#BugsZero

eliminating bugs by not creating them in the first place

Arlo Belshee

http://bit.ly/PromiscuousPairingPdf

http://bit.ly/AgileEngineeringFluency

http://bit.ly/NamingIsAProcess

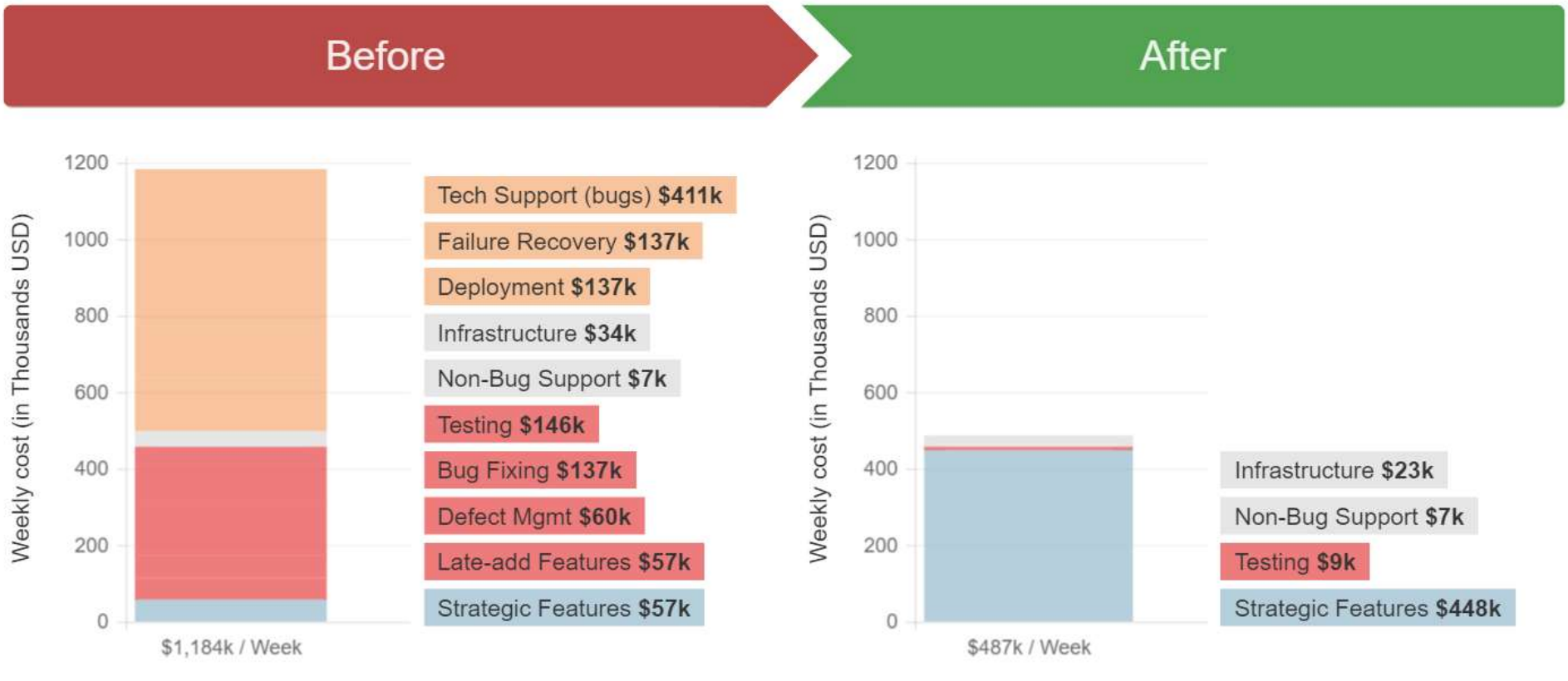
http://bit.ly/BugsZeroSlides (this talk)

Team Craftsman, Legacy Code Mender, and Rabblerouser

Tableau & Innovating Teams

@arlobelshee, github/arlobelshee, http://arlobelshee.com/

The #BugsZero Story: Why We Think We Care



Why we Actually Care

Testers feel like

Product owners feel like


© 2007 GCMGA Inc., All Rights Reserved
http://aggie-horticulture.tamu.edu/galveston

Developers feel like

The architecture feels like

Operations feels like

Sales has to pretend it is

But sales knows it actually is

Customers feel

And execs feel


R.I.P.
OUR
FUTURE
STOP
ing
the Price
of our DREAM

But I Don’t Have Technical Debt

Good design

+

30 years of
careful changes

Quick Poll: How Many Bugs?

- Function named DoContentHit
- In service for 30+ years
- 26,000 lines
- Uses undocumented system calls
- Uses gotos liberally
- Monitors keyboard and mouse

- Need to add support for touch and pen

Quick Poll: How Many Bugs?

- Function named DoContentHit
- In service for 33+ years
- 18,000 lines, x2
- Uses undocumented system calls
- Uses gotos liberally, including jumps between functions
- Monitors keyboard, mouse, touch and pen

- Need to port to new OS, with different input devices

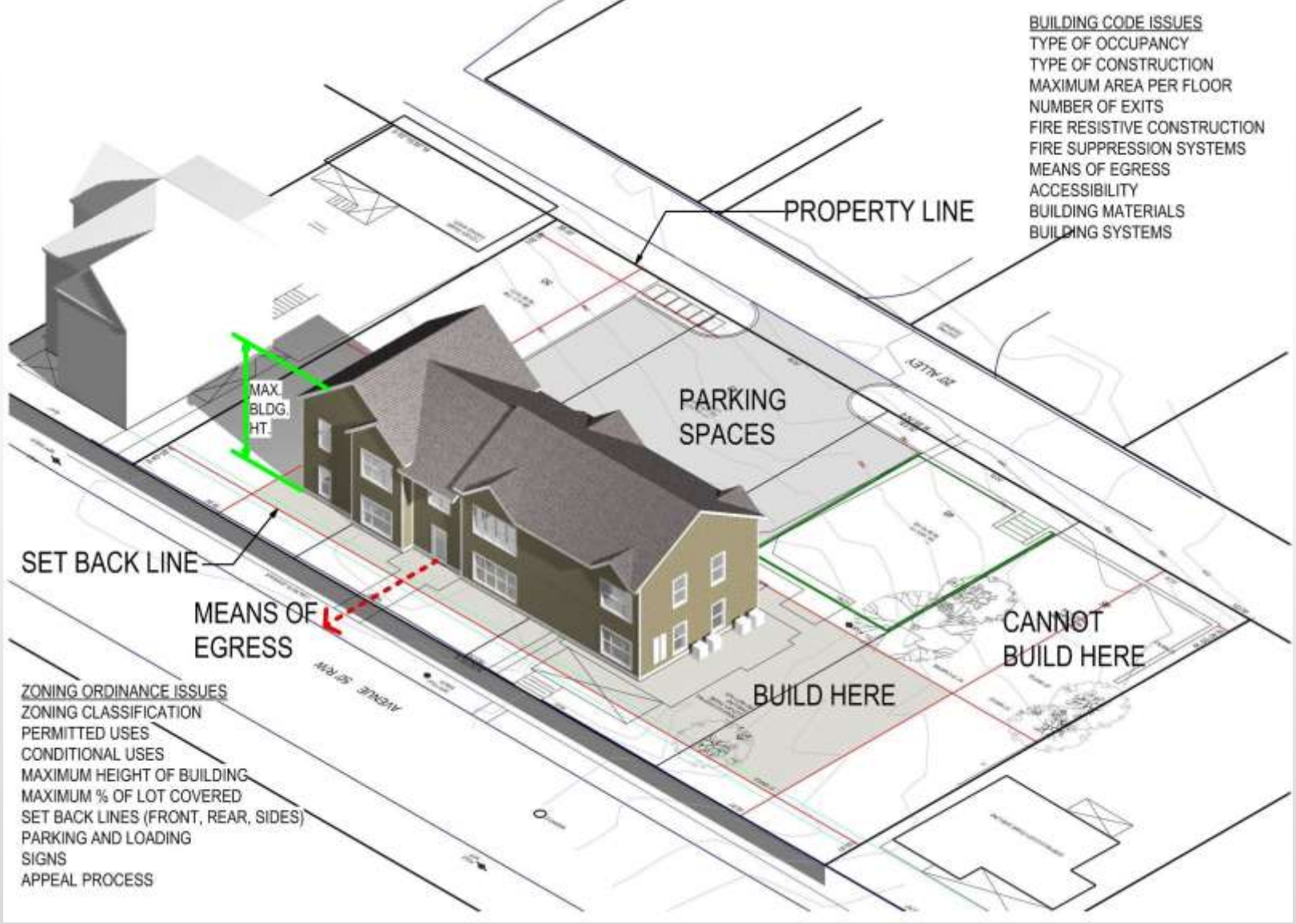
Bugs Annoy Me
I Decided to Stop Having Them

Bugs are Exciting

Like Kitchen Fires

Fires Have Causes

Caused by a cow?



1,000,000 Fires Prevented

When people build a building, are they thinking about fire prevention?

How Do We Prevent Bugs?

“To take an action for a fixed period of time, such that the probability of some undesirable outcome is permanently reduced.”

Arlo

Definition of “Prevent”

Action
+ Context
=========
Outcome

Action
+ Context
+ Spread
=========
Outcome

“Anything that would **frustrate, confuse, or disappoint** some human, and is **potentially visible** to any human other than the one who created it.”

Arlo

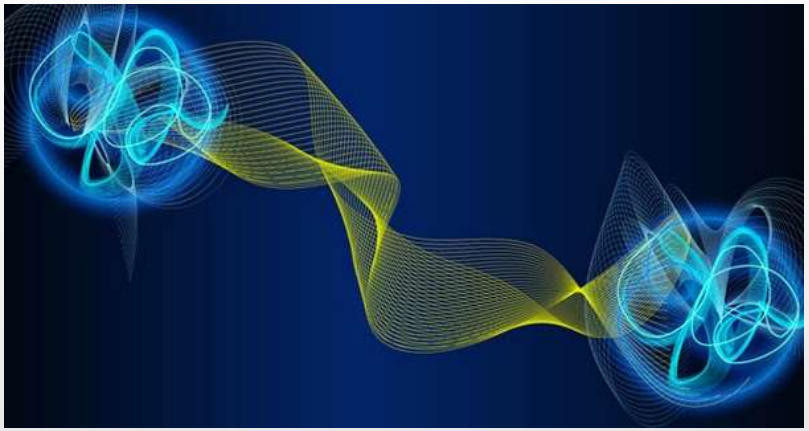
Definition of “Bug”

Actions that Can Cause Bugs

- Changing what a human receives
  - **Coding**
  - Designing
- Changing what a human expects
  - Selling
  - Gaining insight about customer intent / need
  - Marketing
  - UI / UX
  - The world changing

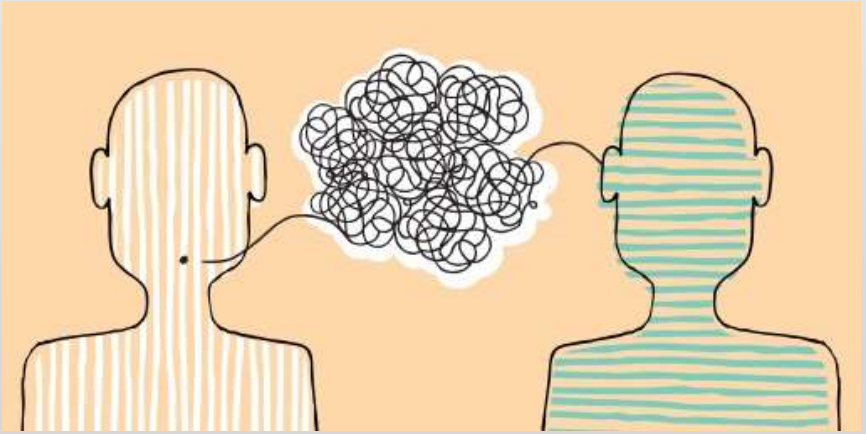
Contexts that Turn Those Actions Into Bugs

Unreadable code



Context sensitivity

Stuff developers don’t know



Miscommunications between customer and developer

Situations that Spread Bugginess

Hurrying

Accepting Errors

Repeating Mistakes

High Risk of Change

People Notice Little

Habit Drives Most Actions

Bugs Don’t Come From What You Do

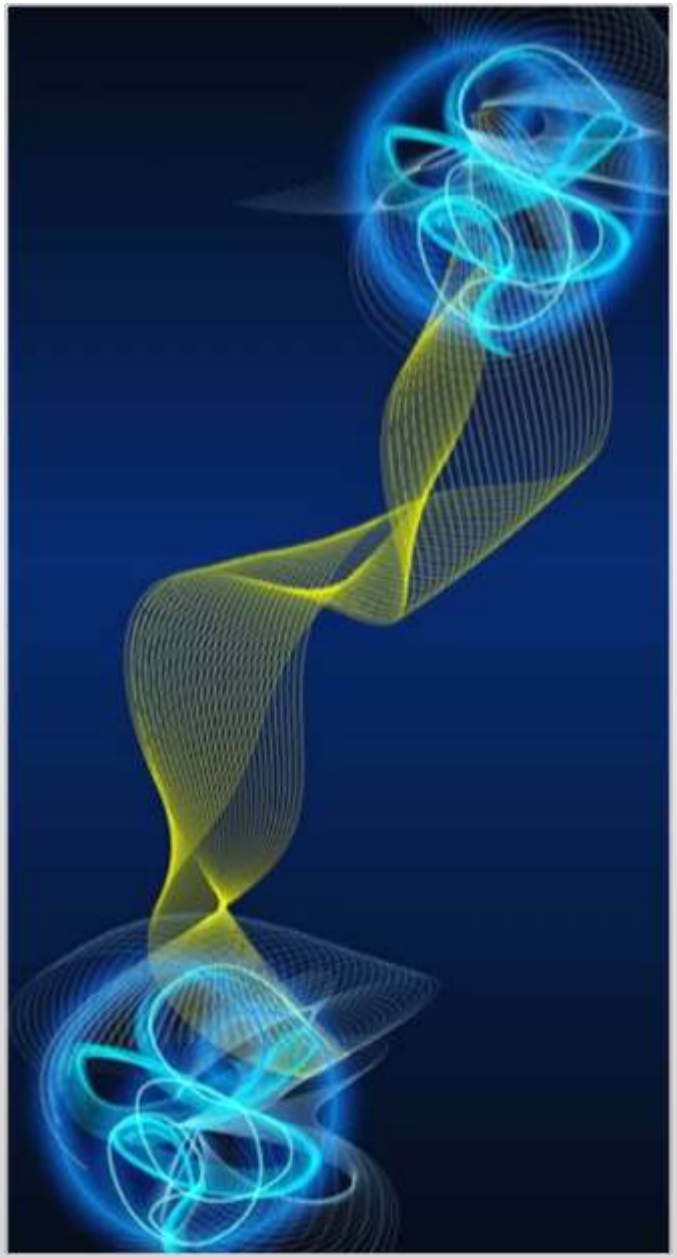
How do we change the habits that set our results?

Unreadable Code

- Read by Refactoring Habits
- Insight Loop
- Core 6 Refactorings
  - Rename to store insights
  - Extract Method to divide operations
  - Intro Variable / Param / Field to divide data
  - Inline to bring things together

Context Sensitivity

- Dependency Elimination Principle
- Dependency-Breaking Patterns
- TDD alone doesn’t help much
- Test by Refactoring Habits
  - Test as Spec
  - Acceptance Microtests
- Design by Refactoring Habits
  - Replace Supplier with Supplies
  - Breaking up God Classes iteratively
  - Follow fields to find multiple responsibilities



What Developers Don’t Know

- Pairing, Tripling, or Mobbing
  - Especially across roles
- Micro-habits & Mind-shifts
- Customer visits
- Listen to the code

Miscommunication on Path from Customer to Developer

- Shorten the path
- Customer telemetry
- Pair with technical support, on both coding and support cases

Hurrying

- Prevent the obvious bugs with common causes
- This is failure demand; it goes away when the failures go away

Accepting Errors

- Count “days since last accident”
- Eliminate triage
- All bugs are more important than remaining features
- Change incentives for PO

Repeating Mistakes

- Immediate stop the line
- Immediate 5-whys
- 15% solutions for each contributing factor, as tasks on the board
- Priority:
  1. Mitigate
  2. Prevent this class of bugs
  3. Fix the bug
  4. Anything other than this bug

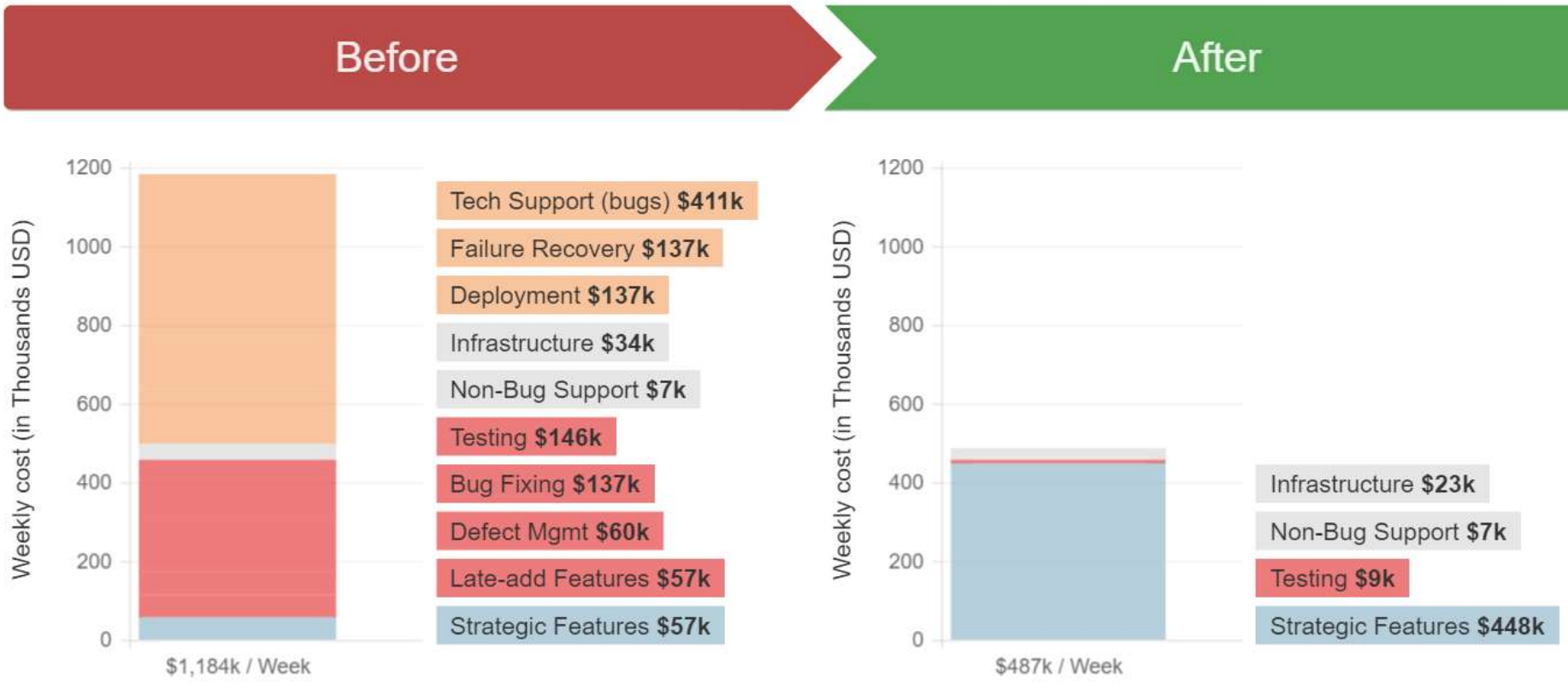
High Risk of Change

- Disciplined Refactoring
- Build / Measure / Learn
  - For product
  - For team capability
- Roll back non-successful experiments
- Simple design beats extensible design

1. Fix the Environment that is causing errors

2. Learn from each bug to prevent more

DoContentHit (and its Application), 2 Subsequent Releases



Thank You

http://bit.ly/PromiscuousPairingPdf

http://bit.ly/AgileEngineeringFluency

http://bit.ly/NamingIsAProcess

http://bit.ly/BugsZeroSlides (this talk)

@arlobelshee, github/arlobelshee, http://arlobelshee.com/