DevOps for Defense

Test Driven / Behavior Driven Development

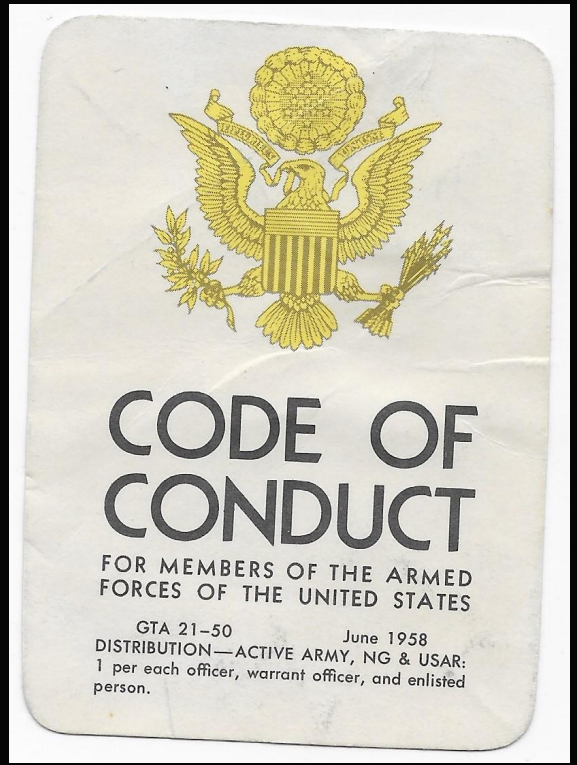
JD Black

https://devopsfordefense.org
https://www.meetup.com/DevOps-for-Defense/
https://github.com/jondavid-black/DevOpsForDefense
devopsfordefense@gmail.com
https://twitter.com/devops4defense

Sponsored by: TEKsystems Own change

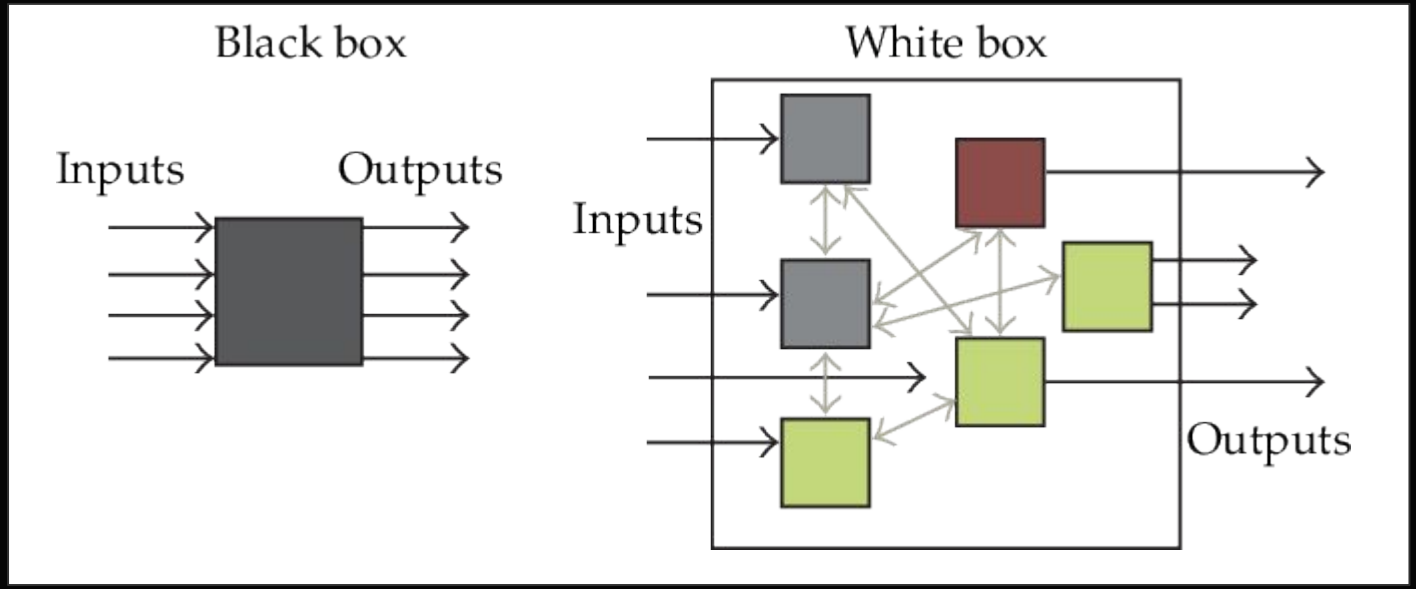
DevOps for Defense Meetup: Code of Conduct

- UNCLASSIFIED ONLY!!!!
- Treat each other with respect and professionalism.
- Do not talk about private, sensitive, or proprietary work.
- Do talk about your experiences, needs, desires to improve work in our domain.
- Do share your thoughts.
- Do learn from others.
- Do respect & tip your bartenders!



How Do You Test When Failure is Not an Option?

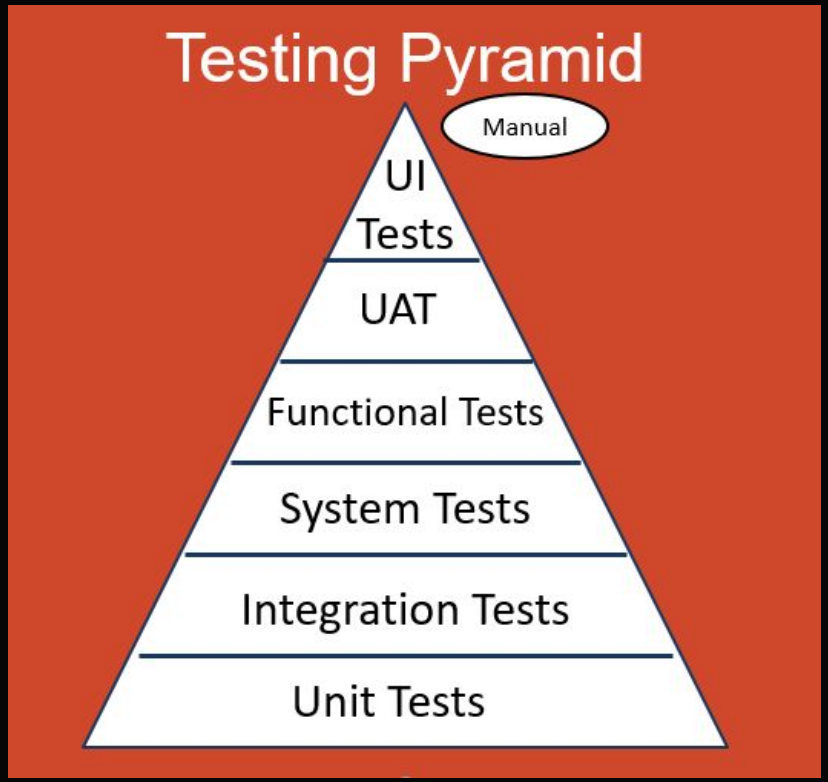
Definitions: White vs Black Box

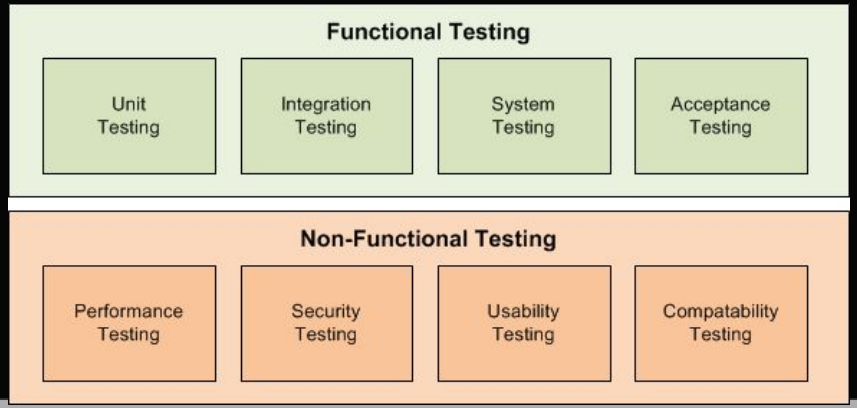


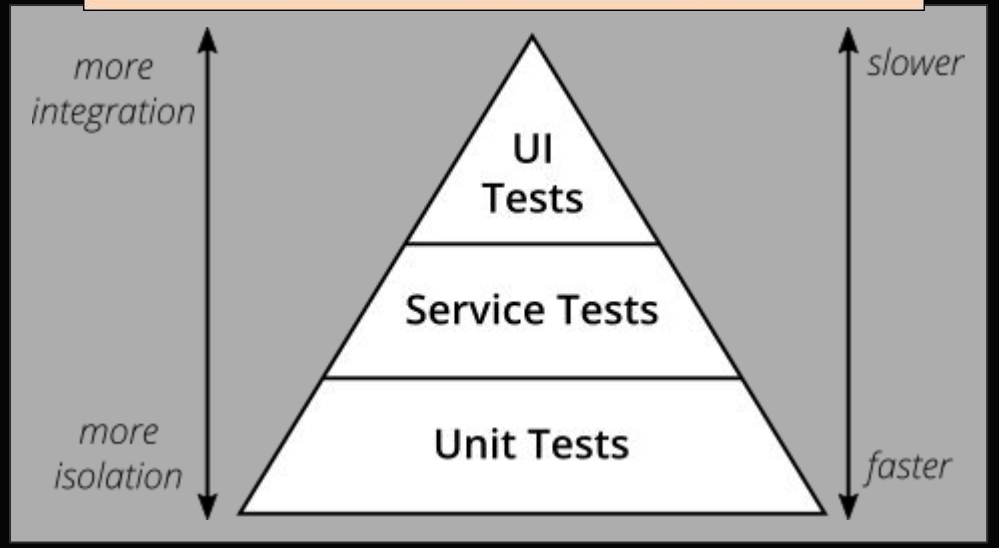
Test as a User
Interface Driven
Requirements Coverage
Nominal & Off-Nominal

Test as a Developer
Design Driven
Path / Branch Coverage
Isolation of Unit Under Test

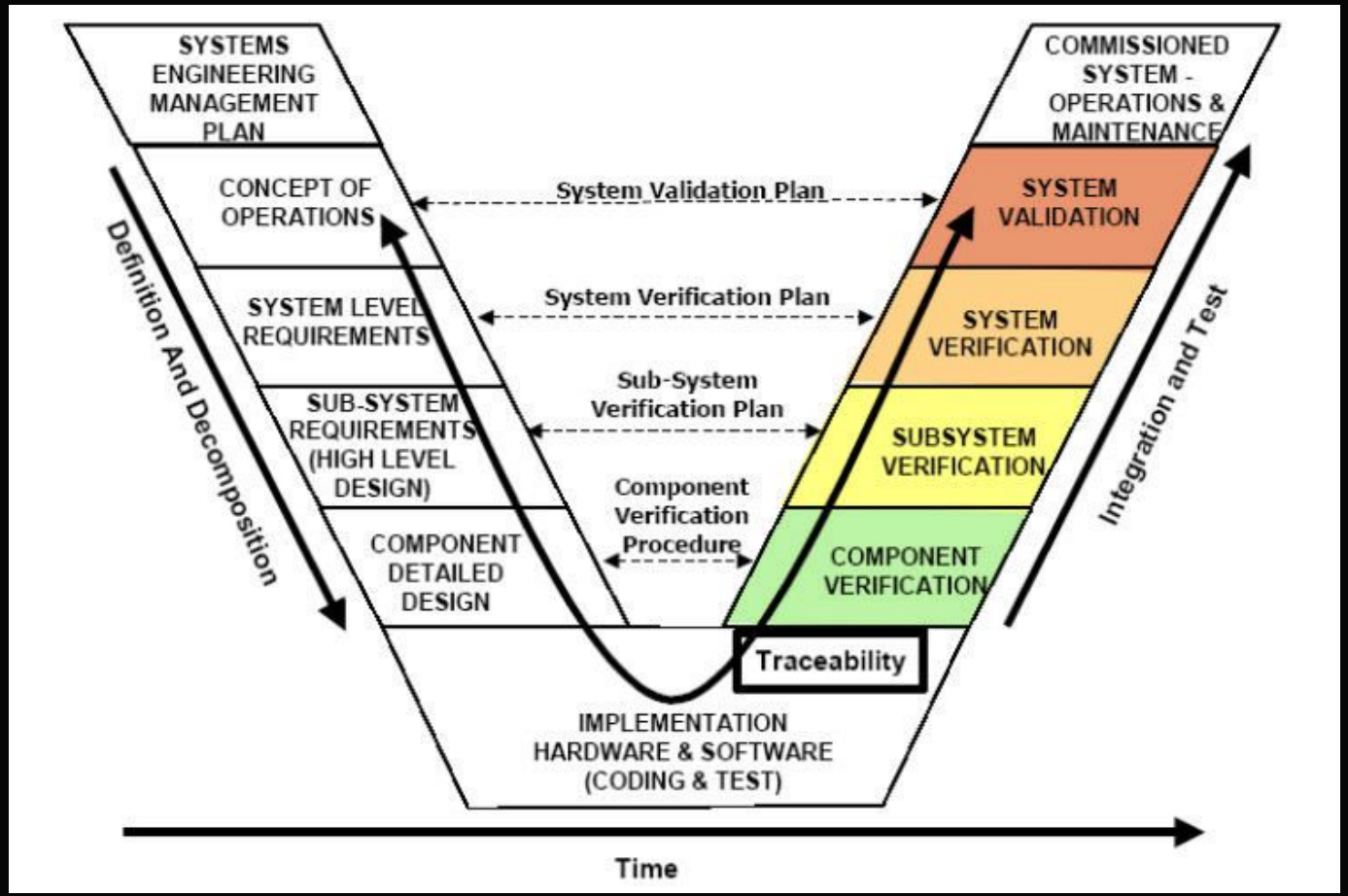
Definitions: Test Pyramid



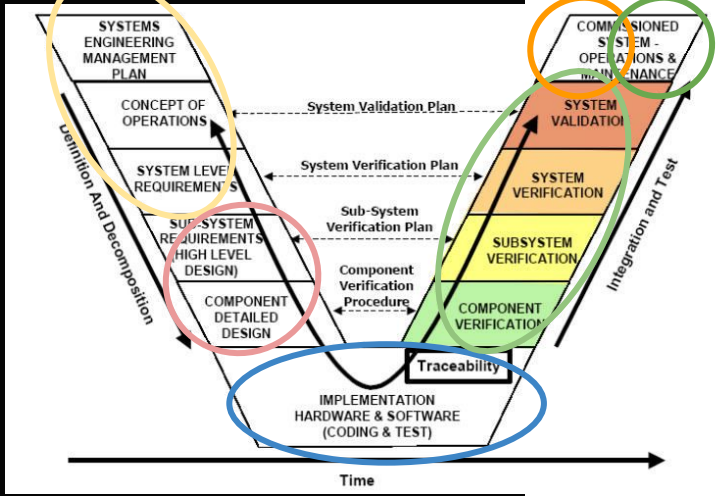


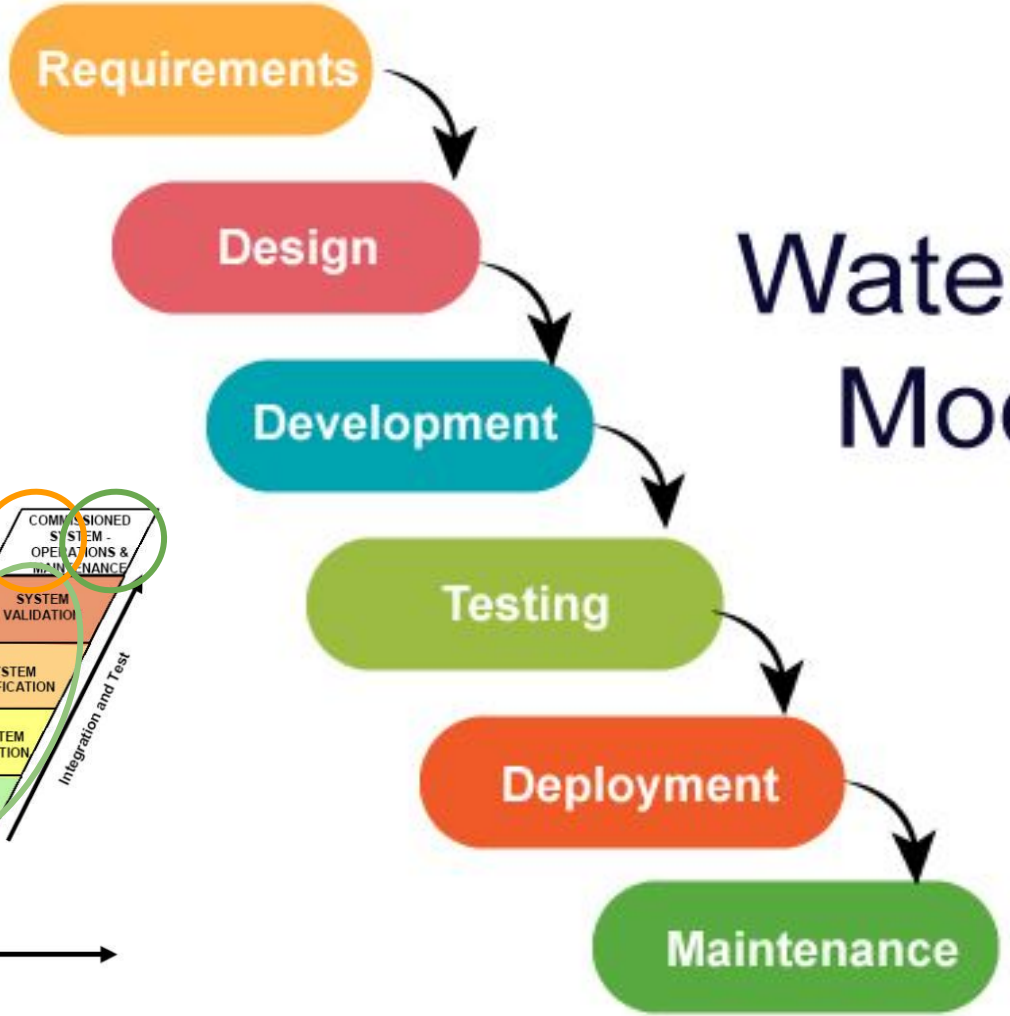


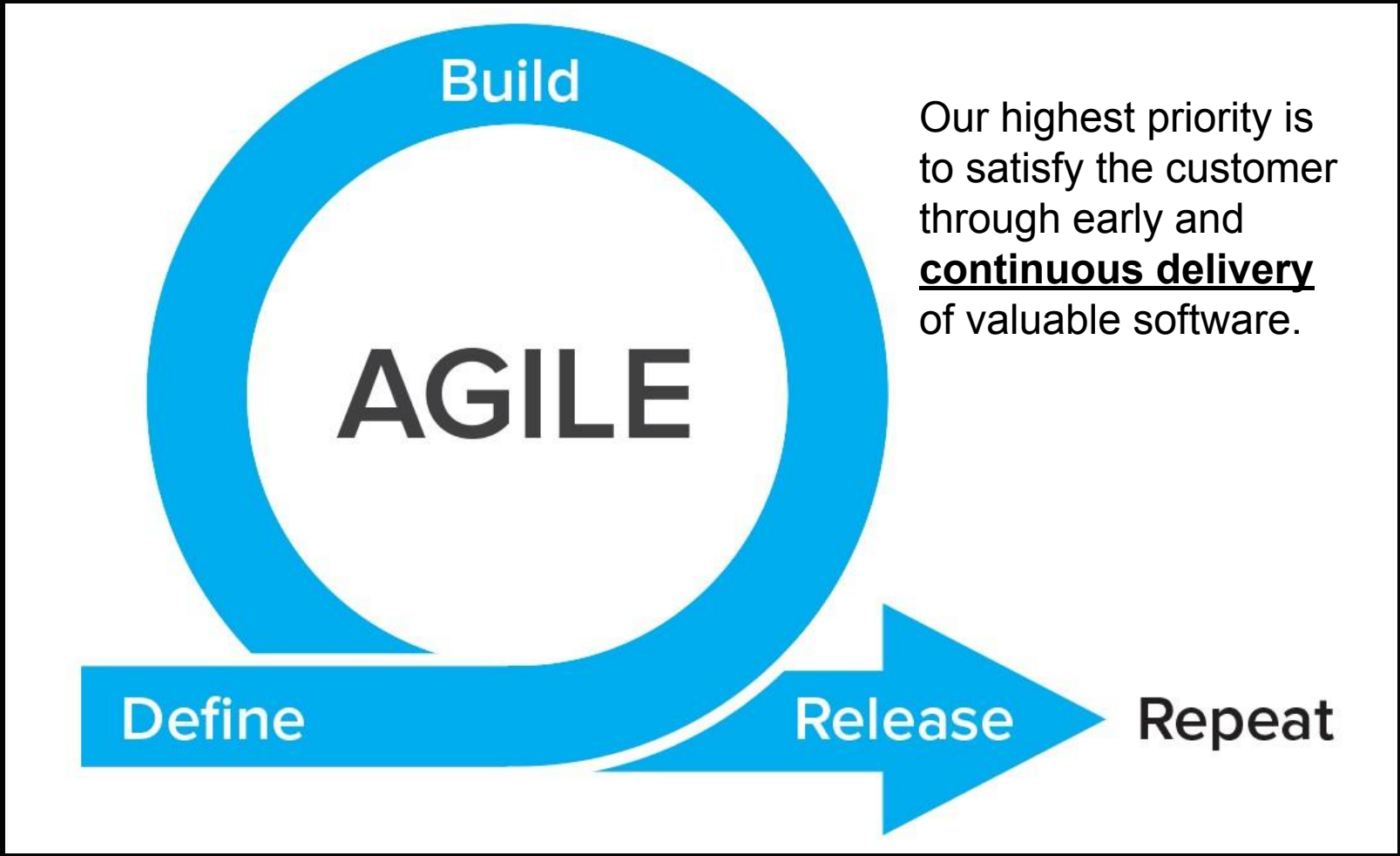
Traditional System Life Cycle



Traditional “Engineering V” aligns well to Waterfall



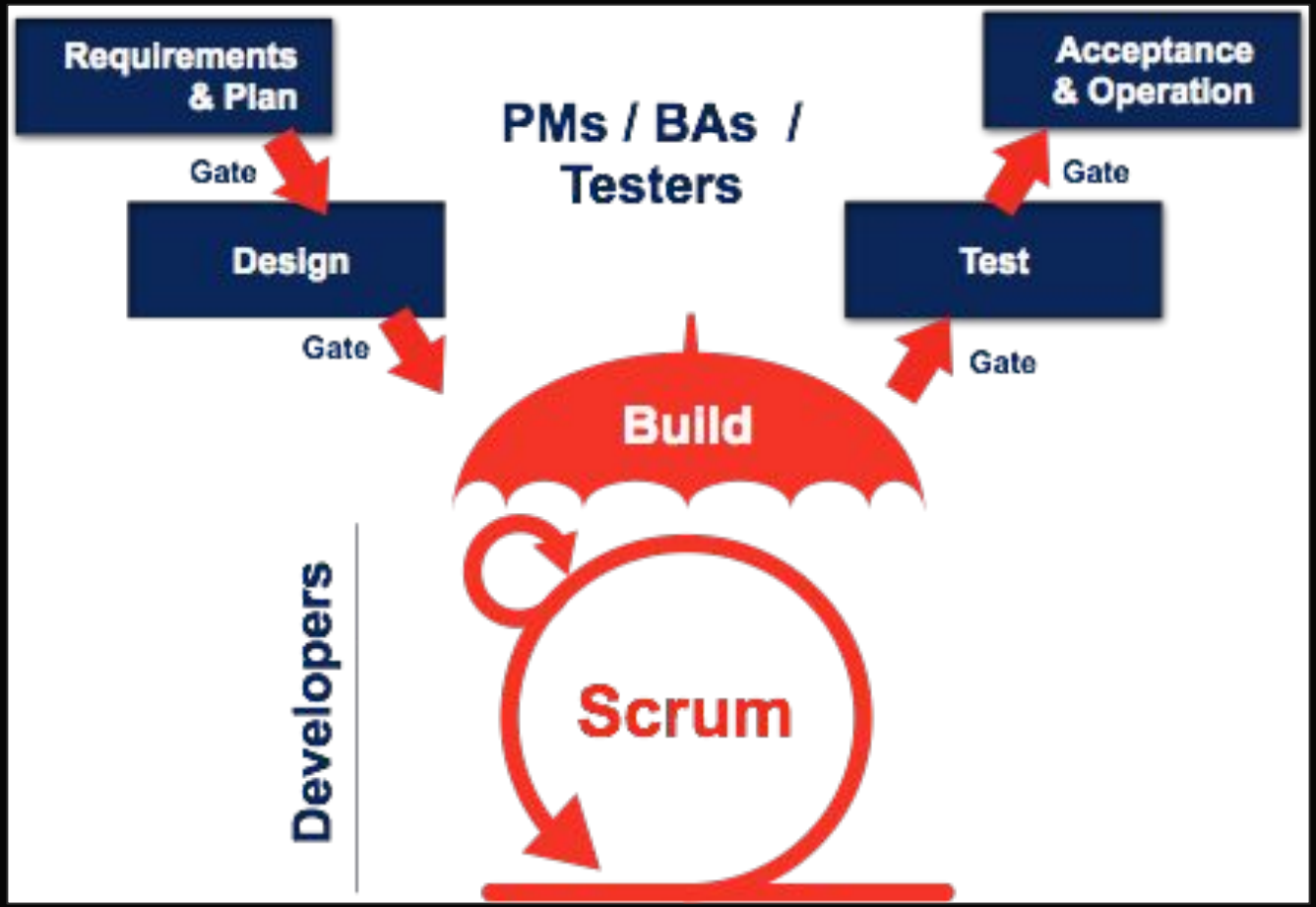




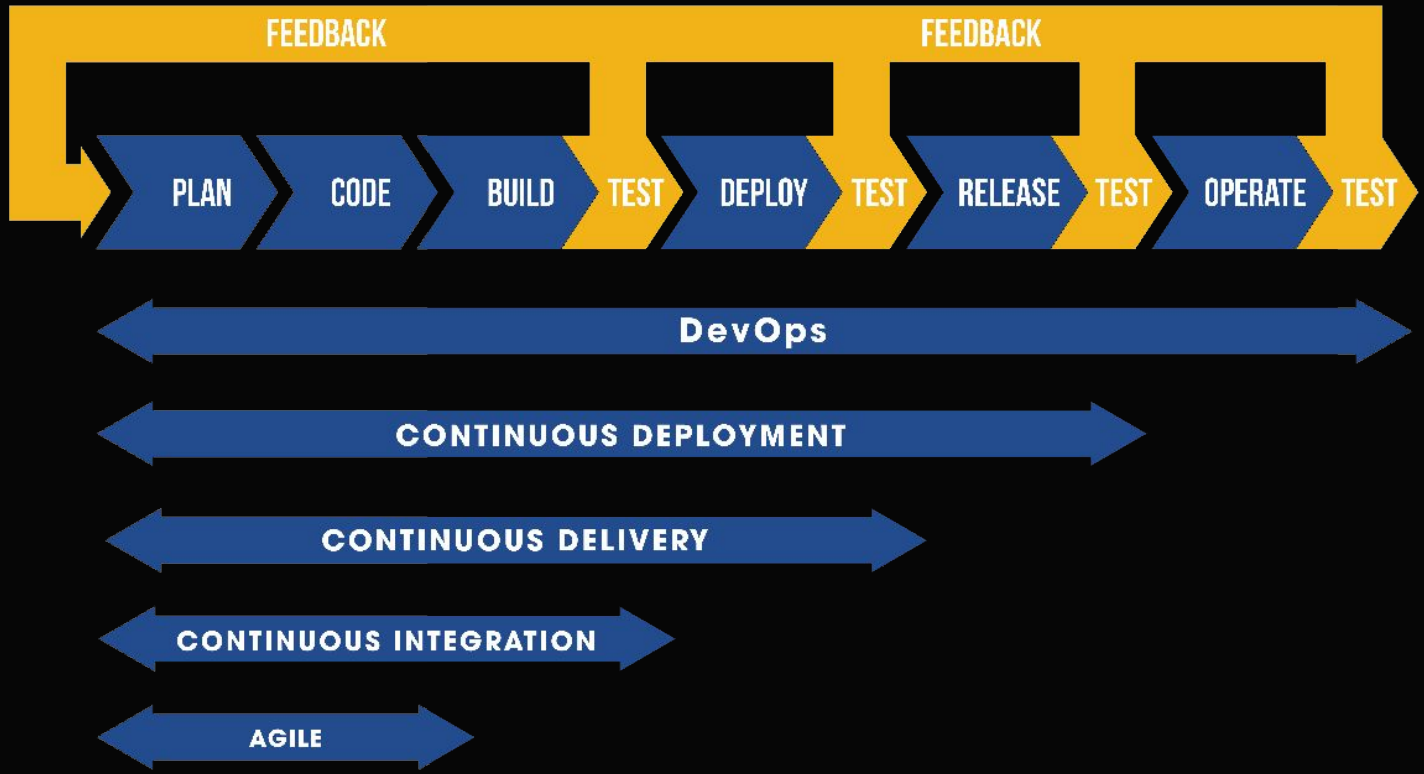
Our highest priority is to satisfy the customer through early and **continuous delivery** of valuable software.

So How Does Integration & Test Fit into Agile/DevSecOps?

Water-Scrum-Fall is the Wrong Answer!



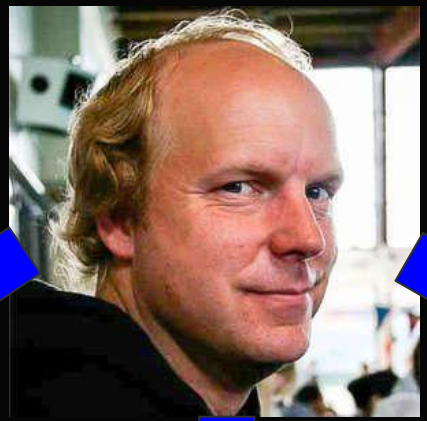
More Testing! (Not Less!)



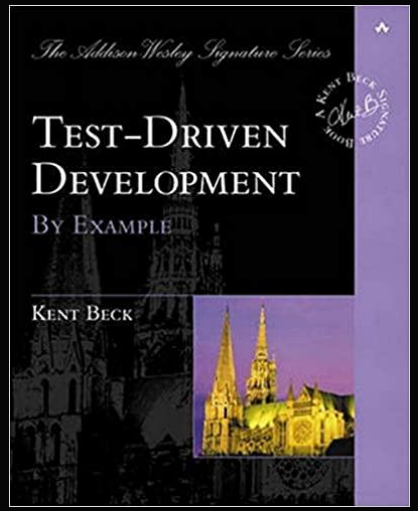
“Continuous is much more often than you think.” - Mike Roberts, Thoughtworks

Test Driven Development - Background

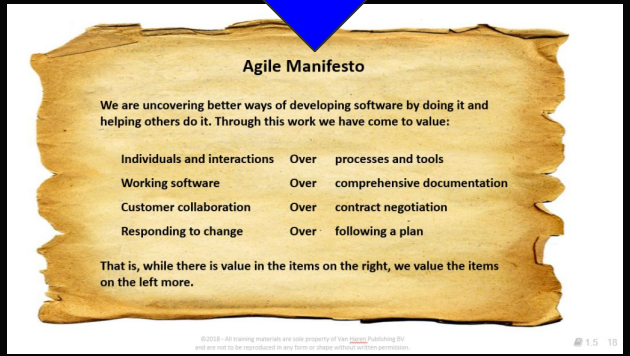
Kent Beck



2002



2001

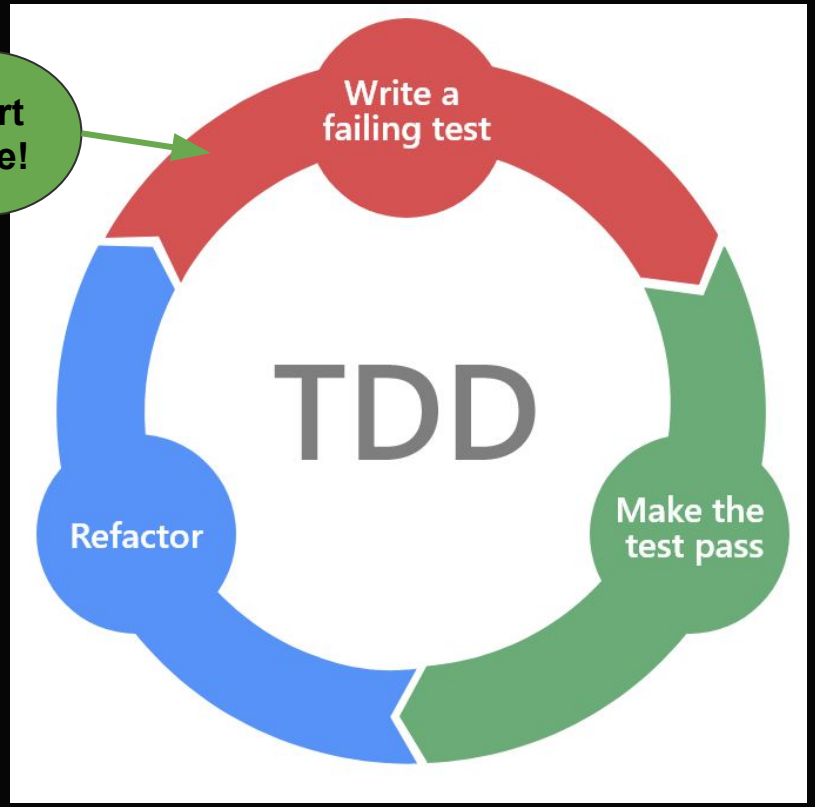


2002

JUnit

Test Driven Development in Practice (FYI...This is Hard!)

- Red
  - Understand the problem (i.e. requirements) and the expected outcome (i.e. success criteria)
  - Write the test to demonstrate the expected outcome
  - Ensure the test fails
    - Or claim success!
- Green
  - Do the minimum work necessary to make the test pass
  - Demonstrate with objective evidence
- Refactor
  - Ensure implementation:
    - Meets Standards
    - Passes Quality Checks
    - Is Adequately Documented
    - Etc.

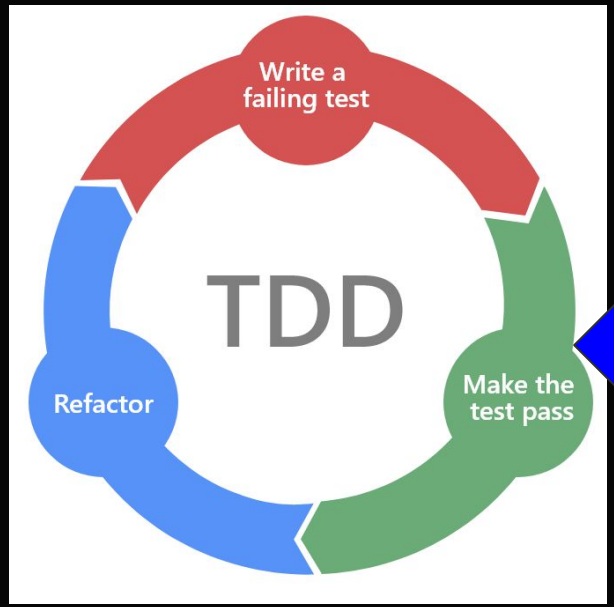


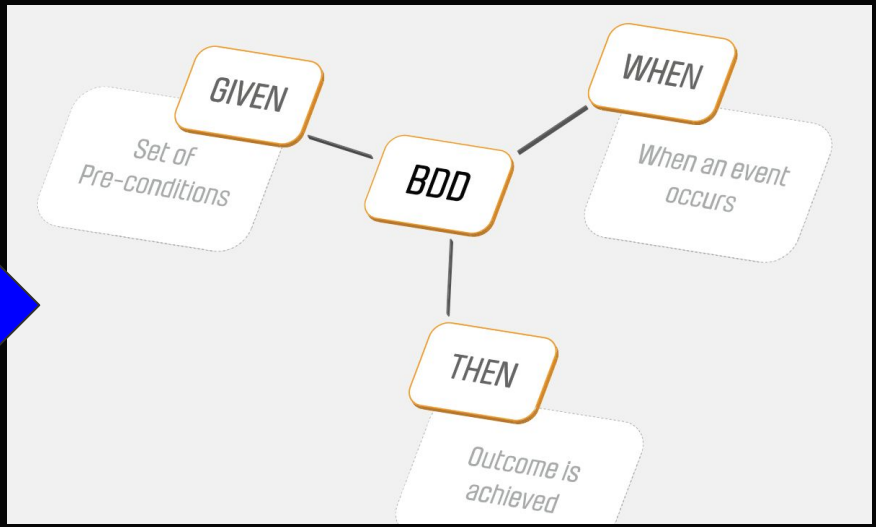
I&T “Shifts Left” and Aligns More Closely with System Engineering

Behavior Driven Development (TDD evolved)

Dan North

2006 https://dannorth.net/introducing-bdd/





BDD follows TDD Cycle but Elevates to Focus on Value/Outcomes

Don’t Just Test an Implementation,
Ensure the Implementation Delivers the Value the Customer Needs

How does TDD/BDD fit into a DevSecOps SW Factory?

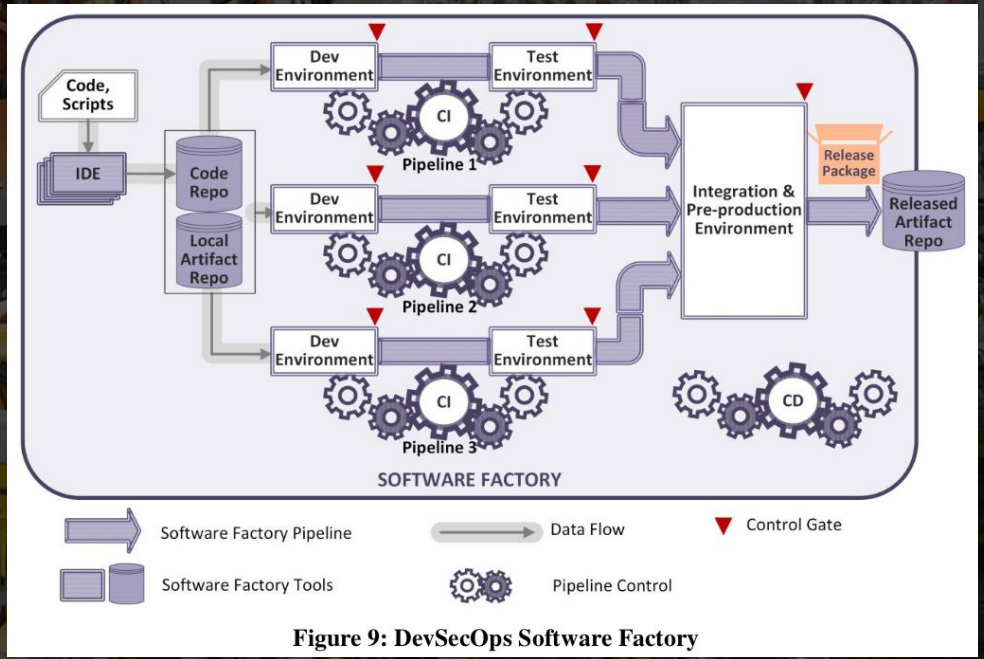


Figure 9: DevSecOps Software Factory

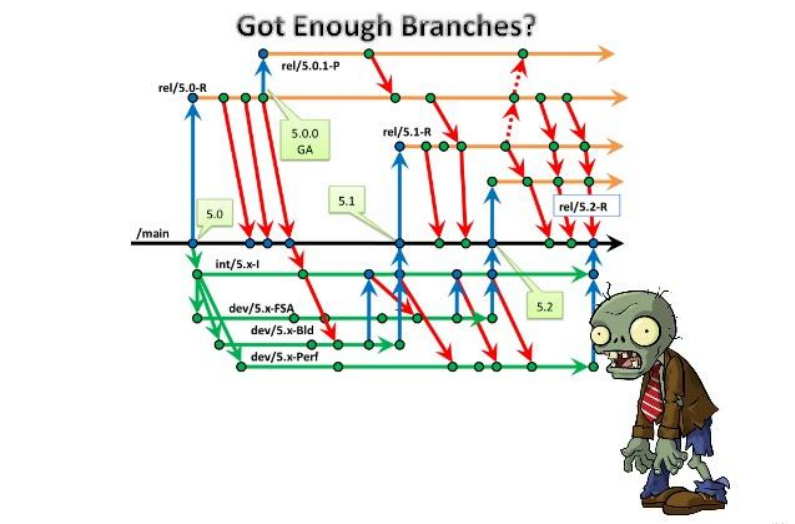
Good Testing

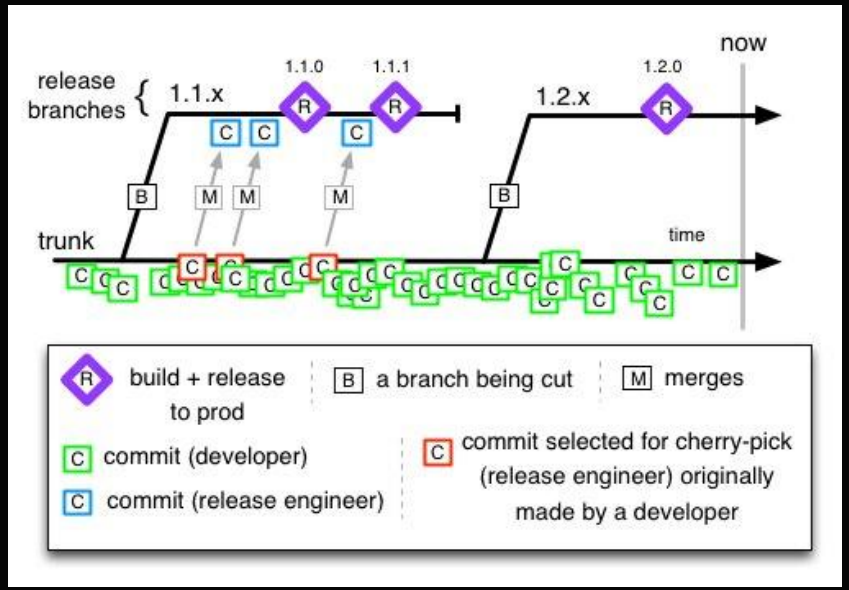
1. Enables Speed
2. Ensures Quality
3. Reduced Rework

Must be

1. Fast
2. Affordable
3. Repeatable
4. Reliable

SW Factory CI/CD Prerequisite: Good CM





"CM Smells"

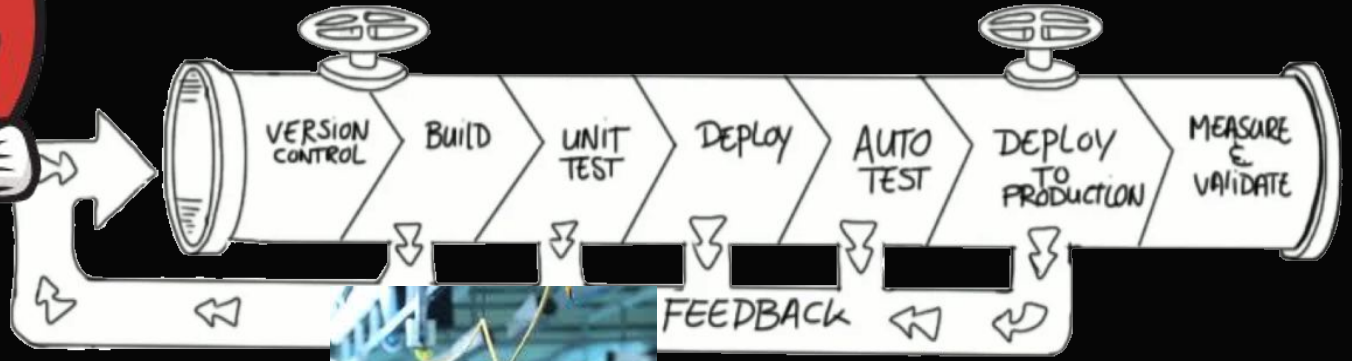
- Lots of branches
- Long lived branches
- Large complex merges take a long time
- Deep branch hierarchy (e.g. based on org)
- You know what a config spec is and have mastered manipulating it

Trunk-Based Development

- Recognize that branches defer integration
- Commit to trunk / master is the ideal
- Use short lived feature branches (aka Pull Requests) if that works best for your team
- Monitor branch lifespan & kill them off

The CI/CD Fundamentals

2) Your pipeline steps delegate to an automate build tool (e.g. Gradle)



1) Use a pipeline tool to automate execution (e.g. Jenkins)

3) When something fails in your pipeline, stop and fix it <u>immediately</u> (e.g. Andon Cord)

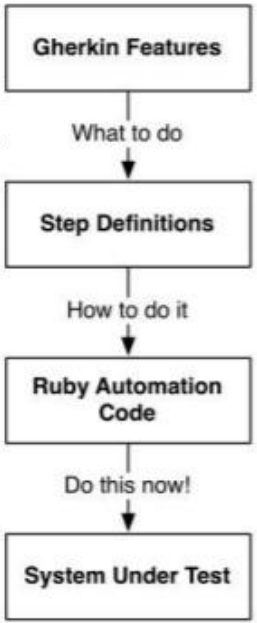
Automated Acceptance Testing...the BDD Way

Cucumber Testing Stack

Gherkin:
1. Specifications from plain-language text files called **features.**
2. Each **scenario** is a list of steps for Cucumber to work through

Step Definitions:
Map the business-readable language of each step into Ruby code to carry out whatever action is being described by the step.

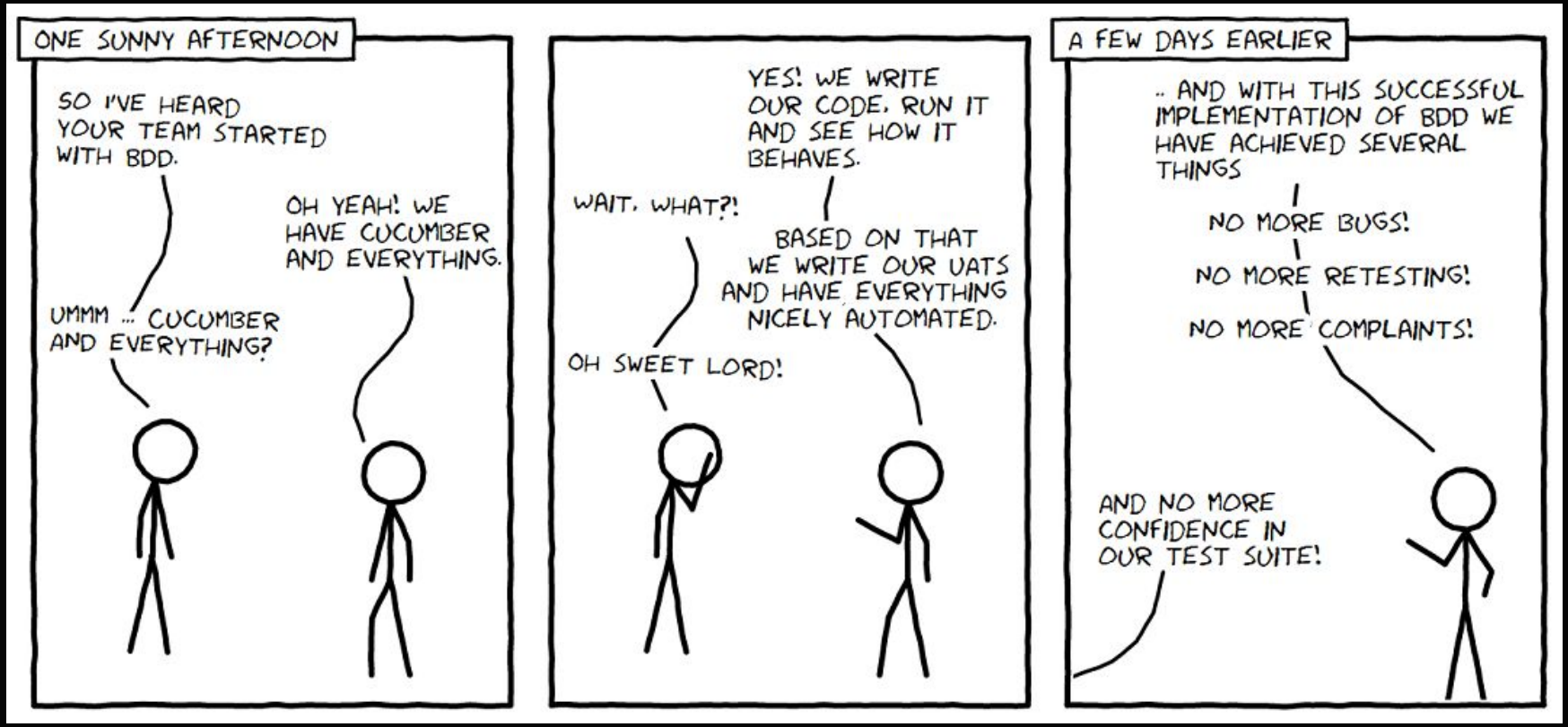
Automation library:
One or two lines of Ruby that delegate to a library of support code, specific to the domain of your application.




```
# language: en

Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario Outline: Add two numbers
    Given I have entered <input_1> into the calculator
    And I have entered <input_2> into the calculator
    When I press <button>
    Then the result should be <output> on the screen

    Examples:
      | input_1 | input_2 | button | output |
      | 20      | 30      | add    | 50     |
      | 2       | 5       | add    | 7      |
      | 0       | 40      | add    | 40     |
```

*Supports multiple implementation languages...including Java

Proceed with Caution…..



BDD Demonstration

1. Test an application I didn't write
2. Strict Black Box / Acceptance Testing
3. Fully Automated Example

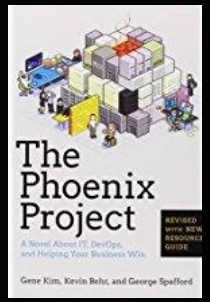
https://github.com/jondavid-black/BDD-Introduction

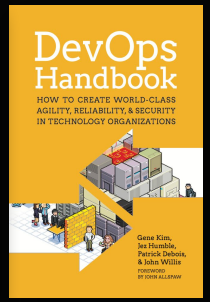
DevOps Resources

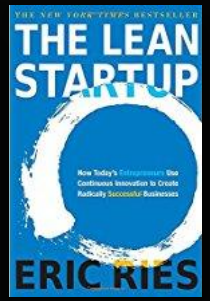
https://devopsfordefense.org/resources/

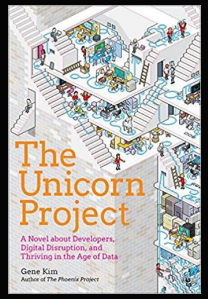
https://www.meetup.com/DevOps-for-Defense/
https://github.com/jondavid-black/DevOpsForDefense
devopsfordefense@gmail.com

Books / Publications:









Conference Presentations (YouTube):

- DevOps Enterprise Summit (DOES)
- IT Revolution
- Velocity
- GoTo

YouTube