CMPT 473
Software Testing, Reliability and Security

User Interface Testing & Automation

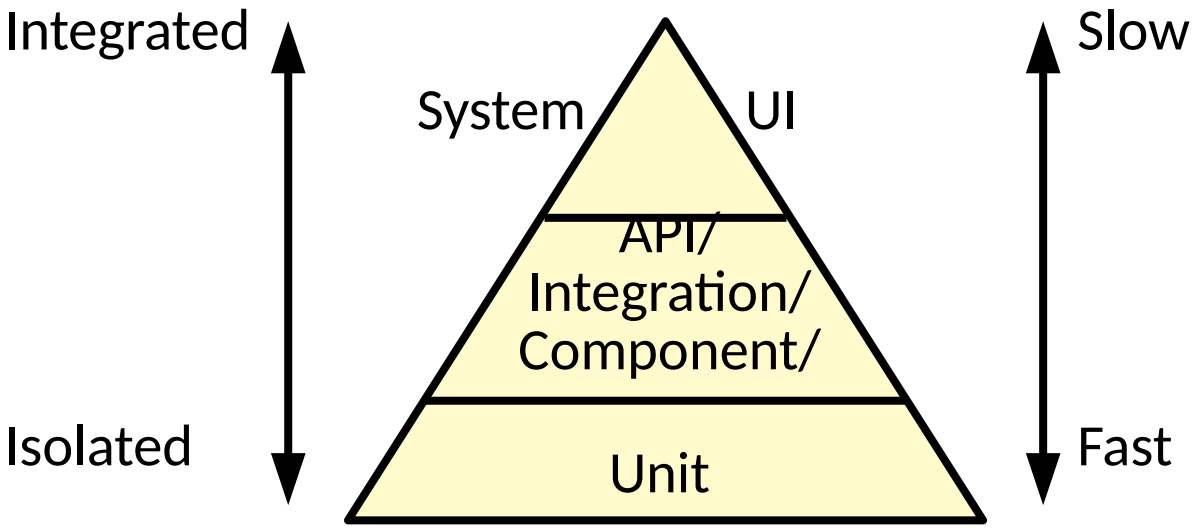
Nick Sumner
wsumner@sfu.ca

How can we automate the top of the pyramid?

- Recall the automated testing pyramid:

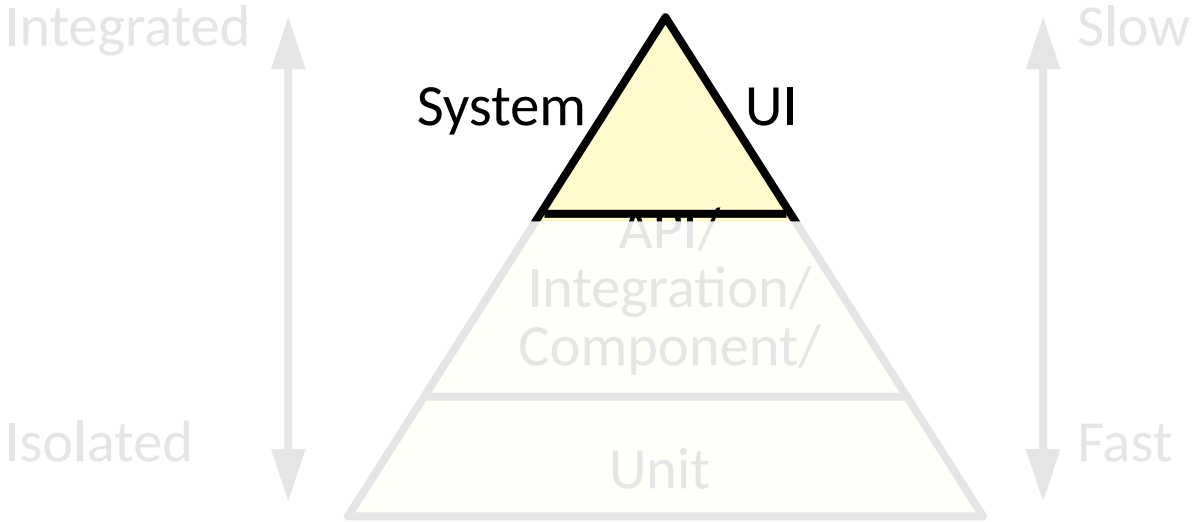
How can we automate the top of the pyramid?

- Recall the automated testing pyramid:



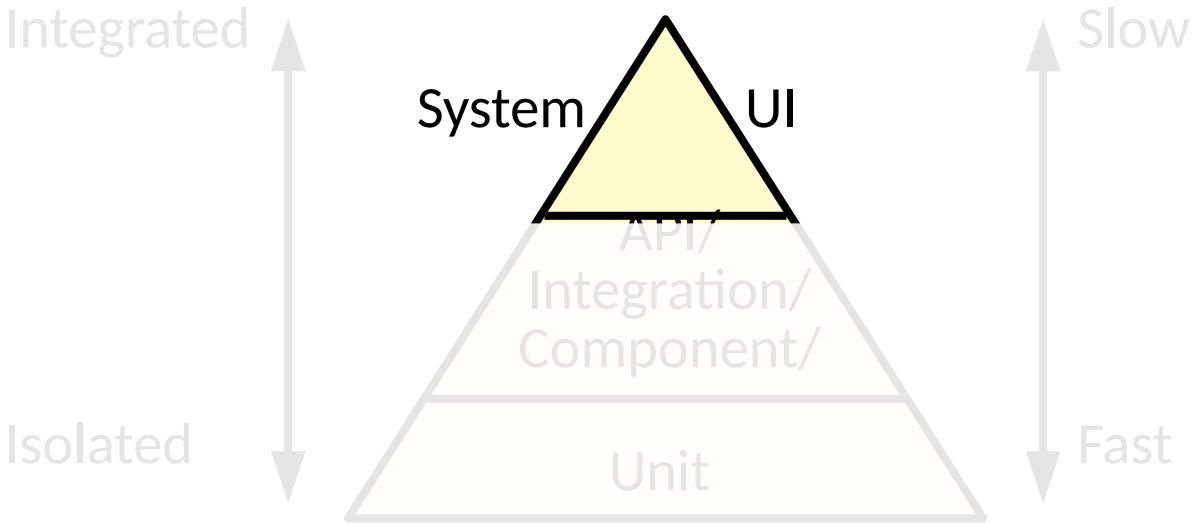
How can we automate the top of the pyramid?

- Recall the automated testing pyramid:



How can we automate the top of the pyramid?

- Recall the automated testing pyramid:
  - The top is: high value, more expensive, challenging to automate
  - But why?!



Challenges

- Think back to the structure of unit tests

Challenges

- Think back to the structure of unit tests

```
TEST_CASE("empty") {
  Environment env;
  ExprTree tree;

  auto result = evaluate(tree, env);

  CHECK(!result.has_value());
}
```

Arrange

Act

Assert

Challenges

- Think back to the structure of unit tests

```
TEST_CASE("empty") {
  Environment env;
  ExprTree tree;

  auto result = evaluate(tree, env);

  CHECK(!result.has_value());
}
```

Arrange

Act

Assert

What implications does testing the UI have for each of these?

Challenges

- Arrange (inputs+scenario)
  - Not a command line or simple API call!

Challenges

- Arrange (inputs+scenario)
  - Not a command line or simple API call!
  - Event based
  - Polyglot & multi system
  - Change: Churn and dynamism
  - Nondeterminism
  - Time matters

Challenges

- Arrange (inputs+scenario)
  - Not a command line or simple API call!
  - Event based
  - Polyglot & multi system
  - Change: Churn and dynamism
  - Nondeterminism
  - Time matters
- Act (running)
  - Nondeterminism
  - Performance

Challenges

- Arrange (inputs+scenario)
  - Not a command line or simple API call!
  - Event based
  - Polyglot & multi system
  - Change: Churn and dynamism
  - Nondeterminism
  - Time matters
- Act (running)
  - Nondeterminism
  - Performance
- Assert (oracles)
  - Nondeterminism
  - Visual results
  - Final vs intermediate states

Challenges

- Arrange (inputs+scenario)
  - Not a command line or simple API call!
  - Event based
  - Polyglot & multi system
  - Change: Churn and dynamism
  - Nondeterminism
  - Time matters
- Act (running)
  - Nondeterminism
  - Performance
- Assert (oracles)
  - Nondeterminism
  - Visual results
  - Final vs intermediate states

And more....

The concerns we had about ***testability*** are only amplified.

We must design UIs to be testable and codesign the testing methods.

UI Testing Frameworks

- Tools to facilitate UI testing will focus on
  - UI Frameworks (e.g. Flutter, React, etc.)
  - Platforms (e.g. Selenium, Robotium, Robot, etc.)

UI Testing Frameworks

- Tools to facilitate UI testing will focus on
  - UI Frameworks (e.g. Flutter, React, etc.)
  - Platforms (e.g. Selenium, Robotium, Robot, etc.)
- These tools provide much needed leverage
  - Can hook into the event system of the UI
  - Synthesize events
  - Programmatic interface

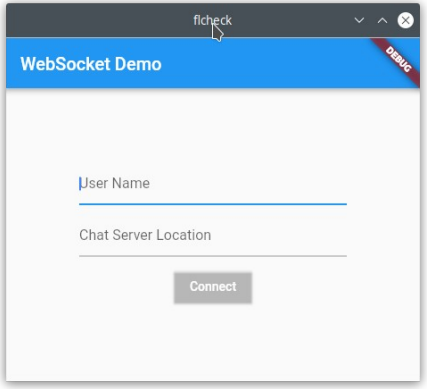
UI Testing Frameworks

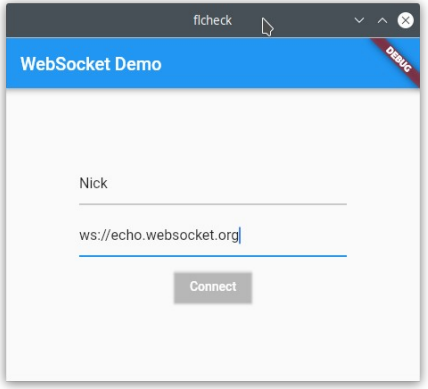
- Tools to facilitate UI testing will focus on
  - UI Frameworks (e.g. Flutter, React, etc.)
  - Platforms (e.g. Selenium, Robotium, Robot, etc.)
- These tools provide much needed leverage
  - Can hook into the event system of the UI
  - Synthesize events
  - Programmatic interface

  - Feed information in
  - Extract information out
  - Provide logical time based on events

Feeding information into a UI

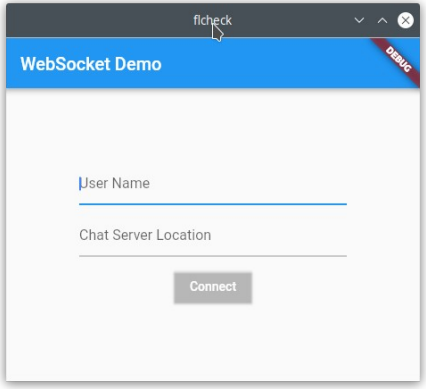
- Let us consider a simple chat program

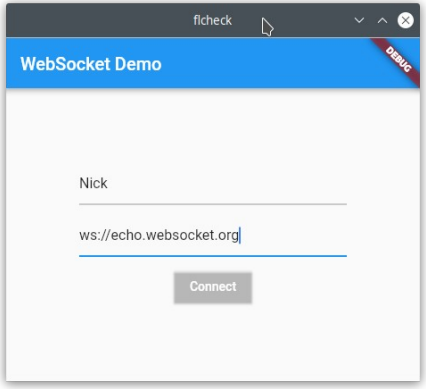


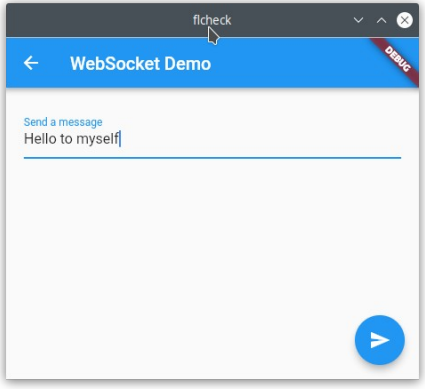


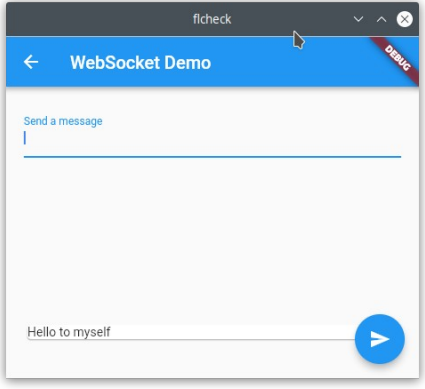
Feeding information into a UI

- Let us consider a simple chat program



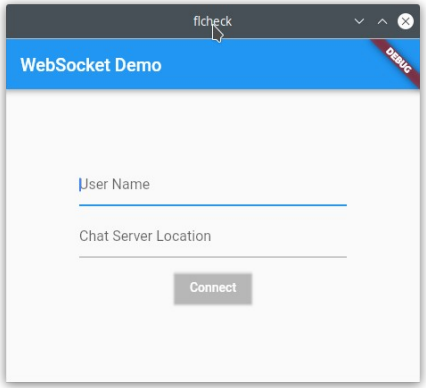


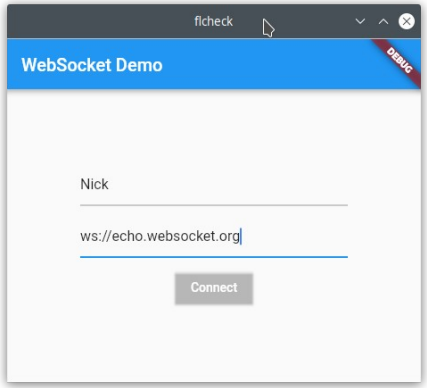


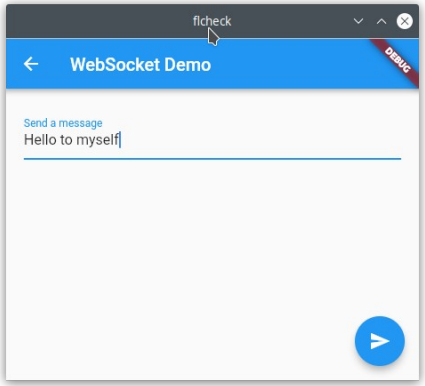


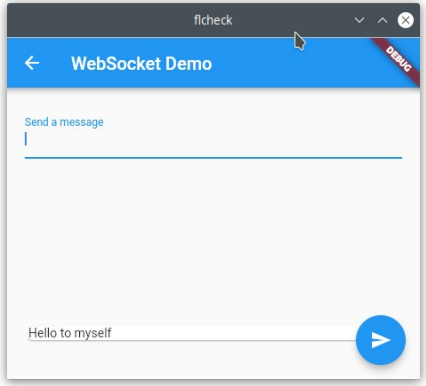
Feeding information into a UI

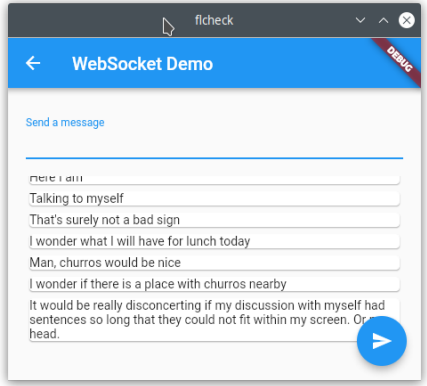
- Let us consider a simple chat program











Feeding information into a UI

- Let us consider a simple chat program
  - What kinds of things make sense to test?

Feeding information into a UI

- Let us consider a simple chat program
  - What kinds of things make sense to test?
  - Wiring and routing
  - Core interactions and user stories
  - …

Feeding information into a UI

- Let us consider a simple chat program
  - What kinds of things make sense to test?
  - Wiring and routing
  - Core interactions and user stories
  - …
- Consider these simple stories
  - A user can
    enter a valid server and click connect
    to reach the chat pane.

Feeding information into a UI

- Let us consider a simple chat program
  - What kinds of things make sense to test?
  - Wiring and routing
  - Core interactions and user stories
  - …
- Consider these simple stories
  - A user can
    enter a valid server and click connect
    to reach the chat pane.
  - A user on the chat pane can
    enter a message in a chat room
    to receive it back in their own chat room display.

Feeding information into a UI

- Let us consider a simple chat program
  - What kinds of things make sense to test?
  - Wiring and routing
  - Core interactions and user stories
  - …
- Consider these simple stories
  - A user can
    enter a valid server and click connect
    to reach the chat pane.

Arrange

Act

Assert

A user on the chat pane can
enter a message in a chat room
to receive it back in their own chat room display.

Finding things to interact with

- There may be several ways you wish to find a component

Finding things to interact with

- There may be several ways you wish to find a component

Finding things to interact with

- There may be several ways you wish to find a component
  - By its ***contents***

Finding things to interact with

- There may be several ways you wish to find a component
  - By its contents
  - By the ***path*** through the UI tree to reach it

Finding things to interact with

- There may be several ways you wish to find a component
  - By its contents
  - By the path through the UI tree to reach it
  - By a ***unique ID***

Finding things to interact with

- There may be several ways you wish to find a component
  - By its contents
  - By the path through the UI tree to reach it
  - By a unique ID
- There are trade offs and use cases for all of these
  - Why can finding by ***contents*** be useful?
  - Why can finding by ***paths*** be useful?
  - Why can finding by ***ID*** be useful?

Finding things to interact with

- There may be several ways you wish to find a component
  - By its contents
  - By the path through the UI tree to reach it
  - By a unique ID
- There are trade offs and use cases for all of these
  - Why can finding by ***contents*** be useful?
  - Why can finding by ***paths*** be useful?
  - Why can finding by ***ID*** be useful?
- But if you plan in advance, you can make your life easier
  - Testability and designing for testing is critical

Finding things to interact with

- There may be several ways you wish to find a component
  - By its contents
  - By the path through the UI tree to reach it
  - By a unique ID
- There are trade offs and use cases for all of these
  - Why can finding by ***contents*** be useful?
  - Why can finding by ***paths*** be useful?
  - Why can finding by ***ID*** be useful?
- But if you plan in advance, you can make your life easier
  - Testability and designing for testing is critical

Managing IDs well helps to deal with churn and evolution

Finding things to interact with

- Manging IDs in Flutter

```
final serverField = TextField(
  key: ValueKey("ServerField"),
  controller: _serverController,
  onSubmitted: _connectToServer,
  obscureText: false,
  autofocus: true,
);
```

Finding things to interact with

- Manging IDs in Flutter

```
final serverField = TextField(
  key: ValueKey("ServerField"),
```

```
test('Connects to echo server and receives message', () async {
  final serverFinder   = find.byValueKey('ServerField');
  final connectFinder  = find.byValueKey('ConnectButton');
  final messageFinder  = find.byValueKey('MessageField');
  final sendFinder     = find.byValueKey('SendButton');
  final receivedFinder = find.byValueKey('Message(0)');
...
```

Feeding information into a UI

- The UI framework will provide functionality for time, events, and data:
  - Text entry
  - Button presses
  - Gestures
  - …

Feeding information into a UI

- The UI framework will provide functionality for time, events, and data:
  - Text entry
  - Button presses
  - Gestures
  - …
- Tests then run sequences of simulated events

Feeding information into a UI

- The UI framework will provide functionality for time, events, and data:
  - Text entry
  - Button presses
  - Gestures
  - …
- Tests then run sequences of simulated events

```
test('Connects to echo server and receives message', () async {
  final serverFinder   = find.byValueKey('ServerField');
  final connectFinder  = find.byValueKey('ConnectButton');
  final messageFinder  = find.byValueKey('MessageField');
  final sendFinder     = find.byValueKey('SendButton');
  final receivedFinder = find.byValueKey('Message(0)');
...
```

Feeding information into a UI

- The UI framework will provide functionality for time, events, and data:
  - Text entry
  - Button presses
  - Gestures
  - …
- Tests then run sequences of simulated events

```
test('Connects to echo server and receives message', () async {
  final serverFinder   = find.byValueKey('ServerField');
  f                                                    ;
  f
  f
  f
...
```

```
...
  // Enter an echo server into the server field.
  await driver.tap(serverFinder);
  await driver.enterText('ws://echo.websocket.org');
...
```

Dealing with time

- Why is time a problem?
  - Nondeterminism
  - Latency
  - Cost

Dealing with time

- Why is time a problem?
  - Nondeterminism
  - Latency
  - Cost
- All of these can be dealt with to some degree
  - Tolerate
  - Abstract away

Dealing with time

- Do we always care about real time?

Dealing with time

- Do we always care about real time?
  - “Wait for the UI to update in response to my action.”
  - “Wait for the server to respond to my request.”
  - “First click this, then click that after the first response was processed”

Dealing with time

- Do we always care about real time?
  - "Wait for the UI to update in response to my action."
  - "Wait for the server to respond to my request."
  - "First click this, then click that after the first response was processed"
- In many cases, time can be abstracted to ***ordered events***.
  - ***Don't*** "wait X seconds"
  - ***Do*** "wait until the page loads"

Dealing with time

- Do we always care about real time?
  - “Wait for the UI to update in response to my action.”
  - “Wait for the server to respond to my request.”
  - “First click this, then click that after the first response was processed”
- In many cases, time can be abstracted to ***ordered events***.
  - ***Don’t*** “wait X seconds”
  - ***Do*** “wait until the page loads”
  - (Unless your framework doesn’t support it….)

Dealing with time

- Do we always care about real time?
  - “Wait for the UI to update in response to my action.”
  - “Wait for the server to respond to my request.”
  - “First click this, then click that after the first response was processed”
- In many cases, time can be abstracted to ***ordered events***.
  - ***Don’t*** “wait X seconds”
  - ***Do*** “wait until the page loads”
  - (Unless your framework doesn’t support it….)
- When this option is available to you it is more robust
  - To change, to nondeterminism, …

Dealing with time

- What if we don’t have a choice?

Dealing with time

- What if we don’t have a choice?
  - You can fall back to time thresholded waits, but you should ***expect***:

Dealing with time

- What if we don't have a choice?
  - You can fall back to time thresholded waits,
    but you should ***expect***: flaky tests,
    higher maintenance costs
    developer ambivalence

Dealing with time

- What if we don't have a choice?
  - You can fall back to time thresholded waits,
    but you should ***expect***: flaky tests,
    higher maintenance costs
    developer ambivalence
  - It is a cost/benefit decision

Dealing with time

- What if we don't have a choice?
  - You can fall back to time thresholded waits,
    but you should ***expect***: flaky tests,
    higher maintenance costs
    developer ambivalence
  - It is a cost/benefit decision
- In Flutter

Dealing with time

- What if we don't have a choice?
  - You can fall back to time thresholded waits,
    but you should ***expect***: flaky tests,
    higher maintenance costs
    developer ambivalence
  - It is a cost/benefit decision
- In Flutter

```
await tester.pumpWidget(MyWidget(title: 'T', message: 'M'));

await tester.enterText(find.byValueKey('greeting'), 'hi');
await tester.tap(find.byValueKey('confirm'));

await tester.pump(Duration.zero);

expect(...)
```

Dealing with time

- What if we don't have a choice?
  - You can fall back to time thresholded waits,
    but you should ***expect***: flaky tests,
    higher maintenance costs
  - It i
- In Flut

```
Future<bool>
isPresent(SerializableFinder byValueKey,
          FlutterDriver driver,
          {Duration timeout = const Duration(seconds: 1)}) async {
  try {
    await driver.waitFor(byValueKey, timeout: timeout);
    return true;
  } catch(exception) {
    return false;
  }
}
```

```
await teste

await teste
await teste

await teste

expect(...)
```

```
expect(await isPresent(messageFinder, driver), true);
```

Revisiting the Chat App (for 1 story)

```
final serverFinder   = find.byValueKey('ServerField');
final connectFinder  = find.byValueKey('ConnectButton');
final messageFinder  = find.byValueKey('MessageField');
final sendFinder     = find.byValueKey('SendButton');
final receivedFinder = find.byValueKey('Message(0)');
final message = 'Hi, there!';
```

Arrange

Revisiting the Chat App (for 1 story)

```
final
final
final
final
final
final
```

Arrange

```
// Enter an echo server into the server field.
await driver.tap(serverFinder);
await driver.enterText('ws://echo.websocket.org');

// Tap the connect button to reach the
await driver.tap(connectFinder);

// Wait for the next page to load
expect(await isPresent(messageFinder, driver), true);

// Enter a message into the message field
await driver.tap(messageFinder);
await driver.enterText(message);
await driver.tap(sendFinder);

// Wait for a response to be triggered
expect(await isPresent(receivedFinder, driver), true);
expect(await driver.getText(receivedFinder), message);
```

Act

Assert

Revisiting the Chat App (for 1 story)

```
final
final
final
final
final
final
```

Arrange

```
// Enter an echo server into the server field.
await driver.tap(serverFinder);
await driver.enterText('ws://echo.websocket.org');

// Tap the connect button to reach the
await driver.tap(connectFinder);

// Wait for the next page to load
expect(await isPresent(messageFinder, driver), true);

// Enter a message into the message field
await driver.tap(messageFinder);
await driver.enterText(message);
await driver.tap(sendFinder);

// Wait for a response to be triggered
expect(await isPresent(receivedFinder, driver), true);
expect(await driver.getText(receivedFinder), message);
```

Act

Assert

Practical Concerns

- What do you actually want to test?
  - Just the front end?
  - The full system?

Practical Concerns

- What do you actually want to test?
  - Just the front end?
  - The full system?

  - You can reduce costs & decrease flakiness by faking the backend!

Practical Concerns

- What do you actually want to test?
  - Just the front end?
  - The full system
  - You can reduce costs & decrease flakiness by faking the backend!
- Who should actually be creating the tests?
  - Acceptance level by client?
  - System level by a developer?

Practical Concerns

- What do you actually want to test?
  - Just the front end?
  - The full system
  - You can reduce costs & decrease flakiness by faking the backend!
- Who should actually be creating the tests?
  - Acceptance level by client?
  - System level by a developer?
  - *The person defining the tests may not be a programmer!*

Recording vs scripting

- For precise control & using IDs well, you may hand write tests

Recording vs scripting

- For precise control & using IDs well, you may hand write tests
  - But it is not necessarily required!

Recording vs scripting

- For precise control & using IDs well, you may hand write tests
  - But it is not necessarily required!
- Tools like ***Selenium*** can record user interactions as an event series

Recording vs scripting

- For precise control & using IDs well, you may hand write tests
  - But it is not necessarily required!
- Tools like ***Selenium*** can record user interactions as an event series
  - A trace of (Command, Target, Value)s
  - Can be replayed

Recording vs scripting

- For precise control & using IDs well, you may hand write tests
  - But it is not necessarily required!
- Tools like Selenium can record user interactions as an event series
  - A trace of (Command, Target, Value)s
  - Can be replayed
- This can make it easier to produce tests for nonexperts, but recorded tests can be more brittle

BDD

- *Behavior Driven Development* (*BDD*) tools provide another route for non-programmers to define tests

BDD

- *Behavior Driven Development* (*BDD*) tools provide another route for non-programmers to define tests
- Originated as a way to facilitate collaboration between business & developer experts

BDD

- *Behavior Driven Development* (*BDD*) tools provide another route for non-programmers to define tests
- Originated as a way to facilitate collaboration between business & developer experts
  - User stories given in natural language with common structure

BDD

- *Behavior Driven Development* (*BDD*) tools provide another route for non-programmers to define tests
- Originated as a way to facilitate collaboration between business & developer experts
  - User stories given in natural language with common structure
  - ***Given*** some initial context
    When some event occurs
    Then ensure some outcome

BDD

- *Behavior Driven Development* (*BDD*) tools provide another route for non-programmers to define tests
- Originated as a way to facilitate collaboration between business & developer experts
  - User stories given in natural language with common structure
  - ***Given*** some initial context
    When some event occurs
    Then ensure some outcome

Scenario: Breaker joins a game
Given the Maker has started a game with the word "silky"
When the Breaker joins the Maker's game
Then the Breaker must guess a word with 5 characters

[Cucumber.io Docs]

BDD

- *Behavior Driven Development* (*BDD*) tools provide another route for non-programmers to define tests
- Originated as a way to facilitate collaboration between business & developer experts
  - User stories given in natural language with common structure
  - ***Given*** some initial context
    When some event occurs
    Then ensure some outcome

Scenario: Breaker joins a game
Given the Maker has started a game with the word "silky"
When the Breaker joins the Maker's game
Then the Breaker must guess a word with 5 characters

[Cucumber.io Docs]

- Tools like Cucumber can translate these into, e.g., Selenium tests

Further Directions

- We have only considered automated *functional* UI testing

Further Directions

- We have only considered automated functional UI testing
- We could also consider
  - User Experience (UX)
  - Performance
  - Security
  - Regulatory compliance
  - Exploratory methods
  - Automated test generation

Summary

- UI testing adds many challenges on top of test automation

Summary

- UI testing adds many challenges on top of test automation
- Frameworks can intercept behavior to facilitate easier test construction

Summary

- UI testing adds many challenges on top of test automation
- Frameworks can intercept behavior to facilitate easier test construction
- Careful design of code to be testable is just as important in this setting.