The R Test Kitchen

Cooking Up Quality in Clinical Analytics and Reporting

Phuse USA 2025
SM11
Emily Yates

Data volume is rapidly increasing...

Genomics Data

Wearables

ePRO

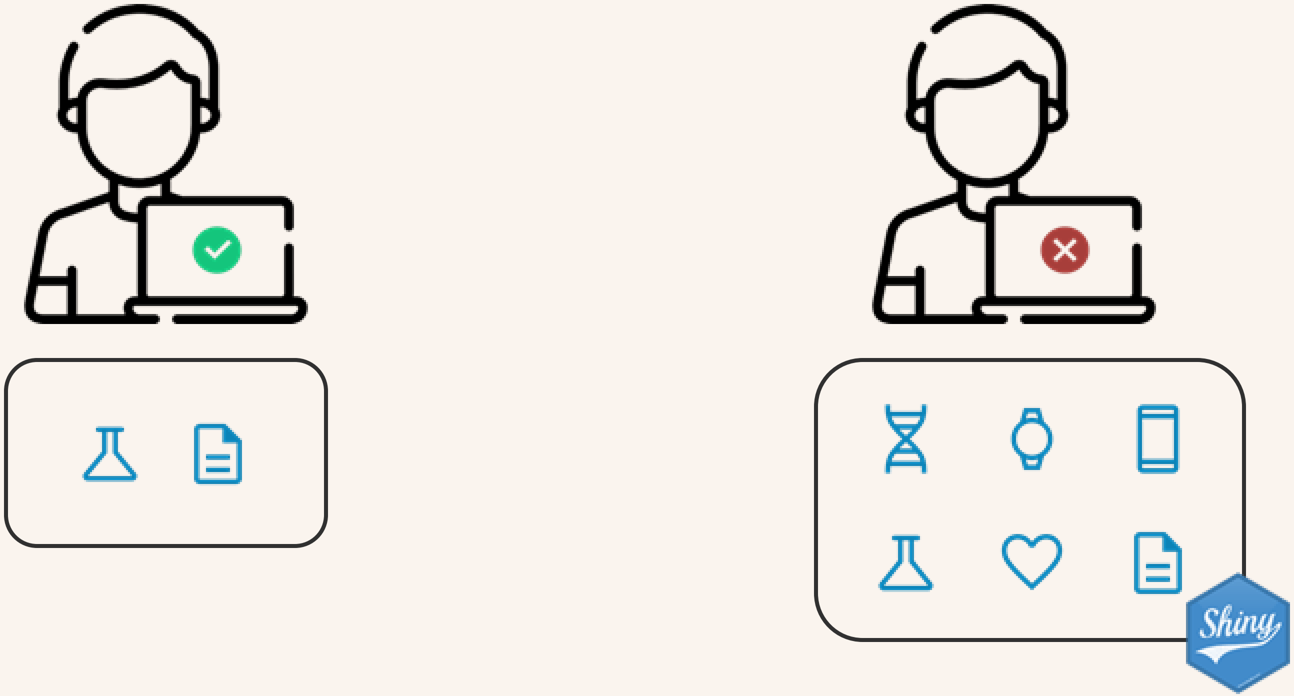
Labs

ECGs

EDC

Output complexity is rapidly increasing…

Manual testing cannot maintain quality at this scale

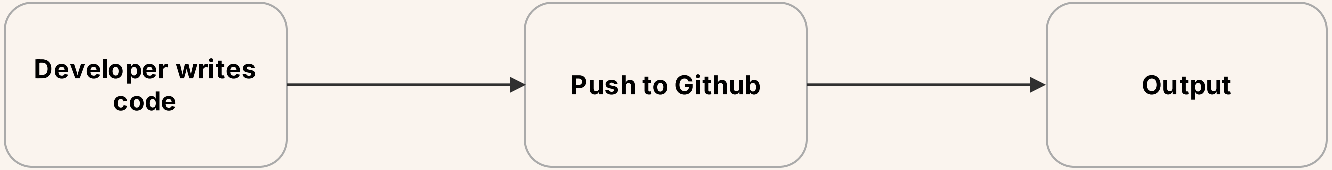


We need automation + AI to maintain quality at scale

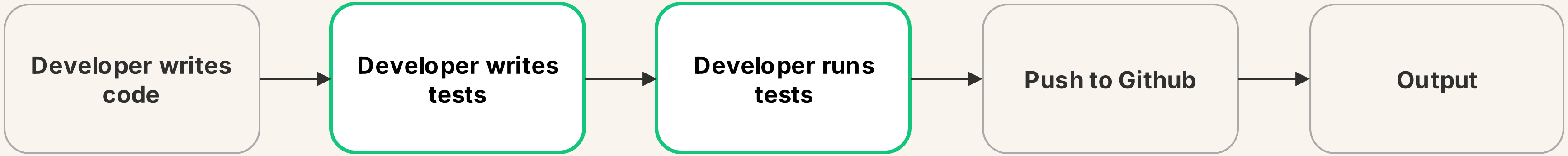
Programmatic testing in R

Locally executed tests to enable reproducibility and shareability of code

Baseline programming workflow



Proposed addition



Programmatic testing in R can increase quality of outputs

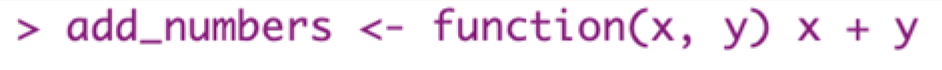
testthat

- The most widely used R package for **unit testing**.
- Designed for testing functions in R scripts and R packages.
- Helps validate correctness, expected outputs, and edge cases.
- Integrates well with Continuous Integration (CI) workflows.

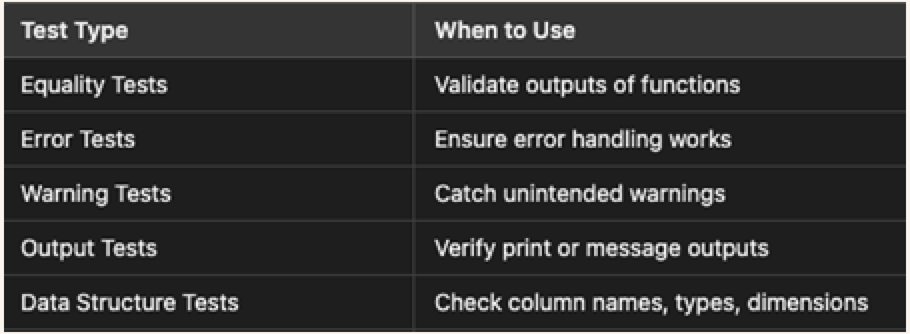
shinytest2

- A package for testing **Shiny applications**.
- Uses a record-and-replay system to validate app behavior.
- Enables snapshot testing to check UI consistency over time.
- Helps detect unintended changes in interactive elements.

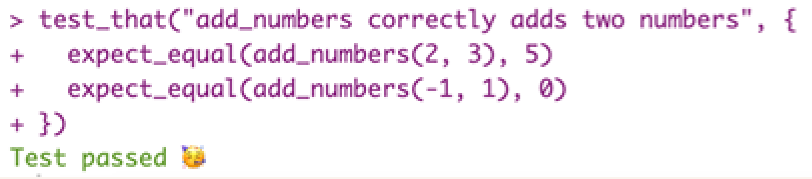
{testthat} supports a variety of unit tests



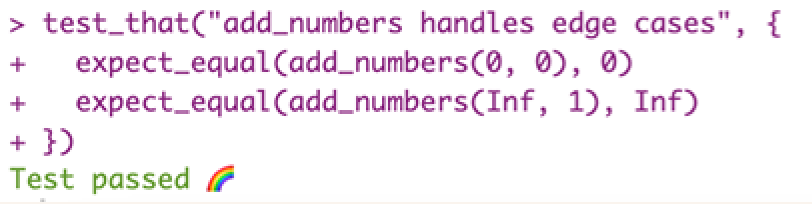
```
> add_numbers <- function(x, y) x + y
```



Test Type	When to Use
Equality Tests	Validate outputs of functions
Error Tests	Ensure error handling works
Warning Tests	Catch unintended warnings
Output Tests	Verify print or message outputs
Data Structure Tests	Check column names, types, dimensions

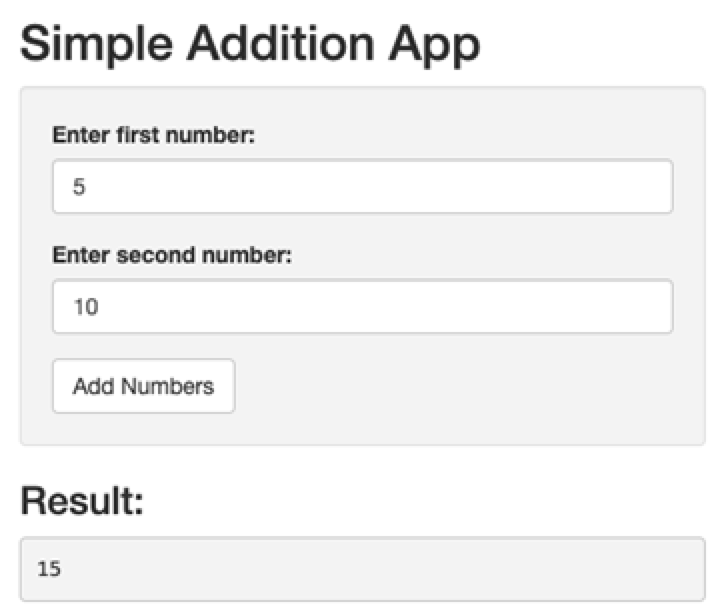


```
> test_that("add_numbers correctly adds two numbers", {
+   expect_equal(add_numbers(2, 3), 5)
+   expect_equal(add_numbers(-1, 1), 0)
+ })
Test passed 🥳
```



```
> test_that("add_numbers handles edge cases", {
+   expect_equal(add_numbers(0, 0), 0)
+   expect_equal(add_numbers(Inf, 1), Inf)
+ })
Test passed 🌈
```

{shinytest2} is used to test RShiny UI and interactive features



Simple Addition App

Enter first number:

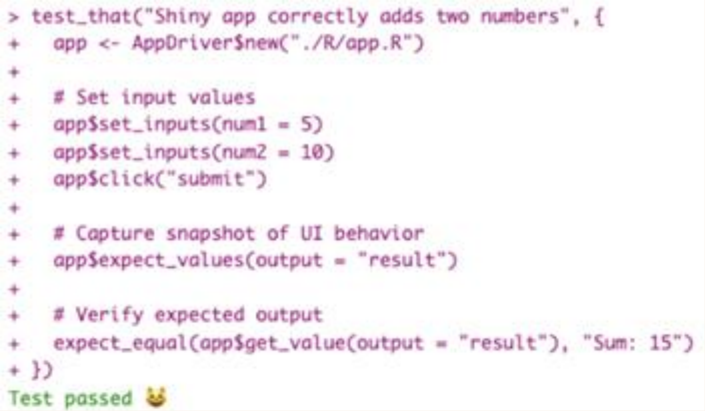
5

Enter second number:

10

Result:

15



```
> test_that("Shiny app correctly adds two numbers", {
+   app <- AppDriver$new("./R/app.R")
+
+   # Set input values
+   app$set_inputs(num1 = 5)
+   app$set_inputs(num2 = 10)
+   app$click("submit")
+
+   # Capture snapshot of UI behavior
+   app$expect_values(output = "result")
+
+   # Verify expected output
+   expect_equal(app$get_value(output = "result"), "Sum: 15")
+ })
Test passed 😸
```

Locally executed tests help with reproducibility but not scale

Benefits

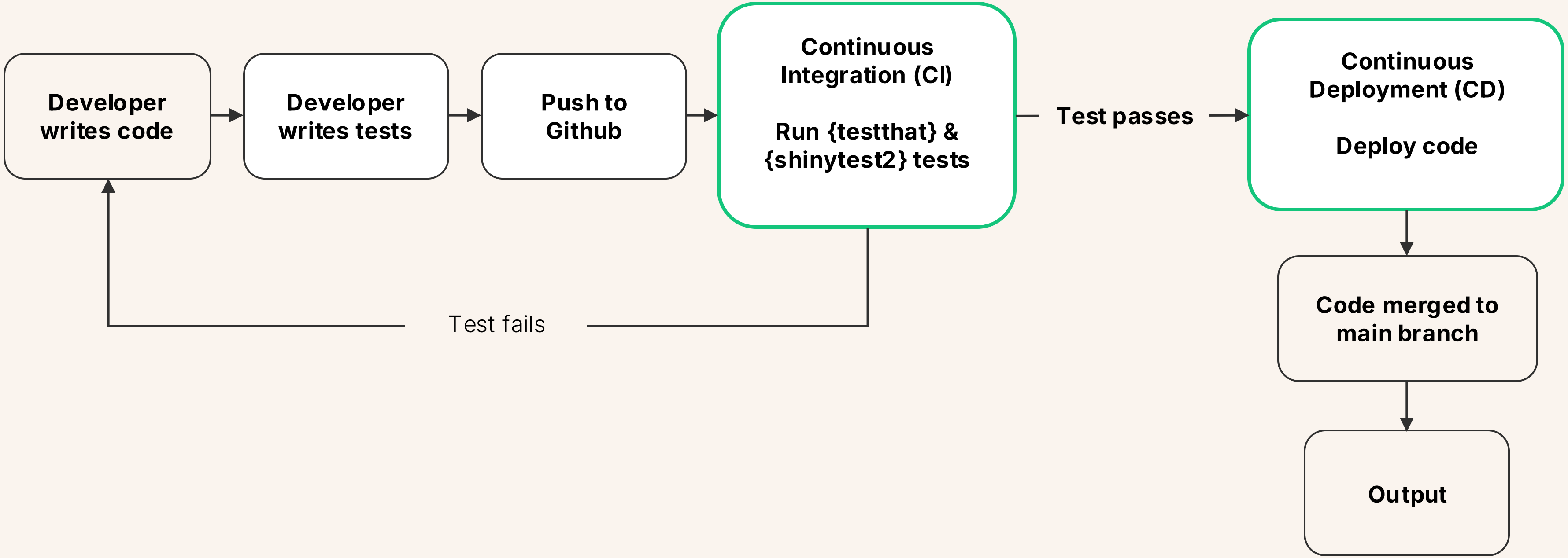
- **Increase quality** by catching errors before sharing output
- **Support reproducibility** of tests especially in interactive development cycles
- **Reduce manual effort** by unlocking ability to quickly re-run tests

Limitations

- **Requires manual execution** of tests locally in posit
- **Not scalable** to support highly iterative development
- **No quality enforcement** as a technical barrier to sharing output

Automatic testing with CI/CD

CI/CD automatically executes tests and ensures quality output


Developer writes code
Developer writes tests
Push to Github
Continuous Integration (CI)
Run {testthat} & {shinytest2} tests
Test passes
Continuous Deployment (CD)
Deploy code
Code merged to main branch
Output
Test fails

CI/CD increases scalability but still can be limited by SME knowledge

Benefits

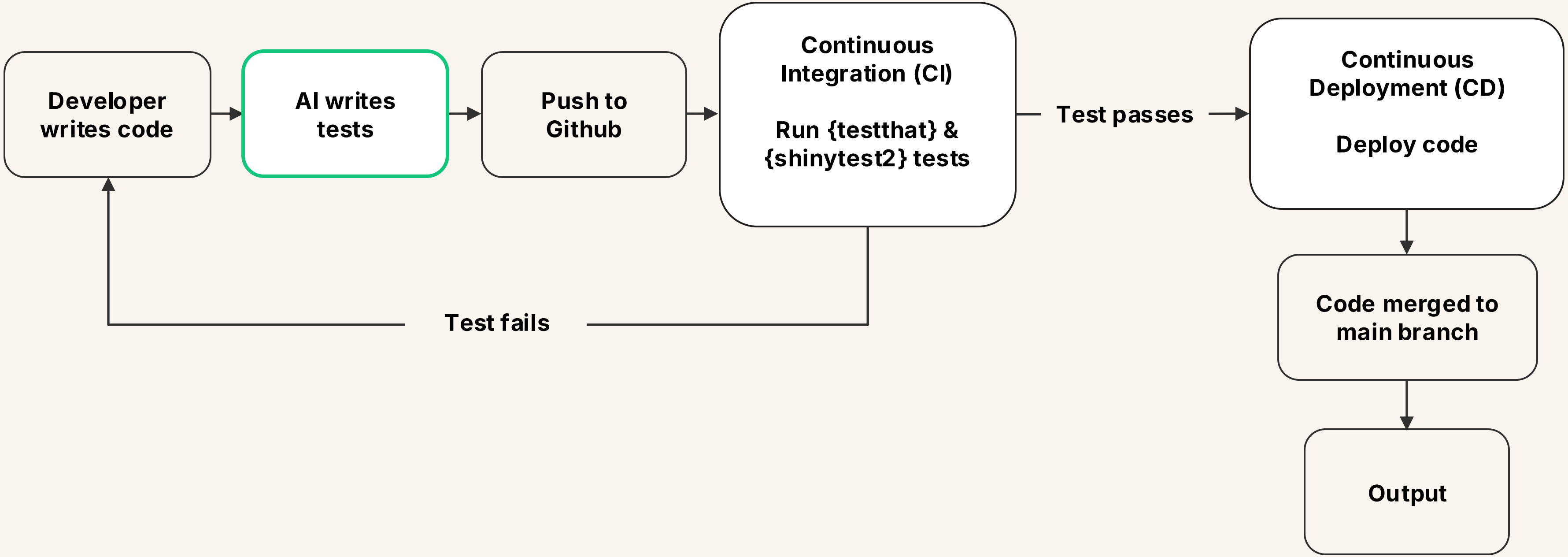
- **Automated testing**, reducing manual effort
- **Consistency across environments** as CI/CD ensures code behaves on local machine and in production environments
- **Ensures quality** by requiring tests to pass before deployment

Limitations

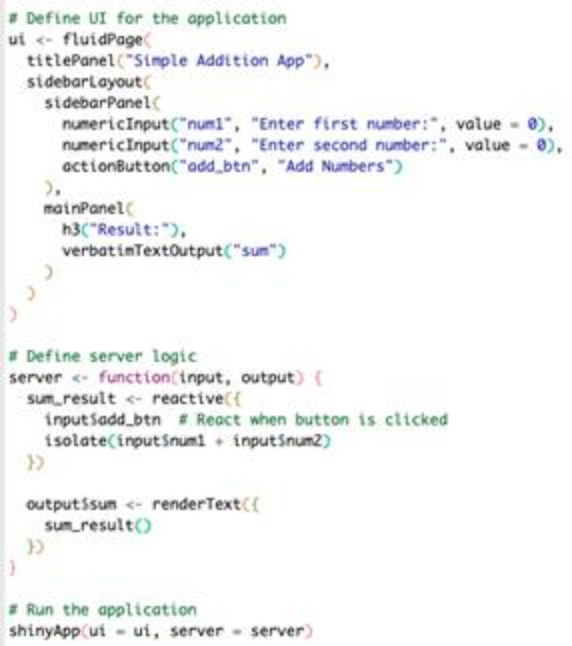
- **Requires initial setup** of YAML configuration files & specific folder structure
- **Requires SMEs** to learn testing tools and testing strategies
- **Does not solve bottleneck** of manual test creation

Write tests with AI

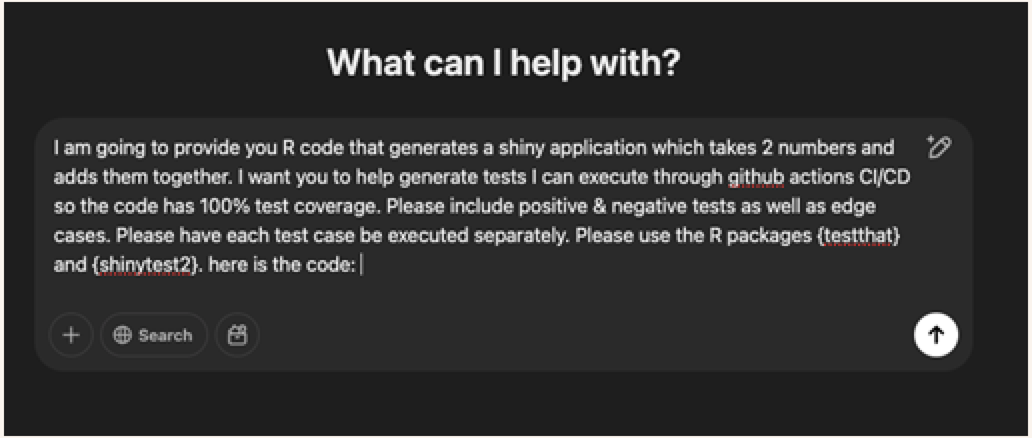
LLMs can help automate code generation to accelerate development


Developer writes code
AI writes tests
Push to Github
Continuous Integration (CI)
Run {testthat} & {shinytest2} tests
Test passes
Continuous Deployment (CD)
Deploy code
Code merged to main branch
Output
Test fails

Pass code to GPT to generate tests



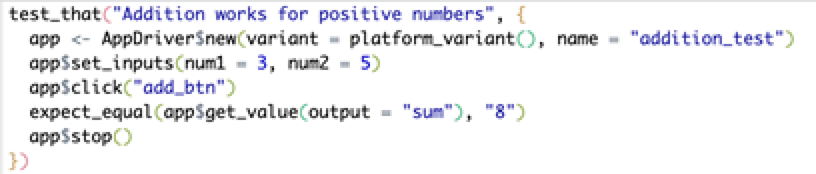
```
# Define UI for the application
ui <- fluidPage(
  titlePanel("Simple Addition App"),
  sidebarLayout(
    sidebarPanel(
      numericInput("num1", "Enter first number:", value = 0),
      numericInput("num2", "Enter second number:", value = 0),
      actionButton("add_btn", "Add Numbers")
    ),
    mainPanel(
      h3("Result:"),
      verbatimTextOutput("sum")
    )
  )
)

# Define server logic
server <- function(input, output) {
  sum_result <- reactive({
    input$add_btn  # React when button is clicked
    isolate(input$num1 + input$num2)
  })

  output$sum <- renderText({
    sum_result()
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```



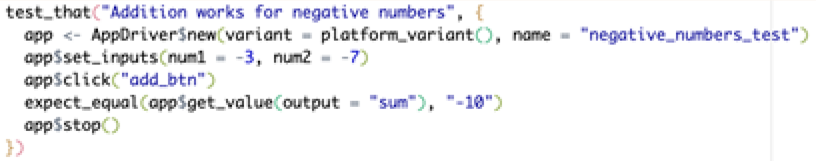
What can I help with?

I am going to provide you R code that generates a shiny application which takes 2 numbers and adds them together. I want you to help generate tests I can execute through github actions CI/CD so the code has 100% test coverage. Please include positive & negative tests as well as edge cases. Please have each test case be executed separately. Please use the R packages {testthat} and {shinytest2}. here is the code:

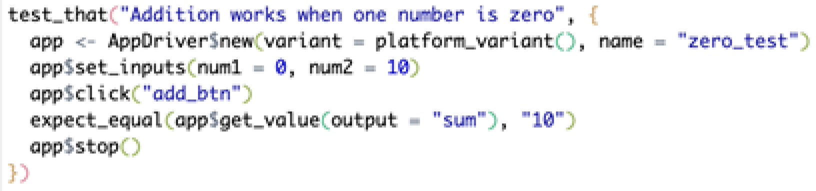
Pass code to GPT to generate tests



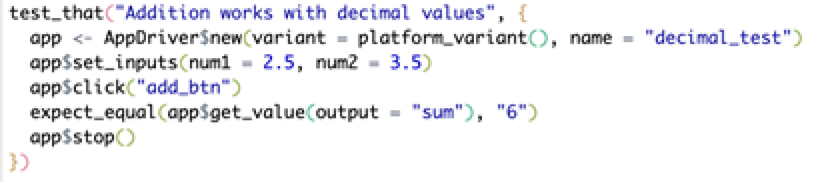
```r
test_that("Addition works for positive numbers", {
  app <- AppDriver$new(variant = platform_variant(), name = "addition_test")
  app$set_inputs(num1 = 3, num2 = 5)
  app$click("add_btn")
  expect_equal(app$get_value(output = "sum"), "8")
  app$stop()
})
```



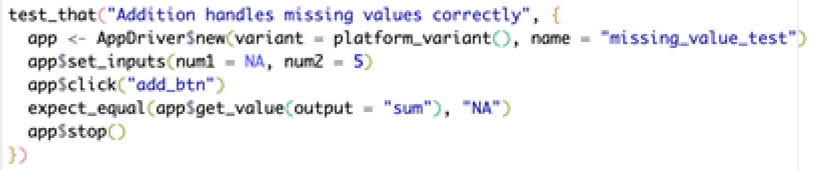
```r
test_that("Addition works for negative numbers", {
  app <- AppDriver$new(variant = platform_variant(), name = "negative_numbers_test")
  app$set_inputs(num1 = -3, num2 = -7)
  app$click("add_btn")
  expect_equal(app$get_value(output = "sum"), "-10")
  app$stop()
})
```



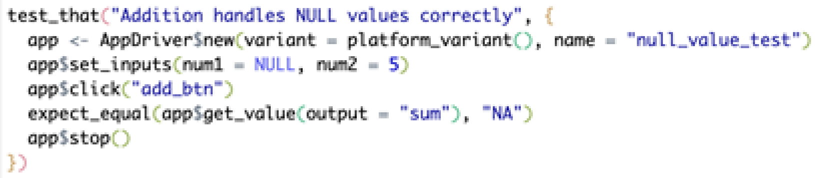
```r
test_that("Addition works when one number is zero", {
  app <- AppDriver$new(variant = platform_variant(), name = "zero_test")
  app$set_inputs(num1 = 0, num2 = 10)
  app$click("add_btn")
  expect_equal(app$get_value(output = "sum"), "10")
  app$stop()
})
```



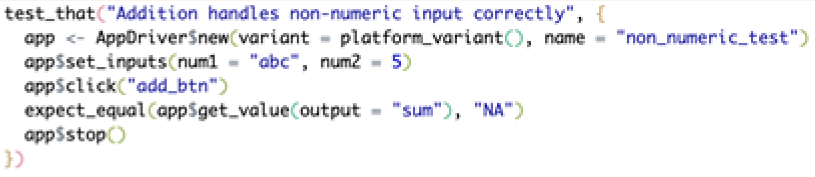
```r
test_that("Addition works with decimal values", {
  app <- AppDriver$new(variant = platform_variant(), name = "decimal_test")
  app$set_inputs(num1 = 2.5, num2 = 3.5)
  app$click("add_btn")
  expect_equal(app$get_value(output = "sum"), "6")
  app$stop()
})
```



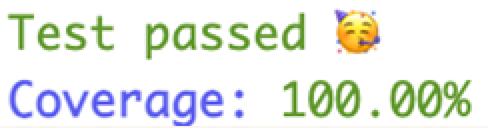
```r
test_that("Addition handles missing values correctly", {
  app <- AppDriver$new(variant = platform_variant(), name = "missing_value_test")
  app$set_inputs(num1 = NA, num2 = 5)
  app$click("add_btn")
  expect_equal(app$get_value(output = "sum"), "NA")
  app$stop()
})
```



```r
test_that("Addition handles NULL values correctly", {
  app <- AppDriver$new(variant = platform_variant(), name = "null_value_test")
  app$set_inputs(num1 = NULL, num2 = 5)
  app$click("add_btn")
  expect_equal(app$get_value(output = "sum"), "NA")
  app$stop()
})
```

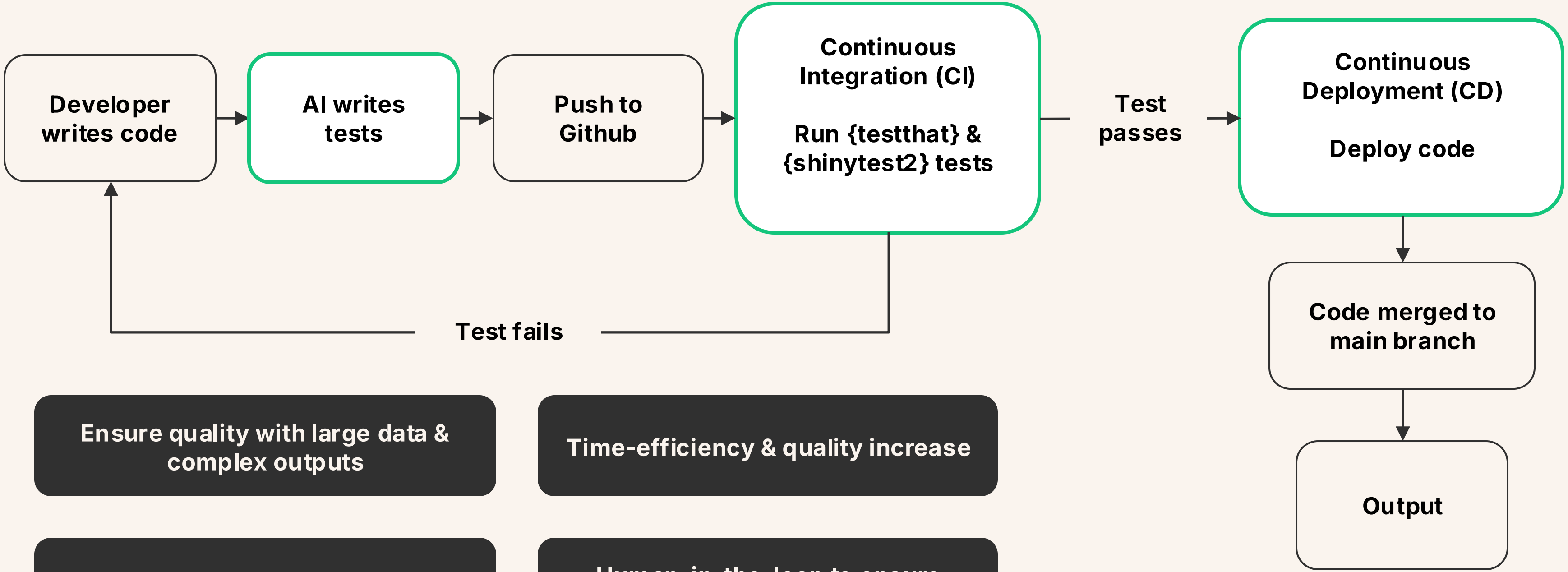


```r
test_that("Addition handles non-numeric input correctly", {
  app <- AppDriver$new(variant = platform_variant(), name = "non_numeric_test")
  app$set_inputs(num1 = "abc", num2 = 5)
  app$click("add_btn")
  expect_equal(app$get_value(output = "sum"), "NA")
  app$stop()
})
```



```
Test passed 🥳
Coverage: 100.00%
```

Automation + AI to maintain quality at scale



Developer writes code

AI writes tests

Push to Github

Continuous Integration (CI)

Run {testthat} & {shinytest2} tests

Test passes

Continuous Deployment (CD)

Deploy code

Code merged to main branch

Output

Test fails

Ensure quality with large data & complex outputs

Time-efficiency & quality increase

Minimize bottleneck on SMEs

Human-in-the-loop to ensure quality

Questions?