Understanding Testing Concepts

Daniel Stern
CODE WHISPERER

@danieljackstern

Why Testing?

Prevents regression

Provides objective success criteria

Facilitates complex, modular applications

What Are Tests?

A suite of tests is an application which checks your application

Composed of assertions about how your code will execute

Test files are committed to the repo with application code

Suite is run quickly and routinely by CI tools

What if Tests Didn’t Exist?

Someone would have to manually check the whole application every change

No easy way to know if your code has broken someone else’s

No way to measure the “correctness” of the code

As the application grows, the cost of manually checking for regression becomes burdensome

Eventually, adding new features becomes too risky and expensive, and the application can no longer grow

Advantages and Disadvantages of Testing

Advantages

Prevents unexpected regression

Reduces the need for manual verification

Verify corner cases

Allows developer to focus on current tasks (versus worrying about past ones)

Allows for modular construction of applications that would otherwise be too complex

Disadvantages

More code to write, debug and maintain

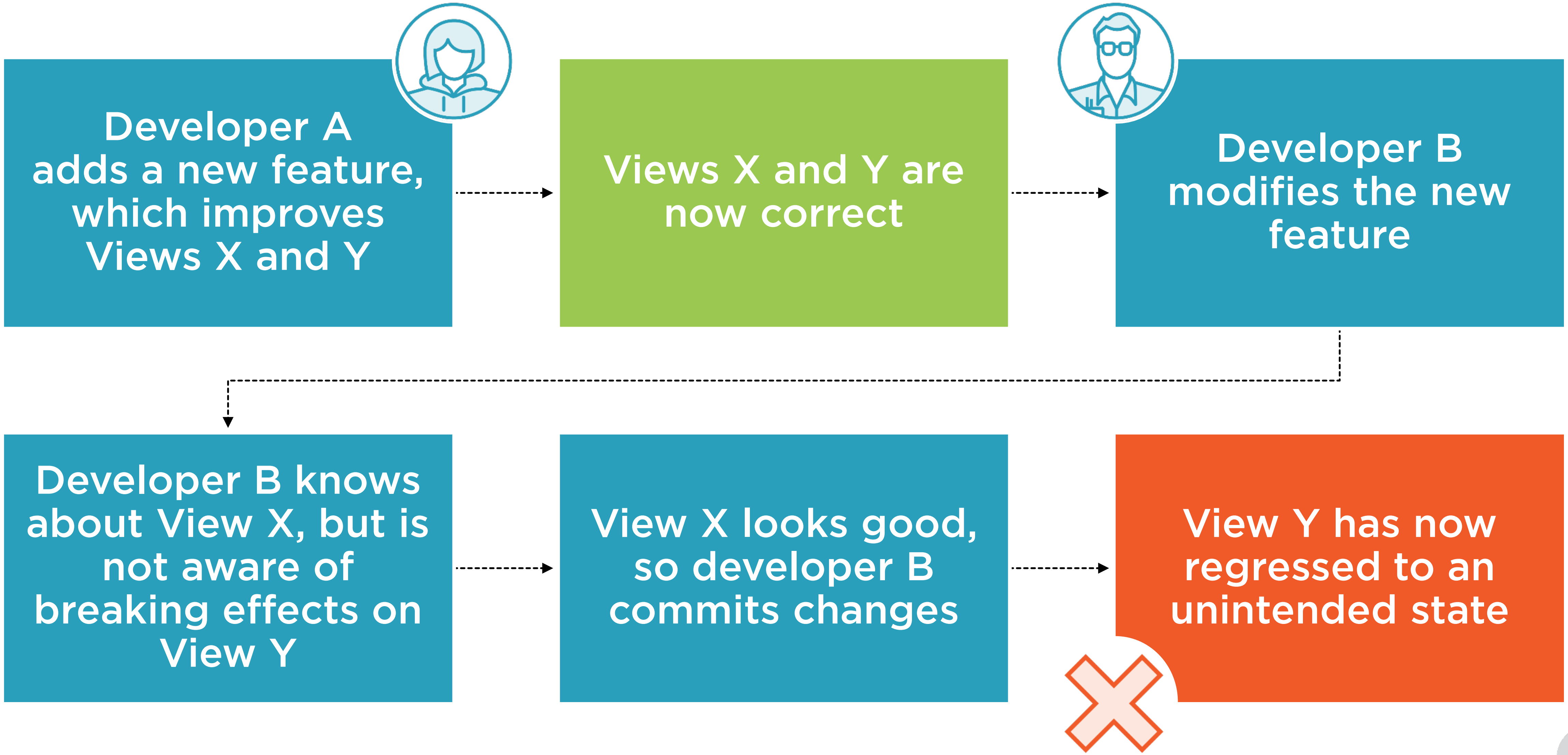
More tools that developers need to be able to use

Additional project dependency and cloud host compatibility concerns

Tests must actually be used and respected to be of value

Non-critical test failures may cause the app to be rejected on the CI level

How Regression Works



Developer A adds a new feature, which improves Views X and Y

Views X and Y are now correct

Developer B modifies the new feature

Developer B knows about View X, but is not aware of breaking effects on View Y

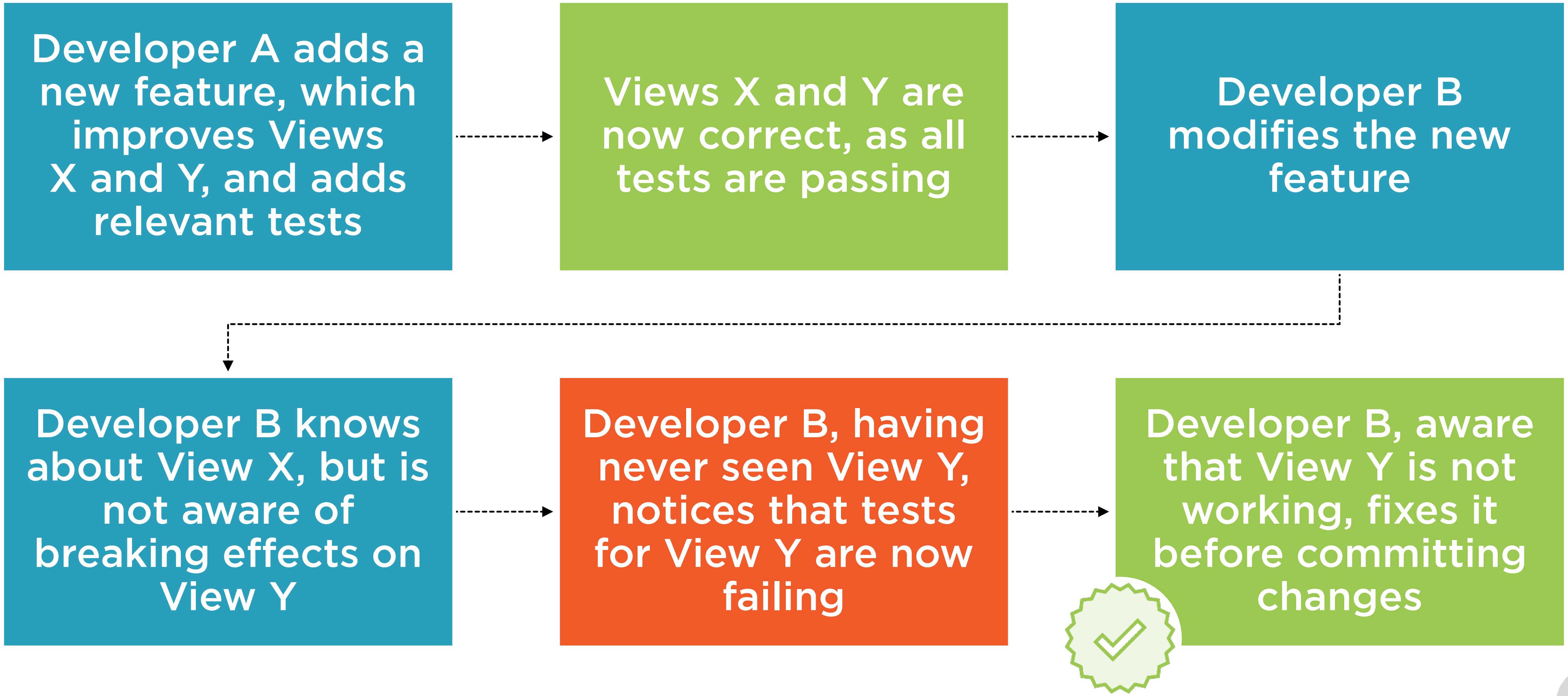
View X looks good, so developer B commits changes

View Y has now regressed to an unintended state

“The evil that men do lives after them.”

– William Shakespeare, Julius Caesar

How Testing Stops Regression



Developer A adds a new feature, which improves Views X and Y, and adds relevant tests

Views X and Y are now correct, as all tests are passing

Developer B modifies the new feature

Developer B knows about View X, but is not aware of breaking effects on View Y

Developer B, having never seen View Y, notices that tests for View Y are now failing

Developer B, aware that View Y is not working, fixes it before committing changes

Different Kinds of Tests

Different Kinds of Tests

Type of Test	What It Tests	Required Tools

Different Kinds of Tests

Type of Test	What It Tests	Required Tools
Unit Test	A single function or service	Mocha / Jest

Different Kinds of Tests

Type of Test	What It Tests	Required Tools
Unit Test	A single function or service	Mocha / Jest
Component Test	A single component (functionality)	Jest / Enzyme

Different Kinds of Tests

Type of Test	What It Tests	Required Tools
Unit Test	A single function or service	Mocha / Jest
Component Test	A single component (functionality)	Jest / Enzyme
Snapshot Test	**A single component (regression)**	**Jest**

Different Kinds of Tests

Type of Test	What It Tests	Required Tools
Unit Test	A single function or service	Mocha / Jest
Component Test	A single component (functionality)	Jest / Enzyme
Snapshot Test	A single component (regression)	Jest
End-to-End Test	**Interaction between multiple components**	**Protractor / Cypress**

Unit Tests

Verifies the functionality of a class or method

Simplest to write and execute

Used to test correctness of application logic

Tests can be written prior to application (TDD)

Component Tests

Verifies the correct appearance and functioning of a component

Highly sensitive to small changes to underlying components and services

Provides a strong defense against regression

Verifies changes to component output in response to change in application state

Does not verify interactions between two components

Snapshot Tests

A subtype of component test

Automatically generated by Jest

Verifies output matches a past record

Tends to fail if even the slightest change occurs

Performance Tests

Measure how long a block of code takes to execute

Can identify bottlenecks in application performance

Can provide insight into performance differences on different devices and cloud hosts

Coverage Tests

A test for your tests

Measures application code which is visited (but not necessarily verified) during tests

Does not indicate whether application works or not, but it is nice to have

“A Jest which will not bear serious examination is false wit.”

– **Aristotle**

End-to-end Tests

Measures the functionality of the whole application

Often executed in a virtual or “headless” browser

Creates a scenario to test by simulating user actions

Different in nature and more difficult to write than other tests

Provide the best assurance that the application works

End-to-end Tests (Cont'd)

Can verify interactions between two different components

Most sensitive to application changes, but challenging to fix

Generally unaffected if changes to code do not affect user experience

Summary

Tests prevent regression, verify functionality, and measure performance

The value of tests is increased for large or distributed teams

Regression is a costly phenomenon, that is prevented by testing, where adding a new feature causes an old one to break

Unit testing is most granular in scope, end-to-end testing the most broad

Coming up in the Next Module…

What Jest Is, How It Can Be Used

How Does Jest Differ from Other Frameworks?

Enzyme – Does It Fit Your Application?

Advantages and Disadvantages of Choosing Jest