Reproducible Shiny apps with shinymeta

Carson Sievert
Software Engineer, RStudio
@cpsievert
Slides bit.ly/RPharma

Joint work with Joe Cheng
Shiny: **Interactive** webapps in R

- Easily turn your R code into an interactive GUI.
- Allow users to **quickly explore** different parameters, models/algorithms, other information.

```r
app.R

library(shiny)
library(plotly)

ui <- fluidPage(
  plotlyOutput("p"),
  selectInput(
    "x", "Choose a variable",
    choices = names(diamonds)
  )
)

server <- function(input, output) {
  output$p <- renderPlotly({
    plot_ly(x = diamonds[[input$x]])
  })
}

shinyApp(ui, server)
```
Interactivity is great, but **reproducibility suffers**

- Reproducing results is *possible* by replicating user events (or *bookmarking*), but results are locked behind a GUI

- Even if you can view the app’s source code, the domain logic is intertwined with Shiny code
  - Methodology is less transparent
  - Harder to verify results are ‘correct’
The goal: interactivity + reproducible code

1. Find interesting results via interactive app
2. Export domain logic, on demand
   - As reproducible code/results that are independent of Shiny app
**shinymeta**: tools for capturing logic in a Shiny app and exposing it as code that can be run outside of Shiny.

*Not yet on CRAN, but can install with:*

```r
devtools::install_github("rstudio/shinymeta")
```
Example: basic Shiny app

Package name

ggplot2

![Timeline graph with count on the y-axis and months of 2018 and 2019 on the x-axis, showing fluctuations over time.]
library(shiny)
library(tidyverse)

ui <- fluidPage(
  textInput("package", "Package name", value = "ggplot2"),
  plotOutput("plot")
)

server <- function(input, output, session) {

  downloads <- reactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- reactive({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))
    downloads() %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend")
  })

  output$plot <- renderPlot({
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })

  shinyApp(ui, server)
The goal: reproducible plot code

```r
library(tidyverse)
downloads <- cranlogs::cran_downloads("ggplot2", from = Sys.Date() - 365, to = Sys.Date())
downloads_rolling <- downloads %>%
  mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
ggplot(downloads_rolling, aes(date, count)) + geom_line()
```
library(shiny)
library(tidyverse)

ui <- fluidPage(
  textInput("package", "Package name", value = "ggplot2"),
  plotOutput("plot")
)

server <- function(input, output, session) {

  downloads <- reactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- reactive({
    validate(need(sum(downloads())$count) > 0, "Input a valid package name")
    downloads() %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend")
  })

  output$plot <- renderPlot({
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })

  shinyApp(ui, server)
server <- function(input, output, session) {

  downloads <- reactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- reactive({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    downloads() %>%
      mutate(count = zoo::rollapply(count, 7, mean, fill = "extend")
  })

  output$plot <- renderPlot({
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
server <- function(input, output, session) {

  downloads <- reactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- reactive({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    downloads() %>%
      mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
  })

  output$plot <- renderPlot({
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
server <- function(input, output, session) {

  downloads <- reactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- reactive({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))
    downloads() %>%
      mutate(count = zoo::rollapply(count, 7, mean, fill = "extend")
  })

  output$plot <- renderPlot({
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
Step 1: Capture domain logic

```r
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))
    metaExpr({
      downloads() %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend")
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
```
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))
    metaExpr({
      downloads() %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}

Step 1: Capture domain logic

reactive becomes metaReactive

render functions must be wrapped in metaRender
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))
    metaExpr({
      downloads() %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
Step 2: Identify reactive reads

```r
server <- function(input, output, session) {
  downloads <- metaReactive(
    cranlogs::cran_downloads(
      input$package,
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  )

  downloads_rolling <- metaReactive2(
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))
    metaExpr(
      downloads() %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    )
  )

  output$plot <- metaRender(renderPlot, {
    ggplot(downloads_rolling(), aes(date, count)) + geom_line()
  })
}
```
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      ..(input$package),
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))

    metaExpr({
      ..(downloads()) %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
  })
}

Step 2: Mark reactive reads
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      ..(input$package),
      from = Sys.Date() - 365,
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))
    metaExpr({
      ..(downloads()) %>%
      mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(.(downloads_rolling()), aes(date, count)) + geom_line()
  })
}
Step 2: Mark reactive reads

server <- function(input, output, session) {

    downloads <- metaReactive({
        cranlogs::cran_downloads(
            ..(input$package),
            from = Sys.Date() - 365,
            to = Sys.Date()
        )
    })

    downloads_rolling <- metaReactive2({
        validate(need(sum(downloads()$count) > 0, "Input a valid package name"))
        metaExpr({
            ..(downloads()) %>%
            mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
        })
    })

    output$plot <- metaRender(renderPlot, {
        ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
    })
}
server <- function(input, output, session) {

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      ..(input$package),
      from = ..(format(Sys.Date() - 365)),
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))
    metaExpr({
      ..(downloads()) %>%
        mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
  })
}
server <- function(input, output, session) {

  output$code <- renderPrint({
    expandChain(output$plot())
  })

  downloads <- metaReactive({
    cranlogs::cran_downloads(
      ..(input$package),
      from = ..(format(Sys.Date() - 365)),
      to = Sys.Date()
    )
  })

  downloads_rolling <- metaReactive2({
    validate(need(sum(downloads()$count) > 0, "Input a valid package name"))
    metaExpr({
      ..(downloads()) %>%
      mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
    })
  })

  output$plot <- metaRender(renderPlot, {
    ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
  })
}
Step 3: Generate code with `expandChain()`

```r
> expandChain(output$plot())

downloads <-
cranlogs::cran_downloads(
  ..(input$package),
  from = ..(format(Sys.Date() - 365)),
  to = Sys.Date()
)

downloads_rolling <-
  ..(downloads()) %>%
  mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
```
> `expandChain(output$plot())`

downloads <-
  cranlogs::cran_downloads(
    ..(input$package),
    from = ..(format(Sys.Date() - 365)),
    to = Sys.Date()
  )

downloads_rolling <-
  ..(downloads()) %>%
  mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(..(downloads_rolling()), aes(date, count)) + geom_line()
Step 3: Generate code with expandChain()

```r
> expandChain(output$plot())

downloads <-
cranlogs::cran_downloads(
  "shiny",
  from = ..(format(Sys.Date() - 365)),
  to = Sys.Date()
)

downloads_rolling <-
downloads %>%
  mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(downloads_rolling, aes(date, count)) + geom_line()
```
Step 3: Generate code with expandChain()

```r
> expandChain(output$plot())

downloads <-
cranlogs::cran_downloads(
  "shiny",
  from = ..((format(Sys.Date() - 365)),
  to = Sys.Date()
)

downloads_rolling <-
downloads %>%
  mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
ggplot(downloads_rolling, aes(date, count)) + geom_line()
```

Marked reactive reads are replaced with a suitable value or name.
Step 3: Generate code with expandChain()

```r
> expandChain(output$plot())

downloads <-
  cranlogs::cran_downloads(
    "shiny",
    from = ..(format(Sys.Date() - 365)),
    to = Sys.Date()
  )

downloads_rolling <-
  downloads %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(downloads_rolling, aes(date, count)) + geom_line()
```

Other code wrapped in ..() is evaluated (i.e. unquoted)
Step 3: Generate code with expandChain()

```r
> expandChain(output$plot())

downloads <-
  cranlogs::cran_downloads(
    "shiny",
    from = "2019-08-01",
    to = Sys.Date()
  )

downloads_rolling <-
  downloads %>%
    mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(downloads_rolling, aes(date, count)) + geom_line()
```

This allows dynamic results to be ‘hard coded’
Step 3: Generate code with expandChain()

```r
> expandChain(quote(library(tidyverse)), output$plot())

library(tidyverse)

downloads <-
cranlogs::cran_downloads(
  "shiny",
  from = "2019-08-01",
  to = Sys.Date()
)

downloads_rolling <-
downloads %>%
  mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))

ggplot(downloads_rolling, aes(date, count)) + geom_line()
```
Huzzah!

**Package name**

`ggplot2`

```r
library(tidyverse)
downloads <- cranlogs::cran_downloads("ggplot2", from = Sys.Date() - 365, to = Sys.Date())
downloads_rolling <- downloads %>%
  mutate(count = zoo::rollapply(count, 7, mean, fill = "extend"))
ggplot(downloads_rolling, aes(date, count)) + geom_line()
```
Not the best user experience :/
Better ways to distribute code (& results)

On Button click:

1. Display code with `displayCodeModal()`

2. Generate zip bundle with code (e.g., R/Rmd), supporting files (e.g., csv, rds, etc), and results (e.g., pdf, html, etc)

Learn about these approaches at https://rstudio.github.io/shinymeta/articles/code-distribution.html
outputCodeButton() + displayCodeModal()
downloadButton() + buildRmdBundle()
Inspiration: ANOVA app

The Shiny app: https://testing-apps.shinyapps.io/diy_anova/
In summary

- Many benefits to having an interactive GUI generate reproducible code (transparency, permanence, automation)

- **shinymeta**: new R package for capturing logic in a Shiny app and exposing it as code that can be run outside of Shiny

- Add **shinymeta** integration to a Shiny app by:
  1. Identify and capture domain logic
  2. Mark reactive reads with ..()
  3. Export domain logic with expandChain()
Acknowledgments

Many people have provided motivation, inspiration, and ideas that have lead to *shinyMETA*. Special thanks to:

- Adrian Waddell for inspiring the over-arching metaprogramming approach
- Doug Kelkhoff for his work in *scriptgloss*
Thank you! Questions?

https://rstudio.github.io/shinymeta/


@cpsievert

cpsievert1@gmail.com

http://cpsievert.me