# Shiny Development Principles

1. Decide naming conventions, must be intuitive (lets stick with Shiny recommendations)
   1. R/histogram.R holds code for the module (functions)
   2. histogramUI() is the module UI, can have input or output added if helpful
   3. histogramServer() is the module server
   4. histogramApp() is the complete app for testing
2. Use functions – remove repeated bits of code
   1. Make a function package:
      1. read Engineering production-grade shiny apps <https://engineering-shiny.org>
      2. use golem package
   2. functional programming with purr::map / lapply()
   3. functions live outside server(), reactivity inside. If reactivity >10 lines, take out what you can into a function.
   4. Keep reactive and non reactive parts of the app separate, i.e. result of function should be passed to reactive.
   5. If absolutely necessary internal functions that need to use input, output or session, may live in server()
3. Make a package for the functions and / or data
   1. put all R code in R/ directory
   2. write a function to start app
   3. description file in root directory
   4. memorise is a package that can cache regular R functions
   5. data from package can be picked up in the outer server function of the module i.e. function(id, data). Then the App is not running the functions to make each data set every time. Means it’s not ‘live’ but we can keep updated.
   6. Maintenance of package data, updates etc. offline from App
4. Use modules – pages run independently and only on demand
   1. skeleton set up as our mock up version
5. Test plan – Automate testing to confirm behaviour of functions, interactions between reactives, end-to-end experience & load test. Make tests for:
   1. non-reactive functions <https://r-pkgs.org/tests.html>
   2. flow of reactivity
   3. parts that use JS
   4. app visuals (layout, CSS, plots)
   5. Consider github actions for continuous integration, runs tests on commit to make sure nothing has broken.
   6. Consider renv to isolate app from version changes over time
   7. Consider config to keep track of file locations
   8. Use git version control for the development phases
6. Hand craft error messages
7. Consider performance
   1. benchmark – shinyloadtest package
   2. profile – profvis package
   3. optimise – caching, remove data prep from app, psychology
      1. cache inputs, reactives (web API), plots
      2. move data out of server, consider alternatives to read\_csv
      3. tabs, start buttons etc.
   4. code review – we can do this at hackathons

# Shiny workflow tasks and responsible person - BOO

|  |  |  |
| --- | --- | --- |
| **Task** | **Person** | **Phase** |
| Develop naming conventions | follow Shiny | Initial |
| Data product access | CD | initial |
| Develop code and functionalise | CD (CPR, NRS) / JE (integration, spatial) | continual |
| Develop package of functions / data | JE | continual |
| Develop template for CSS or use packages to set CSS | CD | initial |
| Develop module structure | CD | initial |
| Develop testing protocols, error messaging | CD | secondary |
| Documentation | CD / JE | continual |
| Optimisation | JE | secondary |

# Shiny workflow tasks and responsible person - MOO

(as per BOO unless listed here)

|  |  |  |
| --- | --- | --- |
| **Task** | **Person** | **Phase** |
| Data product access | MO / MB | initial |
| Develop code and functionalise | CD / JE | continual |
| Develop package of functions | CD / JE | continual |