

JExample

Lea Hänsenberger and Adrian Kuhn

Why testing **Stack** is not trivial.

How to write better tests, quickly locate bugs and manage your fixtures.

@Before

```
public void setup() {  
    stack = new Stack();  
}
```

@Test

```
public void testPush() {  
    stack.push("Foo");  
    assertEquals(1, stack.size());  
}
```

@Test

```
public void testPop() {  
    stack.push("Foo");  
    Object elem = stack.pop();  
    assertEquals("Foo", elem);  
}
```

```
@Test
public Stack setup() {
    stack = new Stack();
    assertTrue(stack.isEmpty());
    return stack;
}
@Test
@Depends("setup")
public Stack testPush(Stack stack) {
    stack.push("Foo");
    assertEquals(1, stack.size());
    return stack;
}
@Test
@Depends("testPush")
public void testPop(Stack stack) {
    Object elem = stack.pop();
    assertEquals("Foo", elem);
}
```

For example "an empty stack" do

```
stack = []
```

```
assert stack.empty?
```

```
return stack
```

end

For example "pushing an element" do

```
stack = given "an empty stack"
```

```
stack.push "Foo"
```

```
assert stack.size == 1
```

```
return stack
```

end

For example "popping an element" do

```
stack = given "pushing an element"
```

```
elem = stack.pop
```

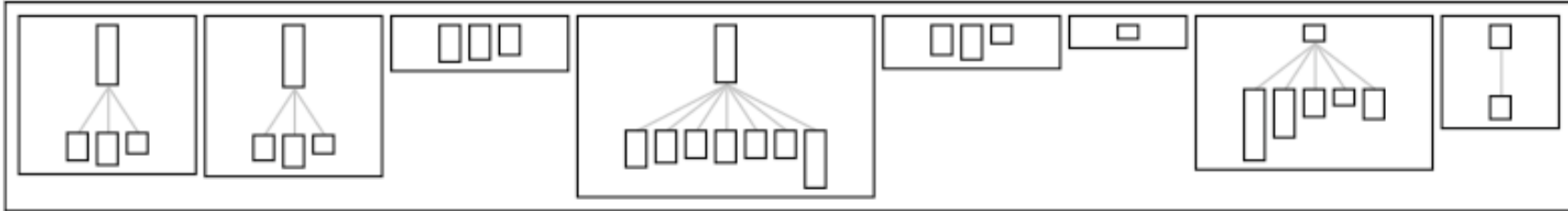
```
assert elem == "Foo"
```

```
assert stack.empty?
```

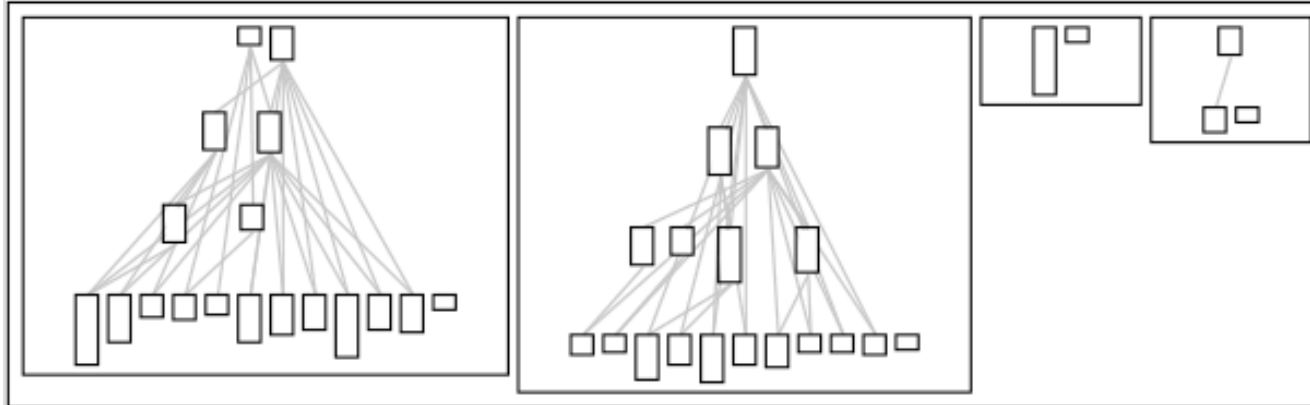
end

Quickly locate bugs!

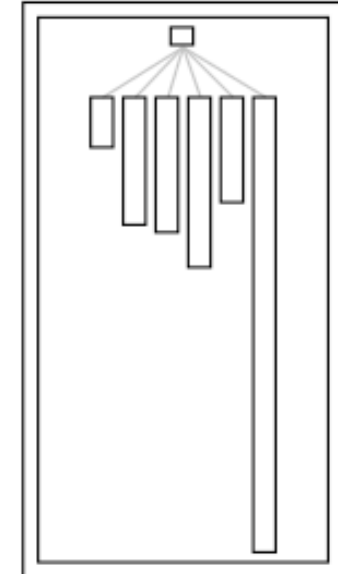
UJ = traditional JUnit implementation



UC = chained JExample implementation

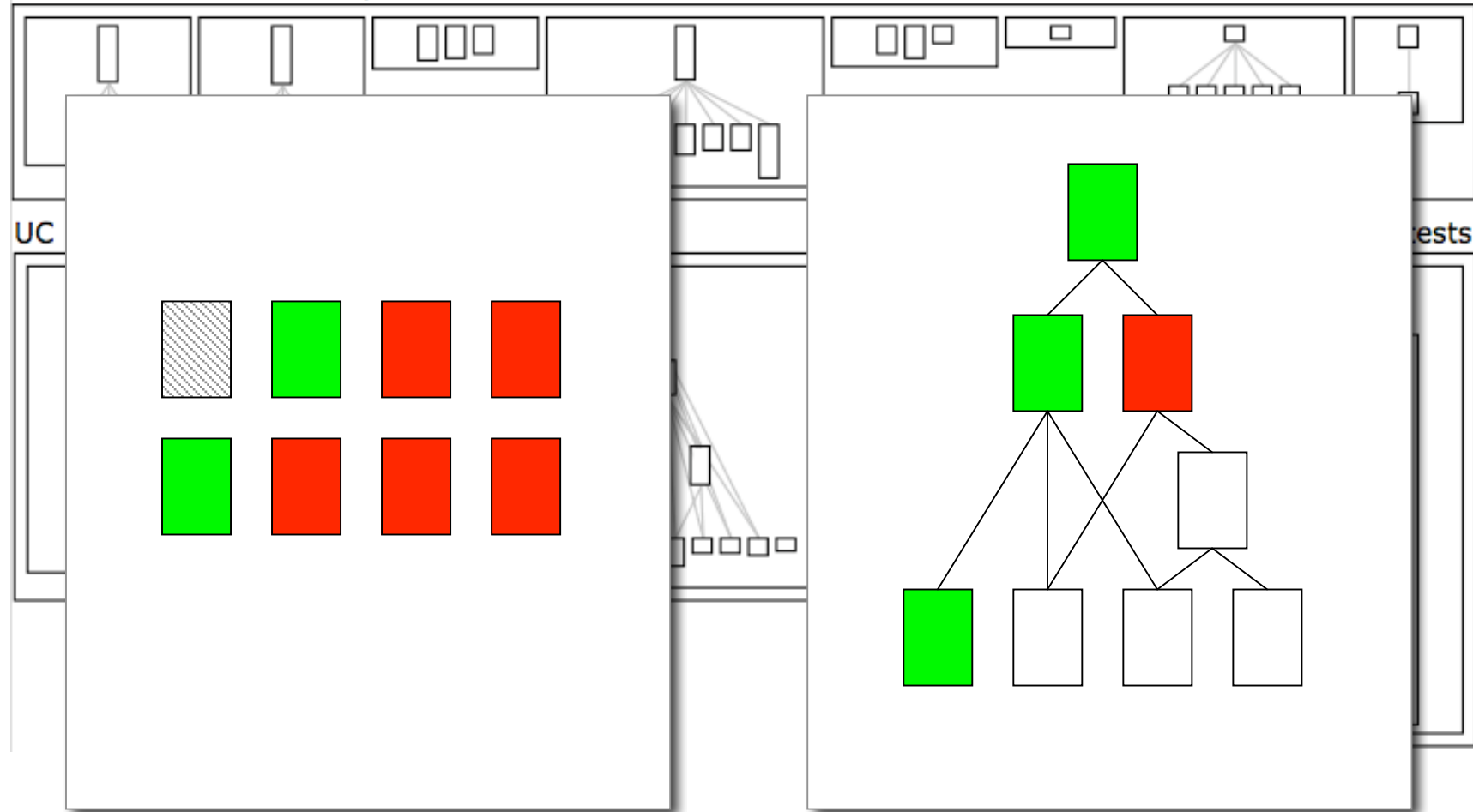


UO = original tests



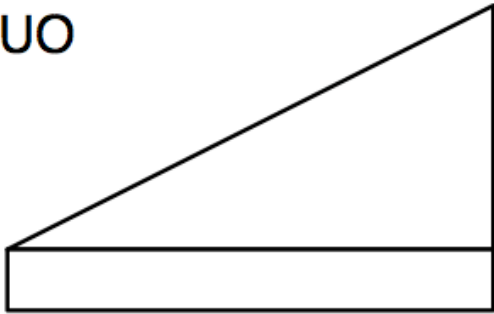
Quickly locate bugs!

UJ = traditional JUnit implementation

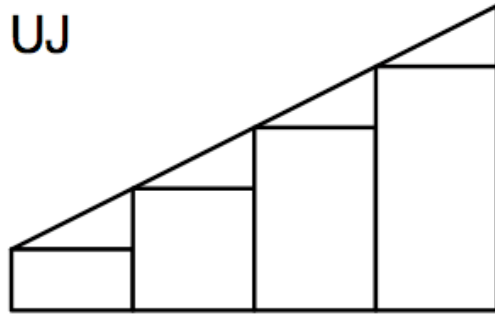


Manage your fixtures!

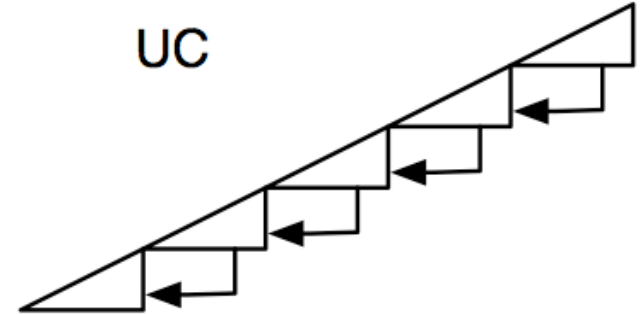
UO



UJ



UC



<http://scg.unibe.ch/Research/JExample/>