

HOW I LEARNED TO LOVE ACCEPTANCE TESTING

JASON CARTER SOFTWARE ENGINEERING TEAM LEAD

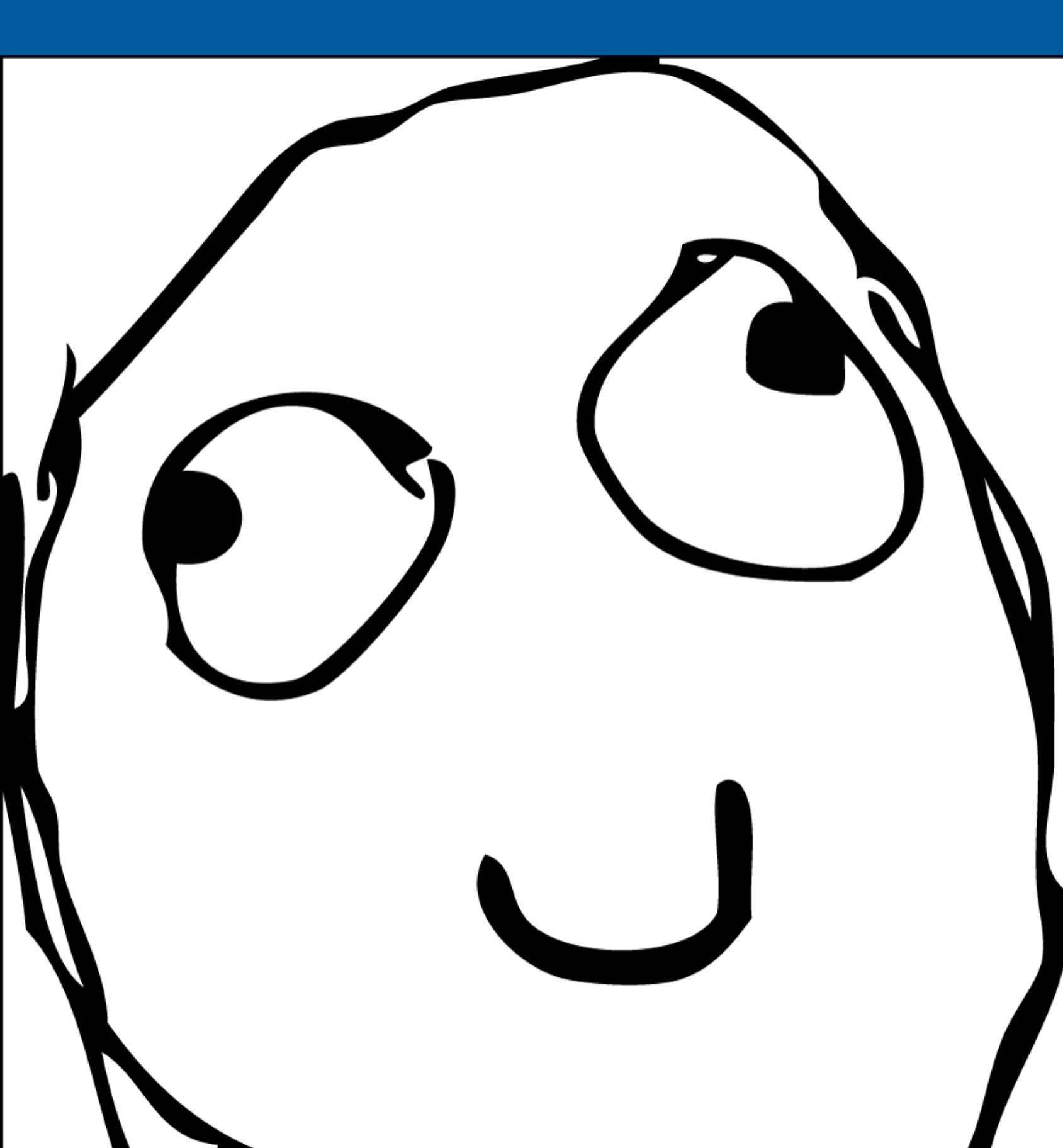
@JASON URUG

A photograph showing two programmers from behind, working at a desk. They are looking at multiple computer monitors displaying lines of code. The monitors are arranged in a row, and the screens are filled with text and some graphical elements. The programmers are wearing dark t-shirts. The background is slightly blurred, focusing on the monitors and the code.

**EXTREME PROGRAMMING
PAIR PROGRAMMING
TEST DRIVEN DEVELOPMENT
RAD HYPE VIDEOS**

MAVENLINK.COM/ENGINEERING

A QUICK
STORY...



LE ME

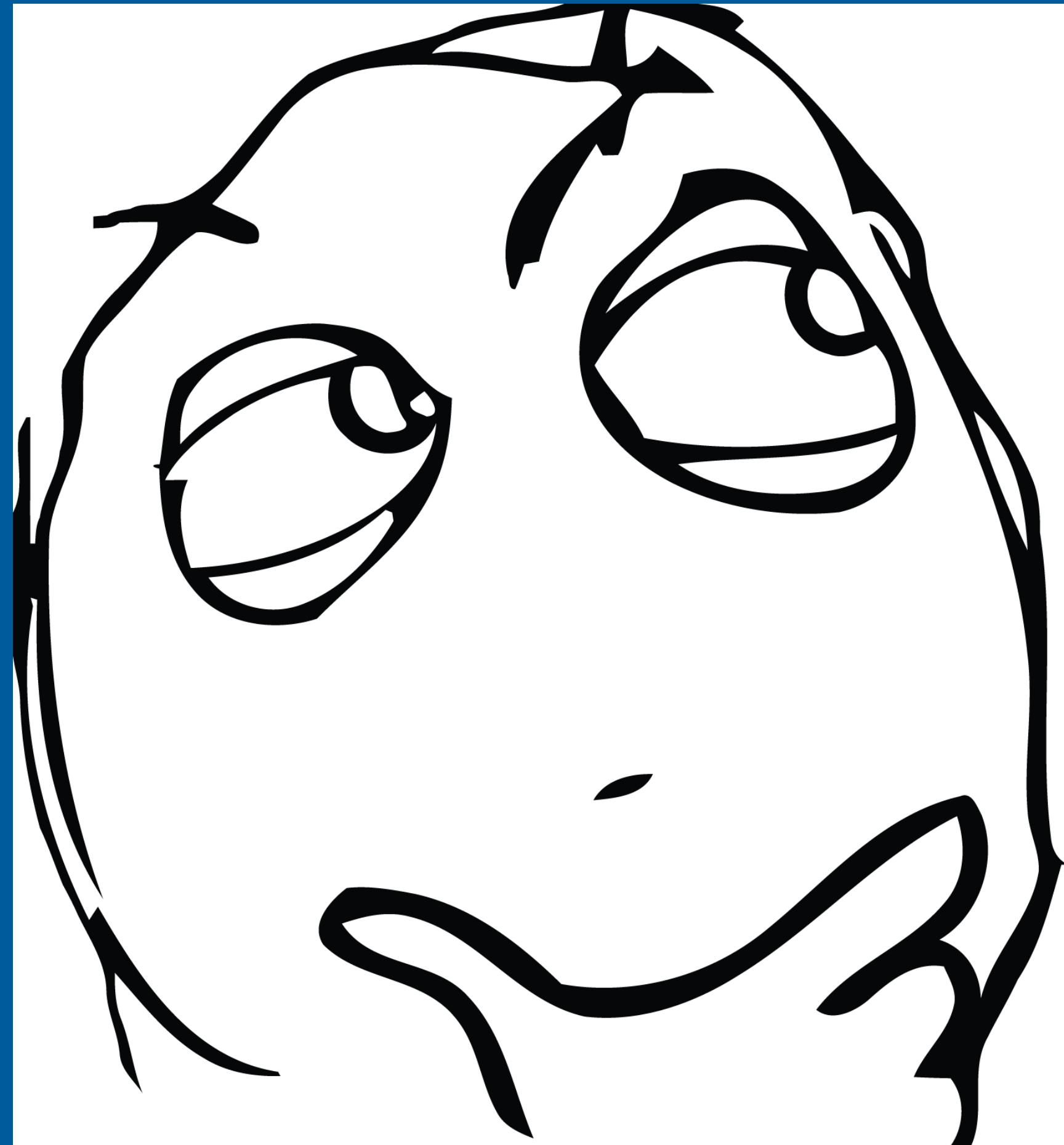
TEST DRIVEN DEVELOPMENT

Write a

Make the

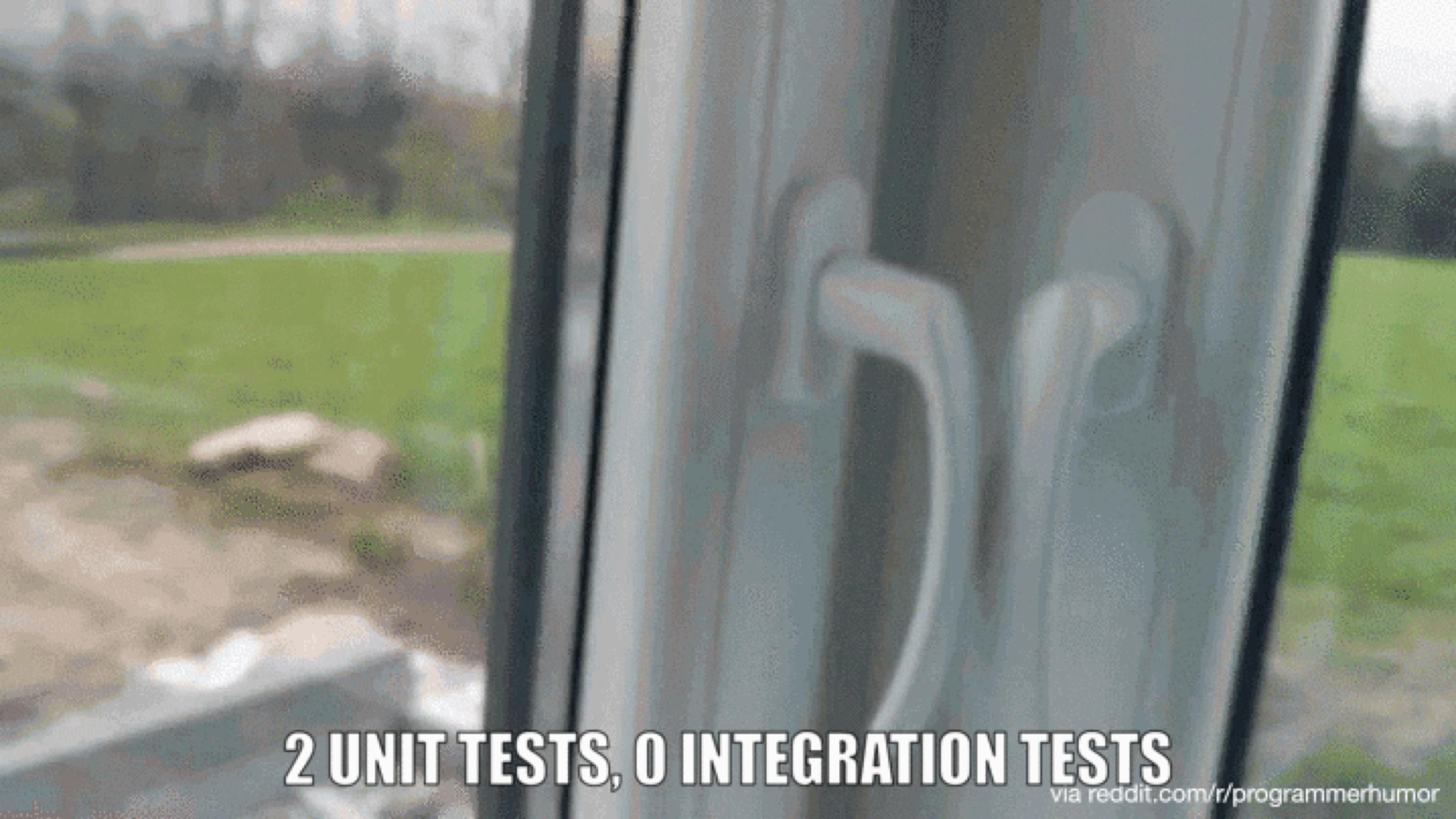
Refactor

**...SO THAT MEANS.
WRITE A UNIT TEST.
IMPLEMENT IT**



UNIT TESTS GAVE ME
CONFIDENCE THAT MY
CLASS WORKED AS
EXPECTED

**BUT THERE WAS A
PROBLEM...**

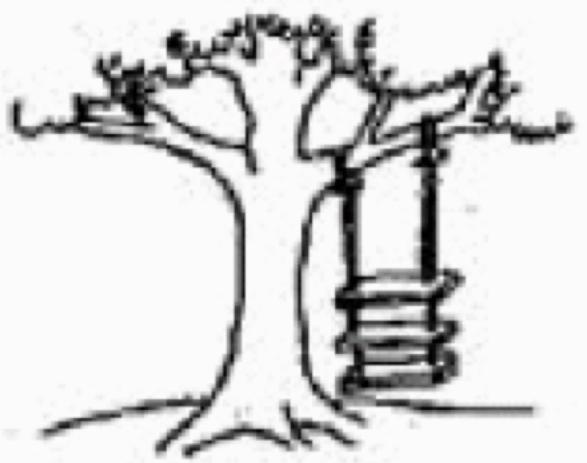


2 UNIT TESTS, 0 INTEGRATION TESTS

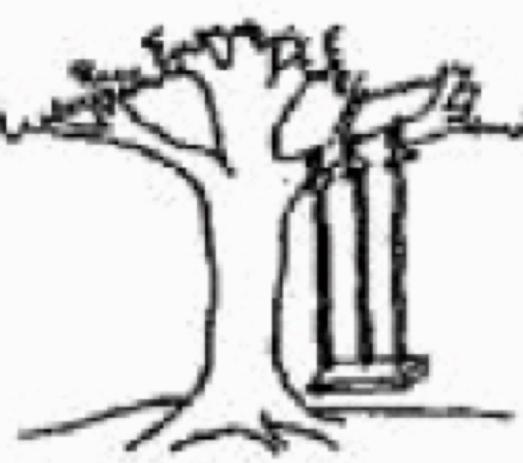
via reddit.com/r/programmerhumor

SO I GOT BETTER AT
WRITING INTEGRATION
TESTS...

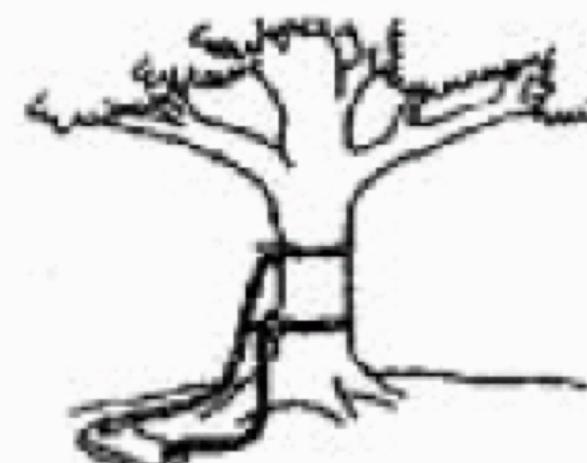
**INTEGRATION TESTS GAVE
ME CONFIDENCE THAT MY
CLASSES WORKED
TOGETHER AS I EXPECTED**



**As proposed
by the project
sponsor.**



**As specified
in the project
request.**



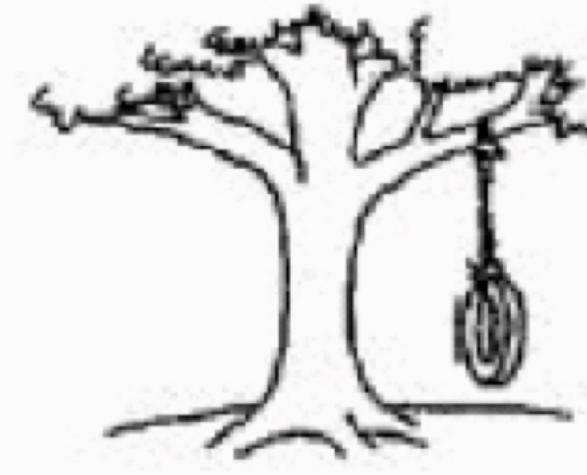
**As designed
by the senior
architect.**



**As produced
by the
engineers.**

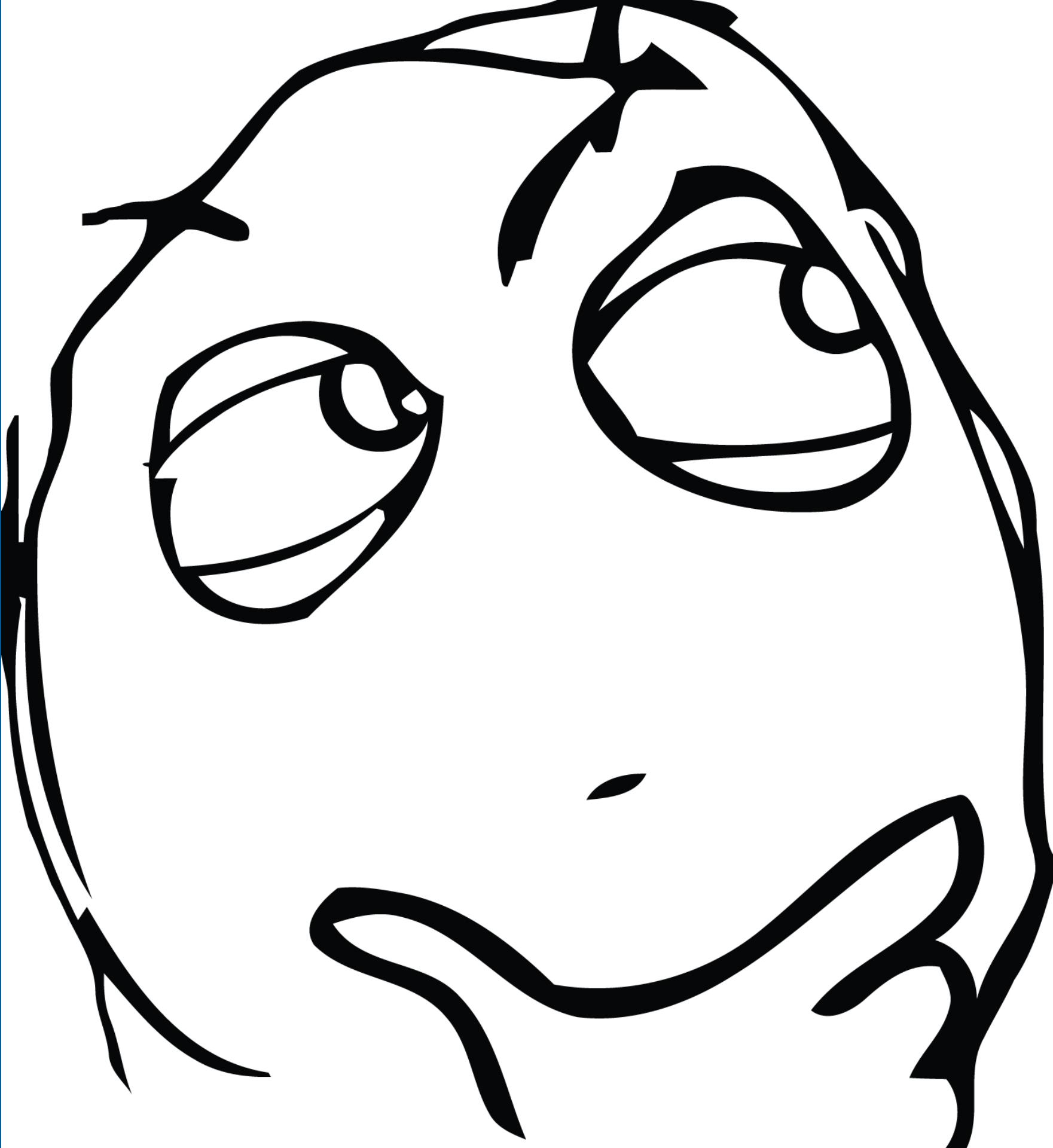


**As installed at
the user's
site.**



**What the
customer
really wanted.**

SO HOW DO YOU MAKE
SURE THAT YOU'RE
BUILDING THE RIGHT THING,
AND THAT IT WORKS?

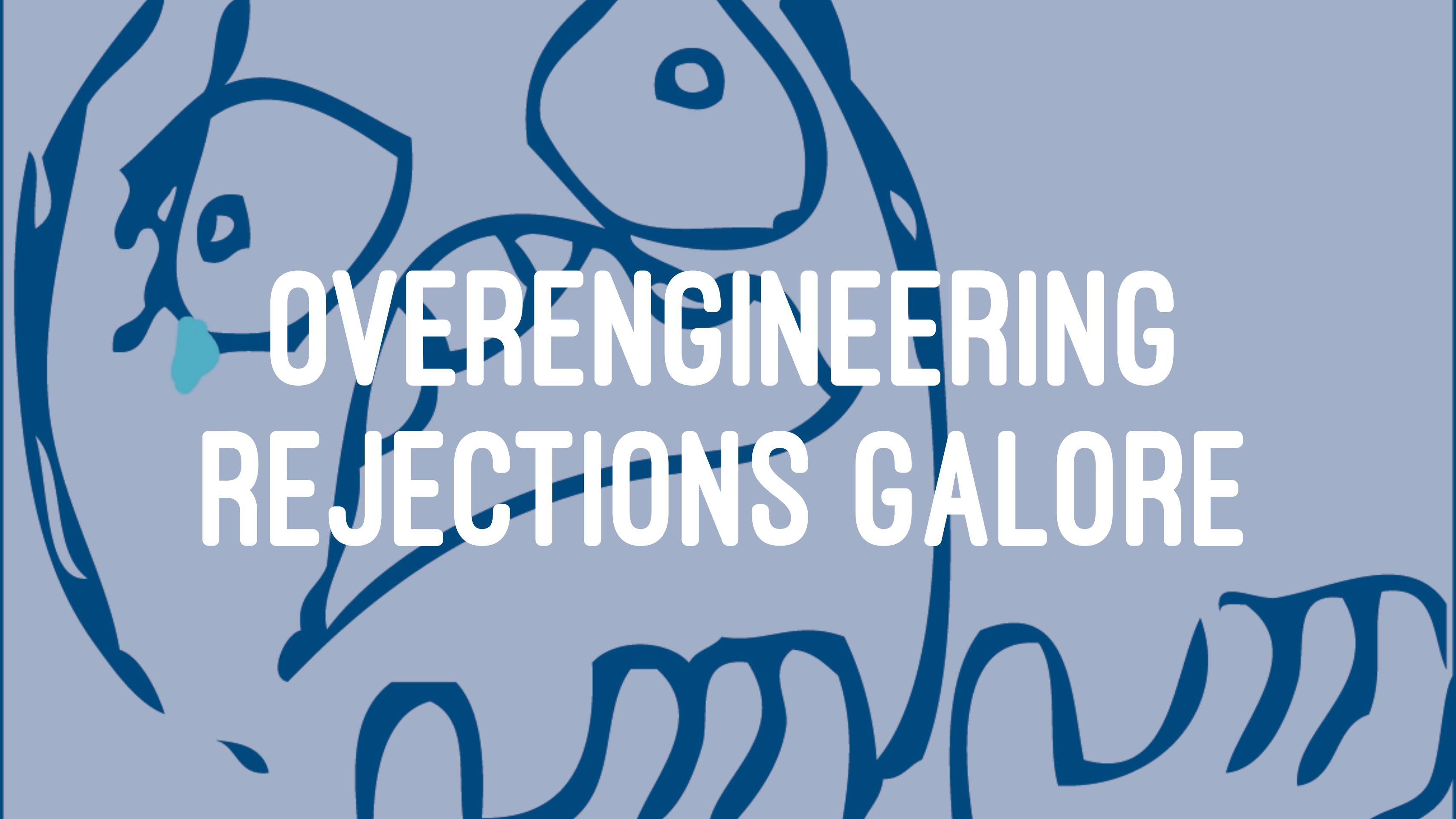




FAST FORWARD A YEAR...



FIRST BIG PROJECT AS A TEAM LEAD



OVERENGINEERING
REJECTIONS GALORE

**START WITH AN
ACCEPTANCE TEST**

ACCEPTANCE TESTING

FORMAL TESTING WITH RESPECT TO USER NEEDS, REQUIREMENTS, AND BUSINESS PROCESSES CONDUCTED TO DETERMINE WHETHER OR NOT A SYSTEM SATISFIES THE ACCEPTANCE CRITERIA AND TO ENABLE THE USER, CUSTOMERS OR OTHER AUTHORIZED ENTITY TO DETERMINE WHETHER OR NOT TO ACCEPT THE SYSTEM.

-- INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD

ACCEPTANCE TESTING

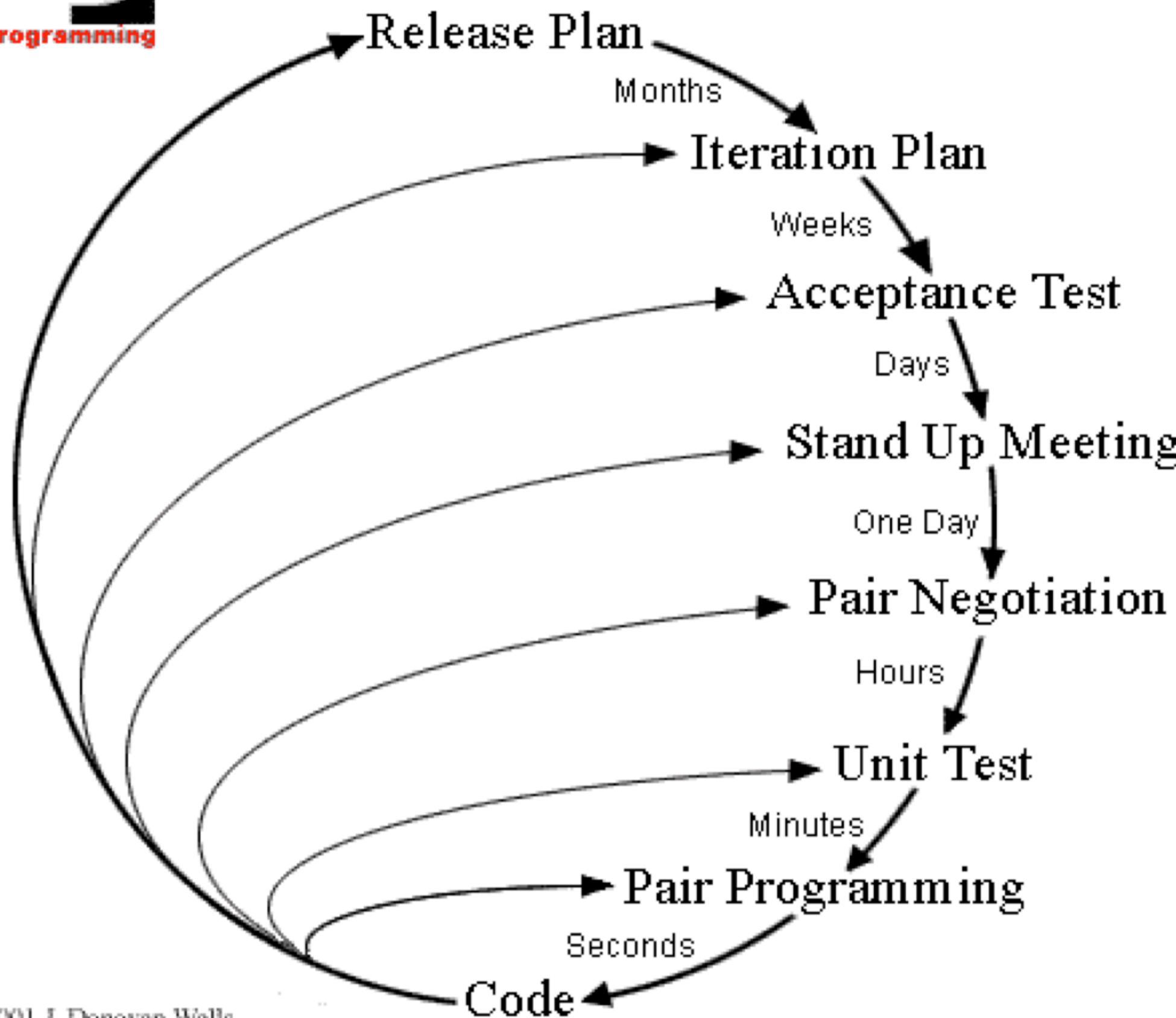
A TEST THAT MUST PASS FOR THE USER STORY YOU'RE WORKING
ON TO BE COMPLETE
- JASON CARTER

**TURNS OUT THIS IS
TOTALLY PART OF XP**



Planning/Feedback Loops

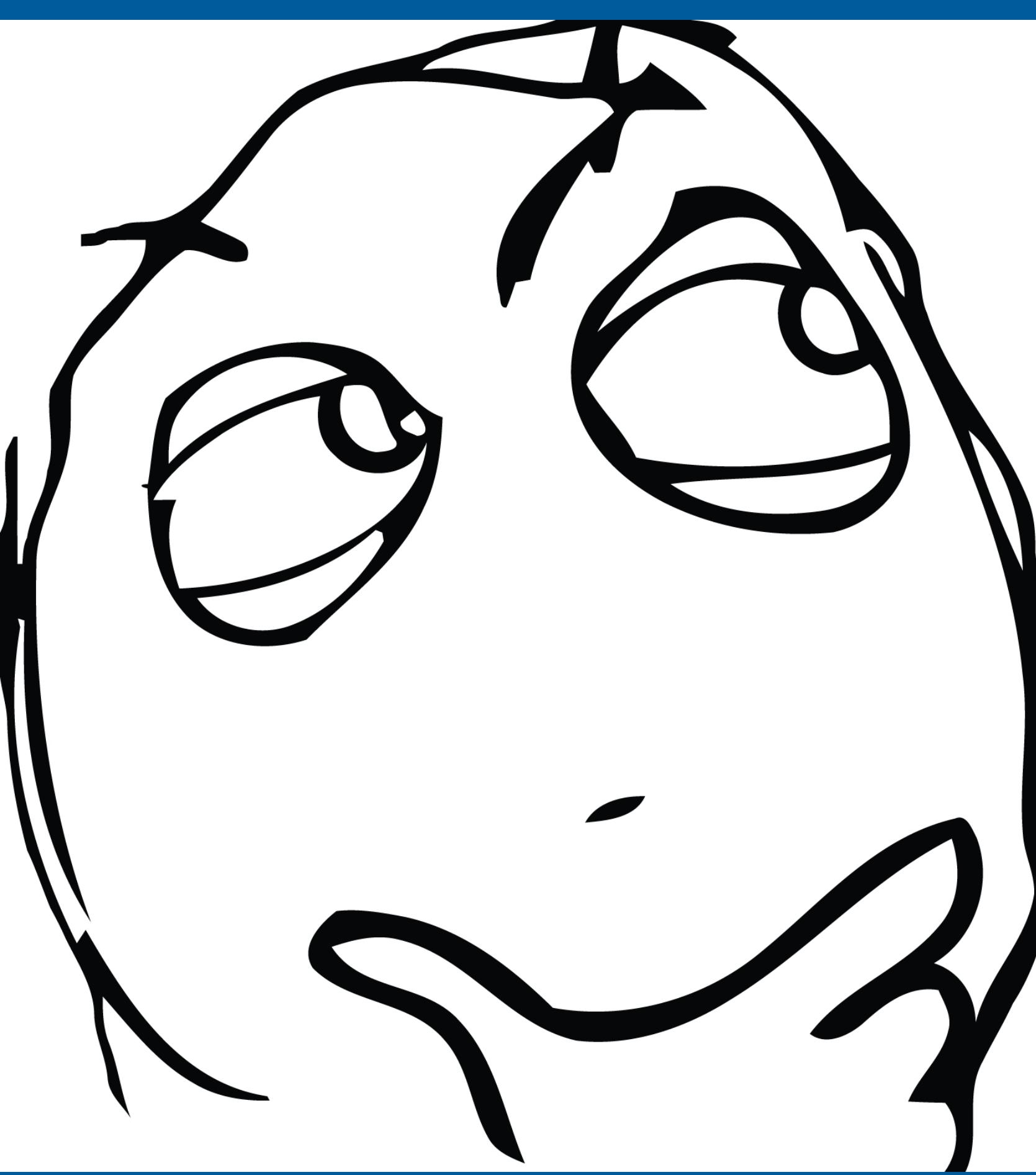
Zoom Out



THE NAME ACCEPTANCE TESTS WAS CHANGED FROM FUNCTIONAL TESTS. THIS BETTER REFLECTS THE INTENT, WHICH IS TO GUARANTEE THAT A CUSTOMERS REQUIREMENTS HAVE BEEN MET AND THE SYSTEM IS ACCEPTABLE.

-- DON WELLS. EXTREMEPROGRAMMING.ORG

NOW YOU MAY BE
THINKING...



ISN'T THAT WHAT QA IS
FOR?

**QA HAS BETTER THINGS
TO DO**

A code tester walks into a bar.
Orders a beer.
Orders ten beers.
Orders 2.15 billion beers.
Orders -1 beers.
Orders a nothing.
Orders a cat.
Tries to leave without paying.

THEY'RE GREAT AT
FINDING EDGE CASES

**THE ACCEPTANCE TEST
TELLS YOU WHEN YOU'RE
DONE... IMMEDIATELY**

WE'RE STARTING TO PAIR
WITH QA ON THESE TESTS

A DEVELOPER SAYS TO A QA PERSON



"WORKS ON MY MACHINE"

PATTERN BECOMES:

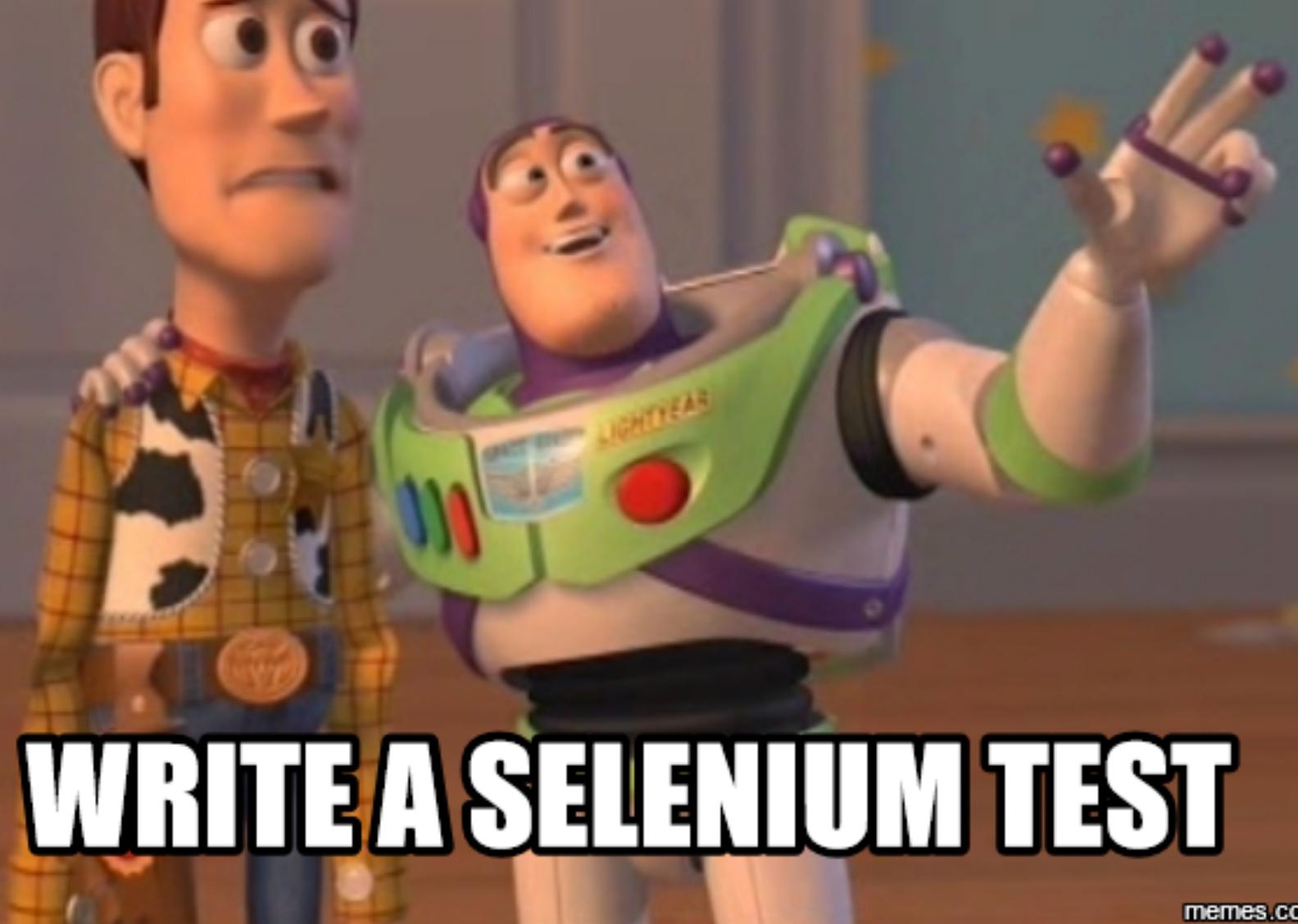
- › WRITE THE ACCEPTANCE TEST TO KNOW WHEN THE FEATURE IS DONE
- › TEST DRIVE THE IMPLEMENTATION WITH UNIT AND INTEGRATION SPECS
- › REFACTOR SAFELY KNOWING THAT THE FEATURE STILL WORKS THE WAY IT SHOULD

**SO HOW DO WE
ACCOMPLISH THIS IN A
RAILS APP?**

SELENIUM

USUALLY PEOPLE HAVE
STRONG OPINIONS ON
SELENIUM...

WHEN EVERYTHING ELSE FAILS



WRITE A SELENIUM TEST

**SHOULD BE USED AS A
LAST RESORT**

THAT'S QAS JOB!



**A tester
walks into a
bar and
asks for -1
pint of
beer.**

by Jokin Aspiazu



IT FAILS FOR NO
REASON!



SO WE SET OUT TO MAKE
IT PALATABLE TO ME...

...AND TO PRODUCT AND QA.

TOPICS

- GENERAL PRINCIPLES
- AVOIDING "INTERMITTENT FAILURES"
 - VANILLA
- SETUP, ACTIONS, AND ASSERTIONS
 - PAGE OBJECT PATTERN
 - RANDOM MUSINGS

PRINCIPLES

**AN ACCEPTANCE TEST
SHOULD...**

**EXERCISE YOUR APP TO
ITS FULLEST**

**AVOID MOCKING AS MUCH
AS POSSIBLE**

ONLY SIMULATE WHAT A
USER COULD DO

**GIVE YOU CONFIDENCE
THAT YOUR APP IS
WORKING AS IT SHOULD**

BE THE FIRST TEST YOU
WRITE IN THE CASE OF A
REGRESSION

**AVOIDING 'INTERMITTENT
FAILURES & SUCCESSES'**

RACE CONDITIONS

EXPLICIT VS IMPLICIT WAIT

NOT SCOPING

VANILLA

```
describe 'when deleting an existing Review', js: true, type: :request do
  # A bunch of lets
  context 'When the user is a review creator' do
    it 'can be deleted' do
      find('.workspace-proof-title', text: proof.title).click
      within '.workspace-proofs-table' do
        within 'tr', text: 'some proof' do
          expect(page).to have_text('In Review ( 1/2 )')
        end
      end
      within '#side-panel' do
        find('span', text: 'Delete Review').click
      end
      within '.show-mavenlink-alert' do
        expect(page).to have_selector('h2', text: 'Delete')
        expect(page).to have_selector('button', text: 'Delete')
        expect(page).to have_selector('button', text: 'Cancel')
        click_button('Delete')
      end
      within '#side-panel' do
        expect(page).to have_button('Create Review')
      end
      find('.js-panel-close-link').click
      find('.workspace-proof-title', text: 'Some Proof').click
      within '#side-panel' do
        expect(page).to have_button('Create Review')
      end
      within '.workspace-proofs-table' do
        within 'tr', text: proof.title do
          expect(page).to have_no_text('In Review ( 1/2 )')
        end
      end
    end
  end
end
```

PROS

**AT THE VERY LEAST
WE'RE AVOIDING
INTERMITTENT FAILURES**

CONS

ONLY A DEV CAN READ
THIS, AND PROBABLY ONLY
ONE WITH CONTEXT

MIXES ASSERTIONS AND
ACTIONS

**TOUGH TO CREATE A NEW
TEST IMMEDIATELY**

IT OFFENDS MY DRY
SENSIBILITIES

**IF I CHANGE THE DOM, I
HAVE TO UPDATE A BUNCH
OF STUFF...**

**... AND I MIGHT NOT KNOW
WHAT THE BEHAVIOR WAS
IN THE FIRST PLACE?**

**BUT WHAT ABOUT
COMMENTS, JASON?**

COMMENTS ARE JUST
ANOTHER THING TO KEEP
UP TO DATE

SURE COMMENTS ARE SOMETIMES NECESSARY

A GOOD METHOD NAME IS
GENERALLY A FINE
COMMENT

AND IF YOU NEED
COMMENTS, CHANCES ARE
YOUR METHOD IS DOING
TOO MUCH*

* JASON OPINION

SO IF WE DON'T LIKE
COMMENTS, HOW DO WE
MAKE THIS MORE
UNDERSTANDABLE?

SETUP, ACTIONS, AND ASSERTIONS

```
describe 'when deleting an existing Review', js: true, type: :request do
  # a bunch of lets
  it 'can be deleted' do
    navigate_to_proof_tab(proof_employee, workspace)
    click_proof_title(proof)
    it_should_have_proof_with_status(proof, 'In Review ( 1/2 )')
    click_delete_review
    confirm_action 'Delete'
    it_has_create_review_option
    it_persists_the_delete(proof)
    it_should_not_have_proof_with_status(proof, 'In Review ( 1/2 )')
  end
end
```

PROS

DEFINITELY EASIER TO
READ

REUSABLE METHODS!

ABSTRACTED AWAY THE
DOM

CONS

NOT ALWAYS EASY TO
TELL WHAT PAGE WE'RE
ON

NOT A GREAT WAY TO
SHOW THAT AN ACTION
LEADS TO AN ASSERTION

**NOT PICTURED IS OUR
GIANT FUNCTION LIBRARY**

PAGE OBJECT PATTERN

```
describe 'inviting people to a workspace', js: true, type: :request do
  describe 'the invitation flow' do
    context 'when adding to consultants' do
      context 'when the account is not free or trialing' do
        let(:project_invitation_form) { Pages::ProjectInvitations::Form.new('maven') }
        let(:inviting_user) { users(:jane) }
        let(:invited_user) { users(:alice) }
        let(:workspace) { workspaces(:jane_car_wash) }

        it 'will invite a maven to a workspace' do
          login(inviting_user)
          visit workspace_path(workspace)
          invite_a(team, invited_user)

          expect(project_invitation_form).to have_save_message_as_default_checkbox
          expect(project_invitation_form).to have_no_subject_and_message_tipsy
          expect(project_invitation_form).to have_no_upgrade_call_to_action
          expect(project_invitation_form.send_invitation).to have_sent_an_invitation
        end
      end
    end
  end
end
```

```
module Pages
  module ProjectInvitations
    class Form
      include Capybara::DSL

      def initialize(team)
        @team = team
      end

      def has_save_message_as_default_checkbox?
        page.has_css?('#save_message_as_default')
      end

      def send_invitation
        click_button 'Send invitation'
        self
      end

      def has_sent_an_invitation?
        within('.flash-container') do
          page.has_css?('.notice', text: '1 invitation was sent for this project')
        end
      end

      def has_not_sent_an_invitation?
        within('.flash-container') do
          page.has_css?('.notice', text: 'The invited user must be a member of your account.')
        end
      end
    end
  end
end
```

A COUPLE
THINGS TO
NOTE

```
def has_sent_an_invitation?  
  within('.flash-container') do  
    page.has_css?('.notice', text: '1 invitation was sent for this project')  
  end  
end  
  
# Selenium does some magic for us  
# has_sent_invitation? -> have_sent_an_invitation
```

RETURNING self LETS US CHAIN THINGS TOGETHER, AND USE THE to SYNTAX

```
def send_invitation
  click_button 'Send invitation'
  self
end
```

```
describe 'inviting people to a workspace', js: true, type: :request do
  describe 'the invitation flow' do
    context 'when adding to consultants' do
      context 'when the account is not free or trialing' do
        # ...
        expect(project_invitation_form.send_invitation).to have_sent_an_invitation
      end
    end
  end
end
```

PROS

MUCH EASIER TO TELL
WHAT PAGE WE'RE ON!

**WE CAN SCOPE ACTIONS
AND ASSERTIONS TO
PAGES (OR SUBPAGES!)**

STILL PRETTY READABLE!
IN FACT, LETS US SEE
CAUSE AND EFFECT MUCH
EASIER

MARTIN FOWLER SAID ITS
AWSOME!

CONS

I HAVEN'T FOUND ANY
YET...

OTHER MUSINGS

```
expect(project_invitation_form.send_invitation).to [
  have_sent_an_invitation,
  have_notified_some_service,
  have_done_something_else
]
```

**WHAT IF I WANT TO MAKE
THE SAME ASSERTIONS
ON MULTIPLE PAGES?**

SHARED ASSERTIONS BETWEEN PAGE OBJECTS

**WHAT ABOUT COMMON
TASKS LIKE LOGGING IN?**

**SHARED ACTIONS
BETWEEN PAGE OBJECTS
OR DONE BEFORE SETTING
UP THE PAGE OBJECT.**

3RD PARTY INTEGRATIONS?

**WE INTEGRATE WITH A
3RD PARTY VIA DELAYED
JOB**

WE CAN ASSERT THAT
DELAYED JOB IS CALLED.
BUT WHAT ABOUT THAT IT
DOES THE RIGHT THING?

```
def run_conceptshare_jobs_immediately
  stub(Delayed::Job).enqueue do |job|
    job.perform if job.respond_to?(:queue_name) && job.queue_name == 'conceptshare'
    true
  end
end
```

```
context 'when adding to a workspace mapped to conceptshare' do
  it 'will create a conceptshare mapping for the participation' do
    run_conceptshare_jobs_immediately
    mock.any_instance_of(Conceptshare::AccountClient).create_user_if_necessary(invited_user)
    mock.any_instance_of(Conceptshare::AccountClient).add_user_to_account(invited_user)
    mock.any_instance_of(Conceptshare::AccountClient).add_user_to_project(invited_user, workspace, "Commentator")

    login(inviting_user)
    visit workspace_path(workspace)
    invite_a(team, invited_user)
    project_invitation_form.send_invitation
  end
end
```

THIS GETS US PART WAY
THERE

**HOW DO I MAKE SURE THE
3RD PARTY DOES WHAT I
WANT?**

VCR?

**WHAT IF YOU STILL GOT
THE BEHAVIOR WRONG?**

**CLARIFY WITH QA AND
PRODUCT UP FRONT**

HAVE THEM LOOK AT YOUR
NOW READABLE
ACCEPTANCE TESTS

BUT JASON, THIS ALL
SOUNDS WELL AND GOOD...

**HOW DO I START DOING
THIS?**



IN SUMMARY

**WE'RE AVOIDING COMMON
PITFALLS...**

WE'RE AVOIDING 'BUILDING
THE WRONG THING'...

IT SERVES AS
DOCUMENTATION THAT'S
READABLE BY ALL
STAKEHOLDERS...

IT MAKES IT EASIER TO DO
SWEEPING REFACTORS
AND MAINTAIN BEHAVIOR...

... AND



ME NOW

QUESTIONS?