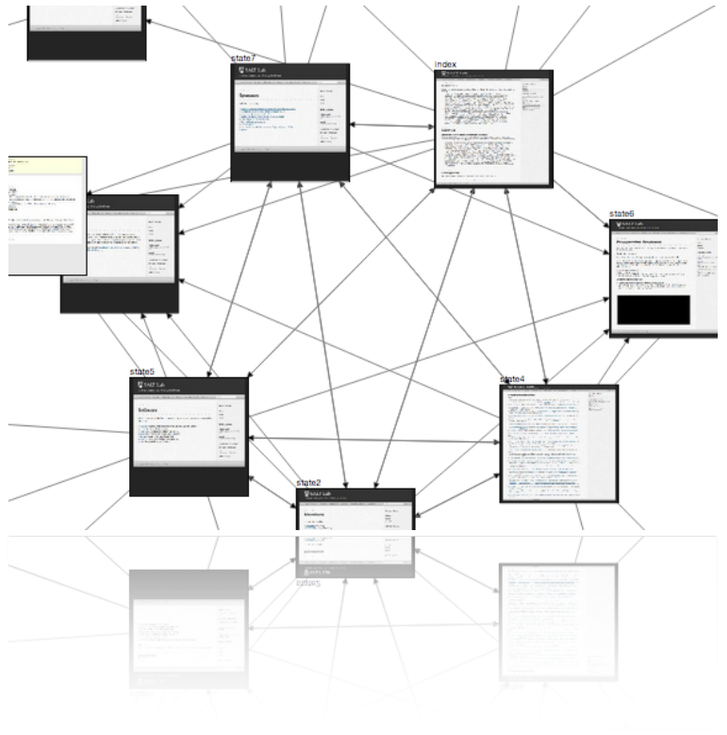
JSEFT: Automated JavaScript Unit Test Generation

Shabnam Mirshokraie
Ali Mesbah
Karthik Pattabiraman

University of British Columbia

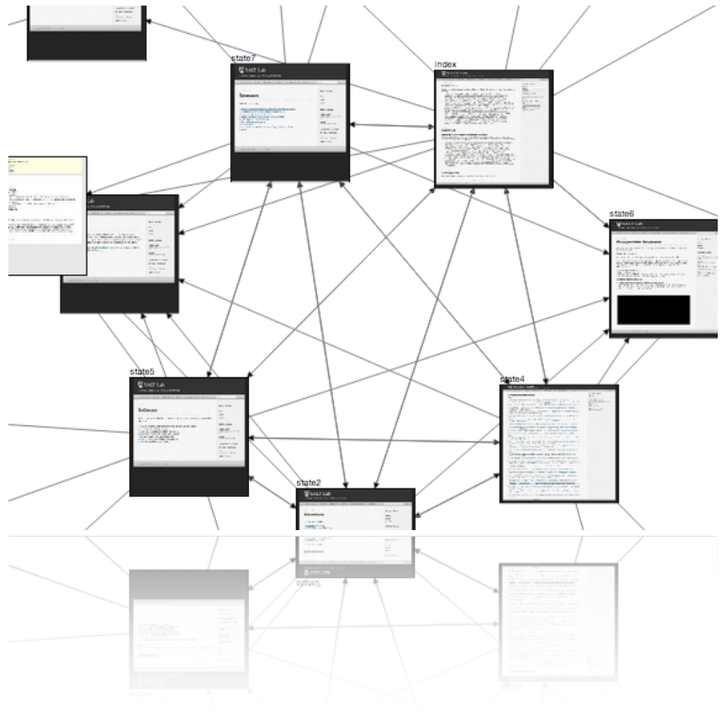
Javascript Test Generation Challenges

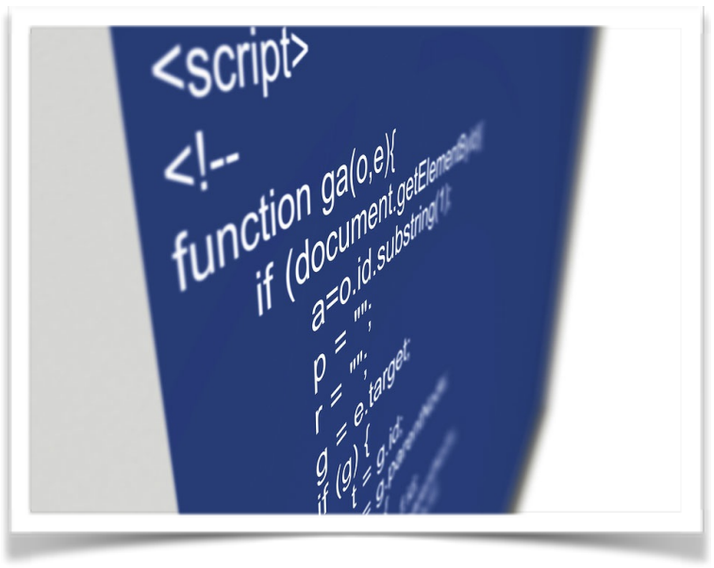
Many execution paths and states



Javascript Test Generation Challenges

Many execution paths and states

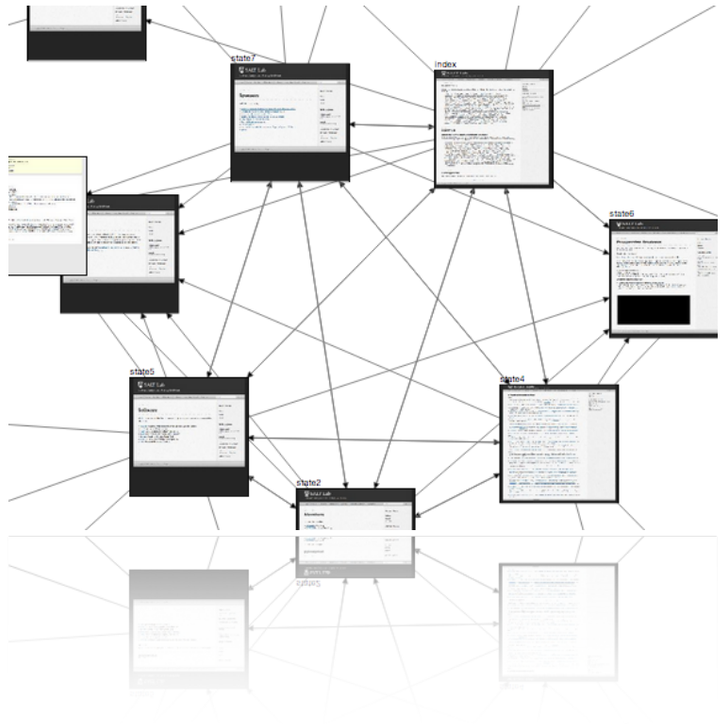




Extensive interaction with the DOM

Javascript Test Generation Challenges

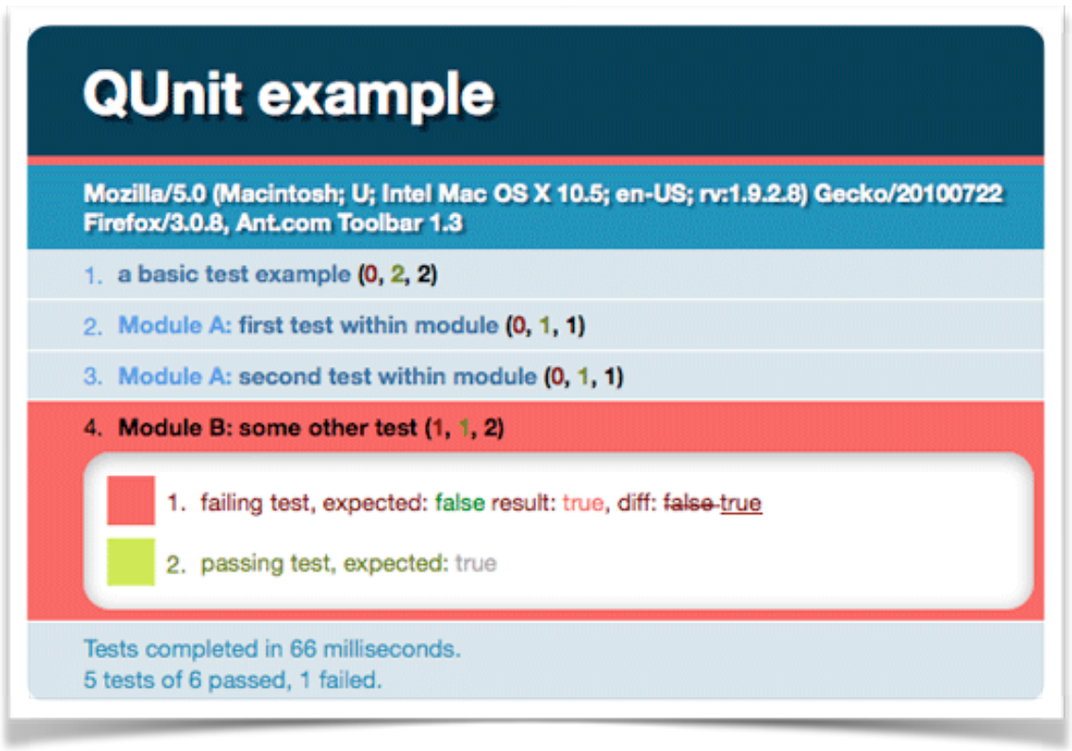
Many execution paths and states

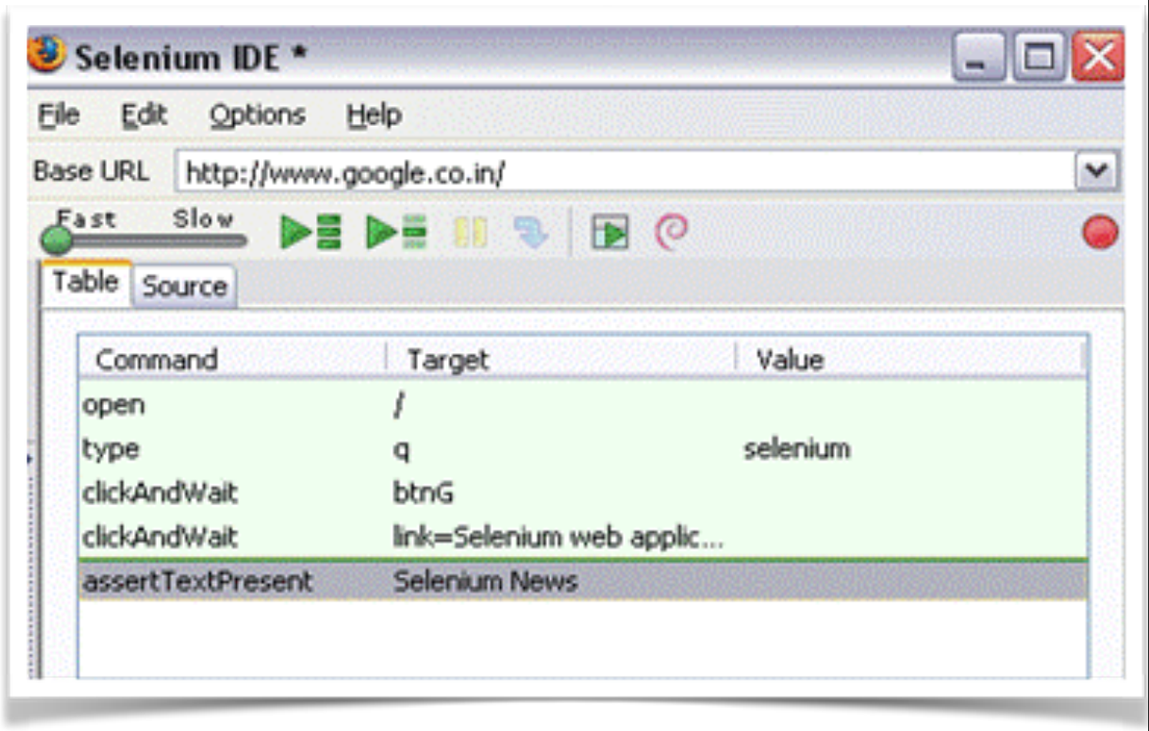


Extensive interaction with the DOM

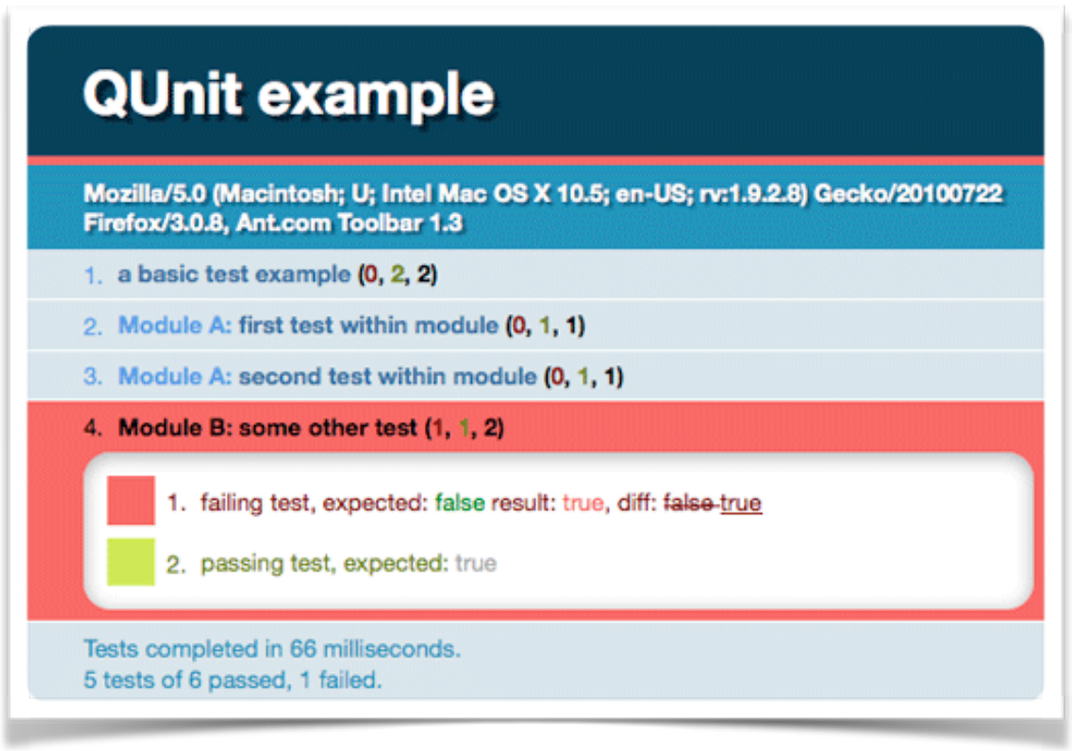
Assertion generation

Web Testing Frameworks





Web Testing Frameworks

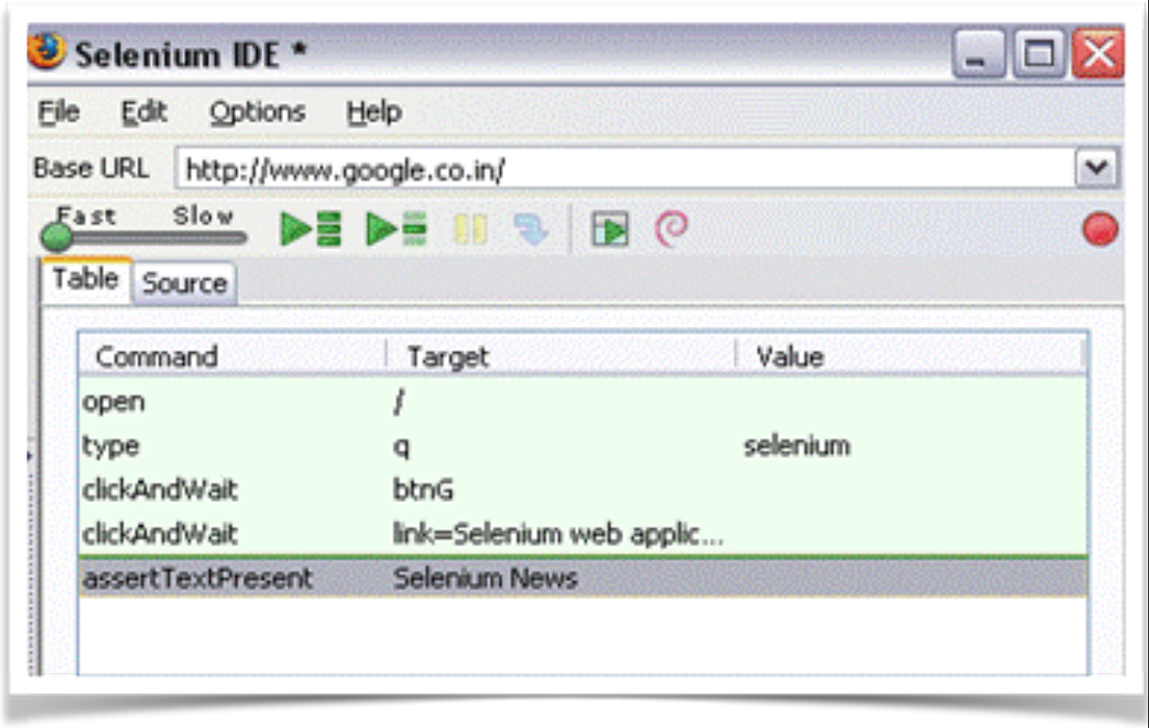


Automate test execution

But

Test cases are written manually

Difficult to write test suites with high fault finding capability



JavaScript Test Generation Techniques

testMethod	
open	http://www.google.com
focus	//*[@value='Google Search']/..
type	//*[@value='Google Search']/..
keyUp	//*[@value='Google Search']/..
keyPress	//*[@value='Google Search']/..
keyPress	//*[@value='Google Search']/..
waitForElementPresent	//INPUT[@name='q']
assertValue	//INPUT[@name='q']

DOM event tests

Fail to capture faults that do not propagate to an observable DOM state

Difficult to find the actual location of the fault even if it propagates to the DOM

Agnostic of the JavaScript code

JavaScript Test Generation Techniques

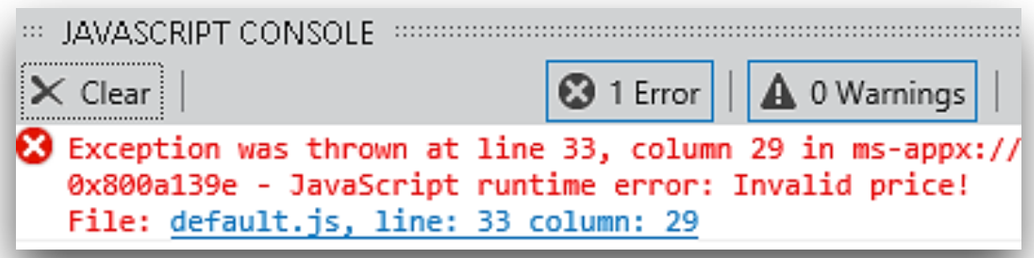
testMethod	
open	http://www.google.com
focus	//*[@value='Google Search']/..
type	//*[@value='Google Search']/..
keyUp	//*[@value='Google Search']/..
keyPress	//*[@value='Google Search']/..
keyPress	//*[@value='Google Search']/..
waitForElementPresent	//INPUT[@name='q']
assertValue	//INPUT[@name='q']

DOM event tests

Fail to capture faults that do not propagate to an observable DOM state

Difficult to find the actual location of the fault even if it propagates to the DOM

Agnostic of the JavaScript code



Soft oracles: Fail to capture logic errors

Goal

Automate test and oracle generation for JavaScript applications

Detecting regression JavaScript and DOM-level faults

Automate test and oracle generation for JavaScript applications

Detecting regression JavaScript and DOM-level faults

Implementation: JSEFT (JavaScript Event and Function Testing)

Automate test and oracle generation for JavaScript applications

Detecting regression JavaScript and DOM-level faults

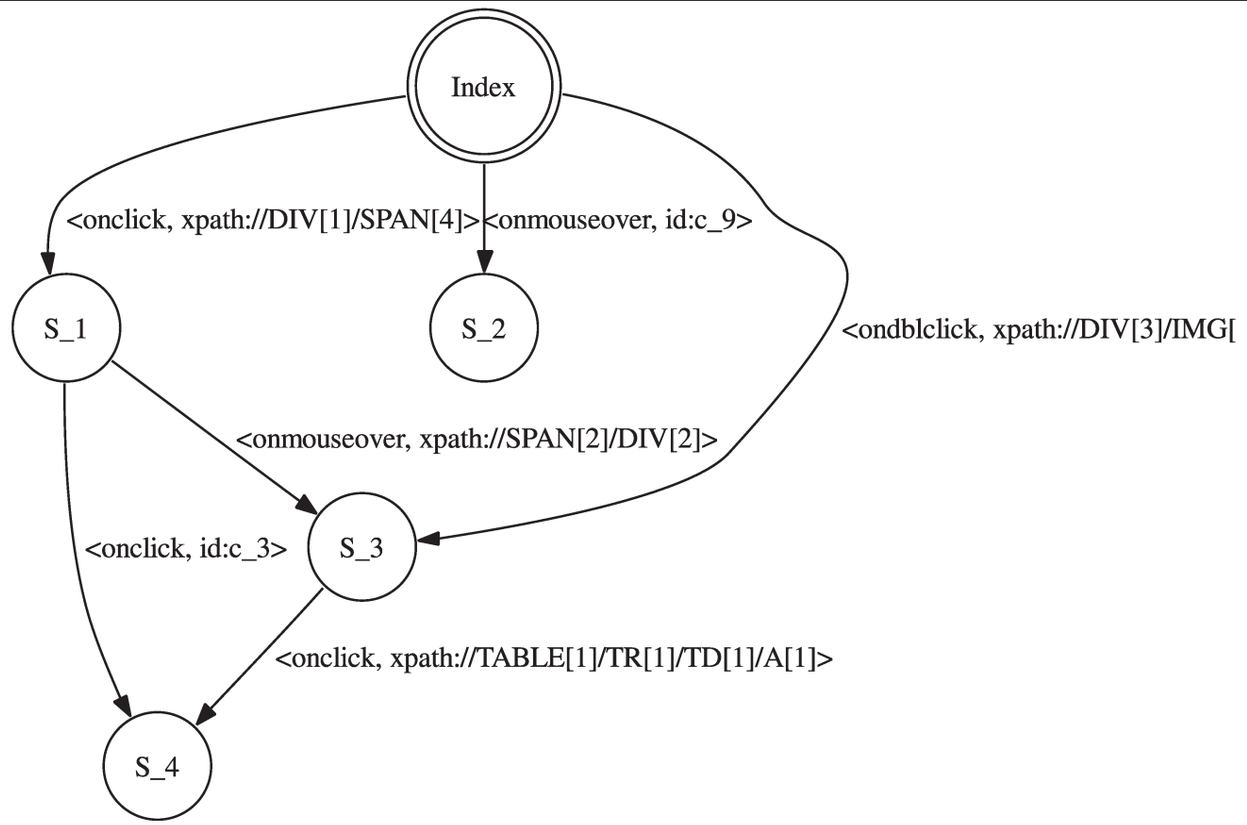
Implementation: JSEFT (JavaScript Event and Function Testing)

DOM-level event sequences and assertions

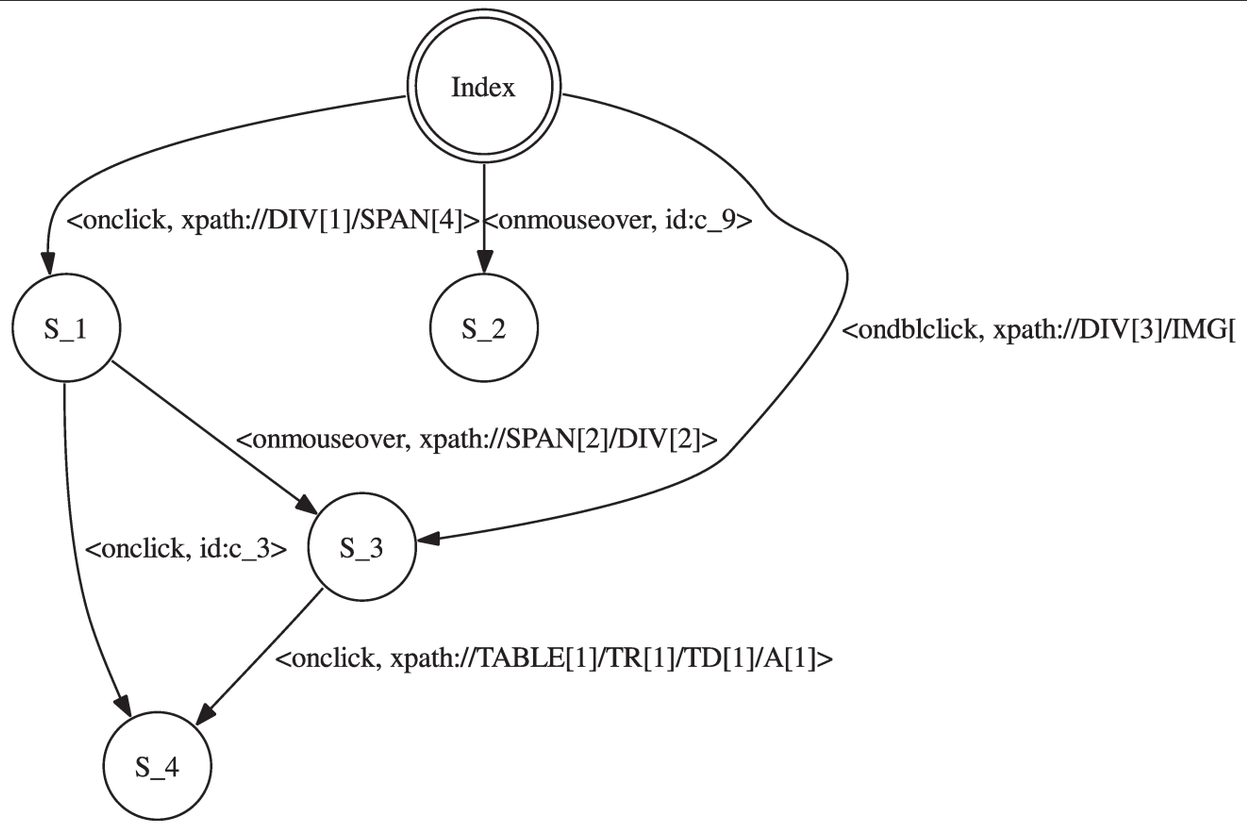
Check the application's behaviour from an end user's perspective

Function-level unit tests and assertions

Verify the functionality of JavaScript code at the function level



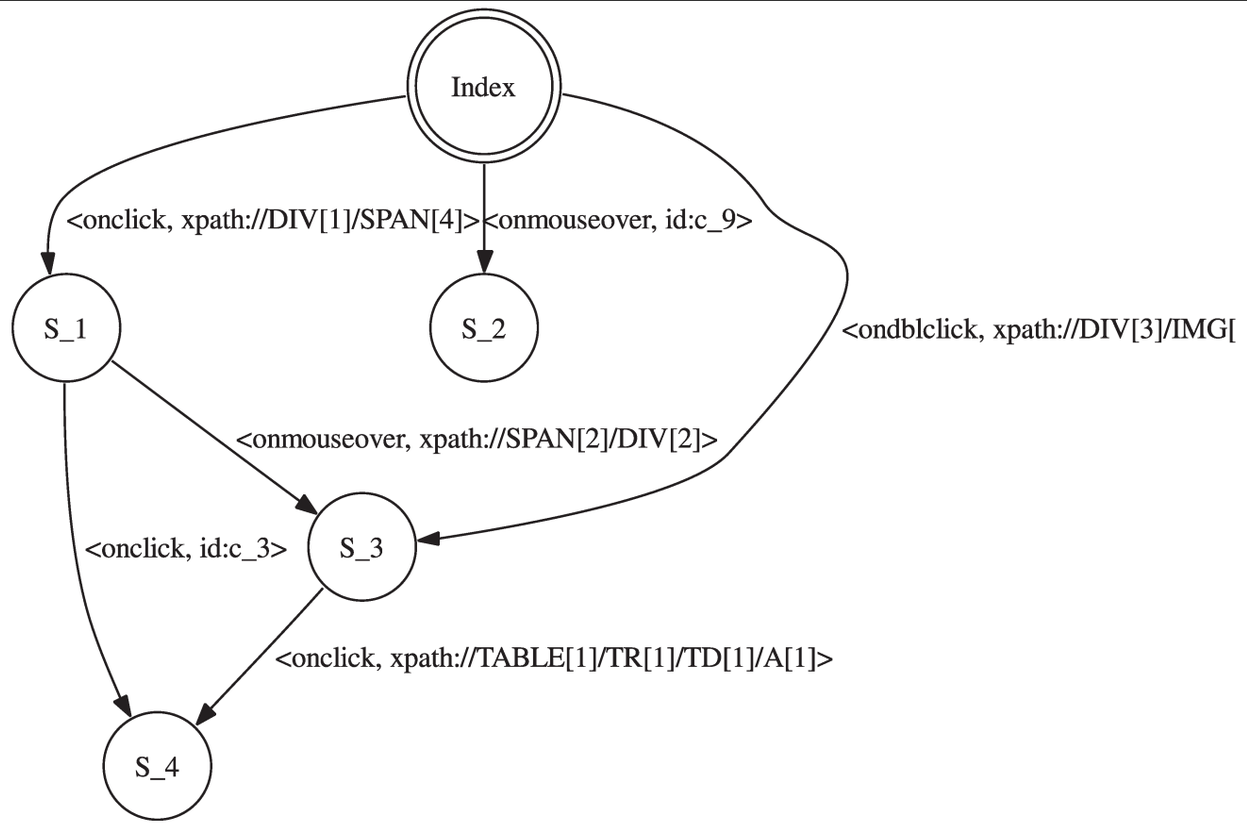
Test model extraction
and
Function coverage maximization



Test model extraction
and
Function coverage maximization

Test case generation

```
test( “test X”, function() {
      var returnVal=X(10);
      …
});
```



Test model extraction
and
Function coverage maximization

Test case generation

```
test( “test X”, function() {
      var returnVal=X(10);
      …
});
```

```
test( “test X”, function() {
      var returnVal=X(10);
      equal(typeof(returnVal), ‘number’);
      equal(returnVal, 30);
});
```

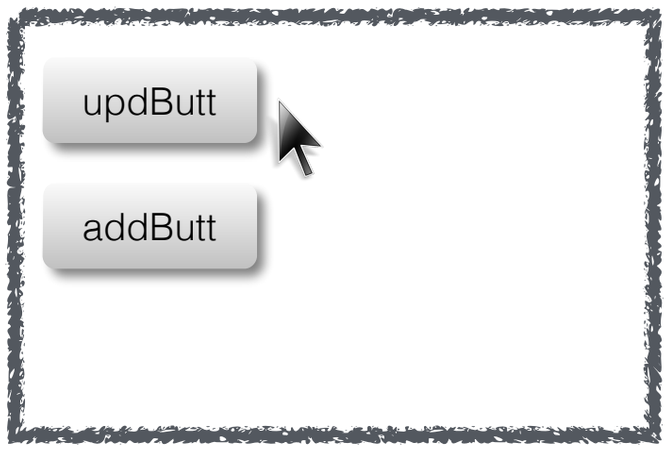
Test oracle generation

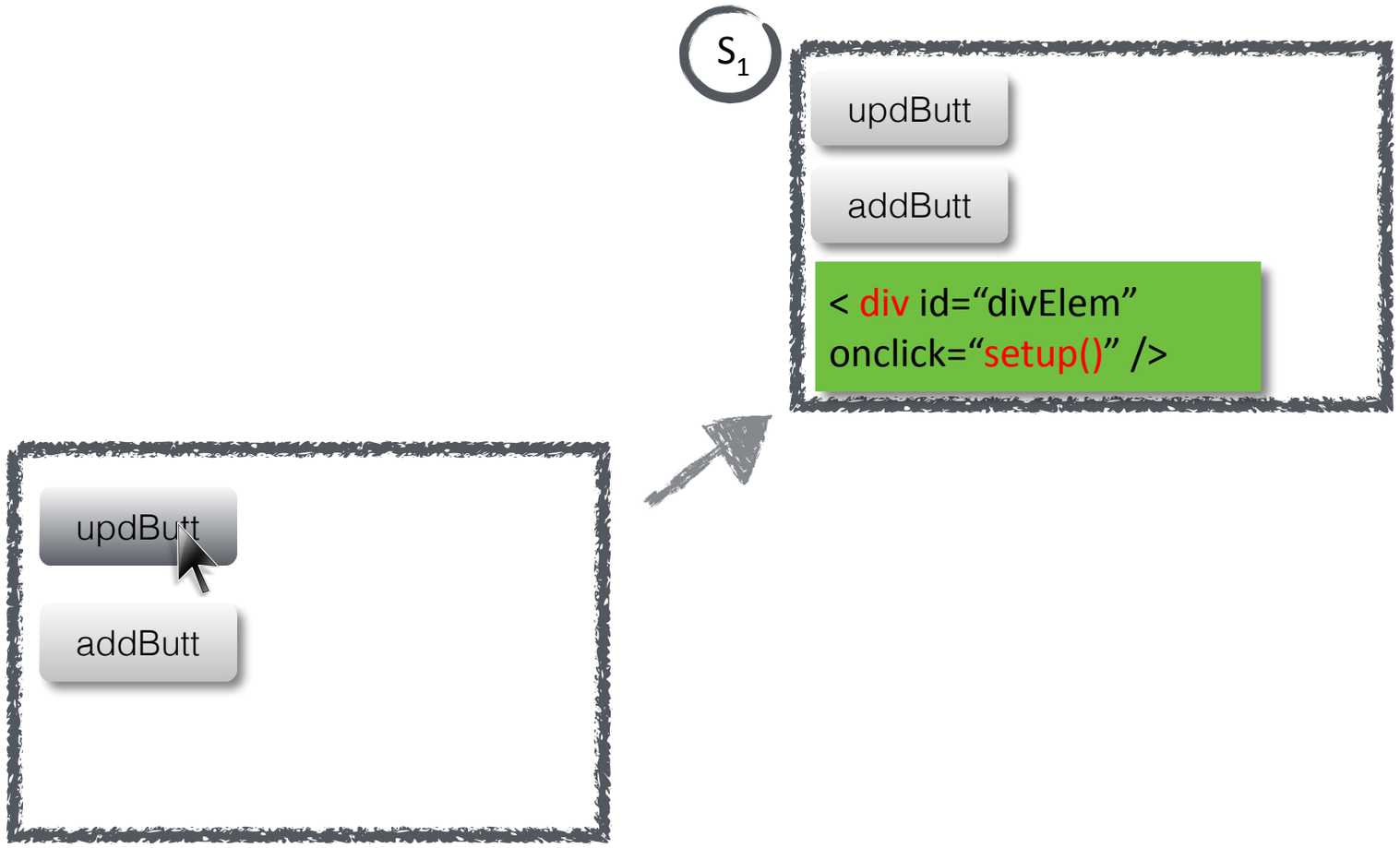
Model Extraction and Function Coverage Maximization

Maximizes the number of functions that can be covered

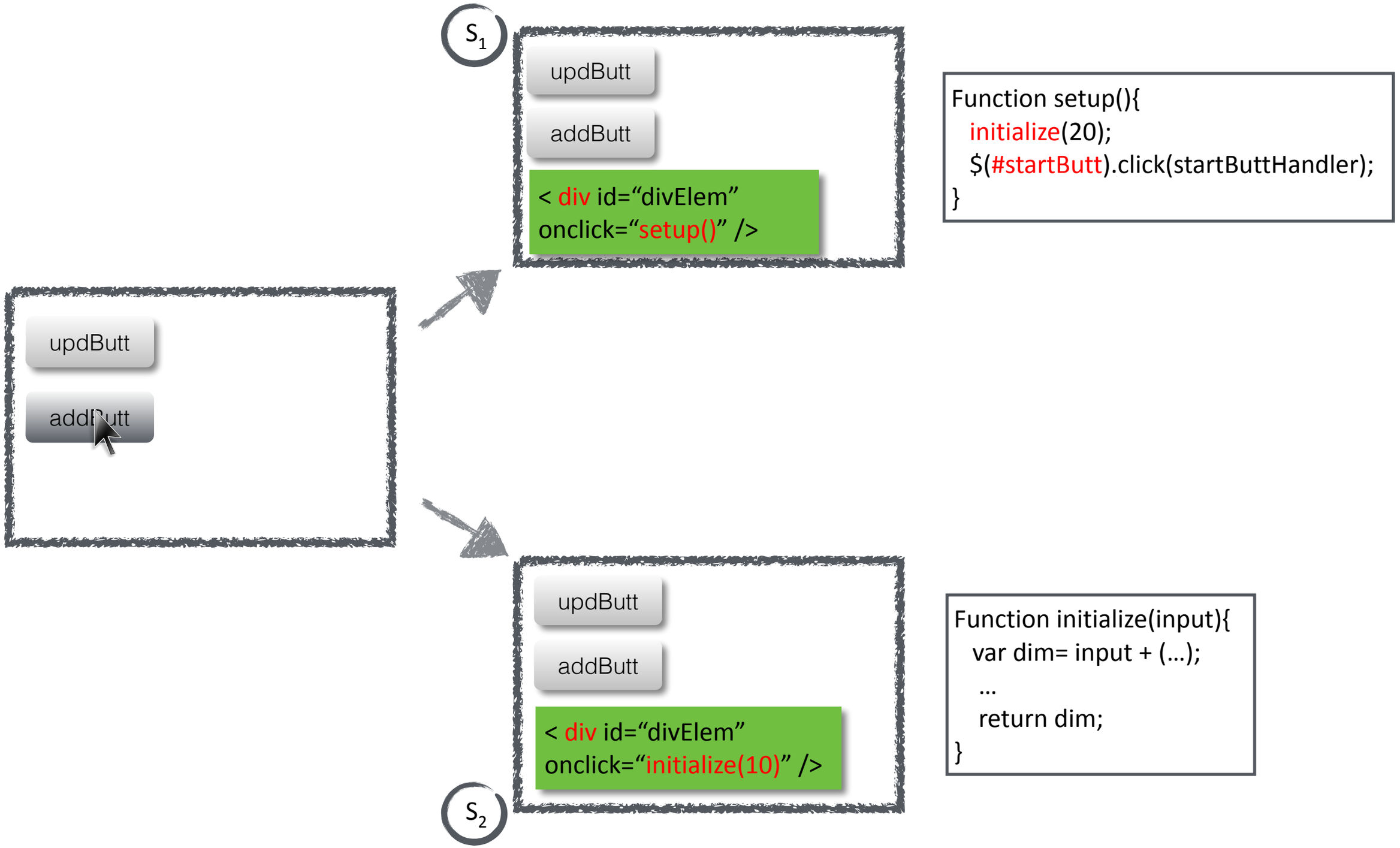
By

Maximizing the number of
(potential uncovered functions + potential clickable elements)


updButt
addButt




```
Function setup(){
  initialize(20);
  $(#startButt).click(startButtHandler);
}
```


S1
updButt
addButt
< div id="divElem"
onclick="setup()" />
Function setup(){
initialize(20);
$(#startButt).click(startButtHandler);
}
updButt
addButt
updButt
addButt
< div id="divElem"
onclick="initialize(10)" />
S2
Function initialize(input){
var dim= input + (…);
…
return dim;
}

Potential New Functions + Potential Clickables
MAXIMIZE THIS SUM!

S_1

updButt

addButt

< div id="divElem"
onclick="setup()" />

```
Function setup(){
  initialize(20);
  $(#startButt).click(startButtHandler);
}
```

updButt

addButt

S_2

updButt

addButt

< div id="divElem"
onclick="initialize(10)" />

```
Function initialize(input){
  var dim= input + (…);
  …
  return dim;
}
```

Potential new functions + Potential clickables

(setup + setDim) + (#startCell) = 3

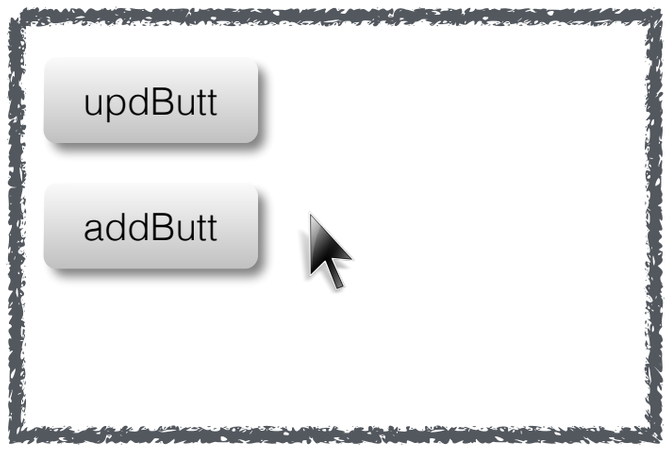
S_1

updButt

addButt

< div id="divElem"
onclick="setup()" />

```
Function setup(){
  initialize(20);
  $(#startButt).click(startButtHandler);
}
```



updButt

addButt

< div id="divElem"
onclick="initialize(10)" />

S_2

```
Function initialize(input){
  var dim= input + (…);
  …
  return dim;
}
```

Potential new functions + Potential clickables

(setup + setDim) + (#startCell) = 3

S_1

updButt

addButt

< div id="divElem"
onclick="setup()" />

```
Function setup(){
  initialize(20);
  $(#startButt).click(startButtHandler);
}
```

updButt

addButt

updButt

addButt

< div id="divElem"
onclick="initialize(10)" />

```
Function initialize(input){
  var dim= input + (…);
  …
  return dim;
}
```

S_2

Potential new functions + Potential clickables

(setDim) + (--) =1

Potential new functions + Potential clickables

(setup + setDim) + (#startCell) = 3

S_1

updButt

addButt

< div id="divElem"
onclick="setup()" />

```
Function setup(){
  initialize(20);
  $(#startButt).click(startButtHandler);
}
```

updButt

addButt

S_2

updButt

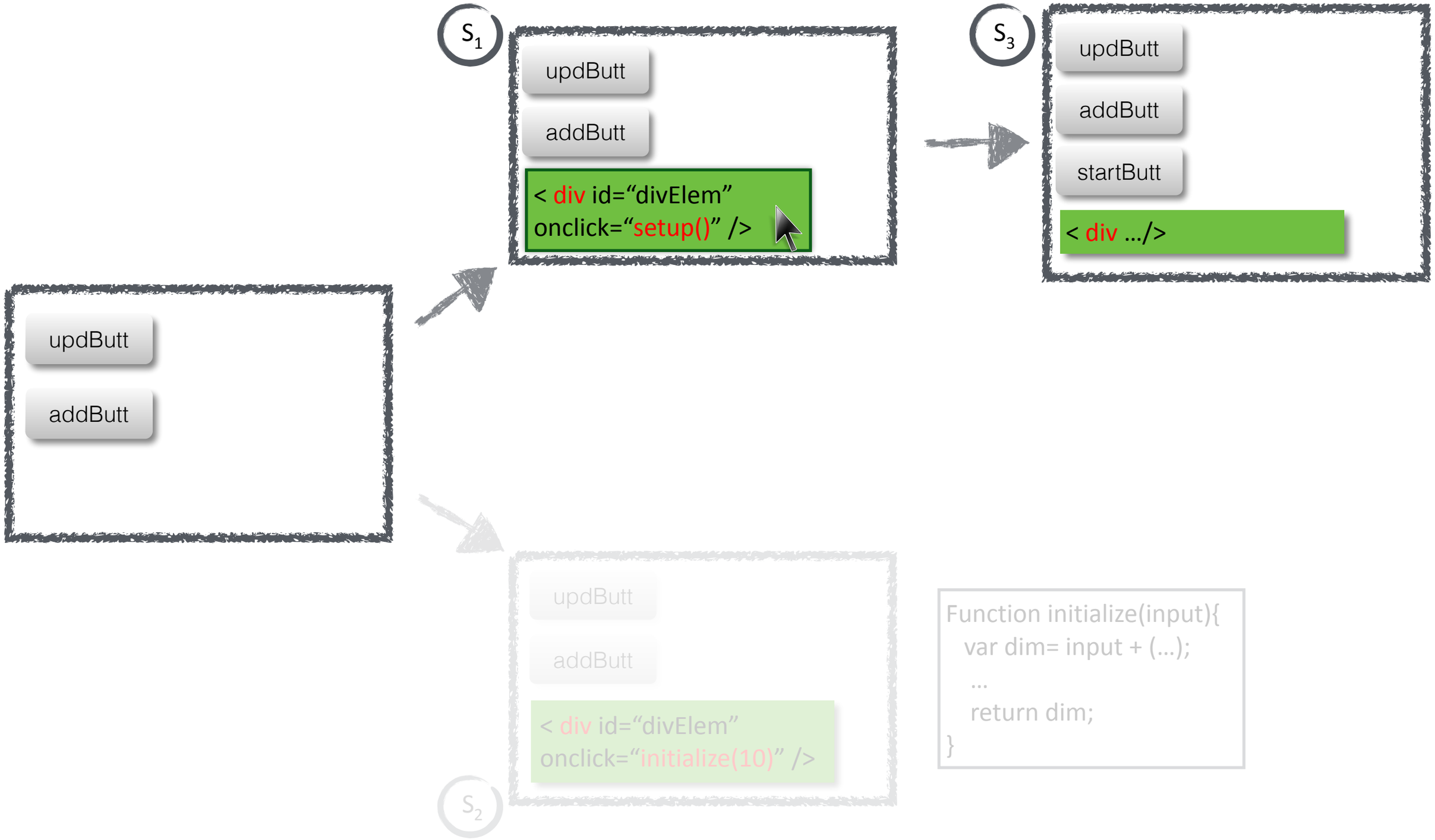
addButt

< div id="divElem"
onclick="initialize(10)" />

```
Function initialize(input){
  var dim= input + (…);
  …
  return dim;
}
```

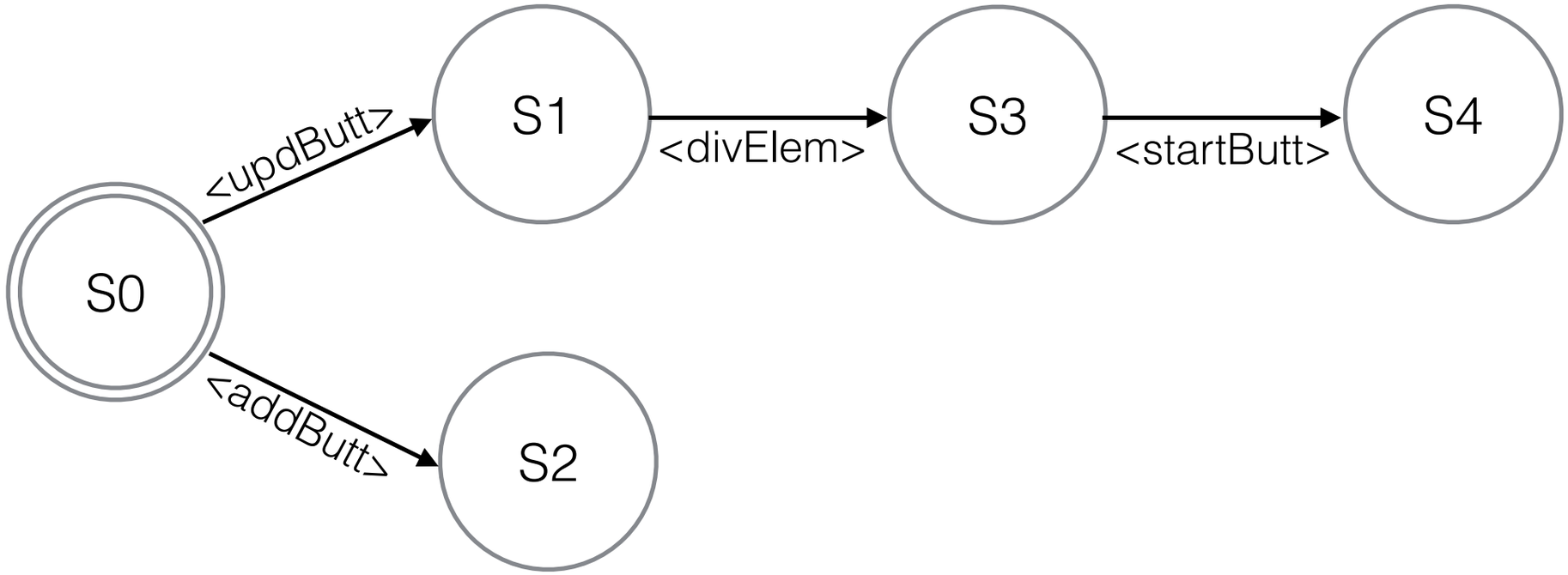
Potential new functions + Potential clickables

(setup + setDim) + (#startCell) = 3

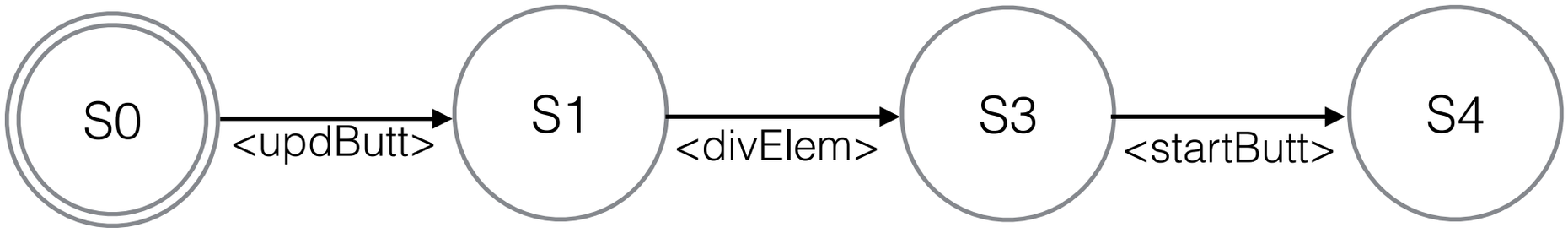



```
Function initialize(input){
  var dim= input + (…);
   …
   return dim;
}
```

Model Extraction and Function Coverage Maximization



Extract Event-based Tests



Selenium Test

```
@Test
public void testCase1(){
  WebElement updButt=driver.findElements(By.id("updButt" ));
  updButt.click();
  WebElement divElem=driver.findElements(By.id("divElem"));
  divElem.click();
  WebElement startButt=driver.findElements(By.id("startButt"));
  startButt.click();
}
```

Extract Unit-Level Tests

Function States

Logs at the entry point

- Input parameters
- Global variables used in the function
- Current DOM structure

Logs at the exit point

- Return value
- Accessed DOM elements and their attributes

Function State Abstraction

Huge number of function states

Negatively affect test suite comprehension

Hinders the scalability

Function State Abstraction

Branch Coverage

Accessed DOM Property

Return Value Type

```
function ButtonClicked() {
    var divTag = '<div id='divElem' />';
    if($(this).attr('id') == 'addButt')
        $('#addButt').after(divTag);
    else if($(this).attr('id') == 'updButt')
        $('#updButt').after(divTag);
}
```

Branch Coverage

Accessed DOM Property

Return Value Type

```
function ButtonClicked() {
    var divTag = '<div id='divElem' />';
    if($(this).attr('id') == 'addButt')
        $('#addButt').after(divTag);
    else if($(this).attr('id') == 'updButt')
        $('#updButt').after(divTag);
}
```

Branch Coverage

```
function removeClickable() {
    if(currentHeight <= 40)
        $(this).remove();
    ...
}
```

Accessed DOM Property

Return Value Type

```
function ButtonClicked() {
   var divTag = '<div id='divElem' />';
   if($(this).attr('id') == 'addButt')
      $('#addButt').after(divTag);
   else if($(this).attr('id') == 'updButt')
      $('#updButt').after(divTag);
}
```

Branch Coverage

```
function removeClickable() {
   if(currentHeight <= 40)
      $(this).remove();
   ...
}
```

Accessed DOM Property

```
function startButtHandler() {
   ...
   var dim=($('#divElem').width() +
         $(‘#divElem').height()))/dimension;
   dim += currentHeight;
   return dim;
}
```

Return Value Type

Branch Coverage
50%

currentHeight=20
dimension=80
currentWidth=60

currentHeight=30
dimension=50
currentWidth=20

currentHeight=20
dimension=80
currentWidth=60

currentHeight=70
dimension=60
<button id="add"/>
currentWidth=30

Accessed DOM Properties
[<#addButt> <#startButt>]

Branch Coverage
75%

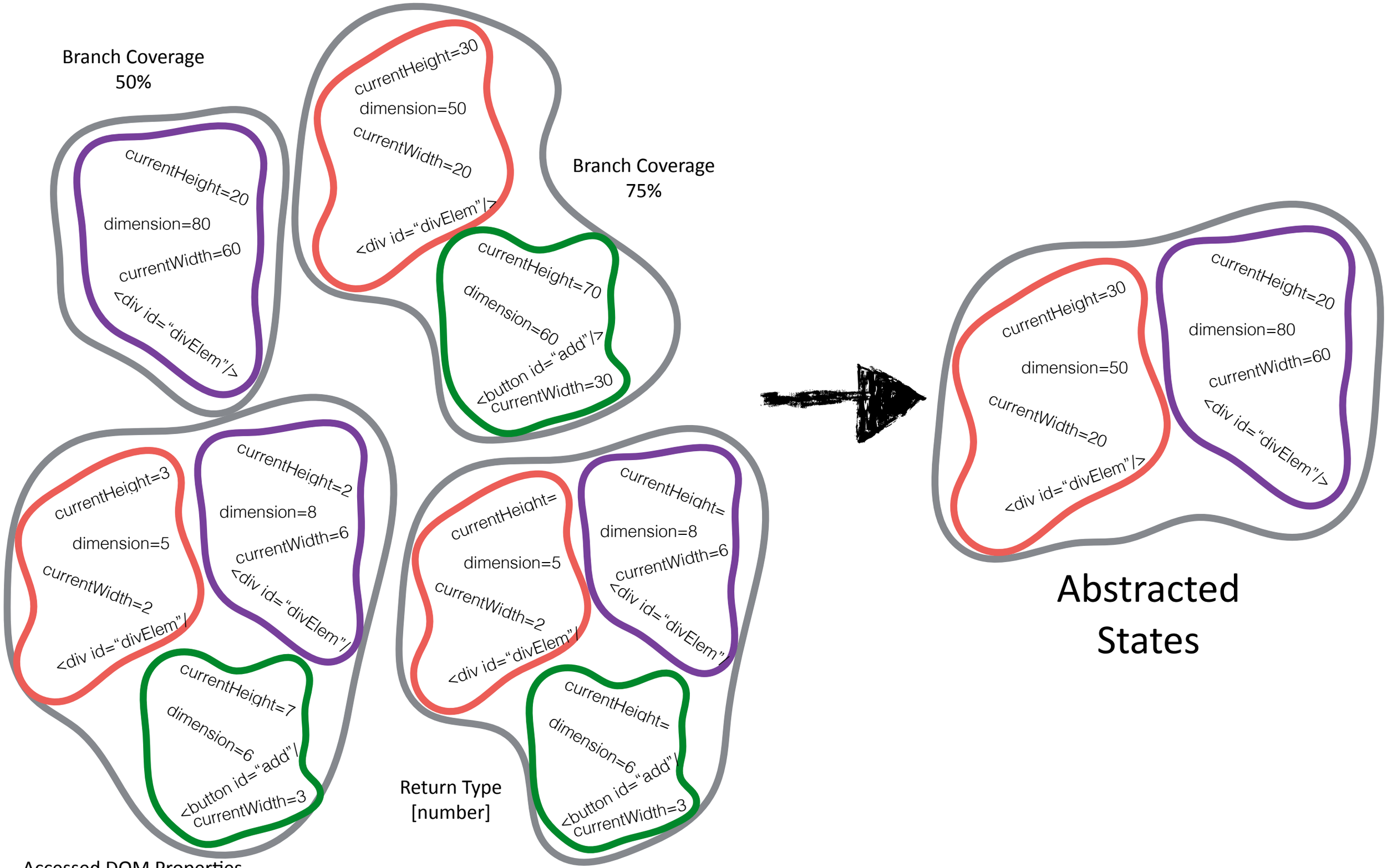
currentHeight=30
dimension=50
currentWidth=20

currentHeight=70
dimension=60
<button id="add"/>
currentWidth=30

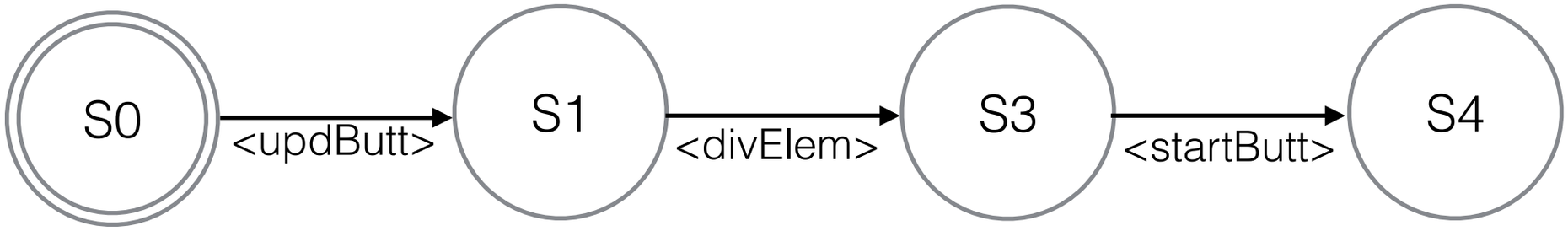
currentHeight=20
dimension=80
currentWidth=60

Return Type
[number]

Function State Abstraction



Extract Unit-Level Tests



Qunit Test

```
test("Testing startButtHandler”, 4, function(){
    var fixture = $("#qunit-fixture");
    fixture.append("<button id=\"updButt\"></button>
                    <button id=\"addButt\"></button>
                    <div id=\"divElem\"/>
                    style=\"height:200px;width:100px;\"/>");
    var currentDim=20;
    var result= startButtHandler(10);
});
```

Test Assertion Generation

DOM-Level Assertions

DOM mutation testing

DOM mutations target accessed elements during execution

Selectively compare DOM attributes affected by an injected fault

Test Assertion Generation

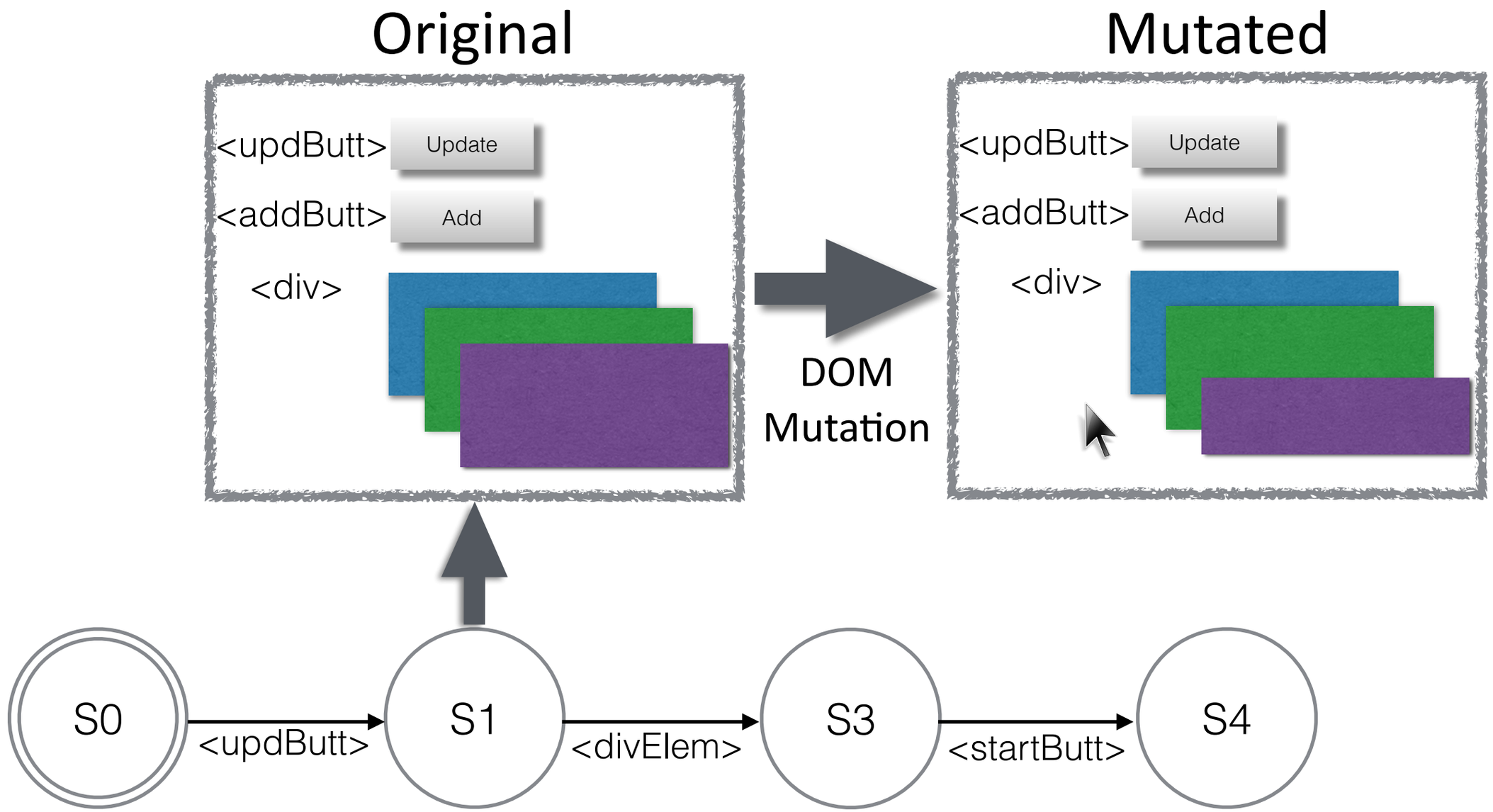
Unit-Level Assertions

Code-level mutations

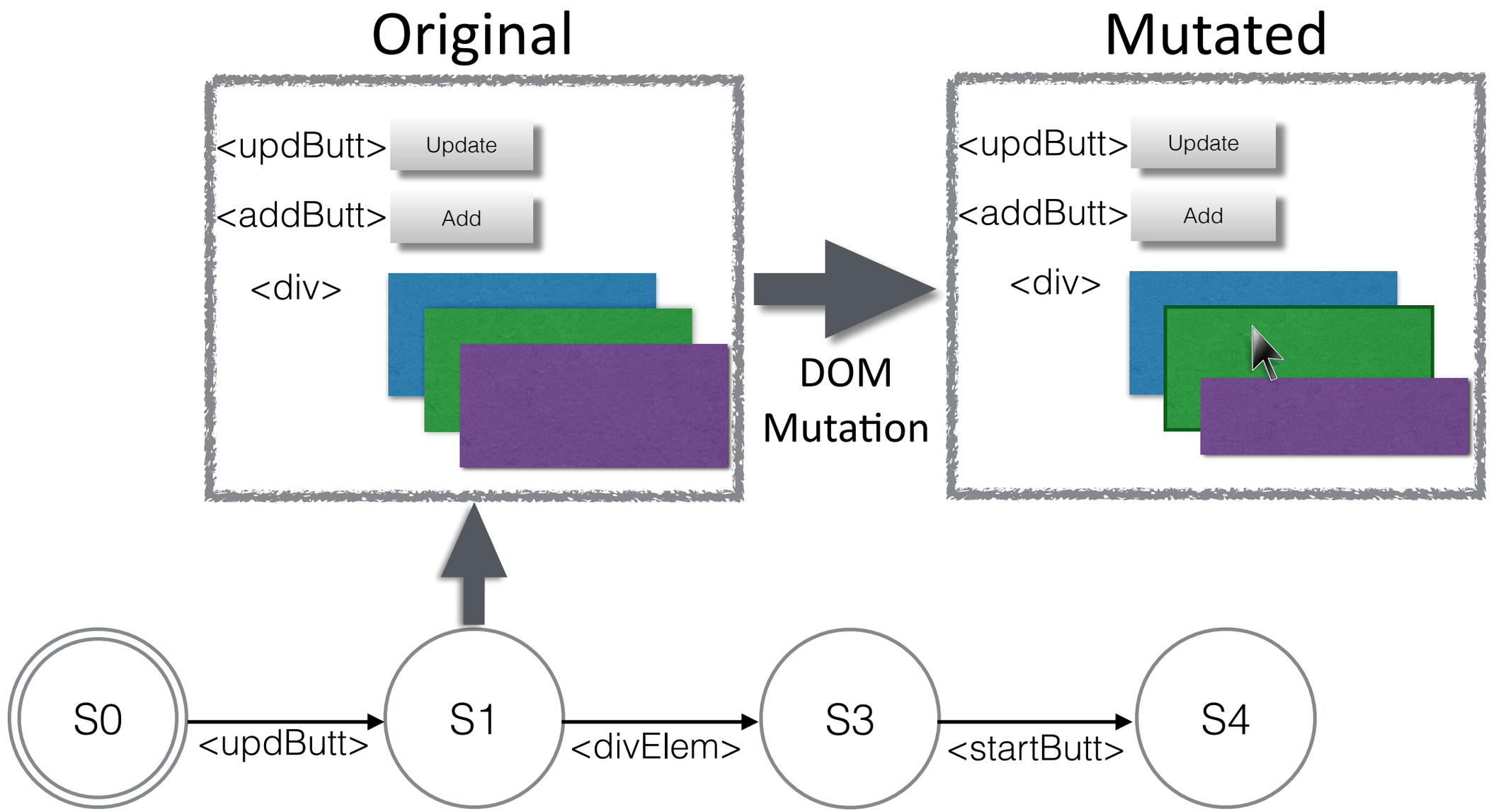
Common code-level mutation operators

(changing the value of a variable, modifying a conditional statement, …)

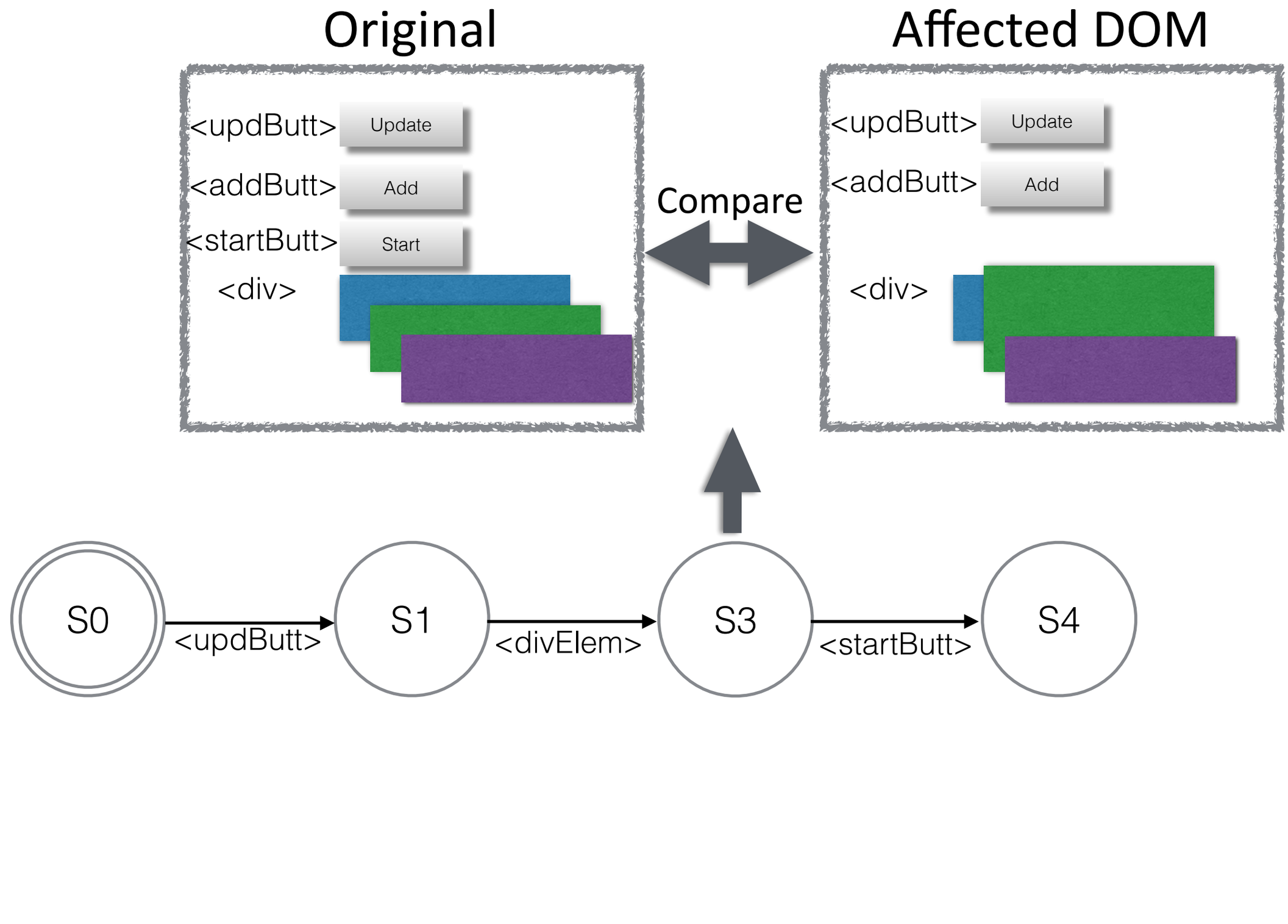
DOM-Level Mutation

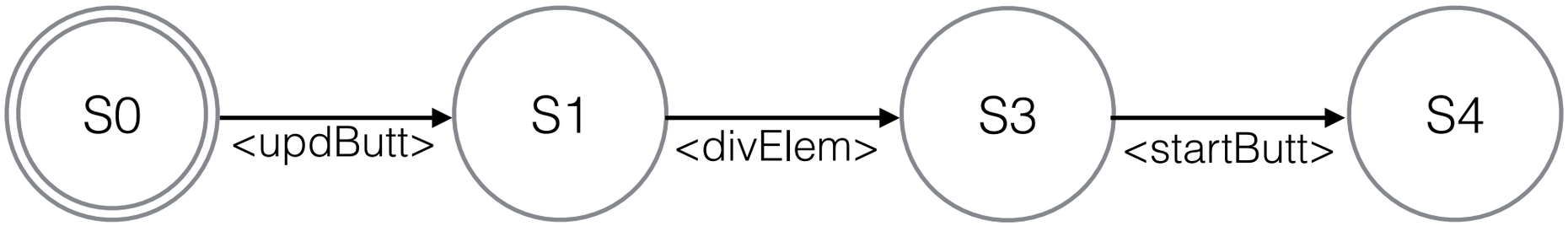


DOM-Level Mutation



DOM-Level Mutation





Selenium Test and Assertion

```
@Test
public void testCase1(){
  WebElement updButt=driver.findElements(By.id("updButt" ));
  updButt.click();
  WebElement divElem=driver.findElements(By.id("divElem"));
  divElem.click();
  boolean exists=driver.findElements(By.id(“startButt”)).size!=0;
  assertTrue(exists);
  WebElement startButt=driver.findElements(By.id("startButt"));
  startButt.click();
  int divElemHeight=driver.findElements(By.id("divElem")).getSize().height;
  assertEquals(divElemHeight, 50);
}
```

Code-Level Mutation

```
currentHeight=0;
currentWidth=20;
function startButtHandler(dimension) {
        currentHeight=10;
        var dim=($('#divElem').width() +
                $('#divElem').height()))/dimension;
      currentWidth += dim;
      dim+=currentHeight;
      return dim;
}
```

Code-Level Mutation

```
currentHeight=0;
currentWidth=20;
function startButtHandler(dimension) {
    currentHeight=10;
    var dim=($('#divElem').width() +
            $('#divElem').height()))/dimension;
    currentWidth += dim;
    dim+=currentHeight;
    return dim;
}
```

Code-Level Mutation

Mutated

```
currentHeight=0;
currentWidth=20;
function startButtHandler(dimension) {
    currentHeight="10";
    var dim=($('#divElem').width() +
            $('#divElem').height()))/dimension;
    currentWidth += dim;
    dim+=currentHeight;
    return dim;
}
```

Code-Level Mutation

Mutated

```
currentHeight=0;
currentWidth=20;
function startButtHandler(dimension) {
    currentHeight="10";
    var dim=($('#divElem').width() +
            $('#divElem').height()))/dimension;
    currentWidth += dim;
    dim+=currentHeight;
    return dim;
}
```

Unit Test and Assertion

```
currentHeight=0;
currentWidth=20;
function startButtHandler(dimension) {
    currentHeight=“10”;
    var dim=($('#divElem').width() +
            $('#divElem').height()))/dimension;
    currentWidth += dim;
    dim+=currentHeight;
    return dim;
}
```

Qunit Test

```
test("Testing startButtHandler”, 2, function(){
    var fixture = $("#qunit-fixture");
    fixture.append("<button id=\"updButt\"></button>
                    <button id=\"addButt\"></button>
                    <div id=\"divElem\"/>
                    style=\"height:200px;width:100px;\"/>");
    currentHeight=0;
    currentWidth=20;
    var result= startButtHandler(10);
    equal(typeof(result), ‘number’);
    equal(result, 30);
});
```

ASSERTIONS

Evaluation and Results

- Code coverage

- Fault finding capability

- Comparison with Artemis

  (Feedback-directed automated random test generation technique)

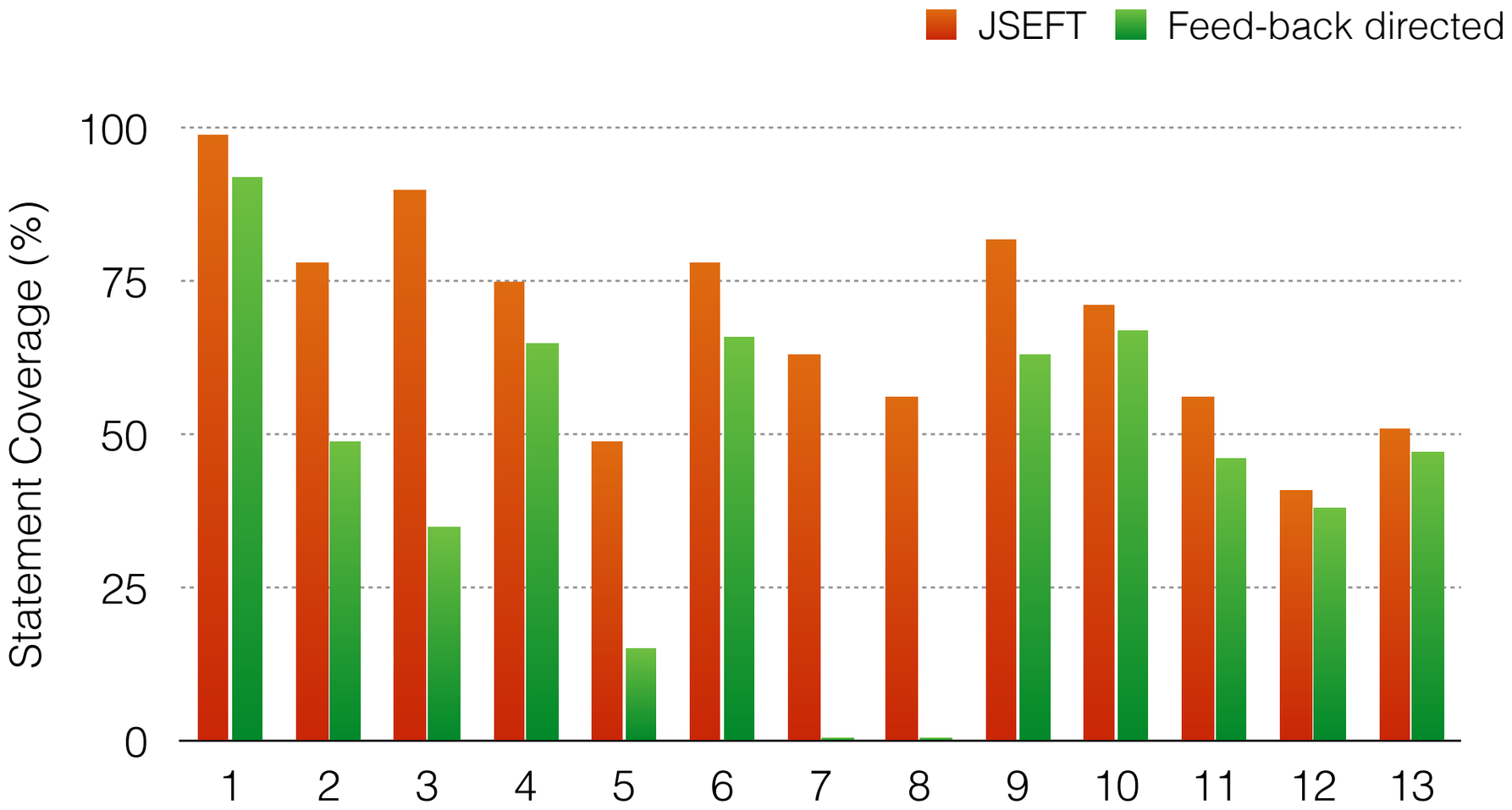
  [S. Artzi, ICSE'11]

13 open-source web applications

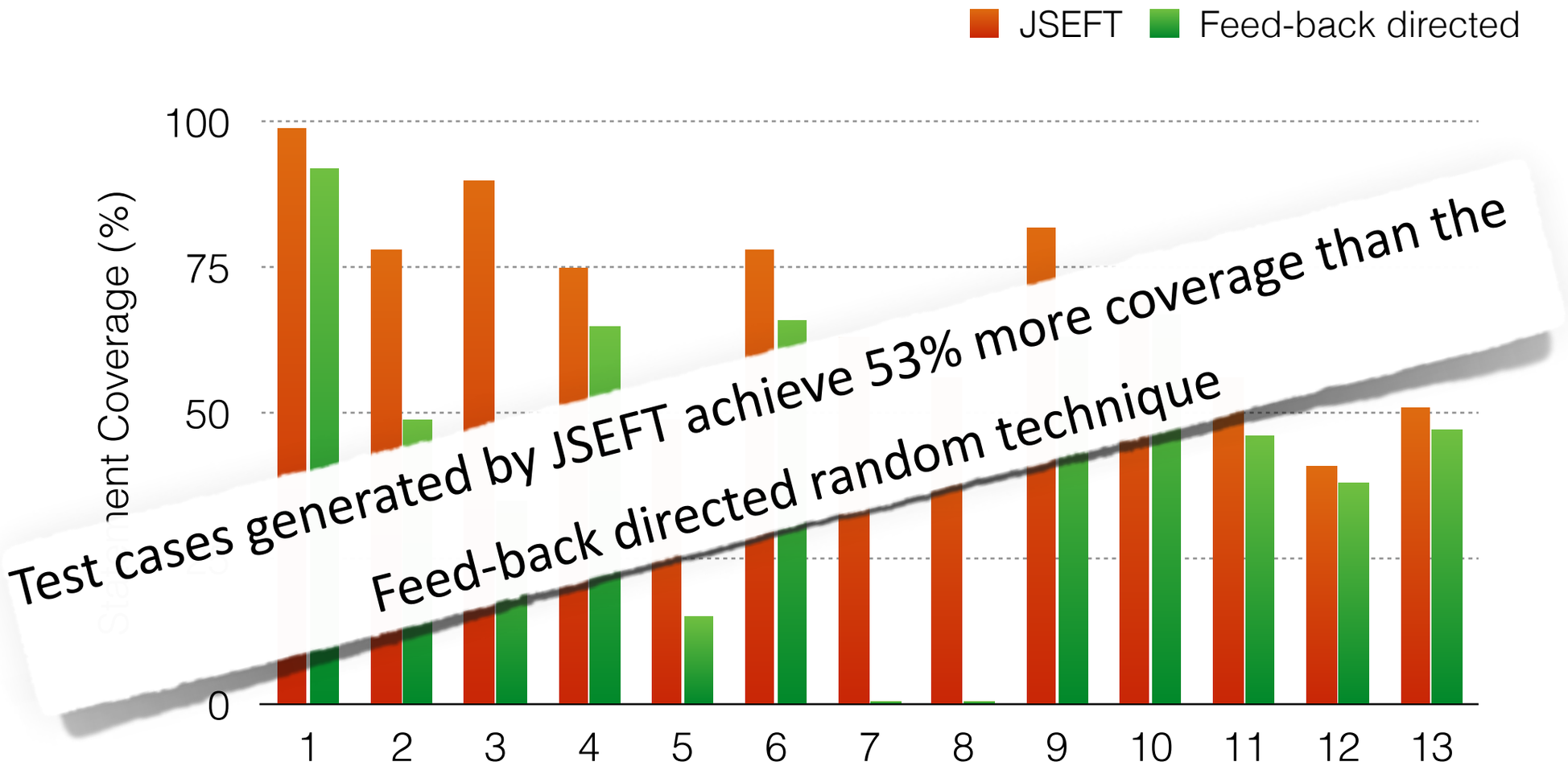
Automatically seeding each application with 50 random faults

Remove the mutant if the mutation used for the evaluation and generating oracles happen to be the same

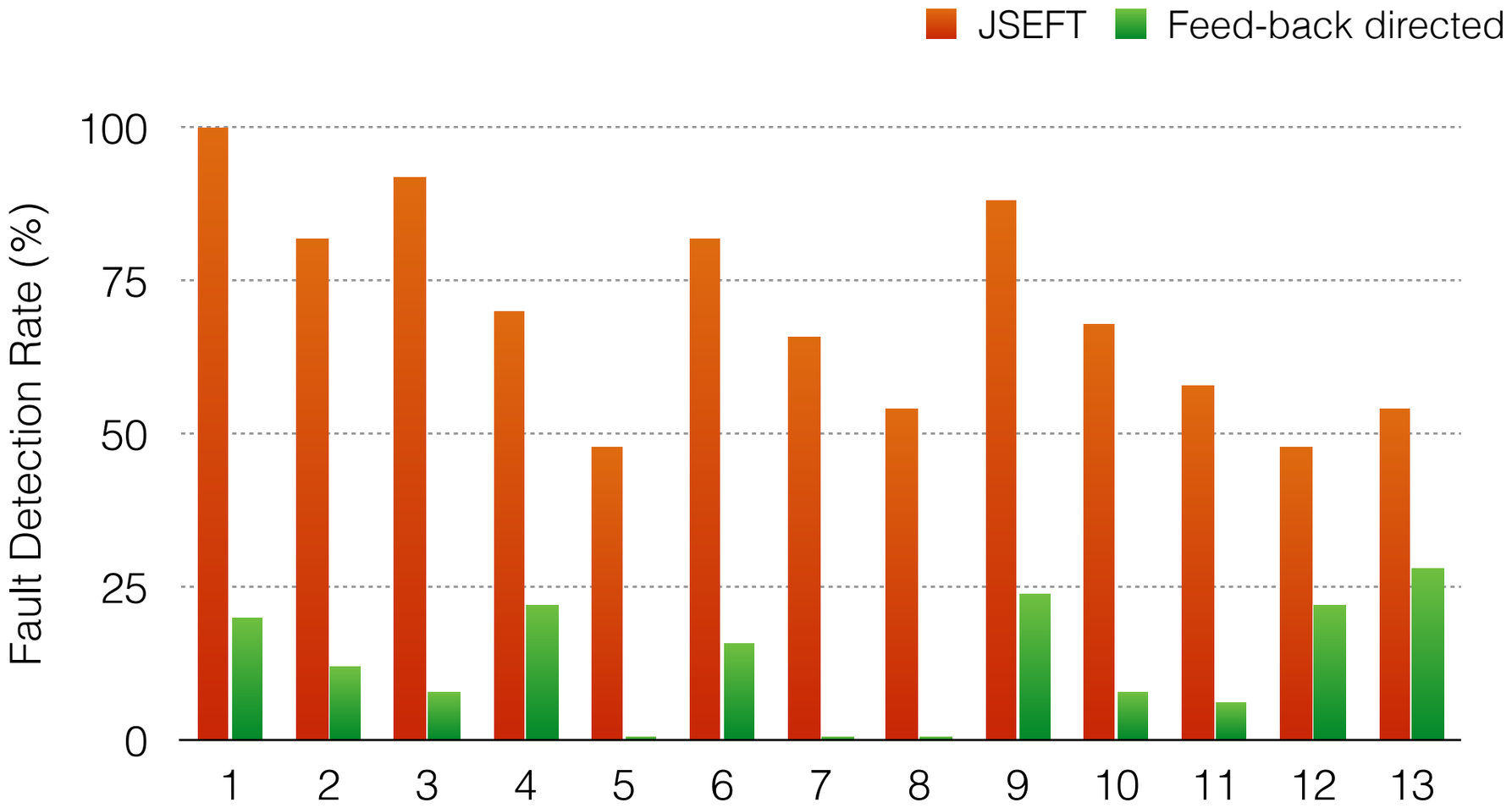
Statement Coverage



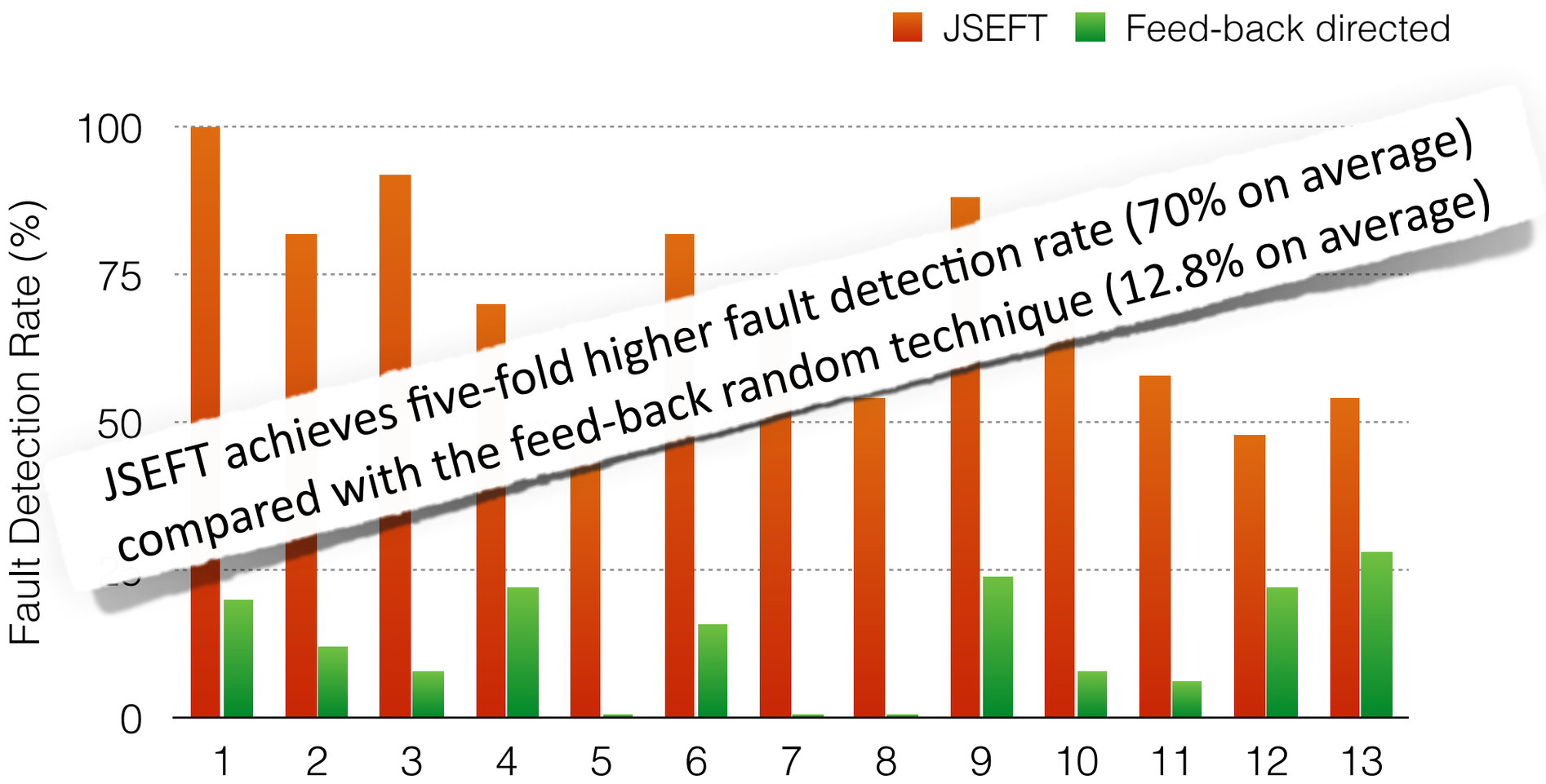
Statement Coverage



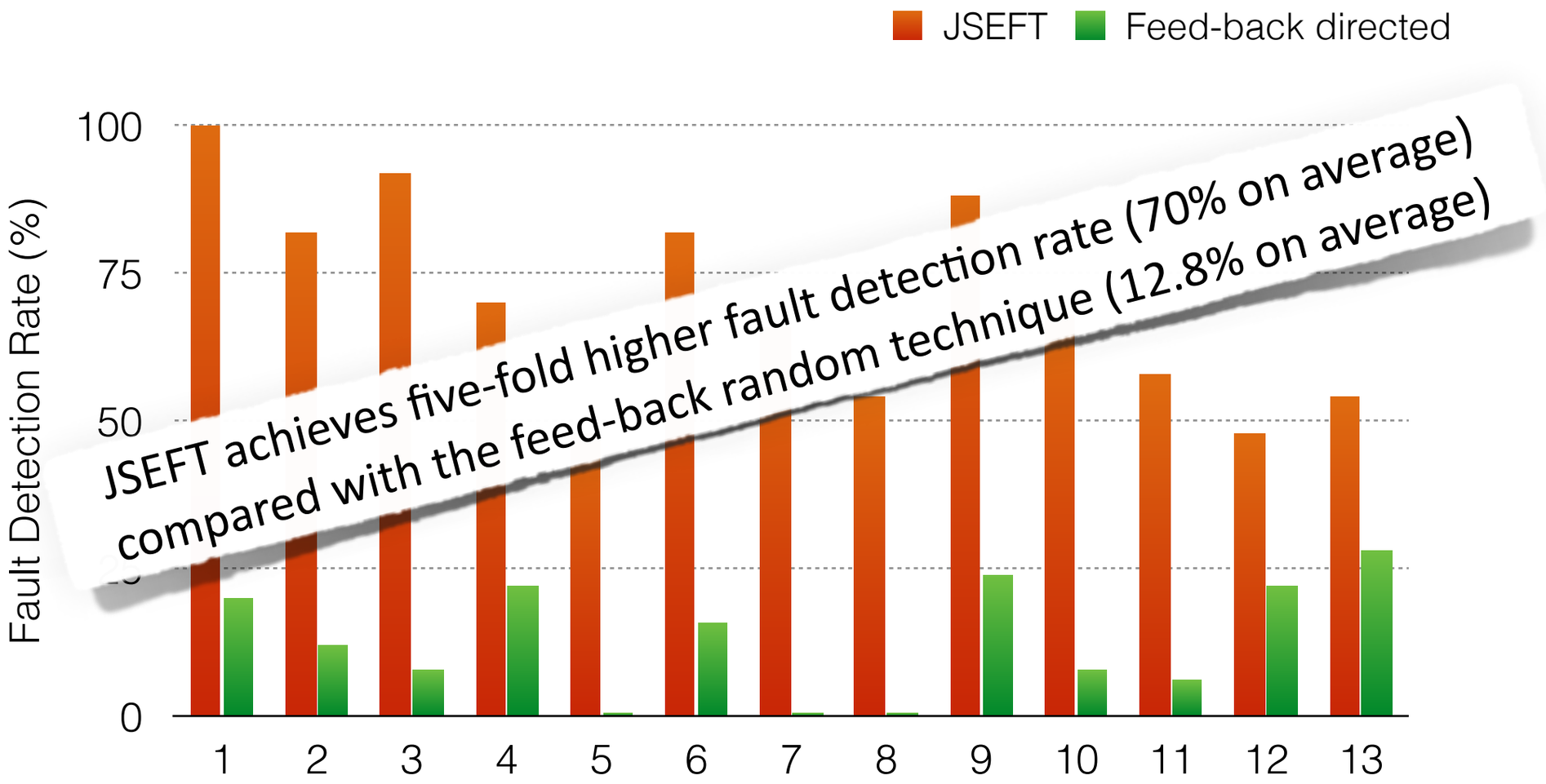
Fault Finding Capability

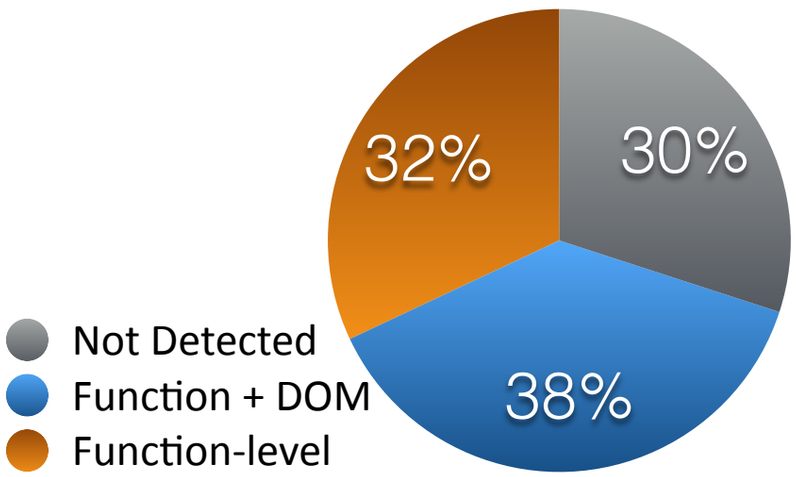


Fault Finding Capability



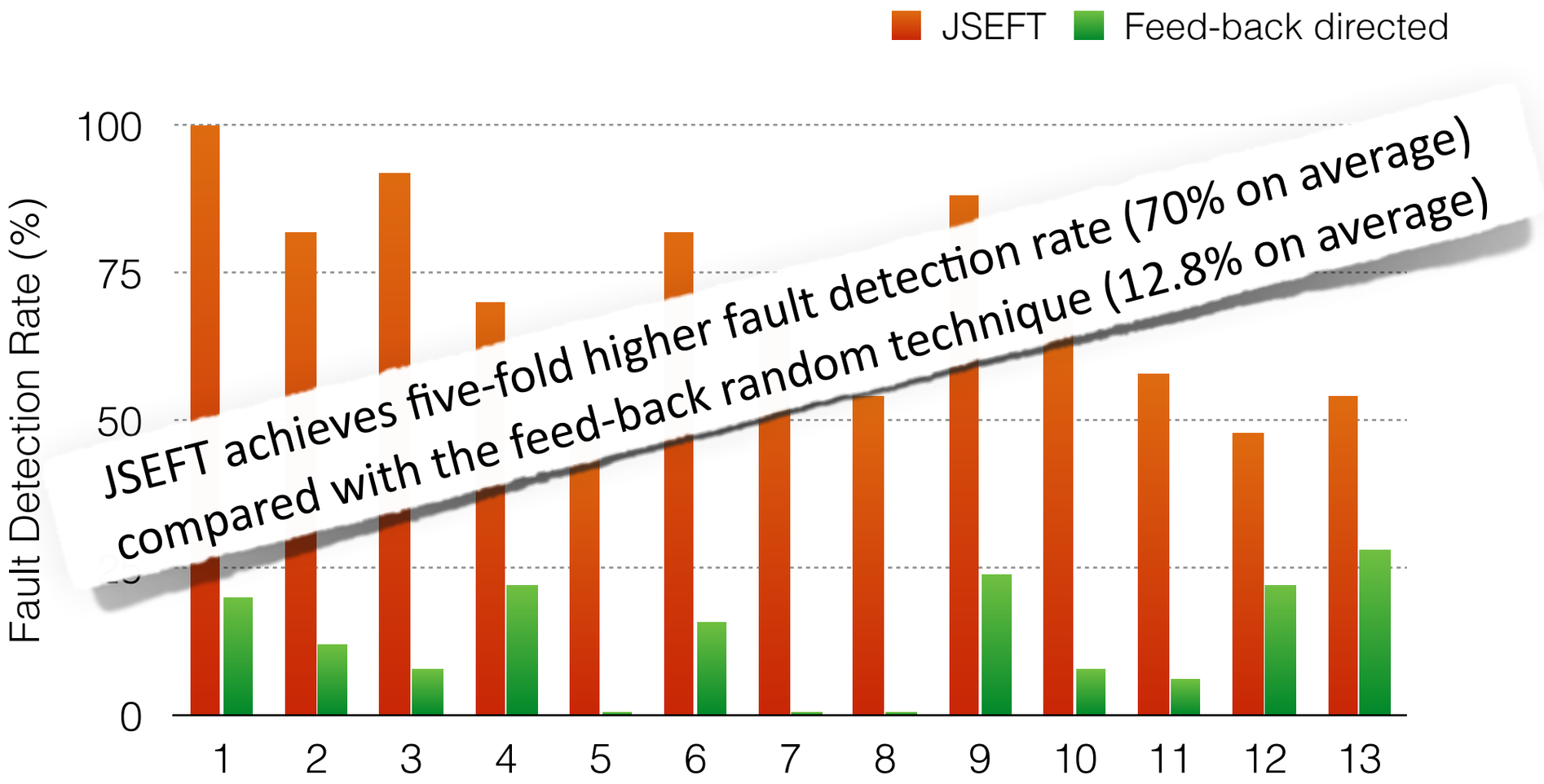
Fault Finding Capability

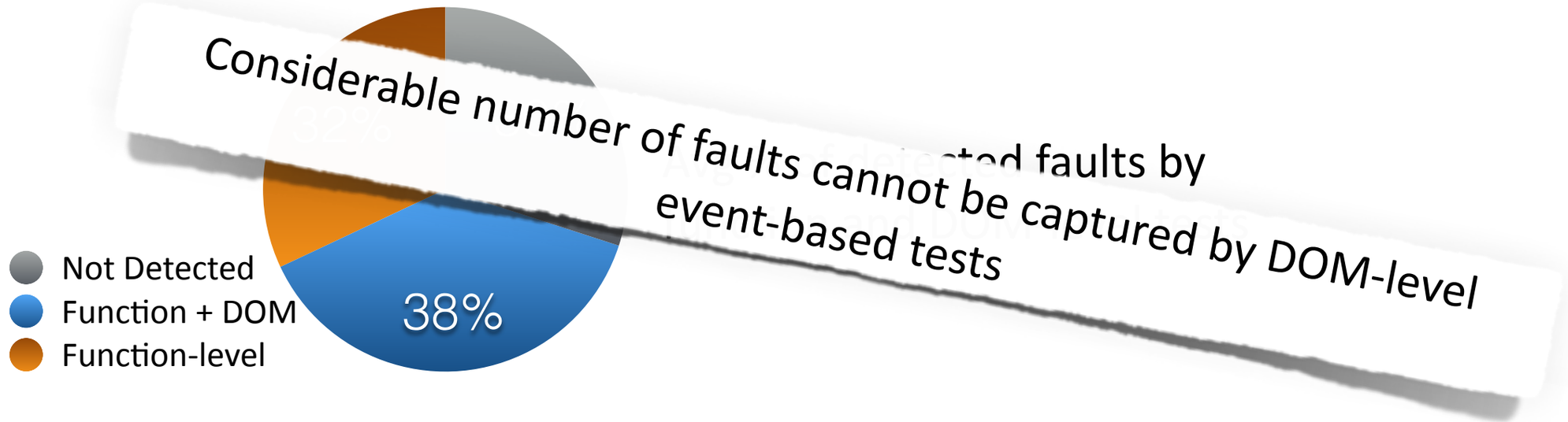




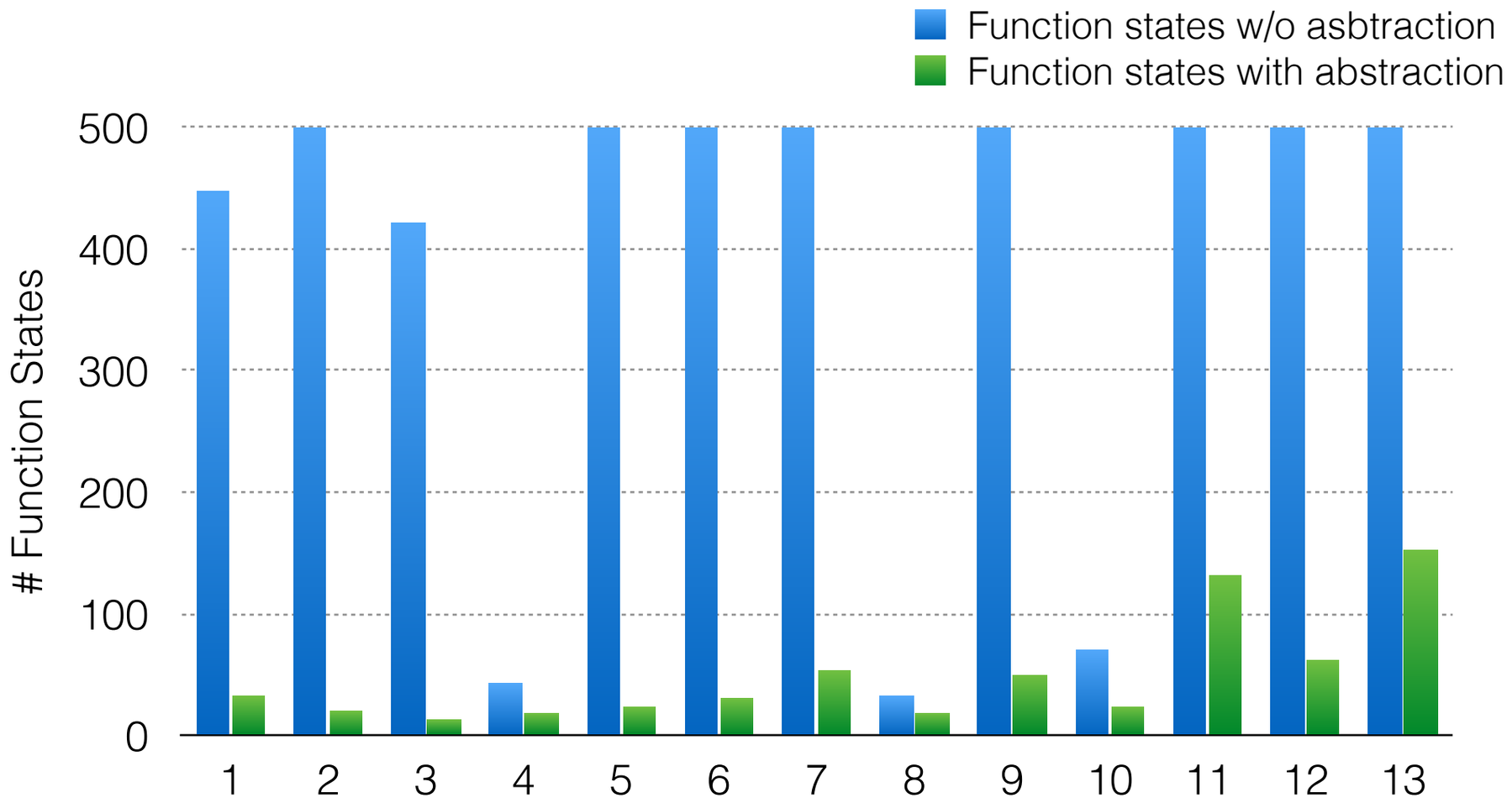
Avg % of detected faults by function and DOM-level tests

Fault Finding Capability

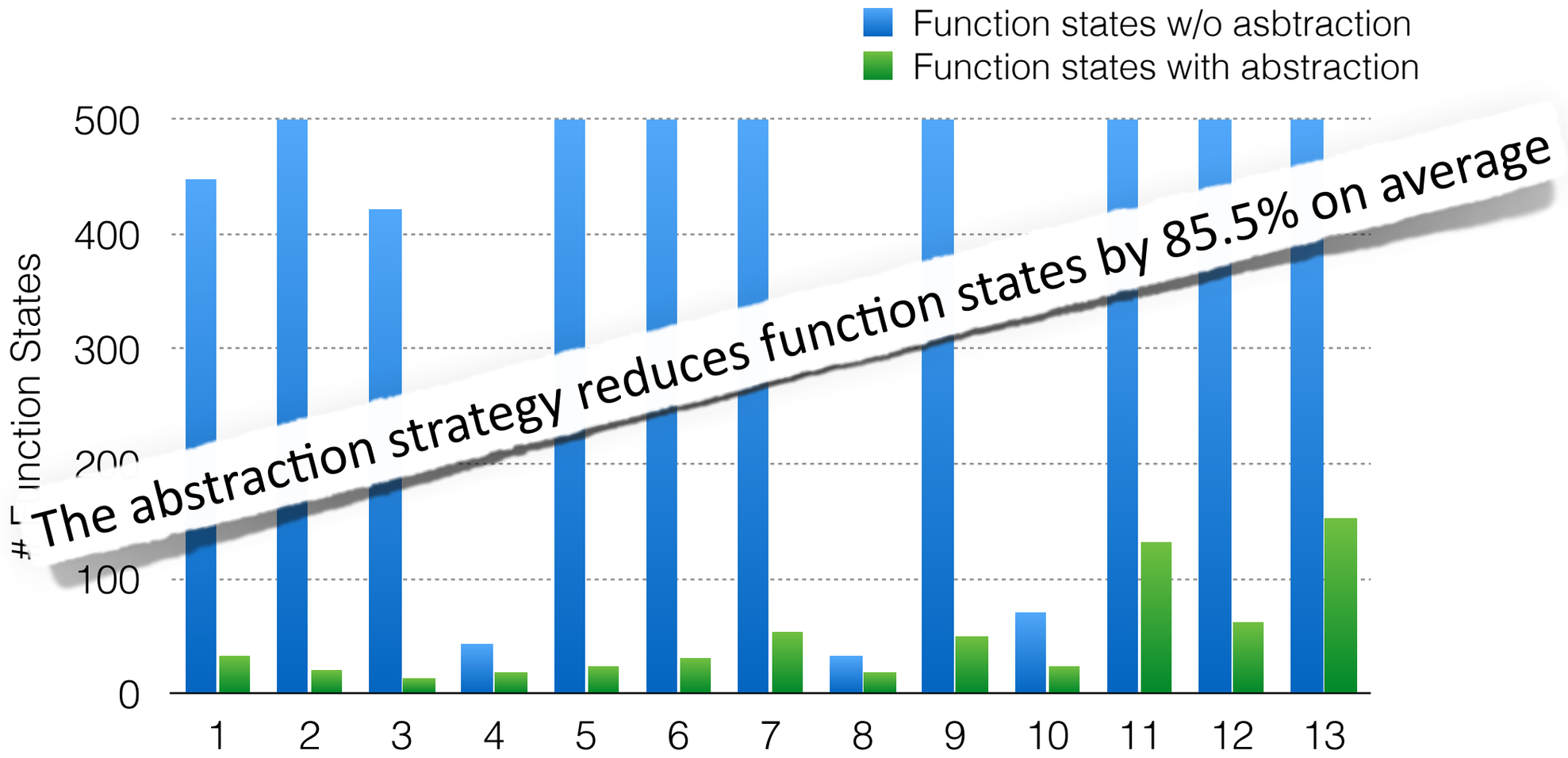




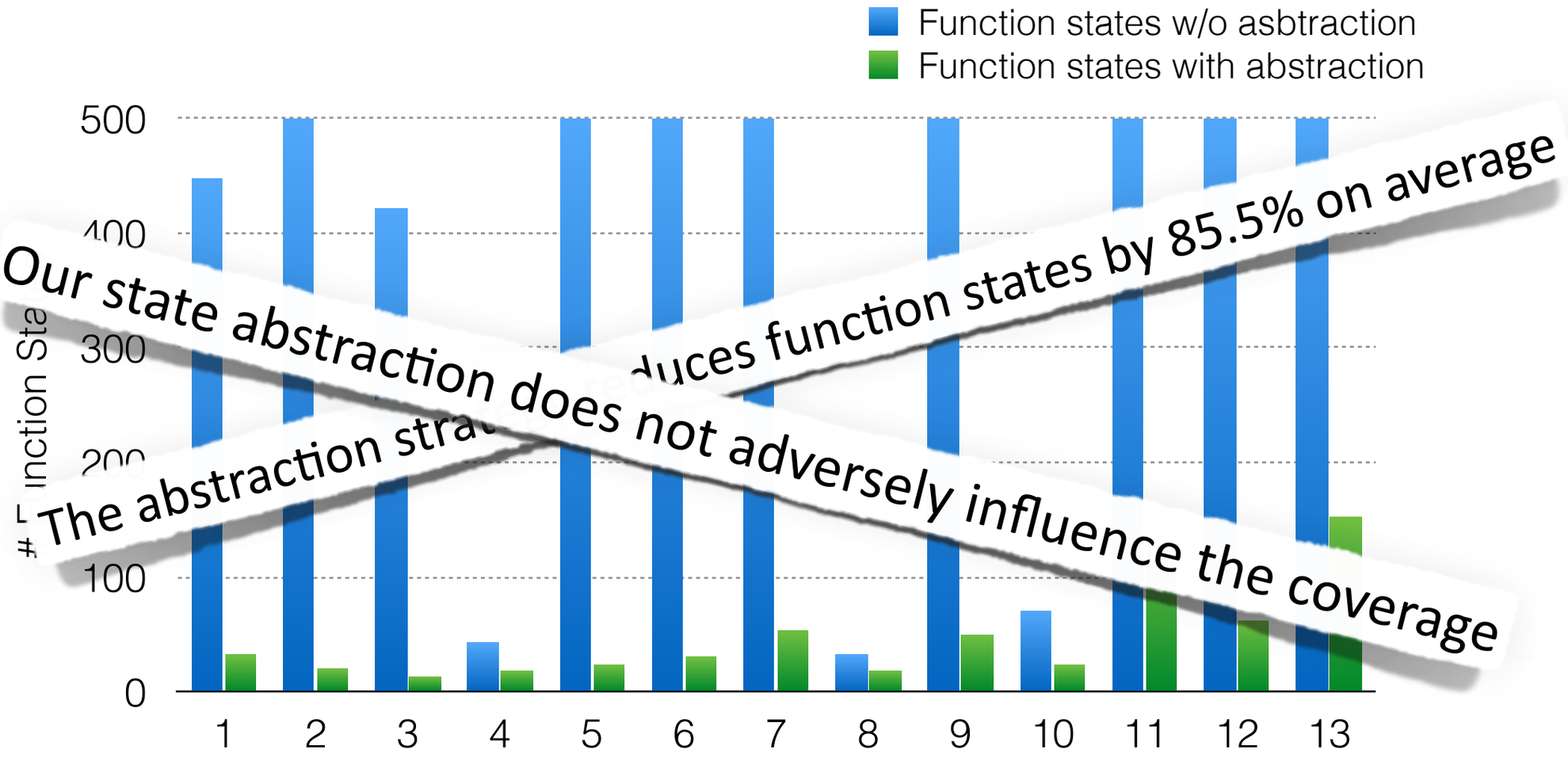
Function State Abstraction



Function State Abstraction



Function State Abstraction



Summary

Automate test and oracle generation for JavaScript applications

Detecting regression JavaScript and DOM-level faults

Implementation: JSEFT (JavaScript Event and Function Testing)

DOM-level event sequences and assertions

Check the application's behaviour from an end user's perspective

Function-level unit tests and assertions

Verify the functionality of JavaScript code at the function level

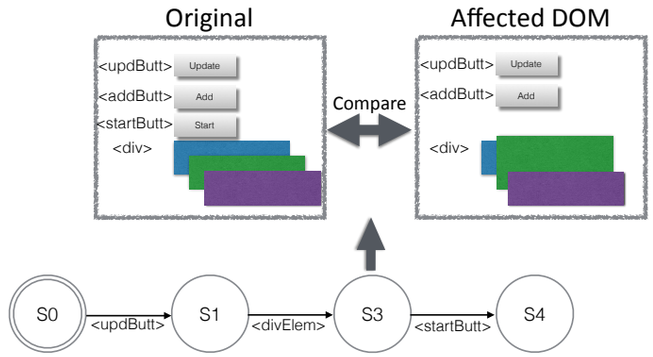
Extract Event-based Tests

Selenium Test

```
@Test
public void testCase1(){
  WebElement updButt=driver.findElements(By.id("updButt" ));
  updButt.click();
  WebElement divElem=driver.findElements(By.id("divElem"));
  divElem.click();
  WebElement startButt=driver.findElements(By.id("startButt"));
  startButt.click();
}
```

25

DOM-Level Mutation



Unit Test and Assertion

```
currentHeight=0;
currentWidth=20;
function startButtHandler(dimension) {
    currentHeight="10";
    var dim=($('#divElem').width() +
            $('#divElem').height()))/dimension;
    currentWidth += dim;
    dim+=currentHeight;
    return dim;
}
```

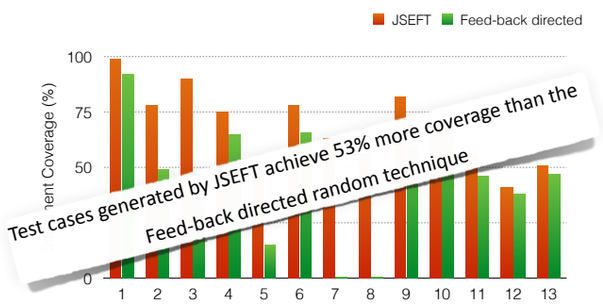
Qunit Test

```
test("Testing startButtHandler", 2, function(){
    var fixture = $("#qunit-fixture");
    fixture.append("<button id=\"updButt\"></button>
                    <button id=\"addButt\"></button>
                    <div id=\"divElem\"/>
                    style=\"height:200px;width:100px;\"/>");

    currentHeight=0;
    currentWidth=20;
    var result= startButtHandler(10);

    equal(typeof(result), 'number');
    equal(result, 30);
});
```

ASSERTIONS

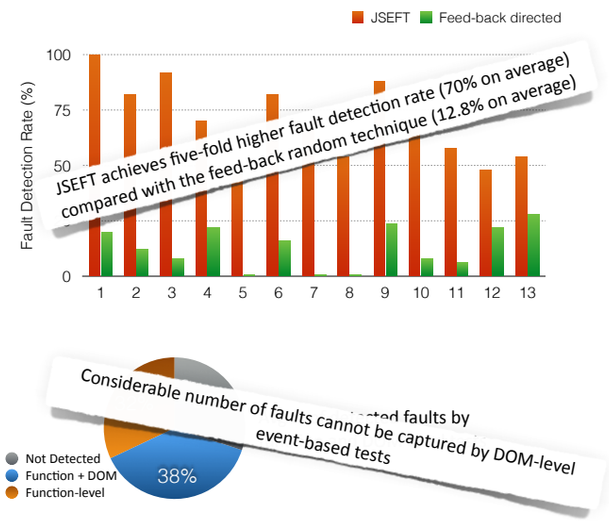
44

Statement Coverage



48

Fault Finding Capability



52