# CS1632, Lecture 19: Smoke and Exploratory Testing

Wonsun Ahn

# Smoke Testing

# Smoke Testing (plumbing)

- Send smoke down the pipes to find leaks BEFORE sending water or other fluids
- Why?
  - Won't waste effort: If there is a leak, nothing to clean up
  - Won't cause further damage

# Smoke Testing (software)

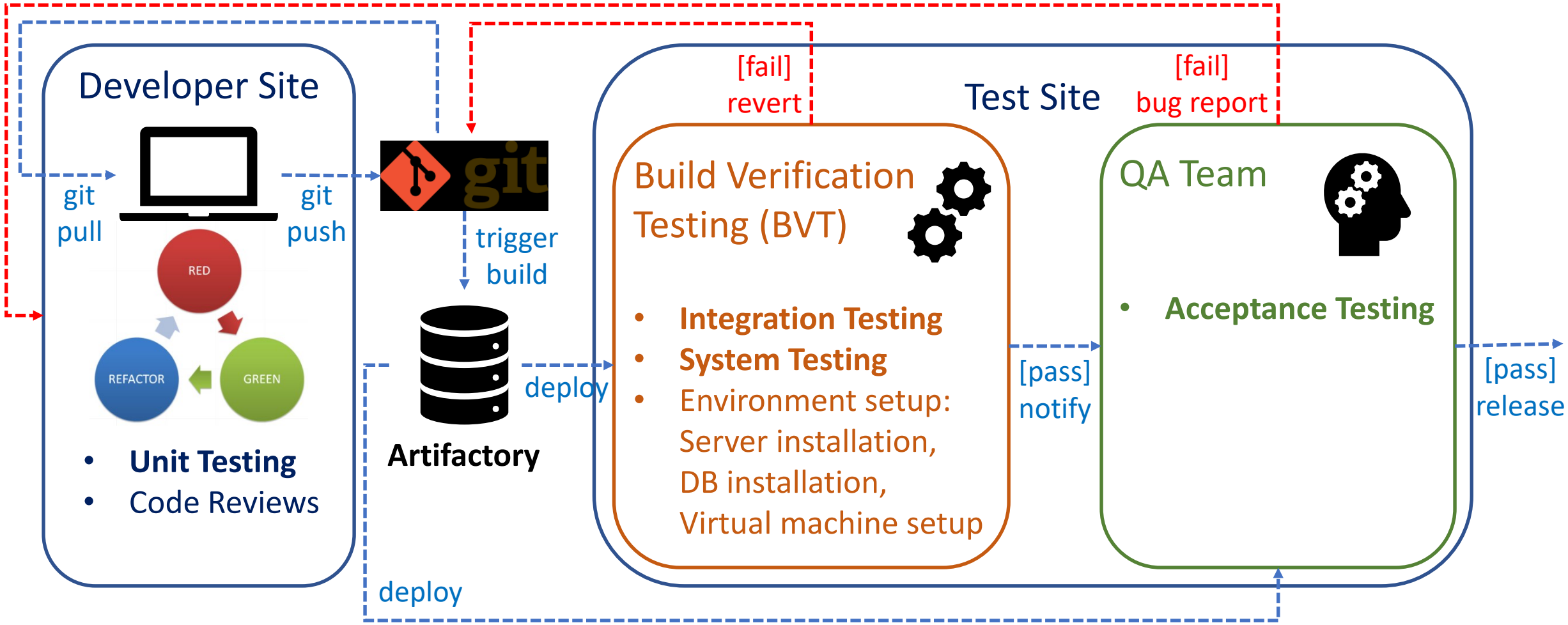
- Minimal testing to ensure that the system is ready for serious testing

- Why?
  - Setting up software / hardware testing environment may be non-trivial
  - No need to spend the effort if system isn’t ready for prime time
  - In essence, avoid wasting testers’ time

# Other names for Smoke Testing

- Confidence Testing
  - Because it’s intended to inspire enough confidence to pass to the QA team
- Sanity Testing
  - Because it’s intended to check that developer was fully awake when coding
- Build Verification Testing (BVT)
  - Because it’s intended to be performed after every build before further testing

# A Typical Software Testing Pipeline



# When is BVT Run?

- BVT is a form of regression test
  - Checks whether program has regressed due to recent change(s)

1. When new code has been committed and a new build created
   - Verifies every single commit – ethos of continuous integration (CI)

2. Periodically (e.g. every night)
   - Verifies program hasn't regressed during the day

# What goes on in a BVT?

- Integration / system tests
  - All modules are integrated for testing (no more mocks!)
  - Typically done on an integration server, inside a virtual machine
  - In the virtual machine:
    - Deploy build from build artifactory
    - Install other software or database required by program
- (Re-)run unit tests that developer should have already run for TDD
  - To catch developers who were too lazy to run them before committing!
- Basic code quality control
  - Linters, bug finders, compilation with full warnings, …

# What happens on BVT Pass / Fail?

- On BVT Pass
  - Notify QA team by (automated) email for further testing

- On BVT Fail
  - That means code repository is in a broken state now!
  - Nobody can add code to the repository until something is done about it.
  - It’s the developer’s responsibility to do one of two things:
    1. Immediately revert repository to known safe version
       (May need to revert all downstream changes along with your change, annoying many)
    2. Immediately patch the bug and run BVT again

# BVT Needs to be Fast!

- BVT is in the critical path of development
  - On BVT fail, may need to revert all changes done while BVT was running
  - On BVT fail, no commit can happen until BVT passes again
  - Every minute you save in BVT is a minute you save in the development cycle

- Recommended BVT time is ~ 10 minutes
  - A lot of care needs to be put into the test cases you select for BVT
  - You should use the time to cover all major features of your program

# BVT Implementation

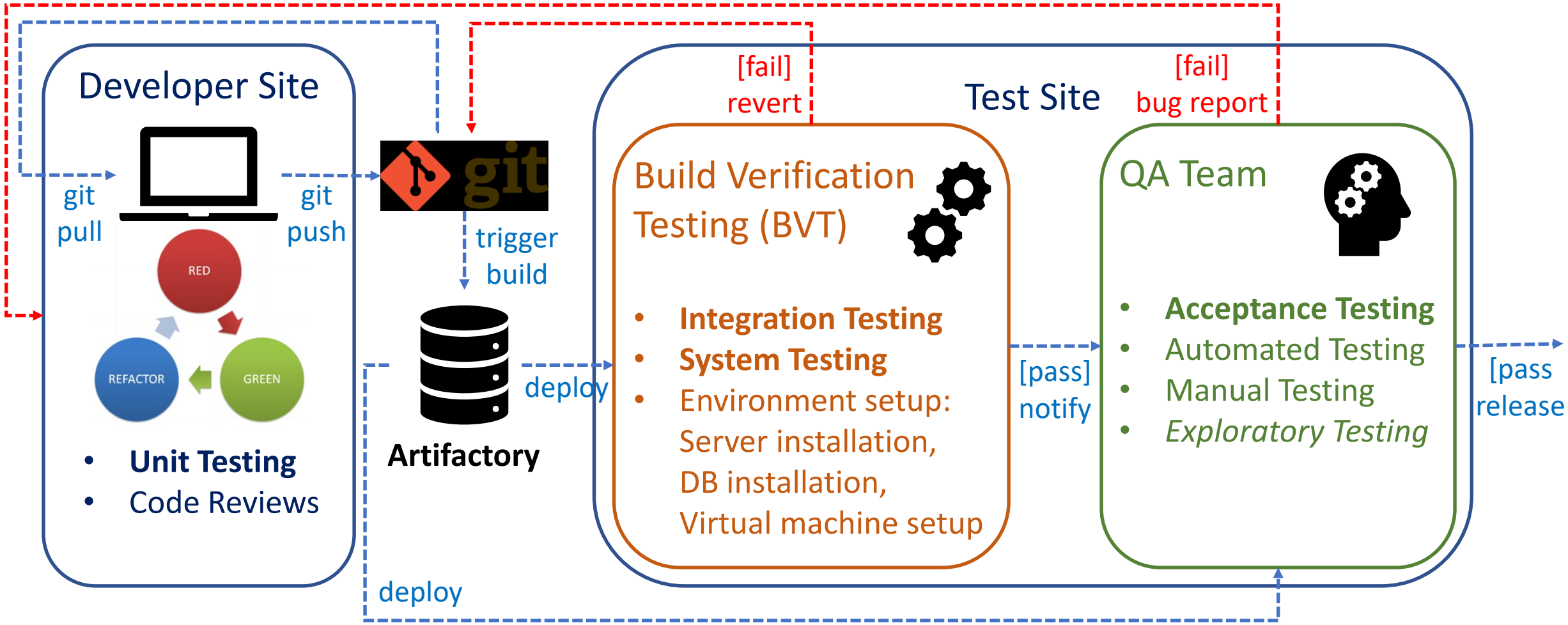
- In theory, BVT can take any form
  - Scripted / Unscripted
    - If scripted, the same pre-made test plan is run on every build
    - If unscripted, an experienced tester selects a set of tests relevant to code change
  - Automated / Manual

- Given BVT must be fast, usually *scripted* and *automated*

- Recently, sometimes *unscripted* and *automated*
  - Artificial intelligence is used to automatically select tests relevant to change

# Exploratory Testing

# A Typical Software Testing Pipeline



# What is Exploratory Testing?

- When development team implements a new feature …
  - Exact requirements of the feature may not have been hashed out yet (What user interface should look like, what user should be allowed to do, …)
  - Some requirements are still subjective or remain implicit

  → A detailed test plan for the feature may not be in place yet

- *Exploratory Testing*: testing without a test plan
  - To have QA team learn more about the system
  - To have QA team influence the development of the system
  - To help crystallize requirements and a formal test plan

# Sometimes called *“ad hoc” testing*

- Not a good term – it implies carelessness
- Less rigid != more careless
  - Same way stochastic testing wasn’t mindless monkey testing

- A lot of thought and care goes into exploratory testing
  - Tester must use judgement to get good coverage of features
  - Tester must know how to come up with edge and corner cases
  - Tester need more experience than when following a test plan!

# How to do Exploratory Testing

1. Use your best judgment
2. If in doubt about next step, see Step 1.

- But seriously, there is no test plan and no expected behavior
  - Tester must know instinctively what to test
  - Tester must know instinctively what is a defect (or needs enhancement)
- So where to start?
  - Start from the happy path: tests major features as they are meant to be used (Also gives a chance for tester to learn how to use the program)
  - Branch out to edge / corner cases: apply lessons learned in *Breaking Software*

# Document Your Actions

- At least, file defects / enhancements into bug management system
  - Be as formal and detailed as possible so that bug is reproducible
- When feature starts to stabilize, also start recording your test cases
  - Can form the beginnings of a more formal test plan

# Exploratory Testing Pros

- *Fast*: Can focus on finding defects quickly
  - Without having to follow minutiae of test plans and record test results
  - Don’t need a test plan to begin with!
- *Flexible*: No overhead in updating tests
  - Test plans (or automated tests) require updating on system change
- *Improved tester’s knowledge*
  - Exploration helps tester learn about system faster than scripted testing
  - Tester can reapply knowledge to better test the program!

# Exploratory Testing Pros / Cons

- *Unregulated*: Quality depends heavily on tester
  - Since there is no test plan, quality of test will not be uniform
- *Unrepeatable*: Defects not be reproducible
  - May not be able to reproduce since exact steps were not written down
- *Unknown coverage*: Hard to tell coverage after testing
  - Don’t have traceability matrix, so no way to tell what features were covered
- *Hard to automate*: Needs a human being to do it, as of now

# Now Please Read Textbook Chapters 10-11