http://robertvbinder.com/wp-content/uploads/rvb-pdf/talks/GTAC-2010-Binder-Testability.pdf
https://www.youtube.com/watch?v=1keyEiJHqPw
https://www.youtube.com/watch?v=XcT4yYu_TTs

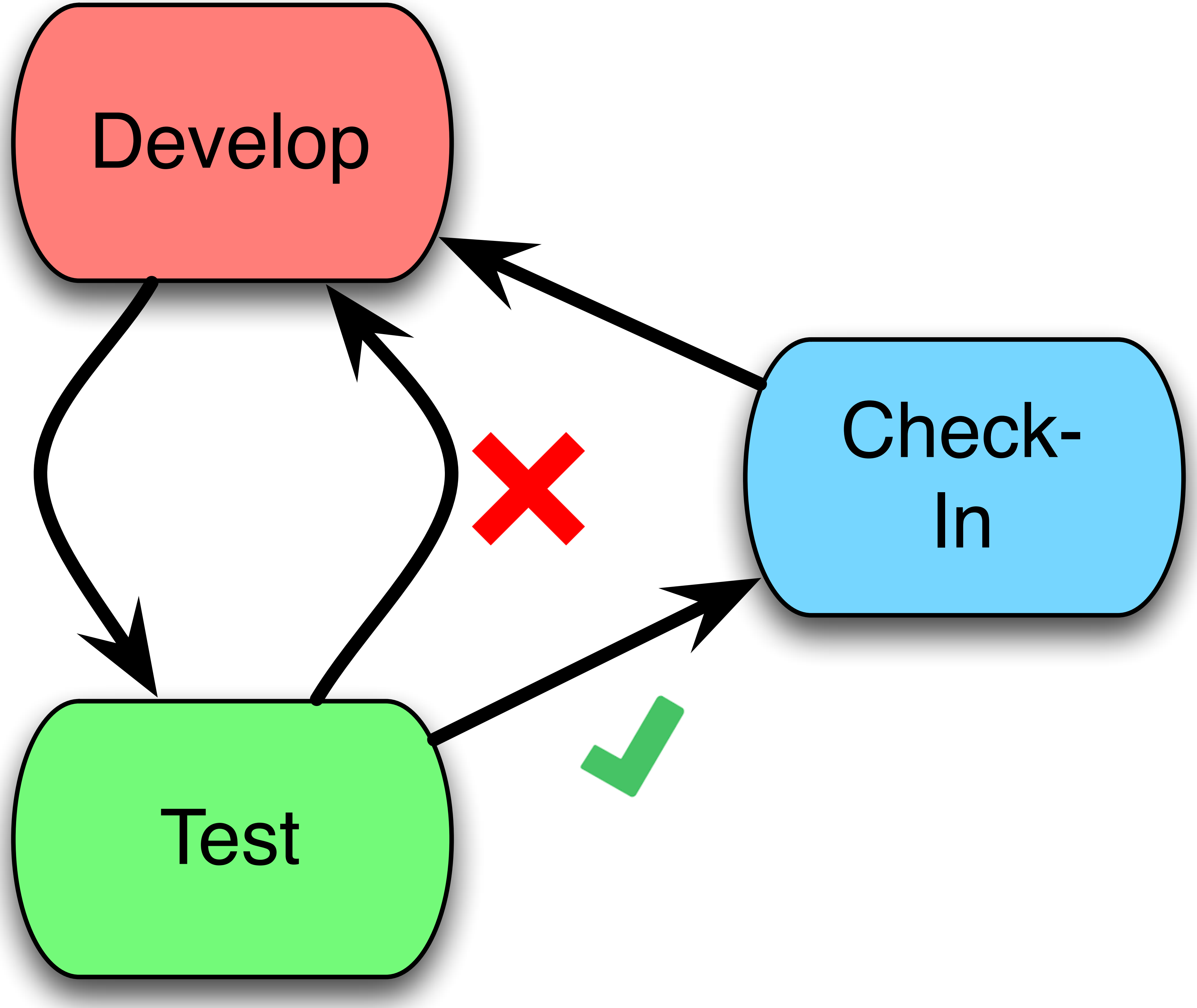
Testability

Reid Holmes

Testability

The degree to which a system or component facilitates the establishment of test objectives and the execution of tests to determine whether those objectives have been achieved.

> Given a finite amount of time and resources, how can we validate that the system has an acceptable risk of costly or dangerous defects.
>
> —Bob Binder


Develop
Check-In
Test
✗
✓

Why not test?

- Good reasons:
  - I don’t know how!
  - Legacy code
- Bad reasons:
  - Bad design
  - Doesn’t catch bugs (now)
  - Slow
  - Boring
  - Hard to change
  - That’s QA’s job

Common assumption

- “The cost of fixing faults rises exponentially with how late [e.g., requirements, design, implementation, deployment] they are detected.”
  - This is commonly stated but is based on evidence from 20+ years ago.
  - This assumption does not seem to hold for modern processes, tools, and languages.
- That said, it is still necessary to validate that the system works.
  - Also important that it continues to work as the system evolves.

Terminology

- Efficiency: number of tests per unit of effort.
- Effectiveness: the probability of detecting a bug per unit of effort.
    - Higher testability: more effective tests, same cost.
    - Lower testability: fewer weaker tests, same cost.
- Repeatability: the likelihood that running the same test twice will yield the same result.
- SUT/CUT: System/Code Under Test
- White-box: tests consider internals of CUT.
- Black-box: tests are oblivious of internals of CUT.

http://robertvbinder.com/wp-content/uploads/rvb-pdf/talks/GTAC-2010-Binder-Testability.pdf

Anatomy of a test

- To reveal a fault, a test must:
  - Reach some code
  - Trigger a defect
  - Propagate an incorrect result
  - The result must be observed
  - The result must be interpreted as incorrect
- Test threats:
  - Non-deterministic dependencies
  - Threading/Race Conditions/Deadlock
  - Shared data

Properties of Testability

- Controllability
  - The extent to which the SUT can be made to perform specific actions of interest.
- Observability
  - The extent to which the response of the SUT to a test can be verified.
- Isolateability
  - The degree to which the element under test can be validated on its own.
- Automatability
  - The ability to execute the test programmatically.

Controllability

- What do we have to do to run a test?
- How expensive is it?
- Does the SUT make running a test **impractical**?
- Give a test goal, do we have enough **information** to create an adequate suite?
- How much tooling can we afford?

Observability

- What do we have to do to identify pass/fail?
- How expensive is it to do this?
- Can we **extract** the result from the SUT?
- Do we know enough to **identify** pass/fail?

Isolateability

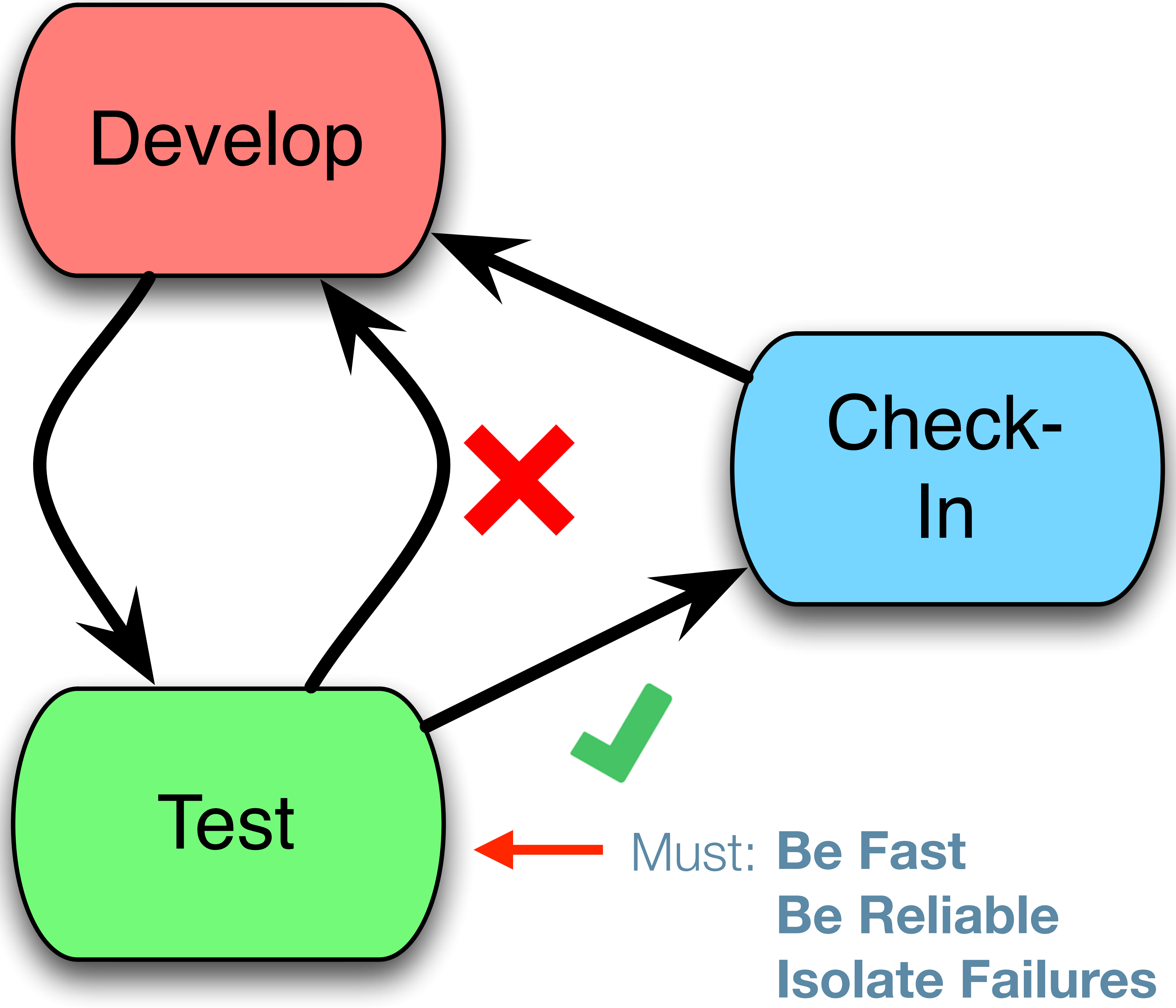
- Can the element being tested be isolated?
- What is the cost to do this?
- If an element cannot be naturally isolated, can we **simulate** it (e.g., with mocks / stubs)?
- Why bother?
- Isolated components are:
  - Simpler to reason about (e.g., root cause analysis)
  - Less prone to non-determinism
  - Faster
- Simulated dependencies can also more enable validating unusual states.

Automatability

- Can tests be executed **without** human intervention?
- Huge economic advantages:
  - Setting up automation: 5 hours
  - Running manual test: 30 minutes.
  - Automation pays for itself after just 10 iterations.
- What is the cost of automated infrastructure?
- What is the benefits of using a test infrastructure?
  - Executions can be batched.
  - Run on same configuration / hardware.
  - Global visibility of results.
- Enables regression testing.

Challenges

- Tests are code too.
  - Also subject to their own faults.
- Not all test failures uncover faults:
  - Defect in test itself
  - Flaky test (due to some form of non-determinism)
  - Requirements shortcoming (undefined behaviour)
    - Implicit assumptions often surfaced by tests
- How to retrospectively recognize a ‘true’ failure?
  - Developer changed the source (not test), test passed on the next iteration.

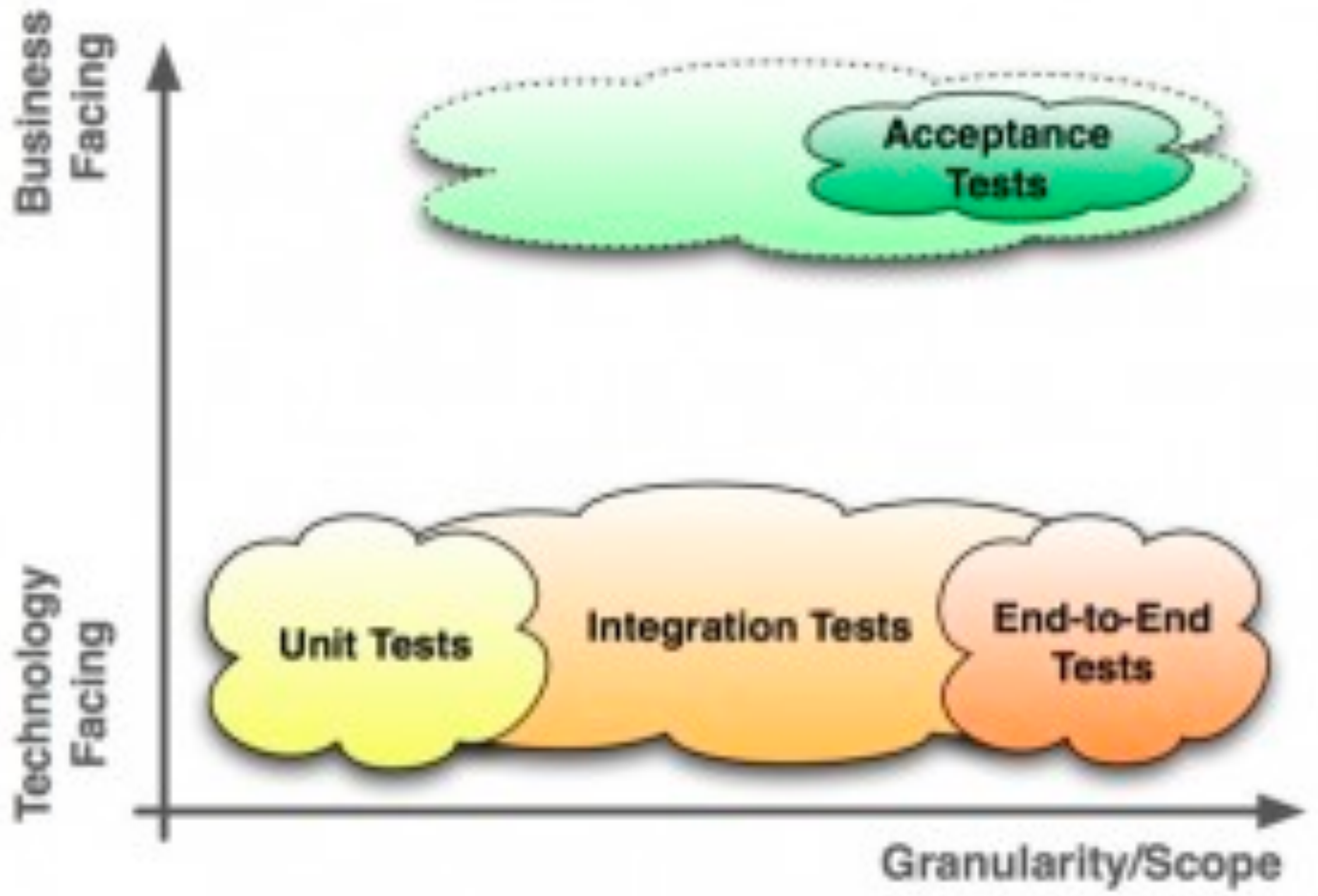


Must: **Be Fast**

Be Reliable

Isolate Failures

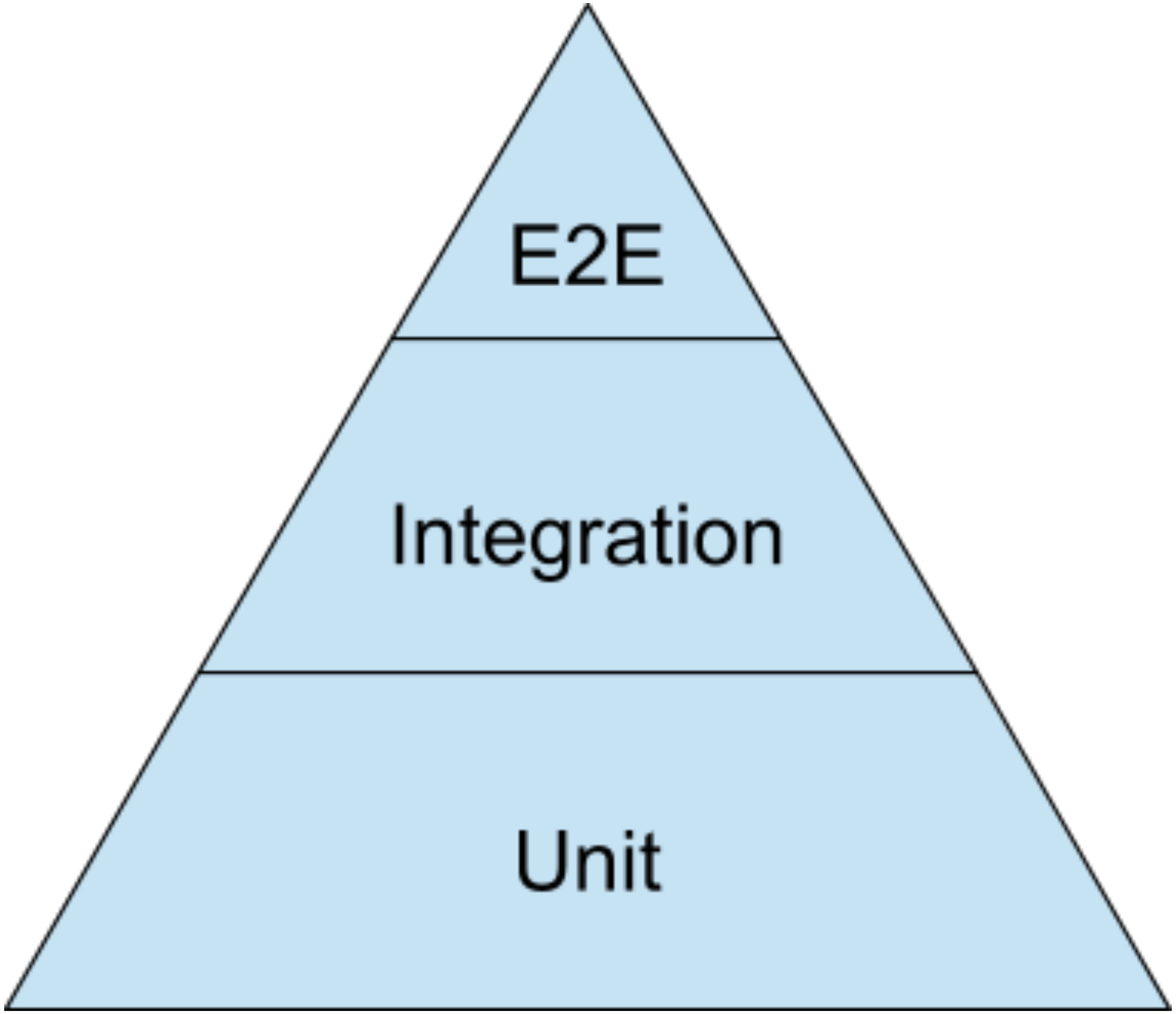
Kinds of tests



Test value

	Unit	End-toEnd
Fast	☺	☹
Reliable	☺	☹
Isolates Failures	☺	☹
Simulates a Real User	☹	☺

Test pyramid



http://googletesting.blogspot.co.uk/2015/04/just-say-no-to-more-end-to-end-tests.html

Continuous Integration

- Every project will build and execute automated tests.
- TravisCI (https://travis-ci.com/ (NOT ORG)) has provided the class with large numbers of commits
  - You can request access online, we will grant it

Activity

- In your groups:
  - Choose one of your use cases
  - Describe one end to end test for this use case
  - Describe 2 integration tests for this use case
  - Describe 3 unit tests for this use case