



# **Practical Process Improvement**

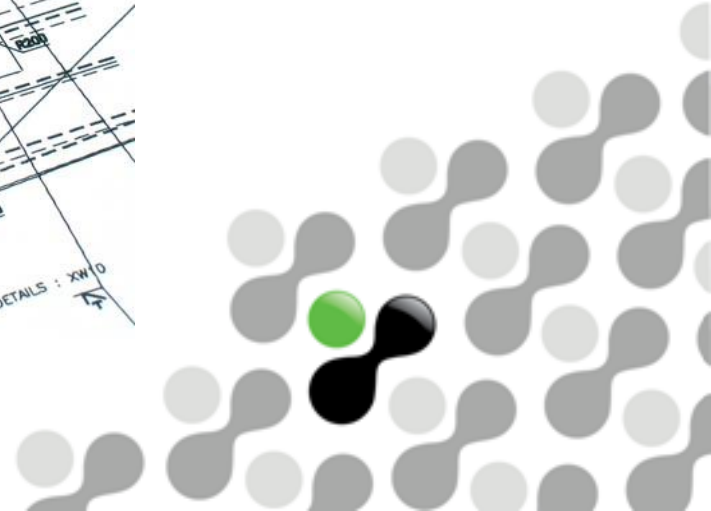
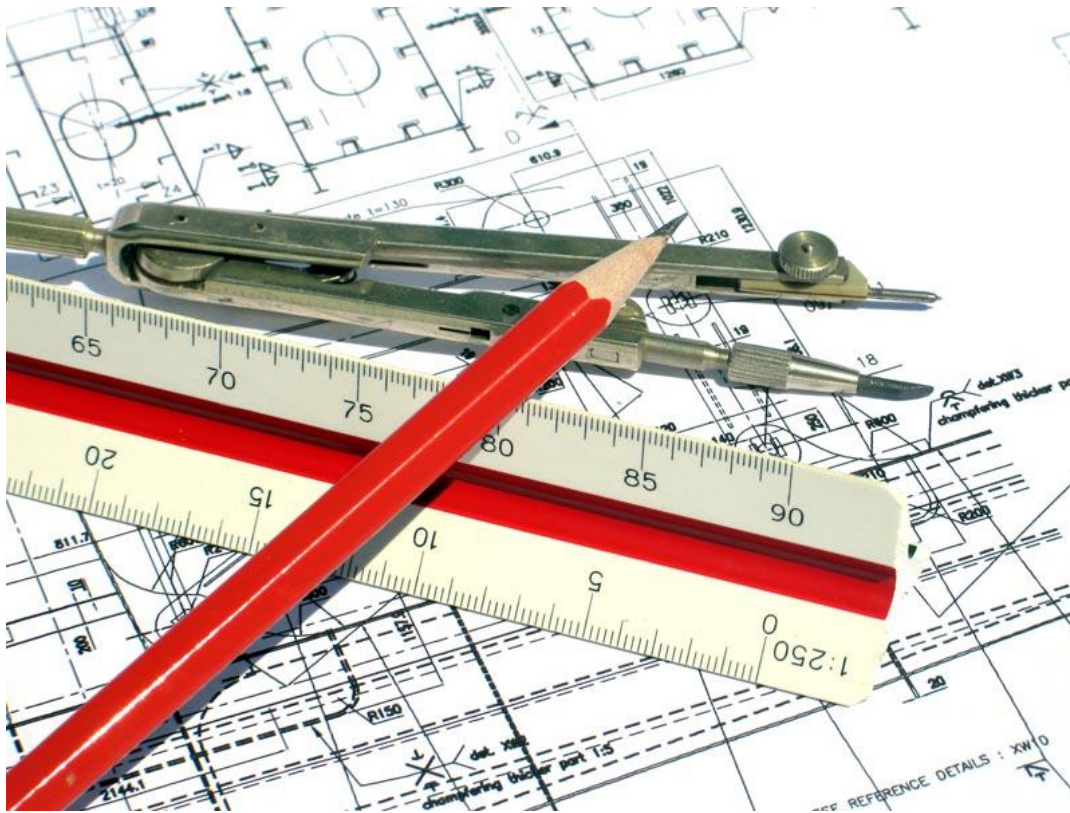
*Or, "How do I use those metrics?"*

# Approaches to Process Improvement



# Intelligent Change

- You can't change what you don't understand



# Missteps Will Occur



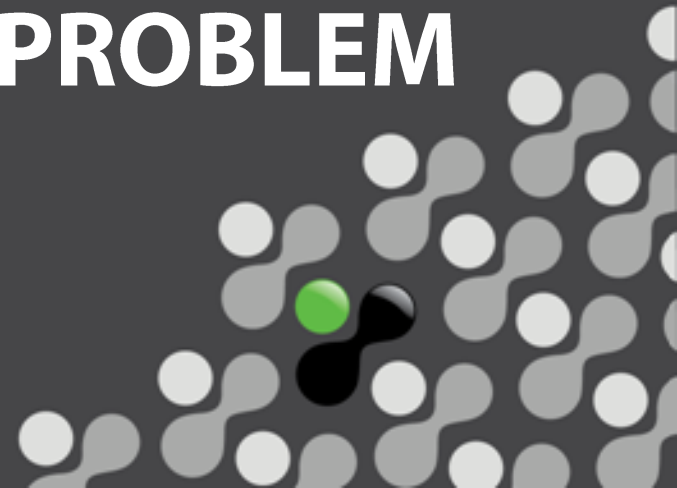
# Now What?

- Start small
  - Pick one or two changes at a time
  - Try to make sure they are isolated changes
- Spend a week or two getting your baseline – more data is better (to a point)
- Get team buy-in
  - Dictating to the team is a guaranteed failure



Reporting & Fixing Bugs

# BREAKING DOWN THE PROBLEM

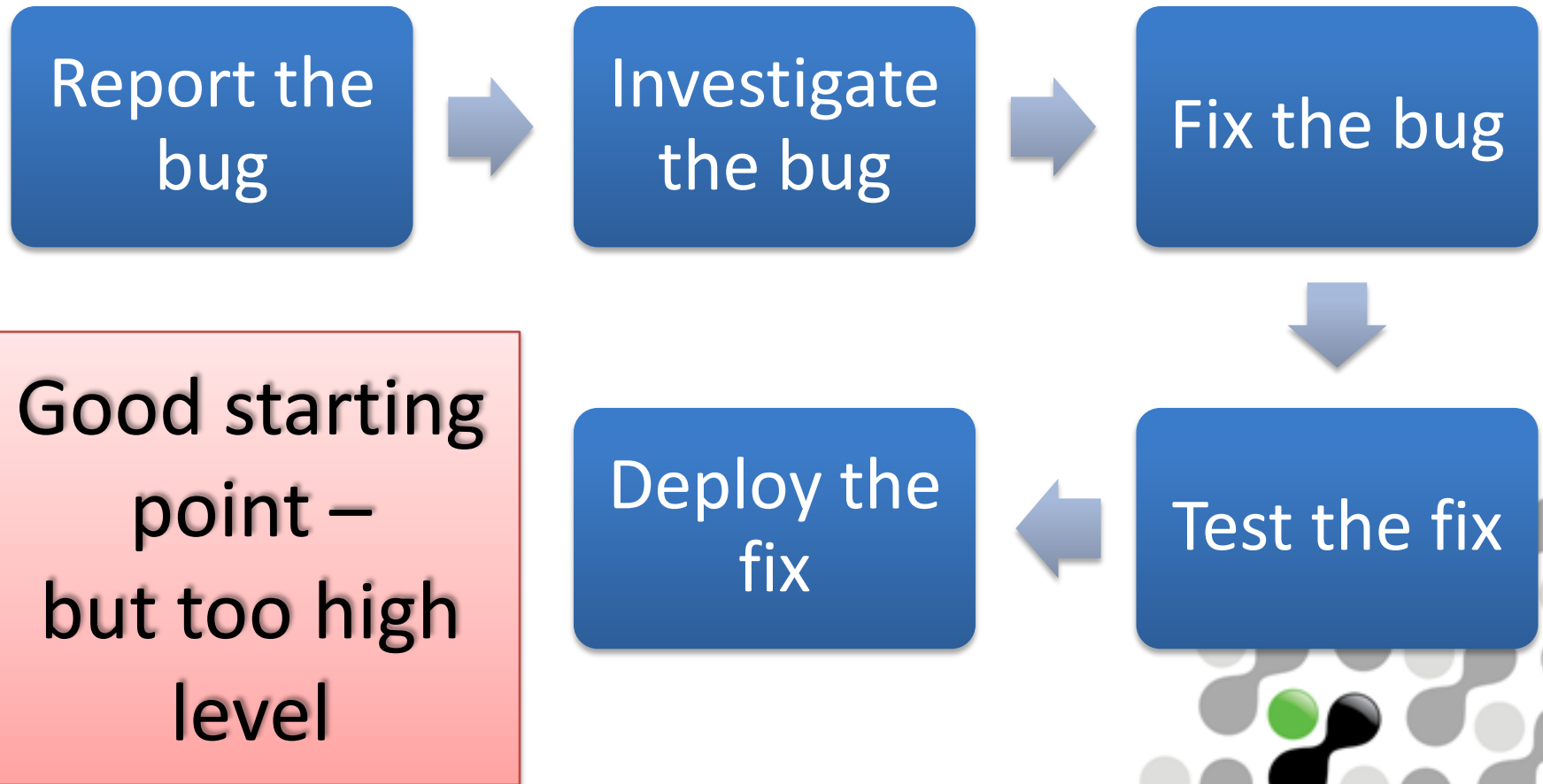


# GQM Approach

- Goal-Question-Metrics approach
  - Usually starts off with a “gut feeling”
- Goal: “We need to address customer issues more quickly by releasing bug fixes faster”
- Question: “How do we reduce the amount of time it takes to fix a bug?”
- Metric: Let’s figure it out...

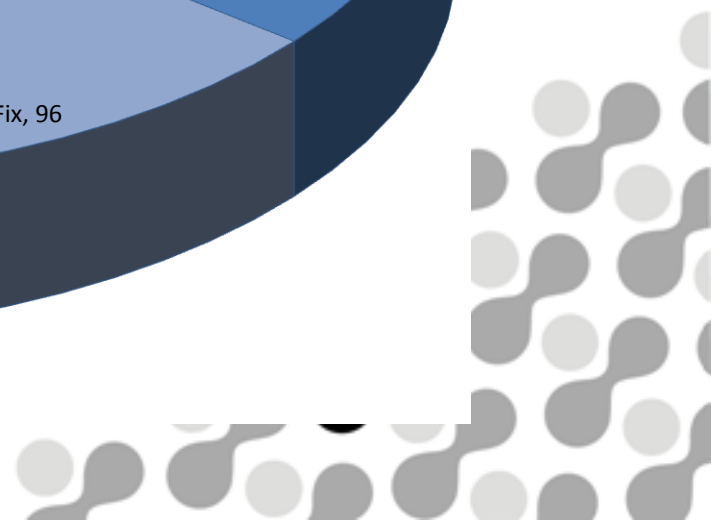
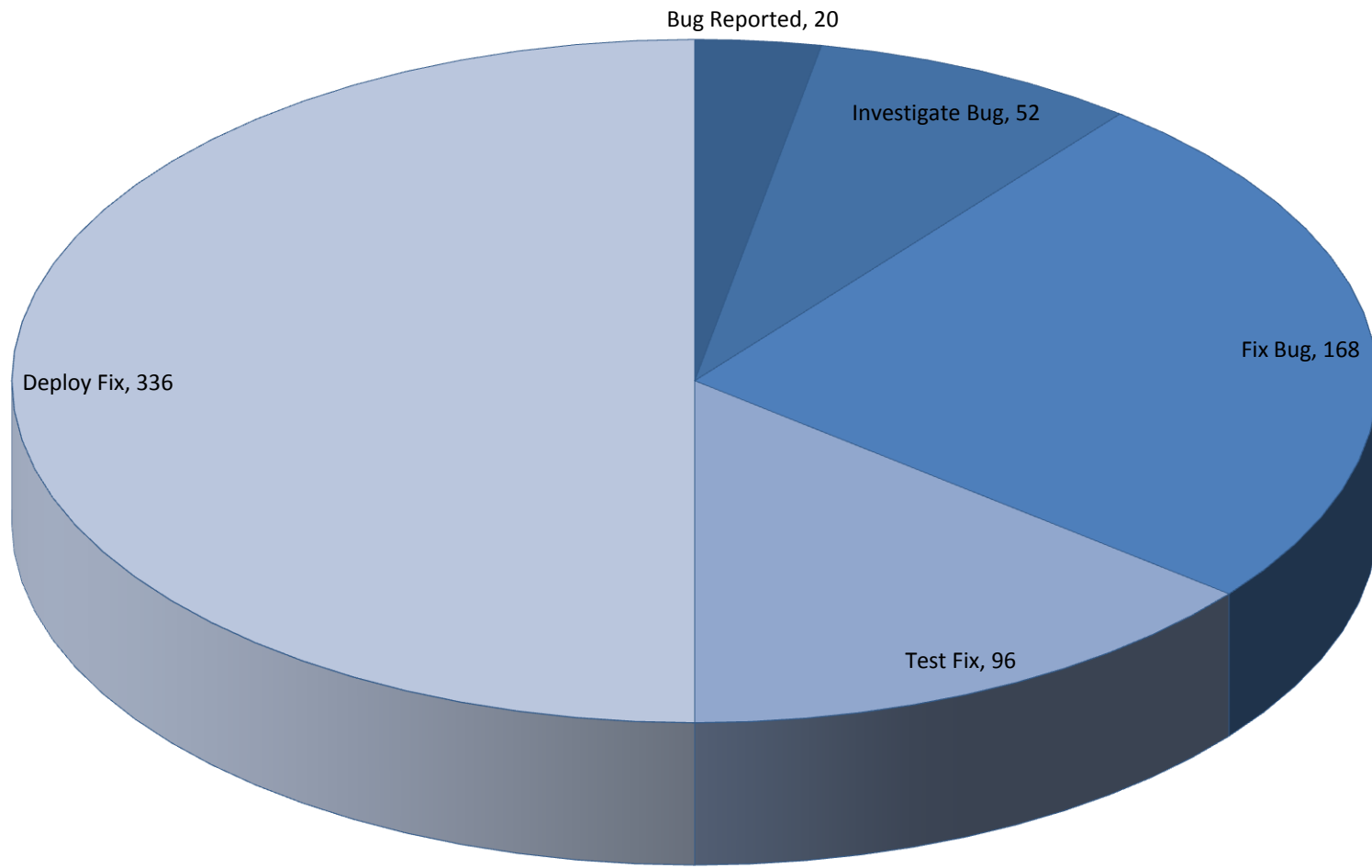


# A Bug Fixing Process

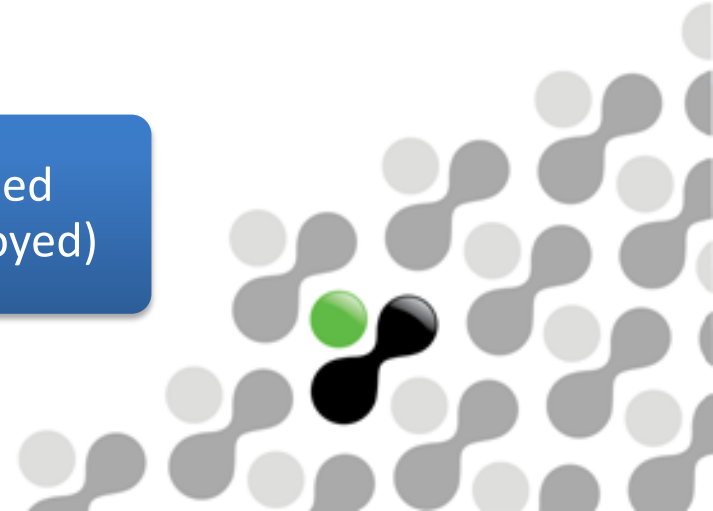
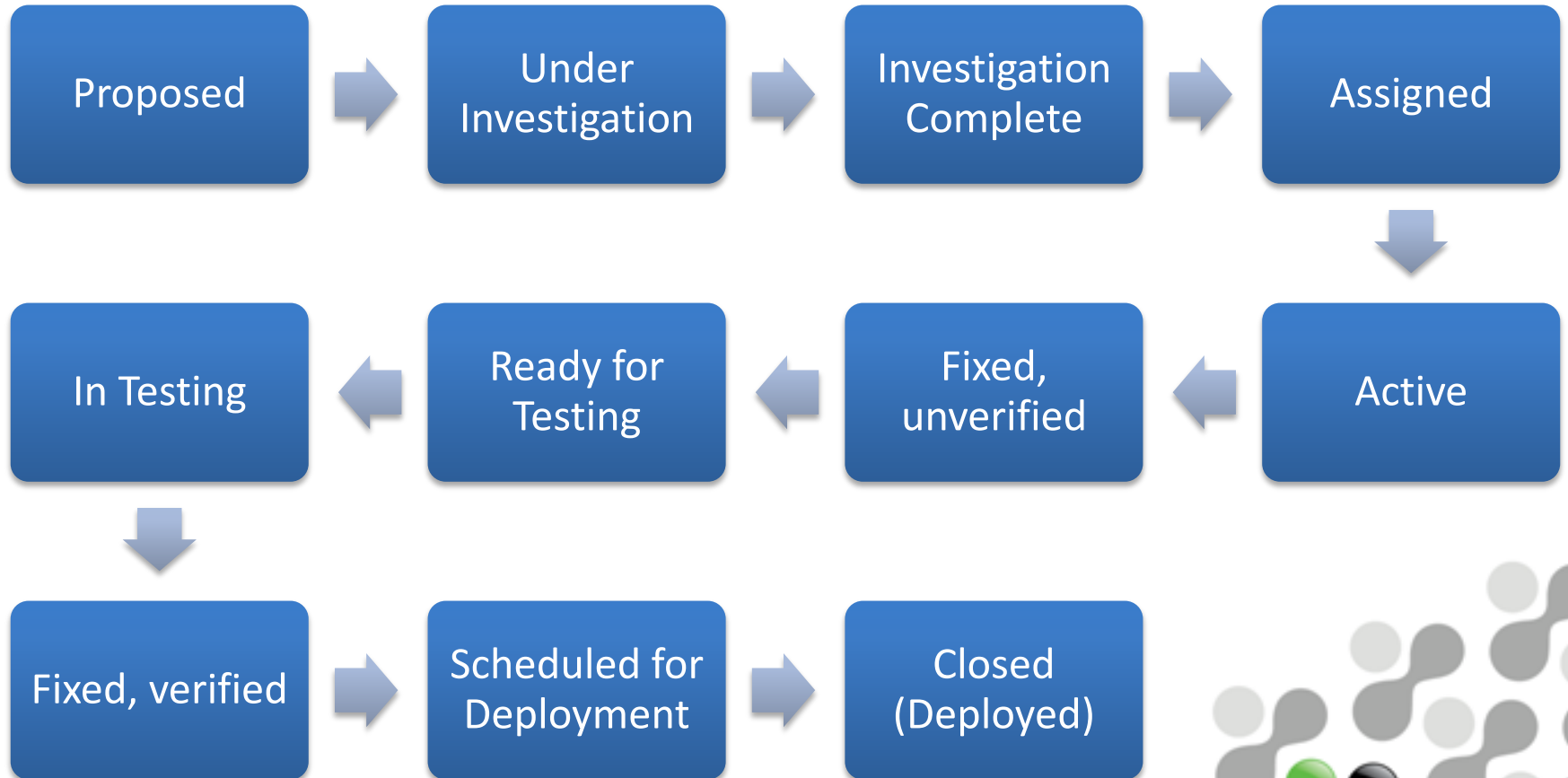




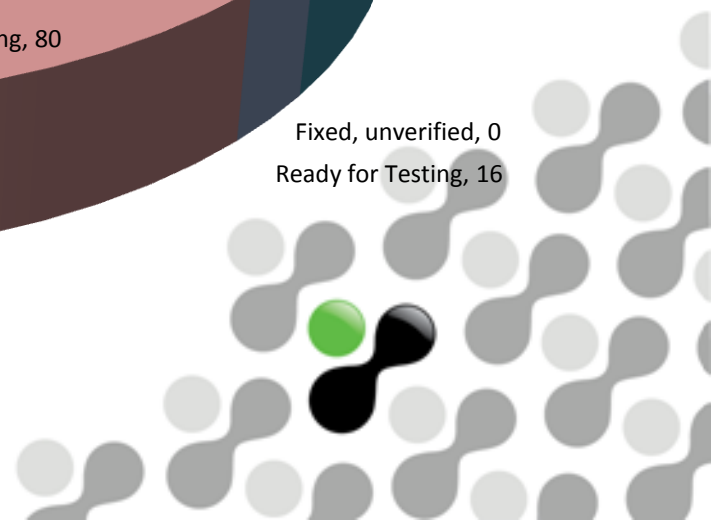
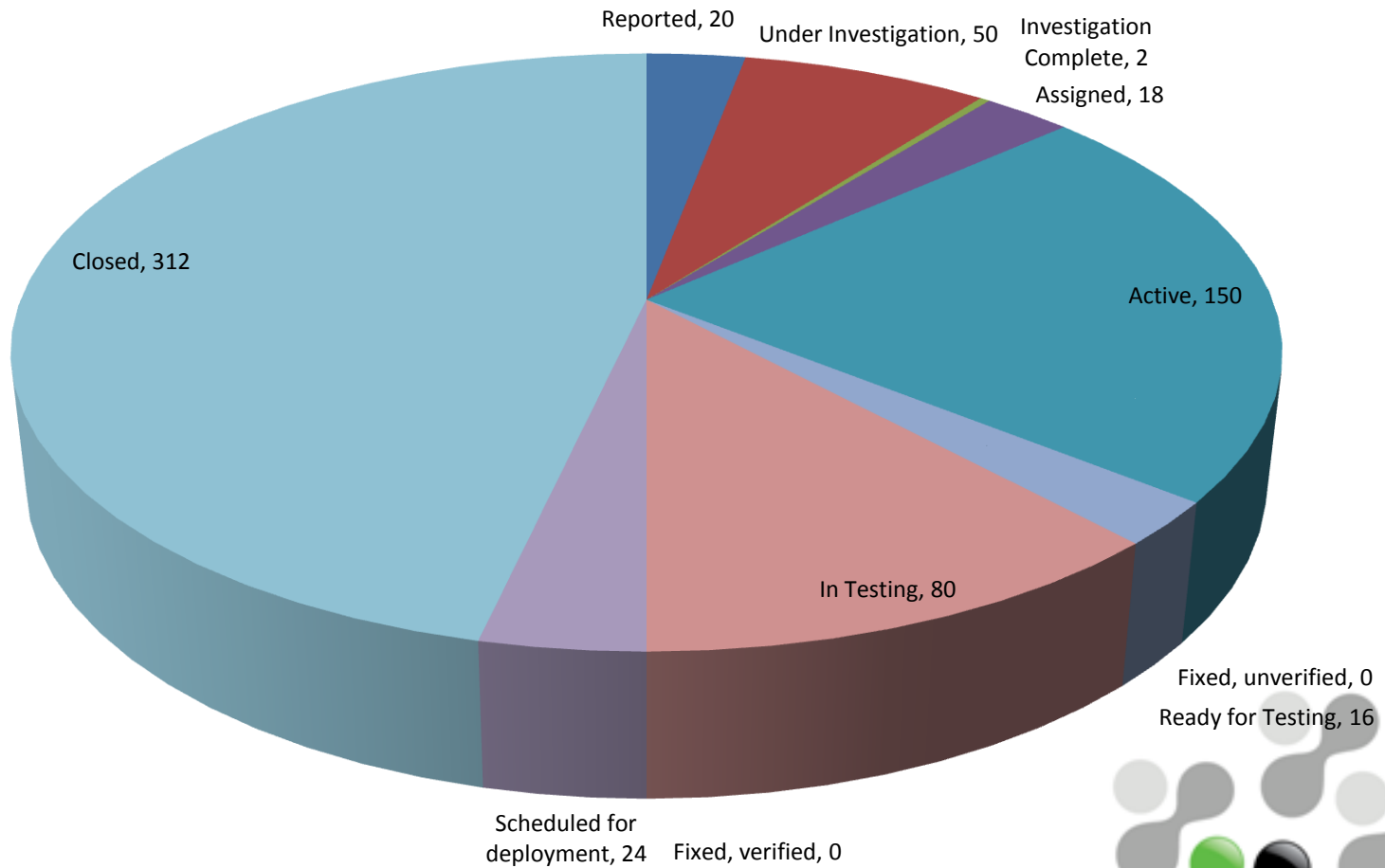
# High Level View



# Using Buffer States – Another Process



# Detailed View



# Which are the process steps?

- Under Investigation
- Active
- In Testing



# Under Investigation

- Check for duplicate bugs
- Find existing test cases
- Is it a change request or a bug?
- Can the bug be reproduced?
- Create or modify the test case



# Active

- Work according to the development process
  - Fix the bug
  - Are you required to create a unit test?
  - Code review
  - Run the functional test(s)



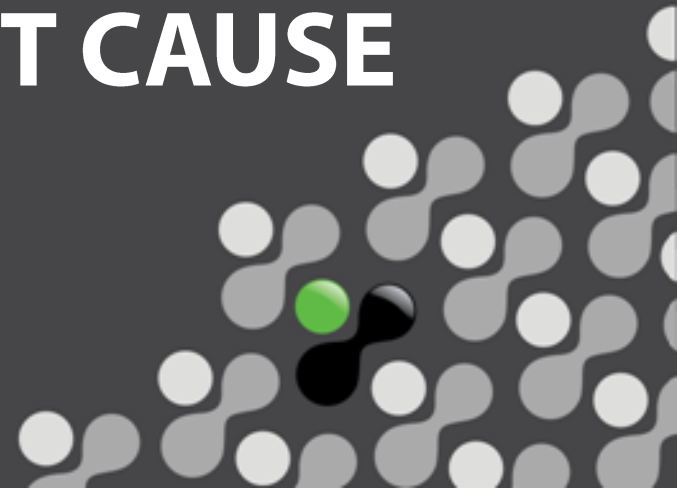
# In Testing

- Work according to the testing process
  - Examine bugs related to this function
  - Run the functional test(s)
  - Determine potential dependencies and test accordingly



Just avoid the bugs!

# ADDRESSING THE ROOT CAUSE





# Why Does Poor Quality Happen?

- Poor Requirements (#1 Reason)
- Poor Design
- Poor Coding (not as often as you think)
- Poor Communication
- Poor Release Management

*Did I miss anything?*



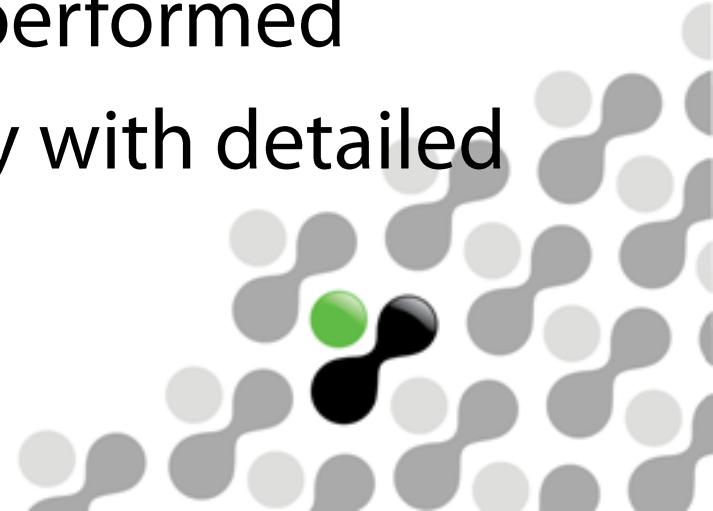
# Addressing Quality

- Quality happens as part of the process, *not* as a transition



# Metrics Can Help...

- ...but there has to be an investigation process
  - Why did it take 168 hours (average) to fix a bug?
  - Why did it take 80 hours to test the fix?
  - Why did it take 360 hours to deploy the fix?
- ...and a root cause analysis performed
- Waste can be eliminated only with detailed metrics

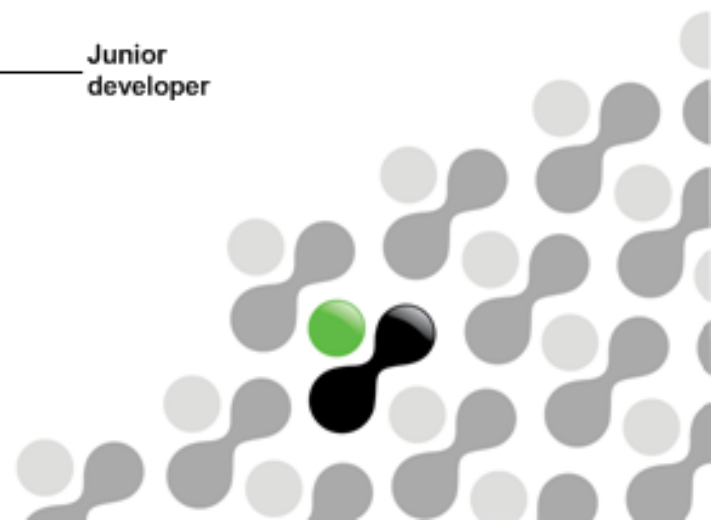
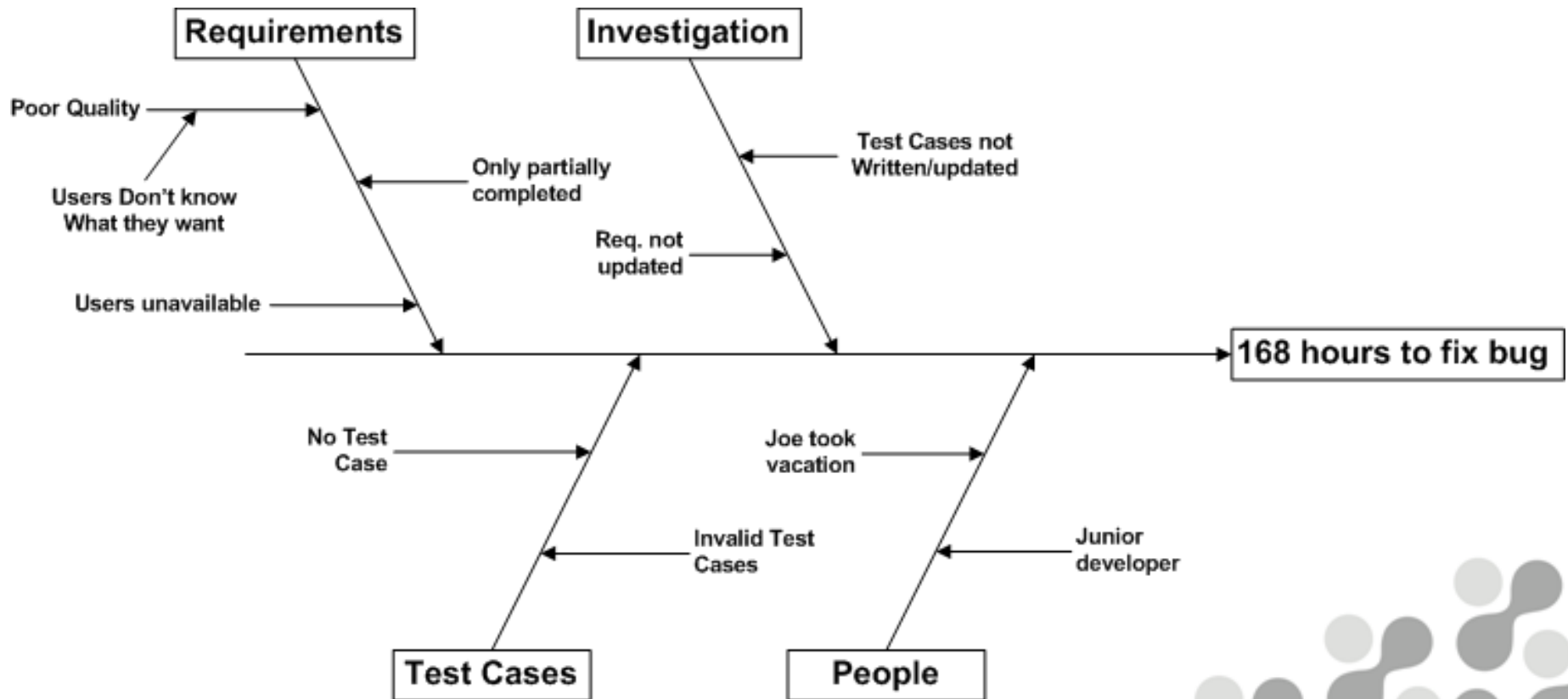


# Root Cause Analysis

- 5 Whys
- Ishikawa Diagram
- Fault Tree Analysis
- Failure Mode & Effect Analysis
- Others...



# Ishikawa Diagram



Try *before* you buy



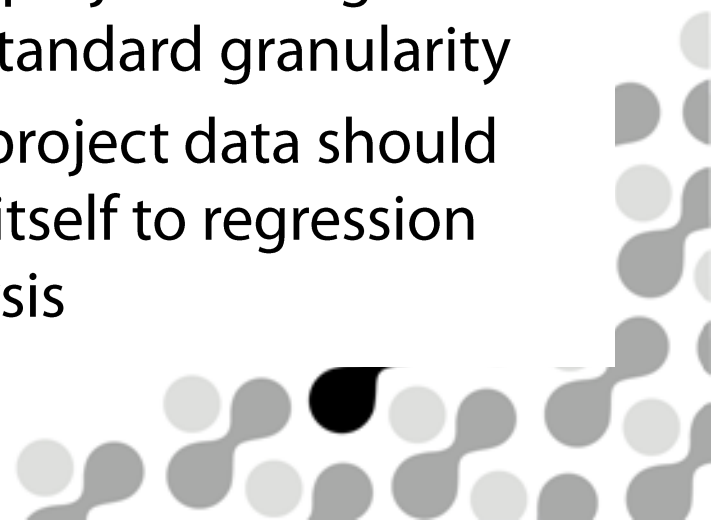
Is the effort worth it?

# USING METRICS





# Goals of Functional Metrics

- Visible features
  - Important to users
  - Early in lifecycle
  - Effect economic productivity
  - Independent of source code
  - Easy to apply and calculate
  - Assist in sizing deliverables
  - Should apply to existing software
  - Should apply to maint. and enhancements
  - Work with all software types
  - Hard project data gathered at a standard granularity
  - Soft project data should lend itself to regression analysis
- 

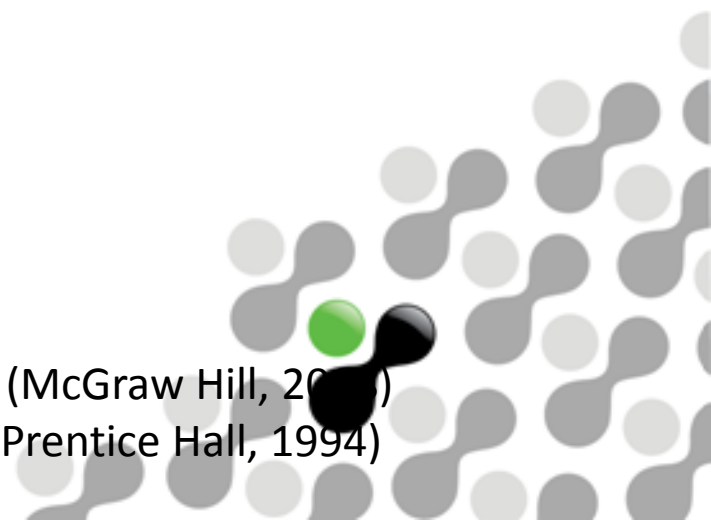


# What does it really get me?

- “Annual costs of accurate and complete measurement systems are 4% - 6% of the total software budget”<sup>1</sup>
- Software Quality Measurements ROI after four years is \$17 for every \$1 spent<sup>2</sup>

<sup>1</sup> Capers Jones, Applied Software Measurement 3<sup>rd</sup> Edition (McGraw Hill, 2001)

<sup>2</sup> Capers Jones, Assessment and Control of Software Risks (Prentice Hall, 1994)



[Jeff.Levinson@nwcadence.com](mailto:Jeff.Levinson@nwcadence.com)

**QUESTIONS?**

