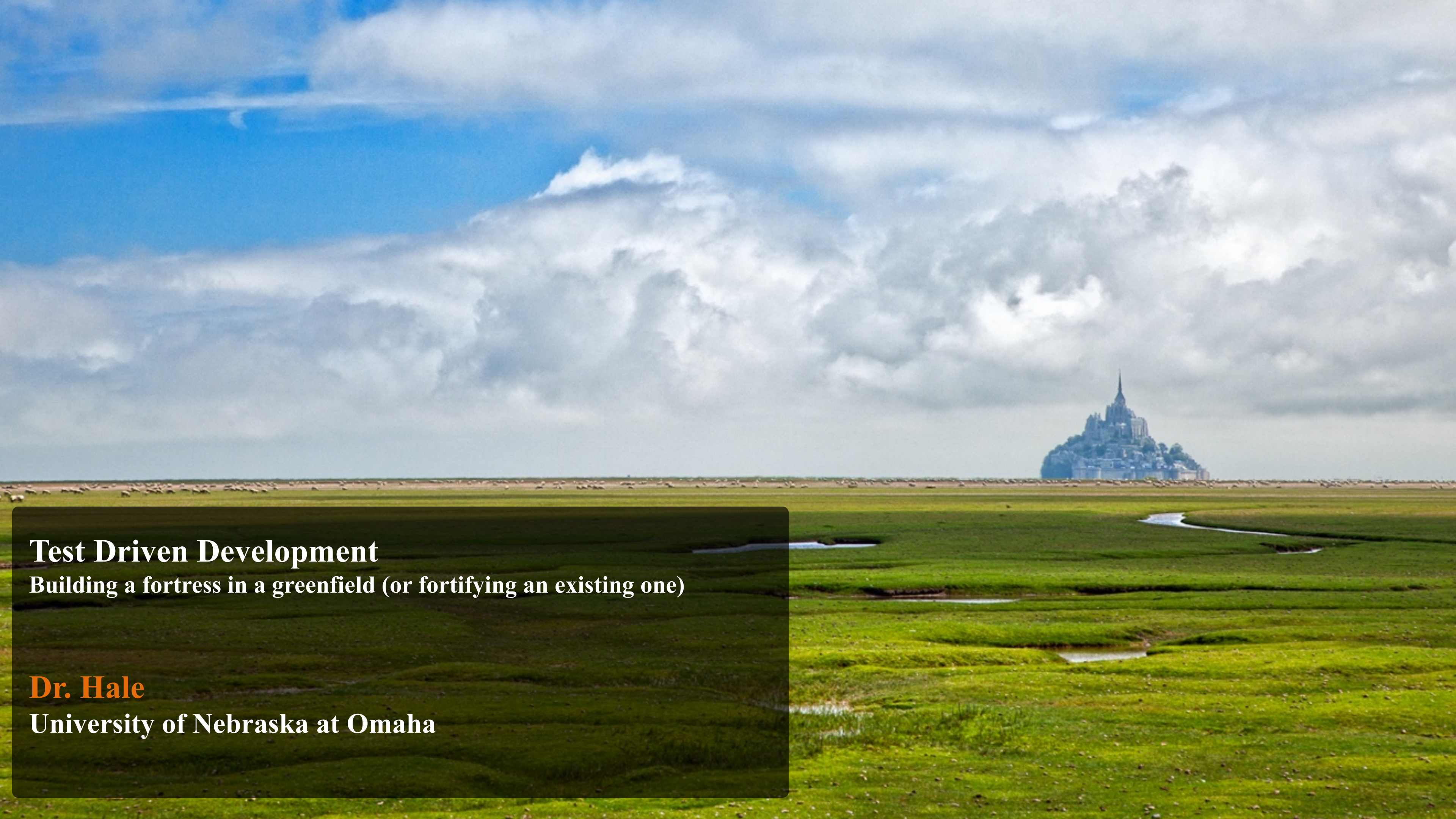


Test Driven Development

Building a fortress in a greenfield (or fortifying an existing one)

Dr. Hale
University of Nebraska at Omaha

Today's topics:

Software Testing and Test driven development

- Unit / integration / acceptance testing
- Think-test-build-test-repeat

Blackbox and Whitebox testing

Vulnerability surface and testing strategies

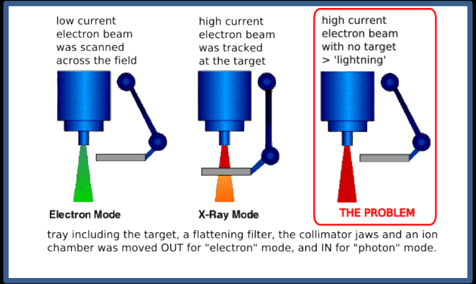
Test-driven Development

Some Material from Bernd Bruegge and Allen Dutoit Object-Oriented SE: Using UML, Patterns, and Java
(because their slides are hilarious)

Famous Problems

- F-16 : crossing equator using autopilot
  - Result: plane flipped over
  - Reason?
    - Reuse of autopilot software

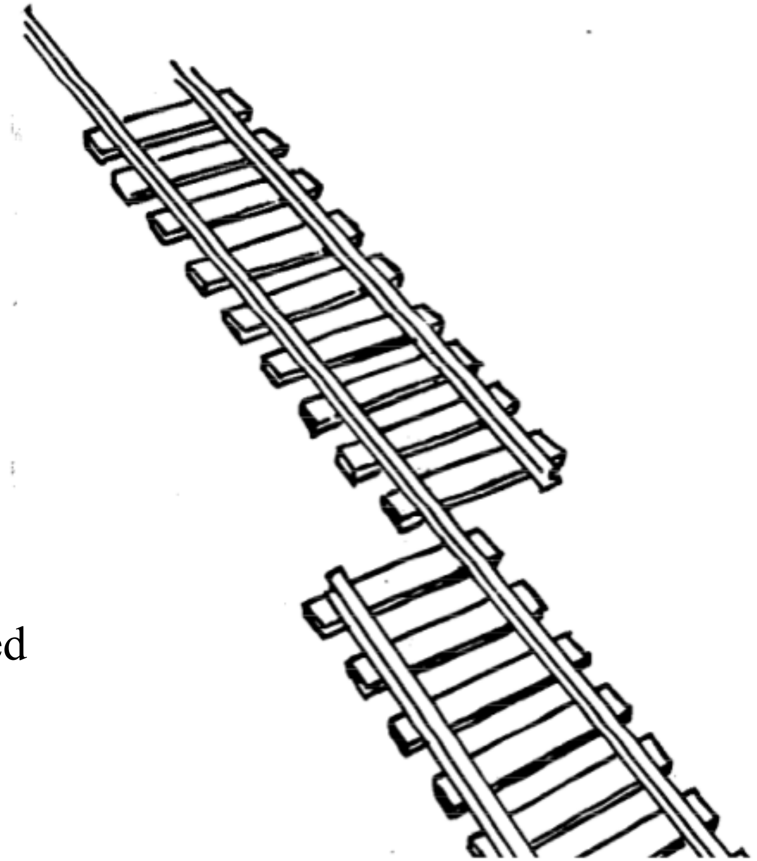
- The Therac-25 accidents (1985-1987), one of the most serious non-military computer-related failure in terms of human life (at least five died)
  - Reason: Bad event handling in the GUI
- NASA Mars Climate Orbiter destroyed due to incorrect orbit insertion (September 23, 1999)
  - Reason: Unit conversion problem.


low current electron beam was scanned across the field
high current electron beam was tracked at the target
high current electron beam with no target > 'lightning'
Electron Mode
X-Ray Mode
THE PROBLEM
tray including the target, a flattening filter, the collimator jaws and an ion chamber was moved OUT for "electron" mode, and IN for "photon" mode.

Terminology

- Failure: Any deviation of the observed behavior from the specified behavior
- Erroneous state (error): The system is in a state such that further processing by the system can lead to a failure
- Fault: The mechanical or algorithmic cause of an error (“bug”)
- Validation/testing: Activity of checking for deviations between the observed behavior of a system and its specification.

What is this?



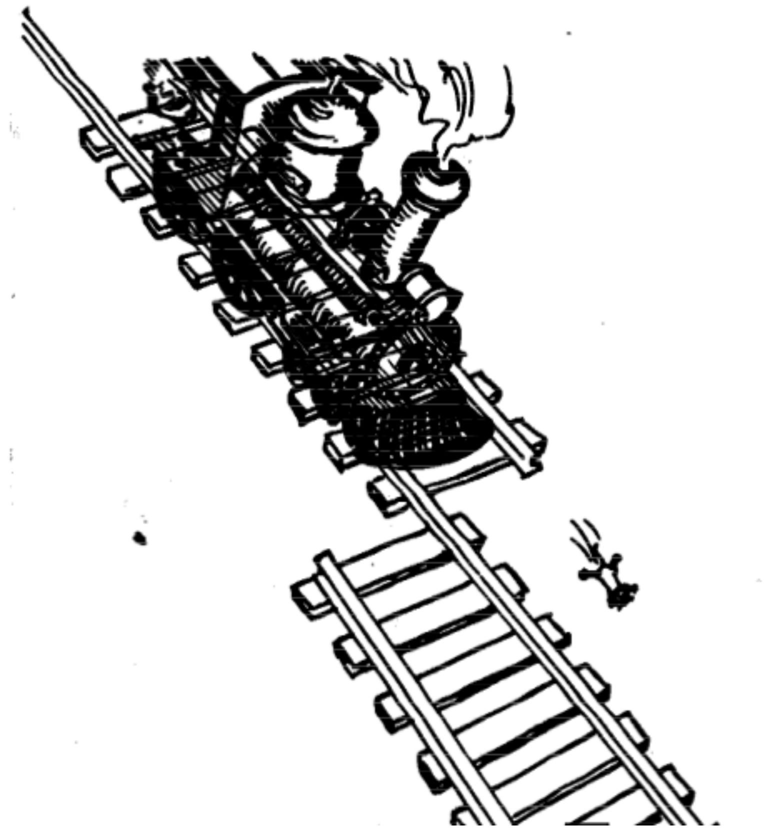
A failure?

An error?

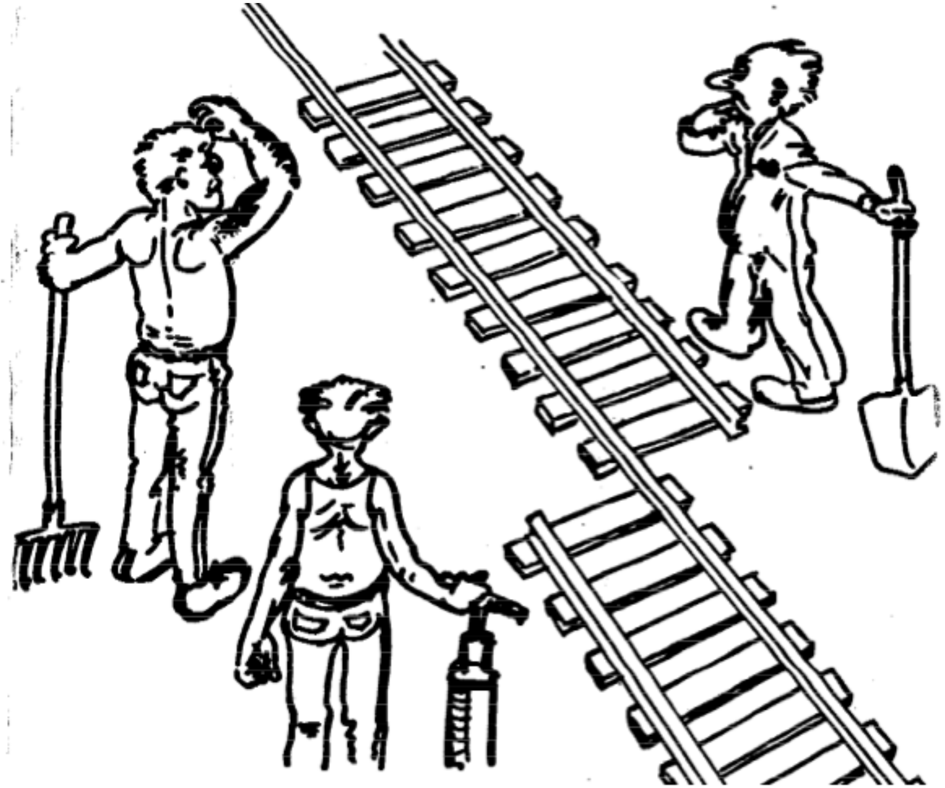
A fault?

We need to describe specified and desired behavior first!

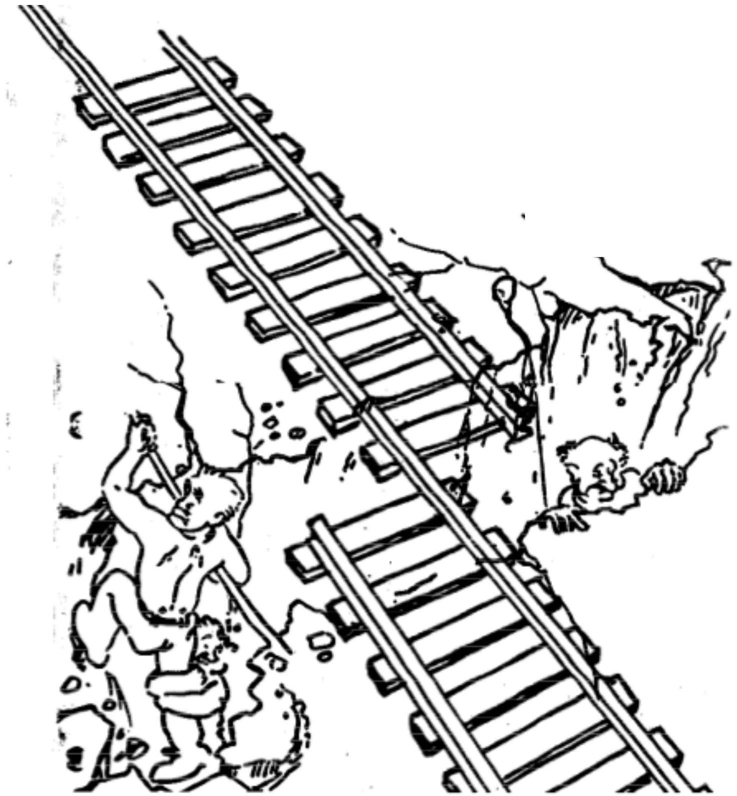
Erroneous State (“Error”)



Algorithmic Fault

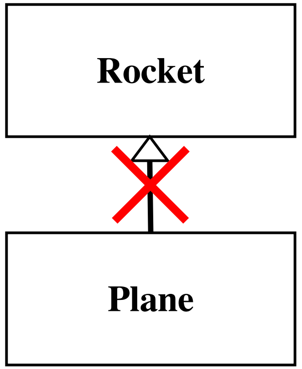


Mechanical Fault



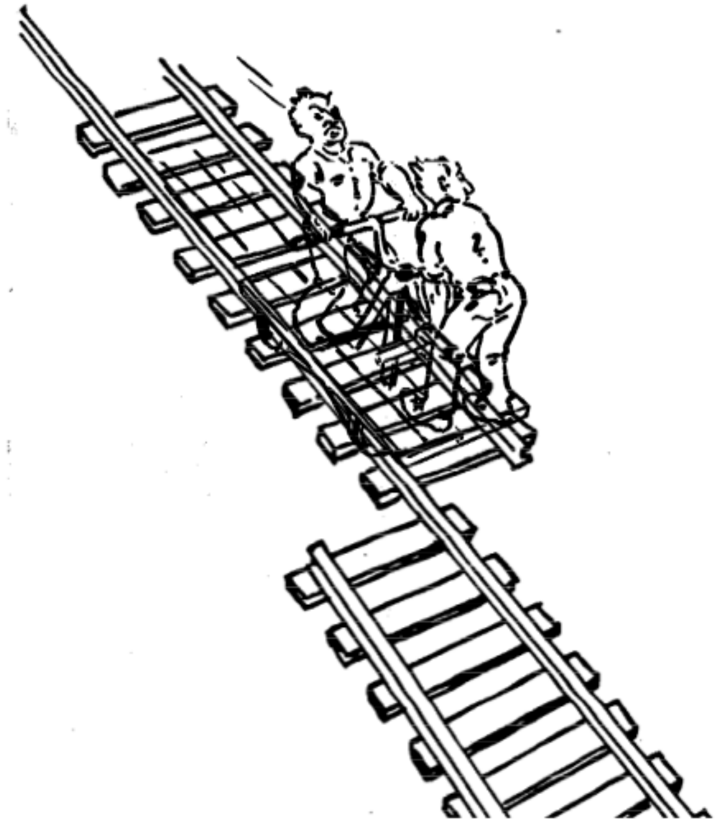
F-16 Bug

- What is the failure?
- What is the error?
- What is the fault?
  - Bad use of implementation inheritance
  - A Plane is **not** a rocket.

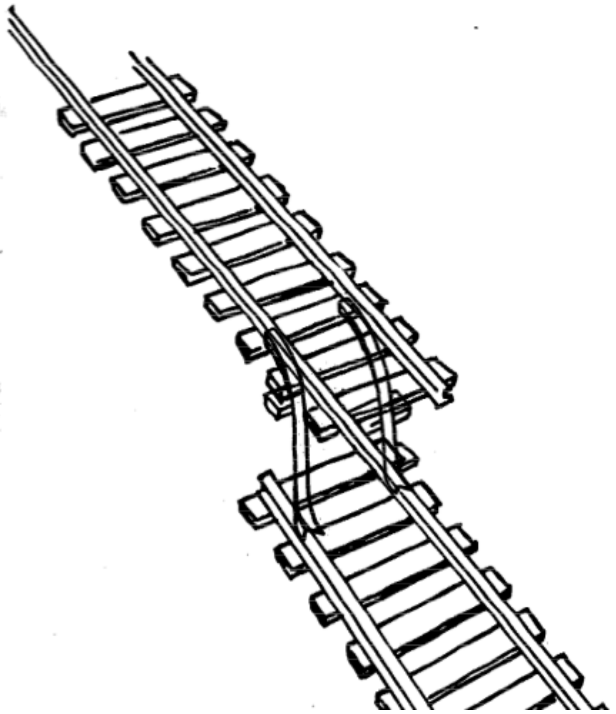


How do we deal with Errors, Failures and Faults?

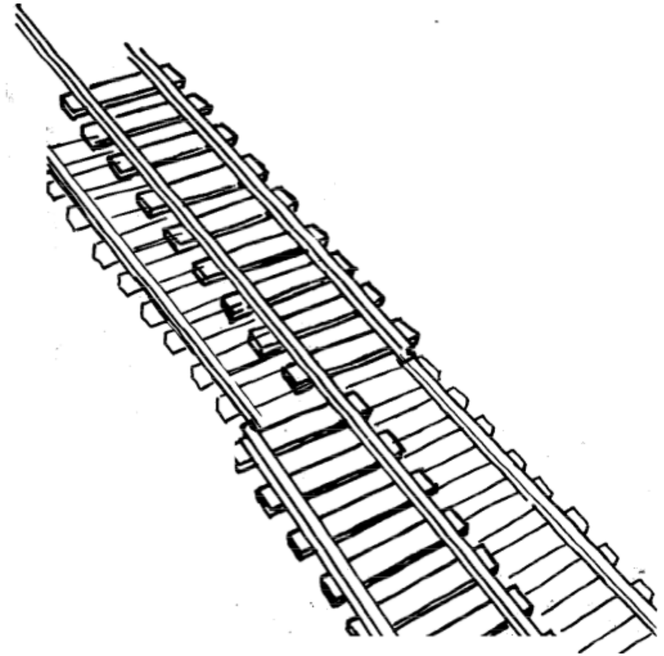
Testing



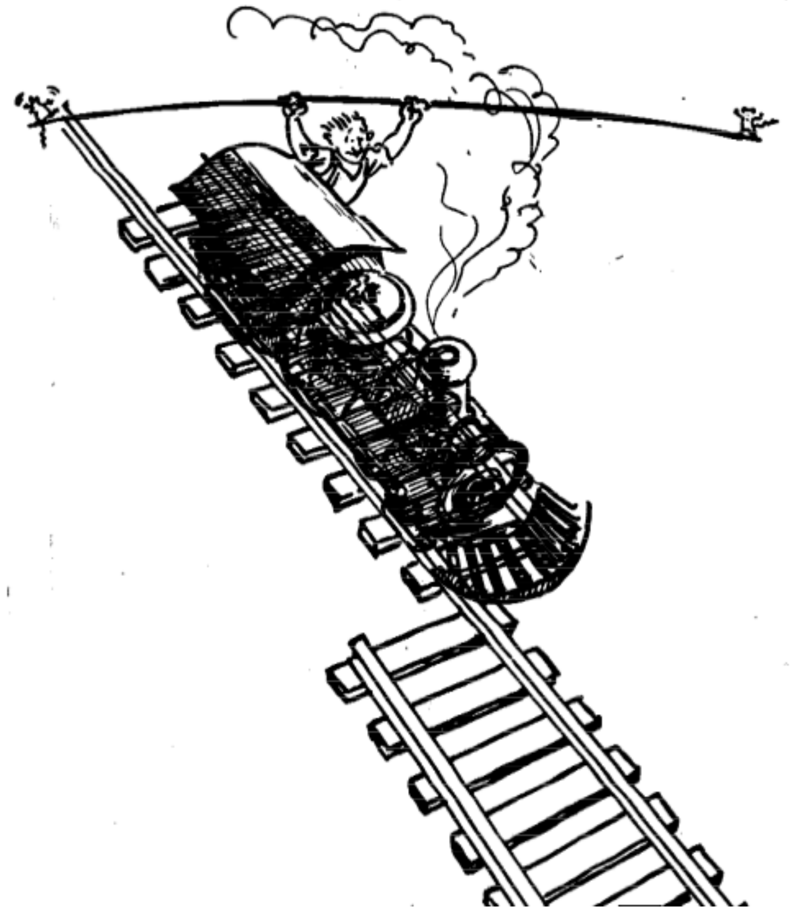
Patching



Building Modular Redundancy



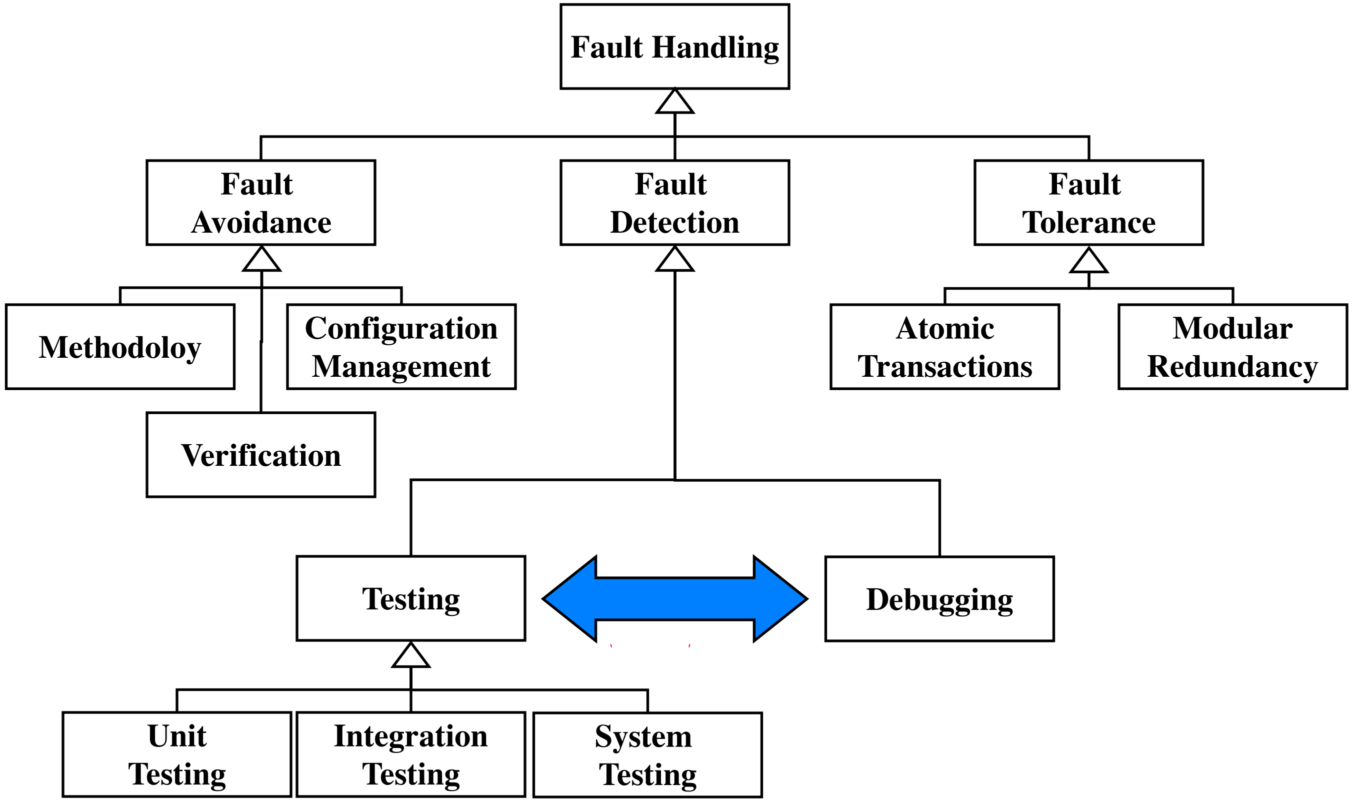
Declaring the Bug as a Feature



Another View on How to Deal with Faults

- Fault avoidance
  - Use methodology to reduce complexity
  - Use configuration management to prevent inconsistency
  - Apply verification to prevent algorithmic faults
  - Use Reviews
- Fault detection
  - Testing: Activity to provoke failures in a planned way
  - Debugging: Find and remove the cause (Faults) of an observed failure
  - Monitoring: Collecting and Delivering information about state => Used during debugging
- Fault tolerance
  - Exception handling
  - Modular redundancy.

Taxonomy for Fault Handling Techniques

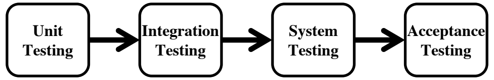


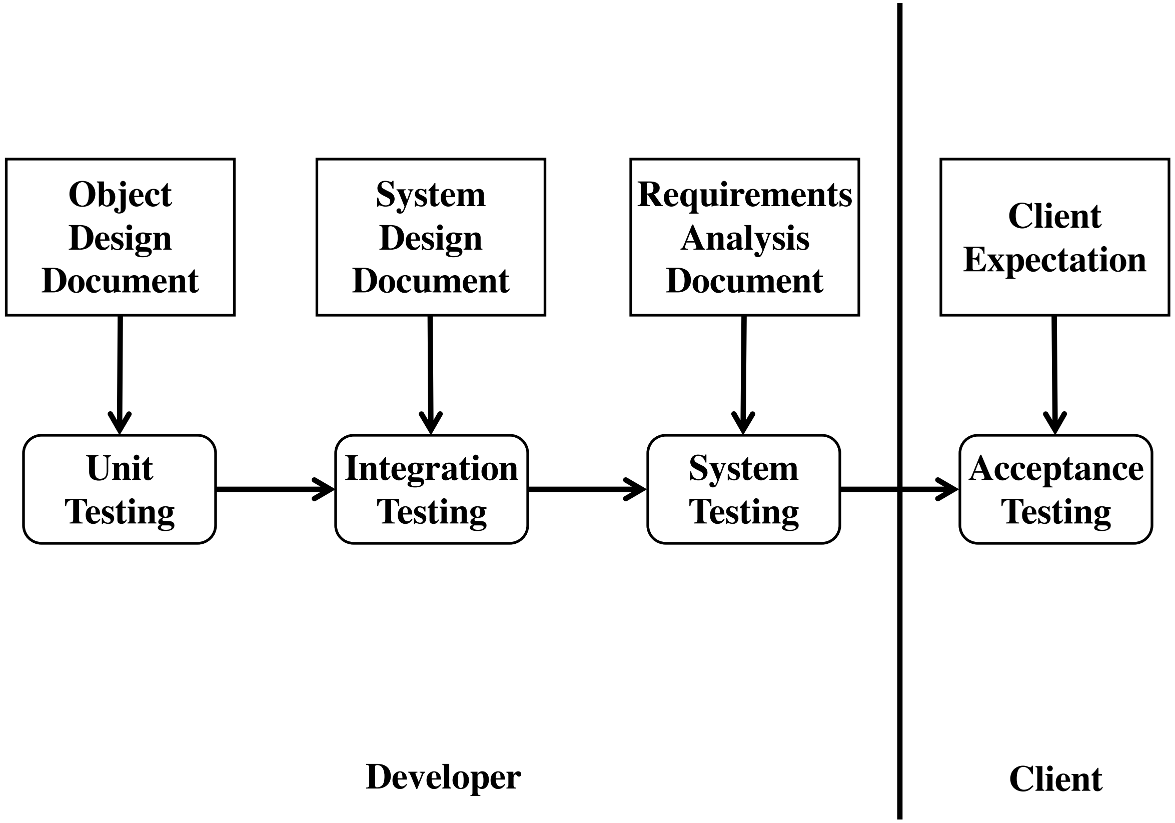
Observations

- It is impossible to completely test any nontrivial module or system
  - Practical limitations: Complete testing is prohibitive in time and cost
  - Theoretical limitations: e.g. Halting problem
- "Testing can only show the presence of bugs, not their absence" (Dijkstra).
- Testing is not for free

=> Define your goals and priorities

Testing Activities





Types of Testing

Acceptance Test – A measure that ensures that a feature meets functional demands. Usually acceptance tests are tied to user stories or use cases.

Unit test – A smaller test that ensures isolated chunks of functionality (known as units) are functional and operating as expected.

Integration tests – Between unit tests and acceptance tests. Focuses on ensuring that different units function together (said to be integrable).

UNIT Testing

Can be done manually or programmatically – want to define them programmatically since your components may change and manually testing each time is onerous

Basically you boil down exactly what a feature or component should be doing and you logically state these criteria. Each time you modify the feature/component you run the unit tests to see if they pass. When they all pass you move on to integration tests.

Integration Testing

Can be done manually or programmatically

Here you define how different components need to interact and state those constraints logically. When all of the integration tests work – it means you move on to acceptance tests and make sure the collected components satisfy the original goals in the user story or use cases.

Acceptance Testing

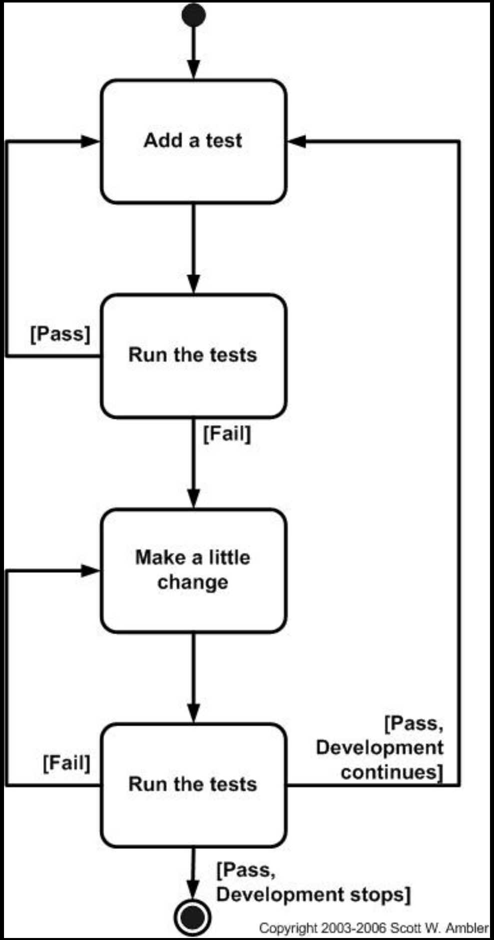
Can be done manually or programmatically – often the former

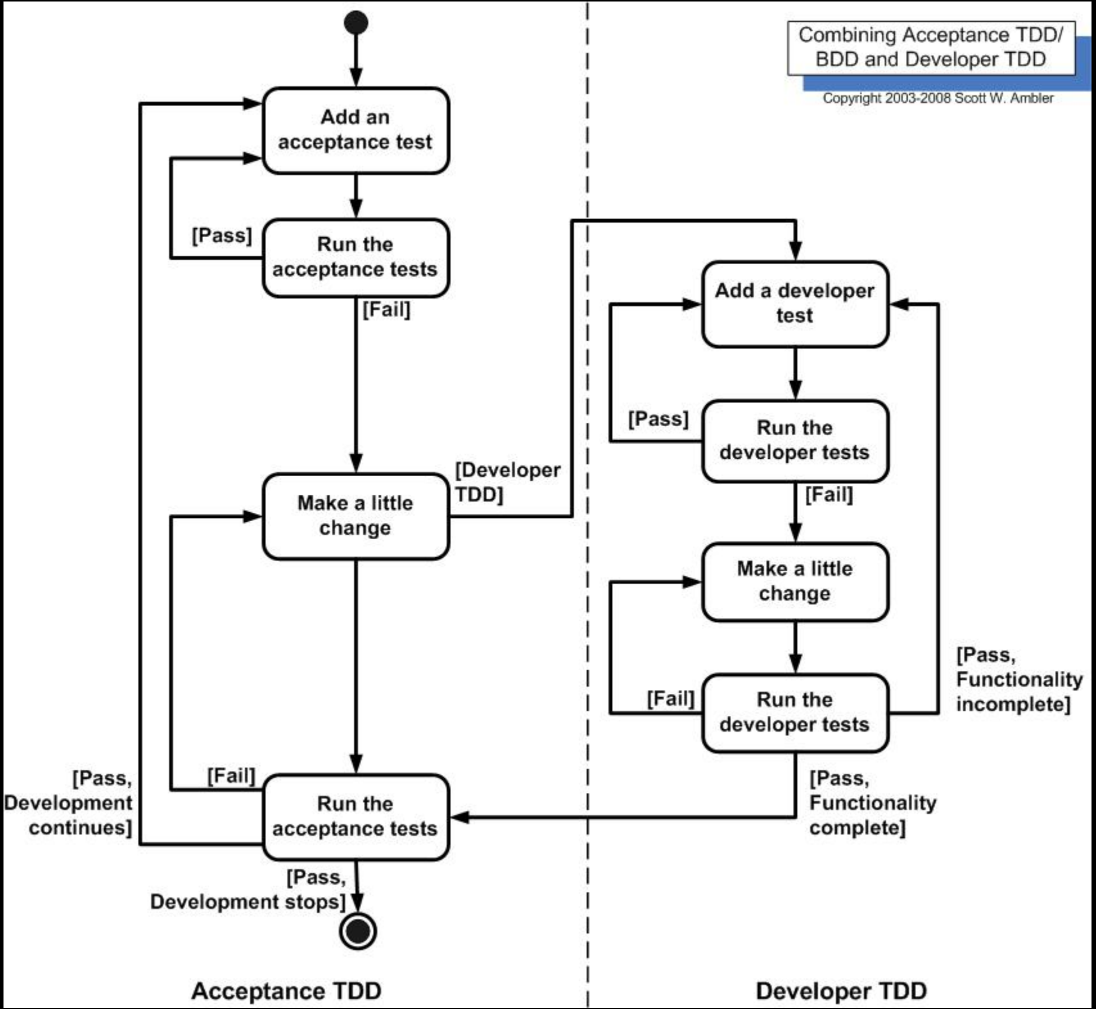
You basically define the set of all acceptance tests related to your user stories and use cases and – when you demonstrate the app passes all of the tests you are done!

Penetration Testing

Pen tests are either unit or integration tests. Most are unit tests. They are integration if they involve evoking multiple separable components at once. A pen test seeks to identify failure conditions that violate security requirements by causing errors. The goal is to identify and mitigate faults that lead to these errors, through patching.

Test Driven Development Core Philosophy


Add a test
[Pass]
Run the tests
[Fail]
Make a little change
[Fail]
Run the tests
[Pass, Development continues]
[Pass, Development stops]
Copyright 2003-2006 Scott W. Ambler


Combining Acceptance TDD/ BDD and Developer TDD
Copyright 2003-2008 Scott W. Ambler
Add an acceptance test
Run the acceptance tests
[Pass]
[Fail]
Make a little change
[Developer TDD]
Add a developer test
Run the developer tests
[Pass]
[Fail]
Make a little change
Run the developer tests
[Fail]
[Pass, Functionality incomplete]
[Pass, Functionality complete]
Run the acceptance tests
[Fail]
[Pass, Development continues]
[Pass, Development stops]
Acceptance TDD
Developer TDD

Blackbox and Whitebox testing

Blackbox Testing

Testing a component, feature, or system without knowledge of the inner workings of the entity.

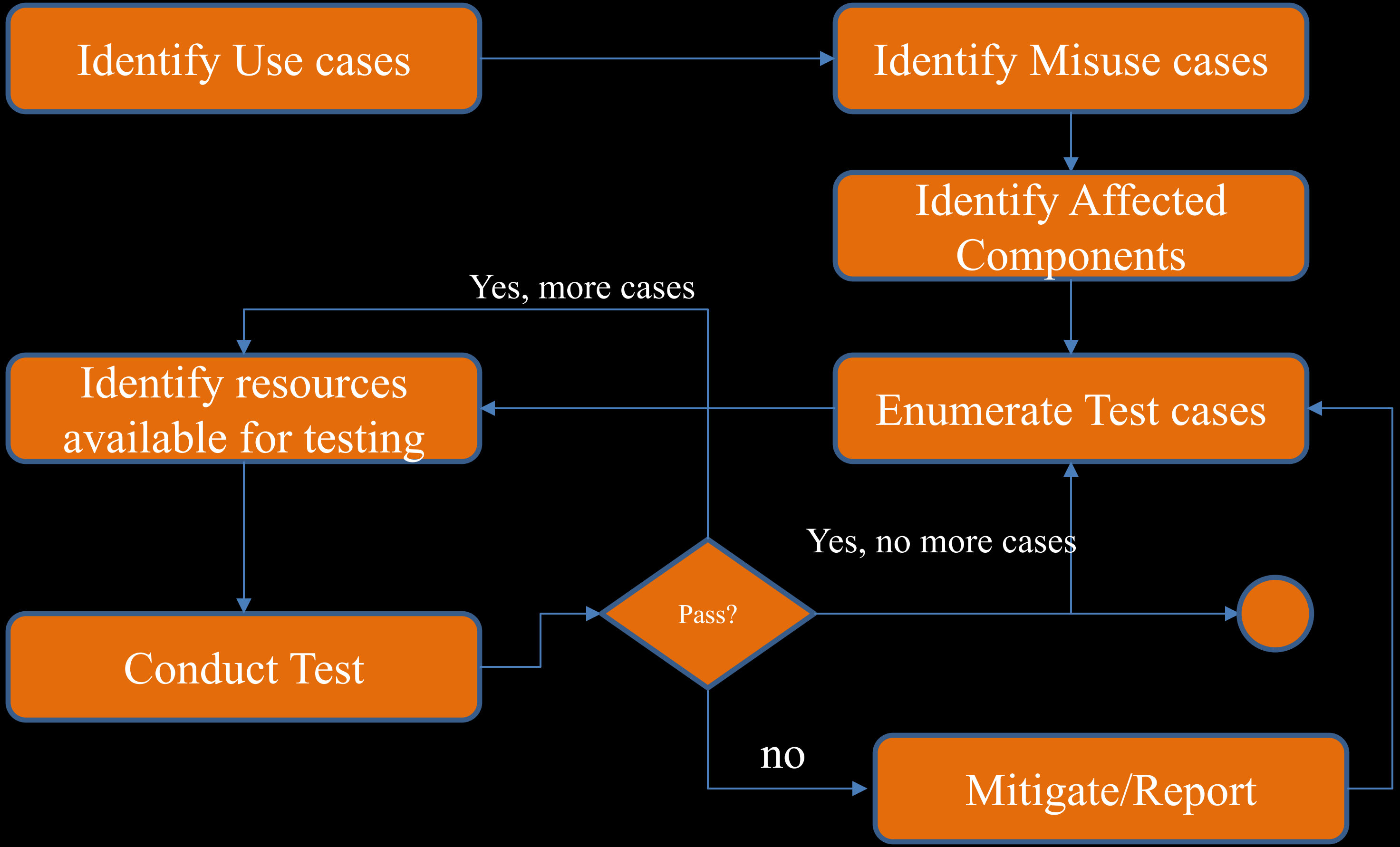
Whitebox Testing

Testing a component, feature, or system with knowledge of the inner workings of the entity.

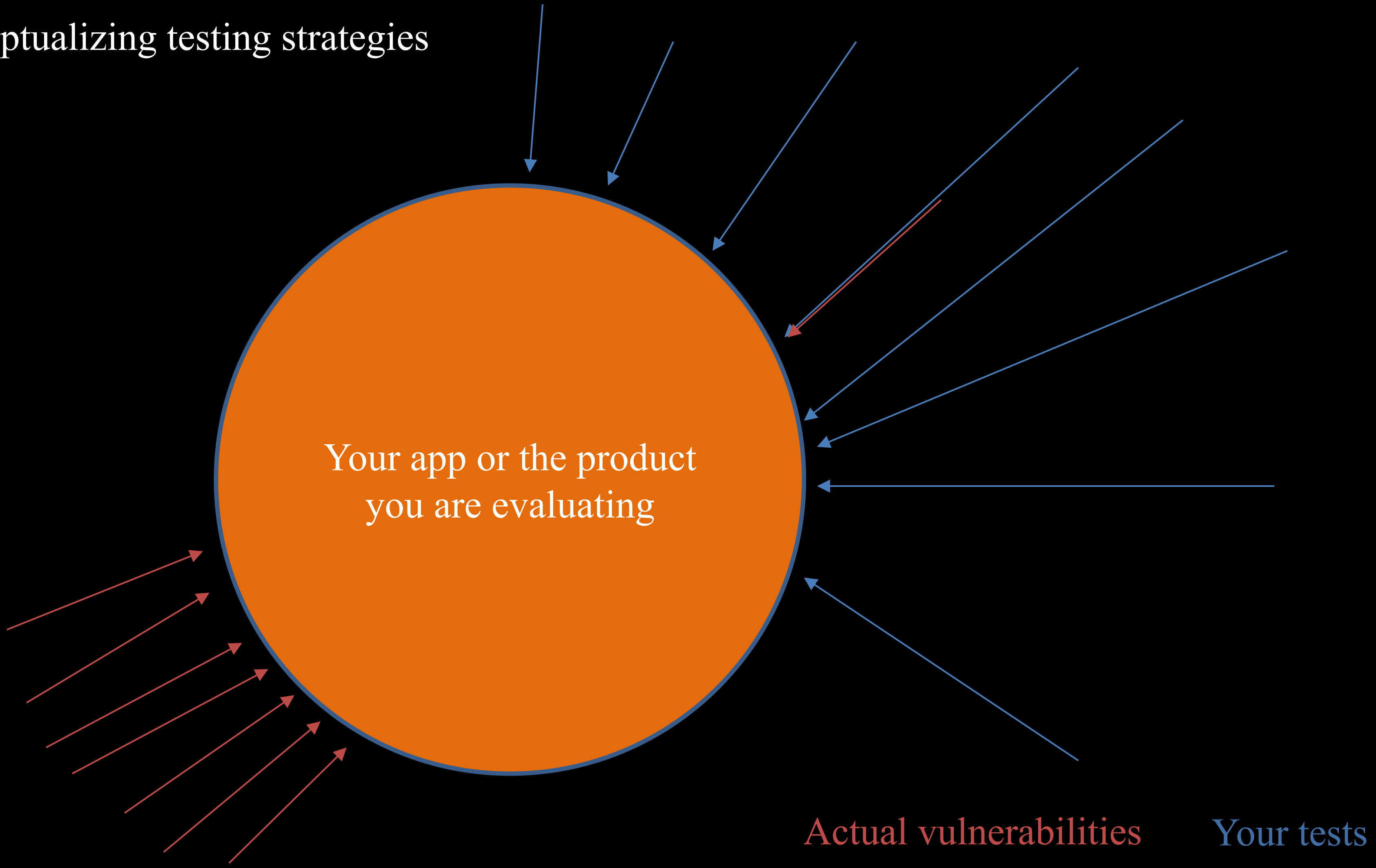
Same basic idea:
Understand what can go wrong so you can mitigate the problem or vulnerability.

Conducting an Evaluation

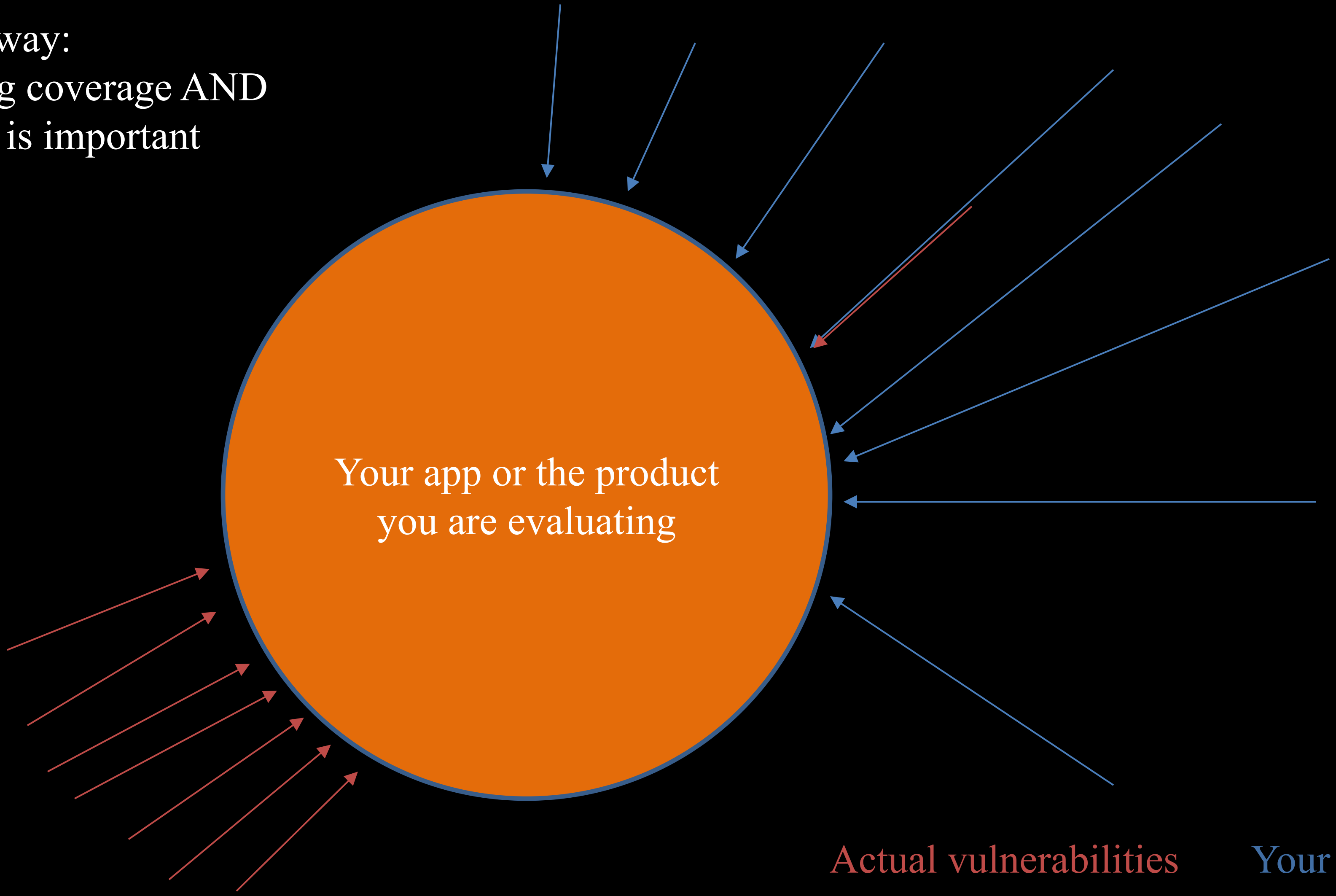
Suggested workflow for security evaluation



Conceptualizing testing strategies

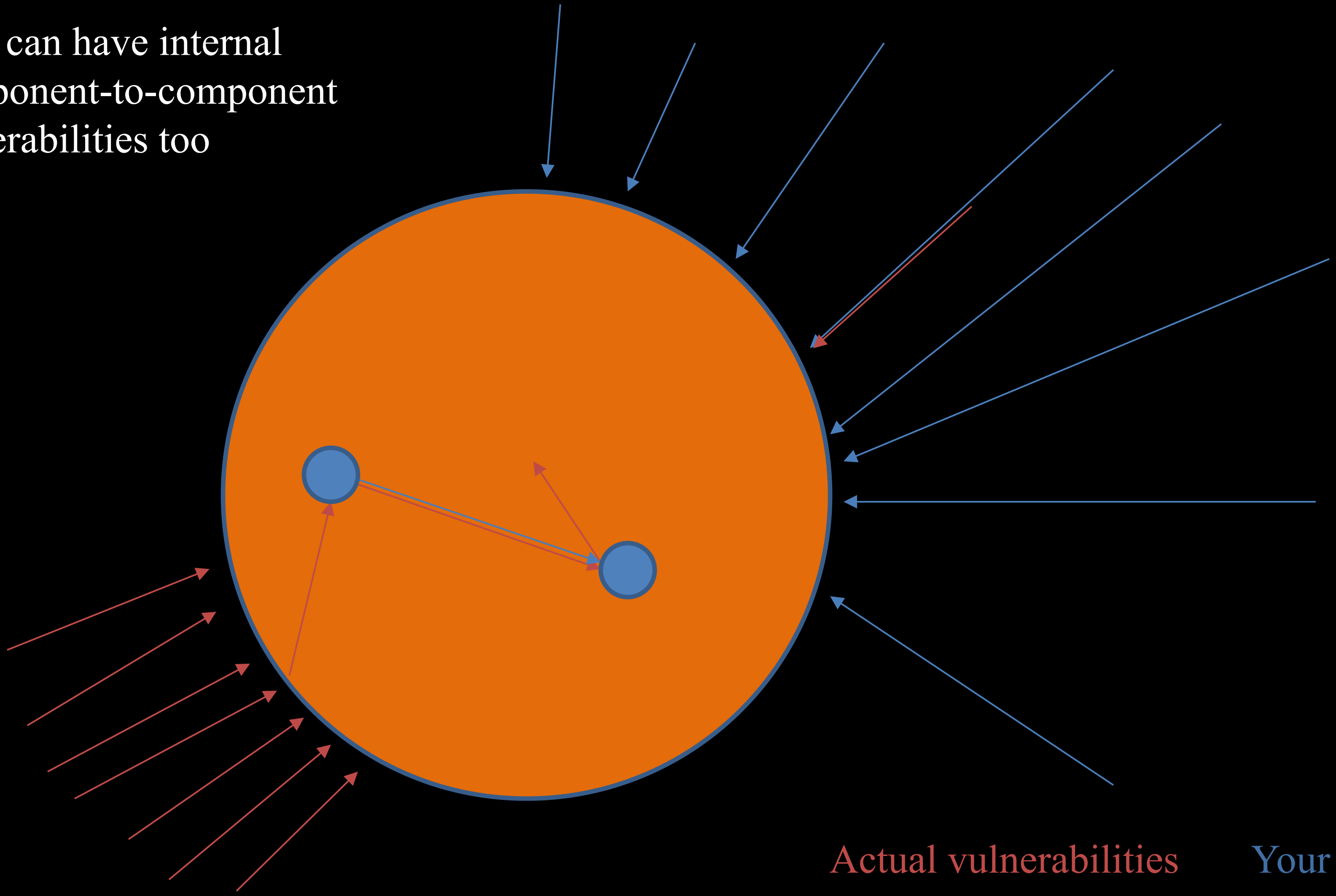


Takeaway:
Having coverage AND
Depth is important



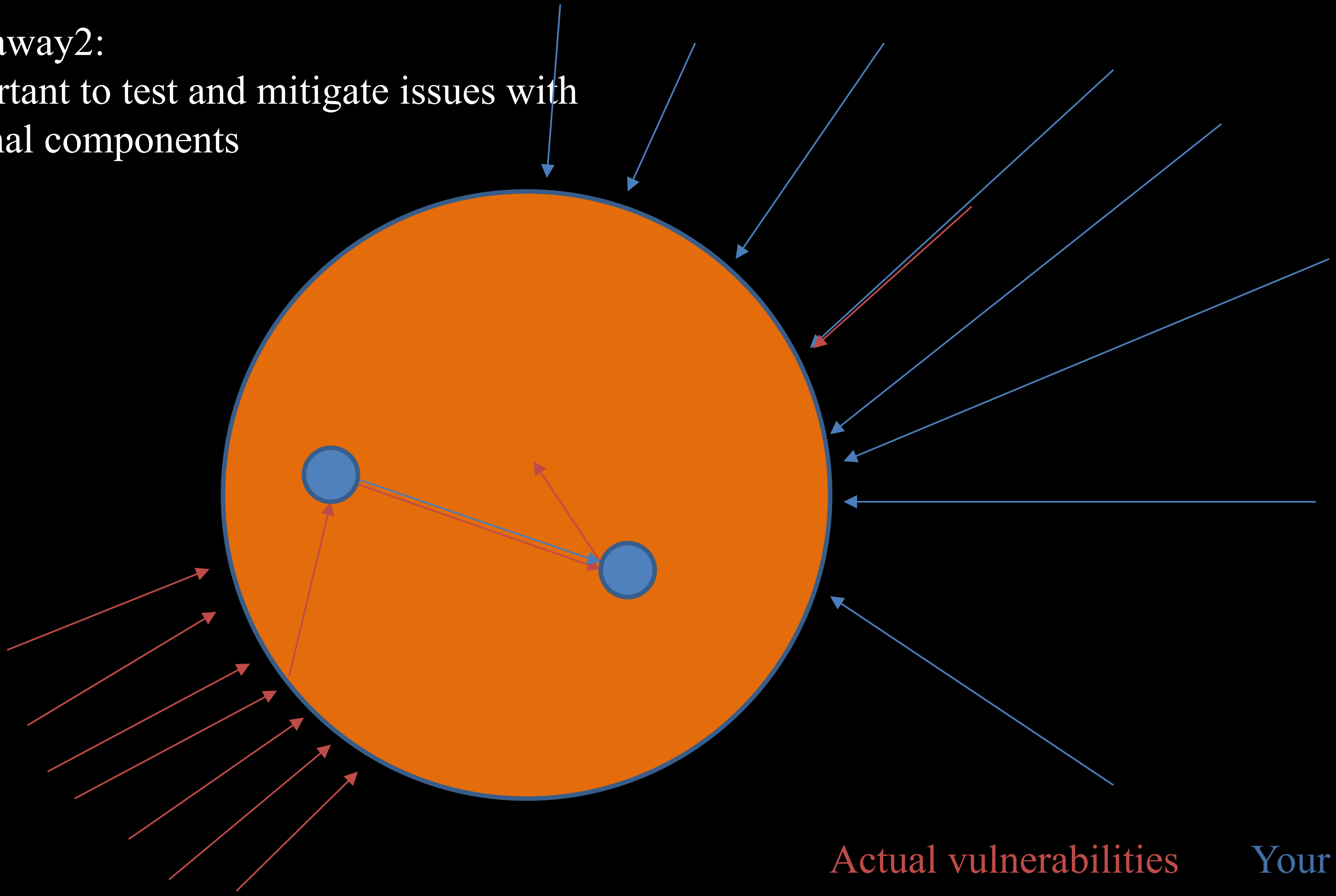
Actual vulnerabilities Your tests

Apps can have internal
Component-to-component
Vulnerabilities too



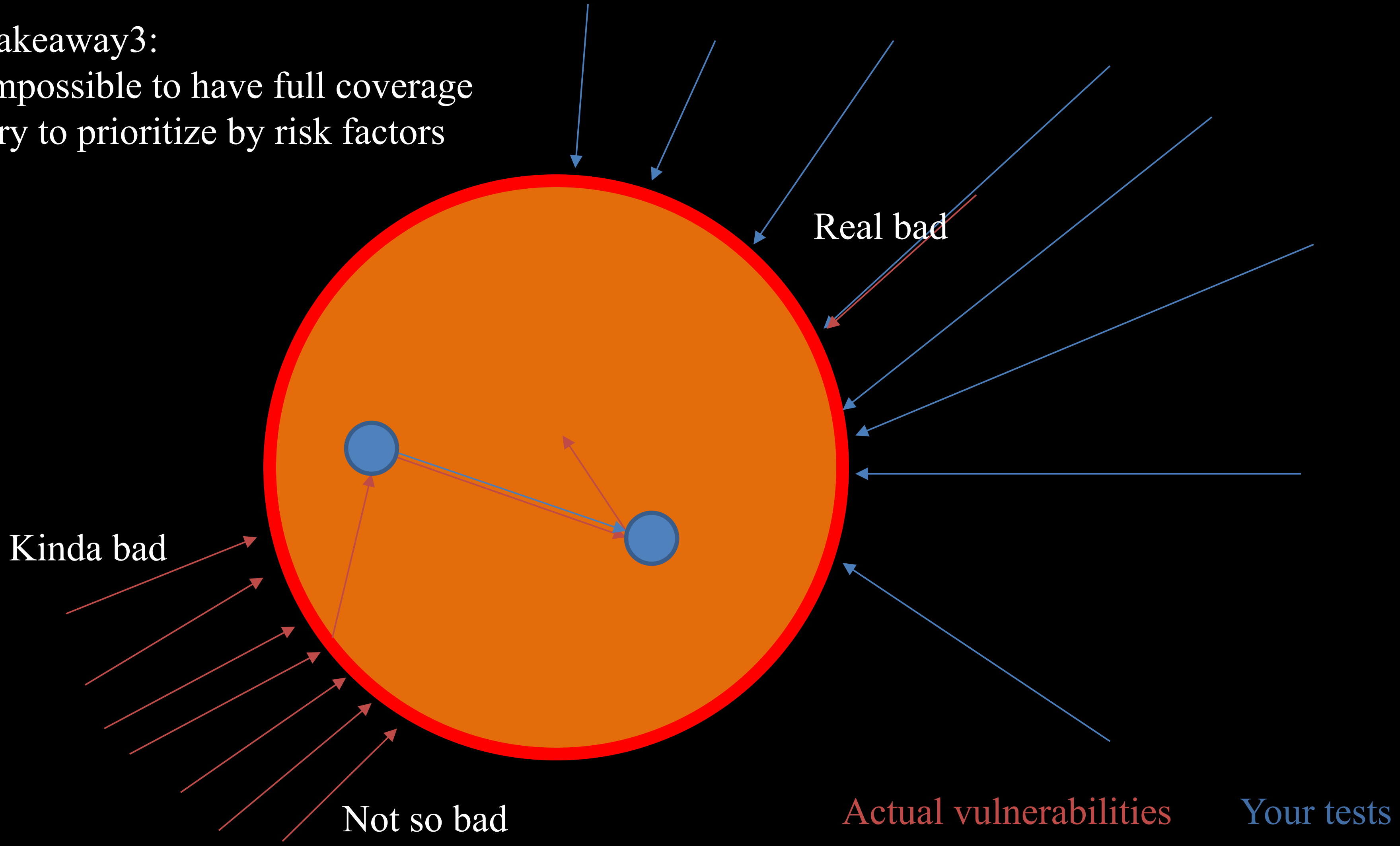
Actual vulnerabilities Your tests

Takeaway2:
Important to test and mitigate issues with internal components

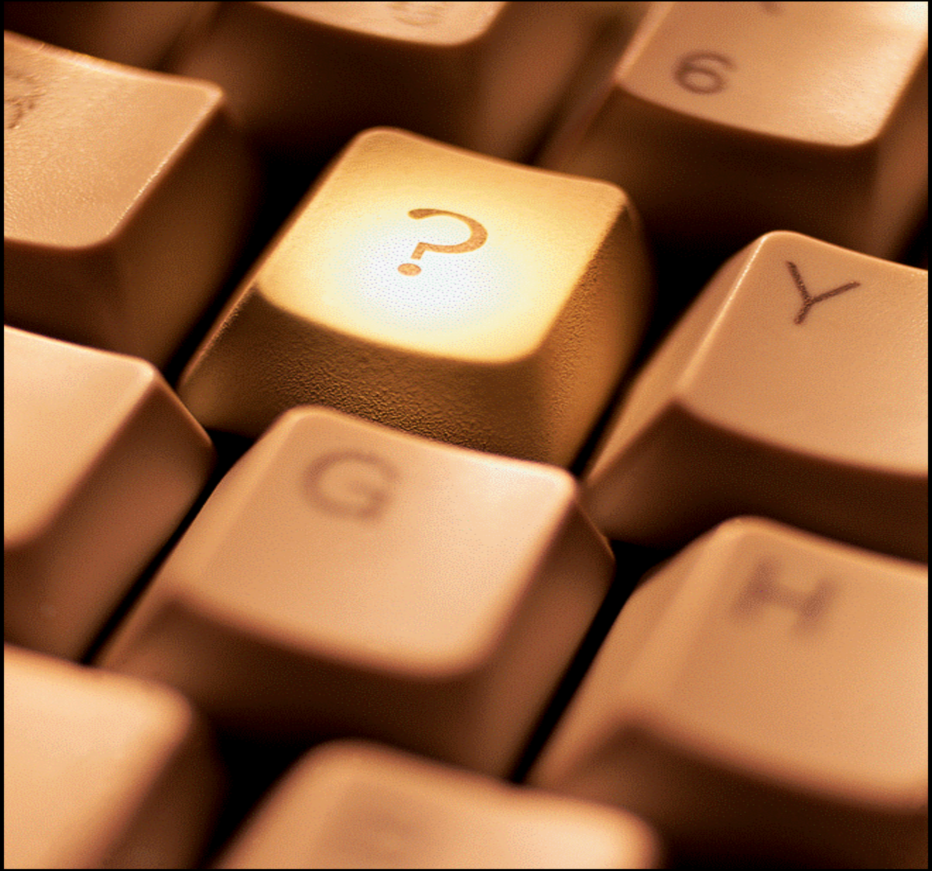


Actual vulnerabilities   Your tests

Takeaway3:
Impossible to have full coverage
Try to prioritize by risk factors



Questions?



Matt Hale, PhD

University of Nebraska at Omaha

Assistant Professor of Cybersecurity

faculty.ist.unomaha.edu/mhale/
mlhale@unomaha.edu

Twitter: @mlhale_