Atrina: Inferring Unit Oracles from GUI Test Cases

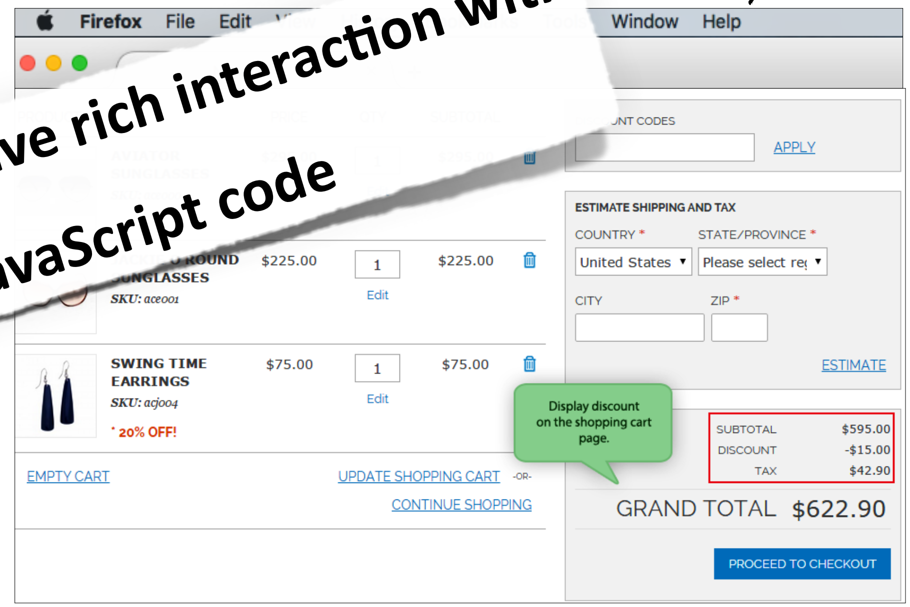
Shabnam Mirshokraie
Ali Mesbah
Karthik Pattabiraman

University of British Columbia

```
@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);
}
```

Test Code

DOM



Testing web applications that have rich interaction with the DOM through JavaScript code

```
function addToC
    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 Code

Testing JavaScript Applications

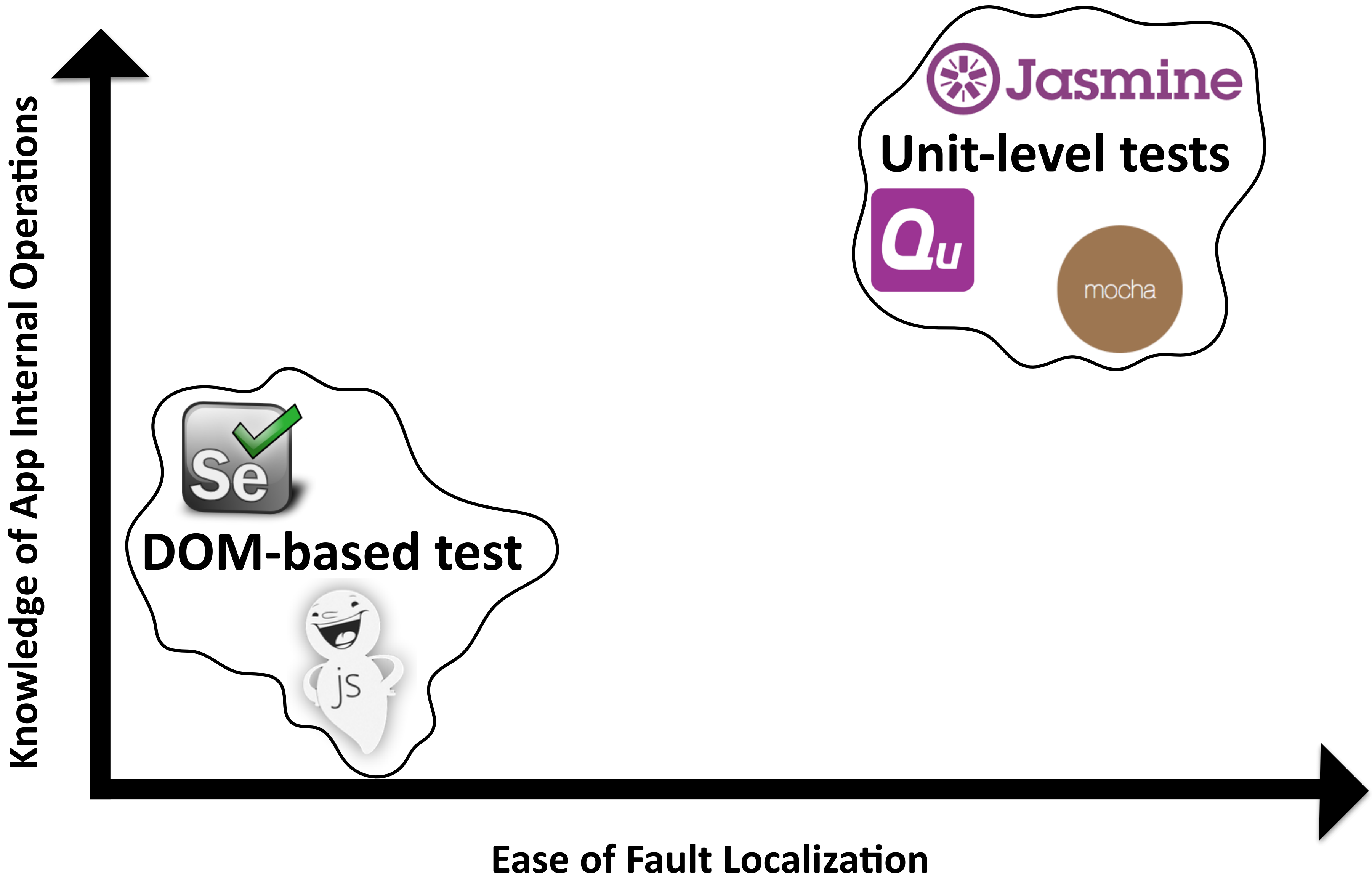
Jasmine

Unit-level tests

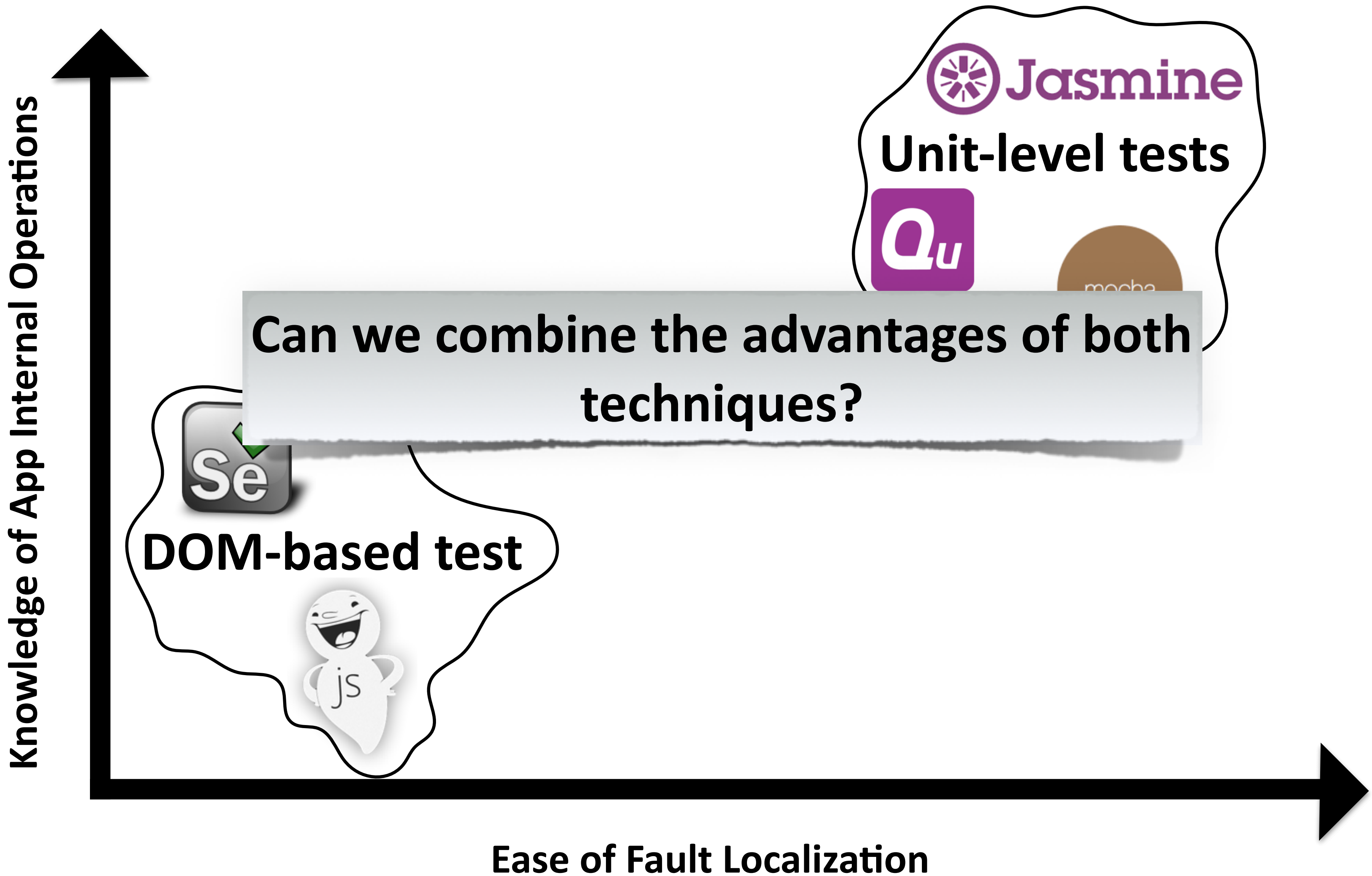
mocha

DOM-based test

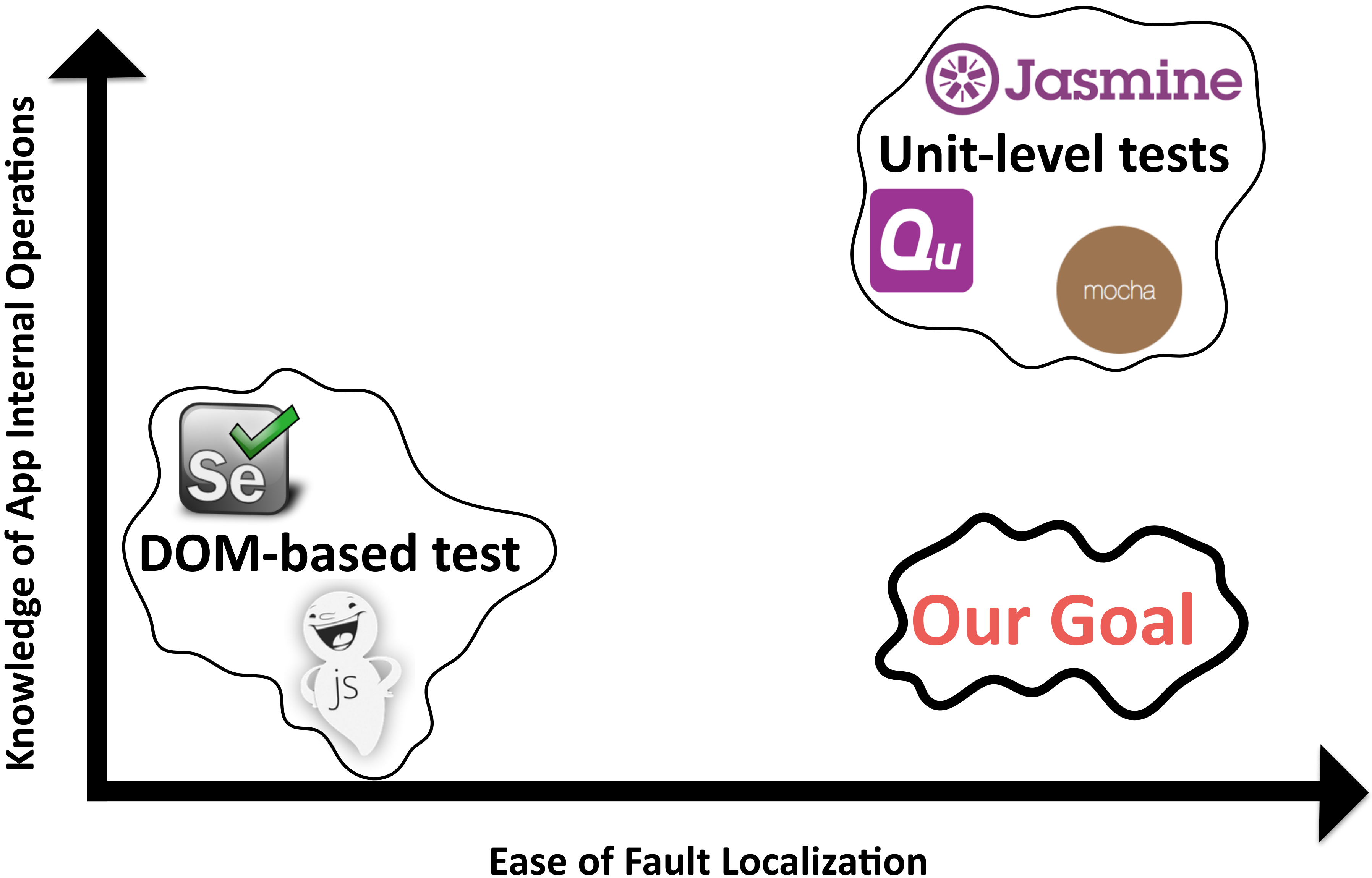
Testing JavaScript Applications



Testing JavaScript Applications



Testing JavaScript Applications



Atrina: Automated unit-level test assertion generation

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

Atrina: Automated unit-level test assertion generation

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

Test Code

```
@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 [shopping cart application]

```
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 Code

```
@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);
}
```

1

JavaScript Code

```
    ...
    $("#coupon").addClass('used');
  }
  customer.payable += price;
}

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

1 **Intra DOM assertion dependency**: DOM elements in the test case pertaining to the assertion

Test Code

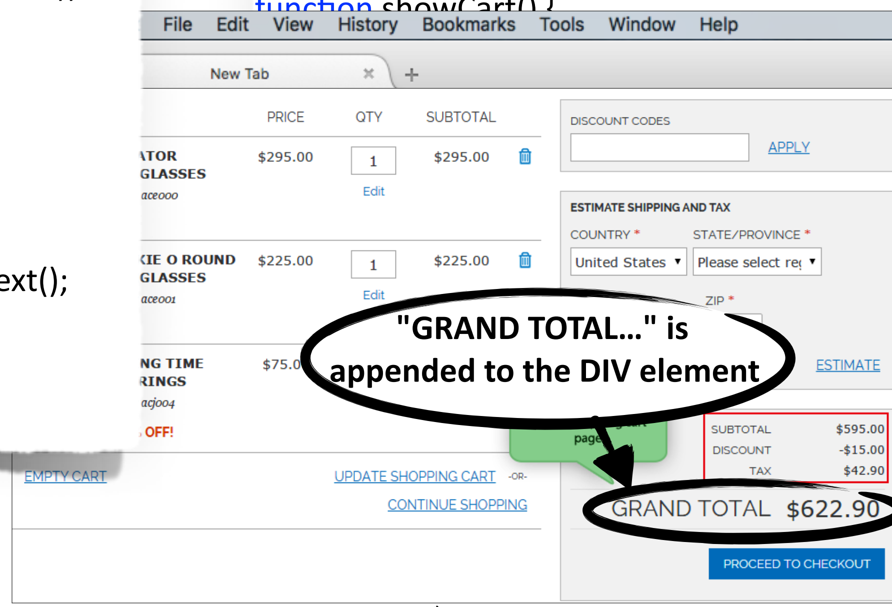
JavaScript Code

```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);
}
```

1

```javascript
    ...
    $("#coupon").addClass('used');
  }
  customer.payable += price;
}

function showCart() {
RAND TOTAL $" +
>");
```



DOM

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);
}
```

1

2

JavaScript Code

```javascript
    ...
    $("#coupon").addClass('used');
  }
  customer.payable += price;
}

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

2 **Inter DOM assertion dependency**: Initial point of contact between the application's code and updated DOM in the test

Test Code

```
@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);
}
```

1

2

JavaScript Code

```
     ...
     $("#coupon").addClass('used');
   }
   customer.payable += price;
}

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

3

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

3

- **Intra code dependency**:
    - Backward and Forward Slicing
    - Data and control dependent statements on criterion

JavaScript Code

```
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>");
}
```

Backward Slicing

JavaScript Code

```
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>");
}
```

Data and control dependent statements that have some effect on the criterion

JavaScript Code

```
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>");
}
```

Data and control dependent statements that have some effect on the criterion

JavaScript Code

```
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>");
}
```

Data and control dependent statements that have some effect on the criterion

Unit-level Assertion Type 1
Explicit Assertions

```
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 Code

```
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>");
}
```

Forward Slicing

JavaScript Code

```
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>");
}
```

Data and control dependent statements affected by the criterion

Unit-level Assertion Type 2
Implicit Assertions

```
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 Code

```
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>");
}
```

#coupon element

Candidate DOM elements: Frequently accessed DOM elements

Unit-level Assertion Type 3
Candidate Assertions

```
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>");
}
```

#coupon element class attribute

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($("#coupon").hasClass("used"));
    notOk($("#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($("#coupon").hasClass("used"));
    notOk($("#coupon").hasClass("ready"));
});
```

Explicit Assertions

Implicit Assertion

Candidate Assertions

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 detetcing 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%

Functions that are called but not instrumented

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%

Functions that are called but not instrumented

Recall: 92%

We do not consider 3rd party libraries in our analysis

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%

Functions that are called but not instrumented

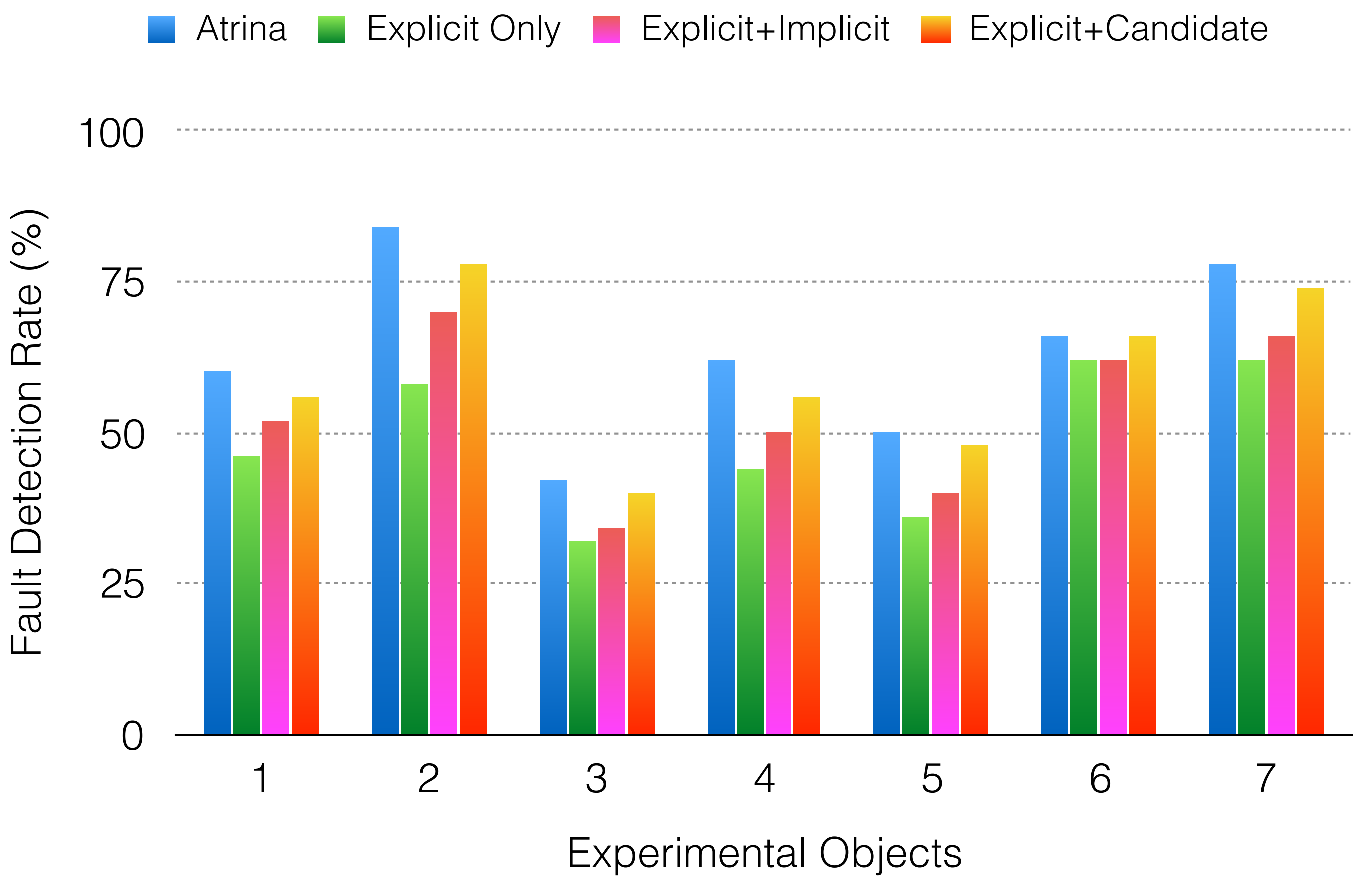
Recall: 92%

We do not consider 3rd party libraries in our analysis

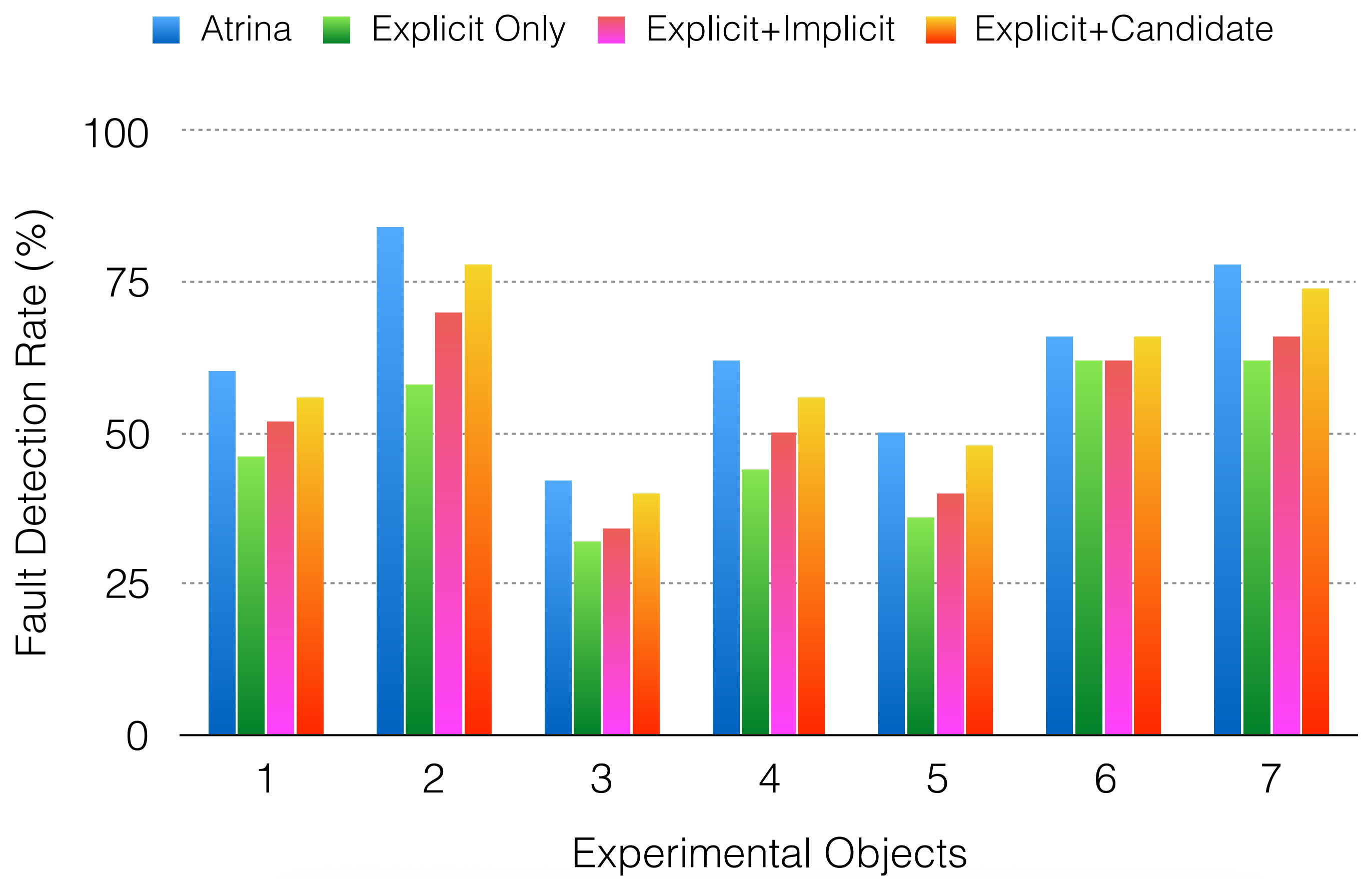
DOM-based assertions that are not connected to the JavaScript code:

- 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

Effectiveness in terms of fault detection capability

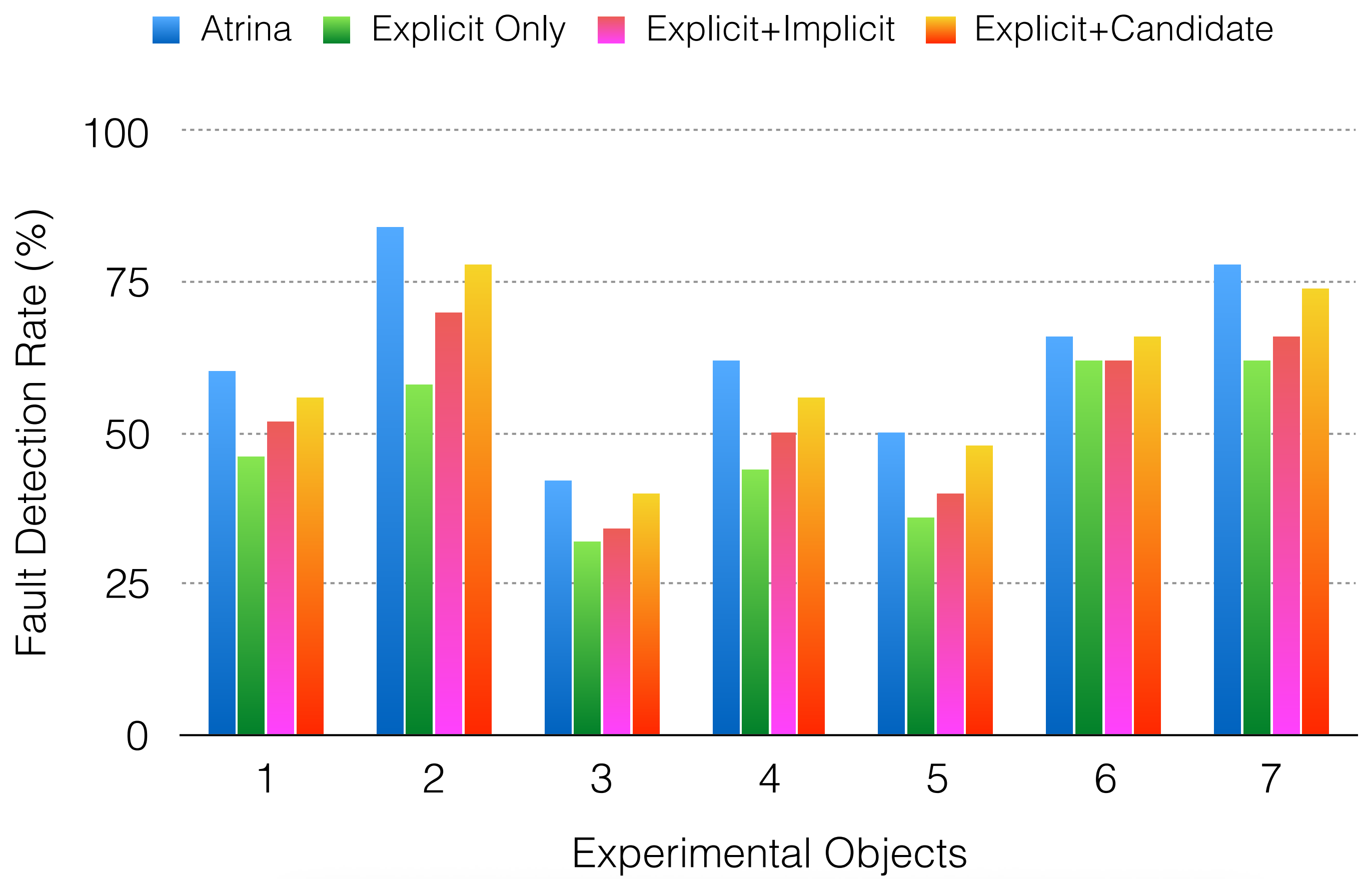


Effectiveness in terms of fault detection capability



Atrina detects on average 63% of the total faults

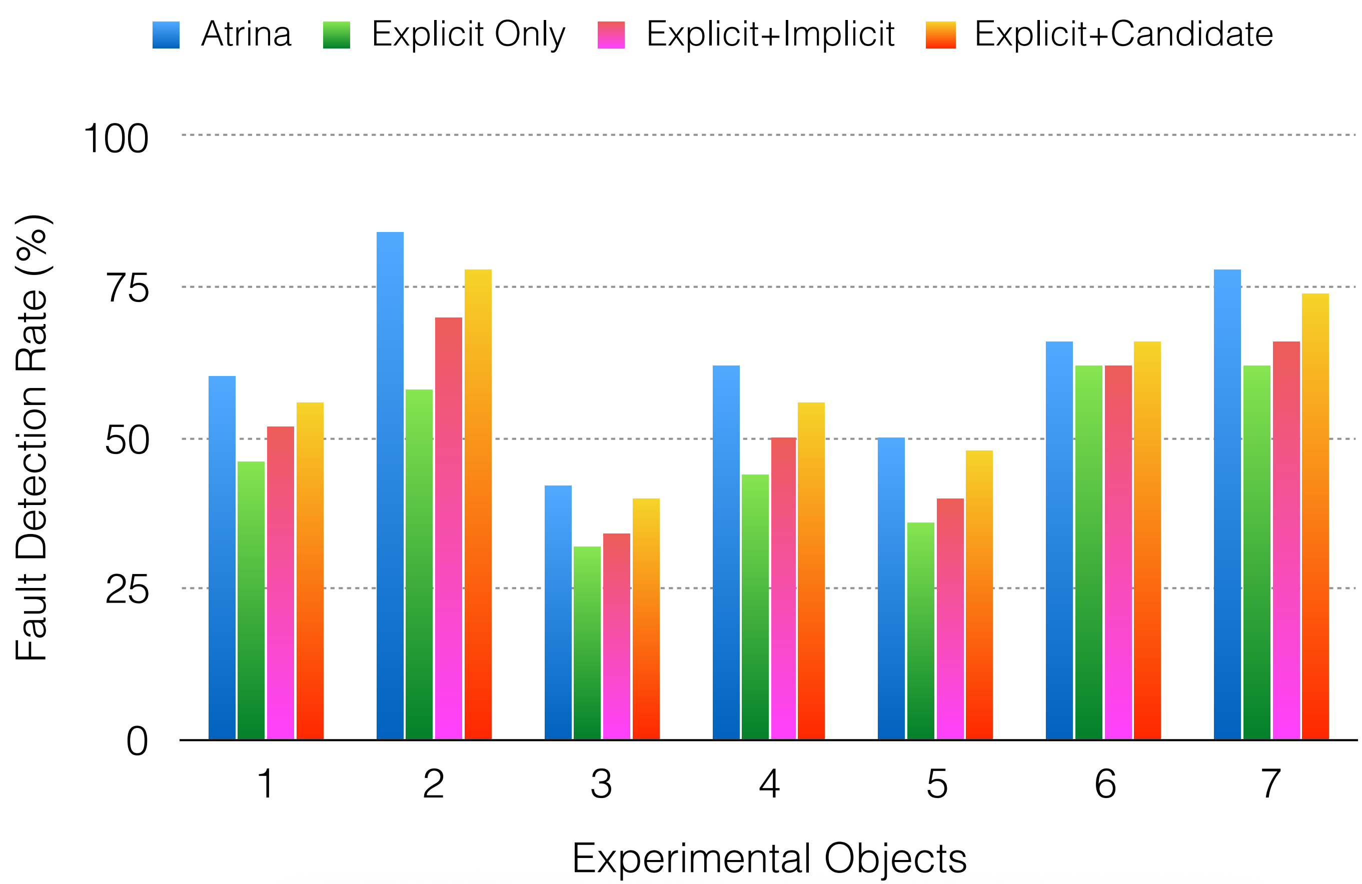
Effectiveness in terms of fault detection capability



Atrina detects on average 63% of the total faults

Explicit assertions detect 76% of the total detected faults on average

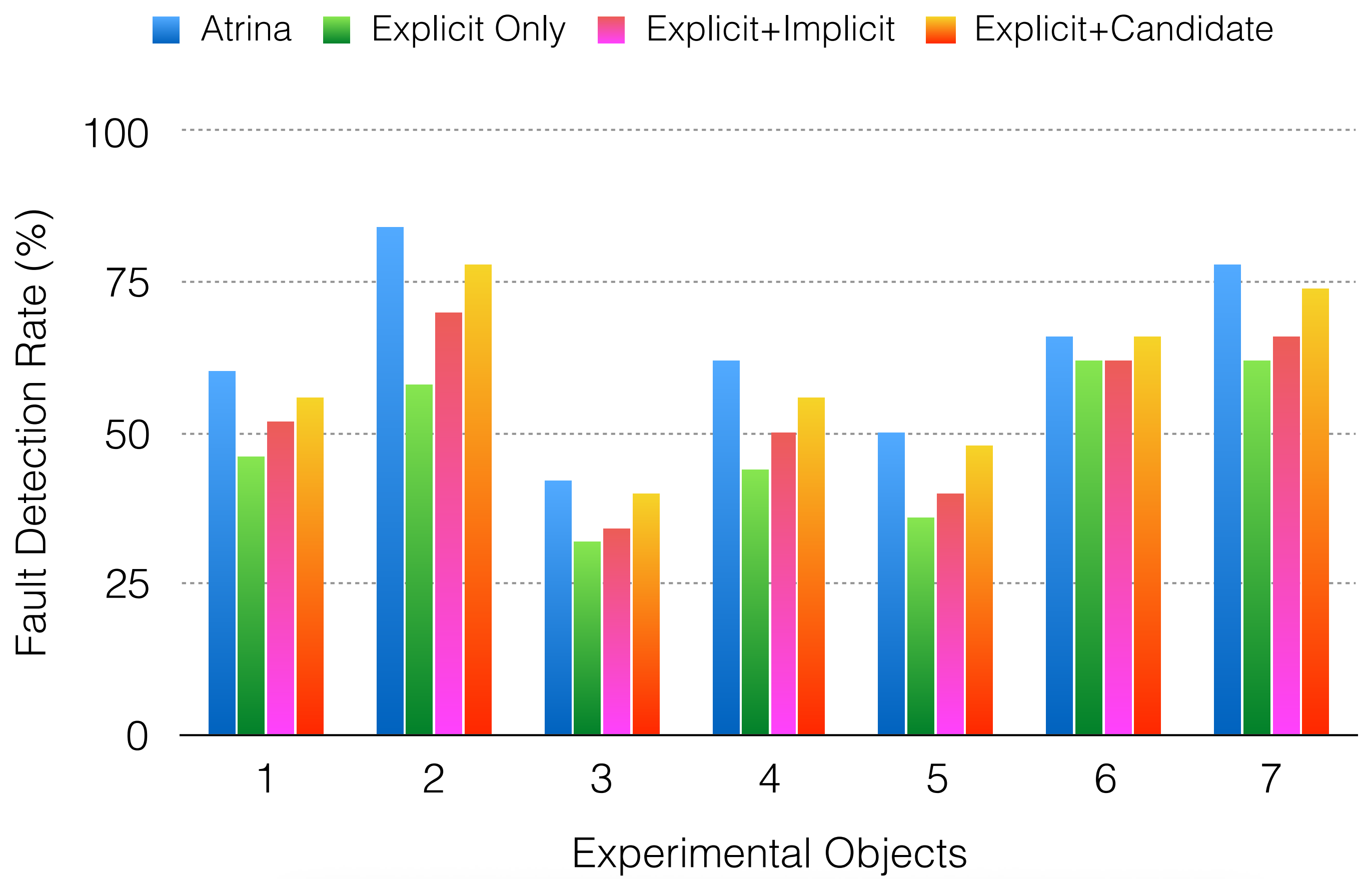
Effectiveness in terms of fault detection capability



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

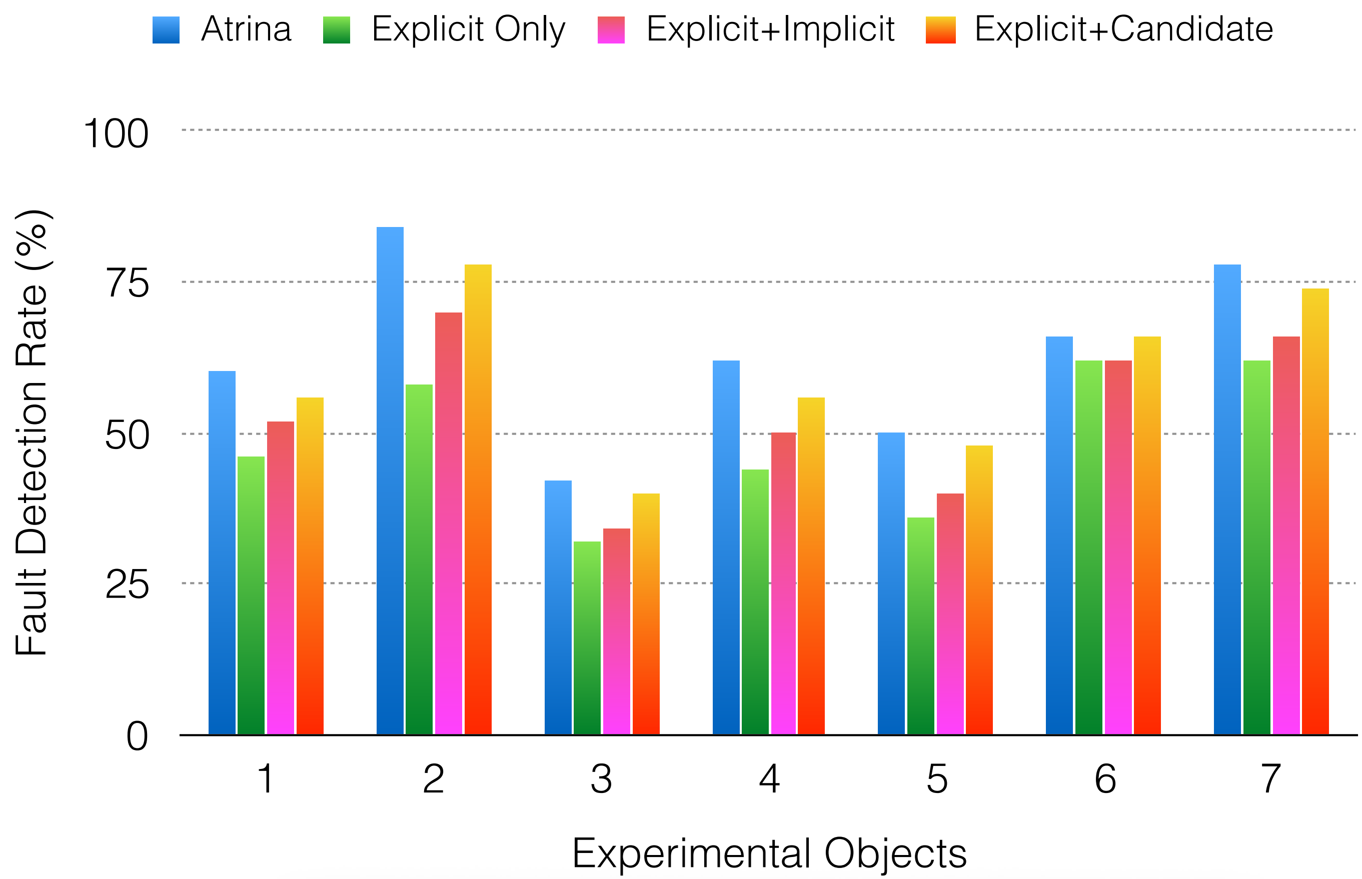


Atrina detects on average 63% of the total faults

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

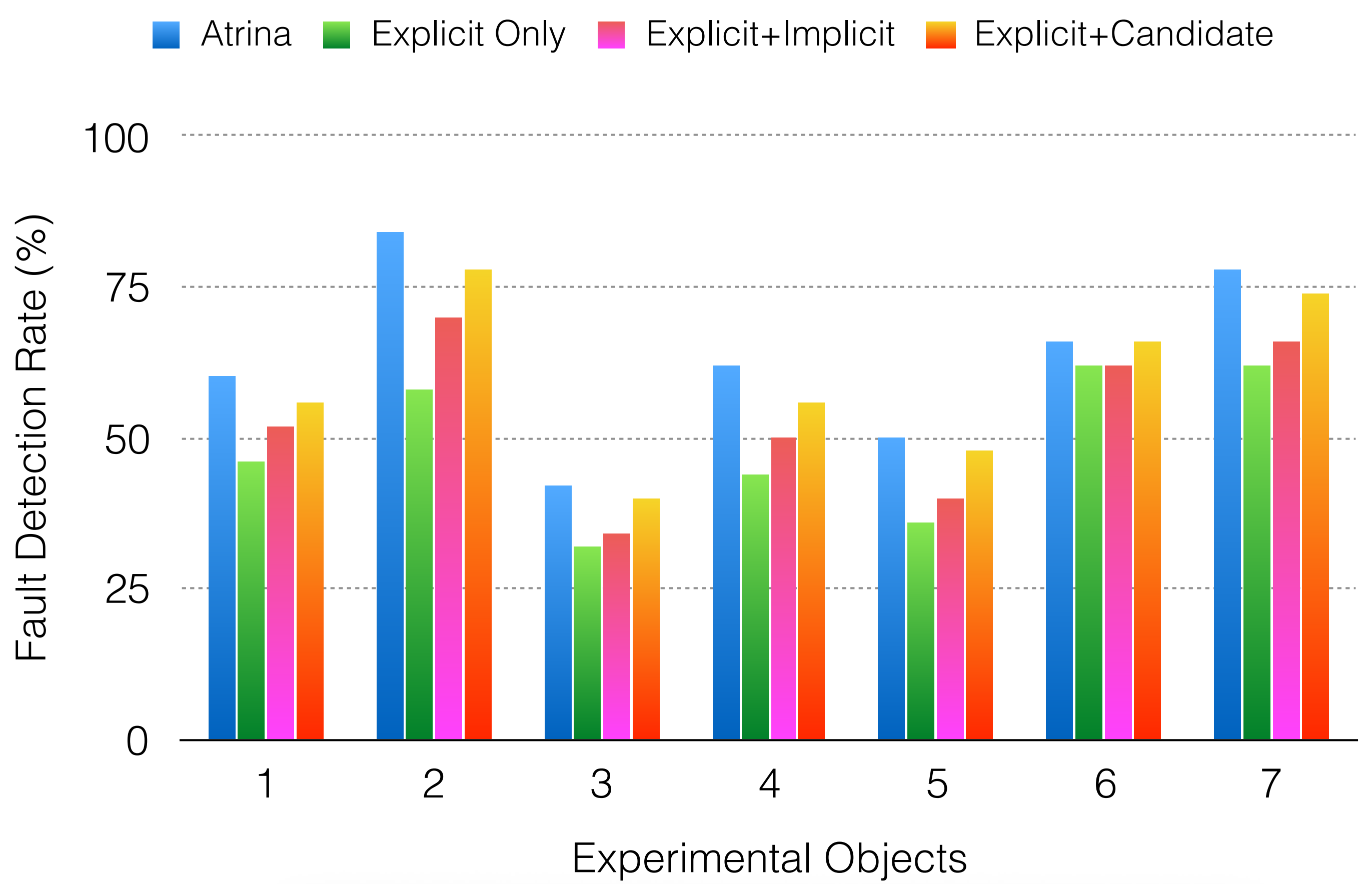


Atrina detects on average 63% of the total faults

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



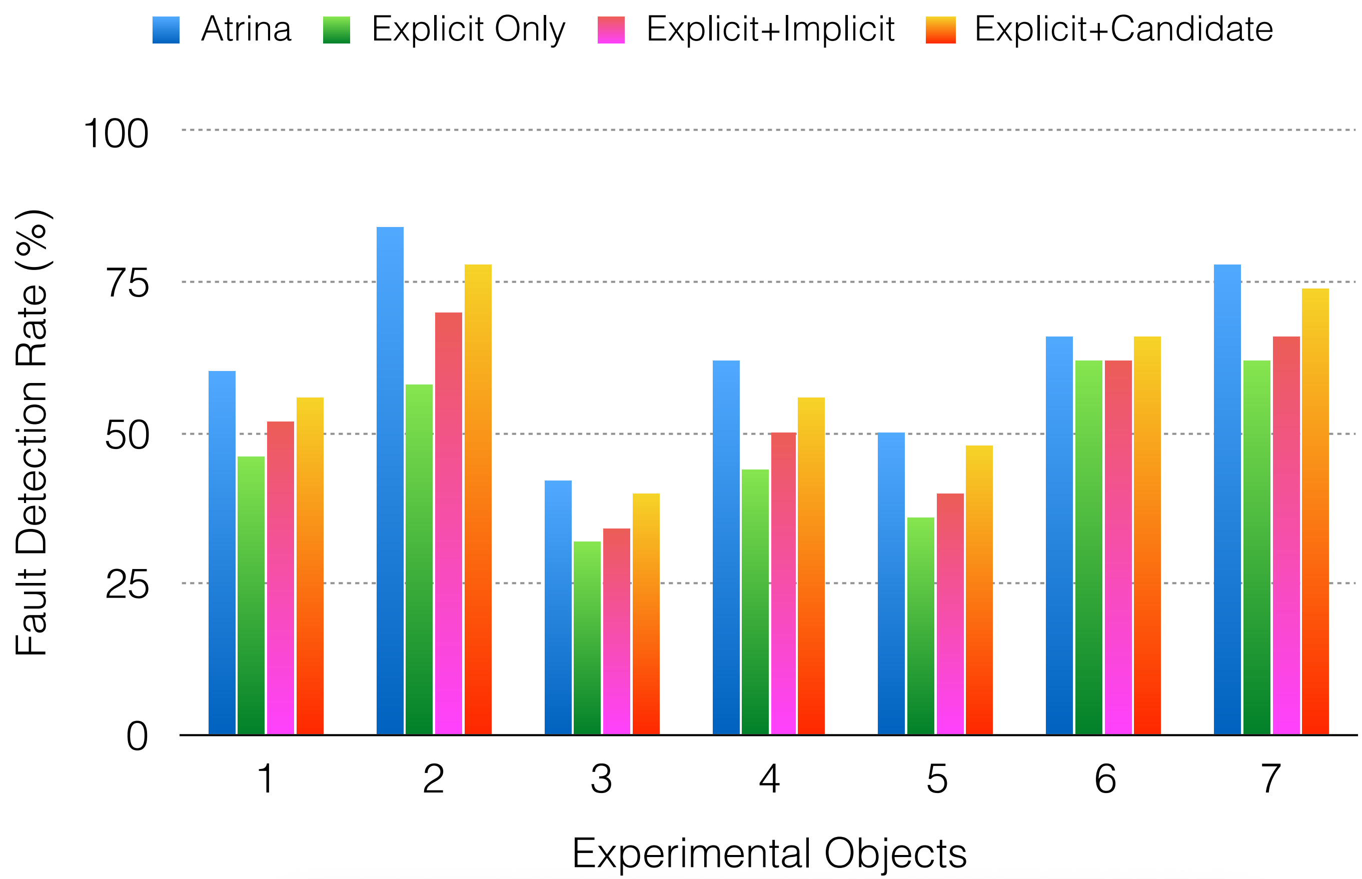
Atrina detects on average 63% of the total faults

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



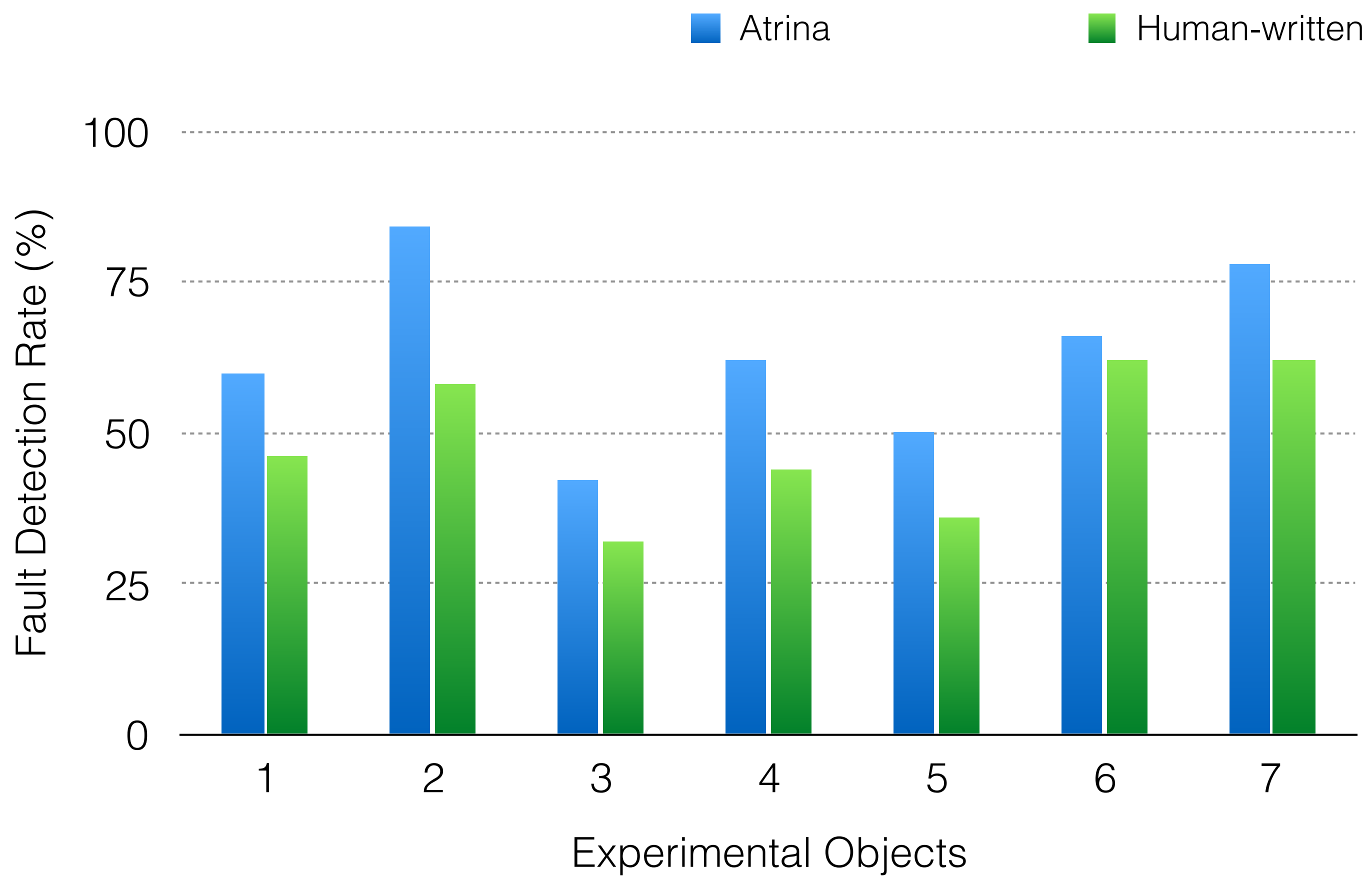
Atrina detects on average 63% of the total faults

Explicit assertions contribute the most among the three assertion types

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

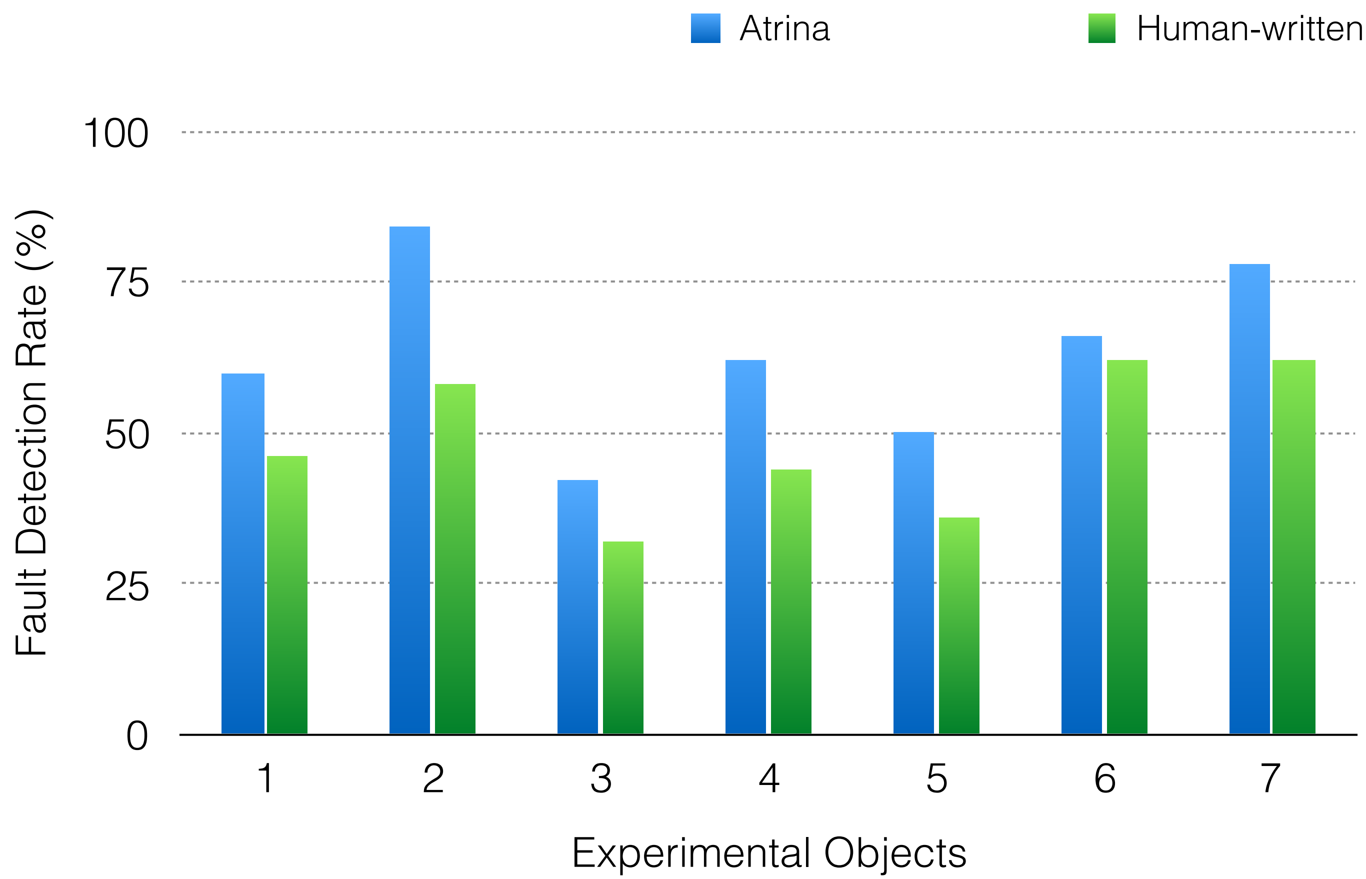
Candidate assertions play a more prominent role in increasing the number of detected faults

Comparison with human-written DOM-based assertions



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

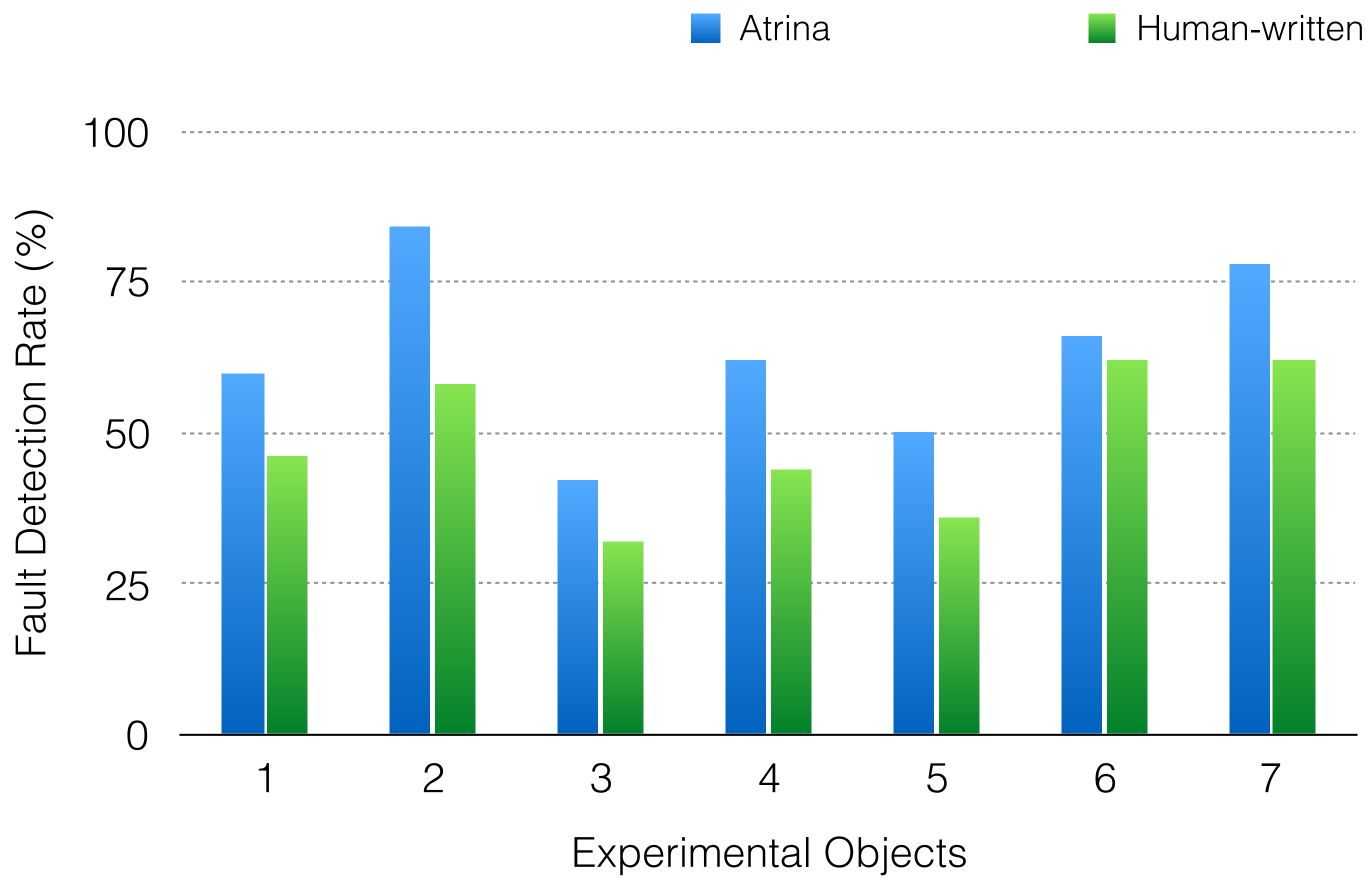
Comparison with human-written DOM-based assertions



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

Comparison with human-written DOM-based assertions

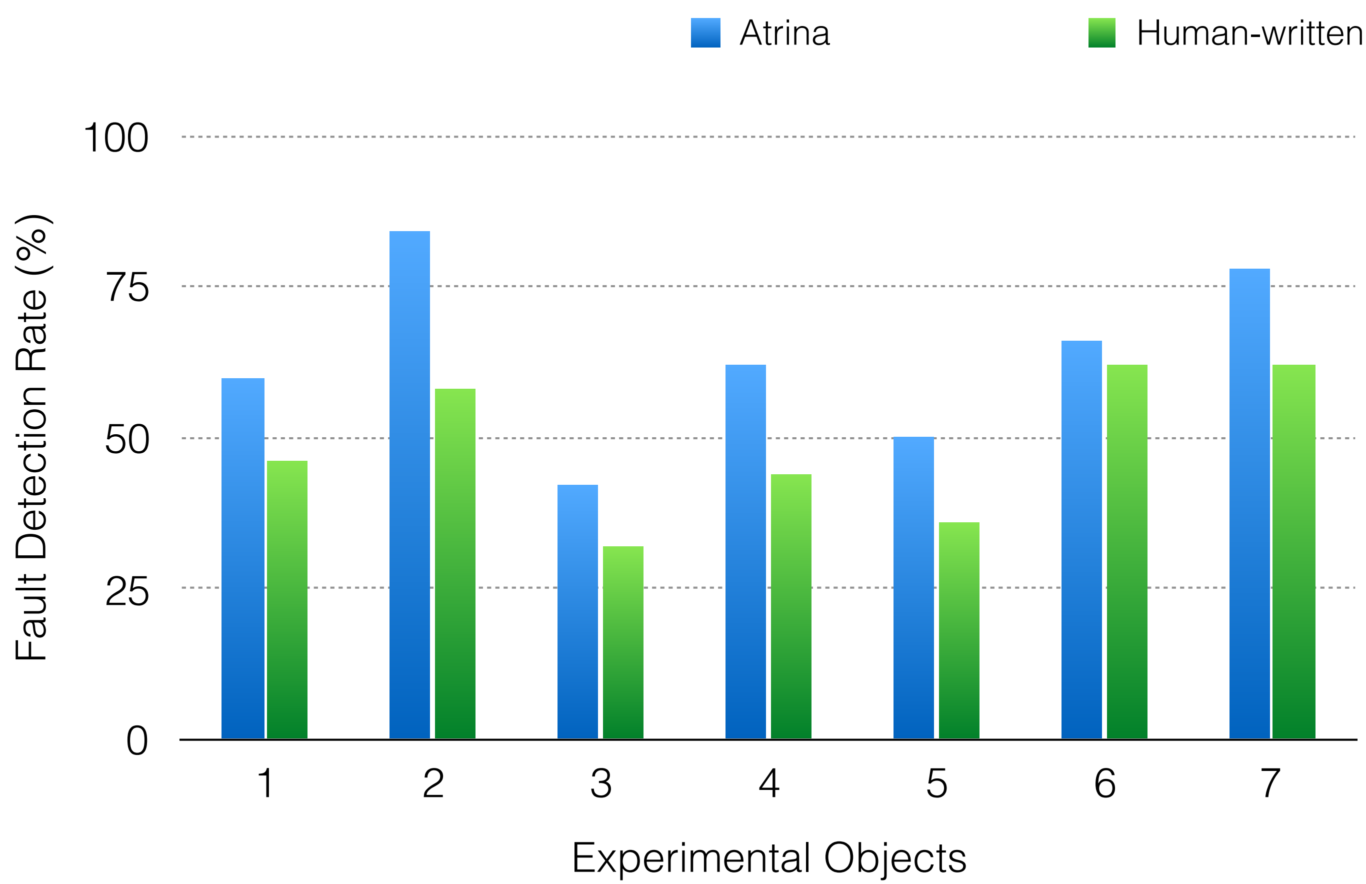


Atrina outperforms manual assertions in terms of fault finding capability by 31%

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

They require executing a more complex sequence of events to propagate to the DOM

Comparison with human-written DOM-based assertions

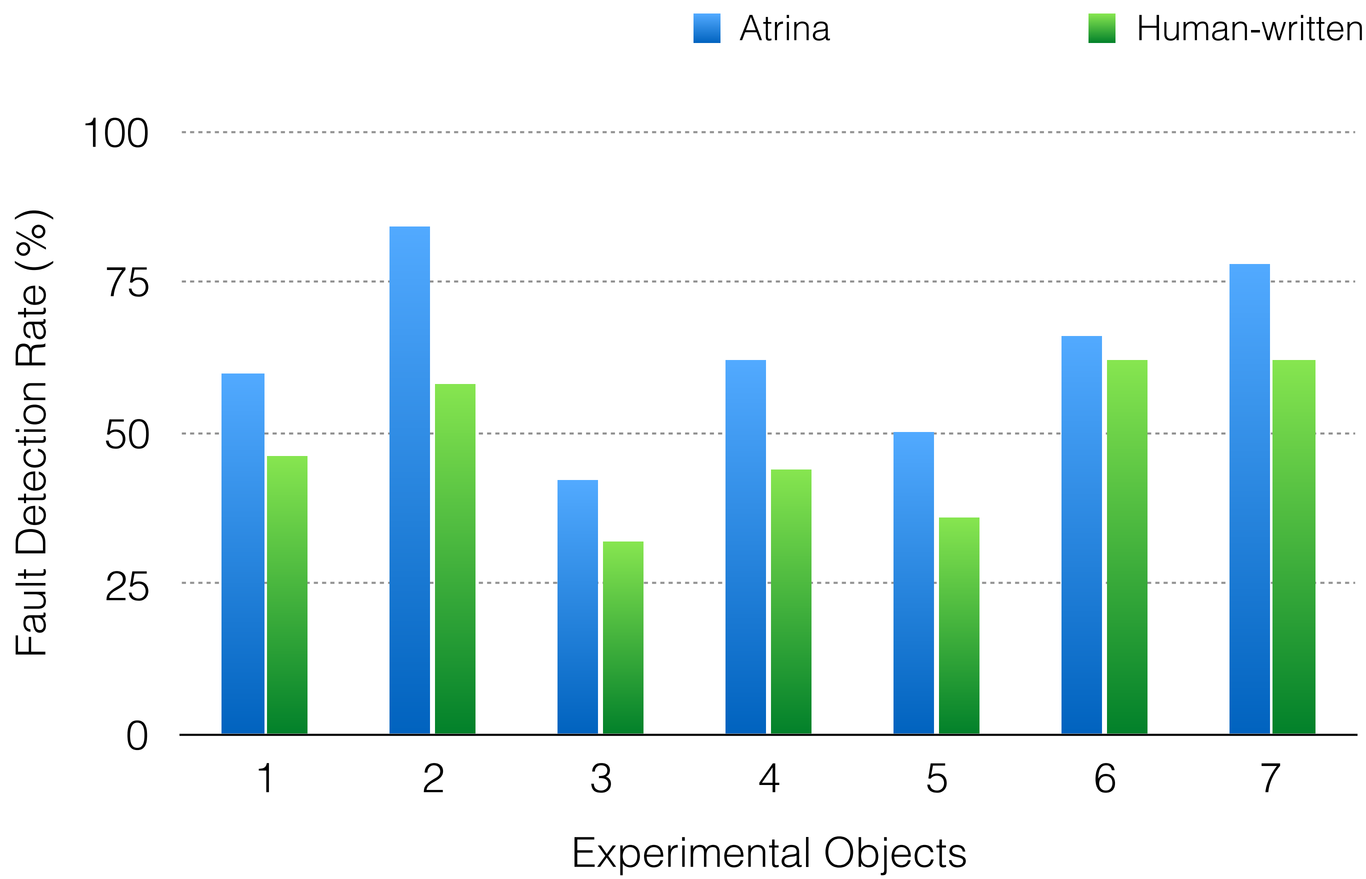


Atrina outperforms manual assertions in terms of fault finding capability by 31%

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

DOM property that is checked in the human-written test is later used in JavaScript code that involves internal computations only

Comparison with human-written DOM-based assertions

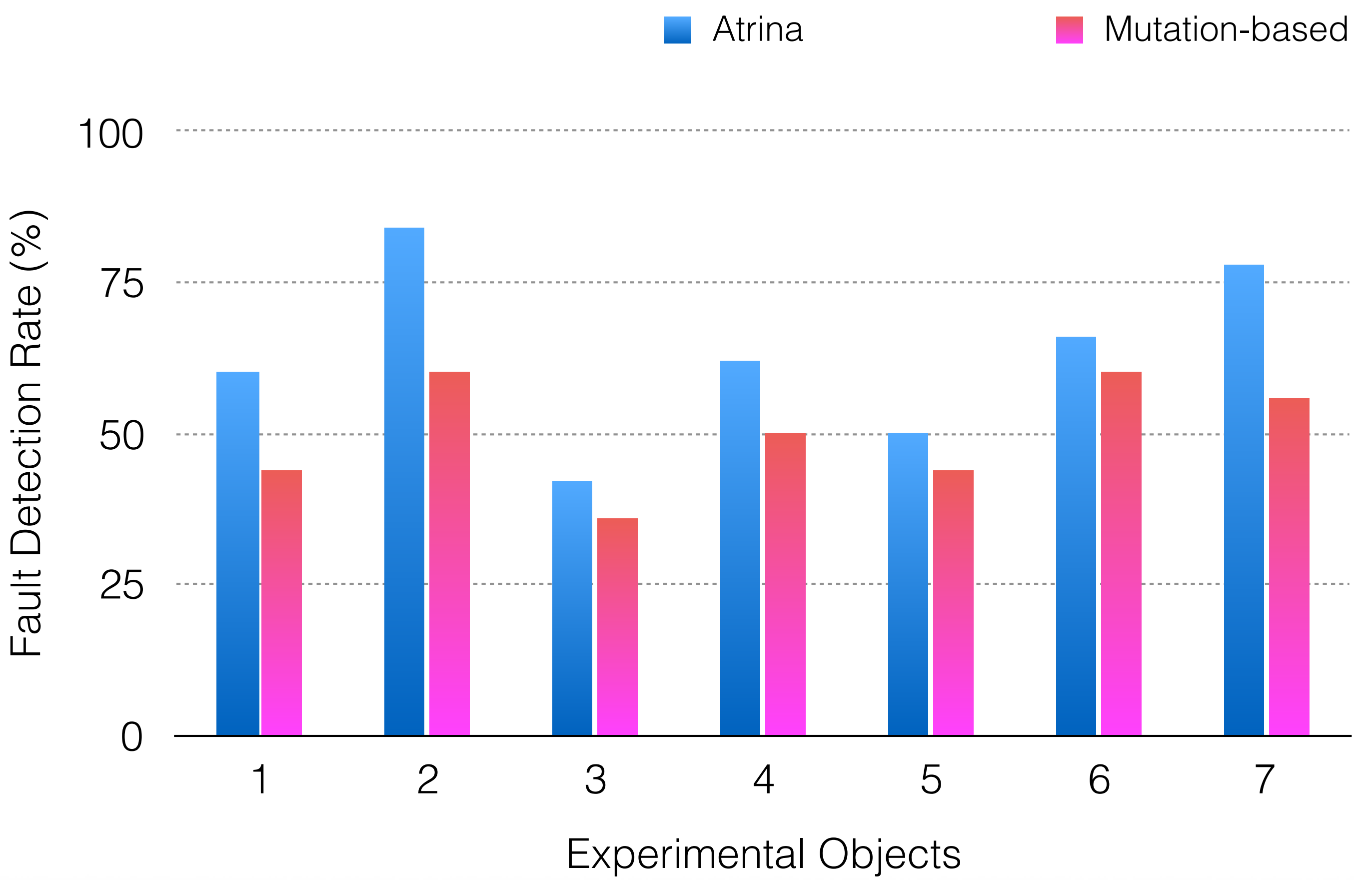


Atrina outperforms manual assertions in terms of fault finding capability by 31%

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

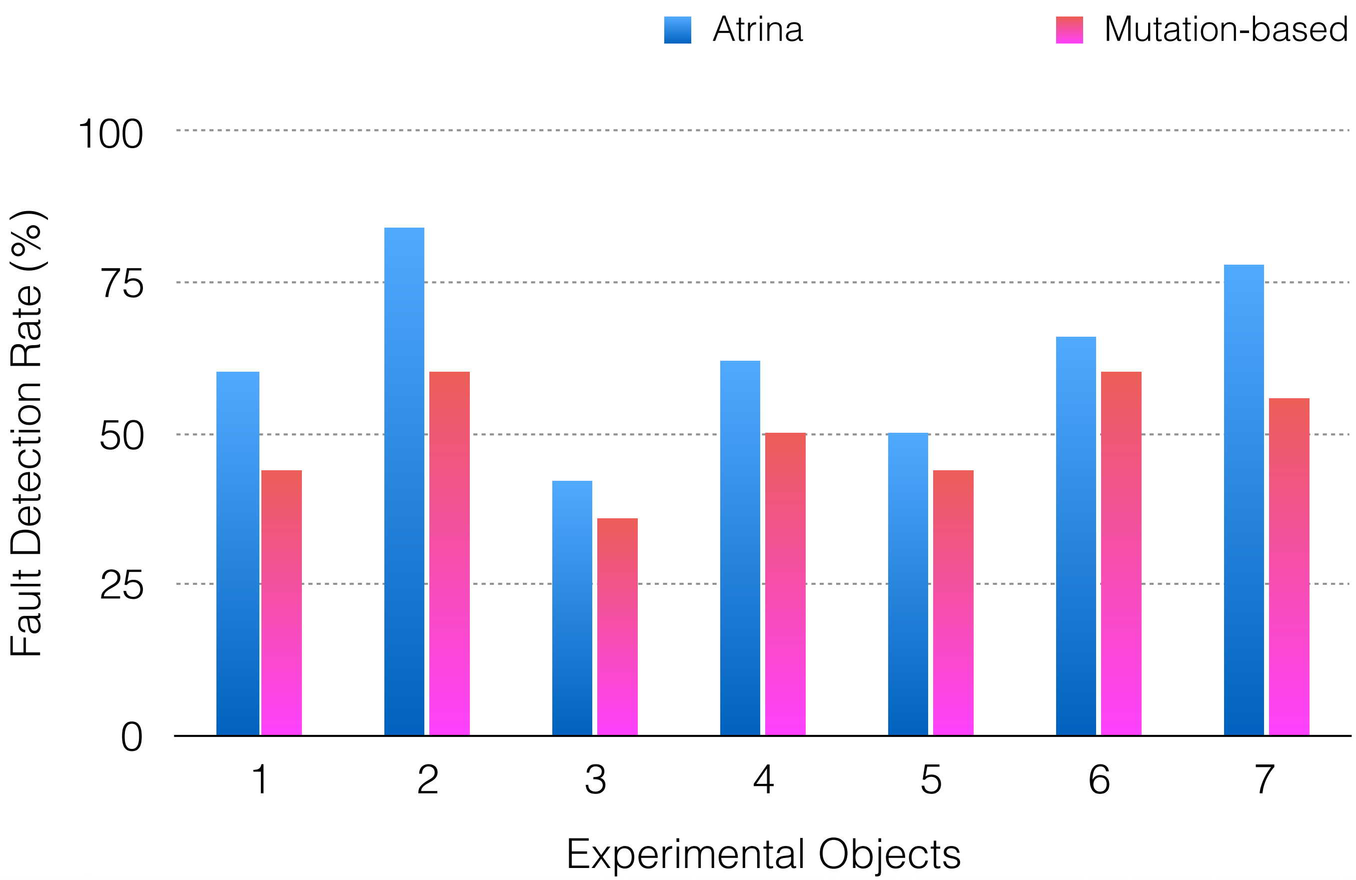
More than 50% of the selected DOM properties in Atrina were ignored in human-written DOM assertions

Comparison with mutation-based assertions



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

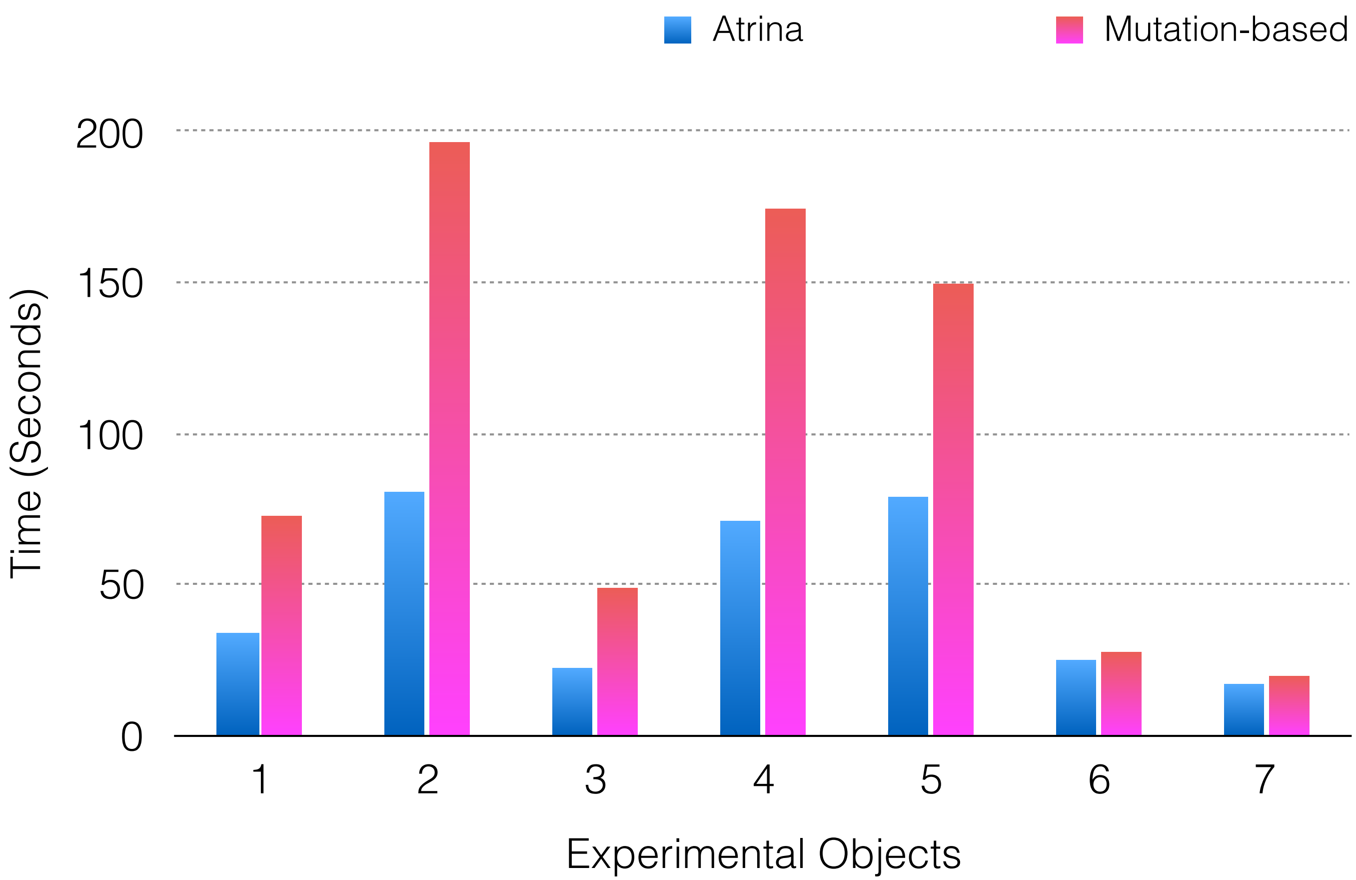
Comparison with mutation-based assertions



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

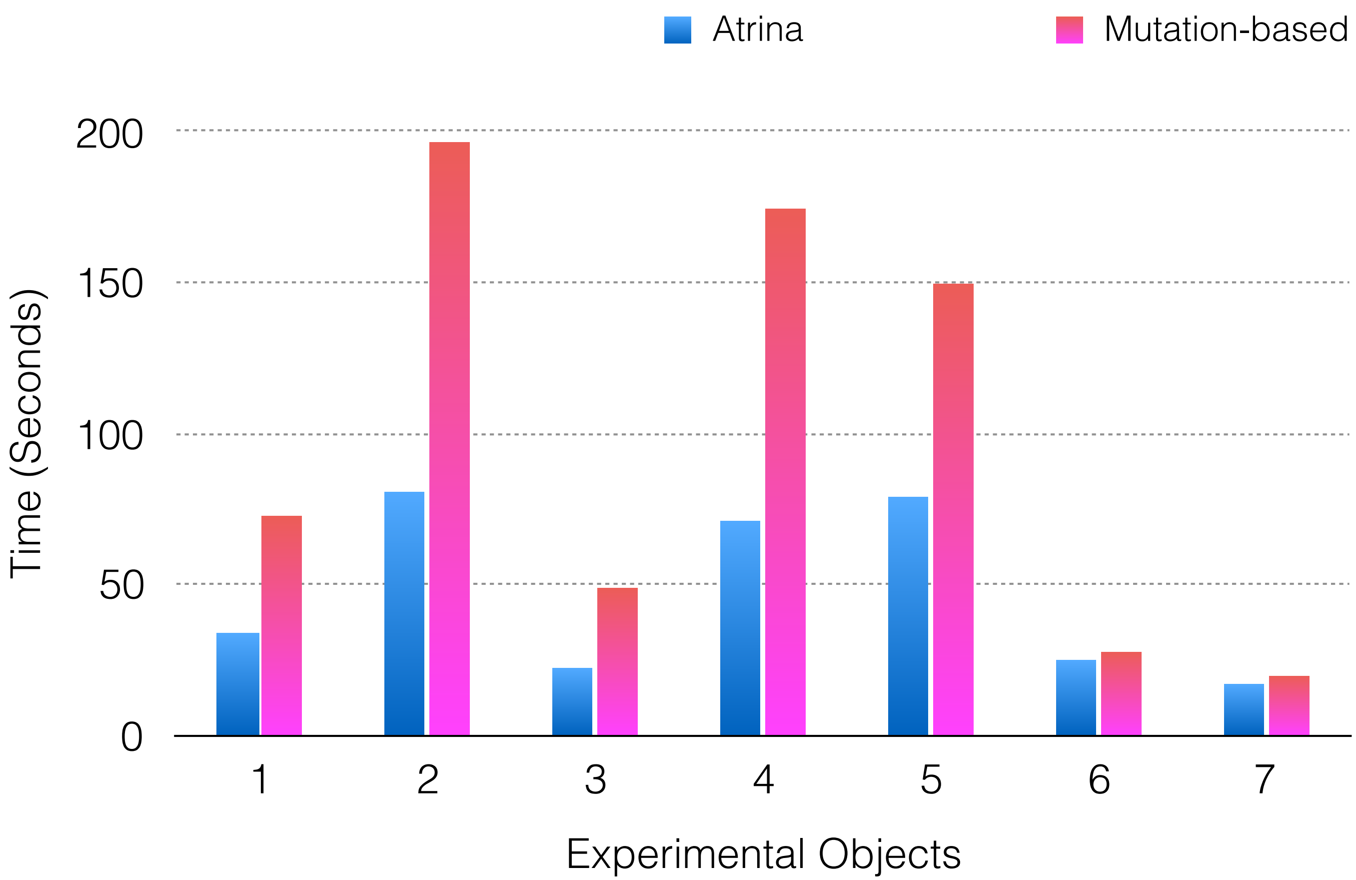
Comparison with mutation-based assertions (Time overhead)



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)

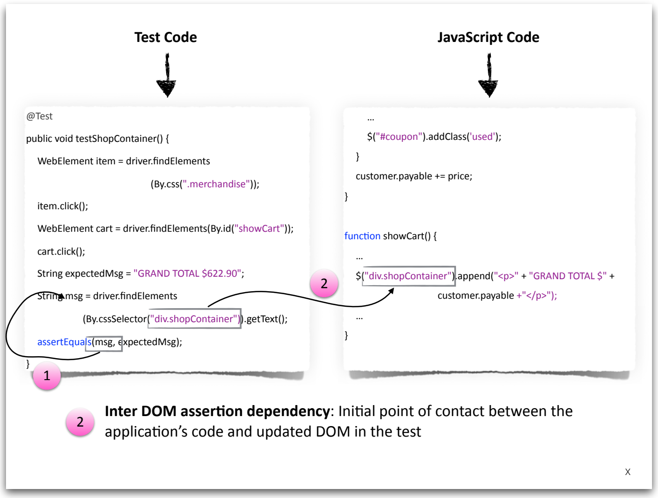


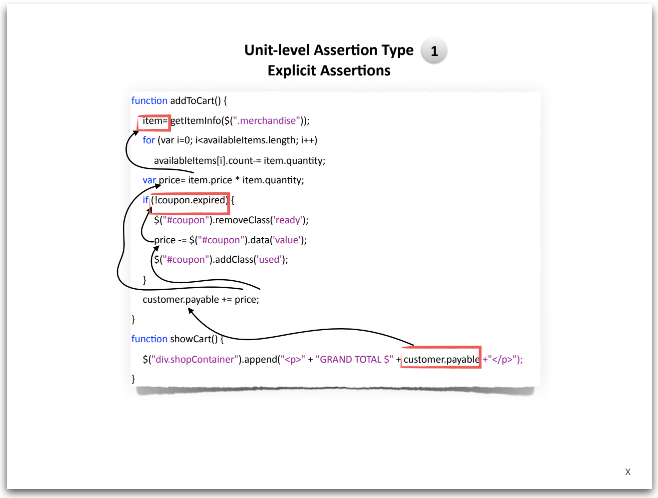
Overhead of Atrina: 47 (Sec)

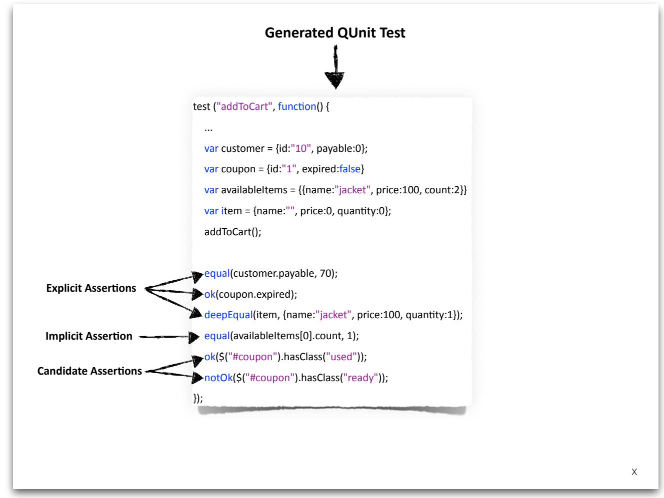
Overhead of the mutation-based approach: 98 (Sec)

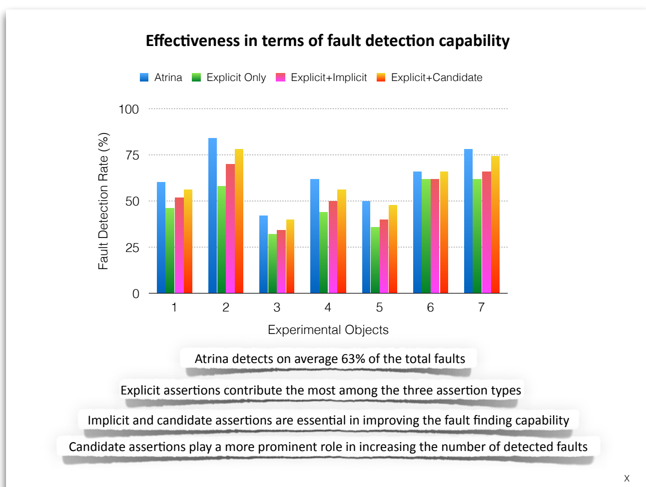
Atrina significantly outperforms mutation-based assertion generation in terms of time efficiency

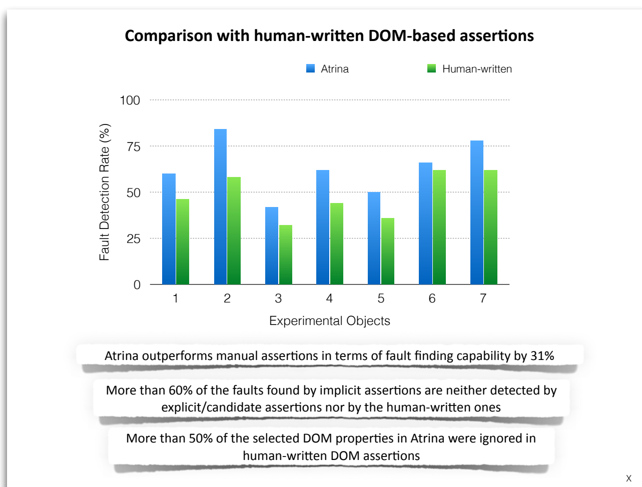
Summary

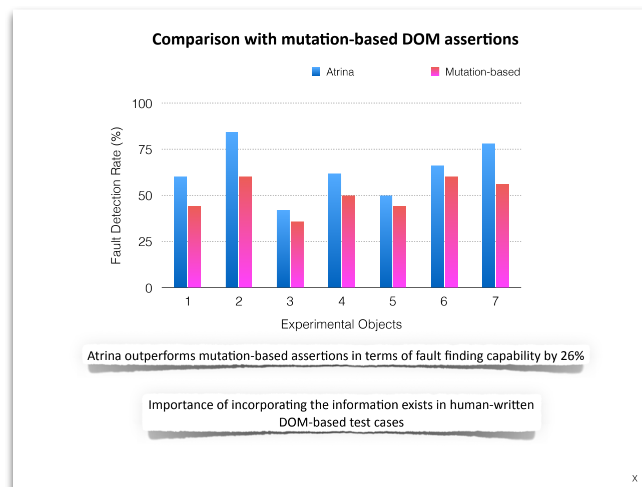

Test Code
JavaScript Code
@Test
public void testShopContainer() {
WebElement item = driver.findElements
(By.css(".merchandise"));
item.click();
WebElement cart = driver.findElements(By.id("showCart"));
cart.click();
String expectedMsg = "GRAND TOTAL $622.90";
String msg = driver.findElements
(By.cssSelector("div.shopContainer")).getText();
assertEquals(msg, expectedMsg);
}
...
$("#coupon").addClass('used');
}
customer.payable += price;
}
function showCart() {
...
$("div.shopContainer").append("<p>" + "GRAND TOTAL $" +
customer.payable +"</p>");
...
}
1
2
2 Inter DOM assertion dependency: Initial point of contact between the application's code and updated DOM in the test
x


Unit-level Assertion Type 1
Explicit Assertions
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>");
}
x


Generated QUnit Test
test ("addToCart", function() {
...
var customer = {id:"10", payable:0};
var coupon = {id:"1", expired:false}
var availableItems = [{name:"jacket", price:100, count:2}]
var item = {name:"", price:0, quantity:0};
addToCart();
equal(customer.payable, 70);
ok(coupon.expired);
deepEqual(item, {name:"jacket", price:100, quantity:1});
equal(availableItems[0].count, 1);
ok($("#coupon").hasClass("used"));
notOk($("#coupon").hasClass("ready"));
});
Explicit Assertions
Implicit Assertion
Candidate Assertions
x


Effectiveness in terms of fault detection capability
Atrina
Explicit Only
Explicit+Implicit
Explicit+Candidate
Fault Detection Rate (%)
100
75
50
25
0
1
2
3
4
5
6
7
Experimental Objects
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
x


Comparison with human-written DOM-based assertions
Atrina
Human-written
Fault Detection Rate (%)
100
75
50
25
0
1
2
3
4
5
6
7
Experimental Objects
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
x


Comparison with mutation-based DOM assertions
Atrina
Mutation-based
Fault Detection Rate (%)
100
75
50
25
0
1
2
3
4
5
6
7
Experimental Objects
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
x