Atrina: Inferring Unit Oracles from GUI Test Cases

Shabnam Mirshokraie
Ali Mesbah
Karthik Pattabiraman

University of British Columbia
```javascript
function addToCart()
{
  item = getItemInfo($(".merchandise"));
  for (var i=0; i<availableItems.length; i++)
    availableItems[i].count-=item.quantity;
  var price = item.price * item.quantity;
  if (!coupon.expired)
    $("#coupon").removeClass('ready');
    price -= $("#coupon").data('value');
    $("#coupon").addClass('used');
  customer.payable += price;
}

function showCart()
{
  $("div.shopContainer").append
    ("<p>" + GRAND TOTAL + customer.payable +"</p>");
}
```

```
@Test
class TestShopApplication {
  @Test
  public void testShopContainer() {
    WebElement item = driver.findElements(By.css(".merchandise"));
    item.click();
    WebElement cart = driver.findElements(By.id("showCart"));
    cart.click();
    String expectedMsg = "GRAND TOTAL 622.90";
    String msg = driver.findElements(By.cssSelector("div.shopContainer")).getText();
    assertEquals(msg, expectedMsg);
  }
}
```
Testing JavaScript Applications

DOM-based test

Unit-level tests

Jasmine

Qu

mocha
Testing JavaScript Applications

- Knowledge of App Internal Operations
- Ease of Fault Localization

- DOM-based test
- Unit-level tests
- Selenium
- Jasmine
- QuUnit
- Mocha
Testing JavaScript Applications

Can we combine the advantages of both techniques?
Testing JavaScript Applications

- Knowledge of App Internal Operations
- Ease of Fault Localization

DOM-based test

Our Goal

Unit-level tests

Jasmine

Se

Qu

mocha
Using existing DOM-based tests as a guide for producing unit-level assertions:

- Towards more important characteristics of the application from the tester points of view
- Prevents creating test cases with too many assertions
Code-level assertions based on human written DOM-based tests

- **Explicit assertions**
  - Directly inferred from analyzing the manually written DOM-based assertions

- **Implicit assertions**
  - Indirectly inferred from the human written DOM-based assertions

- **Candidate assertions**
  - Not considered in the written DOM-based assertions, yet are useful for fault detection
function addToCart() {
    item = getItemInfo($(".merchandise"));
    for (var i = 0; i < availableItems.length; i++)
        availableItems[i].count -= item.quantity;
    var price = item.price * item.quantity;
    if (!coupon.expired) {
        $("#coupon").removeClass('ready');
        price -= $("#coupon").data('value');
        $("#coupon").addClass('used');
    }
    customer.payable += price;
}

function showCart() {
    $("div.shopContainer").append("<p>" + "GRAND TOTAL $" + customer.payable +"</p>");
}
**Intra DOM assertion dependency:** DOM elements in the test case pertaining to the assertion
```java
@Test
class TestApp {
    public void testShopContainer() {
        WebElement item = driver.findElements(By.css(".merchandise"));
        item.click();
        WebElement cart = driver.findElements(By.id("showCart"));
        cart.click();
        String expectedMsg = "GRAND TOTAL $622.90";
        String msg = driver.findElements(By.cssSelector("div.shopContainer")).getText();
        assertEquals(msg, expectedMsg);
    }
}
```
**Inter DOM assertion dependency:** Initial point of contact between the application’s code and updated DOM in the test
**Test Code**

```java
@Test
public void testShopContainer() {
    WebElement item = driver.findElements(By.css(".merchandise"));
    item.click();
    WebElement cart = driver.findElements(By.id("showCart"));
    cart.click();
    String expectedMsg = "GRAND TOTAL $622.90";
    String msg = driver.findElements(By.cssSelector("div.shopContainer")).getText();
    assertEquals(msg, expectedMsg);
}
```

**JavaScript Code**

```javascript
$("#coupon").addClass('used');
}
customer.payable += price;
}
function showCart() {
...
$("div.shopContainer").append("<p>" + "GRAND TOTAL "$ +
customer.payable +"<p>");
...
}
```

**Criterion:** Variable at the initial point of contact

**Intra code dependency:**
- Backward and Forward Slicing
- Data and control dependent statements on criterion
```javascript
function addToCart() {
  item = getItemInfo($(".merchandise"));
  for (var i = 0; i < availableItems.length; i++)
    availableItems[i].count -= item.quantity;
  var price = item.price * item.quantity;
  if (!coupon.expired) {
    $("#coupon").removeClass('ready');
    price -= $("#coupon").data('value');
    $("#coupon").addClass('used');
  }
  customer.payable += price;
}

function showCart() {
  $("div.shopContainer").append("<p>GRAND TOTAL $" + customer.payable + "</p>");
}
```
function addToCart() {
    item = getItemInfo($(".merchandise"));
    for (var i=0; i<availableItems.length; i++)
        availableItems[i].count -= item.quantity;
    var price = item.price * item.quantity;
    if (!coupon.expired) {
        $("#coupon").removeClass('ready');
        price -= $("#coupon").data('value');
        $("#coupon").addClass('used');
    }
    customer.payable += price;
}

function showCart() {
    $("div.shopContainer").append("<p>GRAND TOTAL $" + customer.payable +"</p>");
}
function addToCart() {
    item = getItemInfo($(".merchandise"));
    for (var i=0; i<availableItems.length; i++)
        availableItems[i].count -= item.quantity;
    var price = item.price * item.quantity;
    if (!coupon.expired) {
        $("#coupon").removeClass('ready');
        price -= $("#coupon").data('value');
        $("#coupon").addClass('used');
    }
    customer.payable += price;
}

function showCart() {
    $("div.shopContainer").append("<p> GRAND TOTAL $" + customer.payable +"</p>");
}
function addToCart() {
  item = getItemInfo($(".merchandise"));
  for (var i=0; i<availableItems.length; i++)
    availableItems[i].count -= item.quantity;
  var price = item.price * item.quantity;
  if (!coupon.expired) {
    $("#coupon").removeClass('ready');
    price -= $("#coupon").data('value');
    $("#coupon").addClass('used');
  }
  customer.payable += price;
}

function showCart() {
  $("div.shopContainer").append("<p> GRAND TOTAL $" + customer.payable +"</p>");
}
function addToCart() {
  item = getItemInfo($(".merchandise"));
  for (var i = 0; i < availableItems.length; i++)
    availableItems[i].count -= item.quantity;
  var price = item.price * item.quantity;
  if (!coupon.expired) {
    $('#coupon').removeClass('ready');
    price -= $('#coupon').data('value');
    $('#coupon').addClass('used');
  }
  customer.payable += price;
}

function showCart() {
  $('div.shopContainer').append('<p>GRAND TOTAL $' + customer.payable + '</p>');
}
```javascript
function addToCart() {
  item = getItemInfo($(".merchandise"));
  for (var i=0; i<availableItems.length; i++)
    availableItems[i].count -= item.quantity;
  var price = item.price * item.quantity;
  if (!coupon.expired) {
    $("#coupon").removeClass('ready');
    price -= $("#coupon").data('value');
    $("#coupon").addClass('used');
  }
  customer.payable += price;
}

function showCart() {
  $("div.shopContainer").append("<p>GRAND TOTAL $" + customer.payable +"</p>" didnt hit it);
}
```
function addToCart() {
  item = getItemInfo($(".merchandise"));
  for (var i=0; i<availableItems.length; i++)
    availableItems[i].count -= item.quantity;
  var price = item.price * item.quantity;
  if (!coupon.expired) {
    $("#coupon").removeClass('ready');
    price -= $("#coupon").data('value');
    $("#coupon").addClass('used');
  }
  customer.payable += price;
}

function showCart() {
  $("div.shopContainer").append("<p>" + GRAND TOTAL $" + customer.payable +"</p>");
}
Unit-level Assertion Type  
Implicit Assertions

```javascript
function addToCart() {
    item = getItemInfo($('.merchandise'));
    var price = item.price * item.quantity;

    for (var i=0; i<availableItems.length; i++)
        availableItems[i].count -= item.quantity;

    if (!coupon.expired) {
        $('#coupon').removeClass('ready');
        price -= $('#coupon').data('value');
        $('#coupon').addClass('used');
    }

    customer.payable += price;
}

function showCart() {
    $('#div.shopContainer').append('<p>' + 'GRAND TOTAL $' + customer.payable + '</p>');
}
```
function addToCart() {
    item = getItemInfo($(".merchandise")); 
    for (var i=0; i<availableItems.length; i++)
        availableItems[i].count -= item.quantity;
    var price = item.price * item.quantity;
    if (!coupon.expired) {
        $("#coupon").removeClass('ready');
        price -= $("#coupon").data('value');
        $("#coupon").addClass('used');
    }
    customer.payable += price;
}

function showCart() {
    $("div.shopContainer").append("<p>" + "GRAND TOTAL $" + customer.payable +"</p>"));
}

Candidate DOM elements: Frequently accessed DOM elements
function addToCart() {
  item = getItemInfo($(".merchandise"));
  for (var i=0; i<availableItems.length; i++)
    availableItems[i].count -= item.quantity;
  var price = item.price * item.quantity;
  if (!coupon.expired) {
    $("#coupon").removeClass('ready');
    price -= $("#coupon").data('value');
    $("#coupon").addClass('used');
  }
  customer.payable += price;
}

function showCart() {
  $("div.shopContainer").append("<p>" + GRAND TOTAL $" + customer.payable +"</p>"STORELOCATION);
test ("addToCart", function() {
...

  var customer = {id:"10", payable:0};
  var coupon = {id:"1", expired:false}
  var availableItems = [{name:"jacket", price:760, count:2}]
  var item = {name:"", price:0, quantity:0};

  addToCart();

  equal(customer.payable, 622.90);
  ok(coupon.expired);
  deepEqual(item, {price:760, quantity:1});
  equal(availableItems[0].count, 1);
  ok($("#coupon").hasClass("used"));
  notOk($("#coupon").hasClass("ready"));
});
Generated QUnit Test

test ("addToCart", function() {

  ...

  var customer = {id:"10", payable:0};
  var coupon = {id:"1", expired:false}
  var availableItems = [{name:"jacket", price:760, count:2}]
  var item = {name:"", price:0, quantity:0};

  addToCart();

  equal(customer.payable, 622.90);
  ok(coupon.expired);
  deepEqual(item, {price:760, quantity:1});

  equal(availableItems[0].count, 1);

  ok($("#coupon").hasClass("used"));
  ok($("#coupon").hasClass("ready"));
});
Evaluation and Results

• How accurate is Atrina in mapping DOM-based assertions to the corresponding JavaScript code?

• How effective is our tool in terms of fault detection capability?

• Are the assertions generated by Atrina more effective than DOM-based assertions written by testers?

• How does Atrina compare to existing mutation-based techniques for generating unit test assertions?
Seven open source JavaScript web applications that have Selenium test cases

(range from 0.8K to 57K lines of JavaScript code)

Fault injection to evaluate the ability of the tool in detecting seeded faults

(50 random first-order mutations into the JavaScript code of the applications)
Accuracy in mapping DOM-based assertions to the JavaScript code

Inaccurate computation of slices result in:
- Generating unrelated assertions
- Failing to produce useful assertions
Accuracy in mapping DOM-based assertions to the JavaScript code

Inaccurate computation of slices result in:
- Generating unrelated assertions
- Failing to produce useful assertions

Precision: 99%
Recall: 92%
Accuracy in mapping DOM-based assertions to the JavaScript code

Inaccurate computation of slices result in:
- Generating unrelated assertions
- Failing to produce useful assertions

Precision: 99%
Recall: 92%
Accuracy in mapping DOM-based assertions to the JavaScript code

Inaccurate computation of slices result in:
- Generating unrelated assertions
- Failing to produce useful assertions

Precision: 99%  
Recall: 92%
Accuracy in mapping DOM-based assertions to the JavaScript code

Inaccurate computation of slices result in:
- Generating unrelated assertions
- Failing to produce useful assertions

Precision: 99%
Recall: 92%

DOM-based assertions that are not connected to the JavaScript code:
- HTML is used to transfer the data
- Web server is utilized to perform computations
- HTML fragments are retrieved from the server and injected into the page
- CSS and HTML are used to perform required changes to the user interface

Functions that are called but not instrumented
We do not consider 3rd party libraries in our analysis
Effectiveness in terms of fault detection capability

Fault Detection Rate (%)

Experimental Objects

Atrina  Explicit Only  Explicit+Implicit  Explicit+Candidate
Effectiveness in terms of fault detection capability

Atrina detects on average 63% of the total faults
Effectiveness in terms of fault detection capability

Atrina detects on average 63% of the total faults

Explicit assertions detect 76% of the total detected faults on average
Effectiveness in terms of fault detection capability

Atrina detects on average 63% of the total faults

Explicit assertions contribute the most among the three assertion types
Effectiveness in terms of fault detection capability

Atrina detects on average 63% of the total faults

Explicit assertions contribute the most among the three assertion types

% of faults detected by explicit assertions < combination of explicit with implicit or candidate
Effectiveness in terms of fault detection capability

Atrina detects on average 63% of the total faults.

Explicit assertions contribute the most among the three assertion types.

Implicit and candidate assertions are essential in improving the fault finding capability.
Effectiveness in terms of fault detection capability

Atrina detects on average 63% of the total faults

Explicit assertions contribute the most among the three assertion types

Implicit and candidate assertions are essential in improving the fault finding capability

Fault detection improvement by implicit assertions < Fault detection improvement by candidate assertions
Effectiveness in terms of fault detection capability

Atrina detects on average 63% of the total faults
Explicit assertions contribute the most among the three assertion types
Implicit and candidate assertions are essential in improving the fault finding capability
Candidate assertions play a more prominent role in increasing the number of detected faults
Comparison with human-written DOM-based assertions

- Atrina outperforms manual assertions in terms of fault finding capability by 31%.
Atrina outperforms manual assertions in terms of fault finding capability by 31%.

More than 60% of the faults found by implicit assertions are neither detected by explicit/candidate assertions nor by the human-written ones.
Atrina outperforms manual assertions in terms of fault finding capability by 31%.

More than 60% of the faults found by implicit assertions are neither detected by explicit/candidate assertions nor by the human-written ones.

They require executing a more complex sequence of events to propagate to the DOM.
Atrina outperforms manual assertions in terms of fault finding capability by 31%.

More than 60% of the faults found by implicit assertions are neither detected by explicit/candidate assertions nor by the human-written ones.

DOM property that is checked in the human-written test is later used in JavaScript code that involves internal computations only.
Atrina outperforms manual assertions in terms of fault finding capability by 31%

More than 60% of the faults found by implicit assertions are neither detected by explicit/candidate assertions nor by the human-written ones

More than 50% of the selected DOM properties in Atrina were ignored in human-written DOM assertions
Comparison with mutation-based assertions

- Atrina outperforms mutation-based assertions in terms of fault finding capability by 26%
Atrina outperforms mutation-based assertions in terms of fault finding capability by 26%.

Importance of incorporating the information exists in human-written DOM-based test cases.
Comparison with mutation-based assertions (Time overhead)

Overhead of Atrina:
Instrumentation + Slice Computations

Overhead of the mutation-based approach:
Running the Tests + Generating Mutants + Compare the Original and the Mutated Version
Comparison with mutation-based assertions (Time overhead)

Overhead of Atrina: 47 (Sec)

Overhead of the mutation-based approach: 98 (Sec)

Atrina significantly outperforms mutation-based assertion generation in terms of time efficiency
Summary

Inter DOM assertion dependency: initial point of contact between the application’s code and updated DOM in the test

Effectiveness in terms of fault detection capability

Atrina detects on average 63% of the total faults
Explicit assertions contribute the most among the three assertion types
Implicit and candidate assertions are essential in improving the fault finding capability
Candidate assertions play a more prominent role in increasing the number of detected faults

Comparison with human-written DOM-based assertions

Atrina outperforms manual assertions in terms of fault finding capability by 31%
More than 60% of the faults found by implicit assertions are neither detected by explicit/candidate assertions nor by the human-written ones
More than 50% of the selected DOM properties in Atrina were ignored in human-written DOM assertions

Comparison with mutation-based DOM assertions

Atrina outperforms mutation-based assertions in terms of fault finding capability by 26%
Importance of incorporating the information exists in human-written DOM-based test cases

Atrina
Explicit Only
Explicit+Implicit
Explicit+Candidate

Fault Detection Rate (%)