CS250: Introduction to Software Systems

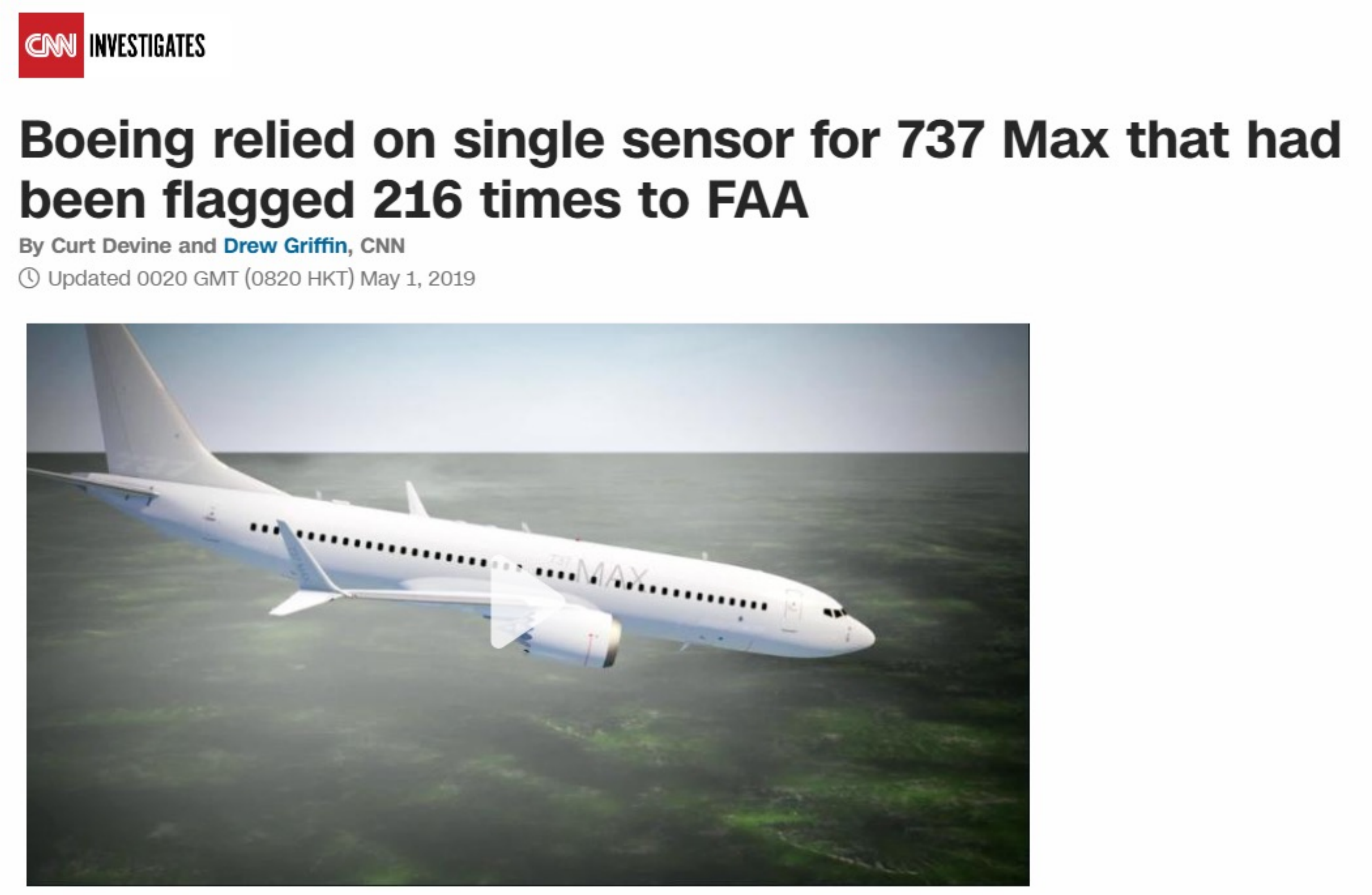
Verification and Validation

Outline

- Verification and Validation

- Testing

Failure



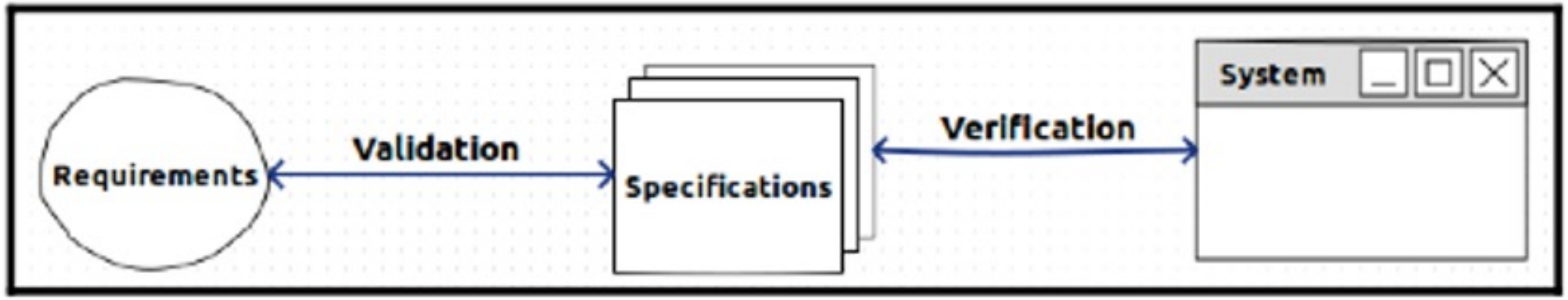
CNN INVESTIGATES

Boeing relied on single sensor for 737 Max that had been flagged 216 times to FAA

By Curt Devine and Drew Griffin, CNN

Updated 0020 GMT (0820 HKT) May 1, 2019

Definitions



Verification

- implementation meets specification
- internal testing
- “are we building the product right?”

Validation

- implementation meets (client) requirements
- user feedback
- “are we building the right product?”

Verification Methods

- Testing
- Static analysis
- Code reviews and inspection
- Formal verification

Testing

- Exercise software to try and make it fail
  - given input and program (test case)
  - success: expected output == actual output
  - failure: expected output != actual output
  - set of test cases is test suite
- Pros:
  - no false positives
- Cons:
  - incomplete (huge input domain)

Static Analysis

- Consider all possible inputs/behavior (complete) without execution
  - generate task graph
  - identify classes of errors, e.g., nullptr dereference
- Pros:
  - complete
- Cons:
  - false positives

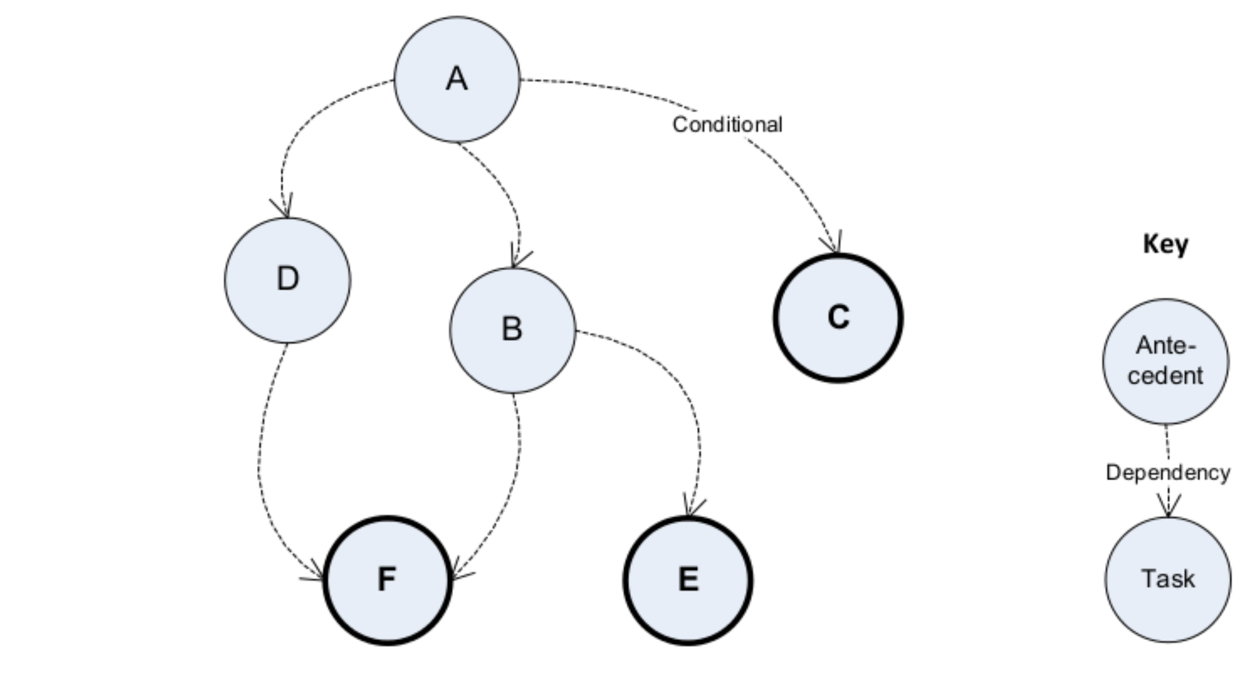


Figure 1: A task graph with dependencies and a conditional dependency.

Inspection

- Review code and other artifacts manually
  - Human-intensive
- Pros:
  - systematic
  - thorough
- Cons:
  - informal
  - subjective

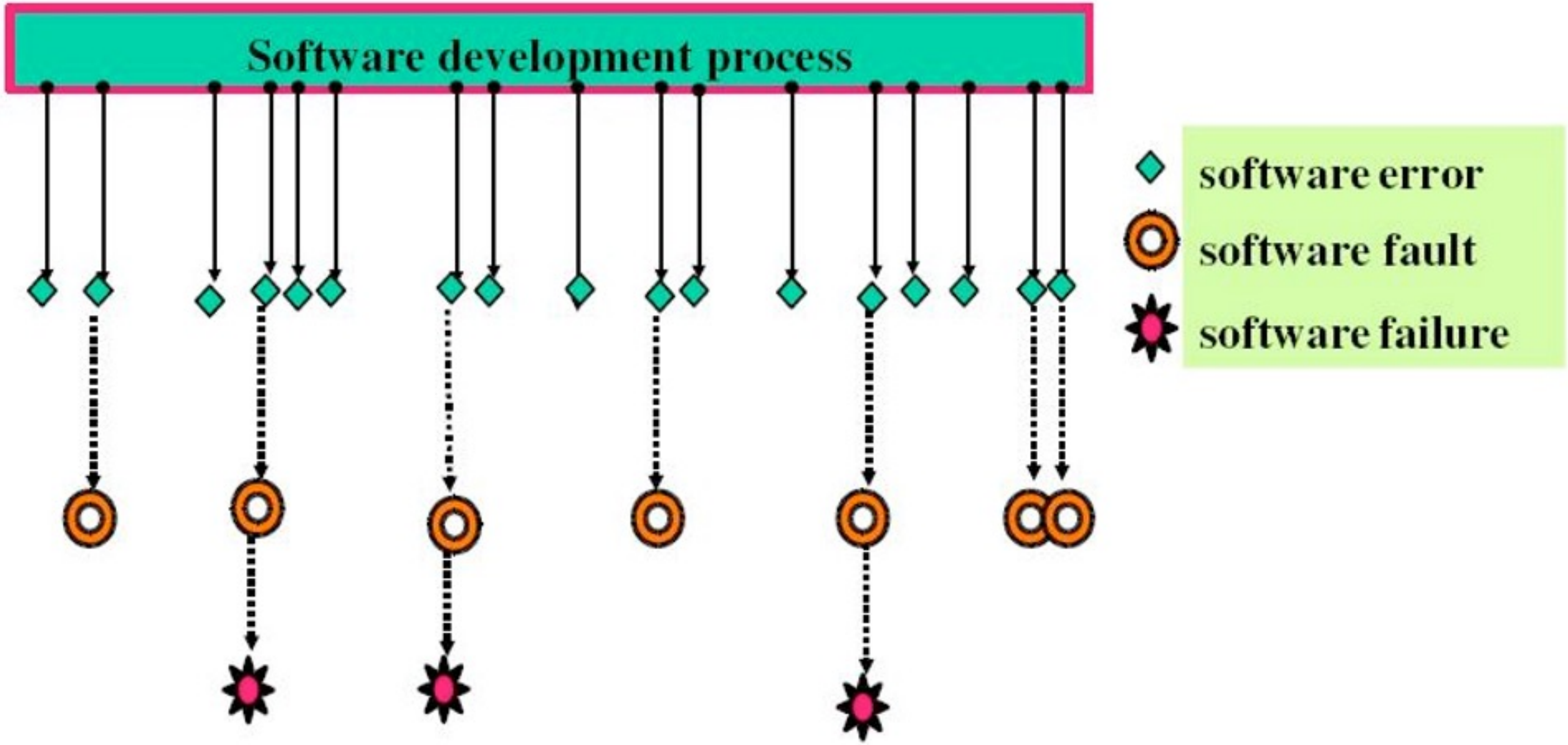
Formal Methods

- Given a formal specification of expected behavior, use formal method to prove implementation satisfies specification
- Pros:
  - guarantees
- Cons:
  - complex

Testing Definitions

- Failure
  - observable incorrect behavior
  - inability to perform specified function
- Fault
  - incorrect code
  - i.e., bug
  - can cause failure
- Error
  - (human) action that causes incorrect result
  - can cause fault

Error/Fault/Failure



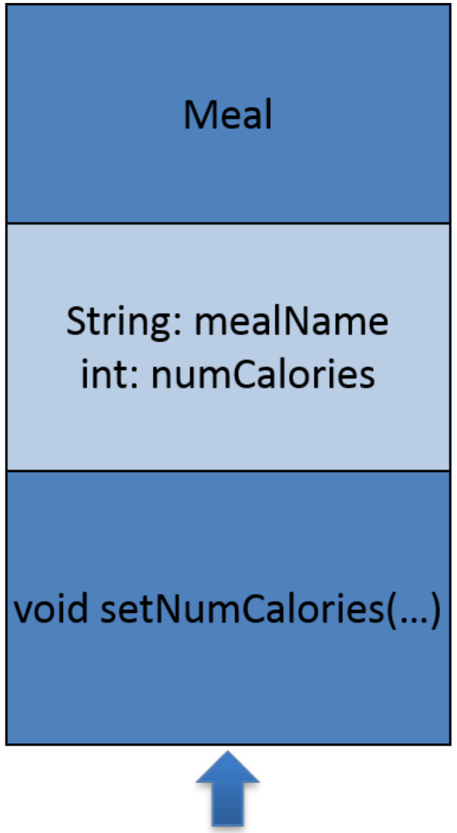
Testing Goals

- Detect failures/faults/errors
- Locate failures/faults/errors
- Fix failures/faults/errors
- Demonstrate correctness
    - w.r.t. design (verification)
    - w.r.t. specification (validation)

Testing Levels

- Unit Testing
  - single code "unit" (e.g., method)
- Integration Testing
  - interfaces among integrated units
- System Testing
  - complete system

Testing Levels



Meal

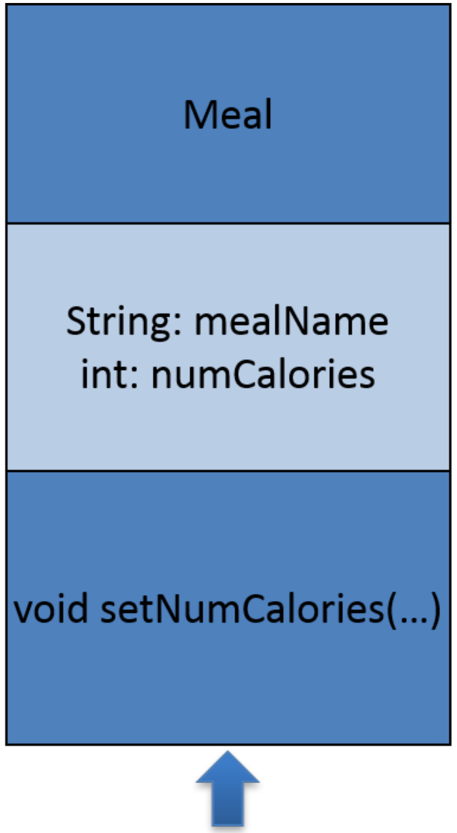
String: mealName
int: numCalories

void setNumCalories(…)

Unit testing

Meal.setNumCalories(…)

Testing Levels



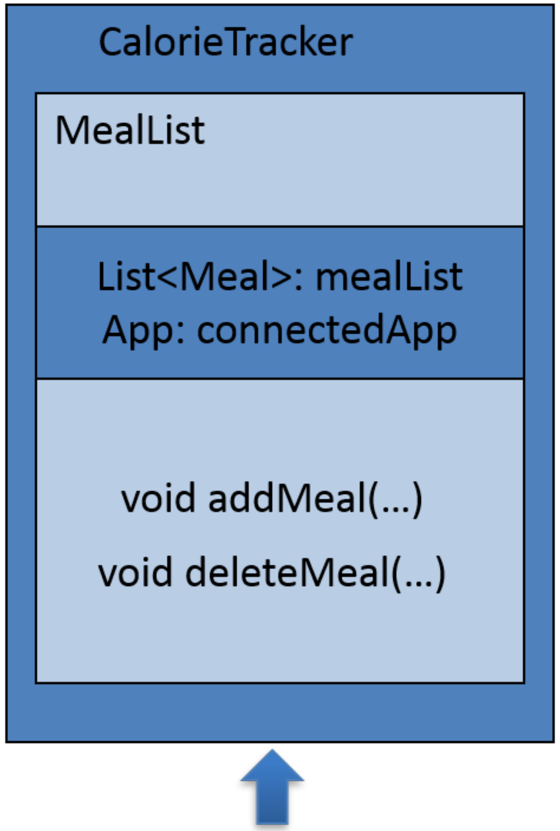
Meal

String: mealName
int: numCalories

void setNumCalories(…)

Unit testing

Meal.setNumCalories(…)



CalorieTracker

MealList

List<Meal>: mealList
App: connectedApp

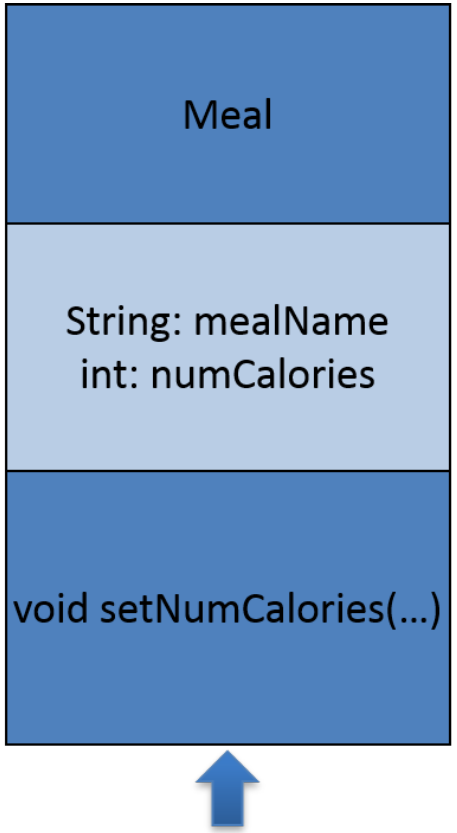
void addMeal(…)

void deleteMeal(…)

Functional/integration testing

CalorieTracker.addMeal(Meal m)

Testing Levels



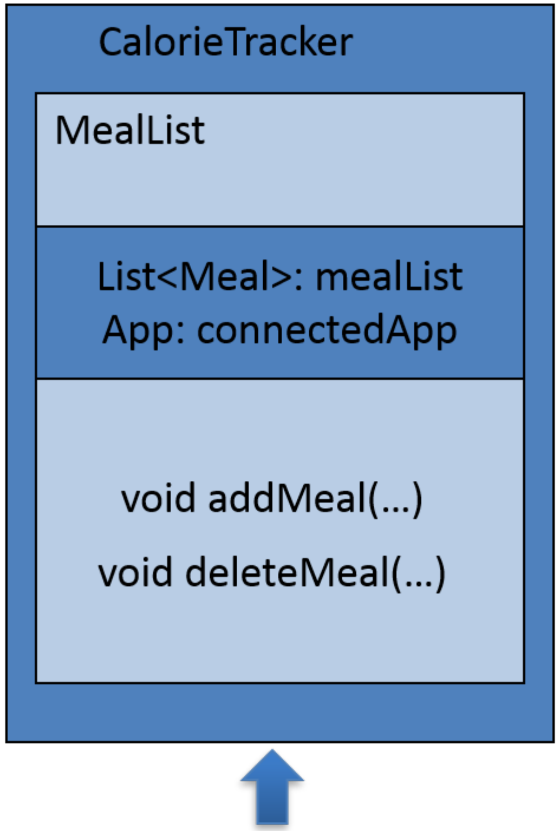
Meal

String: mealName
int: numCalories

void setNumCalories(...)

Unit testing

Meal.setNumCalories(...)



CalorieTracker

MealList

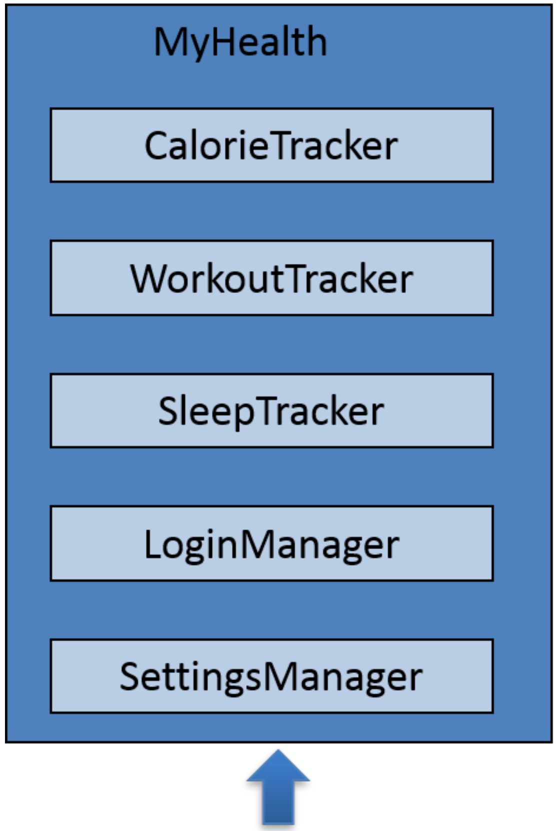
List<Meal>: mealList
App: connectedApp

void addMeal(...)

void deleteMeal(...)

Functional/integration testing

CalorieTracker.addMeal(Meal m)



MyHealth

CalorieTracker

WorkoutTracker

SleepTracker

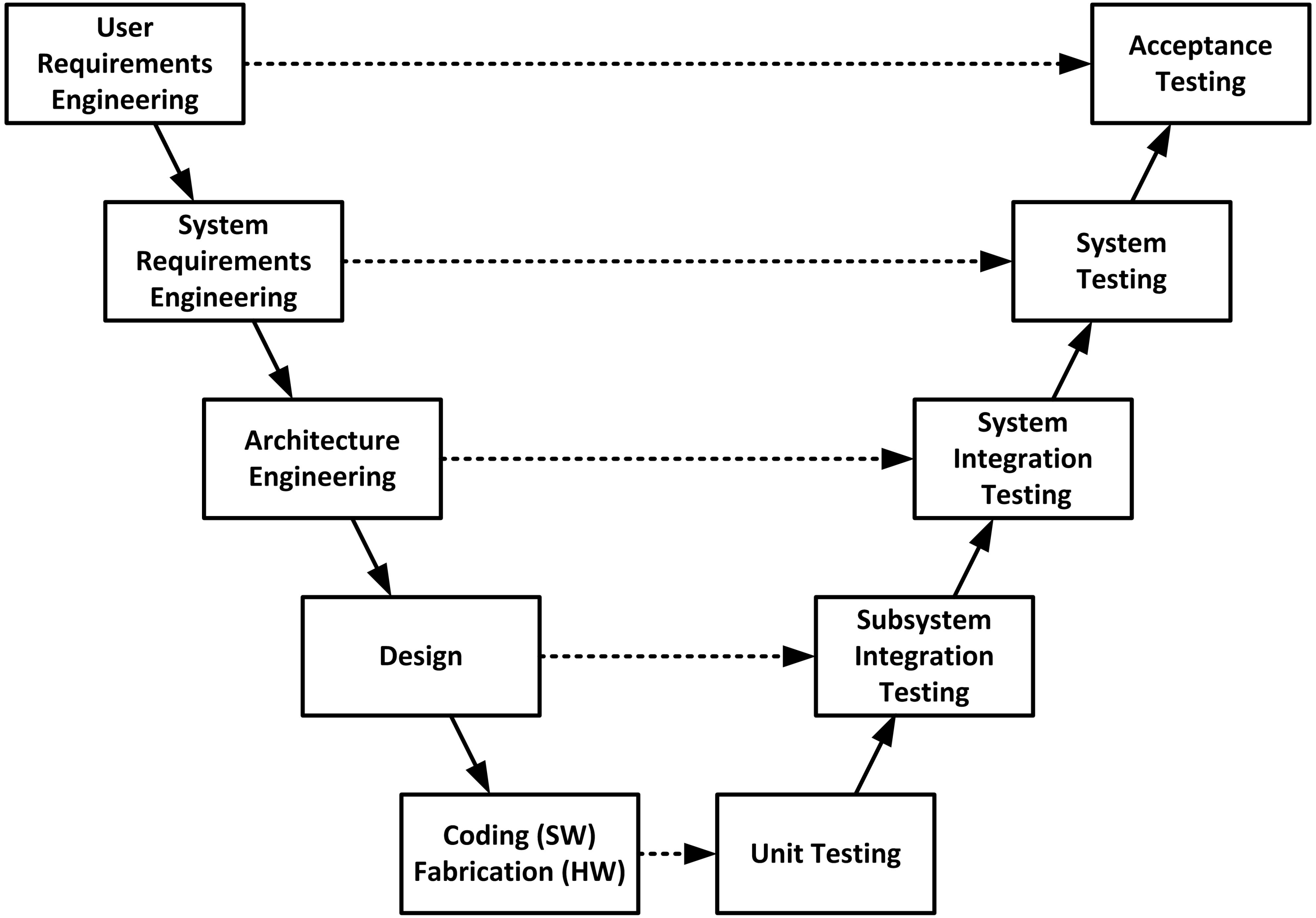
LoginManager

SettingsManager

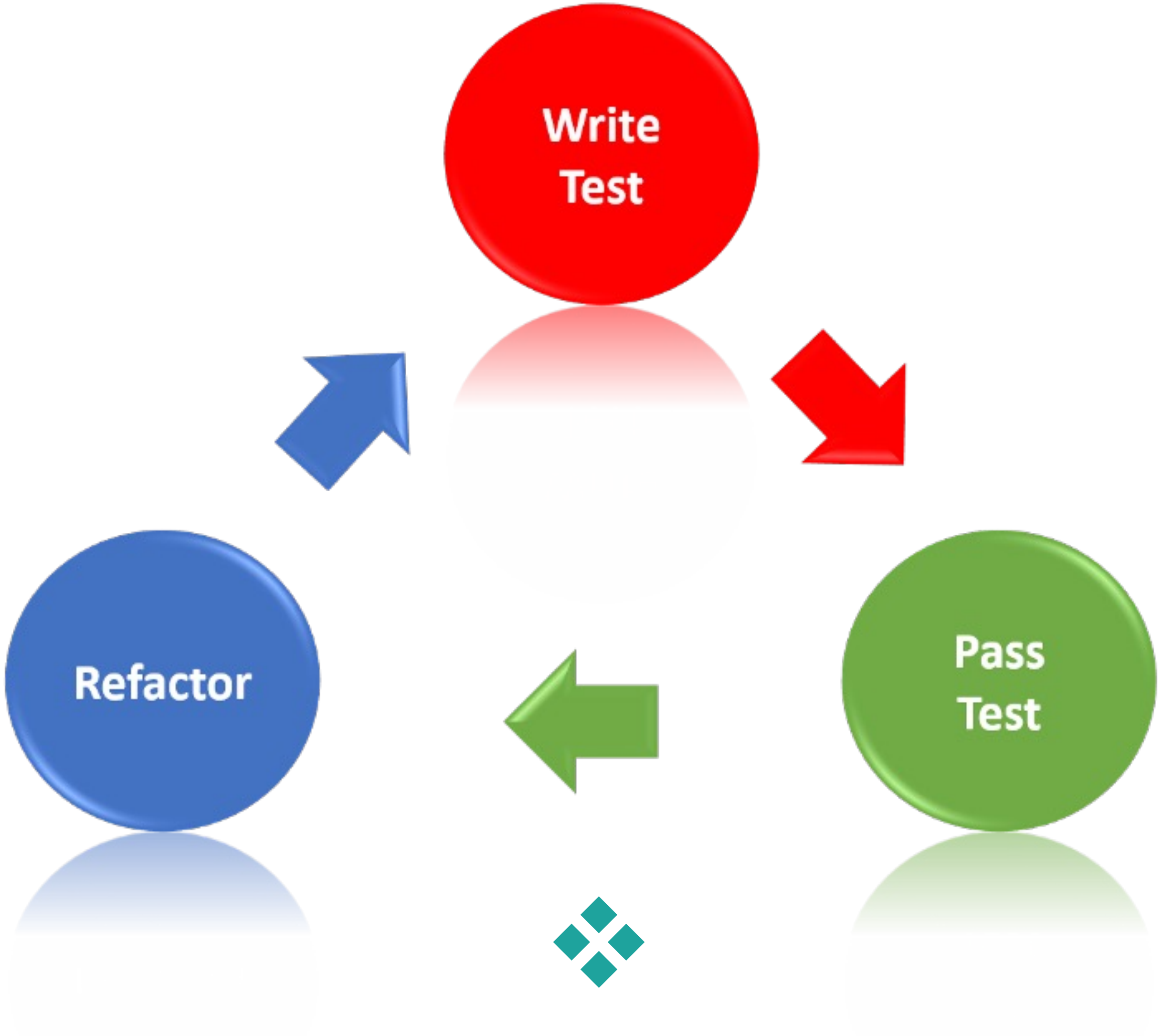
System/acceptance testing

Add a meal
Delete a workout
Login
Logout

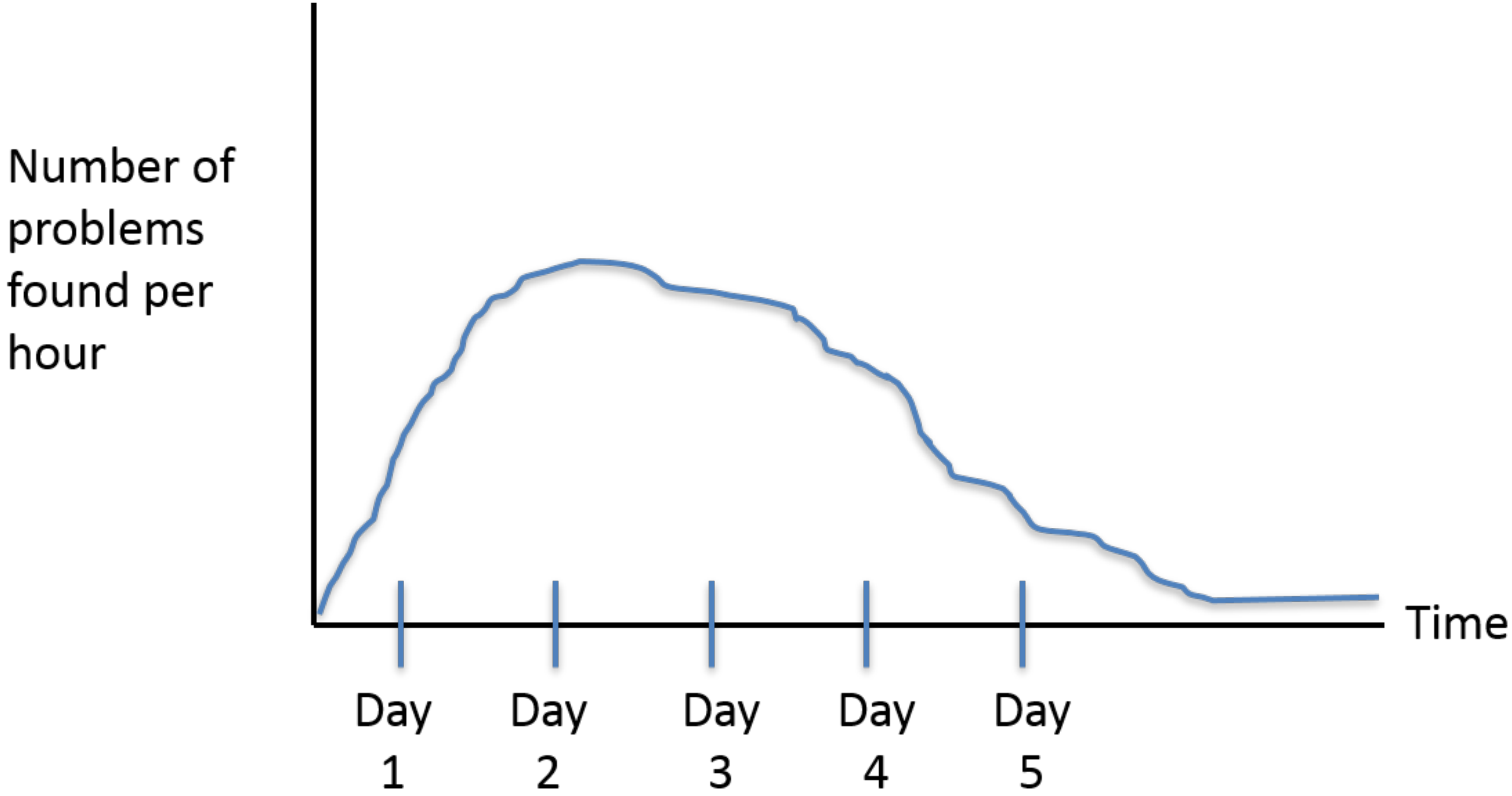
Testing Phases


User Requirements Engineering
System Requirements Engineering
Architecture Engineering
Design
Coding (SW) Fabrication (HW)
Unit Testing
Subsystem Integration Testing
System Integration Testing
System Testing
Acceptance Testing

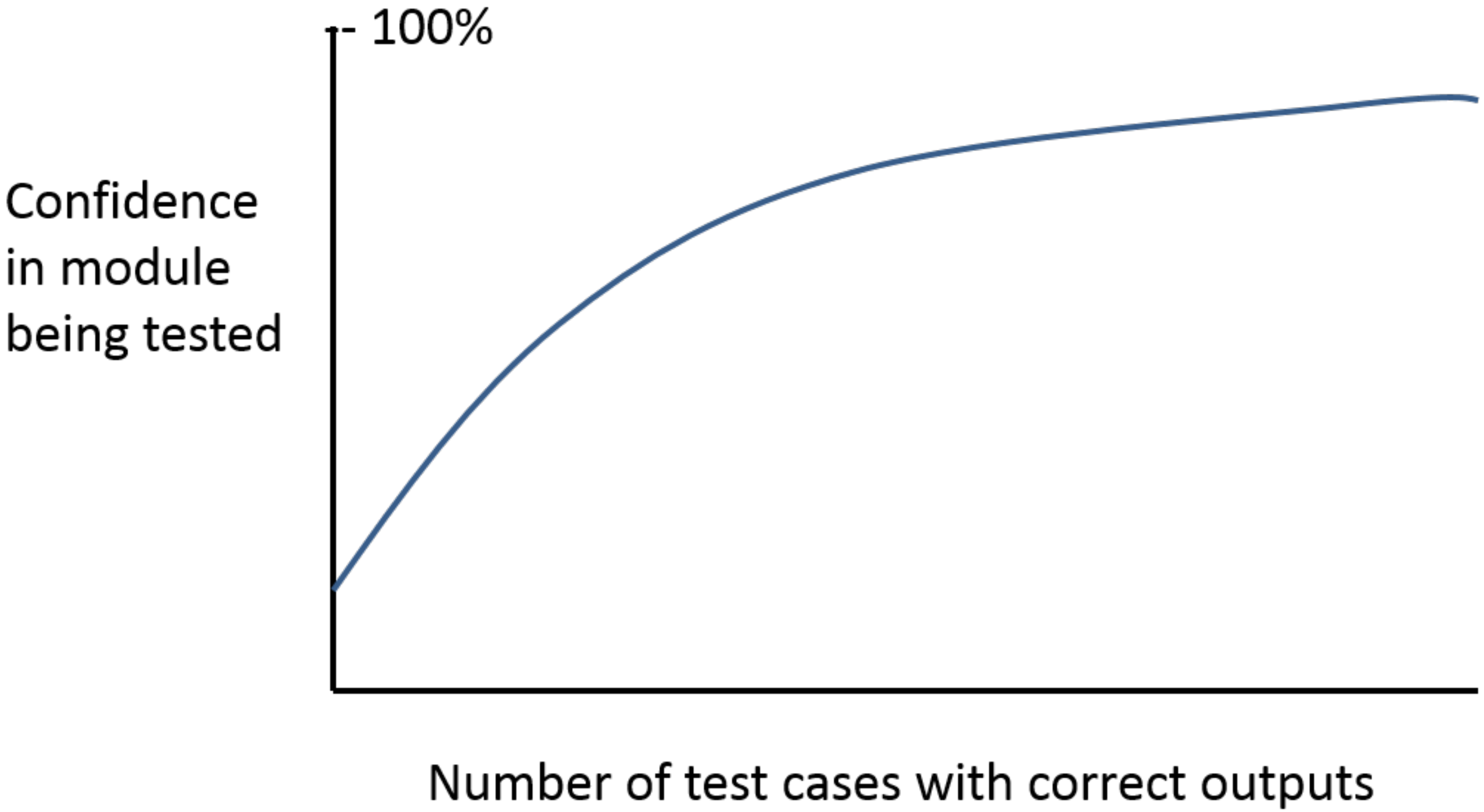
Test-driven Development



How Long?


Number of problems found per hour
Time
Day 1
Day 2
Day 3
Day 4
Day 5

How Long?


- 100%
Confidence in module being tested
Number of test cases with correct outputs

Testing Techniques

- Black-box testing
  - based on software description (specification)
  - cover as much specified behavior as possible
  - only reveals logical failures
  - requires human insight (less systematic)

Testing Techniques

- Black-box testing
  - based on software description (specification)
  - cover as much specified behavior as possible
  - only reveals logical failures
  - requires human insight (less systematic)
- White-box testing
  - based on the implementation (structure of code)
  - cover as much implemented behavior as possible
  - only reveals implemented errors
  - requires source code access

Black box (functional) testing

- Tests logic (specification)
- Focuses on domain (input-specific)
- Code access unnecessary
- Useful at all granularities (unit, function, system)

Black box (functional) testing

How do we get from specification to test cases?

Black box (functional) testing

How do we get from specification to test cases?

1. Identify independently testable features

Black box (functional) testing

How do we get from specification to test cases?

1. Identify independently testable features
2. Identify relevant inputs

Black box (functional) testing

How do we get from specification to test cases?

1. Identify independently testable features
2. Identify relevant inputs
3. Derive test case specifications

Black box (functional) testing

How do we get from specification to test cases?

1. Identify independently testable features
2. Identify relevant inputs
3. Derive test case specifications
4. Generate test cases

Step 1: Identifying Testable Features

Step 2: Test Data Selection

1. Brute-force
    - test all possible inputs

    * exhaustive

Step 2: Test Data Selection

1. Brute-force
   - test all possible inputs

   * exhaustive
2. Random
   - test a subset of inputs selected based on some normal distribution

   * no bias

Step 2: Test Data Selection

1. Brute-force
   - test all possible inputs

   * exhaustive

2. Random
   - test a subset of inputs selected based on some normal distribution

   * no bias

3. Partition
   - select tests based on input subdomains

   * failures typically clustered

Partition Testing

1. Identify partition boundaries

Partition Testing

1. Identify partition boundaries
2. Select input values

Steps 3-4: Test Case Specification, Generation

Black-box Testing Example

White-box (Structural) Testing

- Assumption: if there is a fault in the code, we must execute it to identify it
- Based on the code
- Objective (not test-case dependent)
- Can be performed automatically (using tools)
- Covers coded behavior (as opposed to specified)
- Can be control-flow, data-flow, or fault based

White-box (Structural) Testing

How do we get from code to test cases?

White-box (Structural) Testing

How do we get from code to test cases?

1. Build a model (graph) of the system

White-box (Structural) Testing

How do we get from code to test cases?

1. Build a model (graph) of the system
2. Define test requirements (coverage criteria)

White-box (Structural) Testing

How do we get from code to test cases?

1. Build a model (graph) of the system
2. Define test requirements (coverage criteria)
3. Derive test case specifications

White-box (Structural) Testing

How do we get from code to test cases?

1. Build a model (graph) of the system
2. Define test requirements (coverage criteria)
3. Derive test case specifications
4. Generate test cases

Statement Coverage

- Test requirements: statements in the program (nodes in the graph)
- Coverage measure: $\frac{\#\ executed\ statements}{\#\ total\ statements}$

Branch Coverage

- Test requirements: branches in the program (edges in the graph)
- Coverage measure: $\frac{\#\ executed\ branches}{\#\ total\ branches}$

Branch Coverage

- Test requirements: branches in the program (edges in the graph)
- Coverage measure: $\frac{\#\ executed\ branches}{\#\ total\ branches}$
- Subsumes statement coverage

Condition Coverage

- Test requirements: individual conditions in the program
- Coverage measure: $\frac{\#\ conditions\ that\ are\ both\ T\ and\ F}{\#\ total\ conditions}$

Condition Coverage

- Test requirements: individual conditions in the program
- Coverage measure: $\frac{\#\ conditions\ that\ are\ both\ T\ and\ F}{\#\ total\ conditions}$
- Does NOT subsume branch coverage

Branch and Condition Coverage

- Test requirements: branches and individual conditions in the program
- Coverage measure: consider both coverage measures

Branch and Condition Coverage

- Test requirements: branches and individual conditions in the program
- Coverage measure: consider both coverage measures
- Subsumes branch coverage, and condition coverage

Modified Condition / Decision Coverage

- Key idea: test important combinations of conditions – each condition should independently affect the decision

Modified Condition / Decision Coverage

- Key idea: test important combinations of conditions – each condition should independently affect the decision

* Subsumes branch and condition coverage

Coverage Criteria

1. Statement (node) coverage
2. Branch (edge) coverage
3. Condition coverage
4. Loop coverage
5. Path coverage
6. Data-flow coverage
7. etc..