How Test Automation Drives Agile Software Development

September 10, 2015
Austin, TX

IEEE

CEDA
IEEE Council on Electronic Design Automation

Chris Durand
CTO, Bridge360

About Me

Chris Durand

CTO, Bridge360

Chris_Durand@Bridge360.com

www.Bridge360.com

www.Bridge360Blog.com

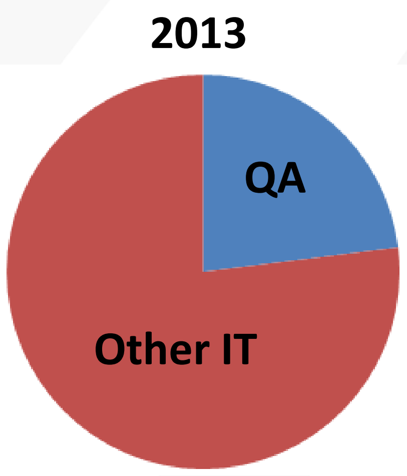
Agenda

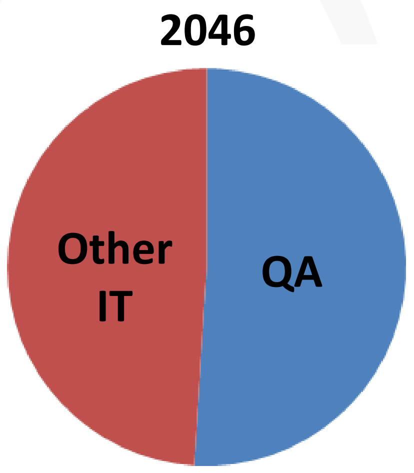
- Money, Money, Money!
- Traditional Testing Review
- Agile & Scrum Review
- How to Do Automated Testing Right
- The Future!
- Additional Resources

Money, Money, Money!

QA Spend

- QA is becoming a larger and larger % of IT and project budgets*
  - Average QA budget is 23% of IT budget
  - QA budget growing 5% per year vs. 2-3% growth in overall IT budgets ➔ not sustainable





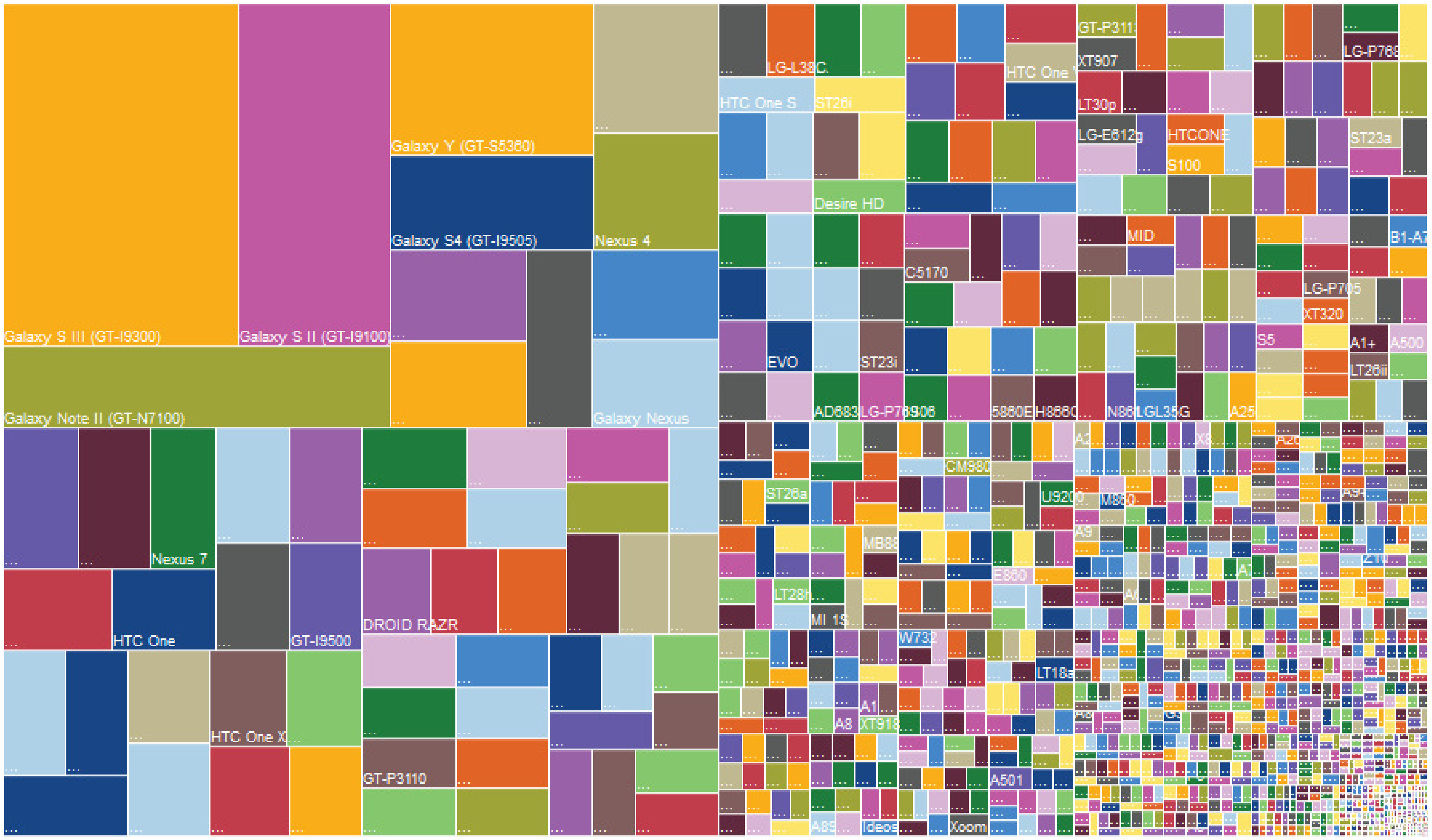
*World Quality Report 2013-14: http://www.slideshare.net/capgemini/world-quality-report-2013-14

Why is QA spend **so high**?

Increasing Application Complexity

- Ever larger and complicated applications
  - More complicated ecosystem for applications / more interconnections
- More environments
  - Mobile platforms (iOS, Android, Windows Phone)
  - Browsers (Chrome, Firefox, IE, Safari, Opera, etc.) and versions
- More features
  - Multi-language support, accessibility, etc.
- More users of software

Android Fragmentation (July 2013) – 11,868 distinct variants seen in past year



http://opensignal.com/reports/fragmentation-2013/

More Efficient Development

- Object oriented development, dynamic typing, higher levels of abstraction, frameworks, etc.

x86 Assembly Language

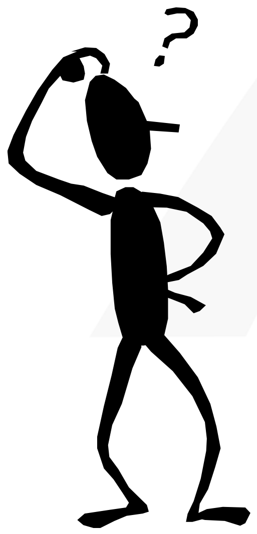
```
.model tiny
.code
org 100h

main  proc

      mov    ah,9
      mov    dx,offset hello_message
      int    21h

      retn


hello_message db 'Hello, world!$'

main  endp
end   main
```

Python

```
print "Hello, world!"
```

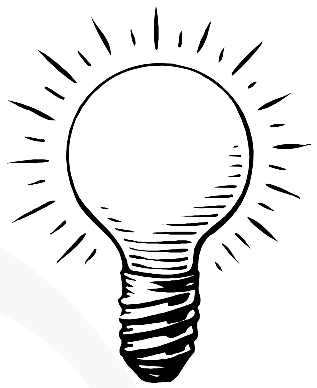
http://en.wikipedia.org/wiki/List_of_Hello_world_program_examples

What does all this **mean**?

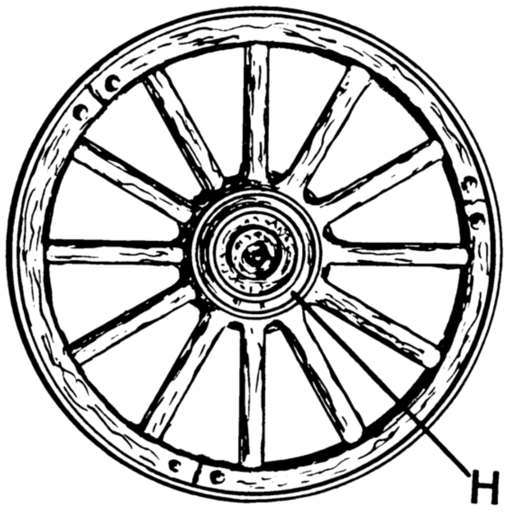


We need to figure out how to

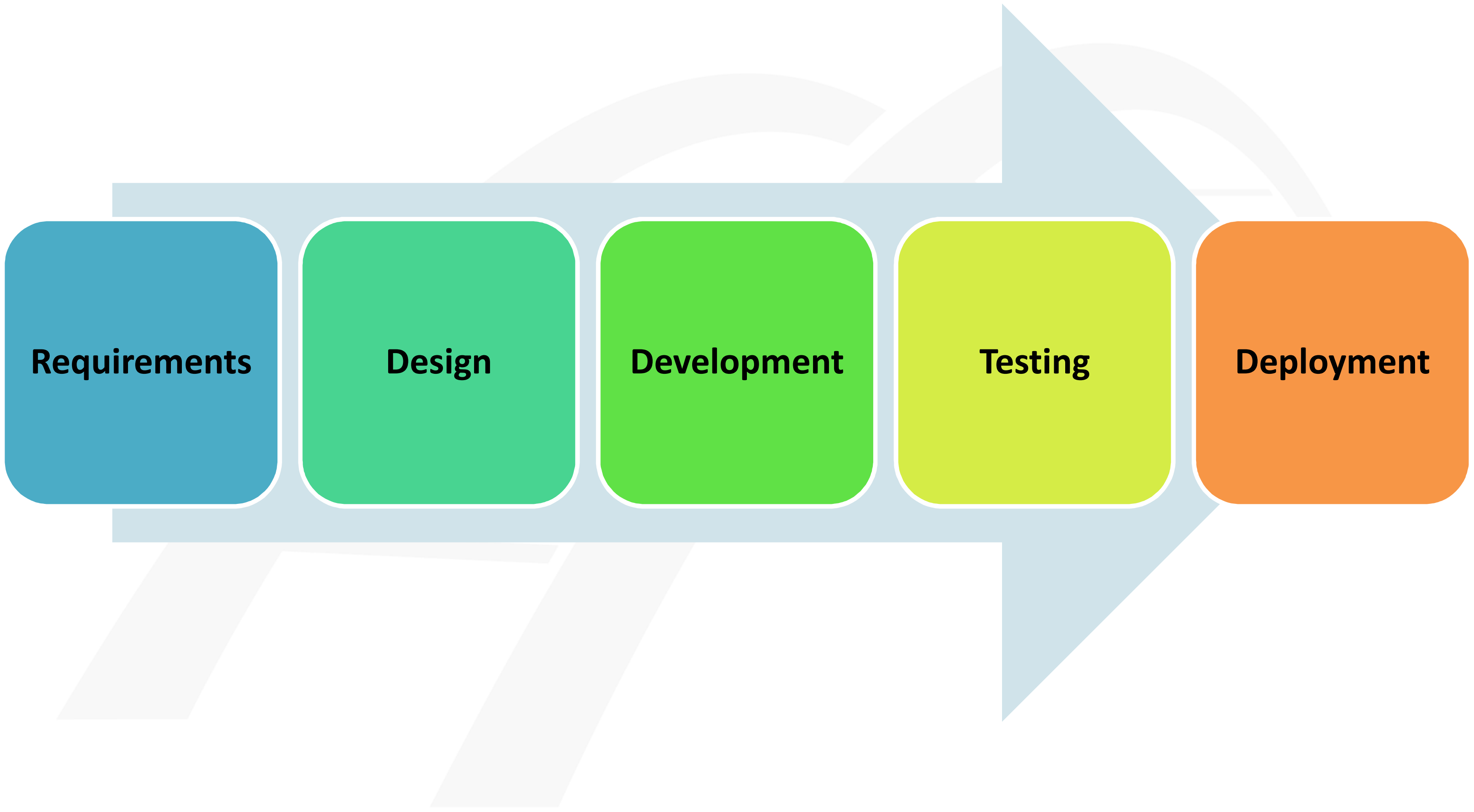
Do QA Better.



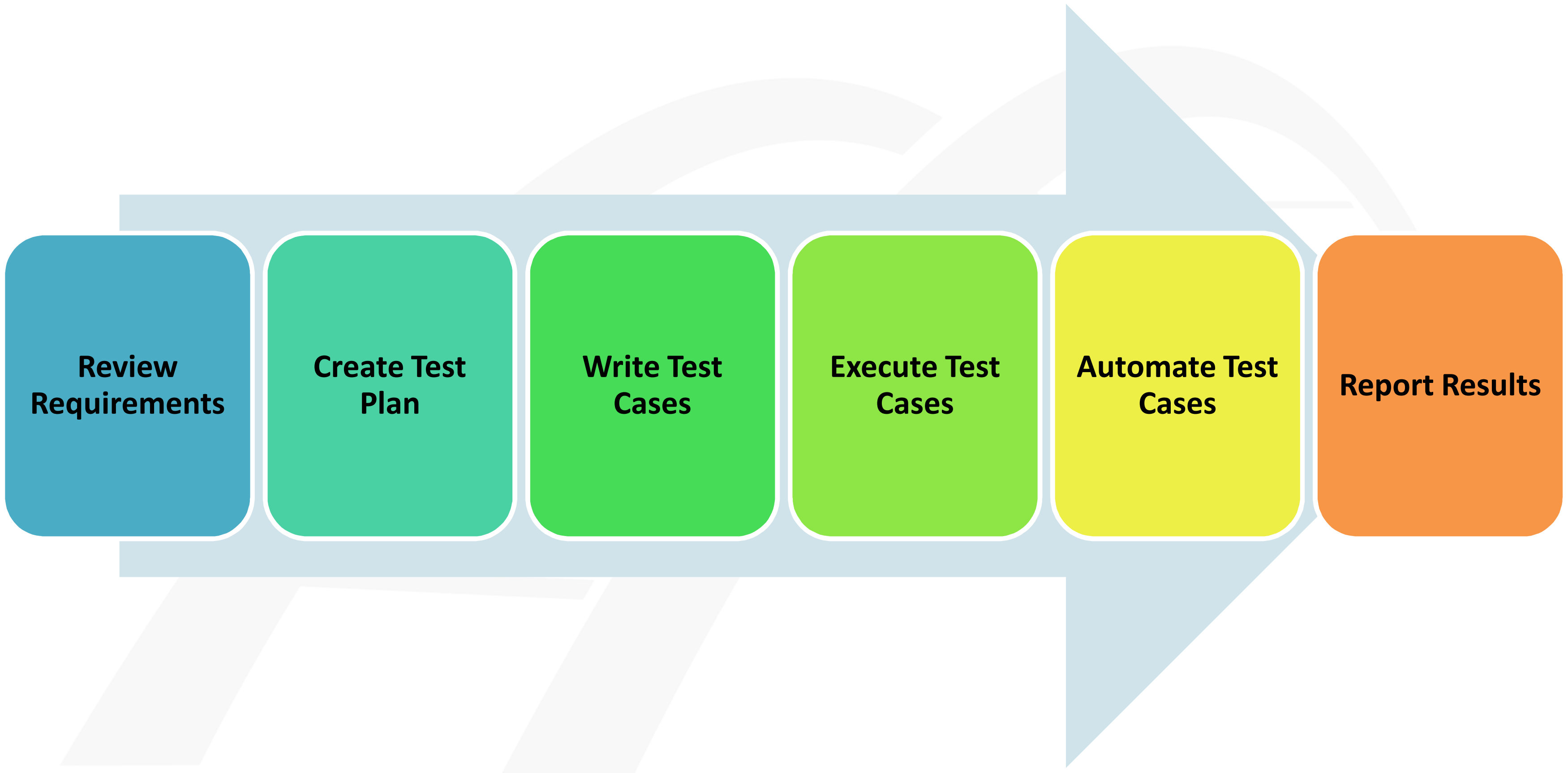
TRADITIONAL TESTING



Waterfall Project Phases



Waterfall Testing Steps

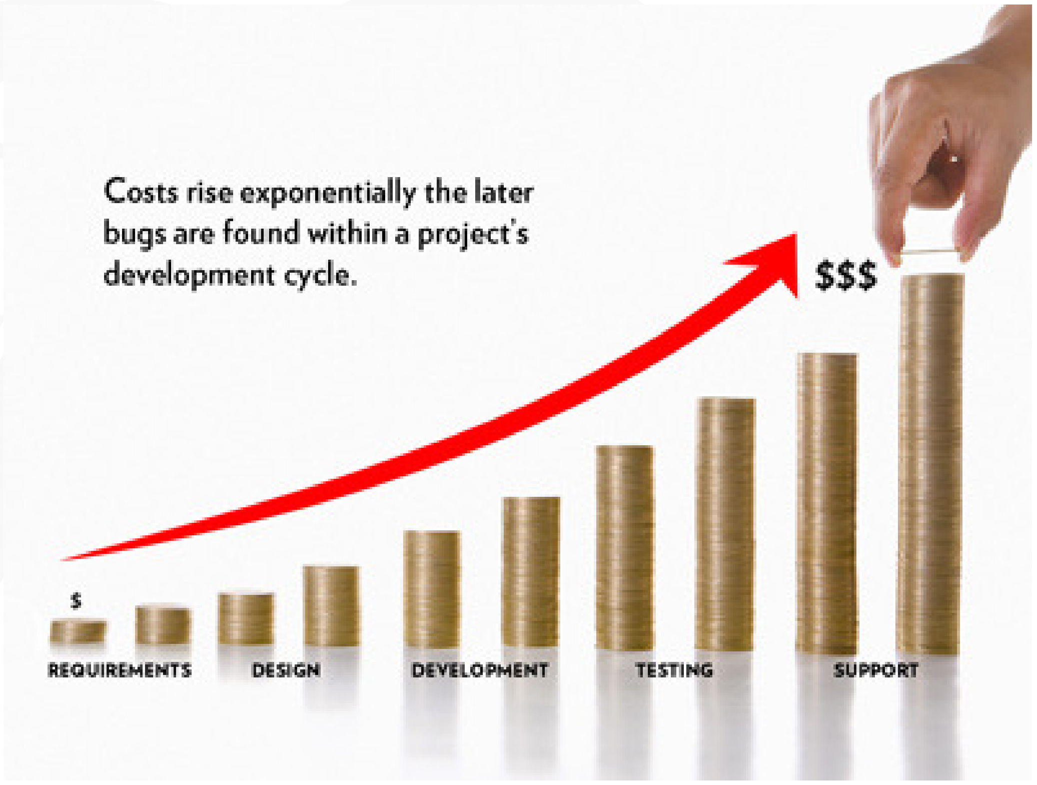


Some Challenges

- **Difficult to get requirements perfect up front**
  - Customers don't know what they want but are great critics
  - Difficult to perfectly envision what we're going to build up front
- **Many handoffs required**
  - Telephone game
  - Difficult to have shared understanding
- **Hard to know where you're really at**
  - How do you verify each stage is really ***DONE***?

Some More Challenges

- **Slow feedback**
  - Exponential cost of defects over time
- **Change happens**
  - External (market conditions, etc.)
  - Internal (team changes, portfolio changes, new knowledge, etc.)



Test Automation – Old School

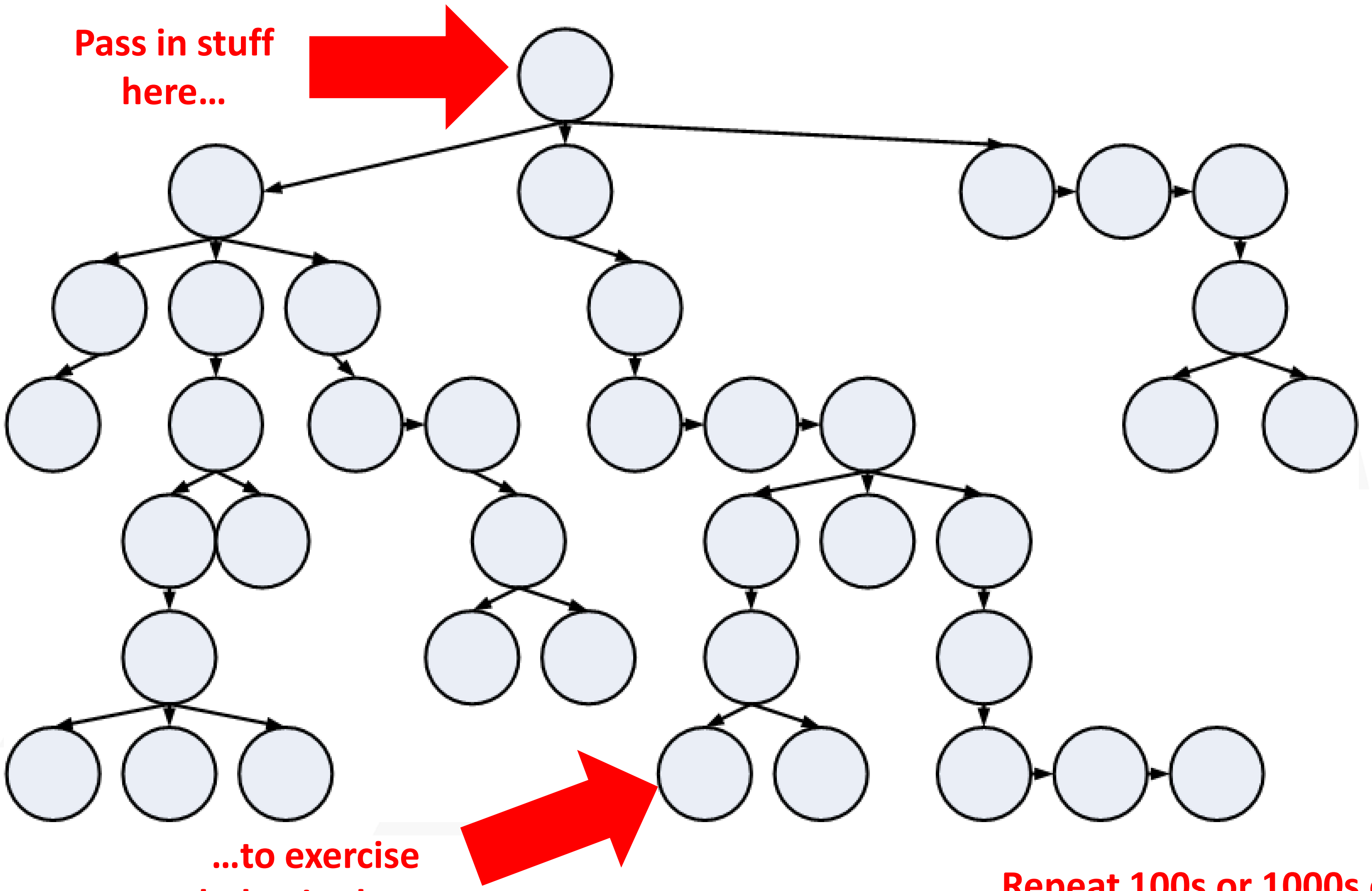
- **Focused on GUI automation**
  - Simulates a user interacting with the application
  - Slow to execute (but still faster than manual testing!)
- **Expensive to develop**
  - Expensive tools
  - Lots of labor hours
  - Specialist expertise required to do it well

Test Automation – Old School

- **Expensive to maintain**
  - Brittle GUI tests require constant maintenance
  - It's just kind of flakey
- **Doesn't scale well**
  - Maintenance costs eventually get out of control
  - Large test suites take too long to execute

Bridge360™

Simple Call Tree



Pass in stuff here…

…to exercise behavior here

Repeat 100s or 1000s or 10000s of times…

So what are we

DOING

about this?

Agile & Scrum Review

Manifesto for Agile Software Development (2001)

Individuals & interactions > processes and tools

Working software > comprehensive documentation

Customer collaboration > contract negotiation

Responding to change > following a plan

Success at agile methods requires a cultural change across the entire company. Cultural change is never easy!

http://agilemanifesto.org

Some Agile "Methodologies"

Scrum

Extreme Programming

Kanban

Crystal Clear

There are many ways to "be agile".
Scrum is the most popular "agile" framework.

Scrum in a Nutshell

- Rely on small (3-7 members), cross-functional, self-organizing teams
- Break project into small pieces of work ("user stories") that each take a few days to implement
- Deliver 100% complete user stories in 1-4 week "sprints"
- Do the highest value work first

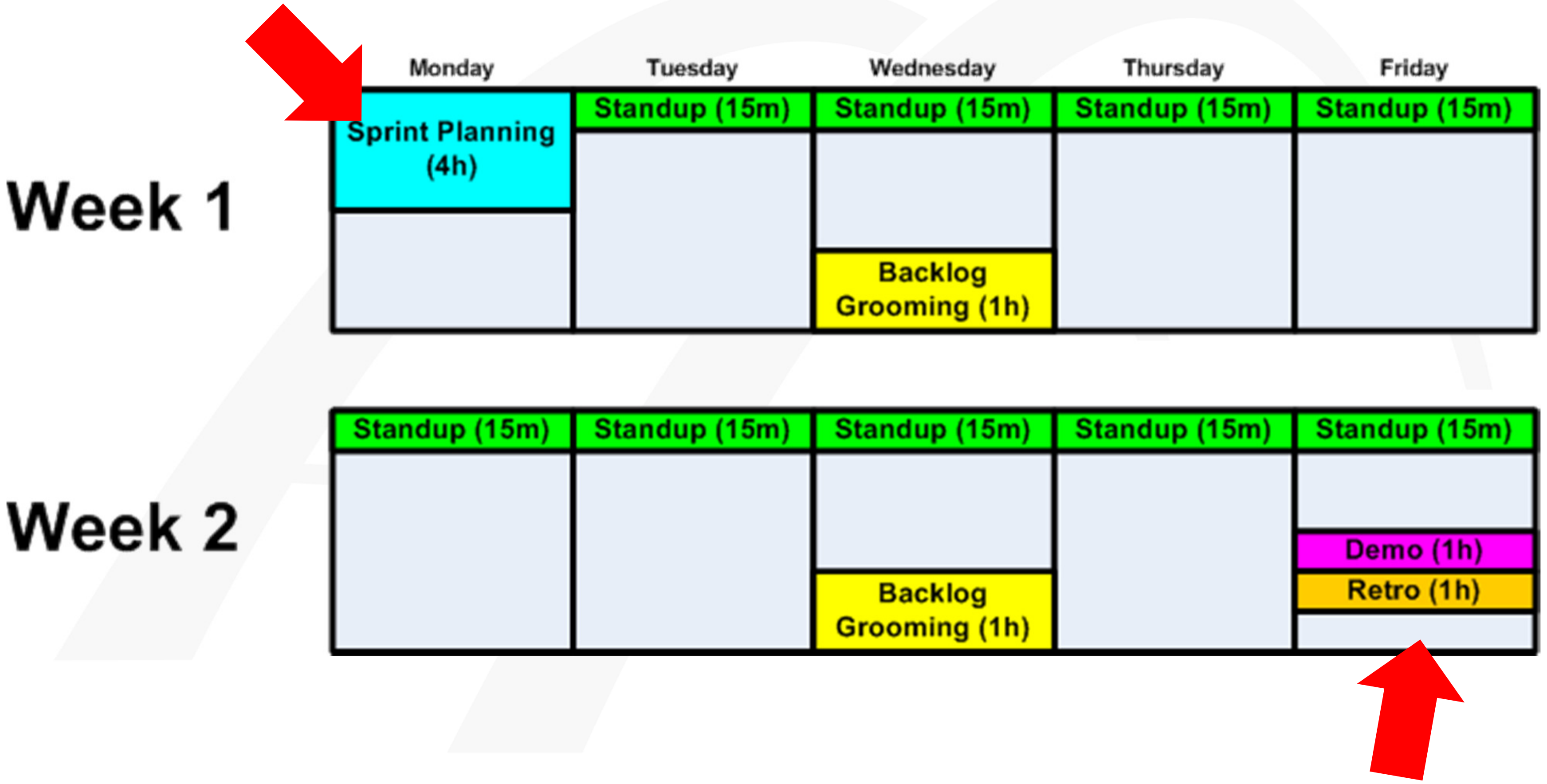
Scrum in a Nutshell (2)

- No changes while is a sprint
  - Cannot add / remove features to current sprint
  - Team does not change during sprint
- Change welcome between sprints!
- Hold a retrospective after each sprint to identify opportunities to improve

“Inspect and adapt”

Scrum “Heartbeat” Example

Start Development Here



	Monday	Tuesday	Wednesday	Thursday	Friday
Week 1	Sprint Planning (4h)	Standup (15m)	Standup (15m) Backlog Grooming (1h)	Standup (15m)	Standup (15m)
Week 2	Standup (15m)	Standup (15m)	Standup (15m) Backlog Grooming (1h)	Standup (15m)	Standup (15m) Demo (1h) Retro (1h)

Code finished, application fully tested by here

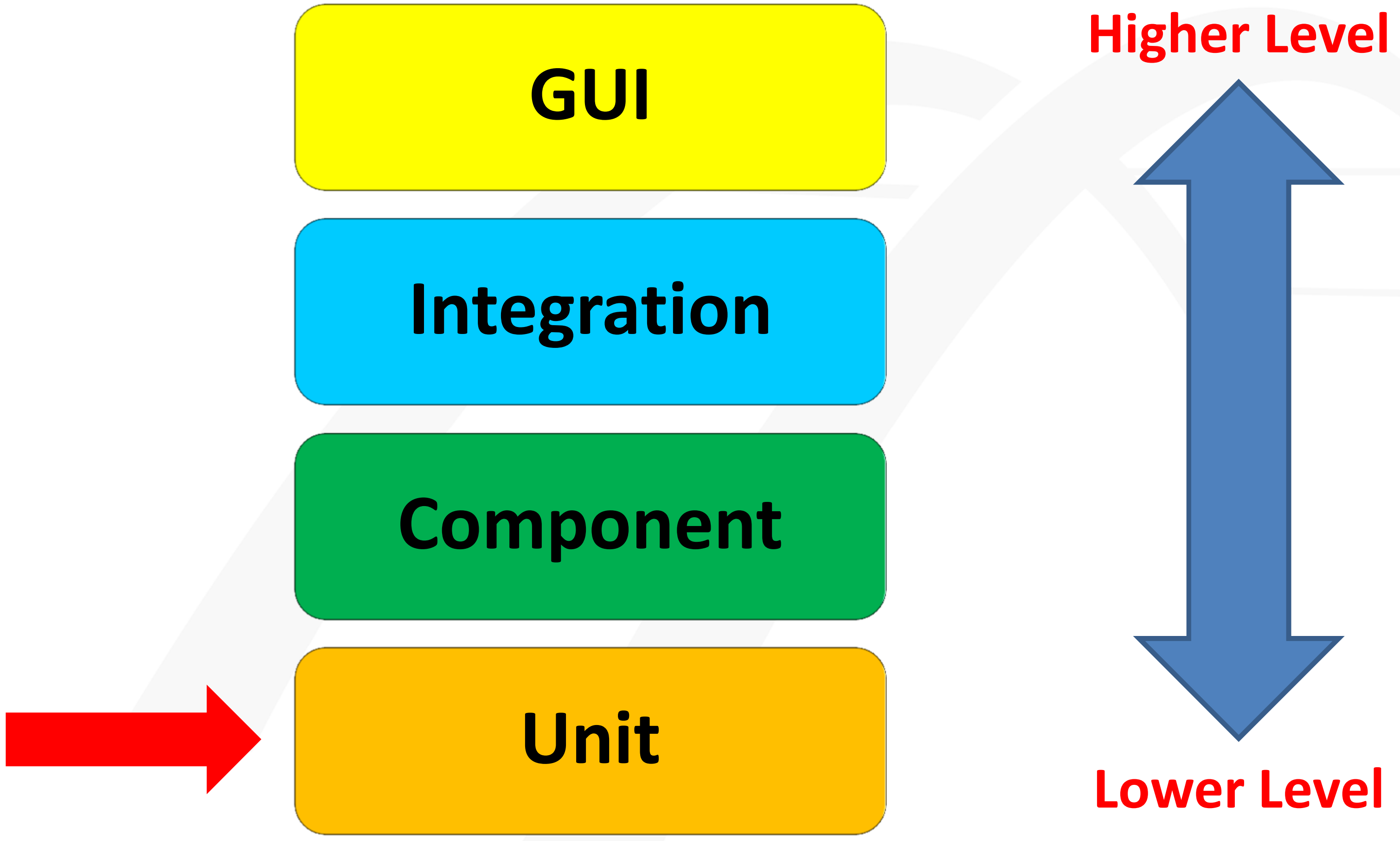
Wait, did you just say we have to test the whole application

every two weeks?!??

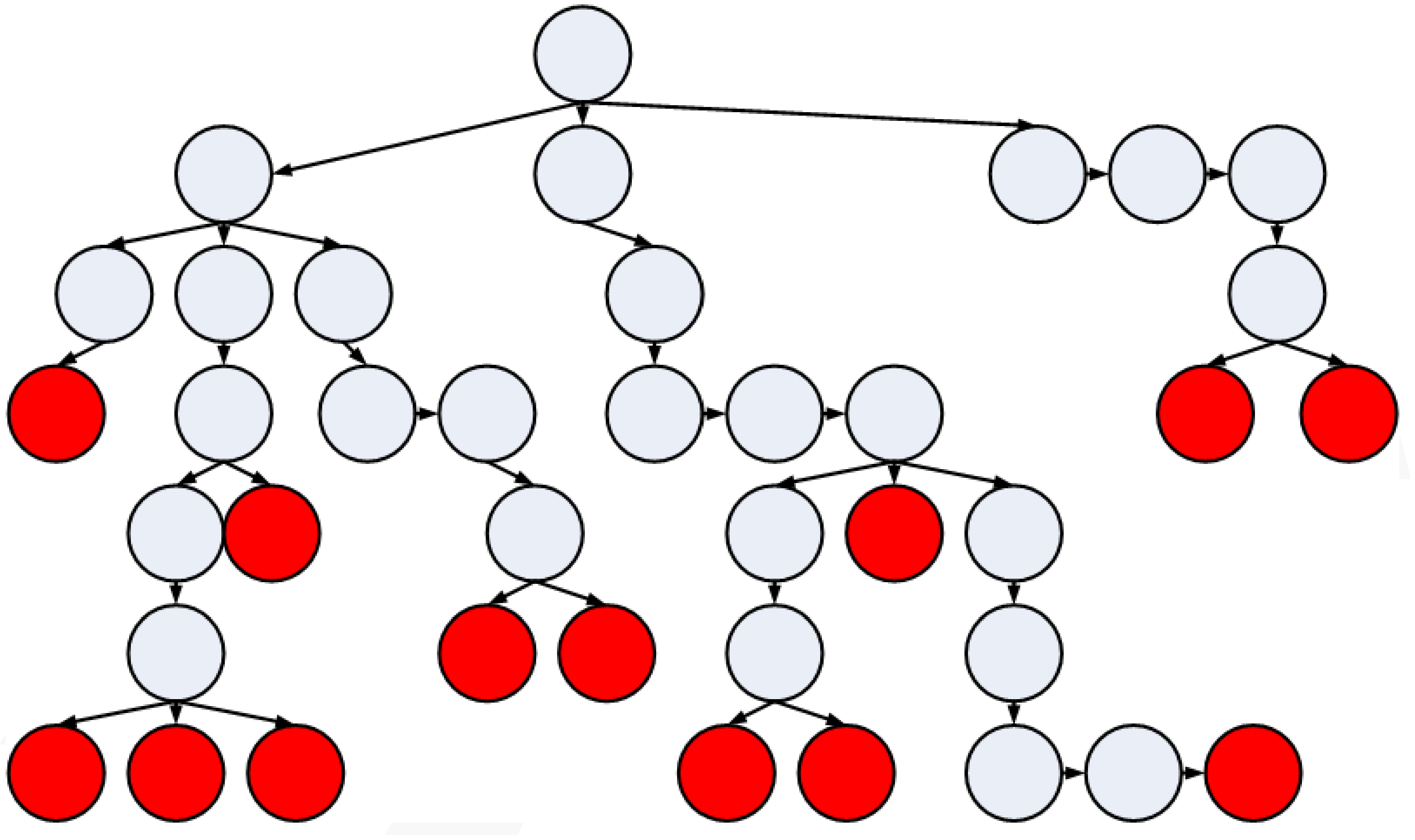
So, um, how are we supposed to do that…?

How to Do Test Automation Right

Types of Test Automation

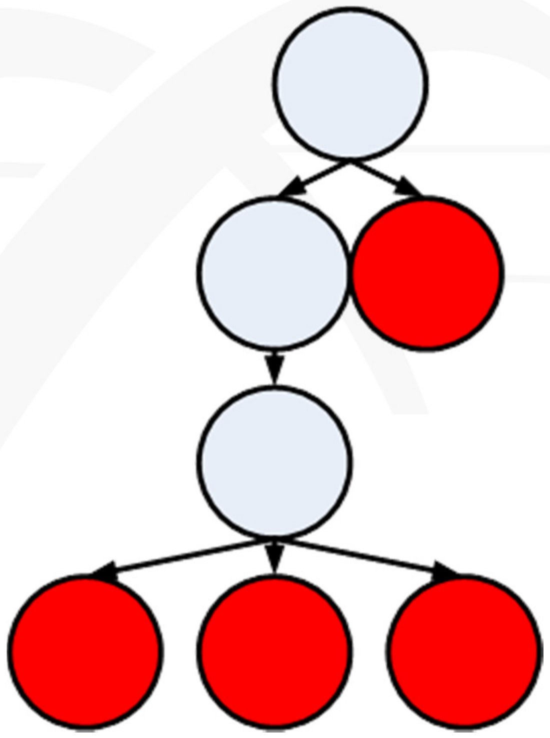


Test Leaf Nodes with Unit Tests

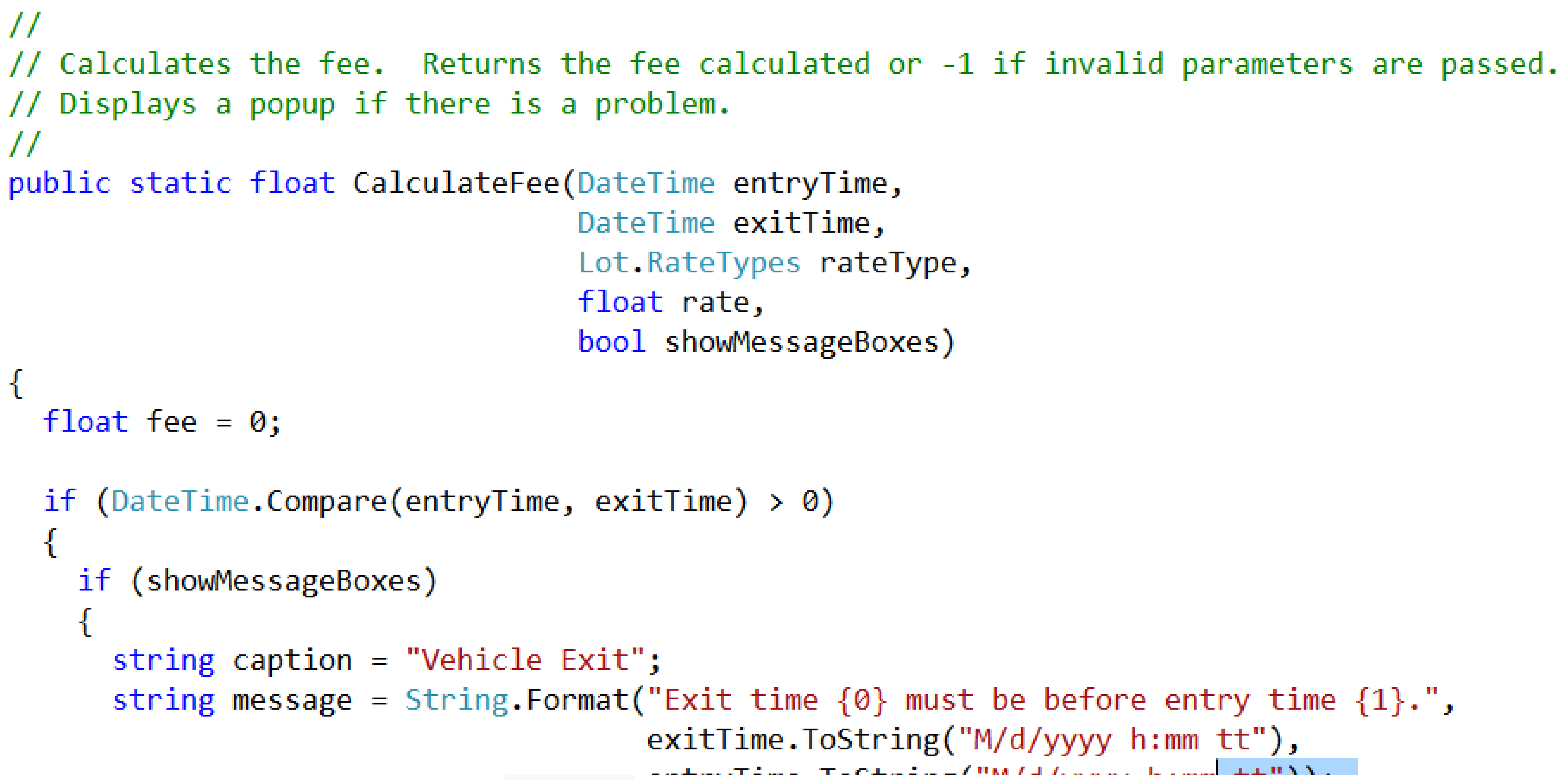


Benefits of Unit Tests

- Simple
- Few dependencies
- Stable
- Repeatable



Example Code: Parking Fee Calculator



```
//
// Calculates the fee.  Returns the fee calculated or -1 if invalid parameters are passed.
// Displays a popup if there is a problem.
//
public static float CalculateFee(DateTime entryTime,
                                 DateTime exitTime,
                                 Lot.RateTypes rateType,
                                 float rate,
                                 bool showMessageBoxes)

{
  float fee = 0;

  if (DateTime.Compare(entryTime, exitTime) > 0)
  {
    if (showMessageBoxes)
    {
      string caption = "Vehicle Exit";
      string message = String.Format("Exit time {0} must be before entry time {1}.",
                                     exitTime.ToString("M/d/yyyy h:mm tt"),
```

Example Unit Tests

```
[TestMethod]
public void ShouldRejectExitTimeBeforeEntryTime()
{
  // arrange
  DateTime      entryTime = new DateTime(2014, 1, 1, 10, 0, 0);
  DateTime      exitTime  = new DateTime(2014, 1, 1,  9, 59, 0);
  Lot.RateTypes rateType  = Lot.RateTypes.Daily;
  float         rate      = 5;

  // act
  float fee = MainForm.CalculateFee(entryTime, exitTime, rateType, rate, false);

  // assert
  Assert.AreEqual(fee, -1f, 0, "Should reject records with exit time before entry time.");
}
```

Example Unit Tests

```
[TestMethod]
public void TestFeeForExitTimeSameAsEntryTime()
{
  // arrange
  DateTime entryTime = new DateTime(2014, 1, 1, 10, 0, 0);
  DateTime exitTime = new DateTime(2014, 1, 1, 10, 0, 0);
  Lot.RateTypes rateType = Lot.RateTypes.Hourly;
  float rate = 10;

  // act
  float fee = MainForm.CalculateFee(entryTime, exitTime, rateType, rate, false);

  // assert
  Assert.AreEqual(fee, 0f, 0, "Fee should be zero for records with entry time the same as exit time.");
}
```

Improvement: Data-Driven Tests

Test Name	Entry Date/Time					Exit Date/Time					Rate Type	Rate	Expected
### Simple happy paths													
Test daily rate for 5 days	2014	1	1	10	0	2014	1	6	10	0	daily	10	50
Test hourly rate for 5 hours	2014	1	1	10	0	2014	1	1	15	0	hourly	2	10
### Edge cases													
Fee should be zero for records with entry time the same as exit time.	2014	1	1	10	0	2014	1	1	10	0	hourly	2	0
Test hourly rate across hour boundary	2014	1	1	10	30	2014	1	1	11	15	hourly	2	2
Test hourly rate across day boundary	2014	1	1	23	30	2014	1	2	0	15	hourly	2	2
Test hourly rate across month boundary	2014	1	31	23	30	2014	2	1	0	15	hourly	2	2
Test hourly rate across year boundary	2014	12	31	23	30	2015	1	1	0	15	hourly	2	2
Test daily rate across day boundary	2014	1	1	23	30	2014	1	2	0	15	daily	10	10
Test daily rate across month boundary	2014	1	31	23	30	2014	2	1	0	15	daily	10	10
Test daily rate across year boundary	2014	12	31	23	30	2015	1	1	0	15	daily	10	10
Verify 2000 is not a leap year	2000	2	1	10	0	2000	3	1	9	30	daily	10	280
#Verify 2004 is a leap year	2004	2	1	10	0	2004	3	1	9	30	daily	10	290
### Invalid scenario tests													
Should reject records with exit time before entry time.	2014	1	1	10	0	2014	1	1	9	59	daily	5	-1

Isolating Dependencies

- **Stubs**
  - Fake or dummy implementations that have defined behaviors
  - Code under test relies on stubs for dependencies
- **Dependency Injection**
  - Getting code to use specific dependencies at runtime
  - Many techniques
    - E.g. pass in all dependencies as parameters

Example: Removing Dependencies

```
Boolean HasWorkDayStarted(WorkStartTime):
    if (WorkStartTime > System.GetCurrentTime()):
        return True
    else
        return False
```

So, um, how do I test this at different times of the day?

Example: Removing Dependencies

```
Boolean HasWorkDayStarted(WorkStartTime, CurrentTime):
        if (WorkStartTime > CurrentTime):
                return True
        else
                return False
```

Dependency Injection

No external dependencies =

Super easy to test =

WIN

Example: Using a Stub

```
Boolean HasWorkDayStarted(WorkStartTime):
        if (WorkStartTime > ResourceManager.Get("System").GetCurrentTime()):
                return True
        else
                return False
```

Replace the call to the built-in System library with a stub we control.

Example: Using a Stub

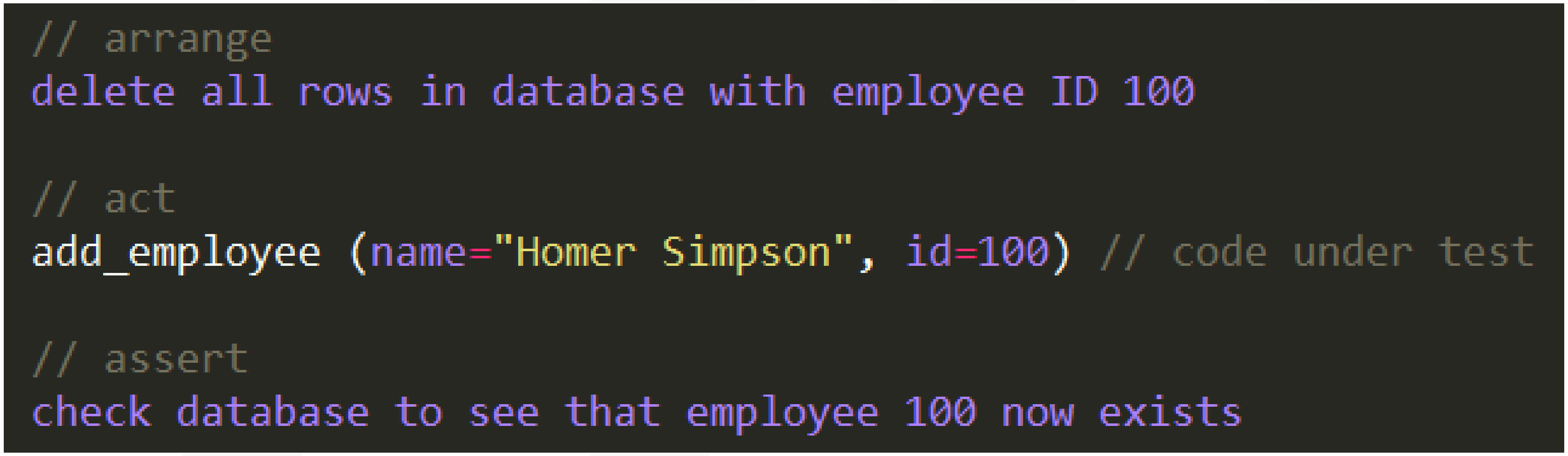
```
Void Test_HasWorkDayStarted_BeforeWorkDay_NotStarted():
        // arrange – set System.GetCurrentTime() to always return 6 a.m.
        ResourceManager.SetReturnValue("System.GetCurrentTime()", 6.00)

        // act – call our code being tested
        result = HasWorkDayStarted(8.00)

        // assert – verify we got the expected result
        assertSame(result,
                   false,
                   "Current time before start time does not return false")
```

Test Verification

- Method #1: Verify state
  - Preferred method to verify results but not always easy
  - General flow:
    - System begins in a known state
    - Run the code under test that changes the system state
    - Verify system state has changed appropriately after code under test completes
  - Examples of state changes:
    - Global variable changed
    - Row added or deleted in a database
    - Function returned a value or result code

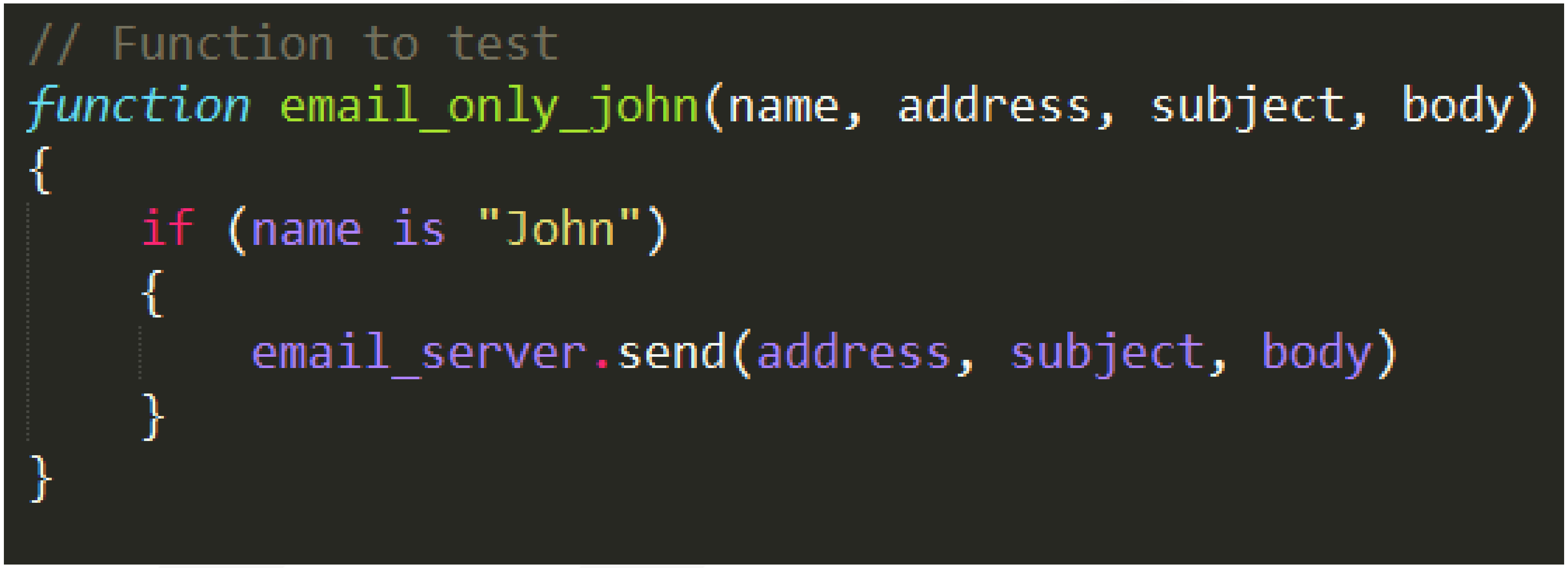
Test Verification

Simple example:



```
// arrange
delete all rows in database with employee ID 100

// act
add_employee (name="Homer Simpson", id=100) // code under test

// assert
check database to see that employee 100 now exists
```

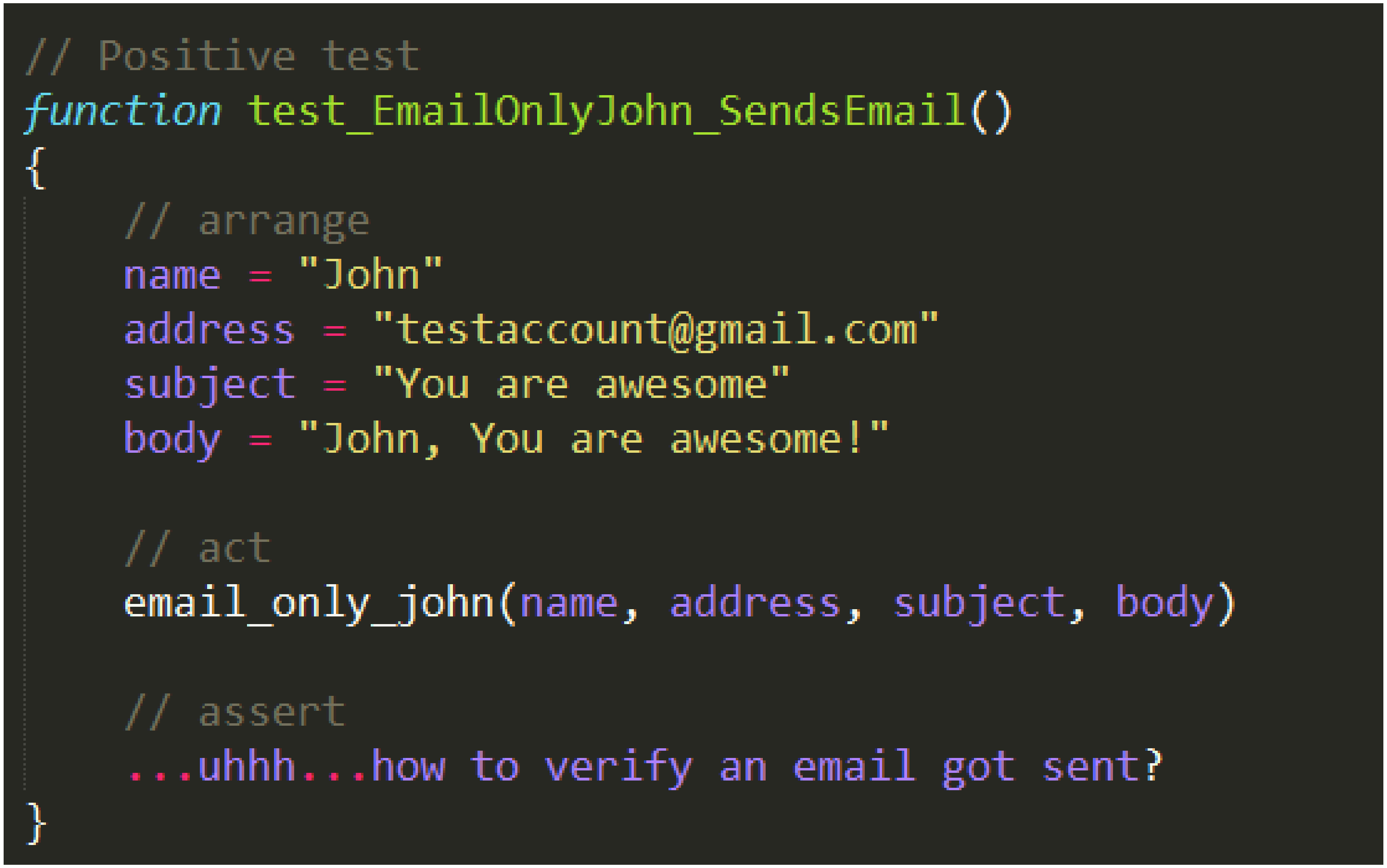
How to Verify Results Here?



```
// Function to test
function email_only_john(name, address, subject, body)
{
    if (name is "John")
    {
        email_server.send(address, subject, body)
    }
}
```

Wow this John guy is super special…!

How to Verify Results Here?


```
// Positive test
function test_EmailOnlyJohn_SendsEmail()
{
    // arrange
    name = "John"
    address = "testaccount@gmail.com"
    subject = "You are awesome"
    body = "John, You are awesome!"

    // act
    email_only_john(name, address, subject, body)

    // assert
    ...uhhh...how to verify an email got sent?
}
```

How to Verify An Email Was Sent?

- Obvious approach
  - Set up test email server
  - Write a method that will log into the email server and check if we got an email sent

How to Verify An Email Was Sent?

- Challenges
  - Slooooooow
  - What if email is delayed due to the network?
  - Now we have to maintain a mail server or service just to do testing (seriously?!?!)
  - What if multiple tests are running simultaneously and generating emails to the same inbox?

Wow this is making my head hurt just thinking about it…

Test Verification

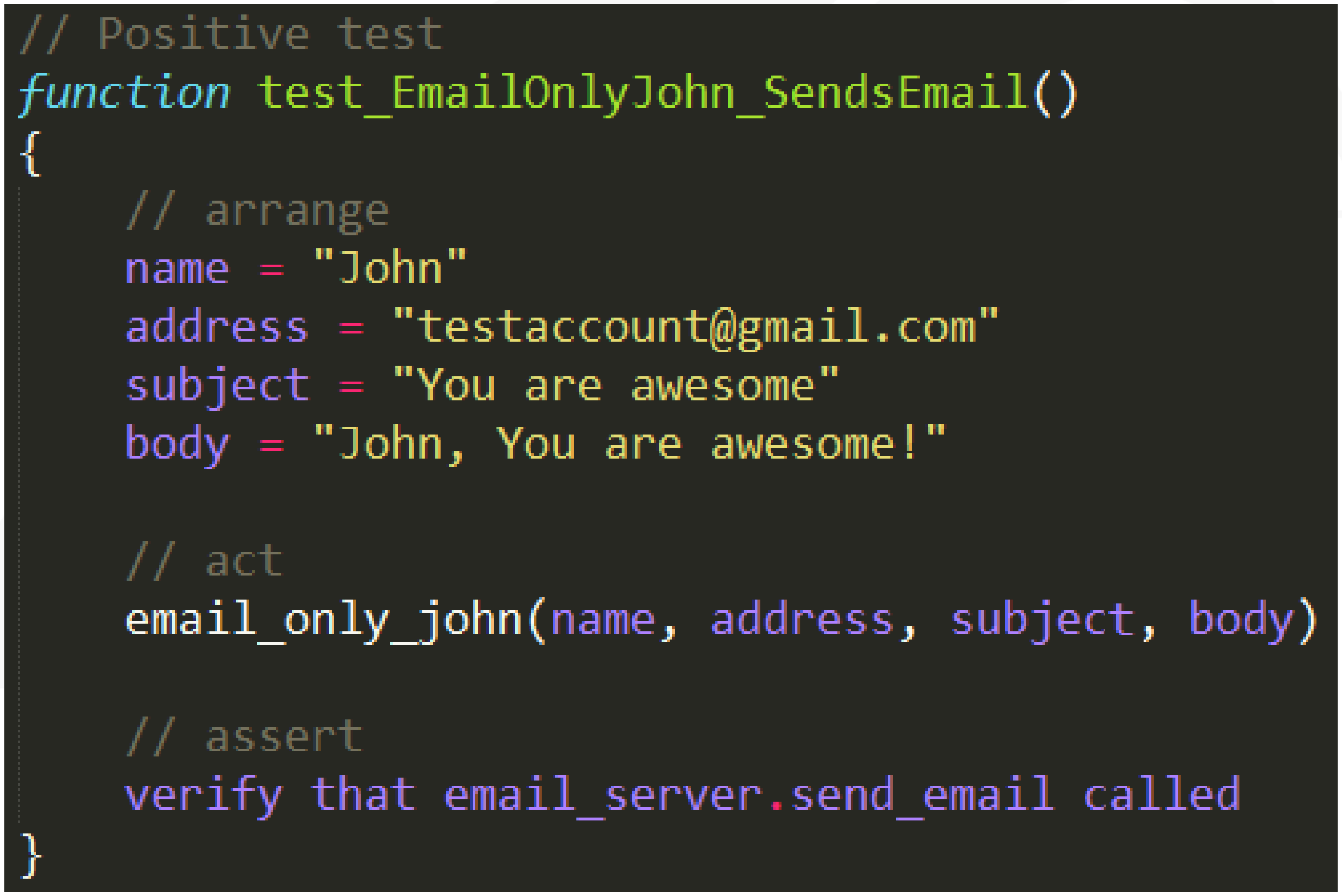
- Method #2: Verify interactions
  - Generally used when it is hard or inconvenient to verify state directly
  - General flow:
    - System begins in a known state
    - Run the code under test
    - Verify the code under test did specific things (e.g. called methods on other objects)

Test Verification

- Method #2: Verify interactions (continued)
  - Examples of interactions to check:
    - Database (or data access object) delete row method called
    - Email sent() method called
  - Tradeoffs
    - Tests become harder to understand
    - Enables overall easier testing in some cases
    - Can reduce setup/teardown

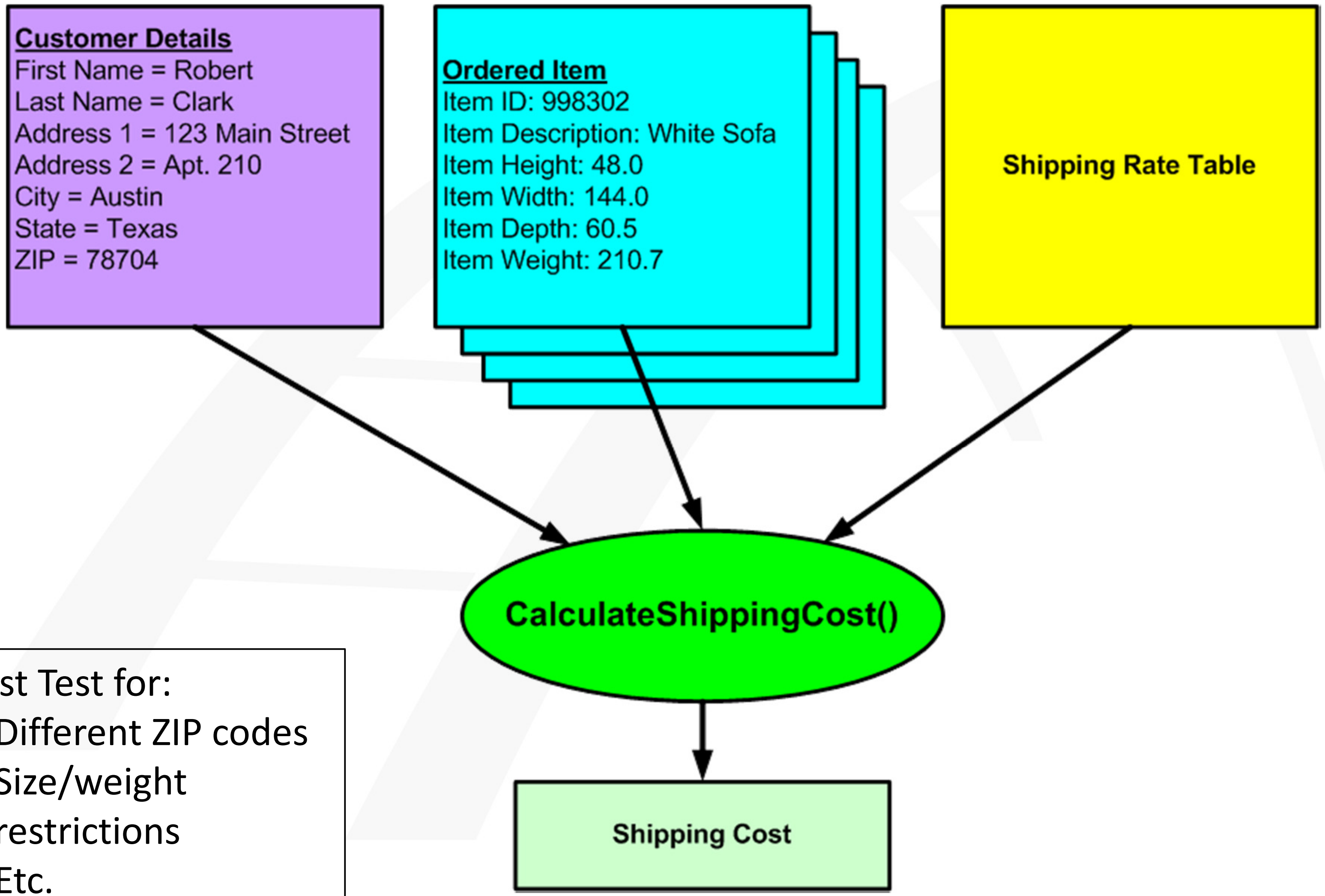
Test Verification

- Method #2: Verify interactions Example



```
// Positive test
function test_EmailOnlyJohn_SendsEmail()
{
    // arrange
    name = "John"
    address = "testaccount@gmail.com"
    subject = "You are awesome"
    body = "John, You are awesome!"

    // act
    email_only_john(name, address, subject, body)

    // assert
    verify that email_server.send_email called
}
```

Test Data



Must Test for:
- Different ZIP codes
- Size/weight restrictions
- Etc.

Test Data

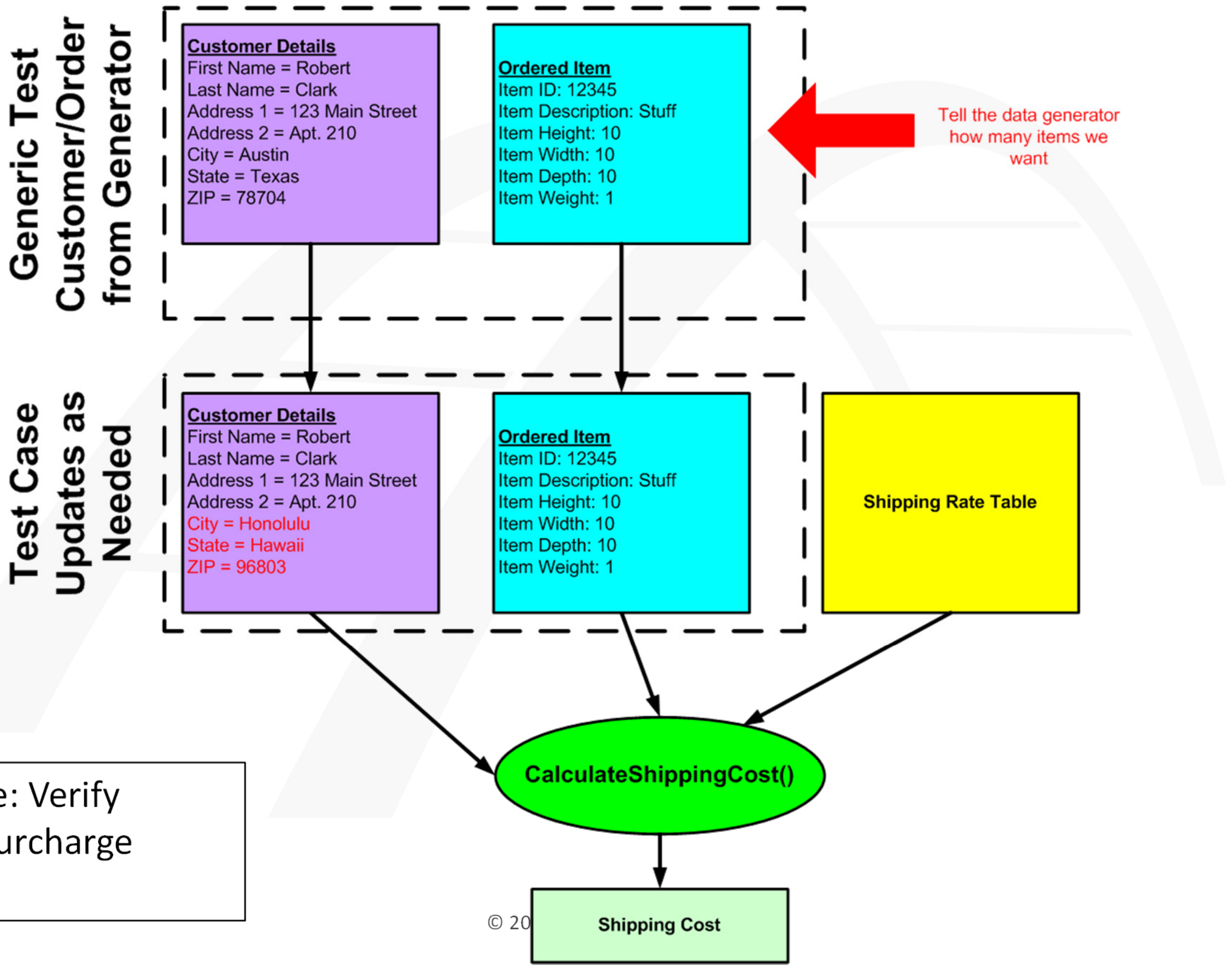
- Old school
  - Store test data in database, files, etc.
  - Maintain shared, reusable "test fixture" data structures in tests
- Issues
  - Gets more complex over time
  - Expensive to maintain (so many edge cases!)
  - Brittle

Wow it sounds like we're in the test data maintenance business…

Test Data

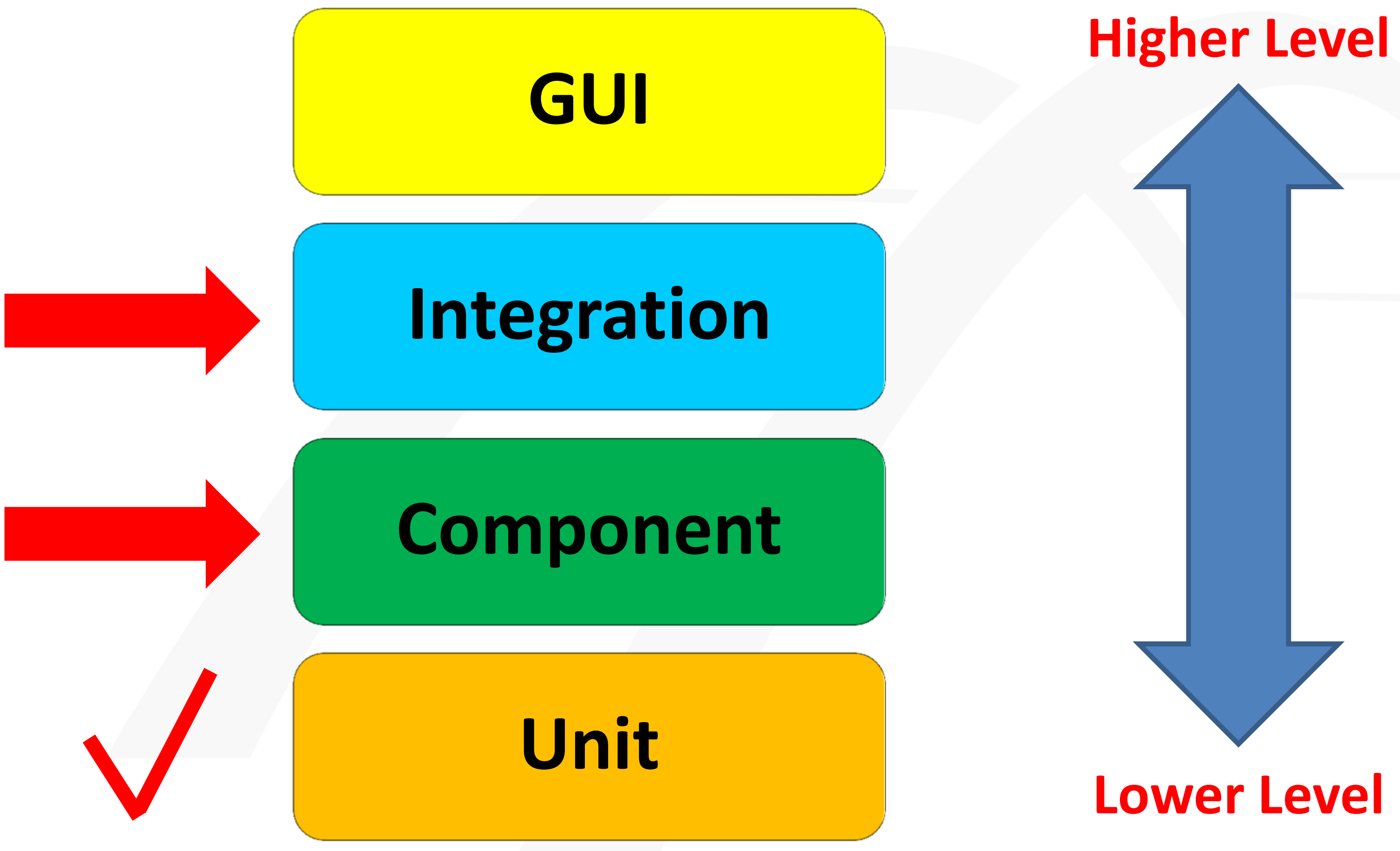
- New school
  - Generate test data on the fly
    - Generate generic test data
    - Tweak the generic data as needed for test case
  - Higher initial cost
    - Harder to understand at first
    - Must build data generators
    - Tools exist to help
  - Much cheaper to maintain
    - Data structures changed? Just tweak the generator

Test Data Generator

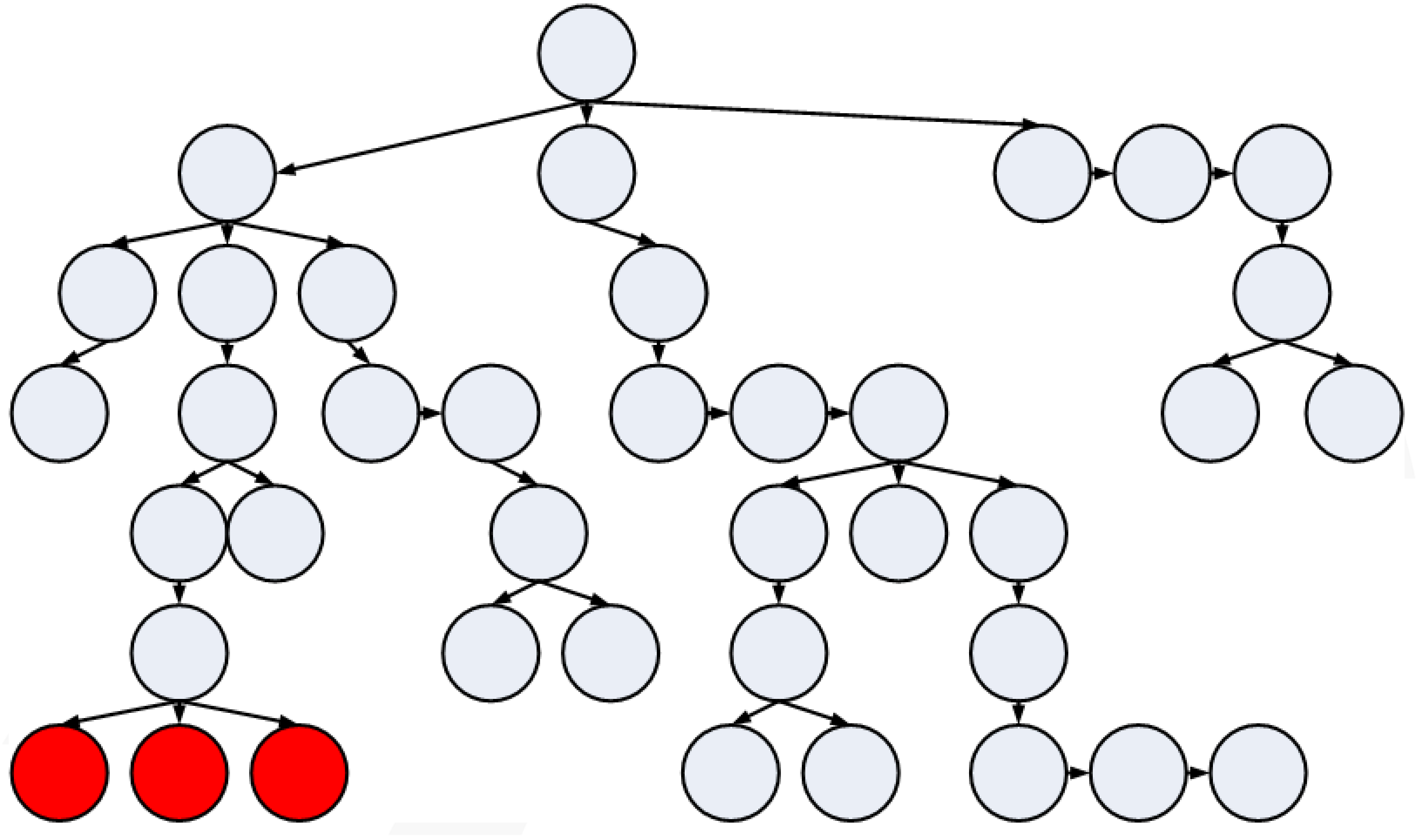


Test Case: Verify Hawaii surcharge applied.

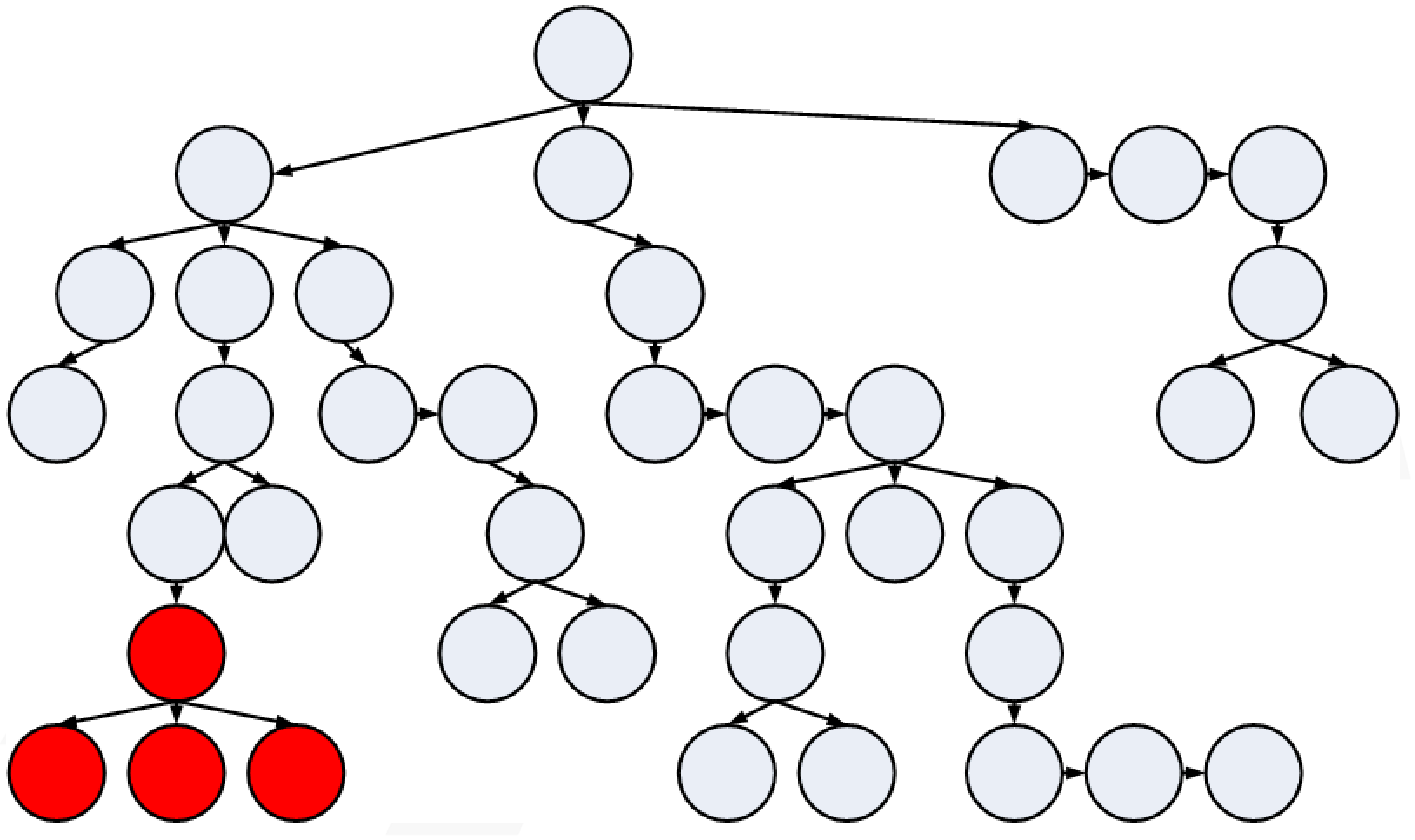
Types of Test Automation



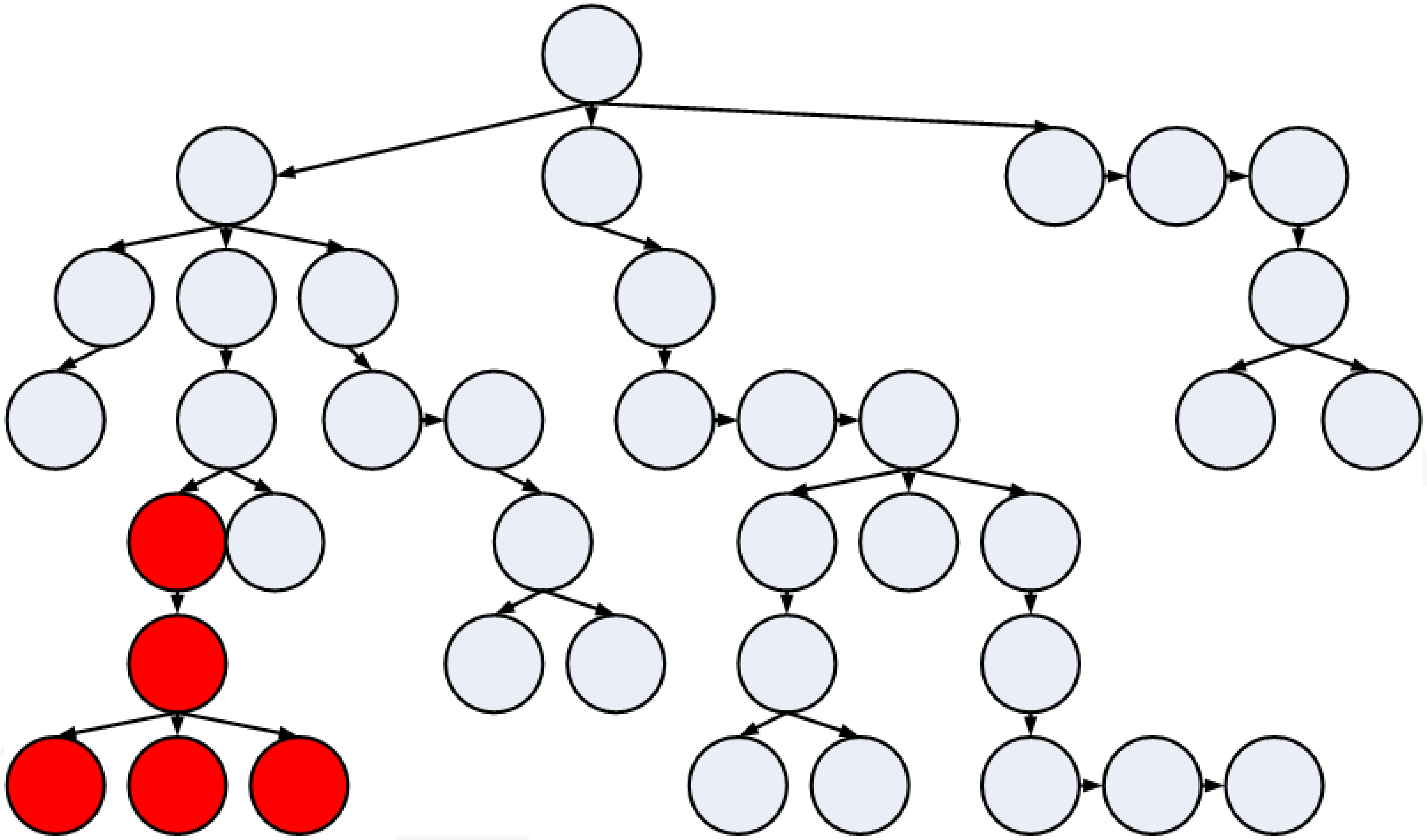
Movin' on Up



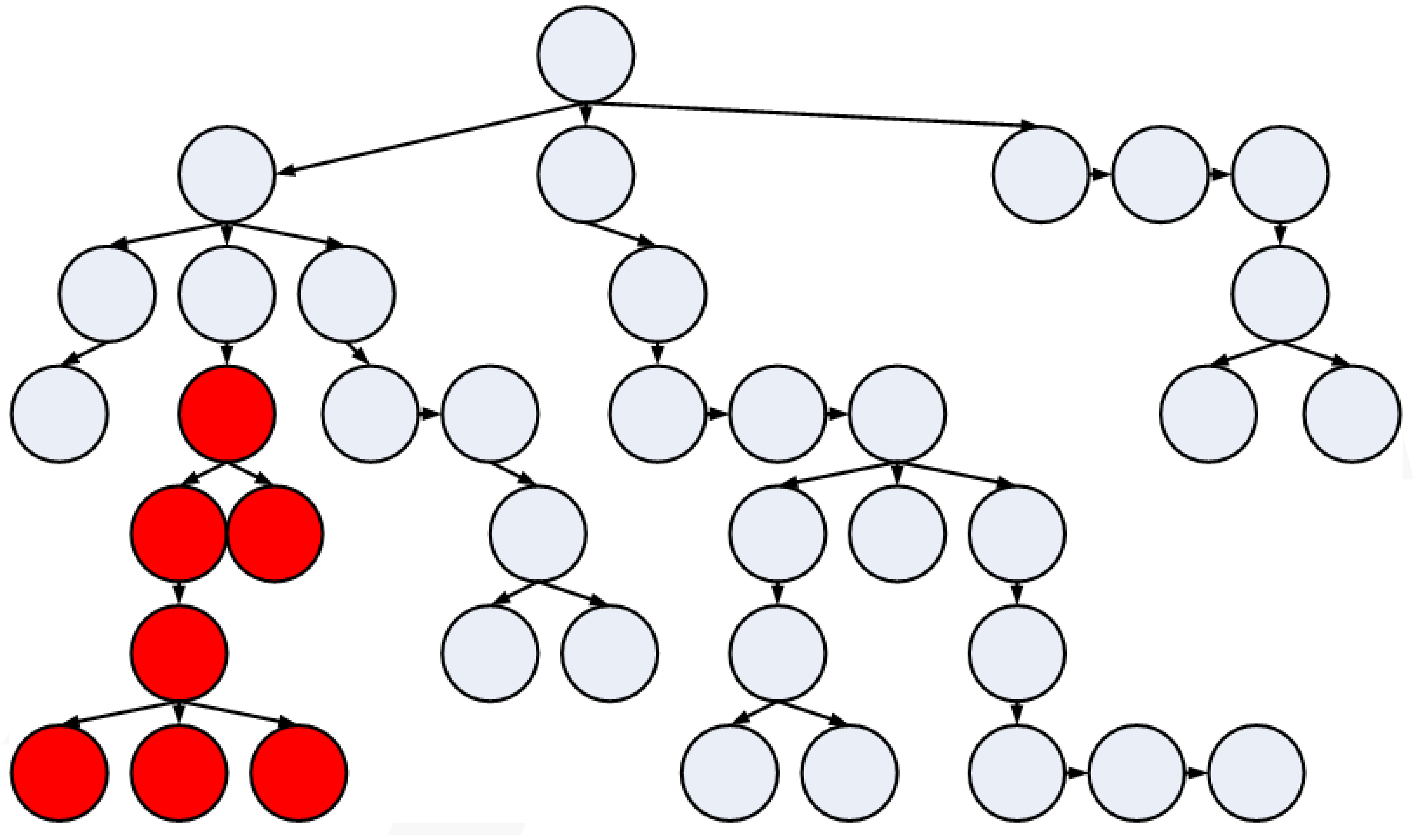
Movin’ on Up



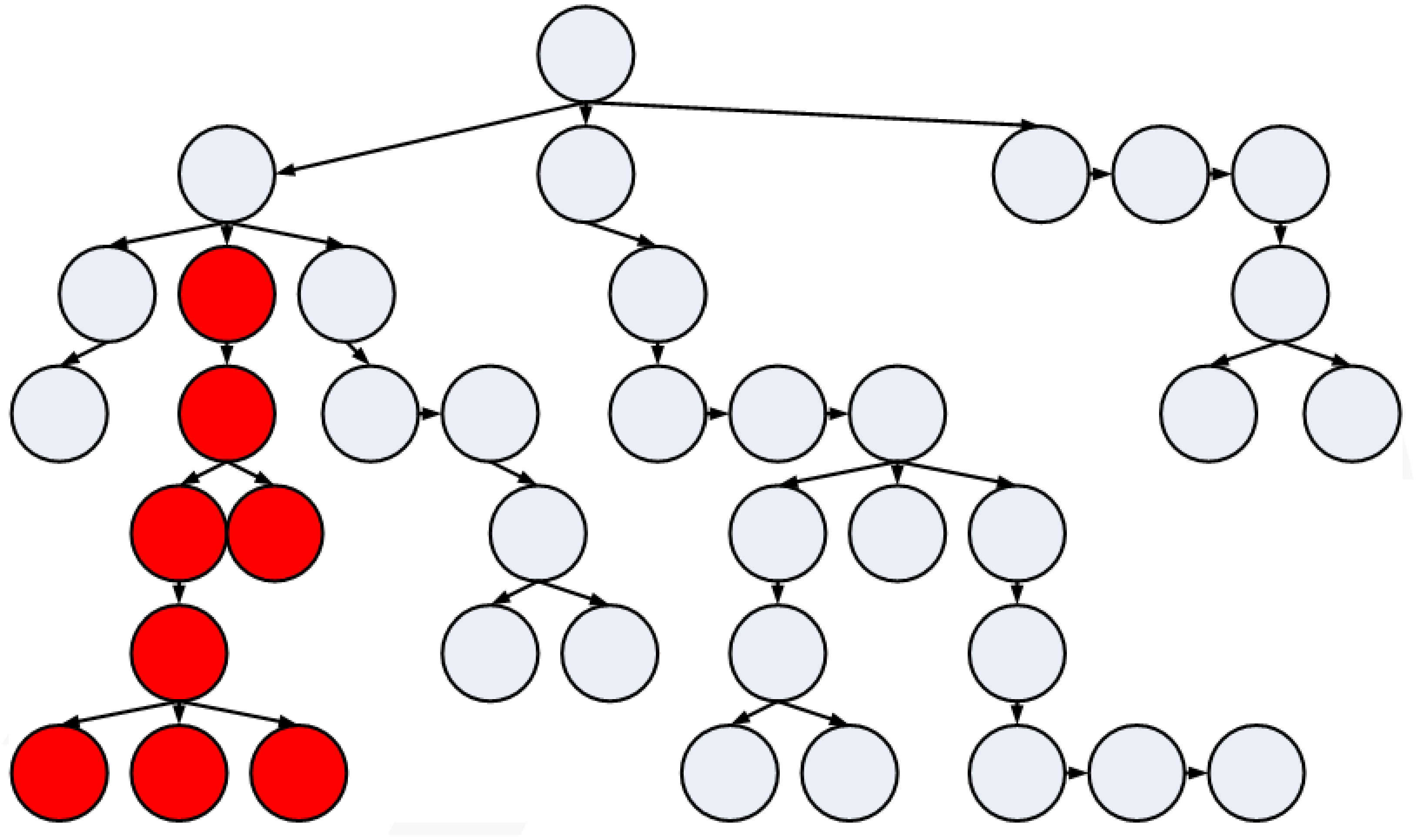
Movin' on Up…?



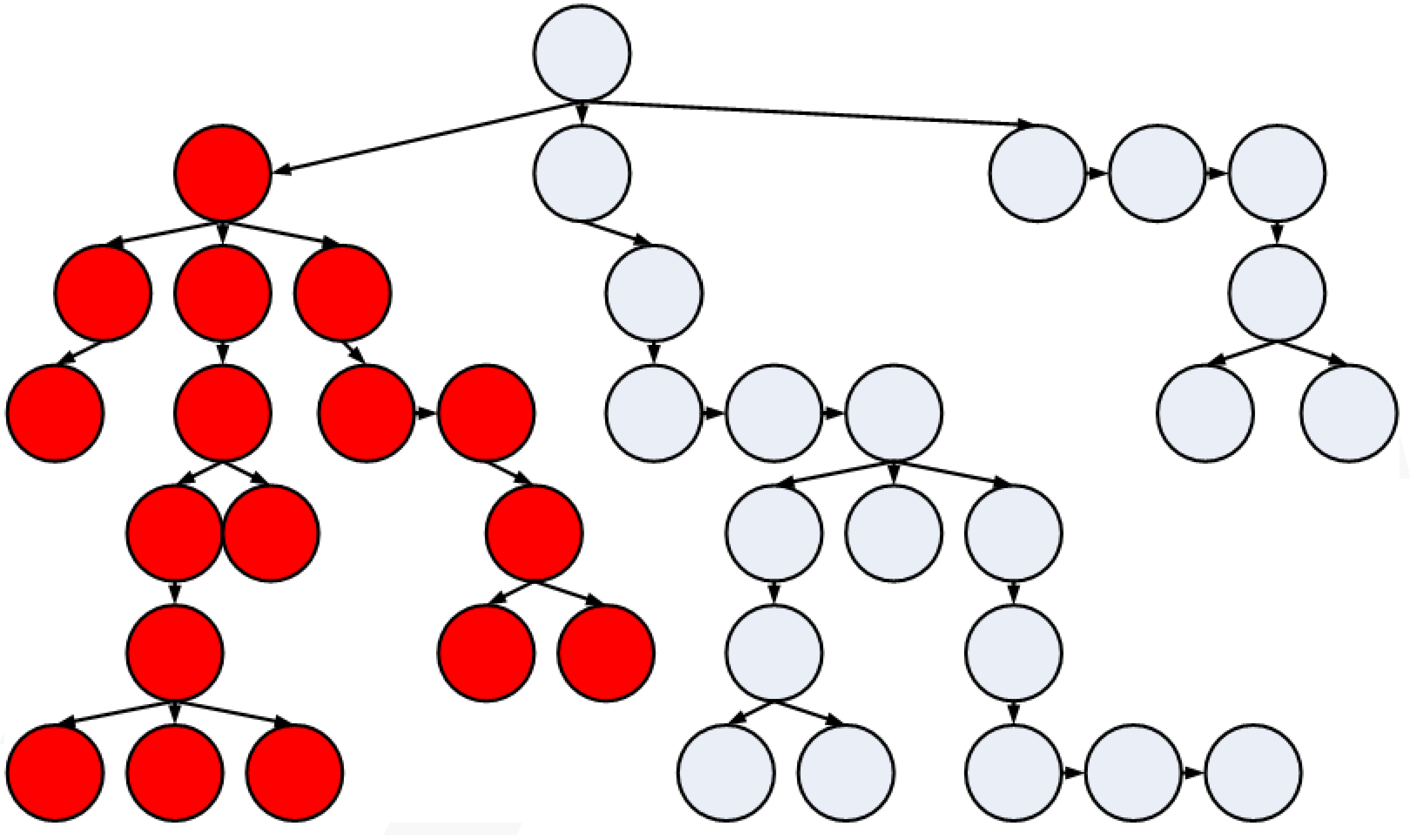
Movin' on Up…???



Movin' on Up…still?



Movin’ on Up…it this thing on?!?



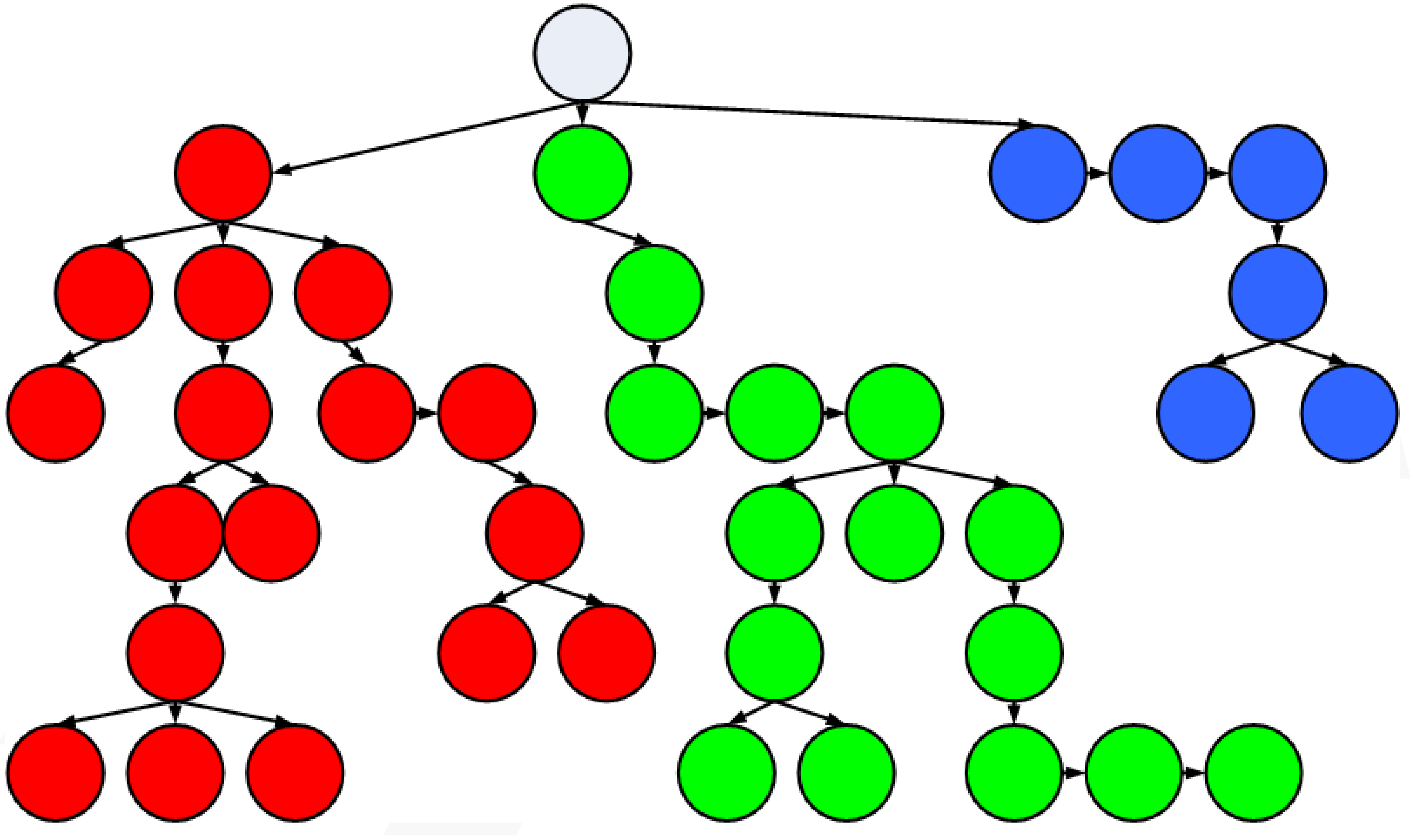
Bottom-Up to Top-Down

- At some point we stop testing bottom-up unit testing and switch to top-down component testing
  - When? Well it depends…
  - The better your code is structured the further up the tree you can take unit testing techniques
  - Usually difficult to do more than 1 or 2 levels up
  - Still can test at multiple levels

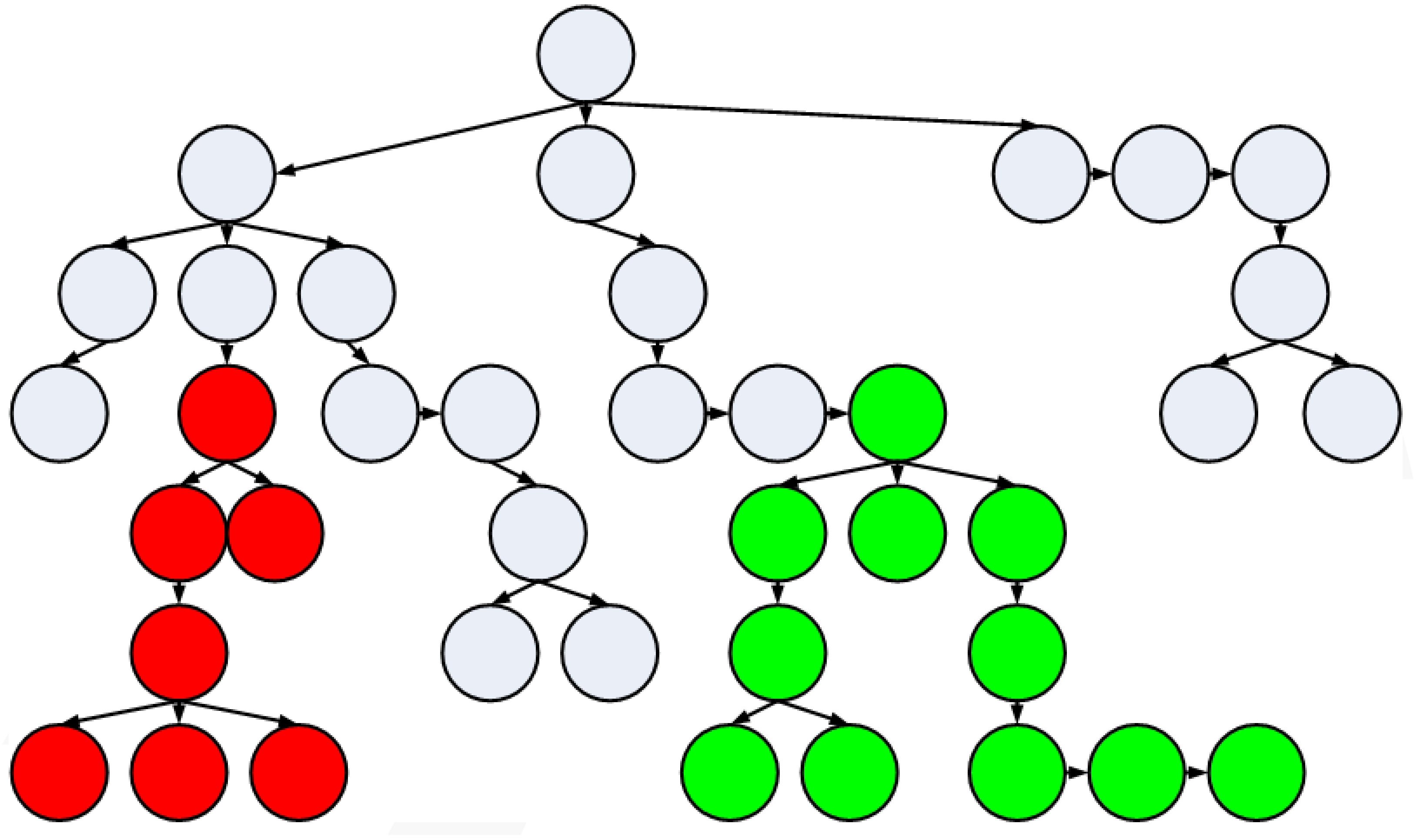
Component Testing

- Usually tests non-human interfaces
  - APIs, services, etc.
  - Important to build these in and expose them for testability
  - Architecting for testability
- Can be a bit blurry where unit testing ends and component-level testing begins
- Dependencies no longer isolated
- Can use similar tools as unit testing + others

Component Level Testing



More Component Level Testing



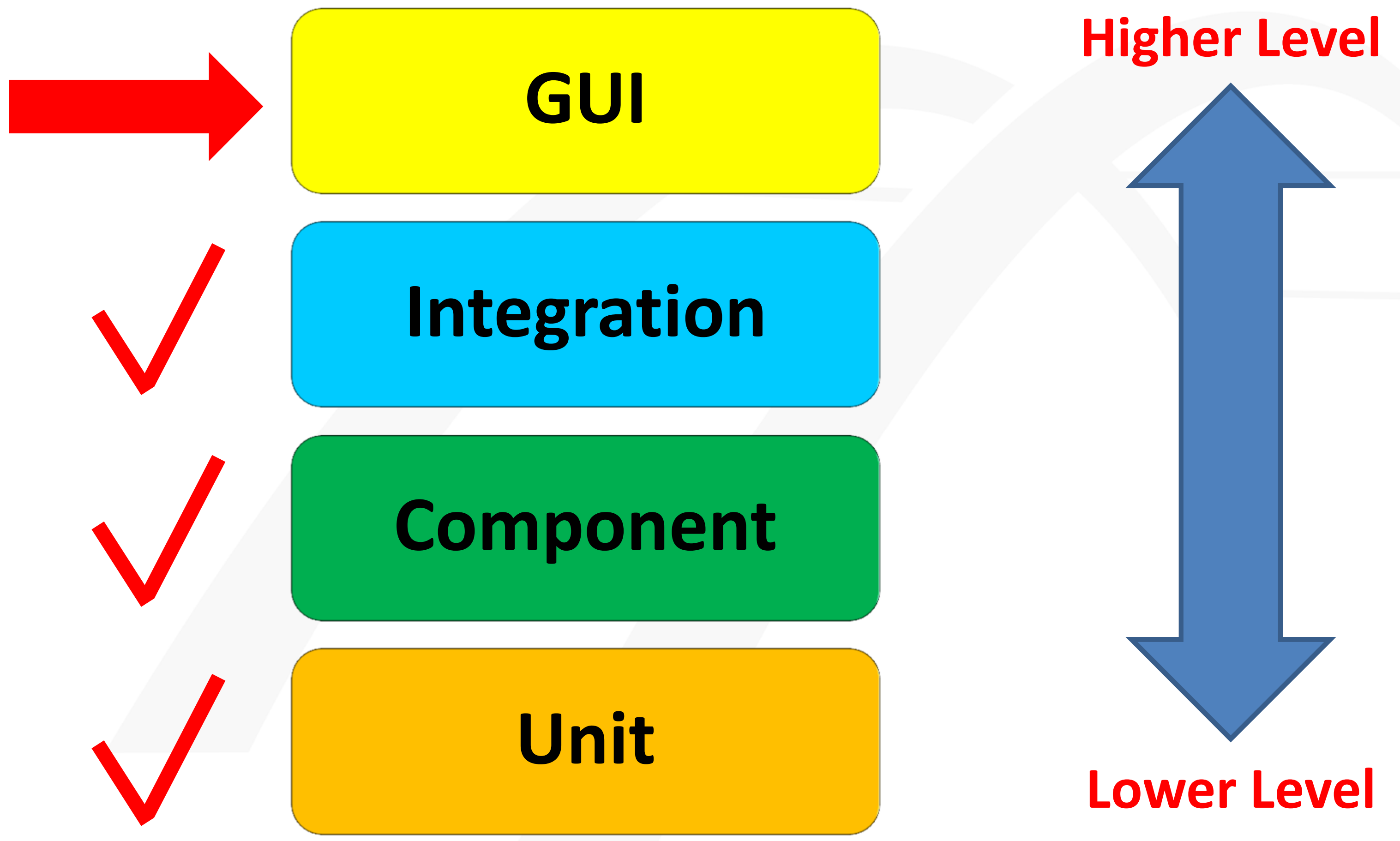
The Balance

- Bottom-up testing
  - Cheaper to do
  - Good at testing low-level details
  - Bad at testing interactions among pieces
- Top-down testing
  - More expensive to do
  - Bad at testing low-level details
  - Good at testing interactions among pieces

Component vs. Integration Testing

- Component testing
  - Usually refers to a group of related functionality but smaller than an entire application
- Integration testing
  - Usually refers to large chunks of functionality, often testing how whole applications work together
  - Basically testing large components working together

Types of Test Automation

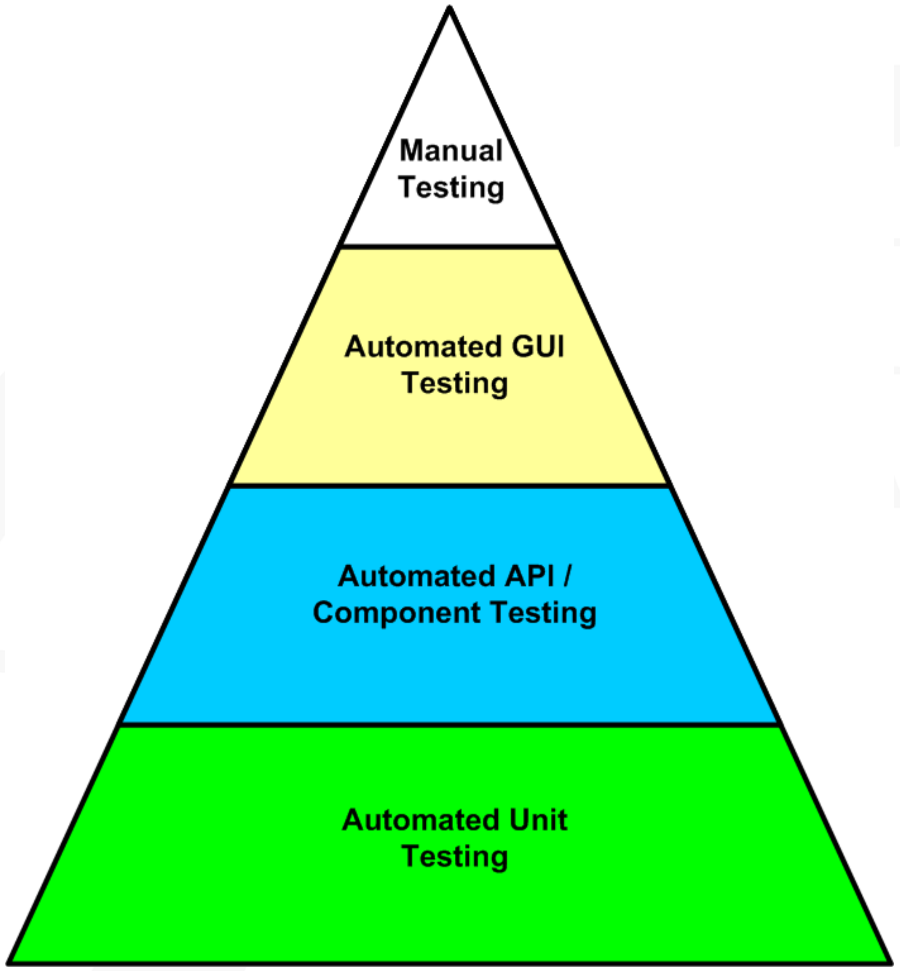


Automated GUI Testing

- Not many recent fundamental changes
  - Browser based apps provide standard interfaces
- Tools are getting cheaper
  - Many excellent open-source options, e.g. Selenium
- Mobile testing still in infancy
  - Lots of hard problems to solve!
- Patterns are emerging
  - E.g. Page Objects
- Still hard to do well

Evolution, not revolution…

Automation Strategy: The Goal



Agile Testing, Crispin & Gregory - http://amzn.to/KnE72I

Don’t Skimp on Integration Tests!

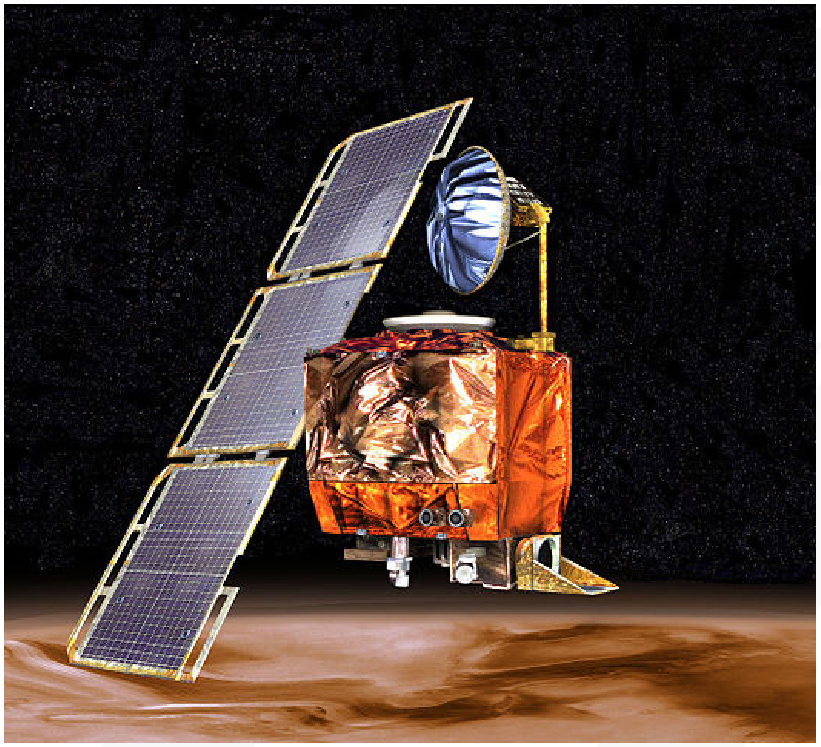


Image: NASA/JPL/Corby Waste

- In 1999 NASA lost the Mars Climate Orbiter
- Reason: Lack of / poor integration testing
- Cost of program: $328 million

Just because each piece works separately doesn’t mean they will work together.

The Future!

The Future of Testing

- So, um, what happens to all the manual testers?
  - Manual testers are freed up to do higher-value work
  - Some testers will write automated test scripts
  - We will always have some manual testing
  - Some testers will exit the field

The Future of Testing

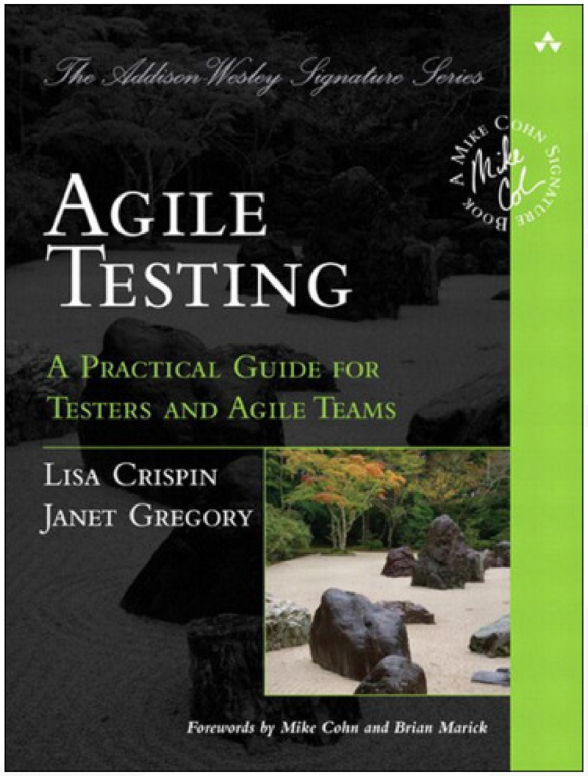
- More and better built-in test features
  - More out-of-the-box support
- More integration with dev ops
  - Still a lot of DIY required to set all this stuff up

Chris's Prediction

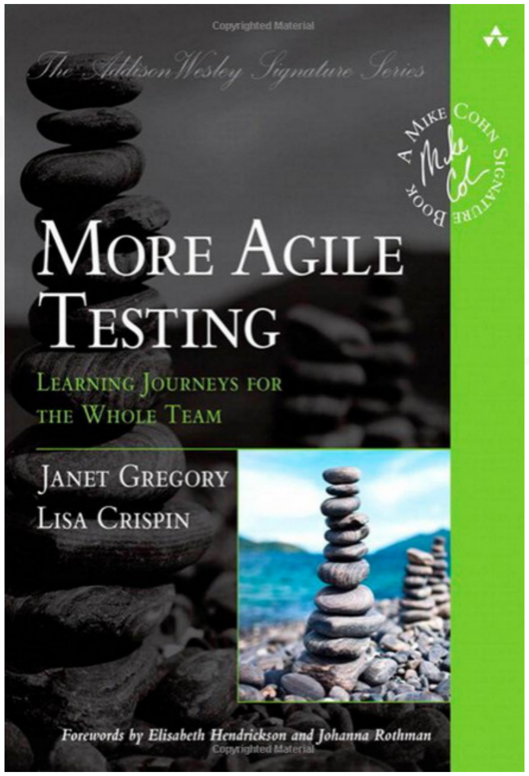
In 5 – 10 years, it will be ***weird*** not to do test automation

Additional Resources

Additional Resources

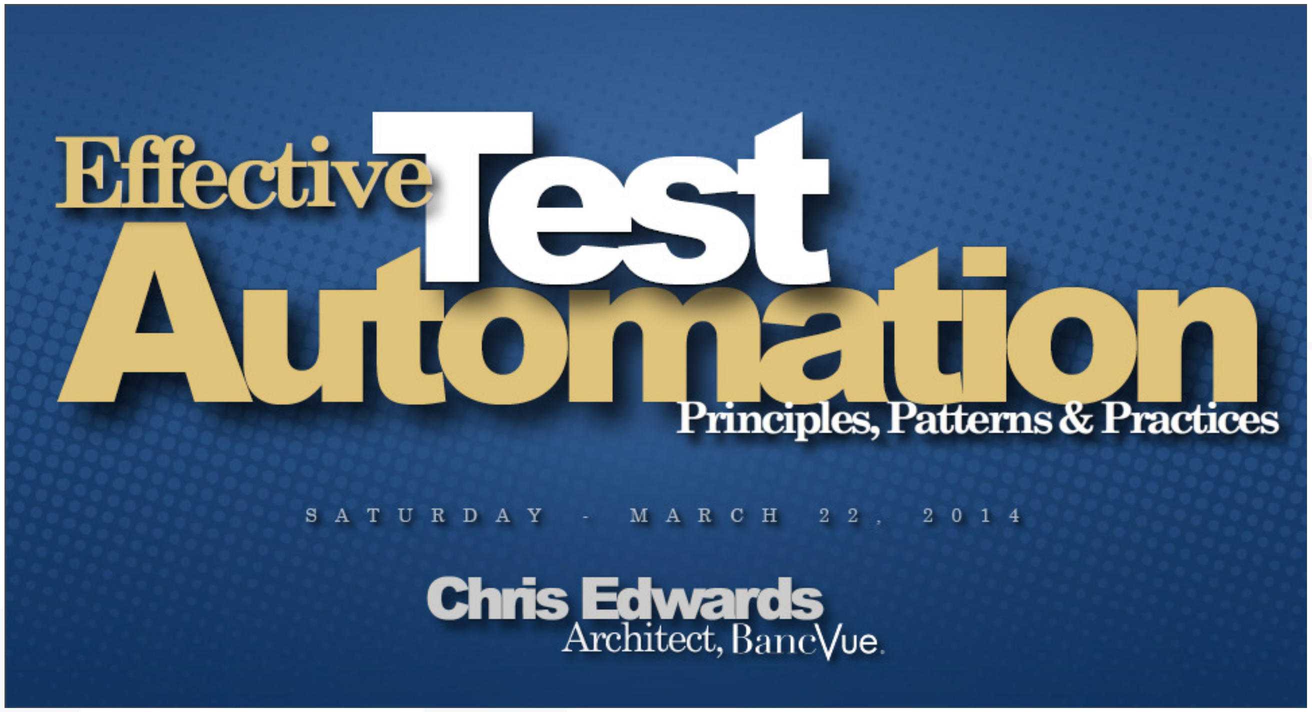

The Addison-Wesley Signature Series
A Mike Cohn Signature Book
Agile Testing
A Practical Guide for Testers and Agile Teams
Lisa Crispin
Janet Gregory
Forewords by Mike Cohn and Brian Marick

The go-to book for agile testing (2009) – http://amzn.com/0321534468


The Addison-Wesley Signature Series
A Mike Cohn Signature Book
More Agile Testing
Learning Journeys for the Whole Team
Janet Gregory
Lisa Crispin
Forewords by Elisabeth Hendrickson and Johanna Rothman

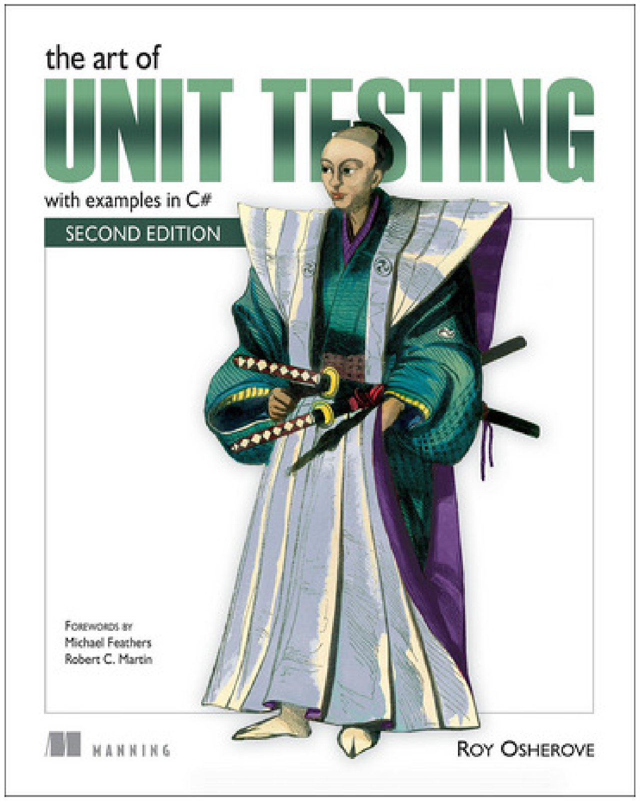
Additional material from the same authors (2014) – http://amzn.com/0321967054

Recommended Reading

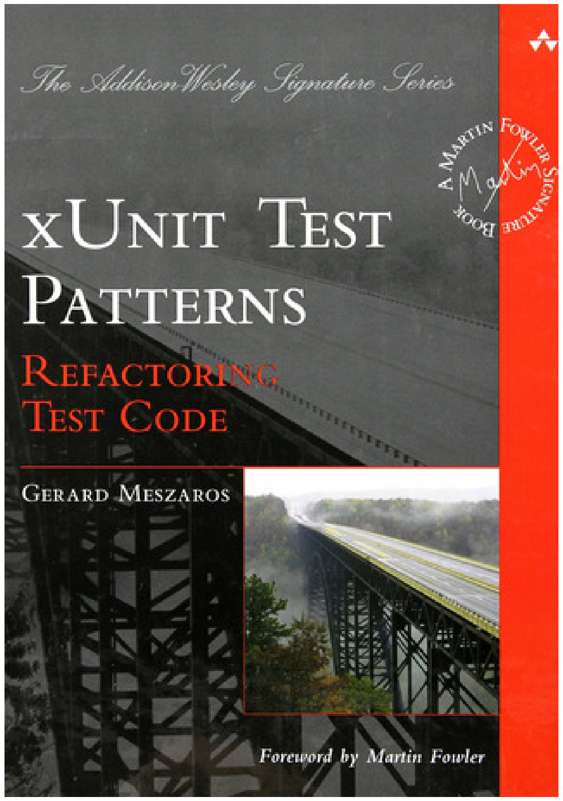

Effective Test Automation
Principles, Patterns & Practices
SATURDAY - MARCH 22, 2014
Chris Edwards
Architect, BancVue.

Excellent presentation on test automation from Keep Austin Agile 2014.
http://architester.com/blog/2014/03/25/effective-test-automation-presentation-keep-austin-agile-2014/

Recommended Reading



Great introduction to unit testing.
Examples are in C# but easy to follow.
http://amzn.com/1617290890


The Addison Wesley Signature Series
A Martin Fowler Signature Book
xUnit Test Patterns
Refactoring Test Code
Gerard Meszaros
Foreword by Martin Fowler

Deeper dive into unit testing.
http://amzn.com/0131495054

Bridge360™

VETERANS 4 QUALITY™

http://www.bridge360.com/v4qlanding.shtml

Thank you!
Questions?