# *Adapting Agile Practices to Fit Your Situation*

**Camille Bell**

**CamillesCareer@yahoo.com**

**Philly Emerging Tech 2011** **April 28, 2011**

# Audience

**Some Assumptions About You:**

- You have read agility books or attended training
- You manage, lead, coach or encourage the use of agile practices
- You, yourself, are agile
- But . . . *sometimes things could be going better*

# Approach

## What We're Going to Do:

- Look a a few common agile transformation issues
- Use my experience and that of some other agile coaches
- For each example issue
  - Examine the **What**, **Impact** and **Why**
  - Suggest potential **Mitigation**s (Things to Try)
- Provide a template to examine issues

## What We're Not Going to Do:

- Solve all Agile issues
- Provide all possible solutions for any issue
- Promise any Silver Bullets

# Issue & Mitigation: General Template

**What:**

- <general problem description>

**Impact:**

- <lesser and greater impacts>

**Why:**

- <likely causes>

**Mitigation** (things to try to solve problem)**:**

- <target each cause> or
- <some general options for multiple causes> or
- <both>

# Issue: Missing Customer

**What:**

- No on-site customer for XP
  - Scrum Product Owner is undesignated or unavailable

**Impact:**

- No direction, no feedback => WRONG PRODUCT

**Why:**

- On-site customers are more the exception than rule
- Not available during business hours
  - Truly overly busy - running the business, on market room floor
  - Physically unavailable (e.g. in battlefield, space, etc.)
  - Speculative customer doesn't yet exist
  - Customer doesn't see value in time spent

# Mitigation: Missing Customer

**Customer truly overly busy:**

- Shadow and observe customers

**Customer physically unavailable:**

- Use proxies who have worked in customer role

**Real customer doesn't yet exist:**

- Work with marketing or others close to potential customers
- Create customer focus groups

**Customer doesn't see value in time spent:**

- Quickly create mini app based on best guess of customer values (implement one or two small features)
- Demonstrate to customer in their space at their convenience
- Get feedback
- Repeat until customer becomes involved

# Issue: No Single Product Owner

**What:**

- Scrum expects a single Product Owner
- But you have many customers

**Impact:**

- No prioritization, wrong prioritization and/or import features entirely neglected => WRONG PRODUCT

**Why:**

- Complex apps have many types of users with different needs
- A single person seldom knows all user roles in detail
- A single person can't weigh competing needs outside his expertise

# Mitigation: No Single Product Owner

Option 1: **Dividing feature allotment by ROI**

- If marketing or sales can determine ROI by user type
  - Give each Product Owner a proportional share of feature development

Option 2: **Dividing feature allotment evenly**

- If you can't determine ROI or similar weighted value
  - Give each Product owner an equal share of feature development

# Mitigation: No Single Product Owner (continued)

Option 3: **Establish Kanban style feature input queue**

- Set a Work in Progress Limit
- Let each Product Owner submit a story to the queue
- Let POs horse trade among themselves for priority queue positioning

Option 4: **Establish a Chief Product Owner**

- Has no features of his own; only prioritizes other POs' features
- Could be dedicated agile coach

# Issue: User Stories are Too Big

**What:**

- User stories should be small enough that several can be completed during a single iteration
- But your customer's user stories take weeks to complete

**Impact:**

- Development cadence is very uneven
- Estimation is difficult and inaccurate
- Completion of any single user story is uncertain

=> WRONG PRODUCT and/or LATE PRODUCT

**Why:**

- Your user stories are complex, compound stories including many features, alternative flows, multi-part conditionals, tackle all levels at once or are one large general case

# Mitigation: User Stories are Too Big

**Split big stories apart into many small stories:**

- Spilt many featured stories into separate stories
- Spilt conditional stories with “and”, “or”, “then” or other connector words into separate stories.
- Split implied conditional stories into separate stories (e.g. “process all credit cards” becomes “process American Express”, “process Master Card”, “process “VISA”, etc.
- Split complex alternative flow stories into separate main flow (happy path) and multiple alternate flow stories
- Split collection stories into some first story with add on stories
- Split recursive general case stories first into a base case, recursive functionality becomes separate stories
- Implement the simplest stories first
- Refactor out commonalities between stories on implementation of later related secondary and tertiary stories

For more ideas see: Bill Wake’s 3 “20 Ways to Split Stories”, & Mike Cohen’s “User Stories Applied”

# Issue: User Stories are Too Vague

**What:**

- You are writing User Stories to avoid requirements bloat
- You even ask the customer questions as you go
- But when you demo, you discover you are way off the mark

**Impact:**

- Incorrect functionality, time wasted correcting

  => WRONG PRODUCT and/or LATE PRODUCT

**Why:**

- User Stories didn't provide enough detail to implement

# Mitigation: User Stories are Too Vague

**User Stories didn't provide enough detail to implement:**

- Cards aren't enough: Need Jeffries 3 Cs*
  - Card: the User Story itself
  - Conversation: talk, repeat what the customer said in different words, get feedback, ask clarifying questions for more details, challenge your assumptions
  - Confirmation: automated tests
- Automated tests demand precision
  - Cucumber tests are very good for high level BDD
  - If possible sit down with customer to as you write tests
  - Show the tests to customer
  - Drill down with Rspec or Shoulda
  - Get test data input and expected results from customer
- Run the tests and show results to the customer
  - Customer may think of something he forgot

# Customer Collaborates with Dev Team on Confirmation of Stories

**User Story: Vetting Sighting**

In order to show confirmation of a sighting,
As a US-CERT analyst,
I want to mark a sighting a vetted

- XYZ sighting is viewed by Charley US-CERT analyst
- Charley marks sighting as vetted
- Tracy from Treasury searches for sightings vetted today
- Tracy sees XYZ as a confirmed sighting

Ref: Ron Jeffries 3 Cs of User Stories

CamillesCareer@yahoo.com

# User Stories Confirmed Through Automated Tests During Iteration

**User Story: Vetting Sighting**

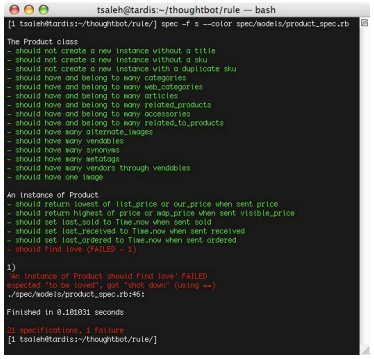
In order to show confirmation of a sighting,
As a US-CERT analyst,
I want to mark a sighting a vetted

- XYZ sighting is viewed by Charley US-CERT analyst
- Charley marks sighting as vetted
- Tracy from Treasury searches for sighting vetted today
- Tracy sees XYZ as a confirmed sighting

```
describe "Imports TEWI sightings" do
  describe "CERT analyst finds a recent TEWI sighting" do
    describe "Vets a sighting" do
      describe " Trusted partner finds un-vetted sighting" do
        describe "Trusted partner finds vetted sighting" do

          context "a vetted sighting should be clearly vetted" do
            before(:each) do
              new_sighting.build_from_TEWI()
              cert_user.new('default_cert')
              new_sighting.vet(cert_user)
              new_partner.new('default_partner')
            end
            it "should have a tag of 'vetted'" do
              new_sighting.tag.should_include == 'vetted'
            end

            it "should be viewable by partner" do
              new_sighting.accesable_by(new_partner).should == true
            end
        end
```



# Issue: High Bug Count

**What:**

- You have a continuing stream of bugs
- Bug count isn't going down and may be going up
- Some previously fixed bugs show up again

**Impact:**

- Unstable product => UNHAPPY USERS, BAD PRESS, PAYING USERS DEFECT TO OTHER VENDORS

**Why:**

- Lack of understanding of end user
- High technical debt

# Mitigation: High Bug Count

**Lack of understanding of end user:**

- Common problem in early product life
  - Release Beta versions to prevent
  - Write **RED** test, to prevent bug recurrence (TDD practice) before fixing bugs
  - Help Product Owner to get closer to customer
  - Follow standard usability guidelines
- If usability problems persist, you may be building the wrong product

**High technical debt:**

- Common problem in mid-late product life
  - Write **RED** test, to prevent bug recurrence (TDD practice) before fixing bugs
  - Set aside a little time ever iteration for **Refactoring**
  - Target code to refactor each iteration (metric_fu gem can help)
  - Write tests as safety net before refactoring code
  - Write all new code following BDD and TDD practices

# Issue: Customers/POs Never Choose Technical Stories for Next Iteration/Sprint

**What:**

- User Stories are written “As a <user>, I want <feature>, so that <value>”
- Some of the stories are have a strong technical focus
- The technical stories are important
- But the customer never selects technical stories

**Impact:**

- No technical stories chosen => INCREASING TECHNICAL DEBT
- Increasing Technical Debt => SLOWER NEW FEATURE DEVELOPMENT
- Increasing Technical Debt => INCREASING BUG COUNT
- Increasing Bug Count => USER DEFECTION TO OTHER VENDORS

**Why:**

- Customer prioritizes by business value

# Mitigation: Customers/POs Never Choose Technical Stories for Next Iteration/Sprint

**Customer prioritizes by business value:**

- The customer is doing the right thing based on the information available
- All user stories need to clearly state customer value
- The format of the user story is part of the problem. Even a technical user story must include:
  - Business value not technical value
  - Some business user not a technical user
- Instead use this format:

"In order to <business value>,

As a < business user>,

I want <feature>"

# Mitigation: Customers/POs Never Choose Technical Stories for Next Iteration/Sprint

| User Story: Refactor Ball of Mud |
| --- |
| As a programmer,<br>I want to refactor the ”big ball of mud" code,<br>So that the code is simpler |

Value last
Technical user
No clear business value

Value first
Business user
Clear business value

| User Story: Faster Features / Cheaper Maintenance |
| --- |
| In order to deliver new features faster and lower maintenance costs,<br>As a program manager <or other applicable business user>,<br>I want the the ”big ball of mud" code refactored |

# Issue: Customers/POs Wont Take the Time to Prioritize the Backlog

**What:**

- There are dozens or possibly hundreds stories in the product backlog
- Scrum says the Product Owner should prioritize all of them 1, 2 … etc.
- That isn't happening

**Impact:**

- Worst case: No prioritization, wrong prioritization

  => IMPORTANT FEATURES NEGLECTED while

  => WASTING TIME  and MONEY 

**Why:**

- The customer has limited time
- The customer doesn't see the value in prioritizing everything in numerical order

# Mitigation: Customers/POs Wont Take the Time to Prioritize the Backlog

**If time is limited and there are too many stories,**

**you don't need to prioritize them all:**

- Make the best use of the time the customer does have
- The customer may be right
  - If you have hundreds of un-prioritized user stories, prioritizing them all can be a waste of time

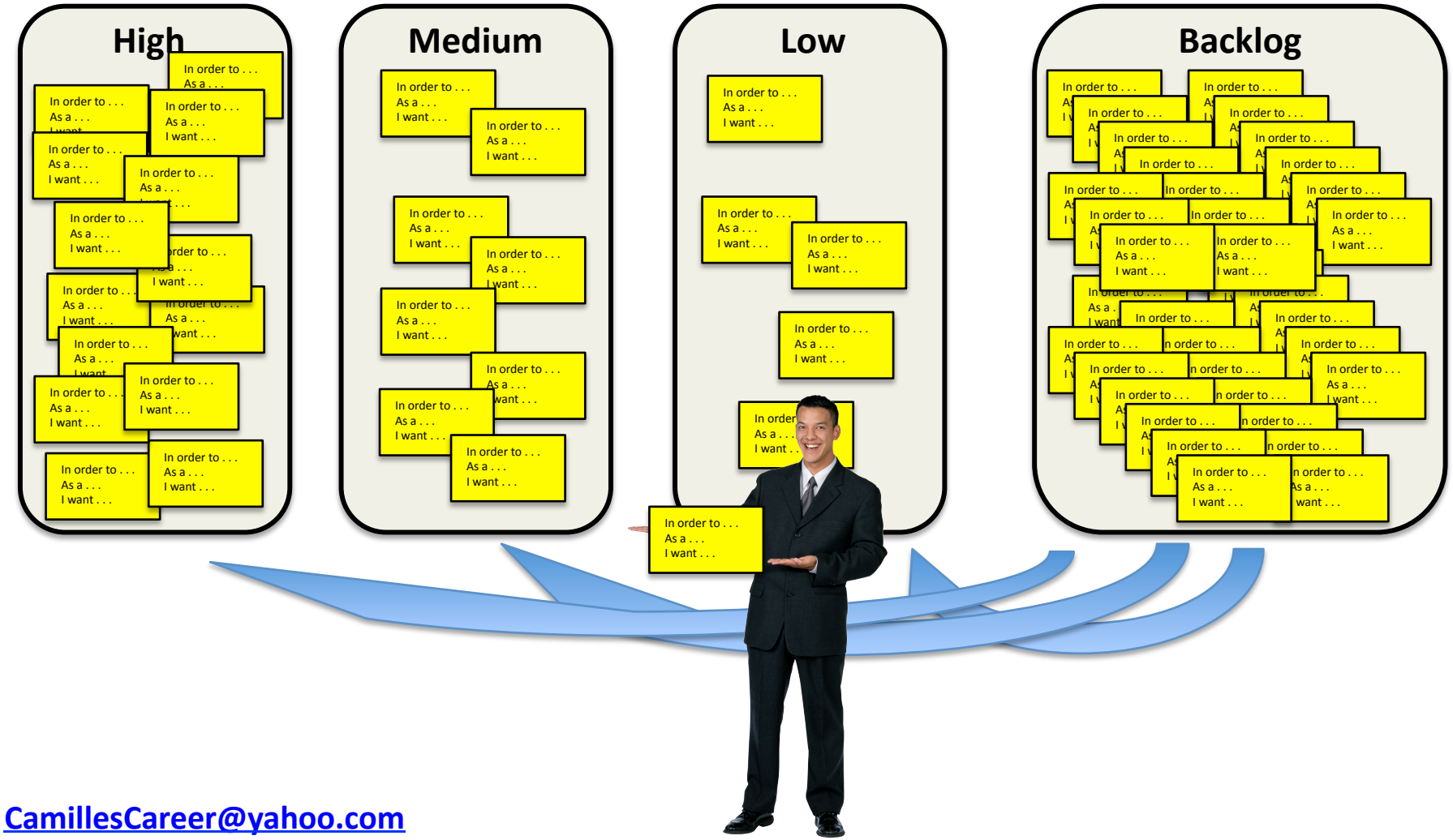
**Use a progressive triage to find top stories**

# Mitigation: Customers/POs Wont Take the Time to Prioritize the Backlog

**Progressive triage to find top stories:**

1. Get 10 or 12 top user stories in priority order (or skip to step 2)
   - Ask the customer for the top 10 or 12 user stories (this may not work)
   - Prioritize those 1, 2, 3 . . .
   - Implement those stories and ignore the rest until down to the 3 or 4 remaining
   - As customer to select and prioritize few more top stories until you have 10 or 12
2. Have customer divide stories into High, Medium & Low
   - Works best with movable cards or sticky notes
   - There will be a disproportionate number of highs (this is normal)
   - Remove all the Medium and Low stories
3. Repeat step 2 until down to 10 or 12 user stories
4. Prioritize those 1, 2, 3 . . .

# Mitigation: Customers/POs Wont Take the Time to Prioritize the Backlog



# Issue: Velocity is Continually Slowing Down

**What:**

- New user stories with the same complexity / story points take longer than similar older user stories

**Impact:**

- Slower feature development

  => INCREASED DEVELOPMENT COST per FEATURE

  and

  FEWER FEATURES or LATE PRODUCT

**Why:**

- Change in staffing
- Increasing technical debt
- Task switching

# Mitigation: Velocity is Slowing Down

**Change in staffing:**

- Staff turnover slows velocity as new members spin up
- Use pair programming, etc. to spin up faster and lessen Truck Factor
- Work to make your team a place people want to stay

**Increasing technical debt:**

- Use "High technical debt" mitigation from "High Bug Count"

**Task switching:**

- Set strict Work in Progress Limits to prevent task switching
  - Recommended WIP limit
    =( # Developers / # Developers per Story) + 1 ( if stories get blocked)
  - e.g. XP Team of 10 Developers: 10/2 + 1 = WIP of 6

# Issue: Too Much Work in Iteration Despite Story Points Matching Prior Velocity

**What:**

- The estimated story point for an iteration are identical to prior iterations
- But . . . completing committed stories is hit and miss
- And . . . the team when the team misses, the team misses by a lot

**Impact:**

- Missed commitments => LOST TRUST, LATE PRODUCT

**Why:**

- Underestimated stories
- Some stories “too small” to estimate
- Staff availability varies
- Too many fire fights
- Change in staffing
- High technical debt
- Too much task switching

# Mitigation: Too Much Work in Iteration

**Underestimated stories:**

- If political pressure is causing team to lower estimates, stand firm, tell the truth and provide options to those pressuring the team
  - Sometimes it is very hard to to management or business how much work something is, but it will be worse when your team doesn't deliver
  - Suggest to management implementing best value split first

**Some stories "too small" to estimate:**

- Team has a number of "0" point stories
  - e.g. "fix spelling error", "change font size", "change background color", etc.
- Individually they aren't worth estimating, but together they add up
  - Lump a similar group together, until worth 1, 2 or 3 story points

**Staff availability varies:**

- Some variability is normal, lower commitments accordingly
  - Remember to account for vacations, holidays, training and other predictable staff absences (ask every iteration, don't assume you know)

# Mitigation: Too Much Work in Iteration

**Too many fire fights:**

- Avoid fire fights if you can, but often you can't
- Consider using Kanban with multiple input queues to handle, Expedite, Bug Fixes, New Development and other Levels of Service
- If "surprise" work is seasonal or predictable, lower other commitments during those times

**Change in staffing:**

- Use "Staff is changing" mitigation from "Velocity is Slowing Down"

**Increasing technical debt:**

- Use "High technical debt" mitigation from "High Bug Count"

**Task switching:**

- Use "Task switching" mitigation from "Velocity is Slowing Down"

# Issue: Standup Meeting Take Forever

**What:**

- Standup meetings are supposed to be short; usually 15 minutes max
- Your standup meeting take much, much longer

**Impact:**

- Standup is dreaded, poorly attended, not held daily
- Attendees space out missing important information

=> STANDUP FAILS PURPOSE, WASTES TIME

**Why:**

- Team members focus and time management is poor
- Management and others hijack standup
- Team is too large

# Mitigation: Standup Meeting Takes Forever

**Team members focus and time management is poor:**

- Remove all chairs and force a real standup
- Hold meeting daily
- Time-box meeting and each speaker with timers
- Use talking stick
- Until team gets really focused limit to only the 3 Questions
- Hold break out sessions afterwards (for those interested) on topics you had to halt discussion during standup

**Management and others hijack standup:**

- First ensure team member focus (above)
- Remind others that standup is for the technical team
- Enforce Scrum-style “pig” and “chicken” roles
- Deal with the politics

# Mitigation: Standup Meeting Takes Forever

**Team is too large:**

Option 1: **Scrum Solution**

- Divide technical team into multiple teams
  - Hold separate Scrum/Standup for each team
  - Hold Scrum of Scrums to roll up progress & impediments

Option 2: **Scrumban Solution**

- If applicable, divide technical team into multiple teams
  - Hold joint Kanban board session of entire large team
  - If needed, hold separate small team stand ups afterwards

# Issue: Standup Meetings Don't Discuss Real Problems

**What:**

- Standup meetings are short and follow Scrum rules
- But real issues aren't discussed

**Impact:**

- Development is slowed by late discovered impediments & risks

  => LATE PRODUCT

**Why:**

- Team member embarrassment, lack of awareness, etc.
- Team members don't believe underlying problem are solvable
- Facilitator lacks insight into hidden road blocks
- Team members don't feel comfortable calling each other on behavior

# Mitigation: Standup Meetings Don't Discuss Real Problems

**Team members and facilitator need non-threatening practice :**

- Read up on Bill Wake's "Scrum from Hell" exercise
  - http://xp123.com/articles/scrum-from-hell/
- If needed invent a "neutral example app" for exercise
  - e.g. resume site, travel site, etc.
  - Don't build anything, just pretend
- Choose misbehavior cards that highlight teams problems
  - e.g. hidden impediment, not answering 3 questions, etc.
  - Add a few good behavior cards to deck
  - Add unique team misbehavior cards if needed
- Run "Scrum from Hell" as a game (about 15 minutes)
- Reveal cards and talk about experience
- Repeat periodically, changing Scrum Master and team roles until everyone has experience with different roles and cards

# Issue: Management Doesn't Value Removing Impediments Quickly

**What:**

- Your team tells management about the impediments blocking progress
- But . . . little is done and what is done happens slowly

**Impact:**

- Impediments are removed slowly or not at all

  => TEAM PRODUCTIVITY IS A FRACTION OF WHAT IT COULD BE

**Why:**

- Management is new to their agile role of road block remover
- Management unaware of the impact of impediments and blockers
- Middle management doesn’t have the power to remove blockers
- Removing some impediments do take a long time in your organization

# Mitigation: Management Doesn't Value Removing Impediments Quickly

**Management is new to their agile role of road block remover:**

- Educate your management team on Agility
  - If possible get management in Scrum Master training
  - Give open brown bags and other “byte sized” training
  - Offer to give short agile presentations at management retreats
  - Speak at PMI meetings (Scrum especially has become very popular)
- Coach your managers

**Management unaware of the impact of impediments and blockers:**

- Use burning visibility
  - Post Impediments Lists – with names when needed
  - Use Kanban boards to highlight blocks
- Keep metrics on wait time
- Show impact in Time  and Money 

# Mitigation: Management Doesn't Value Removing Impediments Quickly

**Middle management doesn't have the power to remove blockers:**

- Gain power for manager
  - Determine what blocks your manager from having the power he needs
  - Examine official (org chart) and un-official (buddies network)
  - Brainstorm with manager to determine strategy to gain needed power
- Go around the system
  - "It's easier to ask forgiveness than to get permission" – Admiral Hopper
  - But you have to be right (and not illegal)

**Removing some impediments do take a long time in your organization:**

- Problem is most common in large organizations with tiers of competing managers, too many checks and no real sense of balance
- Use all mitigations from "So Many Hoops" (next), that apply

# Issue: So many hoops to jump through that it takes forever to get anything done

**What:**

- Your company has many departmental teams which you depend on and which depend upon one another for equipment and services
- Each has lengthy and complex approval processes
- Each has overworked staffs and long wait queues

**Impact:**

- Extremely poor overall efficiency => LATE PRODUCT

**Why:**

- Each sub group is attempting to locally optimize itself to 100% efficiency
- Excessive local optimization harms global optimization
- No charts or metrics alert the CIO of the magnitude and impact of queuing delays

# Mitigation: So many hoops . . .

**Each sub group is attempting to locally 100% optimize itself:**

**Excessive local optimization harms global optimization:**

- Top level management (e.g. CIO, etc.) is tracking the wrong things and providing the wrong incentives, until CIO becomes aware, little will change

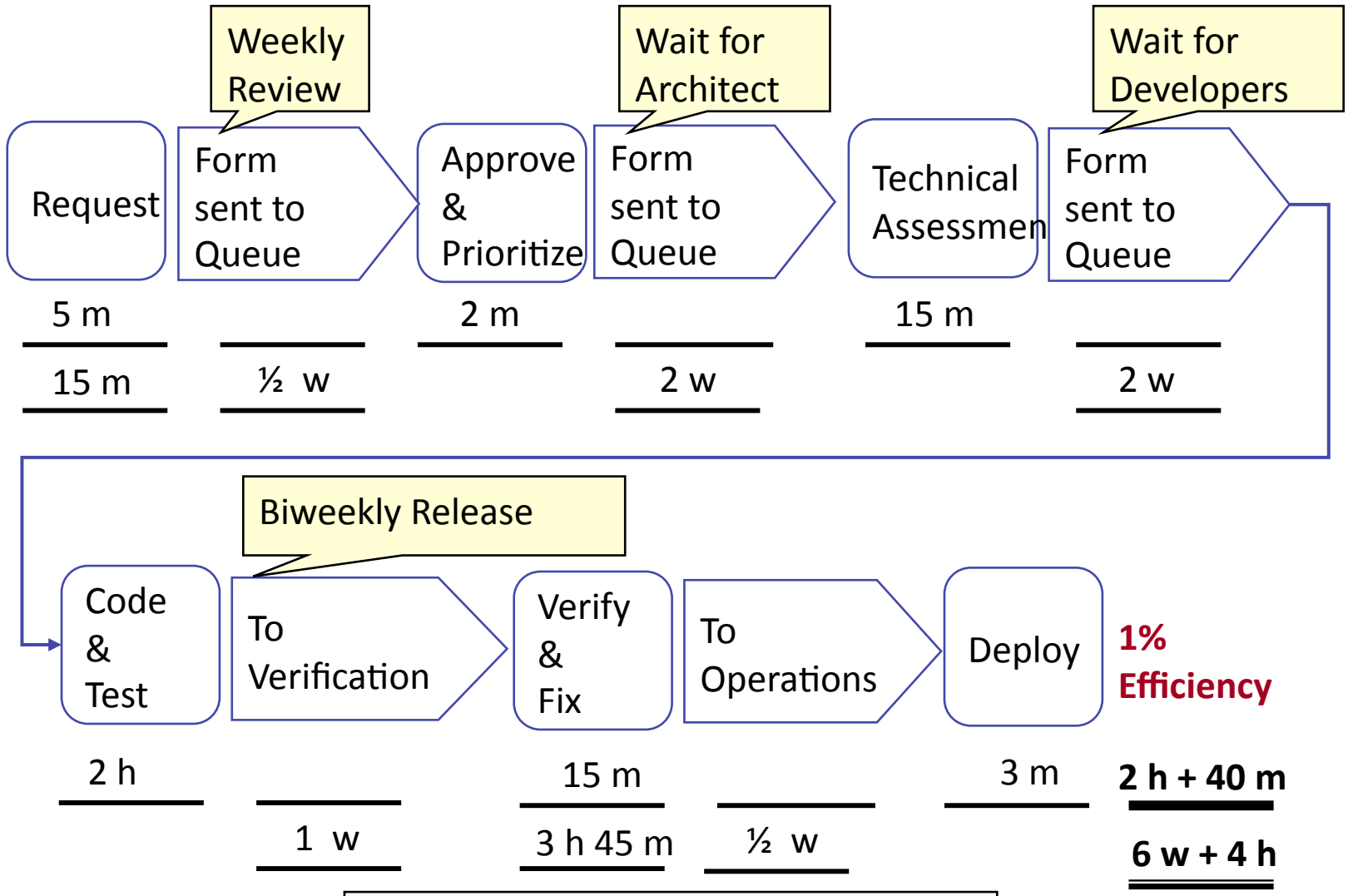
**No charts or metrics alert the CIO of the magnitude and impact of queuing delays:**

- Provide metaphors that middle and top management relate too
- Educate management on Lean, Kanban, Push vs. Pull & Value Stream
- Track wait times, create Value Stream Maps of worst problems and publish findings within company

# Use Relatable Metaphor

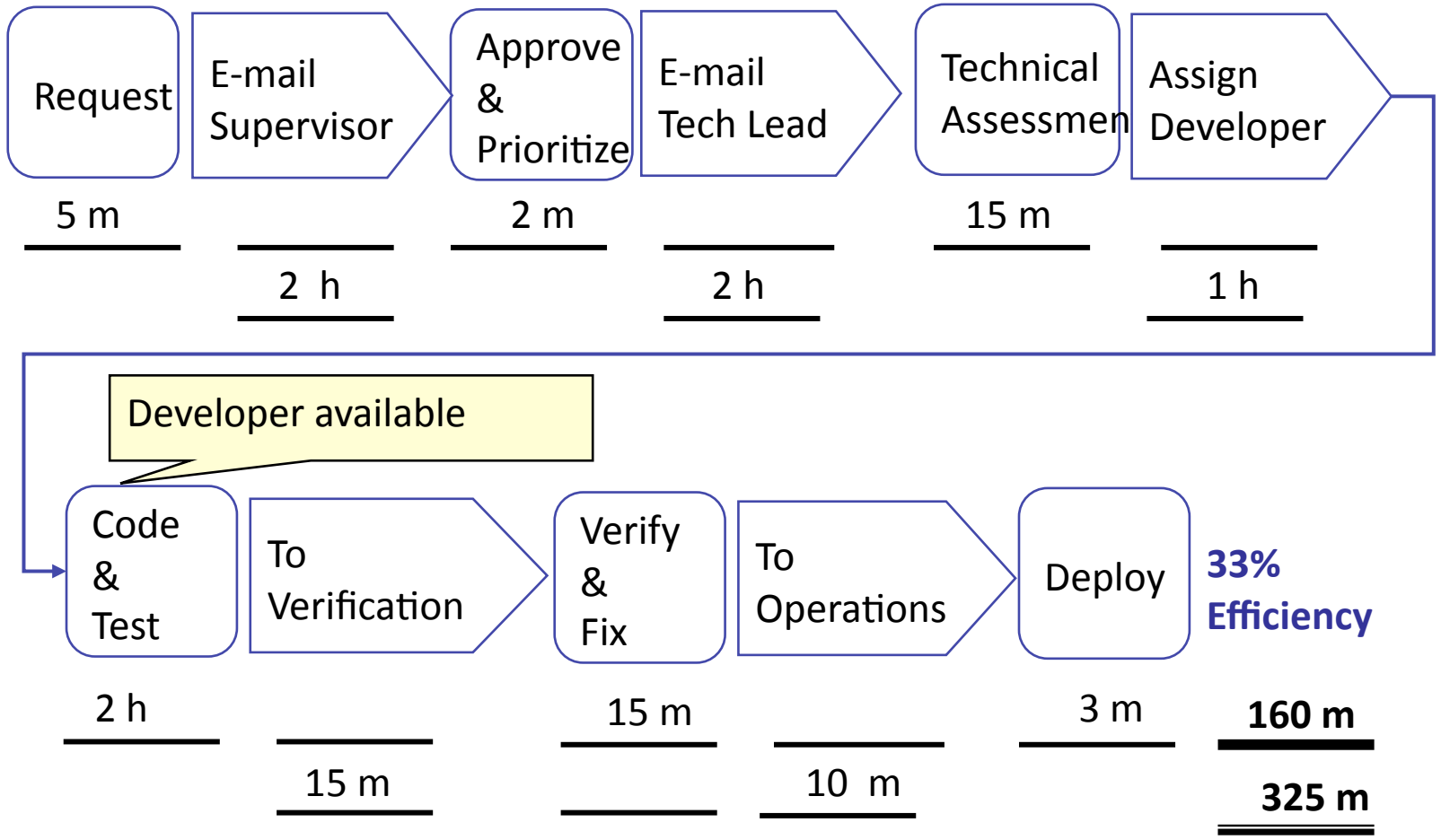
- Imagine a bridge or road trying to carry cars on 100% of its surface (any rush hour in Washington, DC)
  - More cars push to get on the bridge than it can possibly handle
  - Cars back up for miles
  - The rate of cars crossing the bridge is very low; commutes are horrible
  - If the traffic limited itself to capacity of the road or bridge the rate of cars crossing would vastly increase and commutes improve

# Using a Value Stream Map
# Big Delays are Reducible




Ref: Mary and Tom Poppendieck on Lean

# Efficiency Radically Improved by Shortening Wait Queues




CamillesCareer@yahoo.com

Ref: Mary and Tom Poppendieck on Lean

# *Camille Bell*

Agile Consulting
Agile Boot Camps
Agile Training
Updated Slides

or just to chat about things agile

**CamillesCareer@yahoo.com**

**301 424-3729**