** `01_data_import, 02_data_clean, 03_table_01, 04_table_04`.
  - **Do not do this:** `1_data_import, 2_data_clean, table_1, my_other_table`.

- Folders for documents related to peer-reviewed manuscripts should begin with the first author's last name.
  - **Do this:** `Cannell - Protocol Paper`.
  - **Do not do this:** `Protocol Paper`.

- When multiple folders have a similar theme/topic, begin the folder names with a description of the theme/topic. Doing so will ensure that all folders with that theme/topic are grouped together in the file list.
  - **Do this:** `Reliance Agreement UAB`, `Reliance Agreement UCSF`, `Reliance Agreement JHU`.
  - **Do not do this:** `UAB Reliance Agreement`, `UCSF Reliance Agreement`, `JHU Reliance Agreement`.

8.0.3 File Structure

- Files should generally be grouped together and placed in a folder that describes the theme or topic they belong to.
- Folders that contain a single file should be rare.

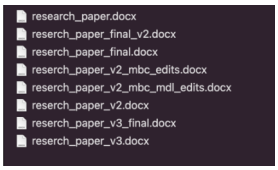
8.0.4 File Names

- Programming code files should be written in snake case. This is to help ensure that the file name works easily with R, Git, Bash, and other software used in the data analysis pipeline.
  - **Do this:** `01_data_import.Rmd`.
  - **Do not do this:** `Data Import.Rmd`.
- All other file names should be written in title case.
  - **Do this:** `Cannell CITI Certificate.pdf`.
  - **Do not do this:** `cannell citi certificate.pdf`.
- File names should be descriptive enough for most people to reasonably be able to figure out what the file contains from the name.
  - **Do this:** `Cannell CITI Certificate.pdf`.
  - **Do not do this:** `CITI.pdf`.
- File names should be succinct.
  - **Do this:** `Approved Protocol.docx`.
  - **Do not do this:** `DETECT-RPC - Phase 1 - Focus Groups - Research Protocol Final Accepted Copy.docx`.
- File names should only include letters, numbers, underscores, or dashes. Other characters can cause syncing failures and/or issues with other file systems.
  - **Do this:** `Materials and Operations.xlsx`.
  - **Do not do this:** `Materials & Operations.xlsx`.

- Files that can be naturally arranged by dates should be. The file name should begin with the date written in the **yyyy-mm-dd** format. This format is important because dates written in this format will naturally be arranged in correct chronological order across years. Some examples of files that may have a natural chronological order include IRB documents, budgets, and meeting minutes.
  - **Do this:** `2023-02-17 Budget.xlsx`.
  - **Do not do this:** `February Budget.xlsx or 02-17-2023 Budget.xlsx`.
- Files that can be naturally arranged in a sequential order should be. When doing so, the files names should be sequentially numbered. Single digits should be prefixed with a zero (0). This format is important because single digits written in this format will naturally be arranged in correct sequential order when the number of files in the sequence exceeds 9. Data analysis projects are an example where files may have a natural sequential order.
  - **Do this:** `01_data_import.Rmd, 02_data_clean.Rmd, 03_table_01.Rmd, 04_table_04.Rmd`.
  - **Do not do this:** `1_data_import.Rmd, 2_data_clean.Rmd, table_1.Rmd, my_other_table.Rmd`.
- Begin file names with a description of a theme or topic when there are multiple files that will have a similar theme or topic. Doing so will ensure that all of the files with that theme are topic are grouped together in the file list.
  - **Do this:** `Reliance Agreement UAB.pdf`, `Reliance Agreement UCSF.pdf`, `Reliance Agreement JHU.pdf`.
  - **Do not do this:** `UAB Reliance Agreement.pdf`, `UCSF Reliance Agreement.pdf`, `JHU Reliance Agreement.pdf`.

9 Collaborating on Files

Have you ever worked on a paper or report and had a folder on your computer that looked something like this?



Saving a bunch of different versions of a file like this is a real mess, and it tends to get worse the more collaborators we have. What is contained in each document again? What order were the documents created in? What are the differences between the documents? Does the version of the paper that Doug emailed this morning contain the edits that Jason made yesterday? Versioning helps us get around these problems, and luckily, we have some good tools for versioning. Therefore, we should generally not be emailing files to each other. Instead, we should strive to work on Microsoft Office Documents online and use Git and GitHub for collaborative computer programming. There are links to instructions for both below.

Please see the Using Word SOP for instructions on how to use Word's versioning system (Excel and PowerPoint have similar systems).

Please see for instructions for versioning other file types with Git and GitHub.

10 Appendix

10.1 Appendix A. Style Cheat Sheet

Theme: Default

The Title Written in Title Case

The Subtitle Written in Title Case

The normal style.

The default paragraph font style.

Level 1 header

Level 2 header

Level 3 header

Level 4 header

The subtle emphasis style.

The emphasis style.

The intense emphasis style.

The strong style.

The quote style.

The intense quote style.

THE SUBTLE REFERENCE STYLE.

THE INTENSE REFERENCE STYLE.

The book title style.

The list paragraph style.

The toc 1 style.

The toc 2 style.

The hyperlink style.

10.2 Appendix B. UTHealth Brand Standards

10.2.1 Logos

Here is a link to the online logo library.

10.2.2 Colors

Here is a link the color standards page.

Color Group	Name
Traditional	University Orange
Traditional	University Blue
Traditional	University Gray
Secondary	Gulf Blue
Secondary	Mustard Gold
Secondary	Dark Sage
Secondary	Dusty Lavender
Neutral	Light Sage
Neutral	Sand

10.2.3 Typefaces

Here is a link to the typefaces standards page.

We don't have easy access to many of these fonts. Here are some that we can use:

- Garamond
- Univers
- Times New Roman
- Avenir
- Helvetica
- Arial
- Calibri

11 Getting Started with SOPs

11.0.1 Overview

What are SOPs and why do we need them? The following quote provides a pretty useful answer.

> *A standard operating procedure (SOP) is a set of written instructions that describes the step-by-step process that must be taken to properly perform a routine activity. SOPs should be followed the exact same way every time to guarantee that the organization remains consistent and in compliance with industry regulations and business standards.*
>
> *Standard operating procedures provide the policies, processes, and standards needed for the organization to succeed. They can benefit a business by reducing errors, increasing efficiencies and profitability, creating a safe work environment, and producing guidelines for how to resolve issues and overcome obstacles.*

— Bush, 2021

Our team is large, diverse, and widely geographically distributed. These traits are extremely important strengths for our team, but they also make organizing and managing our activities more challenging than they would be if our team were small, homogenous, and we were all located down the hall from each other. As the quote highlights, the adoption and use of SOPs are one way we can produce consistent, high-quality work as often as possible. Another advantage of SOPs is that they should reduce our cognitive load. How do SOPs reduce our cognitive load?

- First, they reduce the number of choices we need to make. For example, we don't need to come up with our own answers to questions like, "where should I put this document?" or "what should I name this folder?"

- Second, having the predetermined choices written down for reference reduces the amount of information we need to store in our intentional memory. For example, "OK, remember to always start these file names with the date written as yyyy-mm-dd". Although we don't need to intentionally memorize them, these predetermined choices may eventually bleed over into our memory by accident through repetition.

- Third, having uniformly styled content makes it easier for others – including future us – to find what we are looking for and use it. We can focus our brain power on the content instead of the style and/or organization of the content.

11.0.2 Feedback

At the end of the day, the point of these SOPs is to make our lives easier and improve the quality and efficiency of our work. They can't possibly achieve any of those objectives that if they aren't used. Therefore, we want to encourage every member of the team to provide us with feedback. What is working? What isn't working? What's confusing? Possibly even some positive feedback about tools and processes that *are working* – We won't hold our breath on that one . The easiest way to send feedback is probably to just call 972-546-2925 or email Brad Cannell, who is currently responsible for maintaining the SharePoint site.

11.0.3 Next Steps

At this point, you should know what SOPs are and why we are using them. Now what? Well, depending on your role on the team, you may rarely (if ever) use some of the SOPs. However, any time you engage in an activity that impacts our team or interact with a file that is shared on the SharePoint site, there is a good chance that there is an SOP – or soon will be an SOP – that provides guidelines for that activity or interaction.

Even if you are a team member that rarely engages in an activity that is covered in an SOP, quickly skimming through some of them may provide you with useful information about the organization of the project that makes your life easier.
Finally, these SOPs are intended to be living documents that evolve to meet our changing needs over time. Therefore, it probably isn't a good idea to assume that you never need to check an SOP that you've read in the past.

Thank you for your time and consideration!

Part III

Authoring SOPs

12 Overview

This part contains information about authoring SOPs. Any information we want to record about creating, editing, or deleting SOPs should be included in this part. Any information we want to record that is not exclusively about creating, editing, or deleting SOPs should not be included in this part. For example, even though SOPs use Microsoft Word, an SOP about using Microsoft Word should be its own separate part of the guide.

> **Note**
>
> We hope that all of our team members will use and provide feedback on our SOPs; however, most of our team members will never need to author SOPs. Most of you can safely ignore this part of the guide.

Specifically, this document covers:

- When and how should we author SOPs?
  - Please use the New SOP Template to create a new SOP document.
  - Please see the style guide part for general Word document formatting guidance.
- What technologies (apps) should we use to write our SOPs?
  - Quick Summary: SOPs should be written using Microsoft Word documents.
- Where should our SOPs be stored?
  - Quick Summary: SOPs should generally be stored in `Documents/SOPs`.

Structure

Each document should be about a single, definable topic, concept, or technology. For example, the document you are reading right now is about creating and editing other SOPs. Over time, we may find that what we originally thought of as a single, definable topic, concept, or technology is really composed of multiple different topics, concepts, or technologies that we need to break out into separate SOP documents. That's OK! The goal here is to make it easy to use the SOPs, and by extension, to do good work.

The rest of the structure section of this SOP gives step-by-step instructions for creating and authoring SOPs. It assumes that we are starting with the New SOP Template document located in Documents/SOPs. Note that this template has the margins set to narrow. We did this because, although these are Word documents, we anticipate that they will most often be viewed online. Using narrow margins gives them more of a webpage look and feel.

Step 1: Start by adding a rectangular-shaped picture to the top of the SOP document and center it. It makes the documents more appealing to look at and feel more like reading a webpage. - To do add a picture, click `Insert` > `Picture`. There are multiple options including stock images that come with Microsoft Word. Other useful websites for finding free pictures include Google Images and Unsplash.

Step 2: Immediately under the picture should be the document title. It should succinctly capture the theme of the document. Please apply the `Title` style to the title.

Step 3: Below the title, each page should have a table of contents (TOC). The TOC should be automatically generated and link to the document headers.

Step 4: Each page should also have an "Overview" section under the TOC. The Overview section should contain a brief description of the SOP's topic. If we aren't able to describe the topic in a sentence or two, that may be an indication that the page doesn't have a single, high-level topic.

Step 5: Use the header styles (i.e., Heading 1, Heading 2, etc.) to break the document up into sections and subsections. Make sure to use heading styles as opposed to just changing the color and font manually. This is important for at least two reasons.

1. Using a heading style will create a clickable link in the TOC (new headings require updating the TOC).

2. If we decide to make changes to a heading style, all of the headings will automatically be updated to match.

Paragraphs

New paragraphs, steps, and glossary items should not be indented. They should be aligned with the left margin of the document.

Headers

Each level one headings (i.e., `Heading 1`) should have a single, high-level theme, but not high-level enough to (currently) warrant its own SOP. Over time, we may find that what we originally thought of as a single, high-level theme was really more appropriate to break into multiple themes. That's ok.

Level one headings should be written in title case. All other headings should be written in sentence case.

Tone

The goal is for these SOPs to be user-friendly. As such:

- Use informal, first-person language.
- We will typically use the word “we” instead of the word “I”. This project is a team effort.

Technologies

What technology (application), or technologies, should we use for writing our SOPs? We've experimented, and will continue to experiment, with many different applications, processes, and technologies for organizing and managing all aspects of our projects – including our SOPs. Here is what we've found so far.

Using OneNote, Word Documents, and Teams/SharePoint Pages

For now, we know that we want to try to manage as much of our project as possible inside the Microsoft ecosystem. Because Microsoft products are universally the best? Not necessarily. In fact, some members of our team really dislike Microsoft products. Why, then? We want to manage as much of our project as possible inside the Microsoft Ecosystem because (1) UTHealth pays for it; (2) UTHealth IT security approves of it; (3) The majority of our team members – internal and external to UTHealth – are already familiar with, and have access to, Microsoft products.

However, even within the Microsoft ecosystem, there are often multiple different applications and technologies we can choose to author and store SOPs. So, which technology should we use? In the table and paragraphs below, we list the pros and cons of using different Microsoft technologies for creating, editing, and accessing SOPs.

Trait	OneNote	Word
Formatting		
Overall	Some	⭐ Yes
Custom styles	No	⭐ Yes
Mobile		
Access SOP on mobile device	Yes	Yes
Edit SOP on mobile device	Yes	Yes
Offline use		
Access SOP offline	⭐ Yes	Yes, if synced with OneDrive
Edit SOP offline	⭐ Yes	Yes, if synced with OneDrive
Portability		
Document portability	Somewhat portable	⭐ Highly portable
Permissions		
Control read/write access	Share or not share only	Yes, through SharePoint
Searchability		
Search for document	⭐ Yes	Yes, in OneDrive and SharePoint
Search for text in a document	⭐ Yes	Yes
Versioning		
Have versioning systems	Yes	⭐ Yes

⭐ = Best

Figure 12.1: Summary of the pros and cons of using different Microsoft technologies for creating, editing, and accessing SOPs. A detailed discussion of each trait is in the sections below.

Formatting

When it comes to formatting the SOPS, Word has the edge over OneNote and SharePoint pages. Word gives us finer control over formatting and more robust tools. Especially for things like tables.

Custom Styles

Using styles makes it so much easier to create consistently formatted documents. OneNote, Word, and SharePoint all have some basic built-in styles, but as of this writing, only Word allows us to create custom styles (e.g., the `Code` style and `File Paths` style available in this document). Here is a useful post about sharing styles between Word documents.

Mobile Device Access

Mobile device access refers to the ease with which the SOP can be accessed and edited from a mobile device (i.e., phone or tablet).

All three technologies have a mobile app; however, the OneNote mobile app probably provides the best overall mobile experience. It allows us to search, read, and edit notes with the fewest number of clicks/taps. Having said that, the difference between the three isn't large when the app has an active internet connection.

Offline Access

Offline access refers to the ease with which the SOP can be accessed and edited in the absence of an active internet connection (e.g., when traveling).

12.0.0.0.1 * From a desktop/laptop computer without an active internet connection

OneNote provides the best experience out of the box. We can read and edit notes in the OneNote app, which will be synced with the server when an internet connection is reestablished. The same is true for Word documents, but only if they are being synced with the computer's file system. We highly recommend doing so. In general, SharePoint will not work without an internet connection.

12.0.0.0.2 * From a mobile device without an active internet connection

OneNote provides the best experience out of the box here too. We can read and edit notes in the OneNote app, which will be synced with the server when an internet connection is reestablished. The same is true for Word documents, but only if they are being synced with the computer's file system. We highly recommend doing so. In general, the SharePoint mobile app will not work without an internet connection.

Portability

In this context, "portability" refers to the ease with which the SOP can be shared, viewed, copied, and used outside of its current context. For example, if UTHealth decides to stop paying for Microsoft 365 tomorrow, could we access and use the SOPs?

Word documents are highly portable. OneNote is slightly less so. Why does this matter? We never know what the technology situation will be like tomorrow. There are a number of reasons why we could potentially need to change the technologies and processes we are using. Therefore, the ease with which we could migrate our content to a different technology or process if we needed to is probably worth taking into consideration. Additionally, we often need to collaborate with team members outside of UTHealth. The ease with which team members who do not have internal access to UTHealth's systems is another important dimension or portability.

12.0.0.0.1 * Exporting OneNote notes

- Exporting and importing notebooks through OneNote for the web is only available for notebooks stored on personal OneDrive accounts, not for notebooks stored on OneDrive for Business or SharePoint. For information about exporting notebooks to PDF files from OneNote 2016 for Windows, see Export notes from OneNote as a PDF.
- This is kind of a big deal. If we ever need to adopt a new technology, migrating all of our notes out of OneNote could be very difficult.

12.0.0.0.2 * Exporting SharePoint pages

- Exporting SharePoint pages is also relatively difficult process and requires administrator access.

Permissions

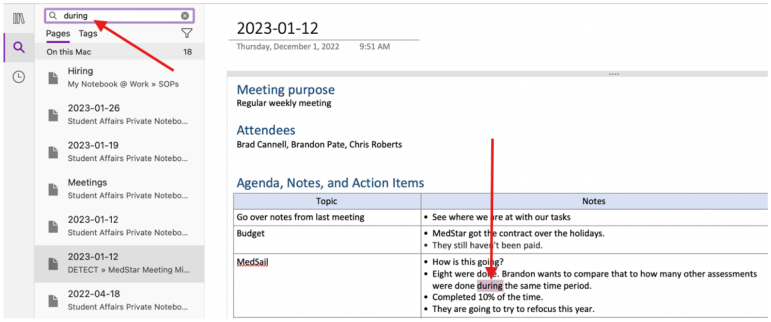
While we definitely value every team member's input into our SOPs, restricting editing permissions to one or two people can be a useful tool for maintaining the integrity of our SOPs. SharePoint has the most robust tools for assigning user permissions. However, these user permissions can be applied to Word documents when they are stored in a SharePoint library.

Searching Documents

Given the number of documents our research projects contain, the ability to search for documents and search for text within documents is an important consideration. We experimented with searching for documents, and text within documents, in OneNote, Teams, and SharePoint. Here is what we found.

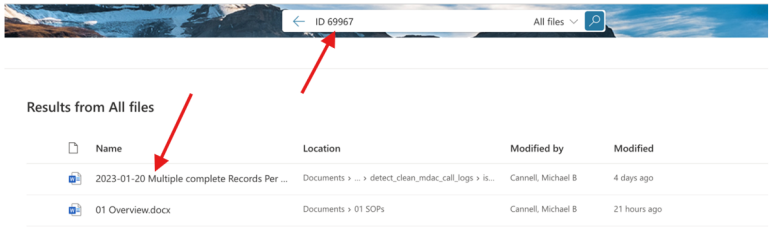
12.0.0.0.1 * Searching in OneNote

OneNote has very robust search capabilities. The screenshot below is from the OneNote desktop app. We arbitrarily searched for the word "during." OneNote returns a list of linked search results, across notebooks, with the search team highlighted in the text.



12.0.0.0.2 * Searching on OneDrive for documents

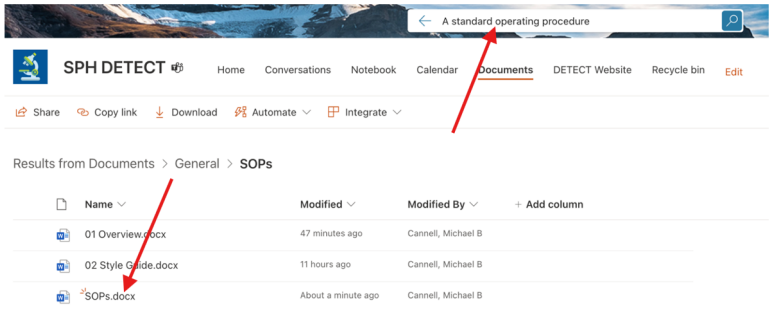
For this test, there was a document in OneDrive titled 2023-01-20 Multiple complete Records Per MedStar ID.docx. The document really isn't that important for this example – we just picked something. It contains the following text, "ID 69967". When we navigate to OneDrive and search for that text string in the search bar, the correct document comes up in the search results. However, the exact spot where the text appears is not shown in the search results.



Having said that, it isn't too big of a deal to just open the document and do a search for the text string in the document.

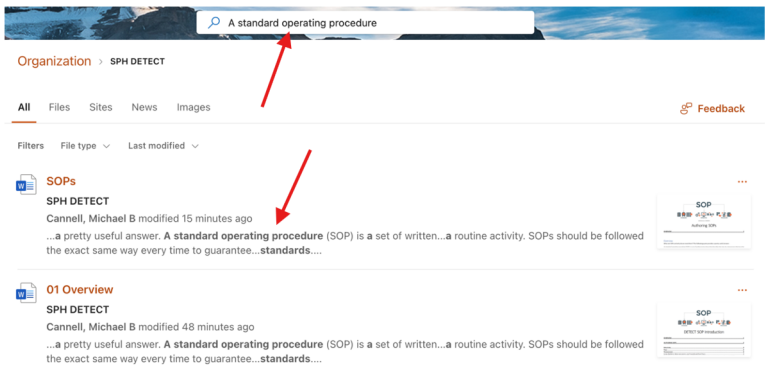
12.0.0.0.3 * Searching on SharePoint for documents

For this test, there was a document in the DETECT SharePoint document library titled, SOPs.docx. Again, there is nothing special about this document. We're just using it as an example. It contains the following text, "A standard operation procedure". When we navigate to the DETECT SharePoint document library and search for that text string in the search bar, the correct document comes up in the search results. However, the exact spot where the text appears is not shown in the search results at first.



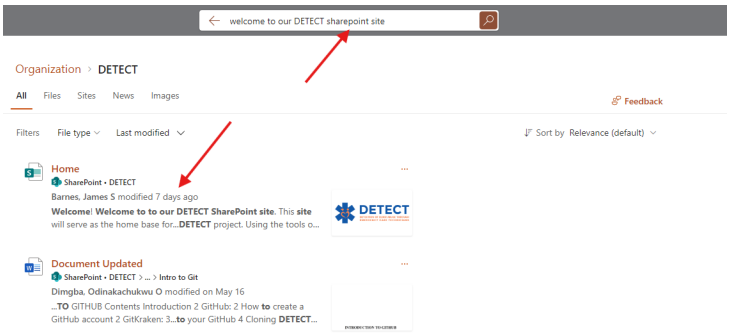
Having said that, it isn't too big of a deal to just open the document and do a search for the text string in the document. Additionally, if we click "Expand search to all items in this site" at the bottom of the search results screen:

Then SharePoint will return a more detailed list of linked search results that include a preview of the text string located in the document.Graphical user interface, text, application, email



12.0.0.0.4 * Searching SharePoint pages

For this test, we added a text web part to the DETECT SharePoint homepage that said, "Welcome to our DETECT SharePoint site!" When we navigate to the DETECT SharePoint homepage and search for that text string in the search bar, the correct document comes up in the search results and it includes a preview of the text string located in the document.

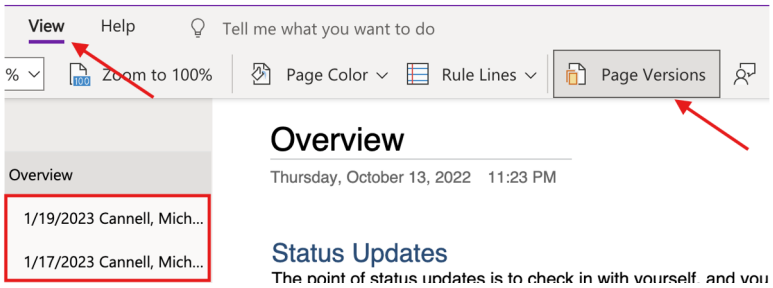


13 Versioning

Versioning is a great tool that OneNote, Word, and SharePoint pages all have. However, Word's implementation of versioning is probably the easiest to work with.

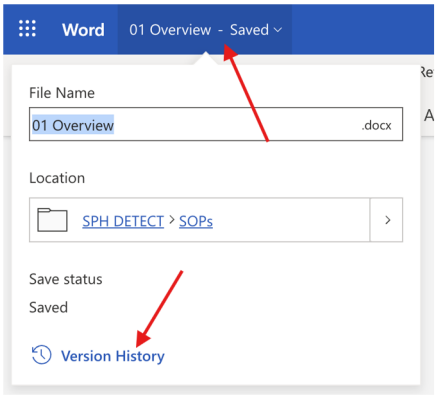
13.0.1 OneNote Online Versioning

To see previous versions of OneNote notes, click on the `View` tab in the ribbon and then click the `Page Versions` button.



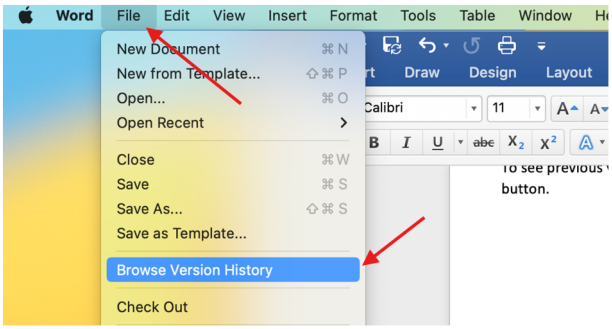
13.0.2 Word Online Versioning

To use versioning in Word online, click on the document title above the ribbon. Then, click `Version History`.



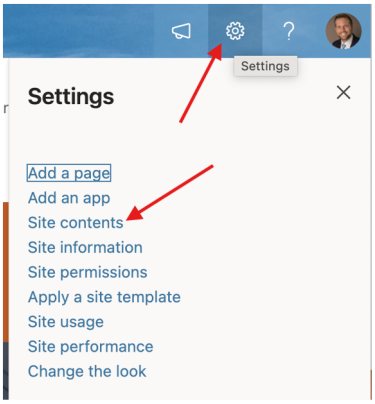
13.0.3 Word for Mac Versioning

To use versioning in Word for Mac, click `File` > `Browse Version History`.

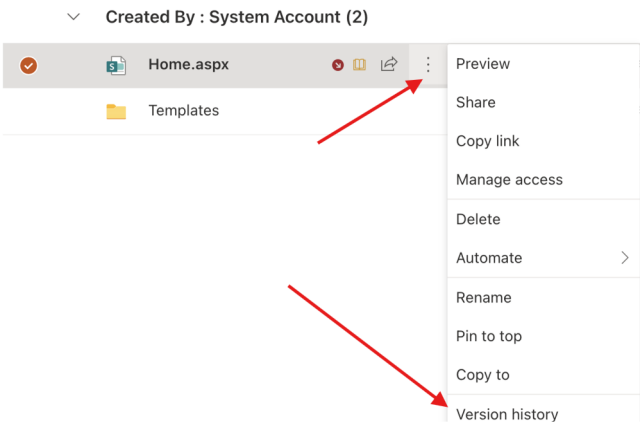


13.0.4 SharePoint Versioning

To see previous versions of SharePoint pages, click `Settings` > `Site contents` > `Site Pages`.



Find the page you want to see versions of. Then, click `Show more actions for this item` (three vertically stacked dots). Finally, click `Version history`.


Created By : System Account (2)
Home.aspx
Templates
Preview
Share
Copy link
Manage access
Delete
Automate
Rename
Pin to top
Copy to
Version history

Part IV

Coding Peer Review SOP

14 Overview

This SOP is intended to provide guidance on how to engage in the Internal Peer Review Process for code-based peer reviews. This includes a guide on how to utilize the Internal Peer Code Review Form. An example scenario is used to demonstrate the process.

14.0.1 Target Audience

14.0.1.1 GRAs

The primary users of the Peer Code Review Forms are Graduate Research Assistants (GRAs) that engage in code-based work on the project(s). These GRAs may request a peer review from each other or be prompted to request such a peer review by project leadership. As such, all GRAs that engage in code-based work should be familiar with the Internal Peer Code Review Form and process.

14.0.1.2 PM/PI or Other Project Leadership

The Principal Investigator (PI) has ultimate responsibility for all products of the study. The Project Manager (PM) or other members of leadership may also oversee administrative tasks, including directing GRAs to complete an internal peer-to-peer review of materials. As such, project leadership such as the PI or PM may review any and all internal peer review documents as part of their own review(s). **As such, all internal peer review documents should be written with the expectation that they may be viewed by any and all member(s) of the project leadership at any time**.

14.0.2 Purpose

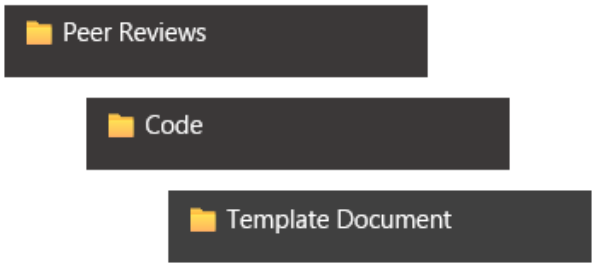
This SOP is intended to provide guidance on how to utilize the Internal Peer Code Review Form. The internal peer code review process, including the form, are intended to achieve the following goals:

1. Provide a mechanism for GRAs to perform an initial peer-to-peer review for major or common errors/issues in code, allowing project leadership to focus on high-level feedback and final oversight.

2. Provide a learning opportunity for GRAs, many of whom are learning how to write code and document code for use in collaborative and/or publishable research projects, by providing structured prompts for common reviewer questions and/or documentation gaps. This includes the need to define task parameters, goals, and process steps, as well as the need to clearly document feedback and provide rationales to support requested changes.
3. Minimize the time required in the process (both for the initial request documentation and the review) through a clear, uniform structure with logical organization. **The primary process should be able to be fully asynchronous, requiring initial code review to be completed without a dedicated meeting**.
4. Facilitate an audit trail for changes made to code that may be specifically due to the peer review process. This is intended to facilitate continuity, transparency, and reproducibility.

14.0.3 File Storage

All internal peer reviews should be kept within a `Peer Reviews` folder on SharePoint where they are visible and accessible to all project staff that may be involved in the internal peer review process. Code-based peer reviews should be stored in a designated "Code" folder within this parent directory, which also contains the template form document. This structure may be conceptualized as follows:



These folders should be specific to the project. As such, the exact location may vary by project.

For files relating to the DETECT projects (including DETECT-RPC), they are stored on the DETECT SharePoint in the top-level folder "Peer Reviews". As such, the path to the Internal Peer Code Review template document is:

`DETECT SharePoint/Documents/Peer Reviews/Code/00_template_p2p_code_01_01_repo_goal_ii_ii.doc`

14.1 Known Limitations and Potential Alternatives

As of August 5, 2024, Microsoft Word 365 has limitations to its functionality in the web browser version of the software. These limitations include a lack of advanced features, such as content controls, which are used in the creation of fillable forms. This limitation, and a guide to utilizing these advanced features in the desktop application version of the software, is documented by Microsoft on their official website. Requests for this additional functionality are documented on the Microsoft website since (at least) 2013, so it is reasonable to conclude that Microsoft does not plan to add this functionality in the near future.

> **!** Important
>
> At this time, it is required that the Internal Peer Code Review Form be opened in the Desktop App Version of Microsoft Word 365 (not the web browser version) for full functionality.

In the creation of the Internal Peer Code Review Form, multiple alternatives were considered. The current system and alternatives are briefly documented below to facilitate future changes to the process, if required, while minimizing duplication of effort.

14.1.0.1 Word Document Forms

Official Documentation This system is the system that was adopted. Advanced features in the Developer ribbon, such as content controls, would allow for restrictions on editing and auto-completion of a limited number of fields. The forms utilize existing SharePoint infrastructure and applications that are provided by UTHealth, which include version tracking and other administrative tools that would be useful in any audit/review. The system would allow flexibility in design, have a familiar interface for users, and keep requests and reviews within a single cohesive document.

14.1.0.2 GitHub Forms

Official Documentation This system would require all requests to be made in the form of GitHub issues, which are public facing. The form schema would only provide format to the initial request, not feedback or further revisions. While GitHub issues facilitate assignment modifications and documentation in a single thread, the inability to standardize both request and response resulted in us deciding against this system.

14.1.0.3 Microsoft Forms

Official Documentation This system would permit a standardized request form, and potentially permit a standardized response. However, this would require both a "request" and "review" form be created, with no built-in linkage between the two. This system was decided against due to limitations in keeping the data clearly understood for potential auditing/review, making clear assignment, and clear tracking of milestones.

14.1.0.4 Microsoft PowerApps, Including PowerBI

Official Documentation These systems had potential to permit the creation of guided apps that standardized and linked both requests and reviews with central storage of all entries. However, there was a significant learning curve to properly utilize or modify any process that utilizes these systems. Due to this complexity, we decided against this system.

14.2 SOP Authorship

This SOP document was originally written by Morrigan Mahady, GRA and Ebiekimie Dambo, GRA on August 5, 2024. Any modifications or revisions must be approved by the project PI, Dr. M.B. Cannell.

The example Internal Peer Code Review Form used in the creation of the Step-by-Step Example section was part of the DETECT project; the file is located on the DETECT SharePoint. The process was developed as a joint effort between Morrigan Mahady and Ebiekimie Dambo, GRAs.

15 Key Terms and Definitions

Requester: The individual(s) requesting peer review. This should be the individual that wrote the original code.

The requester is responsible for:

1. Selecting a reviewer
2. Notifying the selected reviewer of the request to engage in the peer review process
3. Completing all fields in the Requester Provided Information Section of the Internal Peer Code Review Form, in addition to items in the Cover, Executive Summary, and other sections as appropriate
4. Ensuring that the completed form is saved to the appropriate folder on SharePoint, with the required file name format
5. Notifying the reviewer that the form is ready for review as soon as it is ready
6. Responding to reviewer feedback

Reviewer: The individual(s) reviewing the code in the peer review process. This should not be someone who was involved in the original creation of the code, if at all possible.

The reviewer is responsible for:

1. Determining if they will be able to complete a peer review within a reasonable reply time frame
2. Notifying the requester as soon as reasonably possible if they are either able or unable to complete the peer review within a reasonable reply time frame
3. Performing a review of the code based on the information provided by the requester
4. Completing all fields in the Reviewer Feedback Section of the Internal Peer Code Review Form, in addition to items in the Cover, Executive Summary, and other sections as appropriate
5. Ensuring that the completed form is saved to the appropriate folder on SharePoint, with the required file name format
6. Notifying the requester that the form is completed and ready for their review as soon as it is ready
7. Reaching out to the requester for any additional information, such as clarifying questions.
8. Documenting any clarifying questions, meetings, or other relevant items in the Internal Peer Code Review Form. This includes documenting any remaining questions, concerns, or meeting requests.

Reasonable Reply Time Frame: This is the expected timeframe between the time the requester notifies the reviewer, and the time the reviewer completes their review and notifies the requester. This is generally expected to take no more than 5-7 business days (approximately 1 calendar week), unless otherwise agreed. Replies should be completed as soon as reasonably possible.

Review Number: A positive integer (whole number) which indicates the "number" of the peer review. This is an arbitrary number, which increments with each request. As such, Review Number 01 should refer to the first peer review for the project, Review Number 02 the second, and so on. This number has no specific meaning beyond facilitating a somewhat-chronological organization.

Revision Number: A positive integer (whole number) which indicates the "round" of the peer review for a specific request. This is intended to record each distinct "back and forth" between requester and reviewer and increment accordingly. As such, Revision Number 01 should refer to the first request and review for a specific request, Revision Number 02 the second, and so on. It should be uncommon for a revision number to be greater than 02.

Repository: This is the name of the GitHub repository where the code is stored. This should almost always be a personal fork of the project repository.

File Name Protocol: This is the expected file name convention for Internal Peer Code Review Forms. All forms should be saved in the designated folder. Names should follow the following formula:

```
p2p_code_[2 digit review#]_[2 digit revision#]_[repository name]_[brief goal
description]_[requesters initials]_[reviewers initials].docx
```

Values should be in lowercase and avoid special characters (except the included underscores). This format (with the exception of enforcing lowercase) is automatically generated in the header of the form through completion of the cover.

16 Required Software

At this time, the Internal Peer Code Review Form requires the Desktop App Version of Microsoft Word 365. This software is provided by the university at no cost. Training materials provided by IT can be found at the Microsoft 365 at UTHealth webpage. Details about how to access the student provided copies of this software are listed on the Available Technology section of the Information Technology page on the mySPH SharePoint. The mobile app version and web app versions of Microsoft Word 365 do not allow interaction with fillable form fields.

Review of code otherwise has similar software requirements to the writing and creation of the code. Microsoft OneDrive, a web browser, an Integrated Development Environment (IDE) (most commonly RStudio for coding in R), the university Virtual Private Network (VPN), and a stable internet connection are the most common requirements.

17 Process Overview

The overall Internal Peer Code Review process has several uniform steps and expectations:

1. The requester either identifies a need for an Internal Peer Code Review or is requested to do so by project leadership.
2. The requester selects a suitable reviewer from their peers. The following points are intended to be guidance, rather than hard requirements:
    1. It is recommended that both the requester and the reviewer have similar coding experience levels, or that the reviewer has greater coding experience. This is to ensure that the requester receives suitable feedback that provides a learning opportunity.
    2. The requester should keep the reviewer's coding skill in mind when completing the form; reviewers with less coding experience will likely require more detailed explanations to understand why certain choices were made in the code. This may include the choice to use a particular package/library or method.
    3. The requester should contact a potential reviewer in advance of completing their request whenever possible. This is to ensure that the chosen reviewer will be able to complete the review in a reasonable reply time frame, and also allow the requester to tailor the level of detail in their request documentation to their chosen reviewer.
3. The requester completes their portion of the Internal Peer Code Review Form:
    1. The requester creates a copy of the Internal Peer Code Review Form template document.
    2. The requester completes the cover and Requester Provided Information sections of the Internal Peer Code Review Form. This also includes the cover, executive summary, and other sections as appropriate.
    3. The requester verifies that the file name generated in the header matches the File Name Protocol defined in the Key Terms and Definitions section of this document. The requester performs any required modifications (such as conversion to lowercase) and saves the file with the appropriate file name.
    4. The requester uploads this form to the designated SharePoint folder, if it is not already saved or created in that location.

4. The requester notifies the reviewer that the Requester portion of the Internal Peer Code Review Form is complete. The requester should perform this notification via UTHealth email. A direct link to the relevant Internal Peer Code Review Form on SharePoint should be included.
5. The reviewer completes the request in a reasonable reply time frame.
    1. If the reviewer is unable to complete the review in a reasonable reply time frame, they should inform the requester as soon as possible.
    2. The reviewer reviews all code and documentation provided by the requester.
    3. The reviewer reaches out to the requester with any questions or additional items needed to complete the review.
    4. The reviewer completes all Reviewer Feedback sections of the Internal Peer Code Review Form. This also includes the cover, executive summary, and other sections as appropriate.
6. The reviewer notifies the requester that the Internal Peer Code Review Form is complete. The reviewer should perform this notification via UTHealth email. A direct link to the relevant Internal Peer Code Review Form on SharePoint should be included.
7. The requester reviews the feedback given by the reviewer and makes any modifications that are required.

> **Note**
>
> This process should be completed as many times as necessary to achieve a satisfactory final product; each revision cycle requires the completion of a new Internal Peer Code Review Form, with the revision number iterating by 1. It should be rare for a revision number to be greater than 02. Documentation of prior revisions may be referenced or used as a template, so long as it is appropriately updated to reflect the relevant changes.

See Anatomy of the Internal Peer Code Review Form section for further details about individual portions of the Internal Peer Code Review form.

See the Step-by-Step Example section at the end of this part for a Step-by-Step walkthrough of this process in a real-world example.

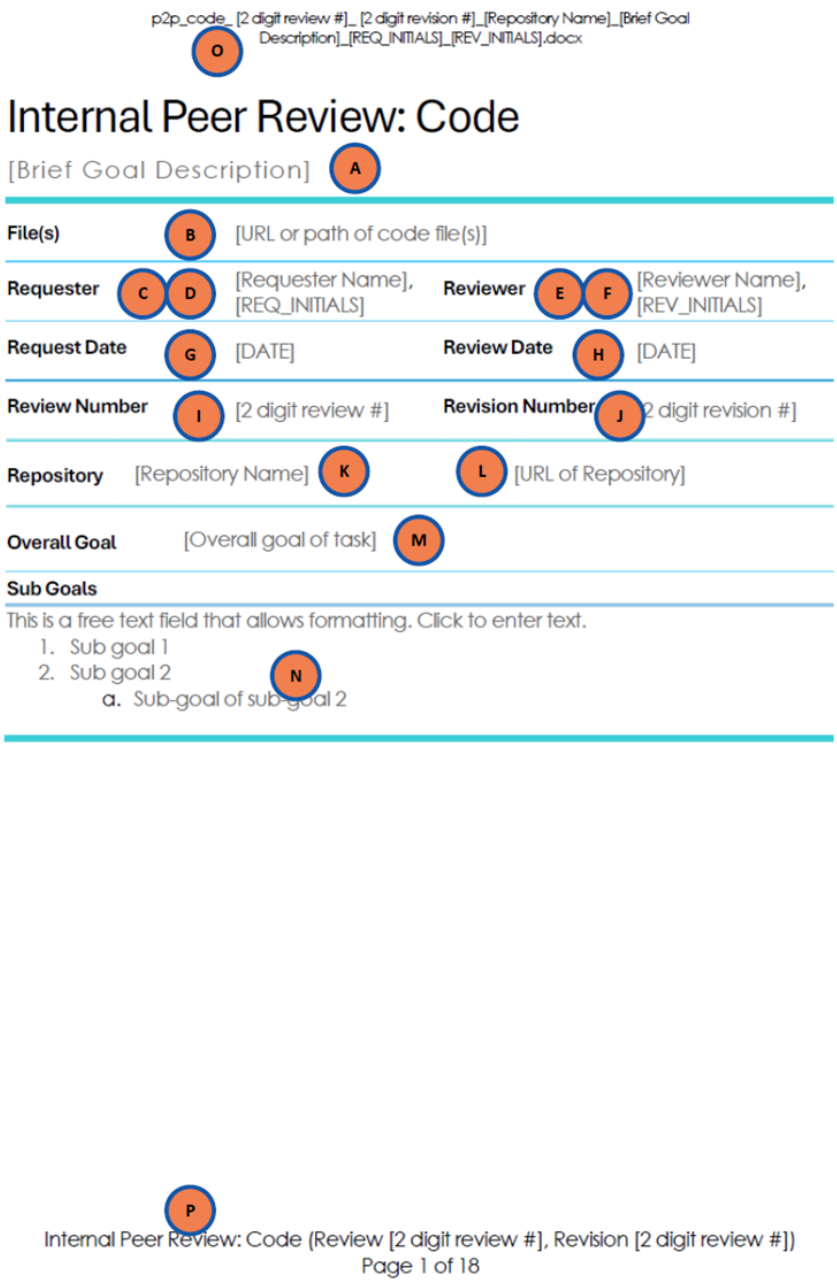
18 Anatomy of the Internal Peer Code Review Form

The Internal Peer Code Review Form is a Word-based Form that features content control fields and other advanced features that limit alteration to the form. The form should not receive substantial alteration to its structure during its completion. Instructions and details are provided within the form to minimize the need to reference this SOP during completion of the form.

The Internal Peer Code Review Form is 18 Pages, organized into 5 main sections: Cover, Executive Summary, Table of Contents, Requester Provided Information, and Reviewer Feedback.

18.0.1 Cover

The Cover is 1 page. It provides an organizational and administrative summary of the Internal Peer Code Review. This should be primarily completed by the requester. The reviewer is responsible for the “Reviewer”, “Reviewer Initials”, and “Review Date” fields.



18.0.1.0.1 Fields

1. **[Brief Goal Description]:** This field is intended to title the peer review, with the title reflecting the goal of the code itself. This title should be informative but brief (< 20

characters).

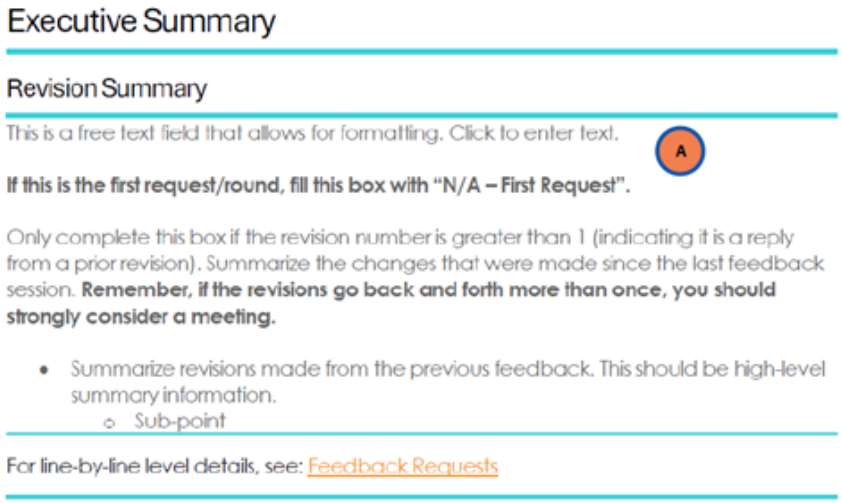
2. **[URL of path of code file(s)]/:** This should be the path or URL that leads directly to the code files involved in the peer review. This text box allows for advanced formatting, such as hyperlinks and bullets. Code files should be on GitHub, unless otherwise specified by the task.
3. **[Requester]:** This should be the first and last name of the requester. Note: This should be clear and reflect other study documents, such as if the author were submitting their name to a peer-reviewed journal publication. For example, Dr. Michael Bradley Cannell, who goes by his middle name, might write "M. Brad Cannell", "Brad Cannell", "Bradley Cannell", or "Michael Bradley Cannell", but should make the same choice for all uses of the form. In the case of similar names on a single team, care should be taken to ensure consistency and clarity while respecting individual identity.
4. **[REQ_INITIALS]:** This should be the first and last initial of the requester.
5. **[Reviewer]:** This should be the first and last name of the reviewer. Note: This should be clear and reflect other study documents, such as if the author were submitting their name to a peer-reviewed journal publication. For example, Dr. Michael Bradley Cannell, who goes by his middle name, might write "M. Brad Cannell", "Brad Cannell", "Bradley Cannell", or "Michael Bradley Cannell", but should make the same choice for all uses of the form. In the case of similar names on a single team, care should be taken to ensure consistency and clarity while respecting individual identity.
6. **[REV_INITIALS]:** This should be the first and last initial of the reviewer.
7. **[Request Date]:** This is the date that the requester submits the peer review request to the reviewer. It is content-locked as a date-picker object and will format dates in YYYY-MM-DD format.
8. **[Review Date]:** This is the date that the reviewer submits their feedback on the peer review request to the requester. It is content-locked as a date-picker object and will format dates in YYYY-MM-DD format.
9. **[Review Number]:** This should be a two-digit integer. See the definition of review number in the "Key Terms and Definitions" section of this document for further details.
10. **[Revision Number]:** This should be a two-digit integer. See the definition of revision number in the "Key Terms and Definitions" section of this document for further details.
11. **[Repository Name]:** This should be the name of the project repository on GitHub.
12. **[URL of Repository]:** This should be the URL of the project repository on GitHub.
13. **[Overall Goal of task]:** This should be the highest-level goal that the code was created to accomplish. This should be a single brief, focused sentence.
14. **Sub Goals:** This should be any major steps or other sub-goals related to the code being reviewed.
15. **Header:** The header uses document fields to automatically generate the appropriate file name. This field is locked to prevent modification but should allow the file name to be copied and pasted.
16. **Footer:** The footer uses document fields to automatically generate values. This field is locked to prevent modification.

18.0.2 Executive Summary

The Executive Summary section is 2 pages. It is intended to capture a high-level summary of key items in the review. This section should be brief and only contain the highest-level key items. The primary target audience for this section is project leadership¸ such as the PI. This section should be the last section completed. There are two subsections in this section, which are 1 page each: one for the requester to summarize changes since the last review of the same material, and one for the reviewer to summarize their feedback and make determinations.

18.0.2.1 Revision Summary

This section is intended to capture a high-level summary of any changes made since the last revision. It should be completed by the requester. It is intended to enable the reviewer to focus their review on these new items, and clearly identify changes when reviewing over multiple revisions. For the first revision, this should have a value of "N/A – First Request".

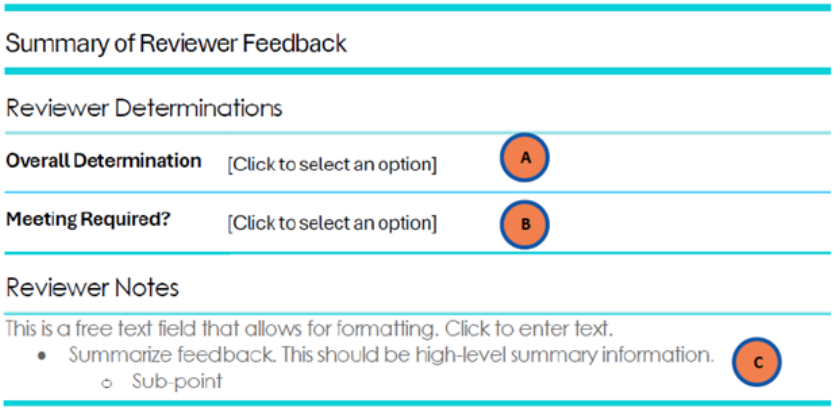


18.0.2.1.1 Fields

1. **Revision Summary:** This box is intended to capture the summary of changes made since the last review cycle, so that the reviewer may focus on these new details. It should be completed by the requester. This is a free-text box that allows for formatting, such as bullets. For the first revision, this should have a value of "N/A – First Request".

18.0.2.2 Summary of Reviewer Feedback

This section is intended to capture a high-level summary of the reviewer's feedback to the requester. It should be completed by the reviewer. This will allow both the requester and project leadership to see the most pertinent items of this feedback.



18.0.2.2.1 Fields

1. **Overall Determination:** This field captures the reviewer's overall determination for the code being reviewed. It has three options:
    1. **Accept (as is, no revision):** This option indicates that the reviewer considers the code fully acceptable without any changes required. As such, the code meets the stated objectives and fits with project SOP requirements such as the style guide or the R4Epi Best Practices.
    2. **Minor Revision:** This option indicates that the reviewer considers the code mostly acceptable. Any requested changes are so minor that another review is not required. This includes things such as small typographical errors or changes to format to comply with the style guide or the R4Epi Best Practices.
    3. **Major Revision:** This option indicates that the reviewer believes the code requires significant changes before it can be accepted, which will require an additional review. This includes things such as converting repetitive code into helper functions, or a change to methods.
2. **Meeting Required?:** This field captures the reviewer's determination of the necessity of additional communication to provide feedback or clarify any of the reviewer's remaining questions. It has three options:

1. **Meeting Needed:** This option indicates that the reviewer believes a synchronous discussion is necessary to answer remaining questions. This should be considered any time there is a possibility for follow-up questions, or if there are more than three questions. If the reviewer reached out to the requester for additional information and had a synchronized discussion (a meeting), this option should be selected to facilitate tracking of this time.
2. **No Meeting or Questions:** This option indicates that the reviewer has no remaining questions. It should only be selected if all information required for the reviewer to understand the code and the task parameters was provided in the Internal Peer Code Review Form or the code itself.
3. **Questions (No Meeting):** This option indicates that the reviewer has a few remaining questions, but answers are not likely to result in additional follow-up questions. If the reviewer reached out to the requester for additional information, but did not have a meeting, this option should be selected to facilitate tracking of this time.

3. **Reviewer Notes:** This box is intended to capture the summary of the reviewer's feedback. This is a free-text box that allows for formatting, such as bullets. This field should also indicate if a meeting has (or has not) already been held or scheduled, as well as any remaining questions.

18.0.3 Table of Contents

The Table of Contents assists readers in locating specific information and fields. It is automatically generated by Word and should be updated by both the requester and the reviewer prior to each submission to ensure it is accurate. The Table of Contents reflects the headings that also appear in the Navigation Pane of Microsoft Windows 365. The Navigation Pane may be useful to users that wish to "jump" to a specific section of the document.

Contents

18.0.4 Requester Provided Information

This section organizes information provided by the requester. It is a minimum of 9 pages, and contains 8 subsections: Requester Process Checklist, Versioning & Dependencies, Input Data Notes, Output Data notes, Test File Notes, Task Process Notes, Helper Function Notes, and Feedback Requests. It should be completed by the requester.

18.0.4.1 Requester Process Checklist

The Requester Process Checklist is a tool to assist the requester in completion of the form. It is a minimum of 2 pages in length (with page break). It should be completed by the requester.

Requester Provided Information

Requester Process Checklist

	Item	Comments
☐	Enter values for the file to review, requester name, date of request, review number, revision number, and repository link	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Clearly outline goals of the code files	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Create brief title from goal	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Add Reviewer ("xx" for initials if unknown)	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Verify header and footer are complete	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Name file and save	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Add Version & Dependency Information	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Provide Input Data link and description	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Provide Output Data link and description	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Provide Test File link and description	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Outline Task Process Notes.	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Provide details of Helper Functions	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	List Feedback Requests	This is a free text field that allows formatting, such as bullets. Click to enter text.

	Item	Comments
☐	Complete Revision Summary in Executive Summary ("NA – First Request" for first revision)	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Check documentation is complete	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Refresh Table of Contents (Update All)	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	Send to Reviewer(s)	This is a free text field that allows formatting, such as bullets. Click to enter text.

18.0.4.1.1 Fields

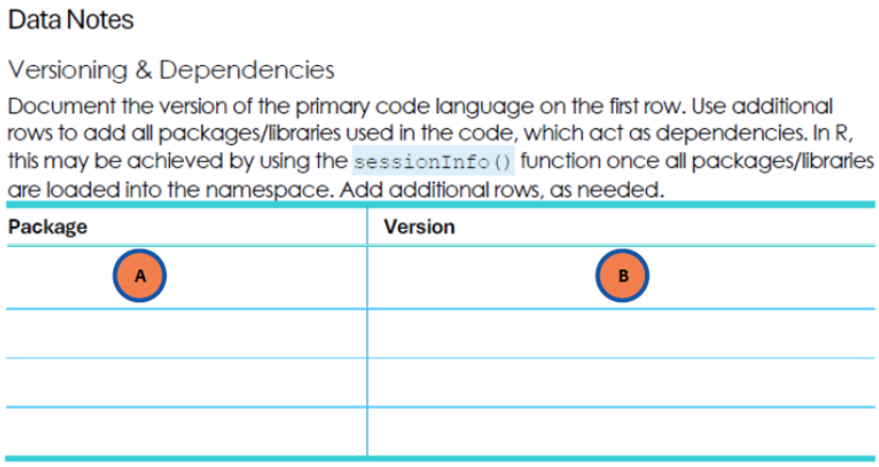
1. **Item:** Each item in the checklist has a checkbox and descriptive name. They are organized in an approximately chronological order to assist the requester in completion of the form.
2. **Comments:** Each item in the checklist has a comment box, which allows for formatting such as bullets. This space is intended to allow the requester to take notes or record

other comments about each step of the form's completion. This may include a mention that a certain process was blocked or otherwise unable to be completed due to a technical error.

18.0.4.2 Versioning & Dependencies

This section is meant to outline the versions used in developing the code. It is a minimum of 1 page in length (with page break). It should be completed by the requester. This section should include the version of the primary language (most likely R) and any imported packages/libraries. This information may be crucial to ensuring that code tests properly, as significant changes may occur between versions that would explain otherwise unpredictable changes in code behavior.

In R, this information may be obtained by using the `sessionInfo()` function after all packages/libraries are loaded into the session namespace. The rare instance of a package/library that is required but does not have versioning information should also be mentioned. Additional rows may be added as required.


Data Notes

Versioning & Dependencies

Document the version of the primary code language on the first row. Use additional rows to add all packages/libraries used in the code, which act as dependencies. In R, this may be achieved by using the `sessionInfo()` function once all packages/libraries are loaded into the namespace. Add additional rows, as needed.

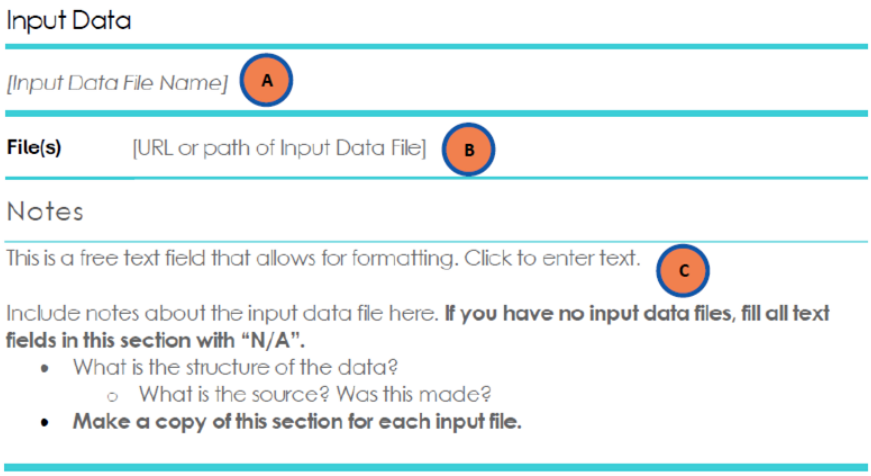
Package	Version
A	B

18.0.4.2.1 Fields

1. **Package:** This is the name of the library or package. The first line should be the primary coding language (most likely R).
2. **Version:** This should be the version of the package/library used when developing the code.

18.0.4.3 Input Data Notes

This section is meant to outline aspects of the input data that the code utilizes, if it exists. It is a minimum of 1 page in length (with page break). It should be completed by the requester. It is intended to be a high-level summary, not a full encapsulation of every aspect of the input data itself. This section is intended to be copied and pasted as many times as necessary, so each distinct input data file has its own distinct documentation.

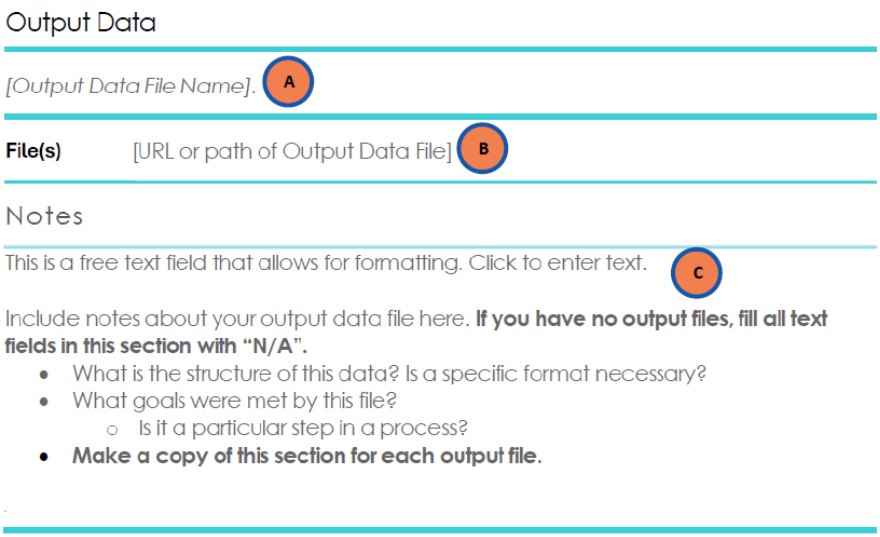


18.0.4.3.1 Fields

1. **[Input Data File Name]:** This is the name of the file. This should be based on how the file is discussed within this form and within the code being reviewed.
2. **[URL or path of Input Data File]:** This should be the direct link or path to the input file, so that the reviewer may also access this file in their review.
3. **Notes:** This section is intended to allow high-level summary notes about the input data. Suggested items include the source and structure of the input data, as well as how it was created (if it was created by the project). Screenshots and figures may be included, as appropriate.

18.0.4.4 Output Data Notes

This section is meant to outline aspects of the output data generated by the code, if it exists. It is a minimum of 1 page in length (with page break). It should be completed by the requester. It is intended to be a high-level summary, not a full encapsulation of every aspect of the output data itself. This section is intended to be copied and pasted as many times as necessary, so each distinct output data file has its own documentation.

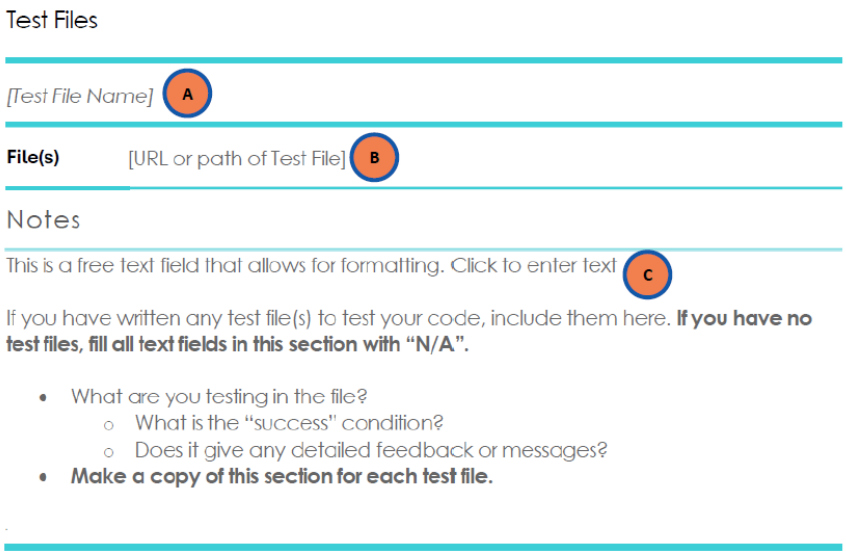


18.0.4.4.1 Fields

1. **[Output Data File Name]:** This is the name of the file. This should be based on how the file is discussed within this form and within the code being reviewed.
2. **[URL or path of Output Data File]:** This should be the direct link or path to the output file, so that the reviewer may also access this file in their review.
3. **Notes:** This section is intended to allow high-level summary notes about the output data. Suggested items include the source and structure of the output data file, what goal is met by this output data, and if it serves as a step within the process (rather than the final product). Screenshots and figures may be included, as appropriate.

18.0.4.5 Test File Notes

This section is meant to outline aspects of any test files for the code, if they exist. It is a minimum of 1 page in length (with page break). It should be completed by the requester. It is intended to be a high-level summary, not a full encapsulation of every aspect of the file(s). This section is intended to be copied and pasted as many times as necessary, so each distinct test file has its own documentation.

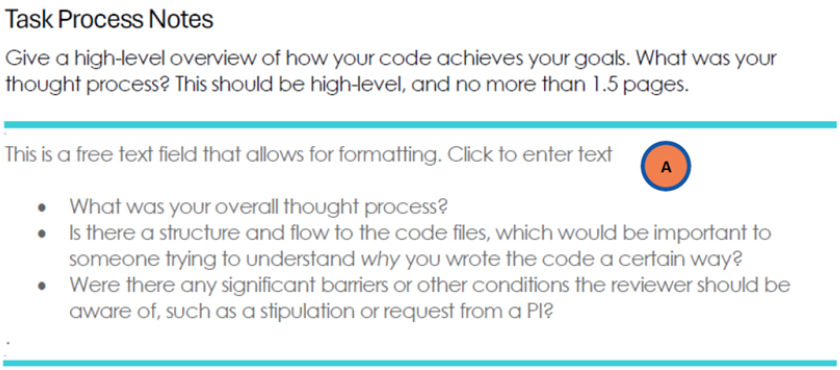


18.0.4.5.1 Fields

1. **[Test File Name]:** This is the name of the file. This should be based on how the file is discussed within this form and within the code being reviewed.
2. **[URL or path of Test File]:** This should be the direct link or path to the test file, so that the reviewer may also access this file in their review.
3. **Notes:** This section is intended to allow high-level summary notes about the test file. Suggested items include a description of the "success" condition, and any detailed feedback or messages provided by the test file. Screenshots and figures may be included, as appropriate.

18.0.4.6 Task Process Notes

This section is meant to outline the process by which the code achieves its stated objectives. It is a minimum of 1 page in length (with page break). It should be completed by the requester. This section is a free-text box that allows for advanced formatting, such as bullets and figures. This field should primarily include identification of barriers that influenced the task solution, the requester's thought process and/or logic that led them to the chosen solution, and other high-level items. Any included items should facilitate the reviewer in understanding the logic behind the code, and the logic behind how the code was written and organized. It is the primary "free-text" area of the requester's documentation, where the requester may include any necessary documentation that does not neatly belong to any other section. It should be no longer than 1.5 pages in length.

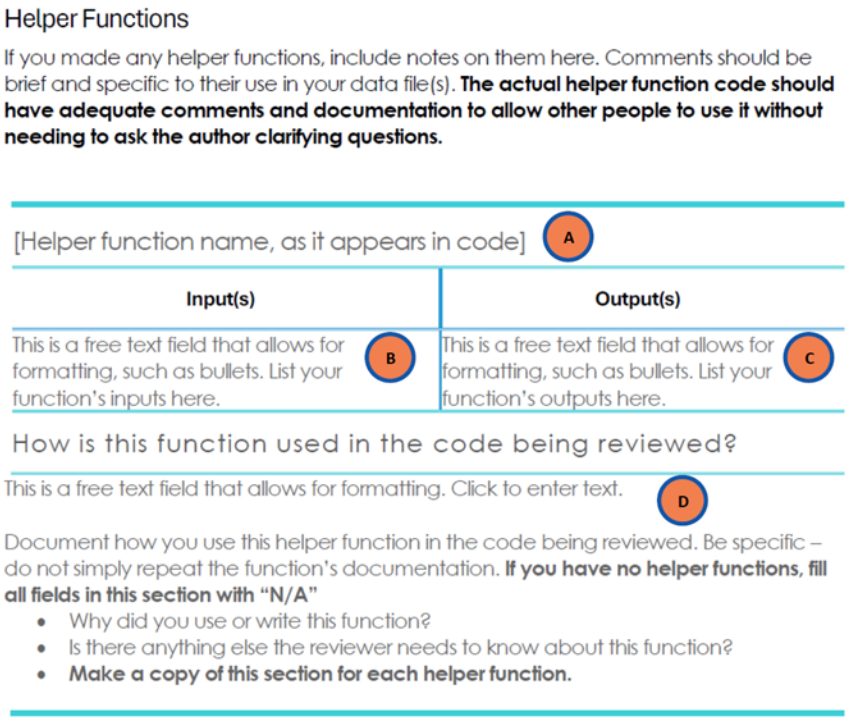


18.0.4.6.1 Fields

1. **Notes:** This section is intended to capture and organize the requester's thought process and organization of their code. Suggested items include the overall thought process, any structure to the code files, and any significant barriers or other conditions that the reviewer should be aware of. Screenshots and figures may be included, as appropriate.

18.0.4.7 Helper Function Notes

This section is meant to identify and summarize any helper functions that were written and utilized within the code file(s). It is a minimum of 1 page in length (with page break). It should be completed by the requester. It is intended to flag these functions for the reviewer, so that the reviewer knows the specific use of this function in the code file and that it is included in the review request. It is not meant to duplicate the documentation that should be in the code for the helper function. However, it may prompt the requester to add missing documentation within their code before submitting their request. This section is intended to be copied and pasted as many times as necessary, so each distinct helper function has its own documentation.

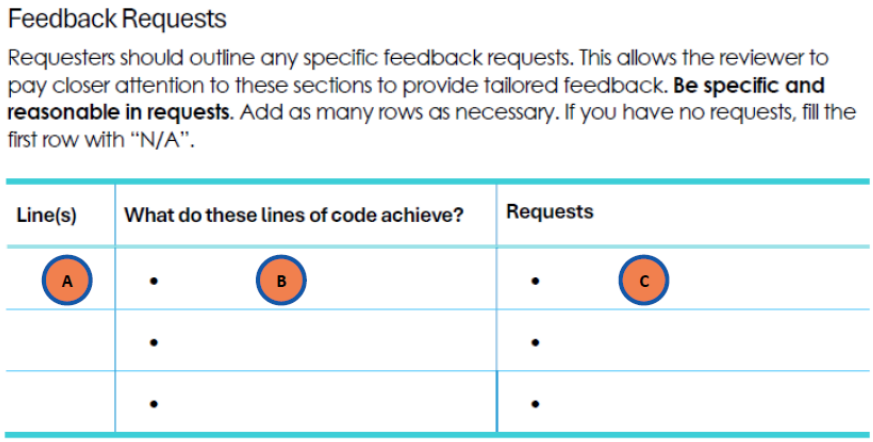


18.0.4.7.1 Fields

1. **[Helper function name, as it appears in code]:** This is the name of the function, as it appears in the code file.
2. **Input(s):** This should be a brief list of the inputs. This may be the specific inputs used in the code file, such as "continuous numeric variable column names, such as age", or more general, such as "column names". Full documentation of the inputs, such as constraints, should be within the code of the helper function itself.
3. **Output(s):** This should be a brief list of the outputs. This may be the specific outputs used in the code file, such as "values for summary statistics used in the stats tibble", or more general, such as "summary statistics of mean and standard deviation". Full documentation of the inputs, such as constraints, should be within the code of the helper function itself.
4. **How is this function used in the code being reviewed?:** This section is intended to provide a summary of how this helper function was used in the code. Suggested items include a rationale for the code's creation. Screenshots and figures may be included, as appropriate.

18.0.4.8 Feedback Requests

This section serves to organize specific feedback requests from the requester. It is a minimum of 1 page in length (with page break). It should be completed by the requester. Any specific area that the requester wants more detailed or specific feedback from the reviewer should be included in this section. These requests should be as specific as possible, and as reasonable as possible. The table allows for the addition of as many rows as required.



18.0.4.8.1 Fields

1. **Line(s):** This column should include the specific lines of code relating to the request. The requester should not include the "entire document" in a single request.
2. **What do these lines of code achieve?:** This column should include a brief summary of what purpose the indicated lines serve. This will allow the reviewer to contextualize this piece of code quickly, without having to review the entire surrounding document to provide a focused review.
3. **Requests:** This column allows the requester to make a specific request to the reviewer. This may be relatively broad, such as "Can you think of a more concise way to achieve this?" or may be highly specific, such as "Does this spacing meet the SOP requirements for code formatting?"

18.0.5 Reviewer Feedback

This section organizes information provided by the reviewer. It is a minimum of 5 pages, and contains 3 subsections: Reviewer Process Checklist, General Feedback, and File Specific Feedback. It should be completed by the reviewer.

18.0.5.1 Reviewer Process Checklist

The Reviewer Process Checklist is a tool to assist the reviewer in completion of the form. It is a minimum of 2 pages in length (with page break). It should be completed by the reviewer.

Reviewer Feedback

Reviewer Process Checklist

	Item	Comments
☐	Check requester documentation is complete	This is a free text box that allows for formatting, such as bullets. Click to text.
☐	Complete reviewer name and date, check file name and save	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Review requester documentation	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Clone reviewee's copy of repository	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Ensure files to be reviewed are available and accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Ensure input file(s) exists and are accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Ensure output file(s) exists and are accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Ensure test file(s) exists and are accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Ensure helper function(s) are available and accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Review code	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Re-run files to check output	This is a free text box that allows for formatting, such as bullets. Click to enter text.

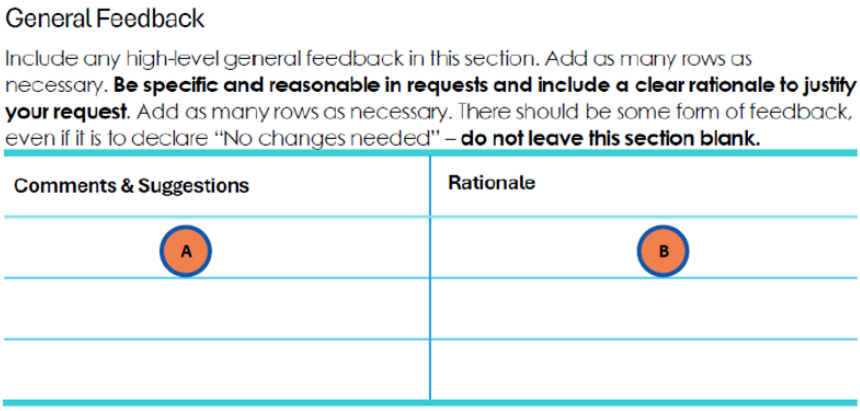
	Item	Comments
☐	Review feedback requests and respond	This is a free text box that allows for formatting, such as bullets. Click to text.
☐	Complete any line-by-line or general feedback	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Complete Summary of Reviewer Feedback in Executive Summary	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Make acceptance determination	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Make meeting determination	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Refresh Table of Contents (Update All)	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Check documentation is complete and return to requester	This is a free text box that allows for formatting, such as bullets. Click to enter text.

18.0.5.1.1 Fields

1. **Item:** Each item in the checklist has a checkbox and descriptive name. They are organized in an approximately chronological order to assist the reviewer in completion of the form.
2. **Comments:** Each item in the checklist has a comment box, which allows for formatting such as bullets. This space is intended to allow the reviewer to take notes or record other comments about each step of the form's completion. This may include a mention that a certain process was blocked or otherwise unable to be completed due to a technical error.

18.0.5.2 General Feedback

This section is meant to capture general feedback relating to all code relating to the task. It is a minimum of 1 page in length (with page break). It should be completed by the reviewer. This should be high-level feedback. Each comment/suggestion should include a rationale, which is meant to ensure the reviewer presents a clear, objective, logical basis for their feedback. The requester should be able to understand the "why" underlying any suggestion posed by the reviewer. The table allows for additional rows to be added, as necessary. This section should never be blank, even if it's only to state "good job, no changes needed" with a rationale of "all requirements met".

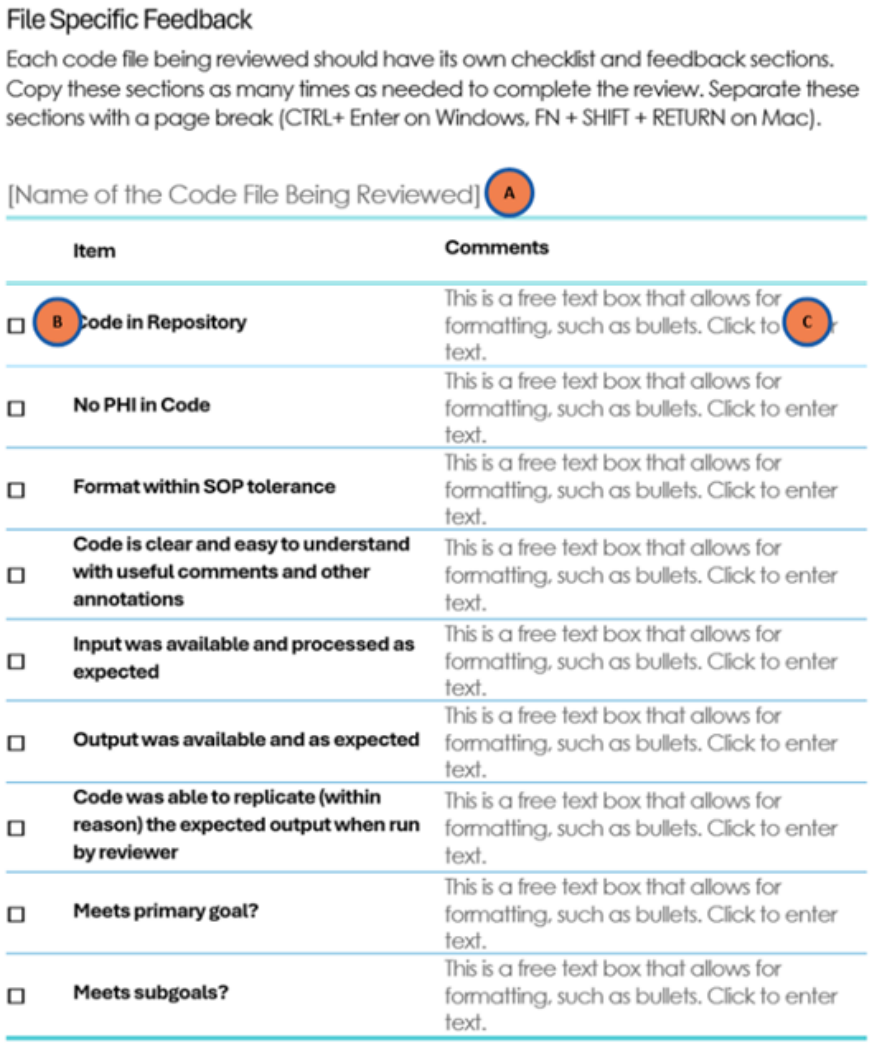


18.0.5.2.1 Fields

1. **Comments & Suggestions:** Each comment or suggestion should receive its own row. The comment or suggestion should be concise and clear.
2. **Rationale:** Each comment or suggestion should have clear, objective, and logical reasoning behind it. This field should allow the requester to understand the "why" behind every comment or suggestion made by the reviewer.

18.0.5.3 File Specific Feedback

This section should form the bulk of the reviewer's feedback. It is a minimum of 2 pages in length (with page breaks). It should be completed by the reviewer. It is intended to collect specific feedback for each code file involved in the code review. This section is intended to be copied and pasted as many times as necessary, so each code file being reviewed has its own documentation. Tables allow for additional rows to be added, as necessary. Tables should not be left entirely blank.



File Specific Feedback

Each code file being reviewed should have its own checklist and feedback sections. Copy these sections as many times as needed to complete the review. Separate these sections with a page break (CTRL+ Enter on Windows, FN + SHIFT + RETURN on Mac).

[Name of the Code File Being Reviewed] A

	Item	Comments
☐	B Code in Repository	This is a free text box that allows for formatting, such as bullets. Click to C text.
☐	No PHI in Code	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Format within SOP tolerance	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Code is clear and easy to understand with useful comments and other annotations	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Input was available and processed as expected	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Output was available and as expected	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Code was able to replicate (within reason) the expected output when run by reviewer	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Meets primary goal?	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Meets subgoals?	This is a free text box that allows for formatting, such as bullets. Click to enter text.

18.0.5.3.1 Fields

1. **[Name of the Code File Being Reviewed]:** This is the name of the specific code file being reviewed in this section.
2. **Item:** Each item in the checklist has a checkbox and descriptive name. They are organized in an approximately chronological order to assist the reviewer in completion of the form.
3. **Comments:** Each item in the checklist has a comment box, which allows for formatting such as bullets. This space is intended to allow the reviewer to take notes or record other comments about each step of the form's completion. This may include a mention that a certain process was blocked or otherwise unable to be completed due to a technical error.
4. **Comments & Suggestions:** Each comment or suggestion should receive its own row.
5. **Rationale:** Each comment or suggestion should have clear, objective, and logical reasoning behind it. This field should allow the requester to understand the "why" behind every comment or suggestion made by the reviewer.
6. **Line(s):** This column should include the specific lines of code relating to the reviewer's feedback. This should never include the "entire document".
7. **Issue:** This column should clearly outline the issue the reviewer has identified in these specific lines of code. This should be concise but specific.
8. **Suggestion(s):** This column should include the reviewer's suggested modification, if one exists. Reviewers should try to make a suggestion whenever possible, though they are not required to write detailed code or otherwise "do it all themselves" to achieve this.
9. **Rationale:** The identified issue and suggestion(s) should have a clear, objective, and logical reasoning behind them. This field should allow the requester to understand the "why" behind every issue identified or suggested change presented by the reviewer.

19 Step-by-Step Example

This step-by-step example utilizes a scenario to demonstrate the process.

19.0.0.0.1 Scenario

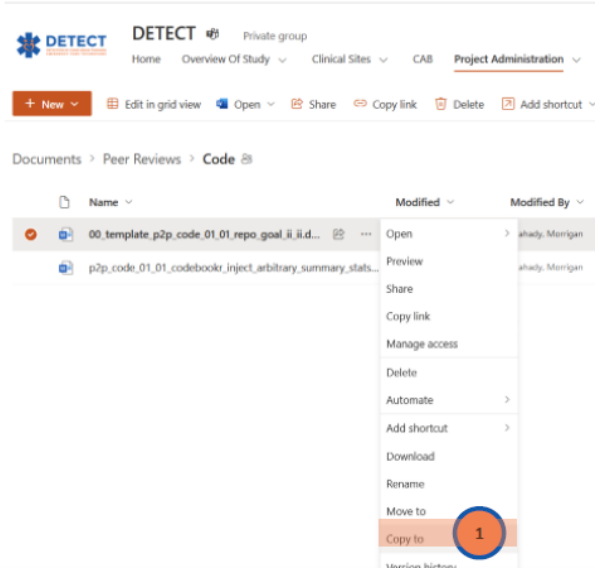
Ebiekimie Dambo (who goes by "Ebie") was assigned a code-based task. This task was to modify the codebookr R package (maintained by Dr. Cannell) so that a user could use a CSV file to add column attributes in bulk. This required the use of two input files to create one output file. There were no helper functions or test files. Ebie had no specific requests for review, as she was primarily concerned about if the code would achieve the task objective reliably for another person. After completing this task, Ebie needed an Internal Peer Code Review. It was discussed at the weekly research team meeting on Monday, June 10, 2024, and Morrigan Mahady (who goes by "Morri") accepted the role of reviewer.

19.0.1 Requester

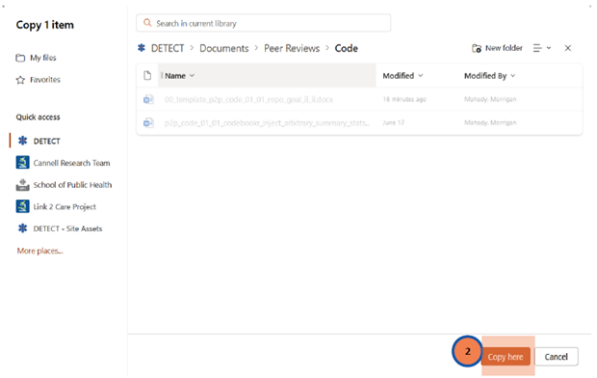
Ebie's process as the requester is broken down into 42 steps, with screenshots or figures to provide additional clarity for the step-by-step instructions.

19.0.1.1 Copying the Template for the Internal Peer Code Review Form

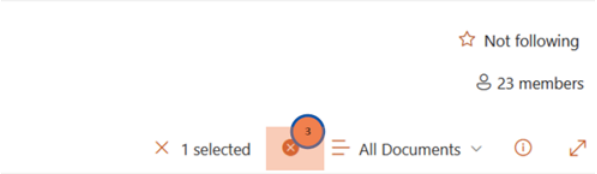
1. First, Ebie must copy the template. She navigates to the Peer Review folder on SharePoint. On the template file, she selects the option to `Copy to`



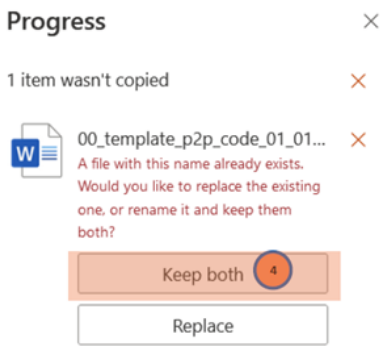
2. She then selects “Copy here” to duplicate the template in the same folder.



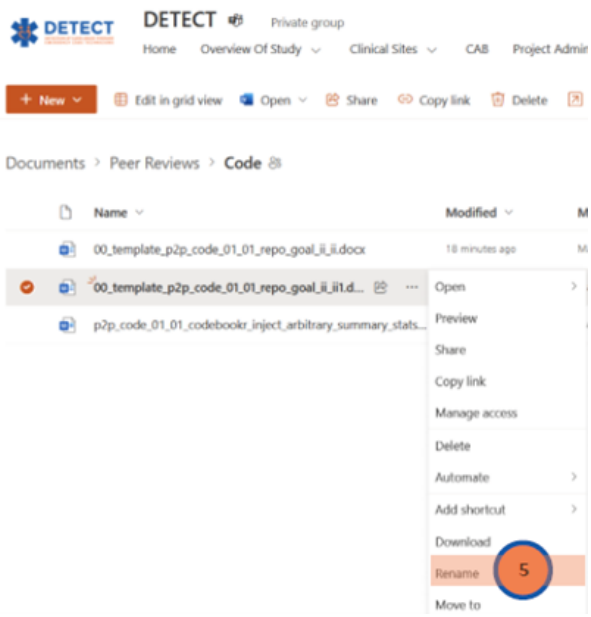
3. She clicks into the small “Error notification” on SharePoint.



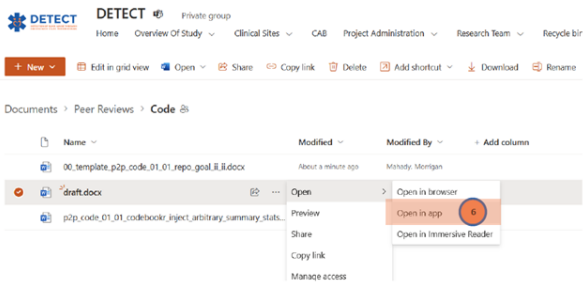
4. She selects “Keep both” to resolve the error.



5. She enters the menu for her new copy of the template file (marked by the orange icon) and selects the option to rename the file. She names it “draft”, for now.

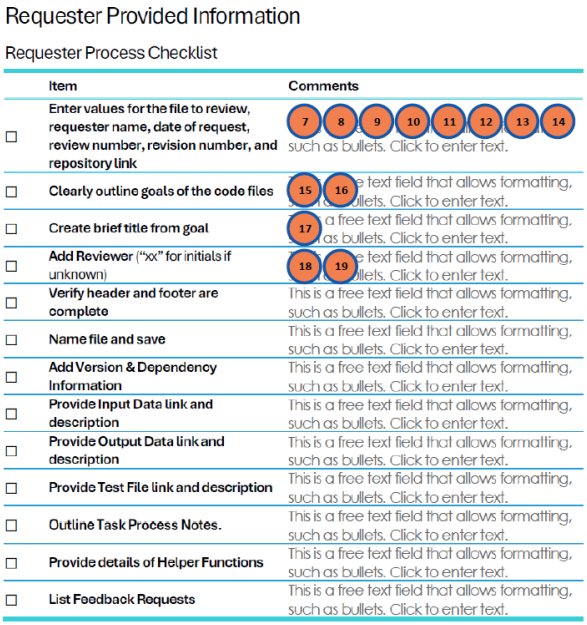


6. She then selects the option to open the file in the Desktop App Version of Microsoft Word 365.



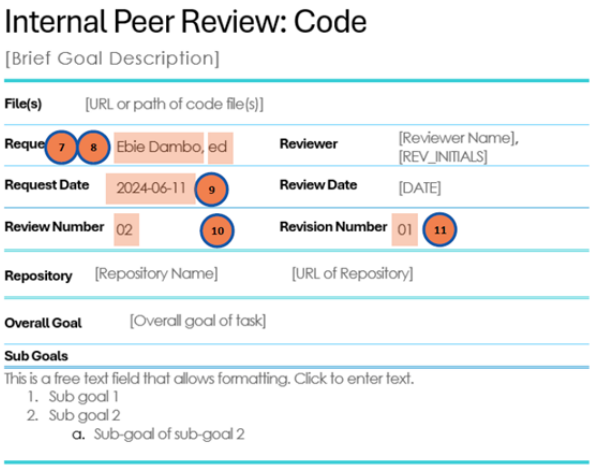
19.0.1.2 Cover

After copying the Internal Peer Code Review form template file, Ebie looks at the Requester Process Checklist that starts on page 5. She sees that the first several items refer to fields in the Cover section, on page 1.

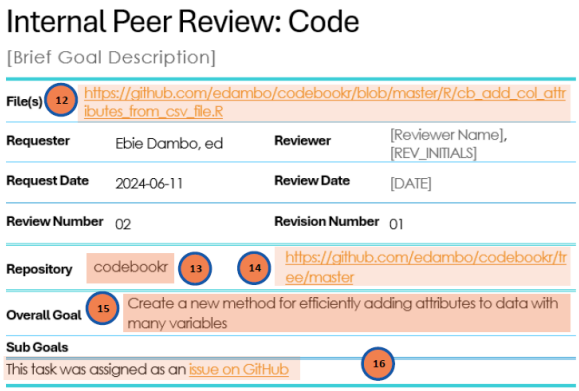


7. Ebie enters her own name in the "Requester Name" field.
8. Ebie enters her own initials, "ed", in the "Requester Initials" field.
9. Ebie enters the date of her request (June 11, 2024) using the date-picker. It is automatically formatted for her.
10. As this is the second Internal Peer Code Review for the project, Ebie enters the 02 in the "Review Number" field.

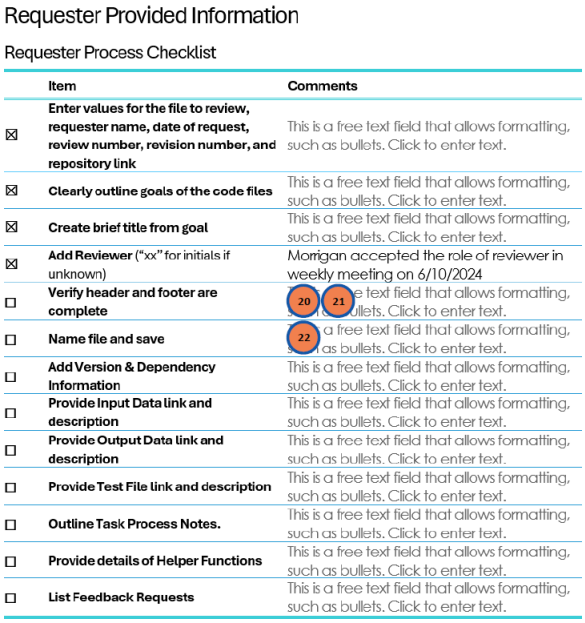
11. As this is the first time Ebie is requesting a review of this code, she enters 01 in the "Revision Number" field.



12. She enters the URL to the code file she wants the reviewer to check. There is only one code file for this request.
13. Ebie enters the name of the repository, which is "codebookr".
14. She enters the URL to her personal fork of this repository, as that's the fork that contains the code she needs reviewed. If it passes review, it might be added to the main repository fork.
15. She enters a brief description of the main goal of her task: to create a new method for efficiently adding attributes to data with many variables. She knows that Morri, her intended reviewer, is familiar with the codebookr package.
16. While there are no sub-goals, Ebie does mention that this task is associated with an issue on GitHub. She utilizes the ability to do advanced formatting in this field by creating a hyperlink in the text.



17. She creates a brief title for this code review based on the Overall Goal she identified in Step 15. She calls it "Add-Col-Attrs-CSV" to keep it clear and under 20 characters.
18. She then adds her reviewer's name to the "Reviewer Name" field.
19. She adds her reviewer's initials, "mm", to the "Reviewer Initials" field. If she did not know her reviewer at this time, she would put "xx" in the "Reviewer initials" field to complete the header. Ebie then checks her progress in the checklist. She adds a comment to the "Add Reviewer" item; this comment documents when Morrigan agreed to the role of reviewer. This is not required.



Requester Provided Information

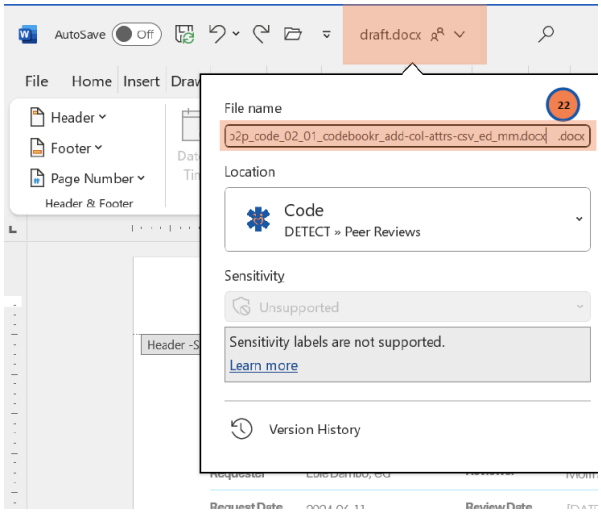
Requester Process Checklist

	Item	Comments
☒	**Enter values for the file to review, requester name, date of request, review number, revision number, and repository link**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Clearly outline goals of the code files**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Create brief title from goal**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Add Reviewer** ("xx" for initials if unknown)	Morrigan accepted the role of reviewer in weekly meeting on 6/10/2024
☐	**Verify header and footer are complete**	20 21 e text field that allows formatting, ullets. Click to enter text.
☐	**Name file and save**	22 a free text field that allows formatting, as bullets. Click to enter text.
☐	**Add Version & Dependency Information**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Provide Input Data link and description**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Provide Output Data link and description**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Provide Test File link and description**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Outline Task Process Notes.**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Provide details of Helper Functions**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**List Feedback Requests**	This is a free text field that allows formatting, such as bullets. Click to enter text.

She sees the final items on the Cover section on Page 1. These items require her to check the header and footer, and save the document with the proper file name protocol. 20. She checks that the footer successfully completed itself. The review number and revision number match what she entered on the Cover fields, so she accepts it as complete.

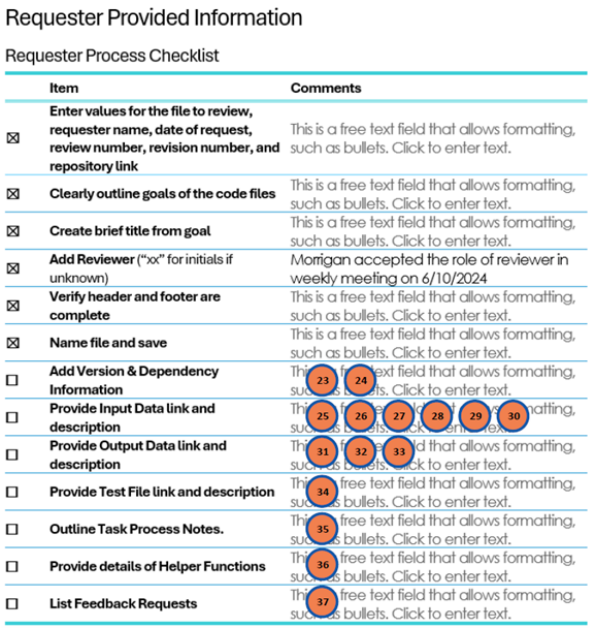
21. She checks that the header successfully completed itself. The review number, revision number, repository name, brief goal title, requester initials, and reviewer initials match what she entered on the Cover fields. She checks the format against the File Name Protocol. Since the file name matches the protocol, she accepts the header as complete.

22. She navigates to the file name, located at the top ribbon of the Microsoft Word 365 Desktop App. She renames the file from copying and pasting the name in the header, and correcting the values to ensure they are lowercase.



19.0.1.3 Requester Provided Information Section

Ebie documents her progress in the checklist. She is able to see that the next several items will take her through the Requester Provided Information Section of the form.



Requester Provided Information

Requester Process Checklist

	Item	Comments
☒	**Enter values for the file to review, requester name, date of request, review number, revision number, and repository link**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Clearly outline goals of the code files**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Create brief title from goal**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Add Reviewer** ("xx" for initials if unknown)	Morrigan accepted the role of reviewer in weekly meeting on 6/10/2024
☒	**Verify header and footer are complete**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Name file and save**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Add Version & Dependency Information**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Provide Input Data link and description**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Provide Output Data link and description**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Provide Test File link and description**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Outline Task Process Notes.**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**Provide details of Helper Functions**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☐	**List Feedback Requests**	This is a free text field that allows formatting, such as bullets. Click to enter text.

23. Ebie opens her code in R Studio. After loading all packages and libraries, she uses the sessionInfo() command to view version information. She finds both expected packages listed: flextable and dplyr. The sessionInfo() command, R version, and dependency packages with versions are highlighted in the figure, for reference. Note that Ebie did not document every single package listed, as those packages are not directly utilized in her code.

```
> sessionInfo()
R version 4.2.2 (2022-10-31 ucrt)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 22631)

Matrix products: default

locale:
[1] LC_COLLATE=English_United States.utf8
[2] LC_CTYPE=English_United States.utf8
[3] LC_MONETARY=English_United States.utf8
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.utf8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods
[7] base

other attached packages:
[1] flextable_0.9.1 dplyr_1.1.1

loaded via a namespace (and not attached):
 [1] zip_2.3.0               Rcpp_1.0.10
 [3] fontBitstreamVera_0.1.1 pillar_1.9.0
 [5] compiler_4.2.2          later_1.3.1
 [7] gfonts_0.2.0            tools_4.2.2
 [9] uuid_1.1-0              digest_0.6.31
[11] evaluate_0.20           jsonlite_1.8.4
[13] lifecycle_1.0.3         tibble_3.2.1
[15] pkgconfig_2.0.3         rlang_1.1.0
[17] shiny_1.7.4             cli_3.6.1
[19] rstudioapi_0.14         crul_1.3
[21] curl_5.0.0              xfun_0.41
[23] fontLiberation_0.1.0    fastmap_1.1.1
[25] xml2_1.3.3              officer_0.6.2
[27] knitr_1.42              askpass_1.1
[29] systemfonts_1.0.4       gdtools_0.3.3
[31] generics_0.1.3          vctrs_0.6.1
[33] grid_4.2.2              tidyselect_1.2.0
[35] glue_1.6.2              httpcode_0.3.0
[37] data.table_1.14.8       R6_2.5.1
[39] textshaping_0.3.6       fansi_1.0.4
[41] rmarkdown_2.21          magrittr_2.0.3
[43] fontquiver_0.2.1        promises_1.2.0.1
[45] htmltools_0.5.5         ellipsis_0.3.2
[47] mime_0.12               xtable_1.8-4
[49] httpuv_1.6.10           ragg_1.2.5
[51] utf8_1.2.3              openssl_2.0.6
[53] crayon_1.5.2
>
```

23

24. She documents her version of R, as well as the version of both packages she used in her code.

Data Notes

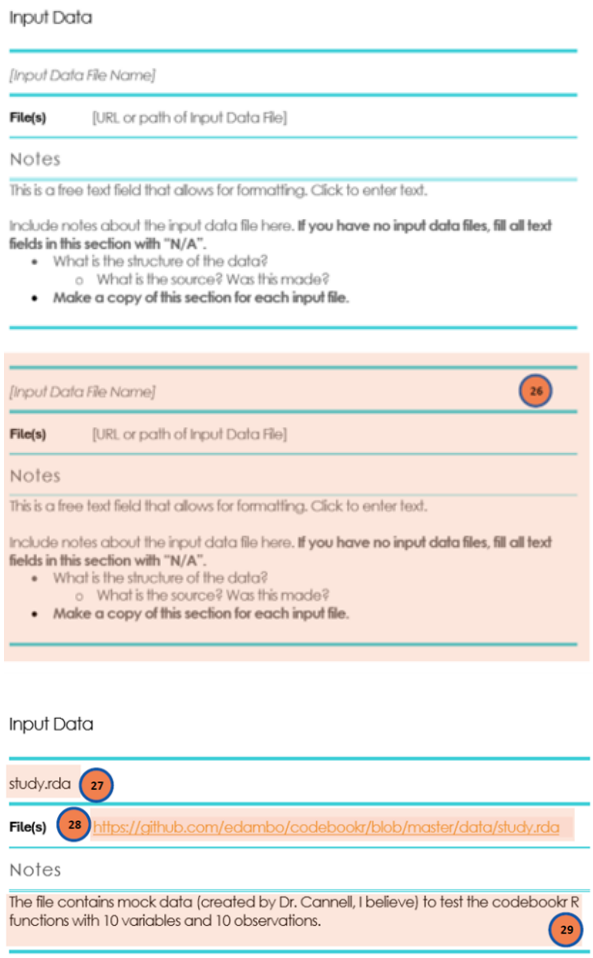
Versioning & Dependencies

Document the version of the primary code language on the first row. Use additional rows to add all packages/libraries used in the code, which act as dependencies. In R, this may be achieved by using the `sessionInfo()` function once all packages/libraries are loaded into the namespace. Add additional rows, as needed.

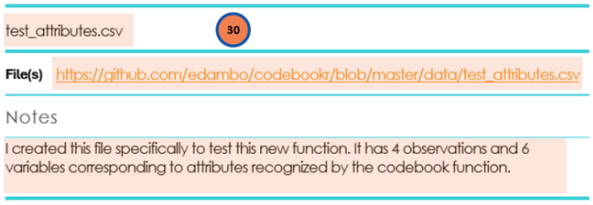
Package	Version
R	4.2.2
dplyr	1.1.1
flextable	0.9.1

24

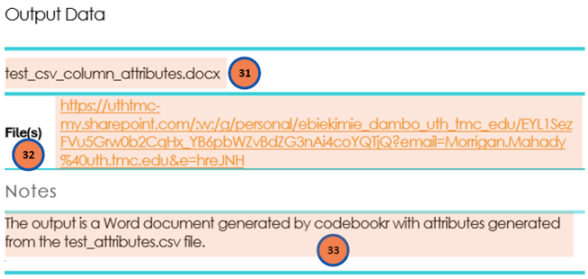
25. Ebie reviews her input data. She has two separate input data files that she used in this code task.
26. She begins by duplicating the Input Data section, so she can document each input data file separately.



27. She documents the name of the first input data file.
28. She documents the URL of the first input data file. It's hosted on GitHub, in the same fork as her code, because this data is testing data that has no PHI.
29. She makes notes about the first input data file. These notes comment on the file's purpose, how it was created, and its form. Note how this is brief and does not give a full exhaustive summary of every aspect of the data.
30. Ebie then repeats this process with the second input data file.

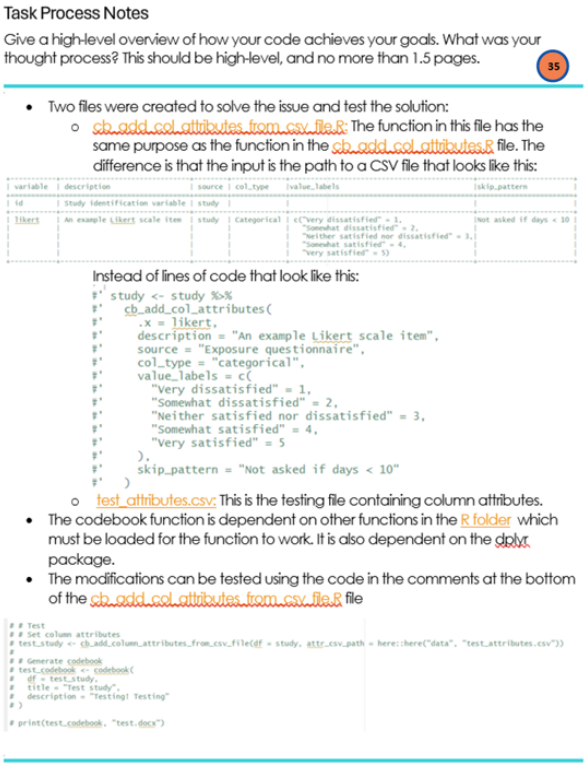


31. Ebie has only one Output Data file, so she does not need to duplicate the Output Data section. She records the name of her output data file.
32. She records the URL of her output data file. This is stored on her personal SharePoint. If a file is stored on a personal SharePoint, it must be shared with the reviewer.
33. She documents notes about the output data file. These notes mention its format, and why it is a "success" condition.



34. Ebie has no Test Files, so she documents "N/A" in all boxes in the Test File section. However, she does have a commented-out snippet of code that acts like a test in her main code file. She makes a mental note to document this elsewhere.

35. Ebie outlines her thought process in the Task Process Notes section. She makes use of the advanced formatting allowed in this text box to include hyperlinks and useful screenshots. She specifically mentions the commented-out test code she noted in Step 34. These detailed notes take up one page.



Task Process Notes

Give a high-level overview of how your code achieves your goals. What was your thought process? This should be high-level, and no more than 1.5 pages.

35

- Two files were created to solve the issue and test the solution:
  - cb_add_col_attributes_from_csv_file.R: The function in this file has the same purpose as the function in the cb_add_col_attributes.R file. The difference is that the input is the path to a CSV file that looks like this:

variable	description	source	col_type	value_labels	skip_pattern
id	Study identification variable	study			
likert	An example Likert scale item	study	Categorical	c("Very dissatisfied" = 1, "Somewhat dissatisfied" = 2, "Neither satisfied nor dissatisfied" = 3, "Somewhat satisfied" = 4, "Very satisfied" = 5)	Not asked if days < 10

Instead of lines of code that look like this:

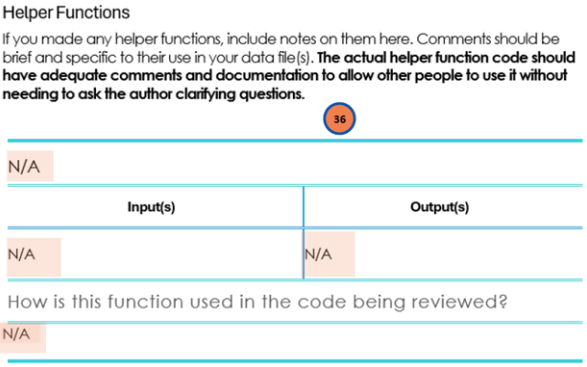
```
#' study <- study %>%
#'   cb_add_col_attributes(
#'     .x = likert,
#'     description = "An example Likert scale item",
#'     source = "Exposure questionnaire",
#'     col_type = "categorical",
#'     value_labels = c(
#'       "Very dissatisfied" = 1,
#'       "Somewhat dissatisfied" = 2,
#'       "Neither satisfied nor dissatisfied" = 3,
#'       "Somewhat satisfied" = 4,
#'       "Very satisfied" = 5
#'     ),
#'     skip_pattern = "Not asked if days < 10"
#'   )
```

  - test_attributes.csv: This is the testing file containing column attributes.
- The codebook function is dependent on other functions in the R folder which must be loaded for the function to work. It is also dependent on the dplyr package.
- The modifications can be tested using the code in the comments at the bottom of the cb_add_col_attributes_from_csv_file.R file

```
# # Test
# # Set column attributes
# test_study <- cb_add_column_attributes_from_csv_file(df = study, attr_csv_path = here::here("data", "test_attributes.csv"))
#
# # Generate codebook
# test_codebook <- codebook(
#   df = test_study,
#   title = "Test study",
#   description = "Testing! Testing"
# )

# print(test_codebook, "test.docx")
```

36. Ebie has no helper functions for her code, so she fills all fields with "N/A". She could choose to document her main function here, but decides against it. If she were going to document a helper function, she would be sure to include the required items listed on the form.



Helper Functions

If you made any helper functions, include notes on them here. Comments should be brief and specific to their use in your data file(s). **The actual helper function code should have adequate comments and documentation to allow other people to use it without needing to ask the author clarifying questions.**

36

N/A

Input(s)	Output(s)
N/A	N/A

How is this function used in the code being reviewed?

N/A

37. Ebie is confident in her code. It has a straightforward purpose, and she has made sure to include helpful comments and annotations in the code itself. She primarily wants feedback on if her code is acceptable to merge into the main repository, which is not a line-by-line specific request. As such, she fills all fields in the Feedback Requests section with "N/A".

Feedback Requests

Requesters should outline any specific feedback requests. This allows the reviewer to pay closer attention to these sections to provide tailored feedback. **Be specific and reasonable in requests.** Add as many rows as necessary. If you have no requests, fill the first row with "N/A". 37

Line(s)	What do these lines of code achieve?	Requests
N/A	• N/A	• N/A
	•	•
	•	•

Ebie reviews the checklist. She has completed the first portion of the checklist! She adds a comment to the Test File row to mention the commented-out code in the main code. This is not required, but it is helpful to a reviewer.

Requester Provided Information

Requester Process Checklist

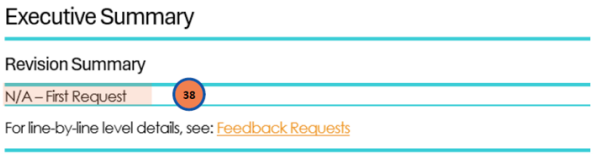
	Item	Comments
☒	**Enter values for the file to review, requester name, date of request, review number, revision number, and repository link**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Clearly outline goals of the code files**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Create brief title from goal**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Add Reviewer** ("xx" for initials if unknown)	Morrigan accepted the role of reviewer in weekly meeting on 6/10/2024
☒	**Verify header and footer are complete**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Name file and save**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Add Version & Dependency Information**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Provide Input Data link and description**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Provide Output Data link and description**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Provide Test File link and description**	A test function is included in the main file. It's commented out.
☒	**Outline Task Process Notes.**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Provide details of Helper Functions**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**List Feedback Requests**	This is a free text field that allows formatting, such as bullets. Click to enter text.

19.0.1.4 Finalizing and Sending the Form for Review

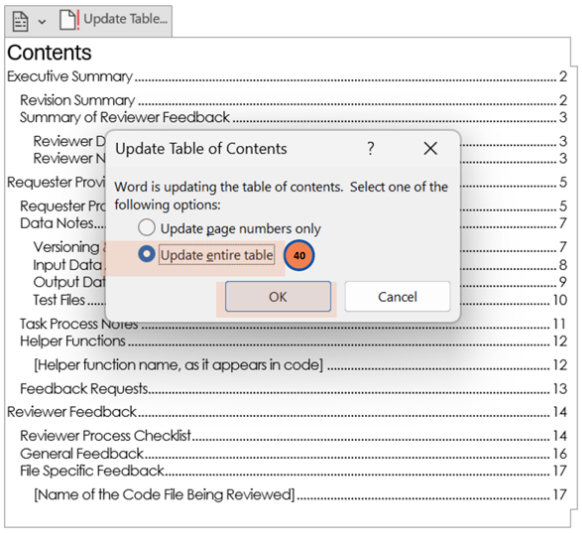
The remaining items on her checklist are finishing touches to her documentation.

	Item	Comments
☐	**Complete Revision Summary in Executive Summary** ("NA – First Request" for first revision)	38
☐	**Check documentation is complete**	39
☐	**Refresh Table of Contents (Update All)**	40
☐	**Send to Reviewer(s)**	41 42

38. As this is the first time Ebie is requesting this code be reviewed, she does not have any revision changes to note in the Executive Summary section. She follows the directions on the form, and completes this section with "NA – First Request"

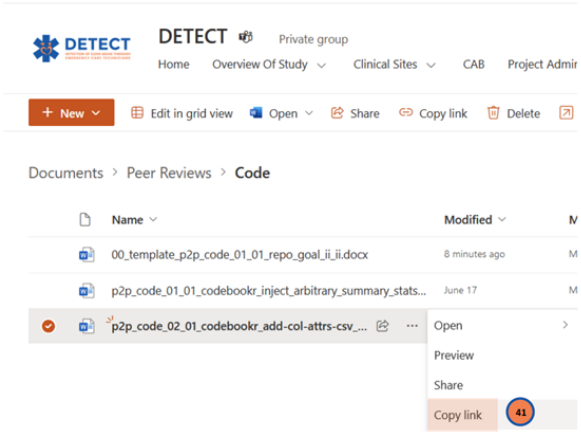


39. Ebie reviews all of her documentation one more time. She does not find anything else to add.
40. She updates the Table of Contents, making sure to select "Update entire table". She also finalizes her checklist before saving her work.

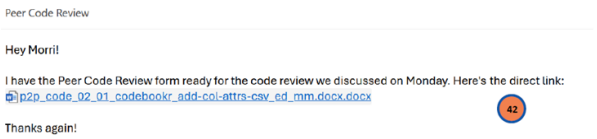


	Item	Comments
☒	**Complete Revision Summary in Executive Summary** ("NA – First Request" for first revision)	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Check documentation is complete**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Refresh Table of Contents (Update All)**	This is a free text field that allows formatting, such as bullets. Click to enter text.
☒	**Send to Reviewer(s)**	This is a free text field that allows formatting, such as bullets. Click to enter text.

41. Ebie then copies the link to the Internal Peer Code Review Form on SharePoint



42. She writes an email to her reviewer, Morrigan Mahady, using her UTHealth email. She includes the link to the Internal Peer Code Review Form on SharePoint in the body of the message. Now, Ebie moves to other work tasks while she waits for Morrigan's review.

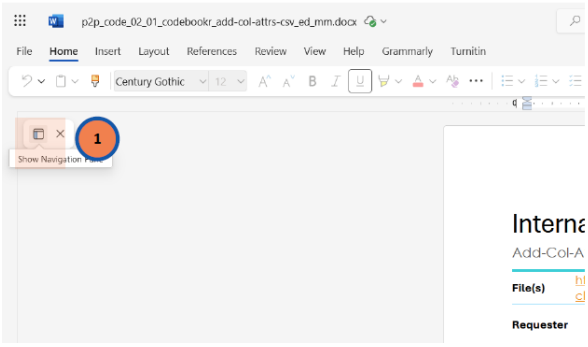


19.0.2 Reviewer

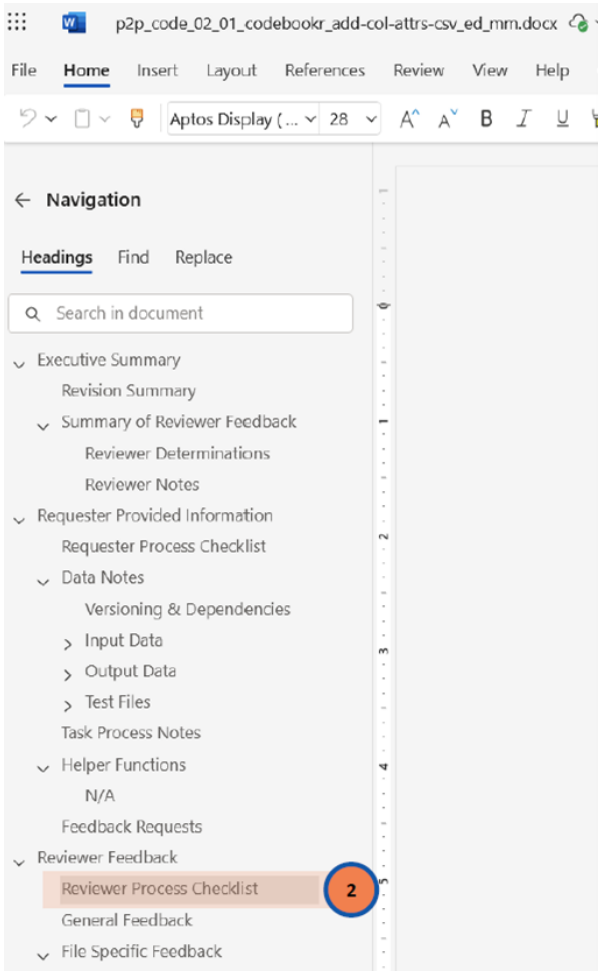
Morri's process as the reviewer is broken down into # steps, with screenshots or figures to provide additional clarity for the step-by-step instructions.

19.0.2.1 Verifying Administrative Details

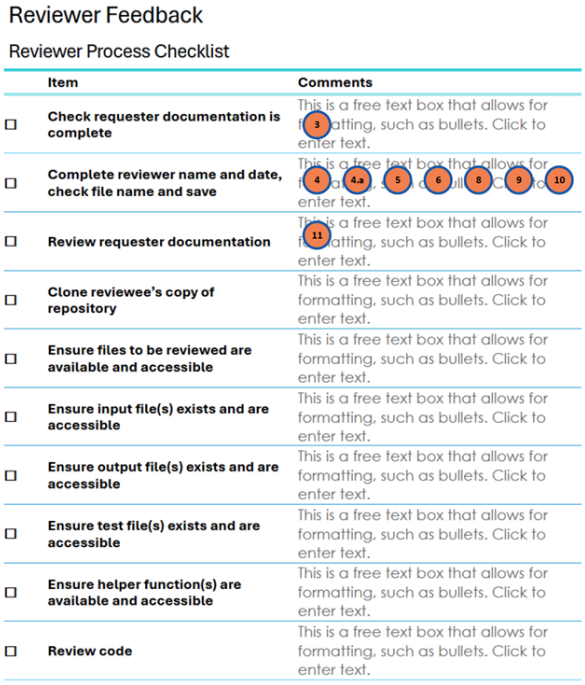
After receiving the notification email from Ebie, Morri opens the file using the direct link. This opens the file in the web browser version of Microsoft Word 365. 1. Morri opens the navigation pane.



2. Morri clicks on "Reviewer Process Checklist" in the navigation pane. This quickly jumps the screen to the Reviewer Process Checklist on page 14. They see that the first few items require them to check the Internal Peer Code Review Form itself.



3. Morri looks over the Internal Peer Review Form, and sees all expected sections exist. They carefully review the Requester Provided Information section, and the Revision Summary sub-section of the Executive Summary section. They verify that no field is left entirely blank. Since Ebie followed the instructions and put "N/A" in fields that were intentionally blank, this is easy for Morri to do.

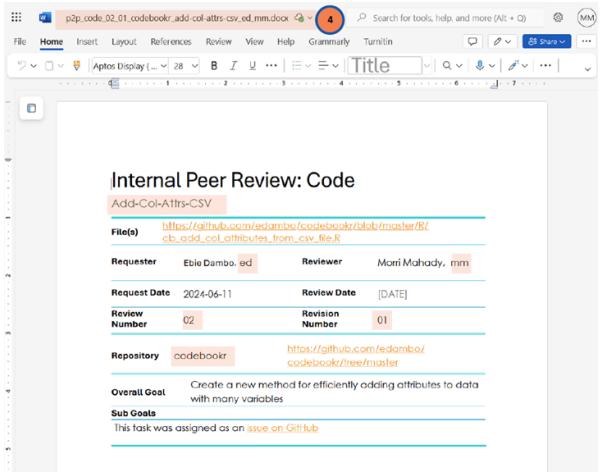


Reviewer Feedback

Reviewer Process Checklist

	Item	Comments
☐	**Check requester documentation is complete**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Complete reviewer name and date, check file name and save**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Review requester documentation**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Clone reviewee's copy of repository**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Ensure files to be reviewed are available and accessible**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Ensure input file(s) exists and are accessible**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Ensure output file(s) exists and are accessible**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Ensure test file(s) exists and are accessible**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Ensure helper function(s) are available and accessible**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Review code**	This is a free text box that allows for formatting, such as bullets. Click to enter text.

4. Morri checks the file name in the top ribbon of the web browser version of Microsoft Word 365. She verifies that it matches the information in the fields of the Cover of the document, and that it meets the File Name Protocol.

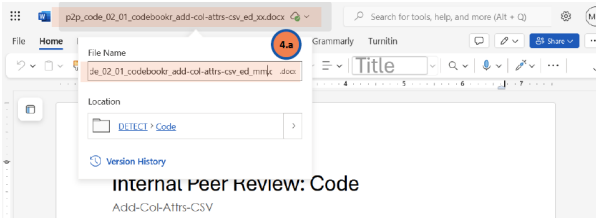


Internal Peer Review: Code

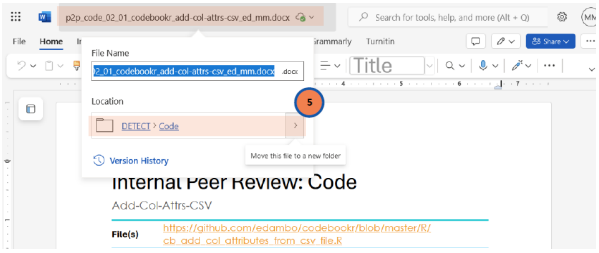
Add-Col-Attrs-CSV

File(s)	https://github.com/edambo/codebookr/blob/master/R/cb_add_col_attributes_from_csv_file.R		
Requester	Ebie Dambo, ed	**Reviewer**	Morri Mahady, mm
Request Date	2024-06-11	**Review Date**	[DATE]
Review Number	02	**Revision Number**	01
Repository	codebookr	https://github.com/edambo/codebookr/tree/master	
Overall Goal	Create a new method for efficiently adding attributes to data with many variables		
Sub Goals			
This task was assigned as an issue on GitHub			

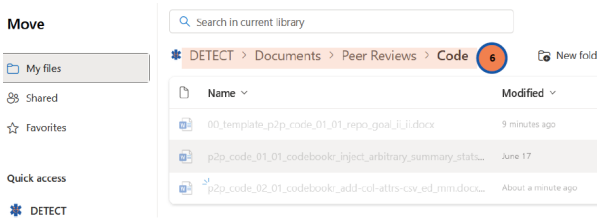
If Ebie (the requester) did not know the reviewer, the "reviewer initials" portion of the file name would be "xx". Morri would need to correct this. This may be quickly done in the web browser version of Microsoft Word 365 by clicking into the file name and correcting the value.



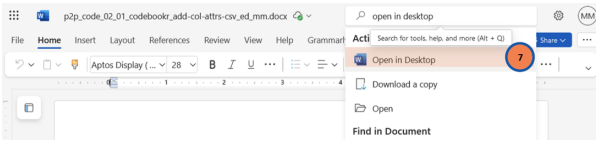
5. Morri checks that the file is saved in the proper location for this project. They click to view the file's location.



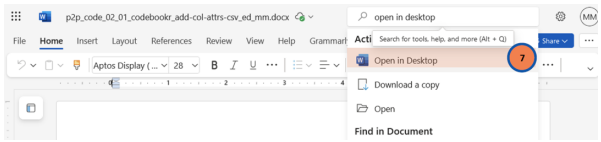
6. Morri reviews the file's location, and confirms it is the proper place for Internal Peer Code Review Forms for this project.



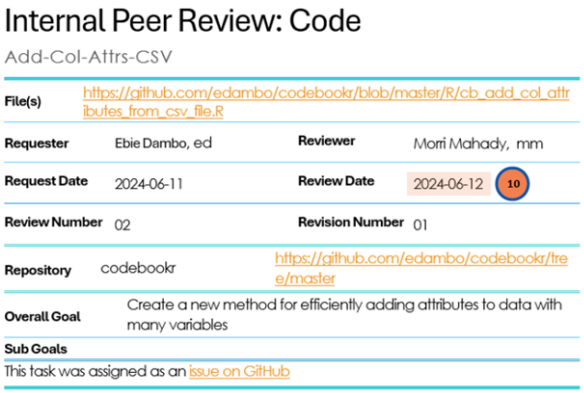
7. Morri then goes to open the file in the Desktop App Version of Microsoft Word 365. This may be found using the search tool, and selecting the appropriate option once it appears.



8. Morri checks the "Reviewer Name" in the Cover, ensuring it is complete and correct. If Ebie (the requester) did not have a reviewer before sending the form, Morri would fill out this section now. Since Ebie (the requester) completed this section, Morri confirms it is correct.
9. Morri checks the "Reviewer Initials" field in the Cover, ensuring it is complete and correct. If Ebie (the requester) did not have a reviewer before sending the form, Morri would fill out this section now. Since Ebie (the requester) completed this section, Morri confirms it is correct.



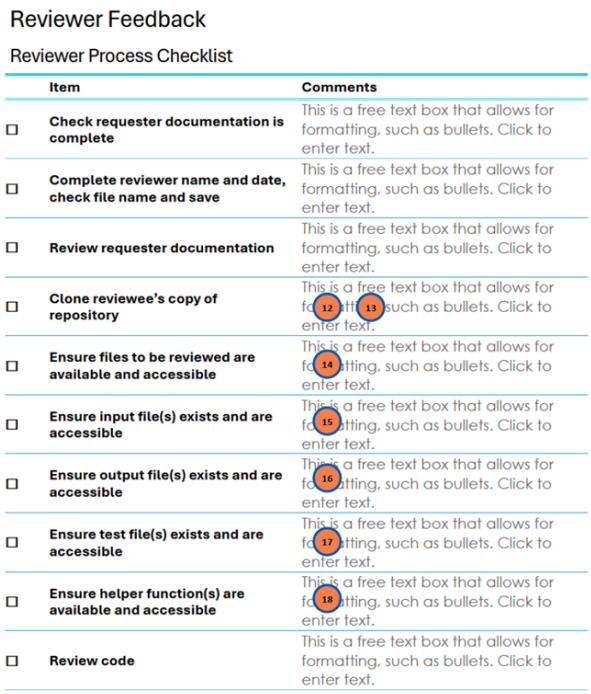
10. Morri adds the date of their review to the "Review Date" field of the Cover, using the date-picker. Morri then checks that all fields of the Cover are complete.



11. Morri now reads all the information that Ebie (the requester) included in their form. This makes sure Morri is familiar with the task, and what Morri should expect to find as they review the code.

19.0.2.2 Verifying Access to Files

The next several steps on the Reviewer Process Checklist require the reviewer to check that they can access all listed files.

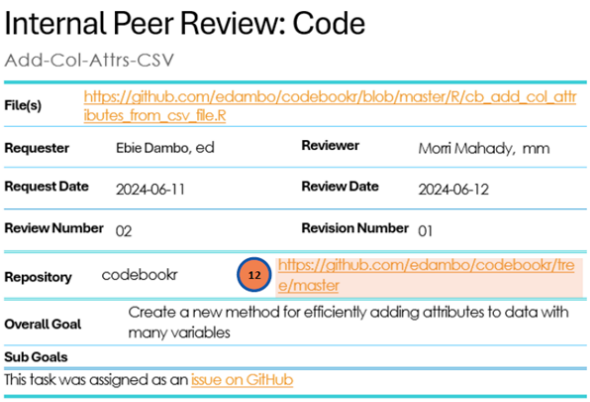


Reviewer Feedback

Reviewer Process Checklist

	Item	Comments
☐	**Check requester documentation is complete**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Complete reviewer name and date, check file name and save**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Review requester documentation**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Clone reviewee's copy of repository**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Ensure files to be reviewed are available and accessible**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Ensure input file(s) exists and are accessible**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Ensure output file(s) exists and are accessible**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Ensure test file(s) exists and are accessible**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Ensure helper function(s) are available and accessible**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Review code**	This is a free text box that allows for formatting, such as bullets. Click to enter text.

12. Morri first navigates to the requester's copy of the repository, which is provided on the Cover.

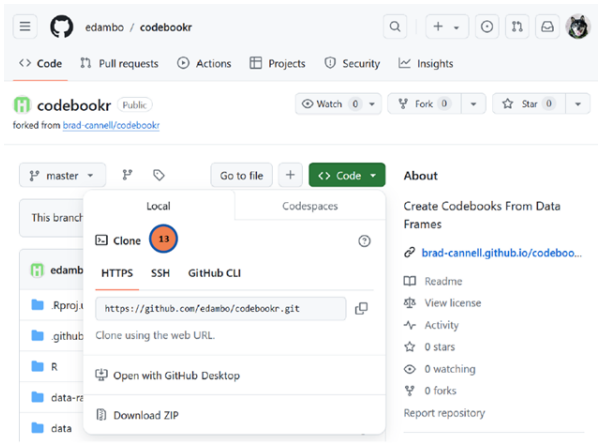


Internal Peer Review: Code

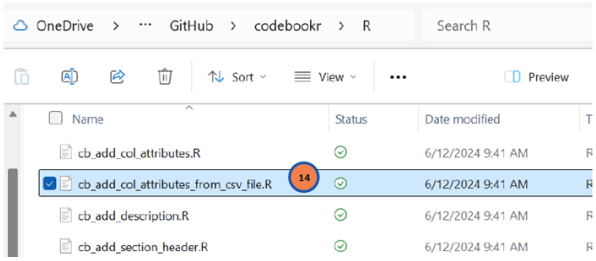
Add-Col-Attrs-CSV

File(s)	https://github.com/edambo/codebookr/blob/master/R/cb_add_col_attributes_from_csv_file.R		
Requester	Ebie Dambo, ed	**Reviewer**	Morri Mahady, mm
Request Date	2024-06-11	**Review Date**	2024-06-12
Review Number	02	**Revision Number**	01
Repository	codebookr		https://github.com/edambo/codebookr/tree/master
Overall Goal	Create a new method for efficiently adding attributes to data with many variables		
Sub Goals			
This task was assigned as an issue on GitHub			

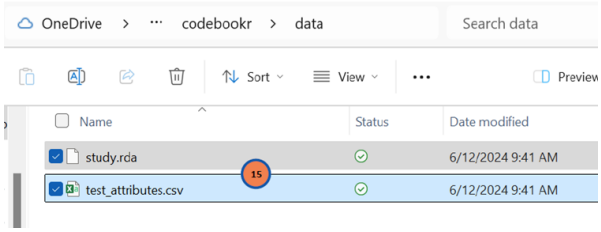
13. Morri clones this repository. We clone the entire repository because code within a repository may be highly interconnected and behave unexpectedly outside of this context. The exact means of cloning the repository are outside of the scope of this SOP.



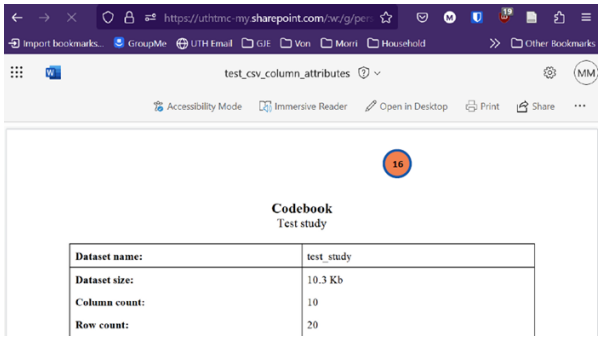
14. Morri then verifies that the code file in question is included in the copy of the repository. They use the link to the repository provided on the Cover to find the correct file. While not shown here, Morri should open the file to ensure it is not corrupted or otherwise inaccessible.



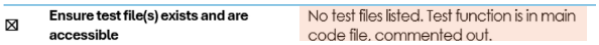
15. Morri then checks the input files. If they were not part of the repository, Morri would copy them into the appropriate folder of the cloned repository, which is almost always the "data" folder. While not shown here, Morri should open the files to ensure they are not corrupted or otherwise inaccessible.



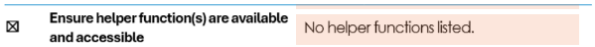
16. Morri then checks the output files. Morri would download this file, if necessary. While not shown here, Morri should open the file to ensure it is not corrupted or otherwise inaccessible.



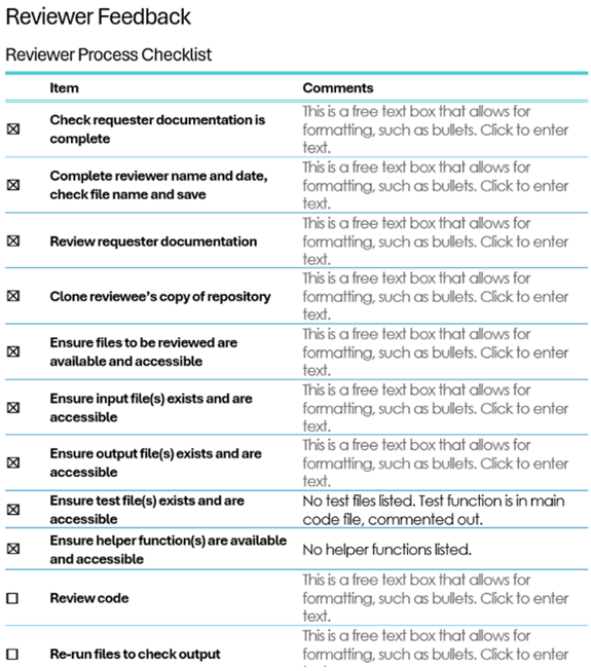
17. There is no test file in this example. Where one exists, the reviewer will verify it exists and is accessible, similar to Steps 14 and 15. Morri documents this in the comment field of the checklist.



18. There are no helper functions in this example. Where one exists, the reviewer will verify it exists and is accessible, similar to Steps 14, 15, and 16. Morri documents this in the comment field of the checklist.



At this point, Morri verifies their progress using the Reviewer Process Checklist.

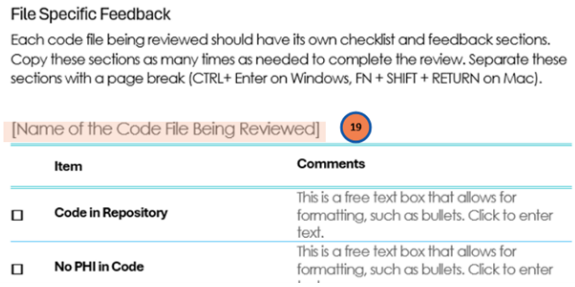


Reviewer Feedback

Reviewer Process Checklist

	Item	Comments
☒	Check requester documentation is complete	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Complete reviewer name and date, check file name and save	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Review requester documentation	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Clone reviewee's copy of repository	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Ensure files to be reviewed are available and accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Ensure input file(s) exists and are accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Ensure output file(s) exists and are accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Ensure test file(s) exists and are accessible	No test files listed. Test function is in main code file, commented out.
☒	Ensure helper function(s) are available and accessible	No helper functions listed.
☐	Review code	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Re-run files to check output	This is a free text box that allows for formatting, such as bullets. Click to enter

19. The next steps will involve reviewing the actual code. To prepare the Internal Peer Code Review Form, Morri ensures that there is a copy of the File Specific Feedback section for all code files. They also complete the "Name of the Code File Being Reviewed" for all copies.



File Specific Feedback

Each code file being reviewed should have its own checklist and feedback sections. Copy these sections as many times as needed to complete the review. Separate these sections with a page break (CTRL+ Enter on Windows, FN + SHIFT + RETURN on Mac).

[Name of the Code File Being Reviewed] 19

	Item	Comments
☐	Code in Repository	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	No PHI in Code	This is a free text box that allows for formatting, such as bullets. Click to enter

19.0.2.3 Code Review

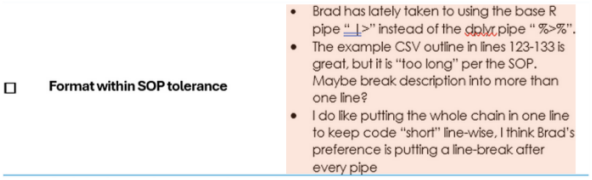
The next several steps, which perform the actual code review, may be repeated as many times as necessary to review all included code files. While every person has their own process, all reviewers are expected to review several specific things to set a minimum standard. The Checklist at the beginning of the File Specific Feedback section allows the reviewer to record notes at each step.

cb_add_col_attribtues_from_csv_file.R

	Item	Comments
☐	Code in Repository	This is a free text box that allows for 20 atting, such as bullets. Click to enter text.
☐	No PHI in Code	This is a free text box that allows for 21 atting, such as bullets. Click to enter text.
☐	Format within SOP tolerance	This is a free text box that allows for 22 atting, such as bullets. Click to enter text.
☐	Code is clear and easy to understand with useful comments and other annotations	This is a free text box that allows for 23 atting, such as bullets. Click to enter text.
☐	Input was available and processed as expected	This is a free text box that allows for 24 atting, such as bullets. Click to enter text.
☐	Output was available and as expected	This is a free text box that allows for 25 atting, such as bullets. Click to enter text.
☐	Code was able to replicate (within reason) the expected output when run by reviewer	This is a free text box that allows for 26 atting, such as bullets. Click to enter text.
☐	Meets primary goal?	This is a free text box that allows for 27 atting, such as bullets. Click to enter text.
☐	Meets subgoals?	This is a free text box that allows for 28 atting, such as bullets. Click to enter text.

20. Morri has already checked that the code file exists in Step 14. Morri opens the file and performs a more thorough check to ensure that this appears to be the correct file. Morri has no issues opening or viewing the code file.

21. While reviewing the code, Morri ensures that there is no PHI visible in the code. PHI may appear in comments, be used to specify a specific observation or value for a point-fix, and/or be displayed in code with tables or other output. Ensuring the code is free of PHI is important because all code on GitHub may be publicly viewed, and we must comply with HIPAA privacy regulations in our research. Morri does not find any PHI in the code file.
22. Morri checks that the code fits with the SOPs for code formatting, including the Style Guide or the R4Epi Best Practices. The most common area of non-adherence to these standards is line lengths in code. Morri identifies several lines which exceed SOP guidelines. They record some general comments in the checklist comment box.



☐	**Format within SOP tolerance**	• Brad has lately taken to using the base R pipe "|>" instead of the dplyr pipe "%>%". • The example CSV outline in lines 123-133 is great, but it is "too long" per the SOP. Maybe break description into more than one line? • I do like putting the whole chain in one line to keep code "short" line-wise, I think Brad's preference is putting a line-break after every pipe

23. Morri checks that the code has a logical organization, and that it is both clear and easy to understand. They pay particular attention to annotations such as comments or parameter definitions. They would compare documentation in helper functions to the description given by the requester. Morri believes Ebie (the requester) has done an exceptional job in making sure the code is clear and understandable, so they record a note in the comment box of the checklist.

☒	**Code is clear and easy to understand with useful comments and other annotations**	Amazing detail in the header descriptions of your code! A+

24. Morri has already checked that the input files were accessible in Step 24. At this stage, Morri checks that the file meets the description given by the requester, and that the code being reviewed is able to utilize this file without issue.
25. Morri has already checked that the output files were accessible in Step 25. At this stage, Morri checks that the file meets the description given by the requester, and that it would be reasonable to expect the code to produce this output file.
26. Morri then executes the code. If there is a test file or test function given by the requester, Morri could utilize it at this point. However, the reviewer should also attempt to independently test the code, where reasonable. Morri then compares the output created by the code to the output file given by the requester, to make sure they are either identical or reasonably similar, depending on the exact parameters of the code and task.
27. Morri then evaluates if the code provided meets the primary goal of the task.
28. Morri then evaluates if the code provided meets any subgoals outlined in the task. As there were no specific subgoals listed, Morri records a comment in the comment box of the checklist.

☒	**Meets subgoals?**	No specific subgoals listed.

29. Morri then organizes their feedback for the file, starting with overall comments and suggestions. For each comment or suggestion, Morri provides a rationale so that the requester can understand how the reviewer came to this opinion. Note that Morri provides both positive (praise) and negative (criticism) feedback. Reviewers are encouraged to keep their feedback balanced, pointing out both were the requester did well and where they can improve, to give a more complete picture of the requester's coding.

File-Specific Feedback

Include any file-specific feedback in this section. Add as many rows as necessary. **Be specific and reasonable in requests and include a clear rationale to justify your request.** Add as many rows as necessary. There should be some form of feedback, even if it is to declare "No changes needed" – **do not leave this section blank.**

Comments & Suggestions	Rationale
Amazing detail for within-code documentation	Clear parameter definitions and use case notes for all functions (29)
Need to adjust line length on some lines	Style SOPs specifies maximum line length
Consider switching "%>%" pipes to "|>" pipes	Brad's recent code exclusively uses the "|>" pipe, and he's mentioned his preference to me in conversation a few times.

30. Morri then creates line-by-line feedback, where they call out specific sections of the code. This is potentially the most beneficial feedback for the requester, as it is the most specific and direct. Reviewers should always make suggestions, where possible. Note how Morri utilized formatting features such as hyperlinks and images to include specific suggestions and clear feedback, so that the requester may view the source(s) influencing Morri's suggestions. This allows the requester to further develop their coding skills and apply feedback to future code tasks (not just this specific code task/file).

Line-by-Line Comments & Suggestions

Include any file-specific line-by-line feedback in this section. Add as many rows as necessary. **Be specific and reasonable in requests and include a clear rationale to justify your request.** Add as many rows as necessary. If there is no line-by-line feedback, fill the first row with "N/A".

Line(s)	Issue	Suggestion(s)	Rationale (30)
123-133	Lines are "too long" per SOP	• Maybe break up the "description" column into more than one line? • It's beautiful and super clear, so I hate to suggest anything that might harm that.	R4Epi references Wickam's *Advanced R*; line length of 80 characters max
152	Lines are "too long" per SOP	• Break up this comment into more than one line	R4Epi references Wickam's *Advanced R*; line length of 80 characters max

169	Lines are "too long" per SOP	• Line break after "eval(", with the closing ")" on a separate line as well
		`attr_list[[1]][["value_labels"]] <- eval(`
		`  parse(text = attr_list[[1]]["value_labels"])`
		`  )`
			R4Epi references Wickam's *Advanced R*; line length of 80 characters max

178	Lines are "too long" per SOP	• Put each parameter for the test execution on its own line
		`# test_study <- cb_add_column_attributes_from_csv_file(`
		`#   df = study,`
		`#   attr_csv_path = here::here("data", "test_attributes.csv")`
		`#   )`
			R4Epi references Wickam's *Advanced R*; line length of 80 characters max

31. Before finalizing this feedback, Morri also checks the Feedback Requests subsection of the Requester Provided Information section. If the requester has made any specific requests,

Morri would address them now, adding their feedback to the tables as appropriate. Since Ebie (the requester) has made no requests, Morri does not have any further requirements for specific feedback.

At this point, the checklist for the file-specific review is complete.

File Specific Feedback

Each code file being reviewed should have its own checklist and feedback sections. Copy these sections as many times as needed to complete the review. Separate these sections with a page break (CTRL+ Enter on Windows, FN + SHIFT + RETURN on Mac).

cb_add_col_attribtues_from_csv_file.R

	Item	Comments
☒	**Code in Repository**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**No PHI in Code**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	**Format within SOP tolerance**	• Brad has lately taken to using the base R pipe "|>" instead of the dplyr pipe "%>%". • The example CSV outline in lines 123-133 is great, but it is "too long" per the SOP. Maybe break description into more than one line? • I do like putting the whole chain in one line to keep code "short" line-wise, I think Brad's preference is putting a line-break after every pipe
☒	**Code is clear and easy to understand with useful comments and other annotations**	Amazing detail in the header descriptions of your code! A+
☒	**Input was available and processed as expected**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**Output was available and as expected**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**Code was able to replicate (within reason) the expected output when run by reviewer**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**Meets primary goal?**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**Meets subgoals?**	No specific subgoals listed.

19.0.2.4 Consolidating Feedback

Now that Morri has completed the review of the file, they are ready to finish their feedback. The next several items guide Morri through that process. Note that Morri has already checked off items that were completed during the code review process, such as "Re-run files to check output" and "Review feedback request and respond".

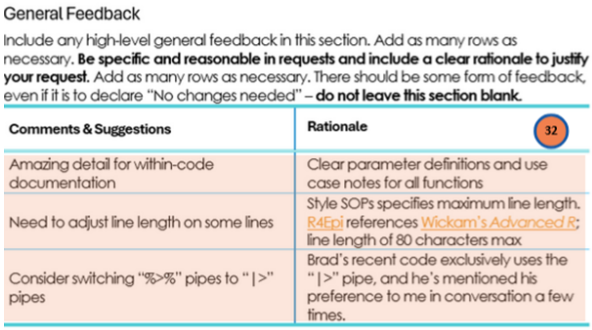
Reviewer Feedback

Reviewer Process Checklist

	Item	Comments
☒	Check requester documentation is complete	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Complete reviewer name and date, check file name and save	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Review requester documentation	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Clone reviewee's copy of repository	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Ensure files to be reviewed are available and accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Ensure input file(s) exists and are accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Ensure output file(s) exists and are accessible	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Ensure test file(s) exists and are accessible	No test files listed. Test function is in main code file, commented out.
☒	Ensure helper function(s) are available and accessible	No helper functions listed.
☒	Review code	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	Re-run files to check output	This is a free text box that allows for formatting, such as bullets. Click to enter text.

	Item	Comments
☒	Review feedback requests and respond	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Complete any line-by-line or general feedback	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Complete Summary of Reviewer Feedback in Executive Summary	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Make acceptance determination	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Make meeting determination	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Refresh Table of Contents (Update All)	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☐	Check documentation is complete and return to requester	This is a free text box that allows for formatting, such as bullets. Click to enter text.

32. Morri completes the General Feedback section. To do this, they review their file-specific feedback and notes made in the checklists. This section should consolidate feedback given to all code files. Since there is only one code file in this review, most of this section is identical to the feedback given for the file itself. Morri adds the links they had used in the rationale of the line-by-line feedback.

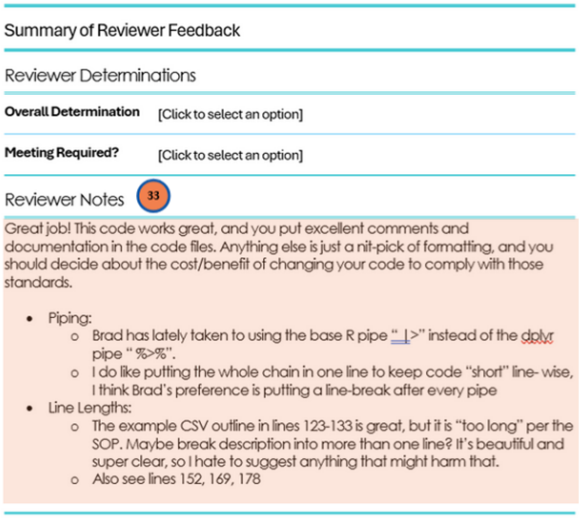


General Feedback

Include any high-level general feedback in this section. Add as many rows as necessary. **Be specific and reasonable in requests and include a clear rationale to justify your request.** Add as many rows as necessary. There should be some form of feedback, even if it is to declare "No changes needed" – **do not leave this section blank.**

Comments & Suggestions	Rationale 32
Amazing detail for within-code documentation	Clear parameter definitions and use case notes for all functions
Need to adjust line length on some lines	Style SOPs specifies maximum line length. R4Epi references Wickam's *Advanced R*; line length of 80 characters max
Consider switching "%>%" pipes to "|>" pipes	Brad's recent code exclusively uses the "|>" pipe, and he's mentioned his preference to me in conversation a few times.

33. Morri then completes the high-level summary of feedback in the Executive Summary. Note how Morri gives a brief summary of their opinions, what they feel the requester should do in response to this feedback, and the highest-level summary of their feedback. Note how Morri references some line-by-line feedback but does not repeat the details in this section. If Morri had any remaining questions or concerns, they would be listed in this section.



Summary of Reviewer Feedback

Reviewer Determinations

Overall Determination [Click to select an option]

Meeting Required? [Click to select an option]

Reviewer Notes 33

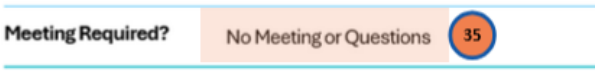
Great job! This code works great, and you put excellent comments and documentation in the code files. Anything else is just a nit-pick of formatting, and you should decide about the cost/benefit of changing your code to comply with those standards.

- Piping:
  - Brad has lately taken to using the base R pipe "|>" instead of the dplyr pipe "%>%".
  - I do like putting the whole chain in one line to keep code "short" line- wise, I think Brad's preference is putting a line-break after every pipe
- Line Lengths:
  - The example CSV outline in lines 123-133 is great, but it is "too long" per the SOP. Maybe break description into more than one line? It's beautiful and super clear, so I hate to suggest anything that might harm that.
  - Also see lines 152, 169, 178

34. Morri selects the option that best corresponds to their "Overall Determination". Morri feels that this code meets the task goals and only has minor formatting issues (which do not impact its function). As such, Morri selects "Accept (as-is, no revision)". Further details on these options are in the Summary of Reviewer Feedback subsection of the Anatomy of the Internal Peer Code Review Form section.

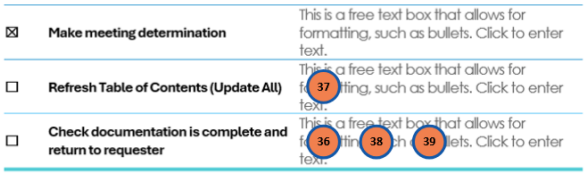
Overall Determination Accept (as is, no revision) 34

35. Morri selects the option for "Meeting Required?" that best corresponds to their opinion on whether a meeting is required to complete the review. As Morri did not have any remaining questions and thus do not want a meeting, they select "No Meeting or Questions". If Morri had to reach out for questions, required a meeting, or had already held a meeting with Ebie (the requester), this would be reflected here. Further details on these options are in the Summary of Reviewer Feedback subsection of the Anatomy of the Internal Peer Code Review Form section.

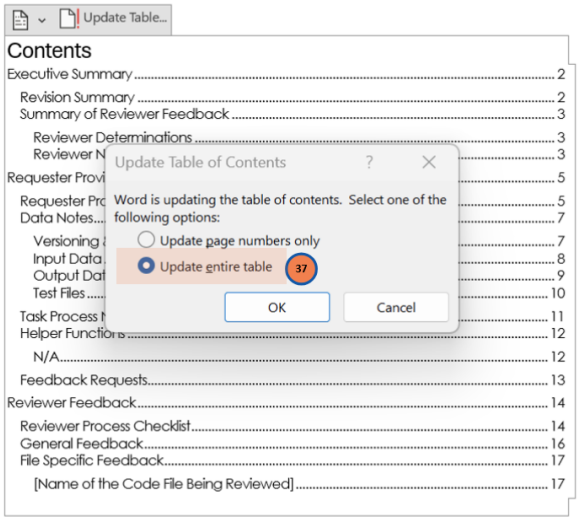


19.0.2.5 Finalization

The final steps require Morri to finalize the document and return it to the requester.



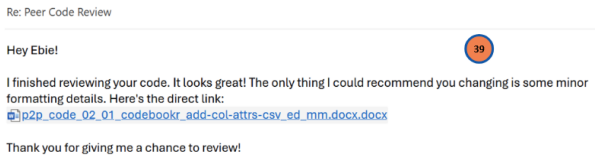
36. Morri checks over their documentation to ensure it is complete. They do not identify any gaps, and thus consider it complete.
37. Morri updates the Table of Contents, taking care to select the "Update all fields" option.



38. After a last check that the document is complete, Morri finalizes the checklists and ensures the document is saved.

	Item	Comments
☒	**Review feedback requests and respond**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**Complete any line-by-line or general feedback**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**Complete Summary of Reviewer Feedback in Executive Summary**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**Make acceptance determination**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**Make meeting determination**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**Refresh Table of Contents (Update All)**	This is a free text box that allows for formatting, such as bullets. Click to enter text.
☒	**Check documentation is complete and return to requester**	This is a free text box that allows for formatting, such as bullets. Click to enter text.

39. Morri replies to the email sent by the requester, Ebie, using their UTHealth email. They include the direct link to the Internal Peer Code Review Form on SharePoint in the body of the message. If Morri had to move the file for SOP compliance, they should obtain the updated link. This review cycle is complete.



This concludes this revision cycle for this review, including the associated Internal Peer Code Review Form.

19.0.3 Notes on Additional Revisions or Disputes

If major revisions or other changes are made, the requester may request another review. This requires a separate Internal Peer Code Review Document. It should have the same review number and iterate the revision number by 1. The reviewer may be the same person, or a different person. The prior Internal Peer Code Review Form may be used as a reference, with updates made in the new form to reflect changes.

If the requester disagrees with feedback given by the reviewer, the requester should note this disagreement in the Revision Summary of the next cycle's form. If the requester disagrees with making any changes (which would preclude a further review and revision cycle), the requester

can respond in writing to the reviewer's email to outline the reasons for their disagreement. These notes outlining the disagreement should be done in a polite, professional manner and be as objective as possible. Significant disagreements that cannot be resolved through open dialogue should be brought to project leadership for guidance and resolution.