

# Living on the Bleeding Edge In The Financial Industry

Using Clojure, AMQP, Chef, Cucumber and JRuby in the  
Financial Industry

Philly Emerging Technologies for the Enterprise, April 2010

# Algorithmics, Inc.

Risk Management

Collateral Management

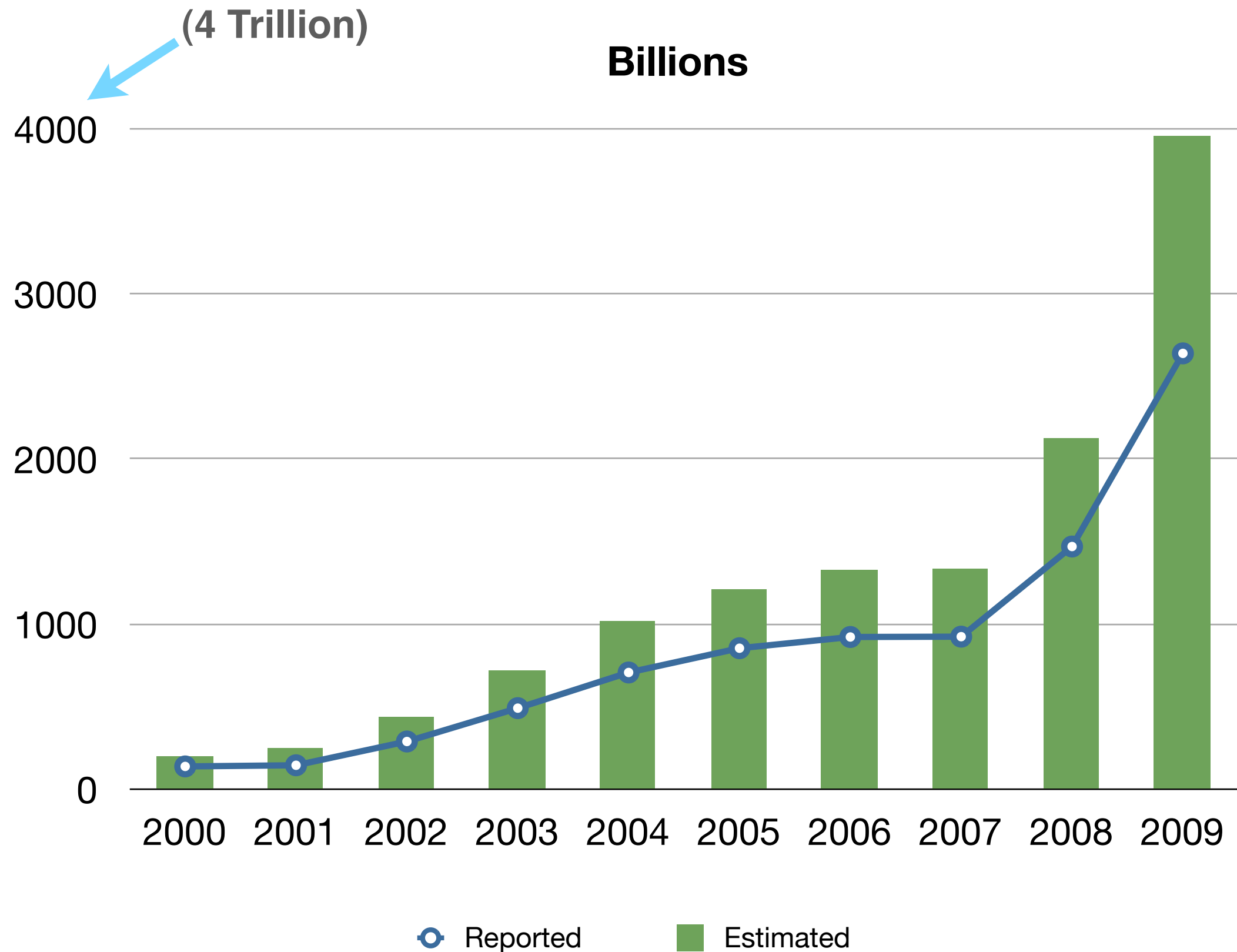
Aaron Feng

Kyle Burton

# Ok, WTF is Collateral Management?

Mitigates Credit Risks for Unsecured Financial Transactions

Unsecured Financial Transactions are OTC (over the counter - there is no 'central exchange' for collateral)



**\*2009 ISDA Margin Survey**

# Collateral Management

- Today
  - Manual Processing
  - Email, Phone, Fax
- Tomorrow
  - Automated Processing
  - Secure Messaging
  - Standard Protocol

# Overview

- What do we all want?
  - What Challenges Did we face
- How did we do/get it?
- What did we do?
  - What challenges did we face
- What has it done for us?

# What do we Want?

- Best People
- Best Technologies
- To Create: new code, not legacy
- Interesting Problems
- Best Processes

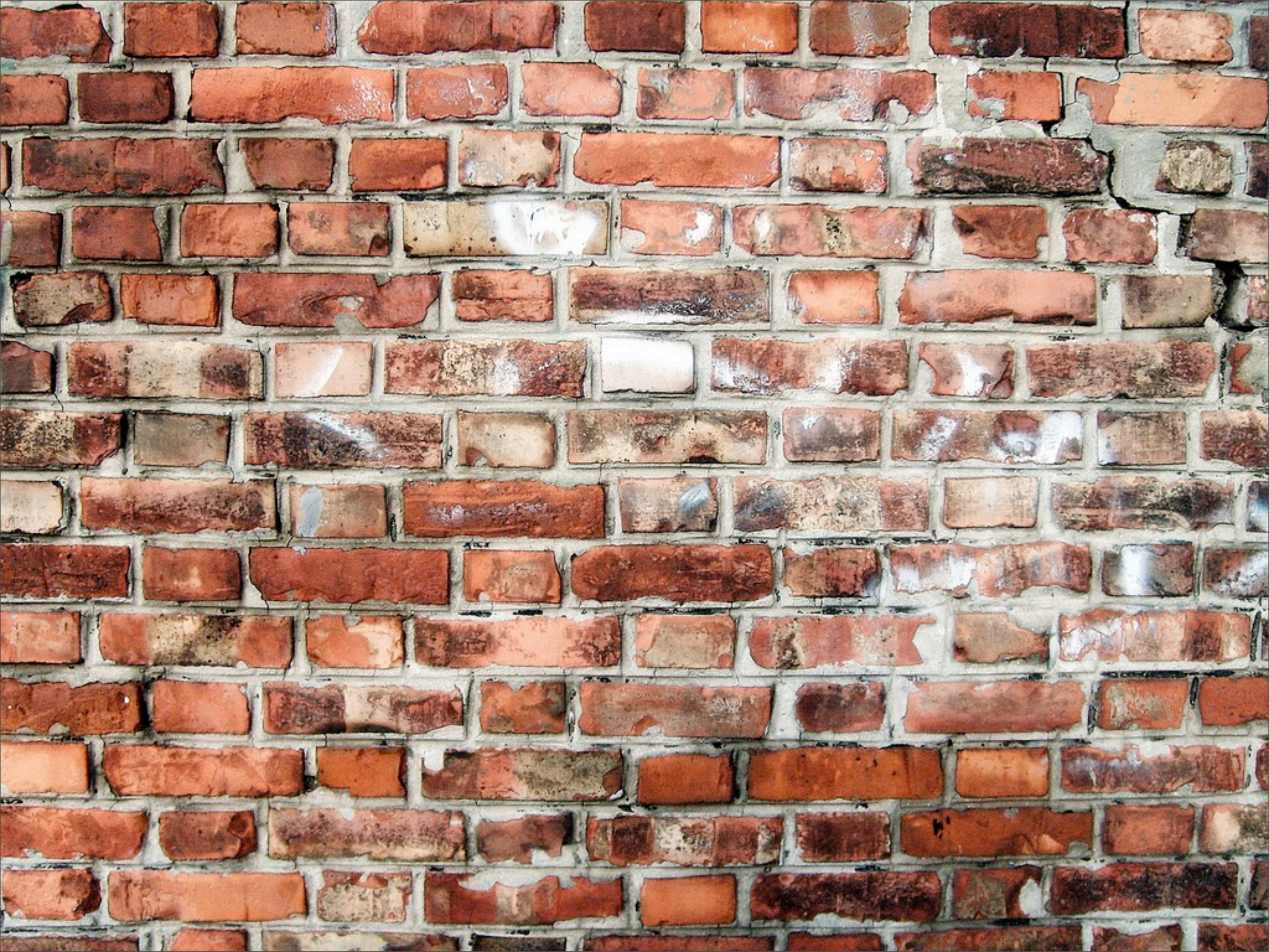




**“Operating  
System not  
Found”**









# Challenges

- Programmers: Skeptical
- Financial Industry: Conservative
- Management: Conservative, Skeptical
- Technology Choices: Clojure, Ruby, AMQP
  - Gasp: Where do we go for developers? support?
- Competition Existed: Needed To Move Quickly









# How we did it: People

- Product Visionaries Sold Higher Ups on Idea
- Time To Market Critical
- Aaron Built a Prototype *very* Rapidly
  - All by His Lonesome
  - Demonstrated Value of Technologies

Project Given Green Light June 2009

# How we did it: People

- Aaron Invested in Philly Lambda
  - Networked with Members
  - Organized, Brought in Speakers
  - Bought Lots of Pizza

First Two Developer Hires July 2009

Fourth Developer Hired August 2009

# How we did it: Support

- LShift: unlike many open source techs, a company sits behind RabbitMQ
- We knew RabbitMQ was great, it was built on Erlang
- Erlang devs don't grow on trees in Philly
- Contracted with LShift to extend and accelerate product roadmap

# How we did it: Support

- Clojure
  - JVM Technology
  - Well Known and Supported
- JRuby
  - Also a JVM Tech
  - Rails a well known commodity



# How we did it: Process

- Small Team (4 devs)
- Agile Methodology
  - Pair Programming
- Continuous Improvement
- Automation

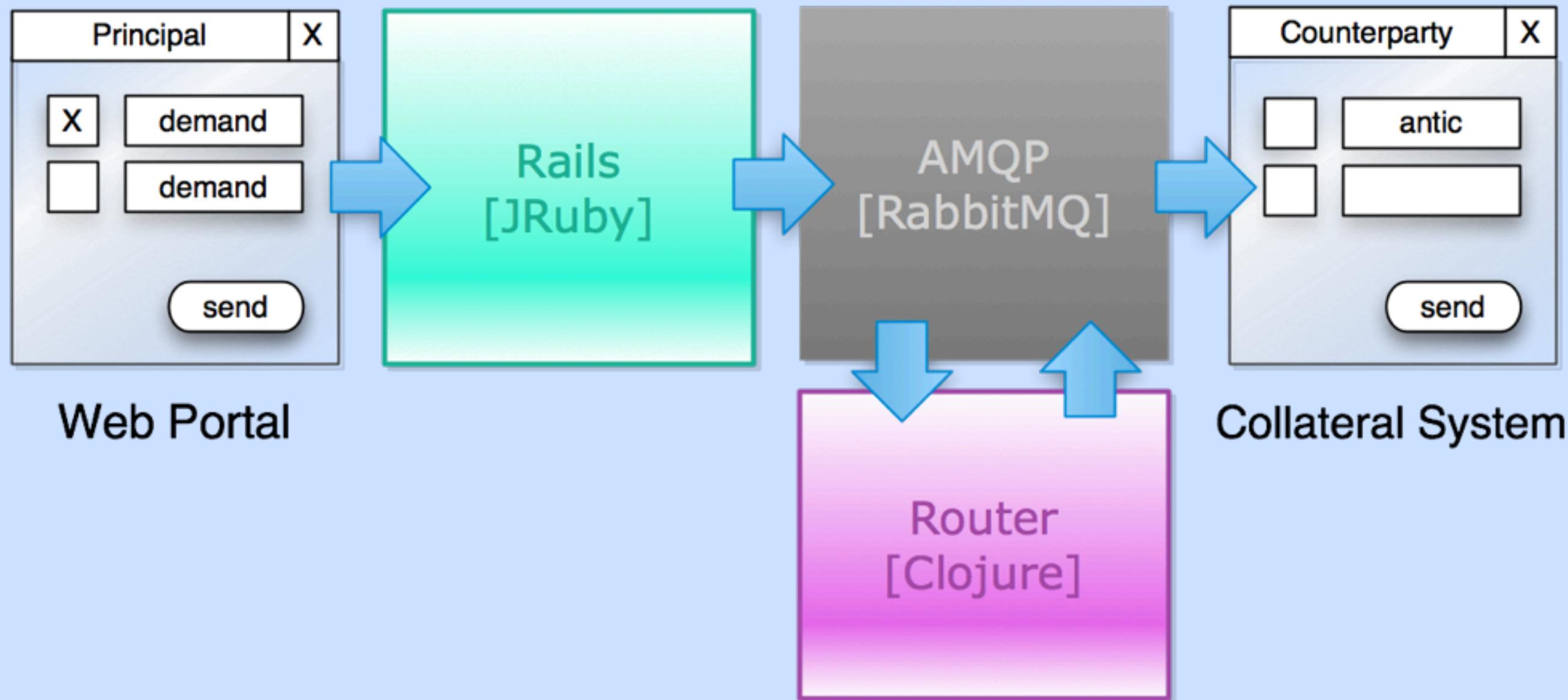


(unibomber)

“So that’s what we wanted and how we got it, now Kyle’s going to talk about what we did and what it did for us...”

(robot)





# What we did:Automation

- Focus on Automation
  - Frequent Releases
  - Provisioning
  - QA / Testing
  - A Mindset, Core Value of Team

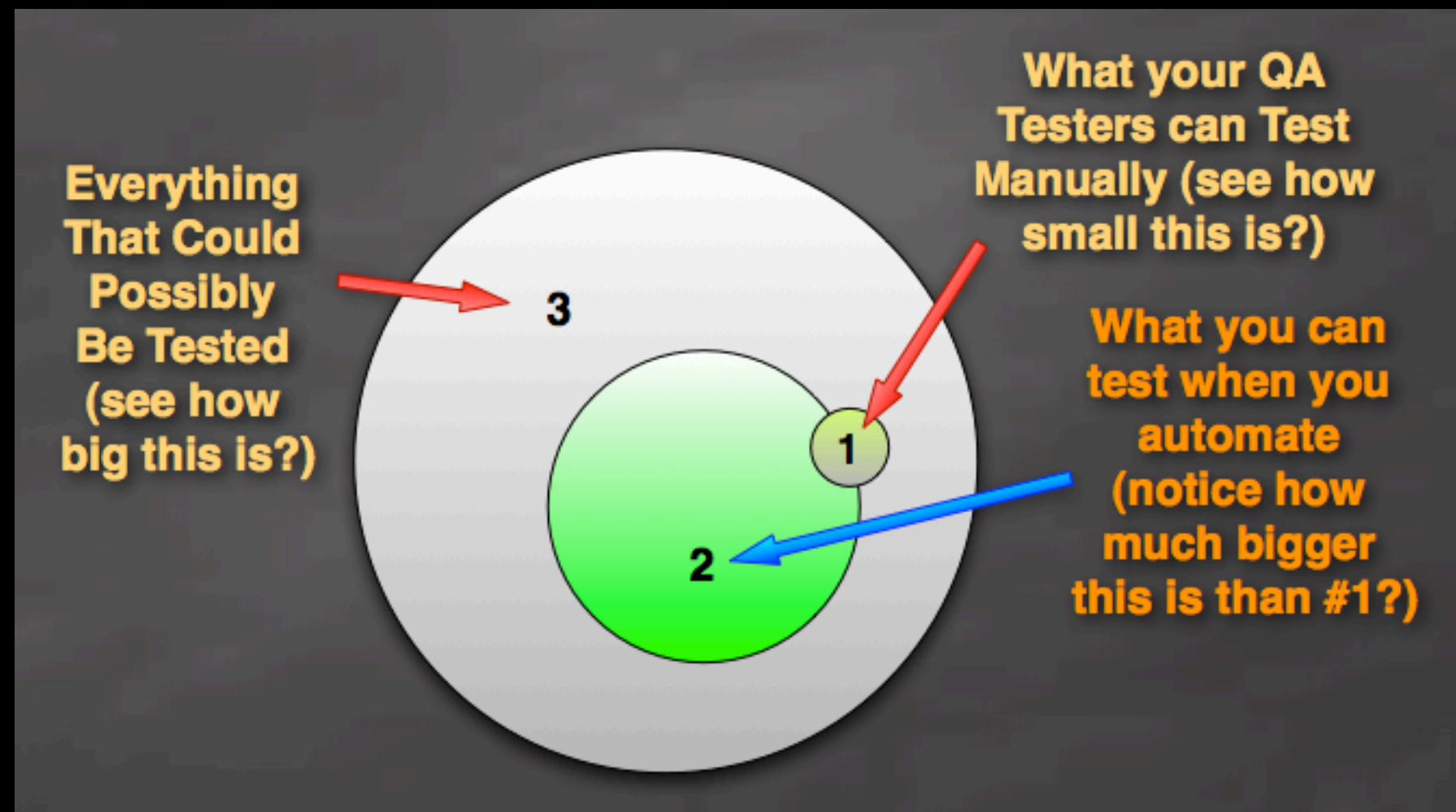
# Automation: Provisioning

- Initially Scripted with Bash
- Migrated to Chef

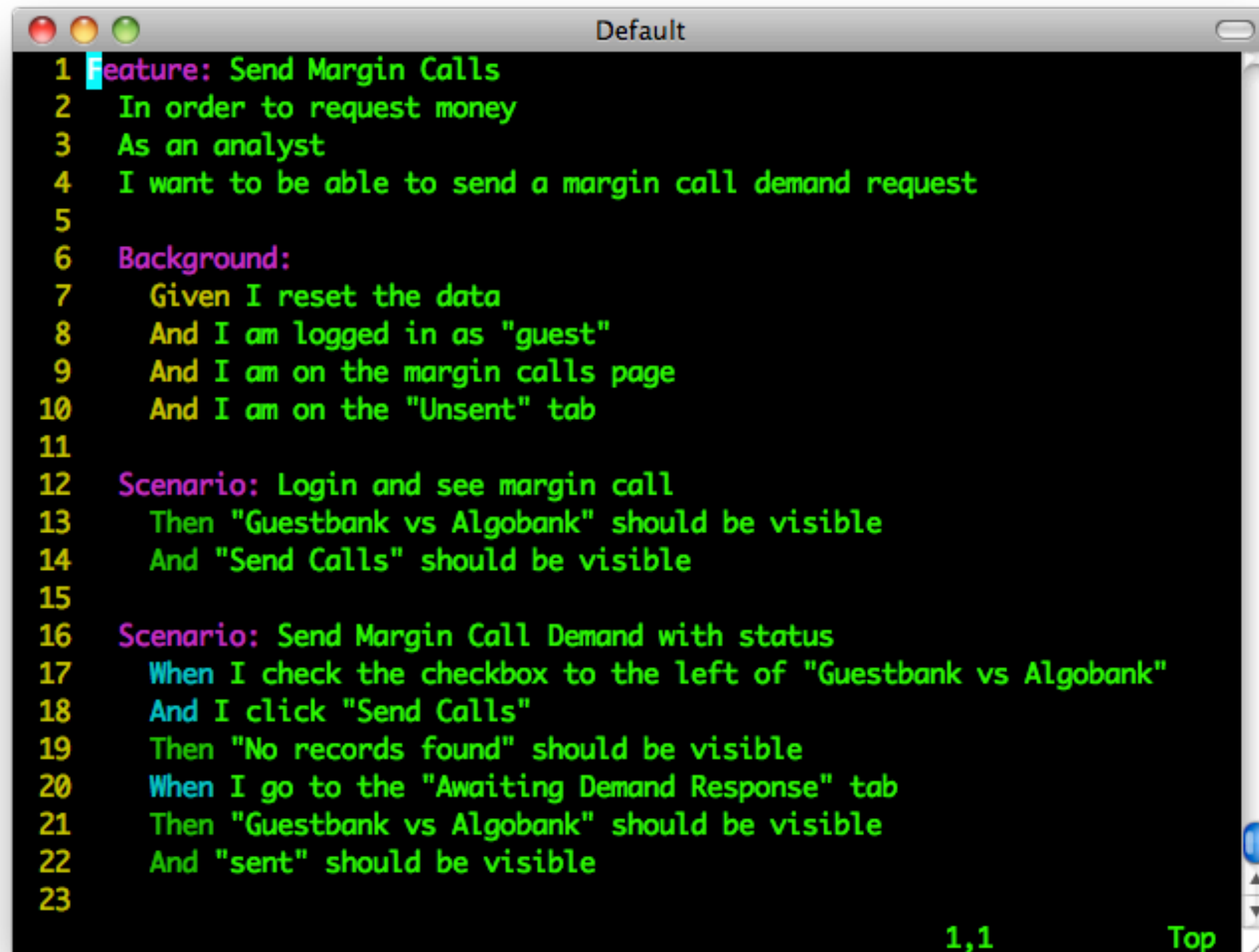
# Automation: QA / Testing

Reduced QA Resource Needs Initially (zero)  
Suite Provided Regression Testing: Reduced Errors  
Sped Up Development

- Cucumber
- RSpec



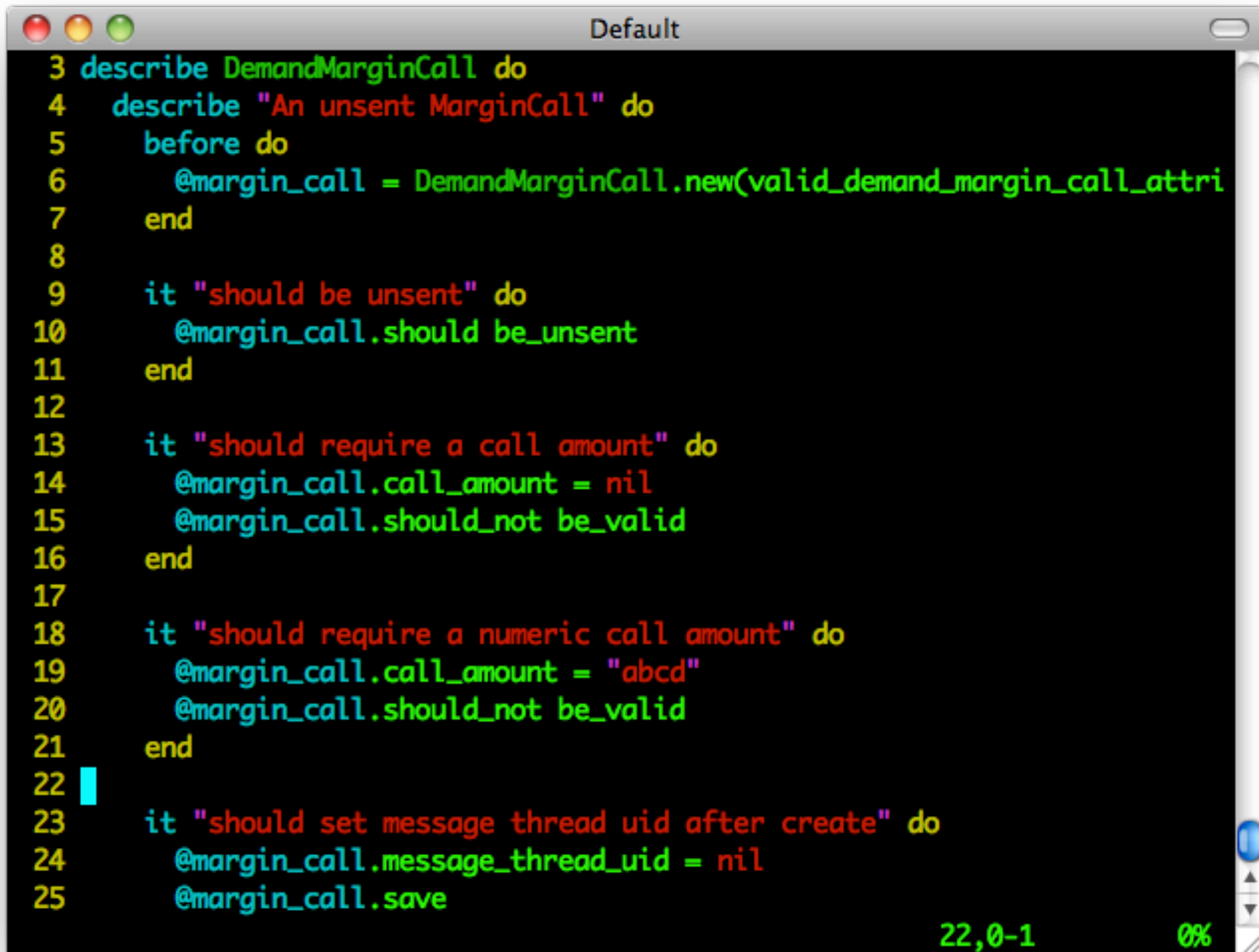
# Testing: Cucumber



```
1 Feature: Send Margin Calls
2   In order to request money
3   As an analyst
4   I want to be able to send a margin call demand request
5
6   Background:
7     Given I reset the data
8     And I am logged in as "guest"
9     And I am on the margin calls page
10    And I am on the "Unsent" tab
11
12   Scenario: Login and see margin call
13     Then "Guestbank vs Algobank" should be visible
14     And "Send Calls" should be visible
15
16   Scenario: Send Margin Call Demand with status
17     When I check the checkbox to the left of "Guestbank vs Algobank"
18     And I click "Send Calls"
19     Then "No records found" should be visible
20     When I go to the "Awaiting Demand Response" tab
21     Then "Guestbank vs Algobank" should be visible
22     And "sent" should be visible
23
```

1,1 Top

# Testing: RSpec



```
3 describe DemandMarginCall do
4   describe "An unsent MarginCall" do
5     before do
6       @margin_call = DemandMarginCall.new(valid_demand_margin_call_attri
7     end
8
9     it "should be unsent" do
10      @margin_call.should be_unsent
11    end
12
13    it "should require a call amount" do
14      @margin_call.call_amount = nil
15      @margin_call.should_not be_valid
16    end
17
18    it "should require a numeric call amount" do
19      @margin_call.call_amount = "abcd"
20      @margin_call.should_not be_valid
21    end
22
23    it "should set message thread uid after create" do
24      @margin_call.message_thread_uid = nil
25      @margin_call.save
```

22,0-1 0%



# What we did: Agile Process

- Pair Programming
    - Continuous Code Review
    - Skill Transfer Happens Quickly
    - Bus Factor == Team Size
    - Lost one of our best members in Feb: OMM\*
- \* OMM: Oh! My!....meh.







# What we did: JRuby

- Web Portal for the low end of the market
  - Ruby on Rails (JRuby), jQuery, YUI
  - Lots of Plugins, Gems, Libraries
  - Avoided many CSS and JS issues by incorporating js-lint and the w3-css-validator into our build process

# What we did: Clojure

- Immutability Strongly Encouraged
  - reduction of bugs caused by common errors
- FP: lots of small re-useable pieces, adapt quickly to changes
- JVM: It's the libraries!
- Concurrency is fantastically easy
- Live Image: easier introspection into running system (and sometimes modifications)

# What it did for us:AMQP

- Lucky Accident
  - JPMorgan, iMatrix, 2004~2006
  - Standards based Messaging (woot!)
  - Platform and Language Neural
  - Many (independent) Broker and Client Library Implementations

So, What Has all this  
Done For us?

# Demo Time



- First Public Demo: Oct 2009, NYC
- Live Demo of Full End to End Stack for Many Wall St Banks and the FED

# Automated Provisioning

- 2 Days Prior: Our Provider Experiences  
*Major Network and System Issues*
- 9am: Decision Made to procure  
alternate hosting
- 1pm: Full Stack Installed and Tested



# NOT LOC Metrics Again!

- Widely Believed that LOC/Dev is Roughly Constant
- Two Devs of Equal Skill/Experience, the one using a higher level language will be more productive
- See also: Mythical Man Month

# NOT LOC Metrics Again!

- sloccount
- For the first 9mo of the project:
  - 24kloc, 687loc per dev per month
  - 5.81 developer years
  - Our People, Tools & Process put us roughly 45% ahead of that estimate

So, uh,  
Everything is great  
right?

# Agile: Hiring is Hard

- Pair Programming *very difficult* environment, not for everyone
- Scaling the team is hard
- Personality and fit is vital (and frequently not good)

# Iterative Process

- Implement the minimum to provide the most business value
- You Continually Evolve your Architecture
- This is very uncomfortable to many
- Makes it harder to plan

# Functional Programming

- Clojure is new, it's changing
- FP is not mainstream (yet)
- Experienced Resources are Uncommon

# Conclusion

- You can leverage these technologies
- Focus on Skill Transfer
- Focus on Delivering Business Value

# Questions?



# Thank you!

- All of you for listening to us!
- Chariot Solutions for inviting us
- Bonnie Aumann, Trotter Cashion, David Kerkeslager for reviewing the presentation

# Picture Credits

- <http://www.flickr.com/photos/vizzzual-dot-com/2226095398/>
- <http://www.flickr.com/photos/epicharmus/2046126318/>
- <http://www.flickr.com/photos/avlxyz/389030408/>
- <http://www.flickr.com/photos/98063470@N00/326044514/>
- <http://www.flickr.com/photos/epicharmus/1481122250/>