## Internal Code QA

Internal code QA (our internal code review process) is about ensuring the code we write for analysis is **reliable**, **reproducible**, and **easy for others to understand**. A systematic code review process will reduce the risk of mistakes and improve clarity in our work. It also enables us to share reusable code, which in turn increases the reliability and reputation of e61’s work.

We recognise that too much process can create compliance costs (e.g. extra time spent on QA), so our approach focuses on effective **minimum standards** that keep the burden low while maintaining rigor.

A screenshot of a checklist

AI-generated content may be incorrect.

## Code QA motivation and explanation

Here we outline an internal code QA approach that emphasises collaboration and learning without heavy bureaucracy.

Internal code QA can be split into two related stages:

* **The creation process** of writing and reviewing code iteratively during development, including sharing code with teammates and embedding checks as you go.
* **The clearance process** associated with final code review and checks before a project’s outputs are finalised and released.

Below we outline the structure of the code QA process in these two stages, so that teams can include these steps in their planning and workflow.

**Creation Process**

During the creation phase of a project, integrate QA into your coding workflow from the start. This means writing code that others can easily read and ***sharing your work early and often***. This includes **placing all project code in a folder or repository that is accessible to the entire team** (not just on your local machine).

Code collaboration is also highly encouraged – you might consider pair programming on key sections or asking a teammate to review a function you wrote. This kind of iterative feedback helps catch issues early and fosters a culture of collaboration.

It’s also good practice to incorporate **minimum viable checks** into your code as you develop it. These are lightweight sanity checks that ensure your analysis is on track without much extra effort. For example:

* **Basic data checks:** verify row counts or sample sizes at critical steps, check for NA values or outliers in your data, and confirm that distributions or summary statistics look reasonable – and comment the check to confirm it has been done. These quick checks can catch obvious problems (like dropped observations, errant joins that duplicate rows or miscoded variables) early in the process.
  + In R the **testthat** package and in STATA the **codebook** will allow you to undertake these checks as you go.
* **Code readability:** write code with clarity in mind – use meaningful variable and function names, and add brief comments to explain non-obvious steps. This way, anyone else reading the code (or you, months later) can follow the logic without confusion. Writing clear code from the start is a form of QA, as it reduces the chance of misinterpretation or error. Importantly, break up your workflow into **multiple scripts** that serve a specific purpose.
* **Use templates and reusable components:** where possible, start with existing project templates or standard scripts. Make use of the e61 Stack Overflow to share such approaches.
* **Peer check-ins:** take advantage of internal catch-ups to discuss your approach. For instance, in team meetings or informal chats, mention the methods you’re using or any issues you’re encountering.

By building these practices into the creation process, quality assurance becomes part of how we work rather than a separate hurdle. We keep this phase low-cost by focusing on essentials: **write clean code, include basic checks, and invite early feedback**. This avoids large reworks later and makes the eventual clearance step much smoother.

**Clearance Process**

The clearance process for code QA happens when you’re nearing the end of the project and preparing to release results. This stage is about doing a thorough check and getting a fresh pair of eyes on the code to ensure everything is correct and reproducible before any findings go out the door. It’s essentially a final safety net.

At a minimum, before finalising any research output, ensure the following **minimum code QA standards** are met:

* **Reproducibility:** Run the entire analysis from scratch in a clean environment to confirm that it works end-to-end without errors. This confirms that all required data, packages, and steps are properly accounted for. The code should ideally be runnable by someone else with access to the same data.
  + The degree of re-analysis should be determined at the start by the team. For DataLab projects such a clear re-rerun is likely to be overly burdensome, so a lower compliance option (i.e. workflow checks on a random sample) can be used instead.
* **Team accessibility:** Double-check that all code (and relevant data or outputs) are saved in the shared project folder or repository and organised logically. No one-off scripts should be sitting only on someone’s laptop. Ensuring the code is centrally available is important for transparency and future reference.
  + Due to the lack of a Git repository in DataLab, it is recommended that a locked version of scripts are saved in a *past projects* folder – to prevent accidental overwriting.
* **Output verification:** Cross-verify the key results. For every figure or table in your report/presentation, make sure it’s directly produced by the code and that the values match. This step catches any copy-paste mistakes or last-minute changes that weren’t reflected in the code.
* **Basic quality checks included:** Ensure that the code includes the basic QA checks discussed in the creation phase (counts, NA checks, etc.) – if you haven’t already, this is the time to insert them.
* **Documentation and clarity:** Verify that the code is sufficiently documented for someone else to follow.

In addition to these minimum standards, if it is a larger project and time allows, consider a **peer review** of the code at clearance. Ideally, have a team member who was not deeply involved in the project do a quick review or at least run through the code.

We also tailor the depth of code QA to the type of project. Not every project will get an equally intensive review – a simple exploratory analysis for an internal memo won’t need the same scrutiny as code for a major public research paper. However, **every** project should meet the basic checklist above as a baseline.