November 16, 2025

JazzAIR

Advancing Clinical Study Reporting Through Modern Development Practices and Testing Strategy

Jakub Sobolewski, Staff Engineer @ Appsilon

Jazz Pharmaceuticals

Appsilon

AGENDA

01 What is JazzAIR?

Code starter for study interactive analysis apps.

02 How software architecture impacts quality assurance?

Finding the testing strategy in complex software systems.

03 Breaking it down into testable pieces.

What, when and how to test.

04 Putting it all back together.

Saving testing effort and getting more confidence.

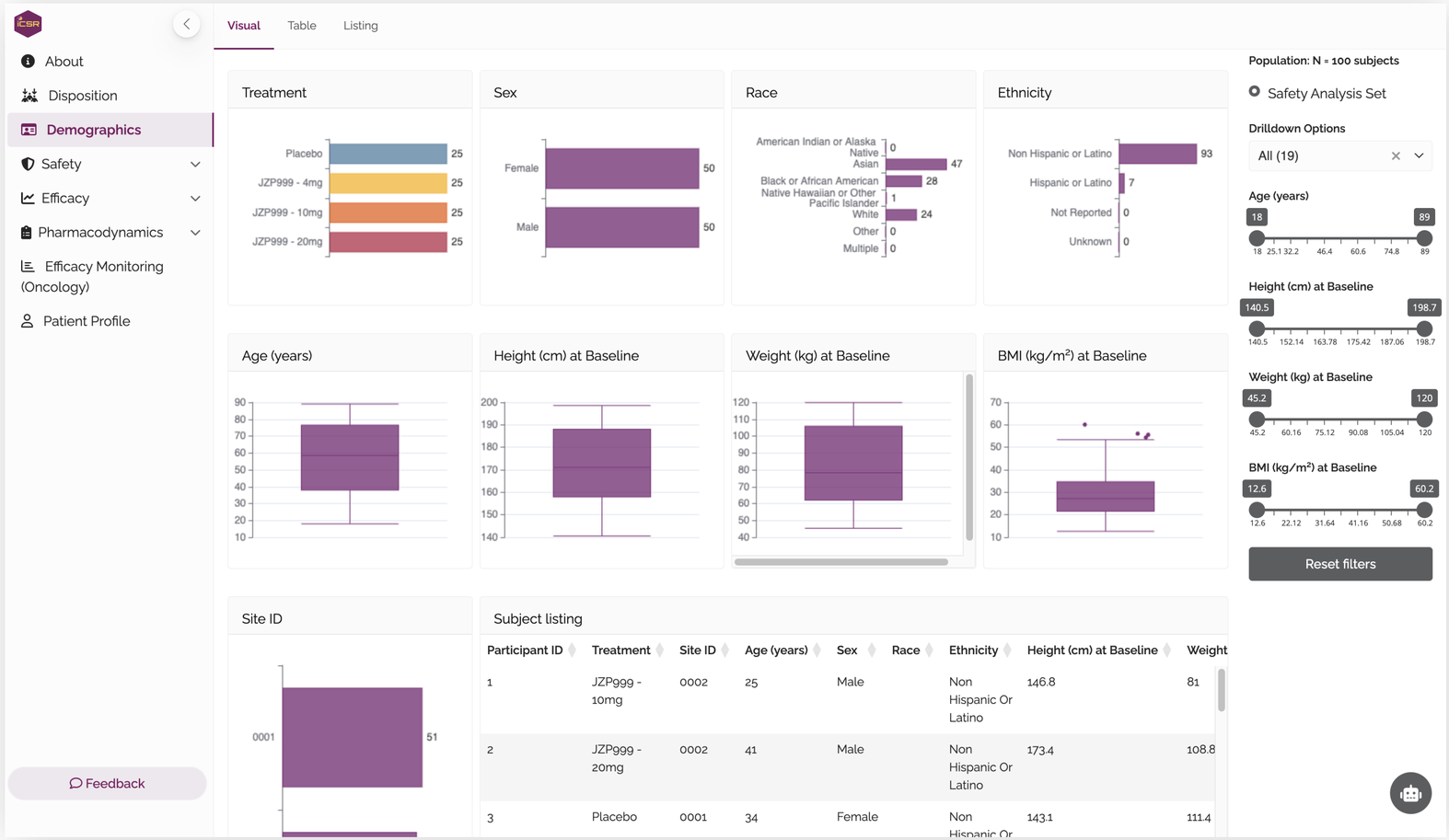
What is JazzAIR?

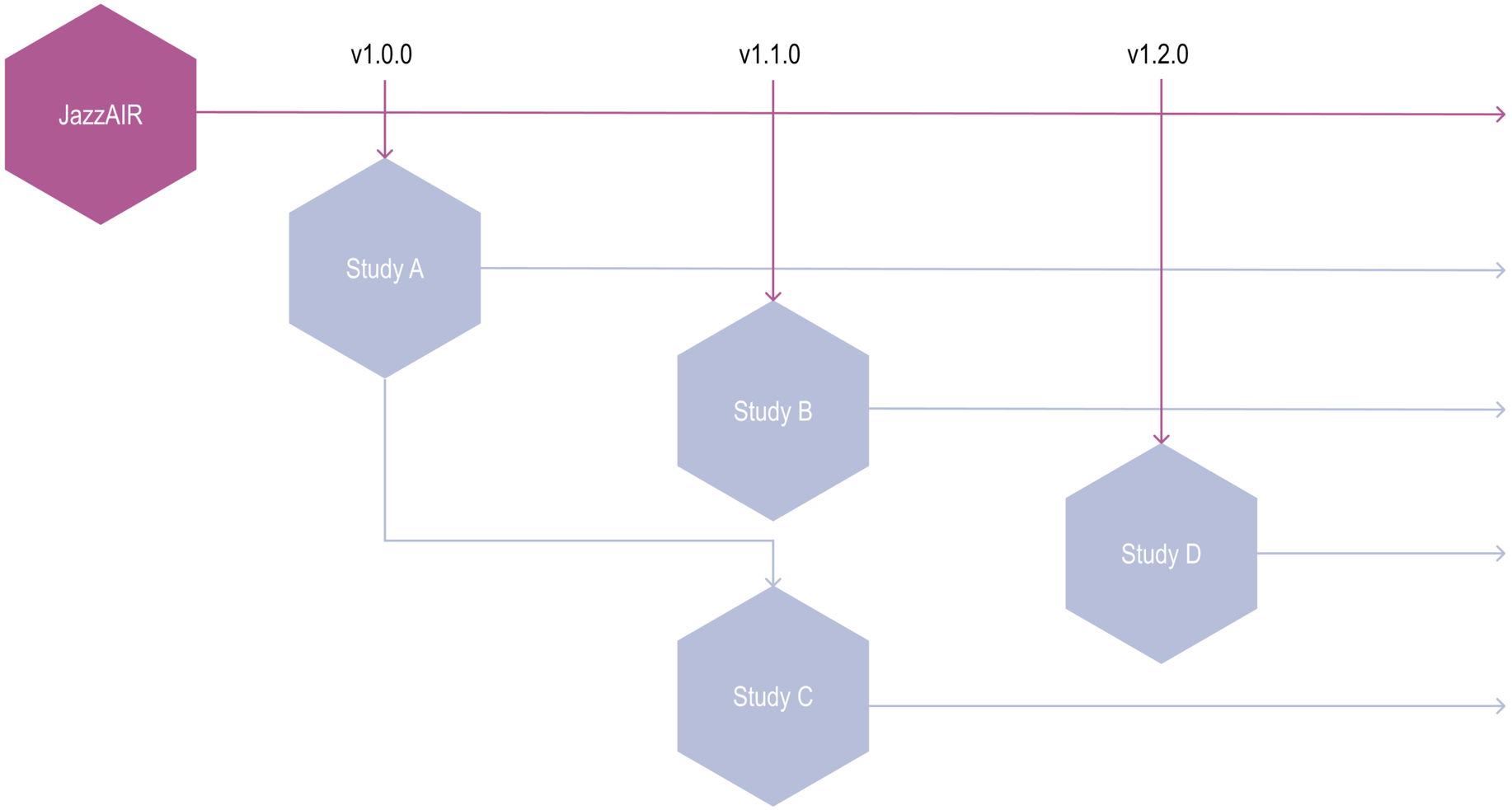
Code starter for study interactive analysis apps

The goal – reduce the time to analysis.

The solution – a full stack R/Shiny code template.

- ✓ Data preparation
  - ✓ Reading ADaM tables from storage.
  - ✓ Mapping variables.
  - ✓ Custom subgroups.
- ✓ Data presentation:
  - ✓ Common TLFs libraries.
  - ✓ Common User Interface style.
- ✓ Coding standards:
  - ✓ Common code structure
  - ✓ Common Shiny module structure.
  - ✓ Easy to switch between studies.




JazzAIR
v1.0.0
v1.1.0
v1.2.0
Study A
Study B
Study C
Study D

How software architecture impacts quality assurance?

Finding the testing strategy in complex software systems.

JazzAIR

Demographics

- Tables
- Listings
- Figures

Adverse Events

- Tables
- Listings
- Figures

...

Study A

Demographics

- Tables
- Listings
- Figures

Adverse Events

- Tables
- Listings
- Figures

...

The more components, the more testing is needed.

The more code copies, the more testing is needed.

Each study app team needs to validate:

- ✓ **Components** are rendered correctly.
- ✓ **Calculations** are correct.
- ✓ **Shiny** app works (it doesn't crash).

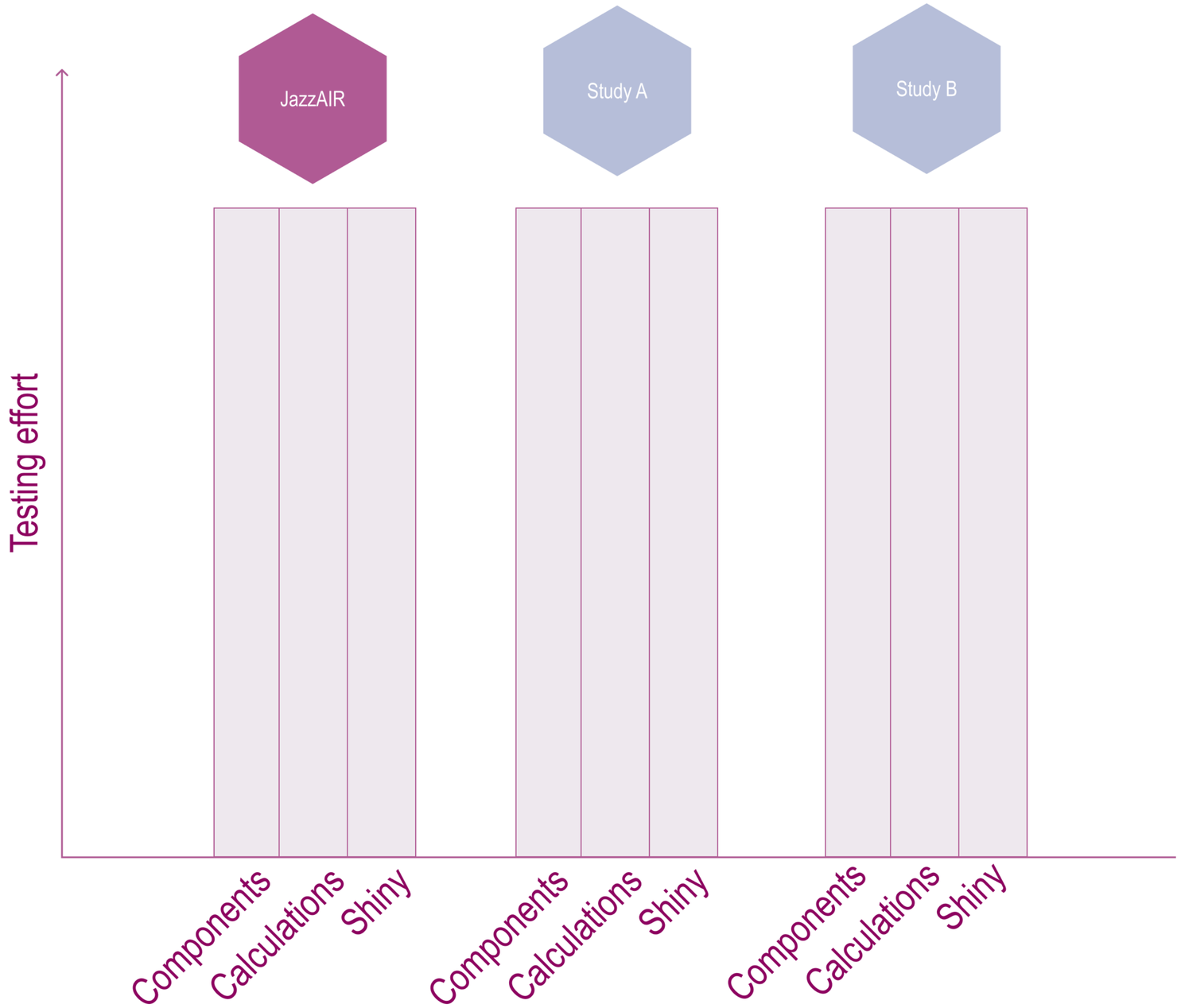
Each study app team needs to do all testing activities themselves.

Manual or automated.

Breaking it down into testable pieces.

What, when and how to test

Before


JazzAIR
Study A
Study B
Testing effort
Components
Calculations
Shiny
Components
Calculations
Shiny
Components
Calculations
Shiny

Each study app team needs to validate:

- ✓ **Components** are rendered correctly.
- ✓ **Calculations** are correct.
- ✓ **Shiny** app works (it doesn't crash).

Each study app team needs to validate:

- ✓ **Components are rendered correctly.**
  Common figures can be unit-tested in package.
  Custom figures are tested by the study team.
- ✓ Calculations are correct.
- ✓ Shiny app works (it doesn't crash).

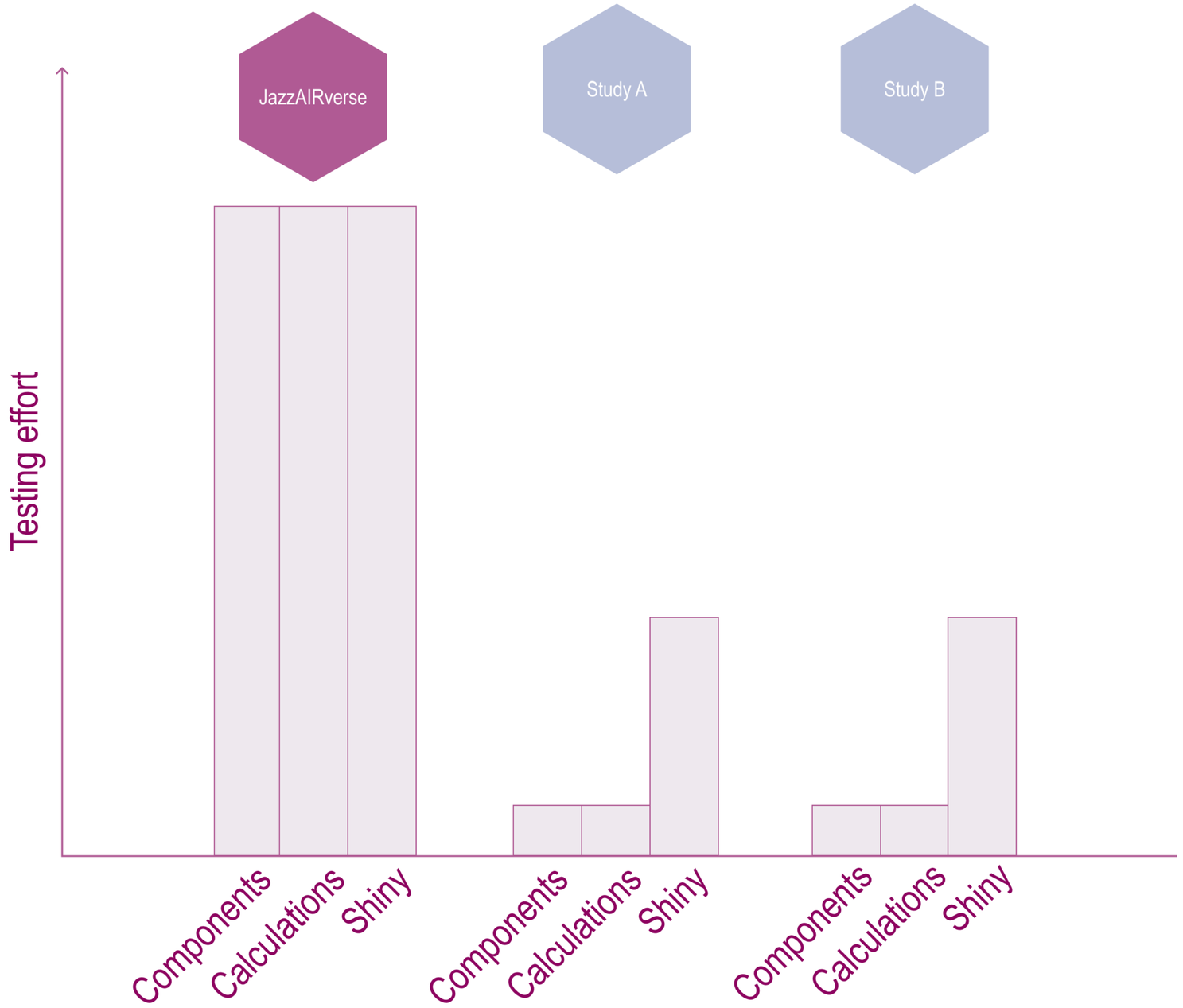
Each study app team needs to validate:

- ✓ Components are rendered correctly.
- ✓ **Calculations are correct.**
  Common calculations can be unit-tested in package.
  Custom calculations are tested by the study team.
- ✓ Shiny app works (it doesn't crash).

Each study app team needs to validate:

- ✓ Components are rendered correctly.
- ✓ Calculations are correct
- ✓ **Shiny app works (it doesn't crash).**
  The app template is tested in a package.
  The app itself is tested by the study team.

Target


JazzAIRverse
Study A
Study B
Testing effort
Components
Calculations
Shiny
Components
Calculations
Shiny
Components
Calculations
Shiny

Finding where testing responsibility lies

Testing correctness

- Are those numbers correct?
- Do those figures look correct?

- Components tests
- Calculations tests

Testing specifications/user experience

- Can user see a breakdown by this variable?
- Can user see this subset of data?
- Can user export this artifact?

- Shiny tests

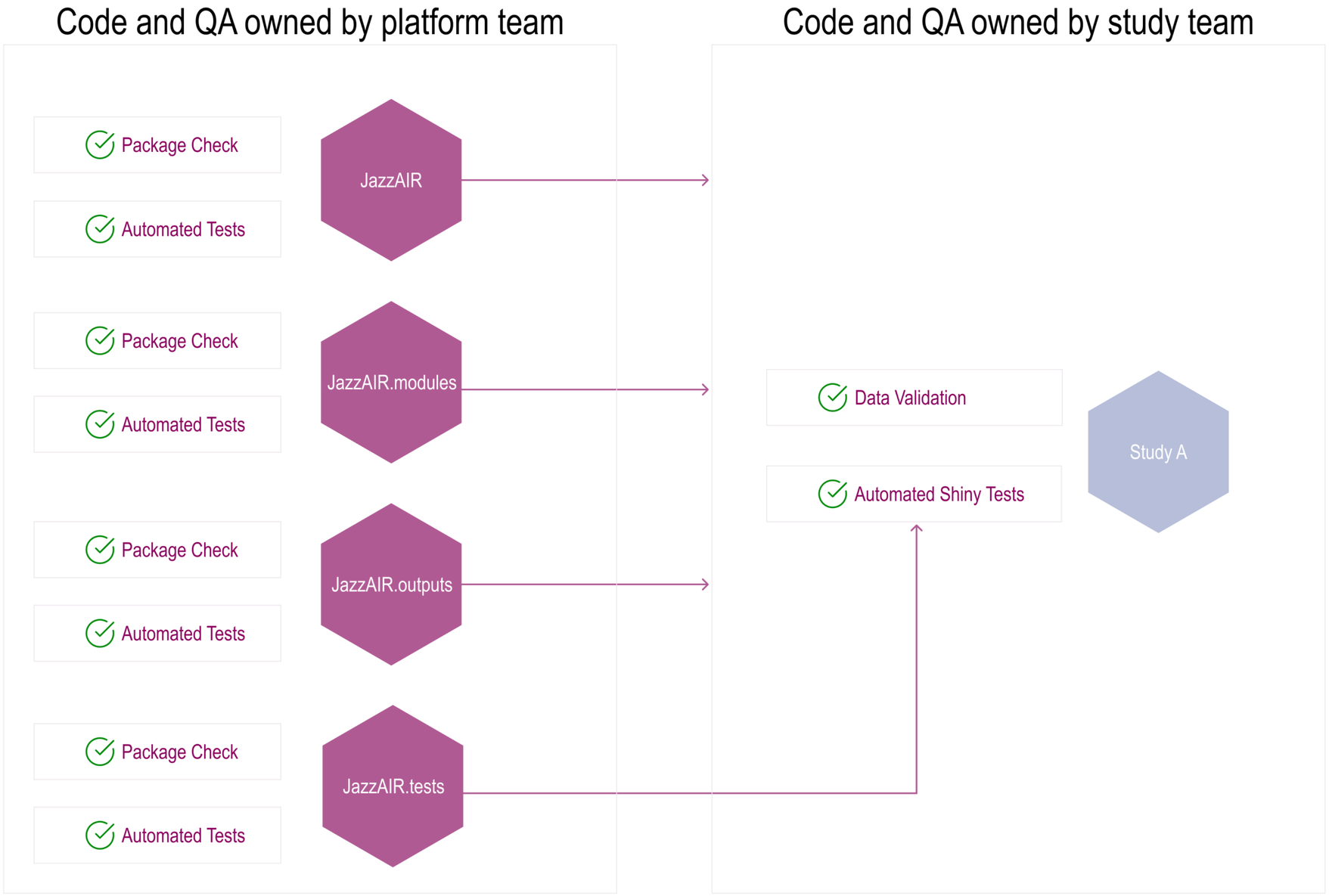
Finding where testing responsibility lies

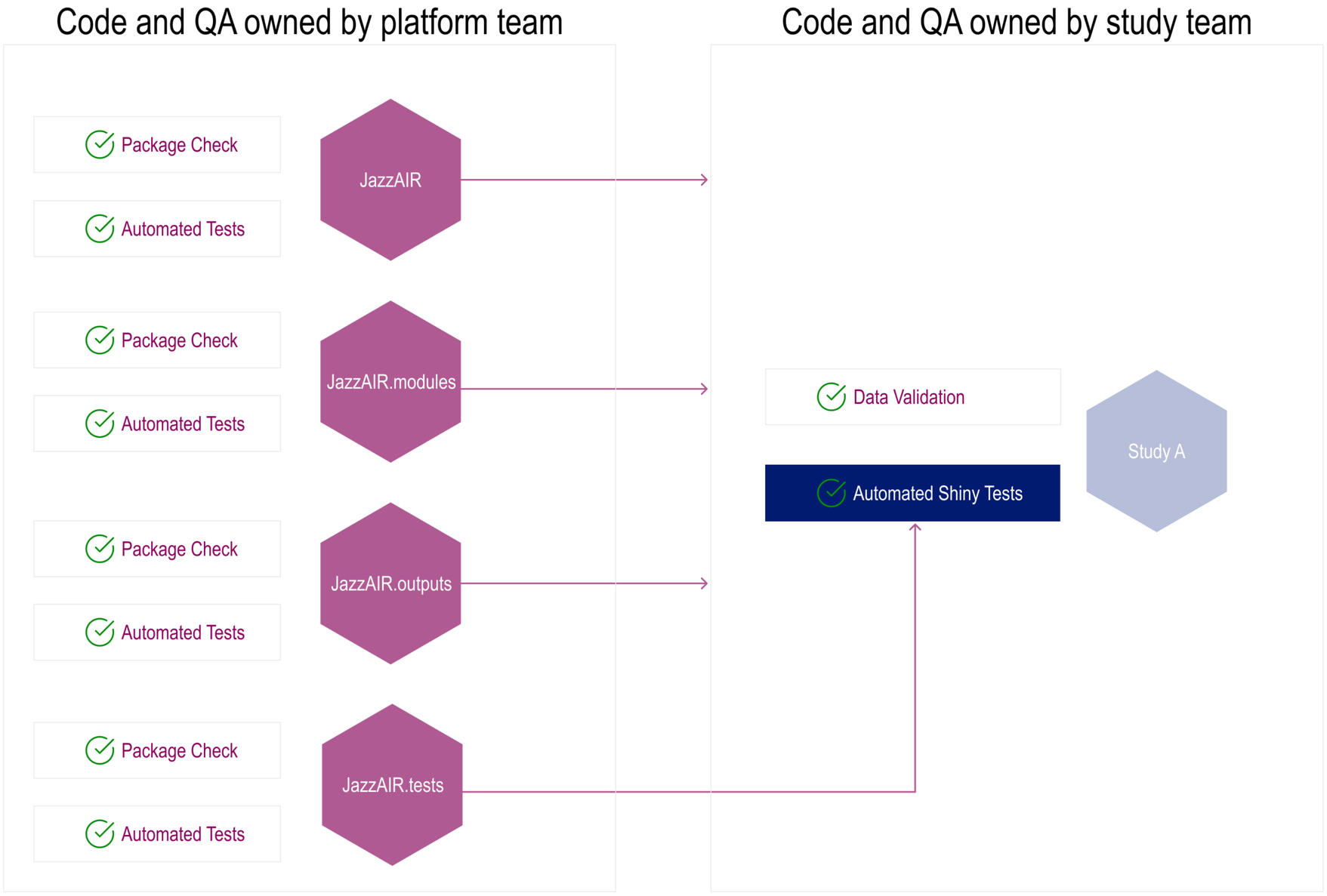
Testing correctness

Extract common logic to packages to test and reuse it.

Testing specifications/user experience

Give study app teams tools to write Shiny tests themselves.


Code and QA owned by platform team
Code and QA owned by study team
Package Check
Automated Tests
JazzAIR
Package Check
Automated Tests
JazzAIR.modules
Package Check
Automated Tests
JazzAIR.outputs
Package Check
Automated Tests
JazzAIR.tests
Data Validation
Automated Shiny Tests
Study A

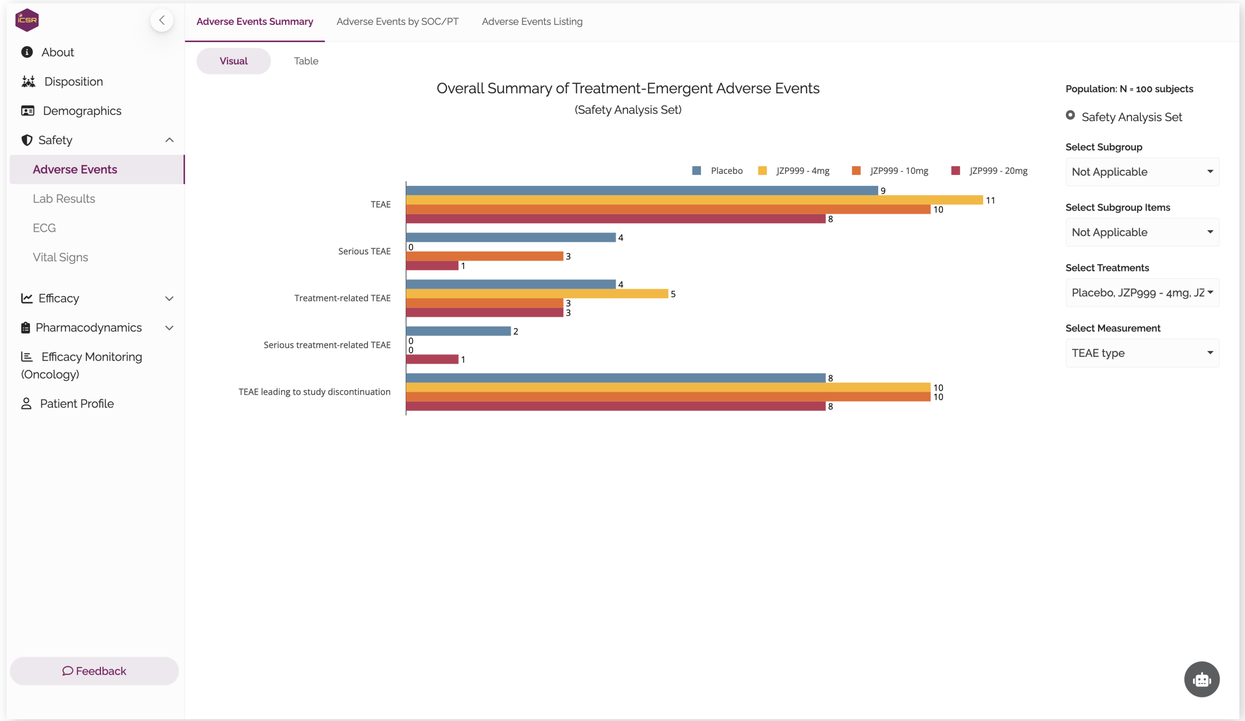

Code and QA owned by platform team
Code and QA owned by study team
Package Check
Automated Tests
JazzAIR
Package Check
Automated Tests
JazzAIR.modules
Package Check
Automated Tests
JazzAIR.outputs
Package Check
Automated Tests
JazzAIR.tests
Data Validation
Automated Shiny Tests
Study A

Describe specs in plain language


```
Feature: Adverse Events Summary Visual
  Background:
    * I navigate to "Safety"
    * I navigate to "Adverse Events"
    * I navigate to "Adverse Events Summary"
    * I navigate to "Visual"

  Scenario Outline: Plotting by subgroup
    When I set "subgroup" to <subgroup>
    Then there are no visible errors
    Examples:
      | subgroup          |
      | "Not Applicable" |
      | "Sex"             |
```

Automated Shiny Tests

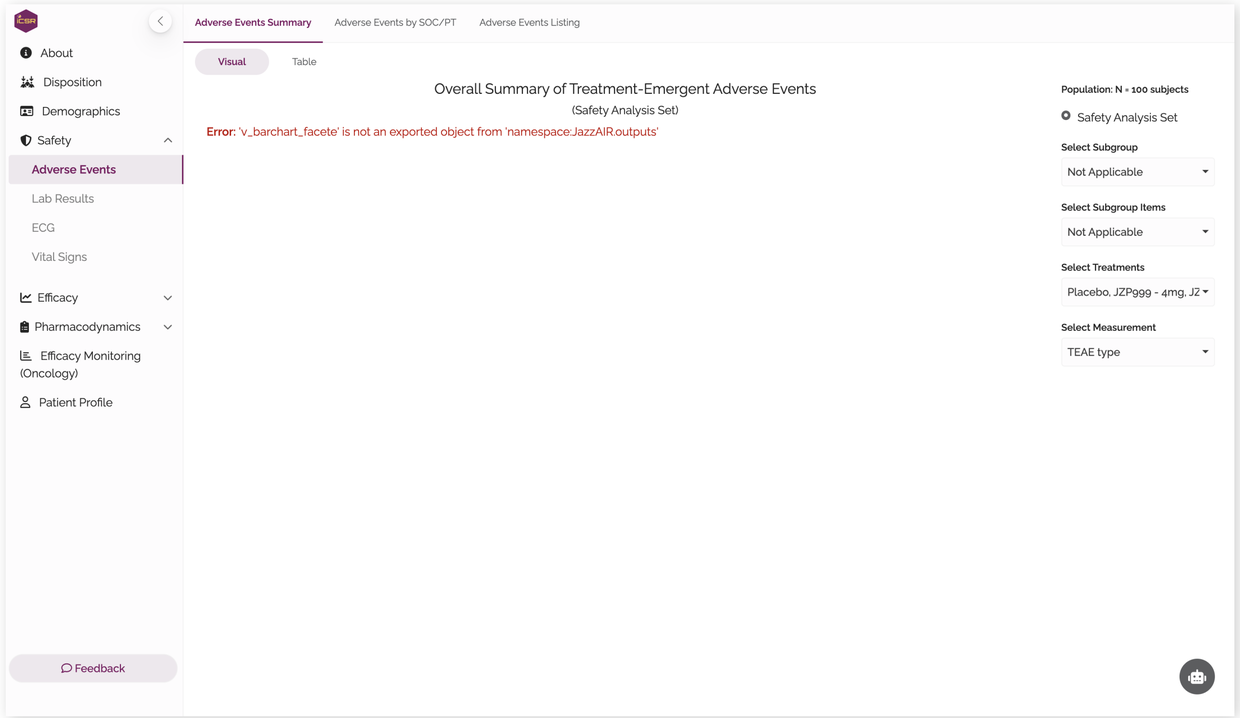
Describe specs in plain language




```
Feature: Adverse Events Summary Visual
  Background:
    * I navigate to "Safety"
    * I navigate to "Adverse Events"
    * I navigate to "Adverse Events Summary"
    * I navigate to "Visual"

  Scenario Outline: Plotting by subgroup
    When I set "subgroup" to <subgroup>
    Then there are no visible errors
    Examples:
      | subgroup          |
      | "Not Applicable" |
      | "Sex"            |
```

Automated Shiny Tests

Describe specs in plain language




```
Feature: Adverse Events Summary Visual
  Background:
    * I navigate to "Safety"
    * I navigate to "Adverse Events"
    * I navigate to "Adverse Events Summary"
    * I navigate to "Visual"

  Scenario Outline: Plotting by subgroup
    When I set "subgroup" to <subgroup>
    Then there are no visible errors
    Examples:
      | subgroup          |
      | "Not Applicable" |
      | "Sex"            |
```

Describe specs in plain language

- ✓ Only check if outputs are rendered.
- ✓ Correctness of rendering is tested in unit tests in a package.
- ✓ Abstract interaction with the interface.
  "I set <something> to <value>"
  "I navigate to <where>"

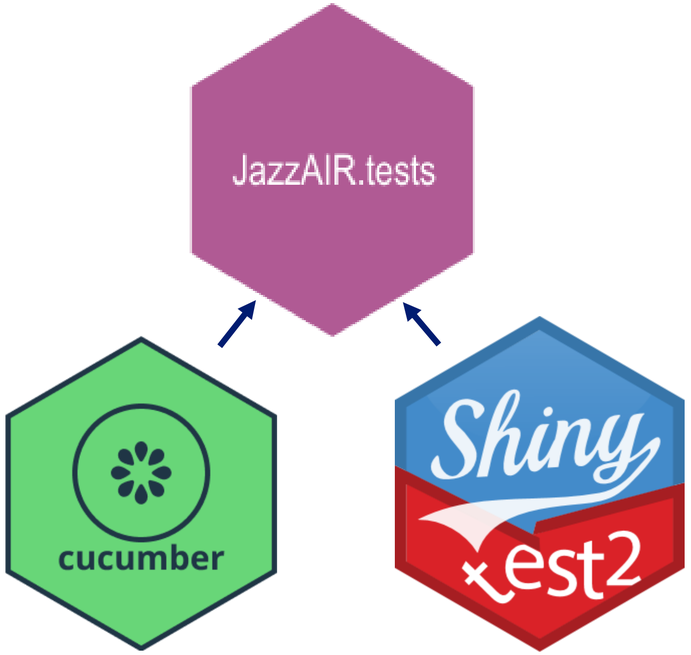
```
Feature: Adverse Events Summary Visual
  Background:
    * I navigate to "Safety"
    * I navigate to "Adverse Events"
    * I navigate to "Adverse Events Summary"
    * I navigate to "Visual"

  Scenario Outline: Plotting by subgroup
    When I set "subgroup" to <subgroup>
    Then there are no visible errors
    Examples:
      | subgroup          |
      | "Not Applicable" |
      | "Sex"             |
```

Adjust and extend according to study specification


```
Feature: Adverse Events Summary Visual
  Background:
    * I navigate to "Safety"
    * I navigate to "Adverse Events"
    * I navigate to "Adverse Events Summary"
    * I navigate to "Visual"

  Scenario Outline: Plotting by subgroup
    When I set "subgroup" to <subgroup>
    Then there are no visible errors
    Examples:
      | subgroup          |
      | "Not Applicable" |
      | "Sex"             |
      | "Race"            |
      | "Site"            |
```

Automated Shiny Tests

- ✓ Write code to run test once.
- ✓ Reuse across study apps.
- ✓ Add more interactions with app as needed.
- ✓ Hide all details of Shiny testing from developers.

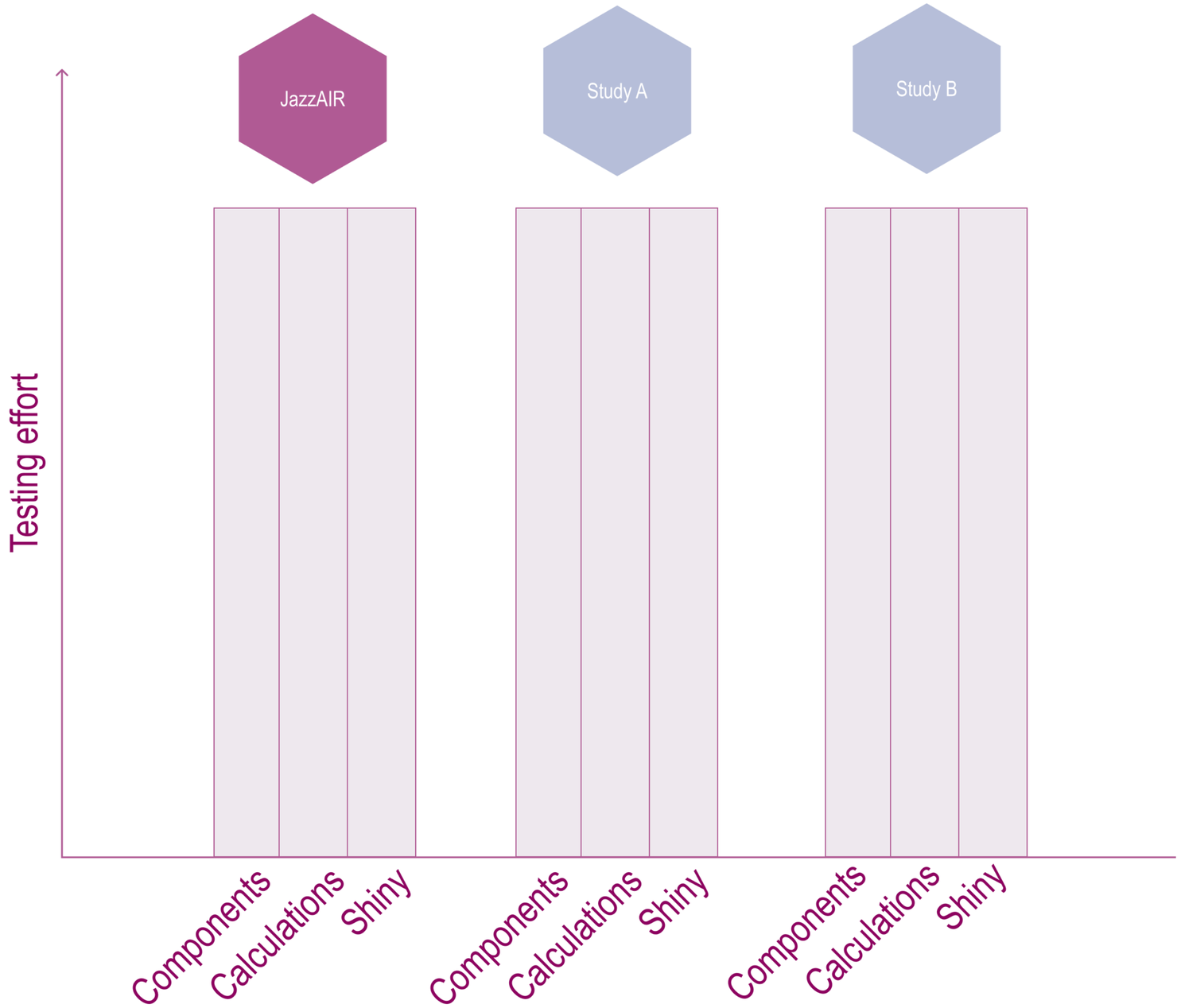



```r
library(cucumber)
library(JazzAIR.tests)

before(function(context, scenario_name) {
  context$driver <- JazzAIR.tests::ShinyDriver$new()
})

after(function(context, scenario_name) {
  context$driver$stop()
})

given("I navigate to {string}", function(where, context) {
  context$driver$i_navigate_to(where)
})

when("I set {string} to {string}", function(what, how, context) {
  context$driver$i_set(what, how[[1]])
})

when("I set {string}", function(what, how, context) {
  context$driver$i_set(what, how)
})

then("there are no visible errors", function(context) {
  context$driver$verify_no_errors()
})
```

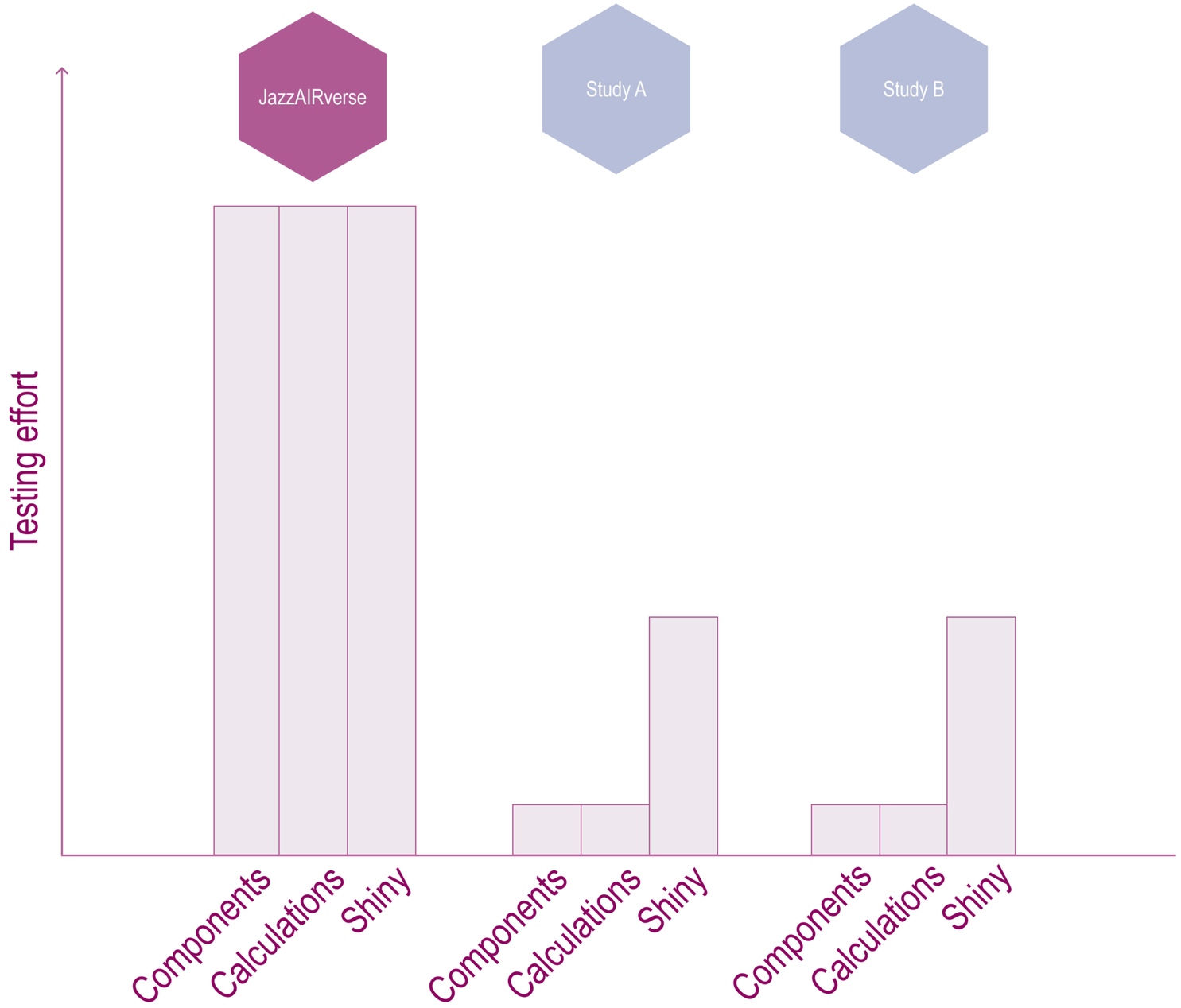
Putting it all back together.

Saving testing effort and getting more confidence.

Before


JazzAIR
Study A
Study B
Testing effort
Components
Calculations
Shiny
Components
Calculations
Shiny
Components
Calculations
Shiny

After


JazzAIRverse
Study A
Study B
Testing effort
Components
Calculations
Shiny
Components
Calculations
Shiny
Components
Calculations
Shiny

Before

- Study app teams are responsible for end-to-end testing their apps.
  - Components
  - Calculations
  - Shiny tests

After

- Study app teams are responsible for Shiny tests using plain-language specifications.
- Support packages are responsible for testing
  - Common components.
  - Common calculations.
  - Code to run tests.

Reduced testing and effort in study app teams.

Key questions

Who's responsible for testing it?

What assurance do I get from this test?

How do I test it?

Thank You