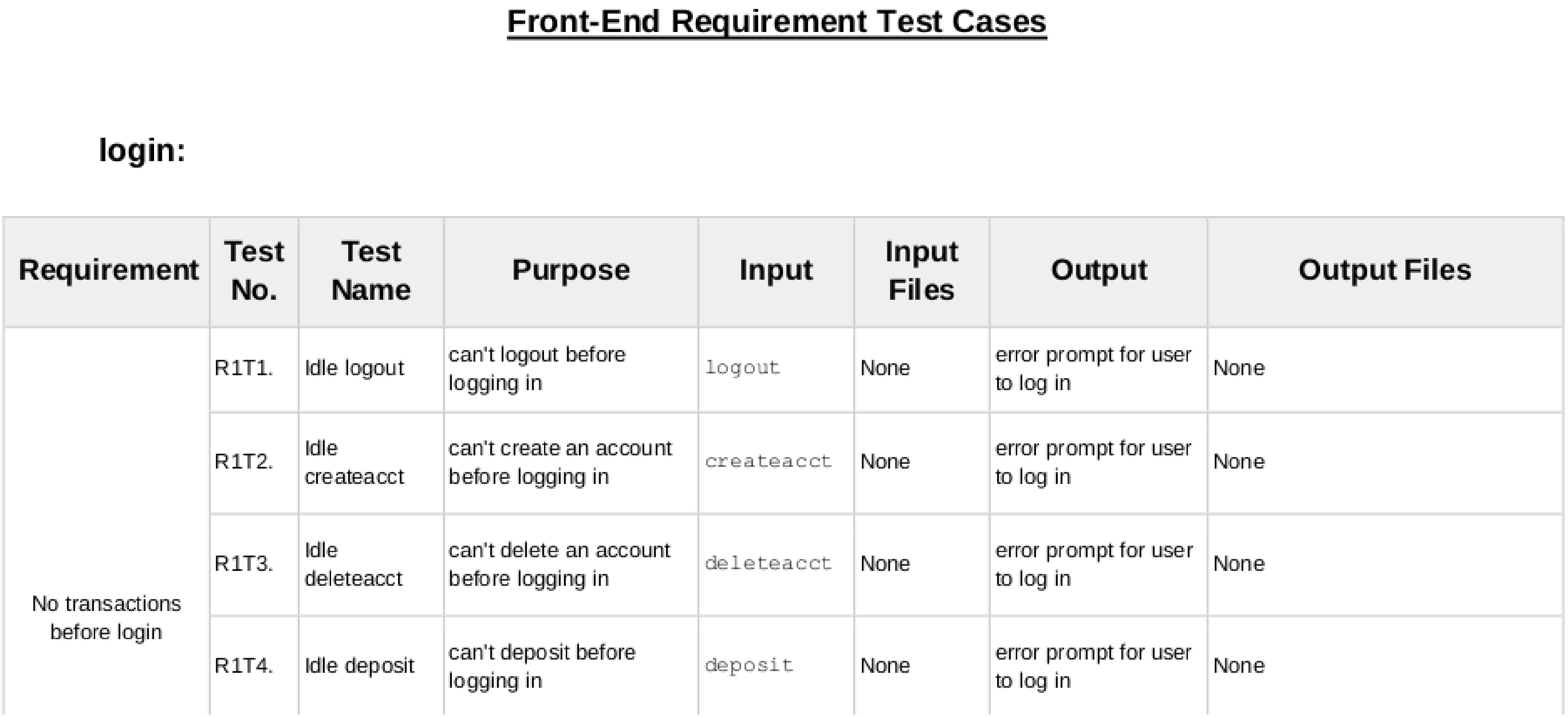
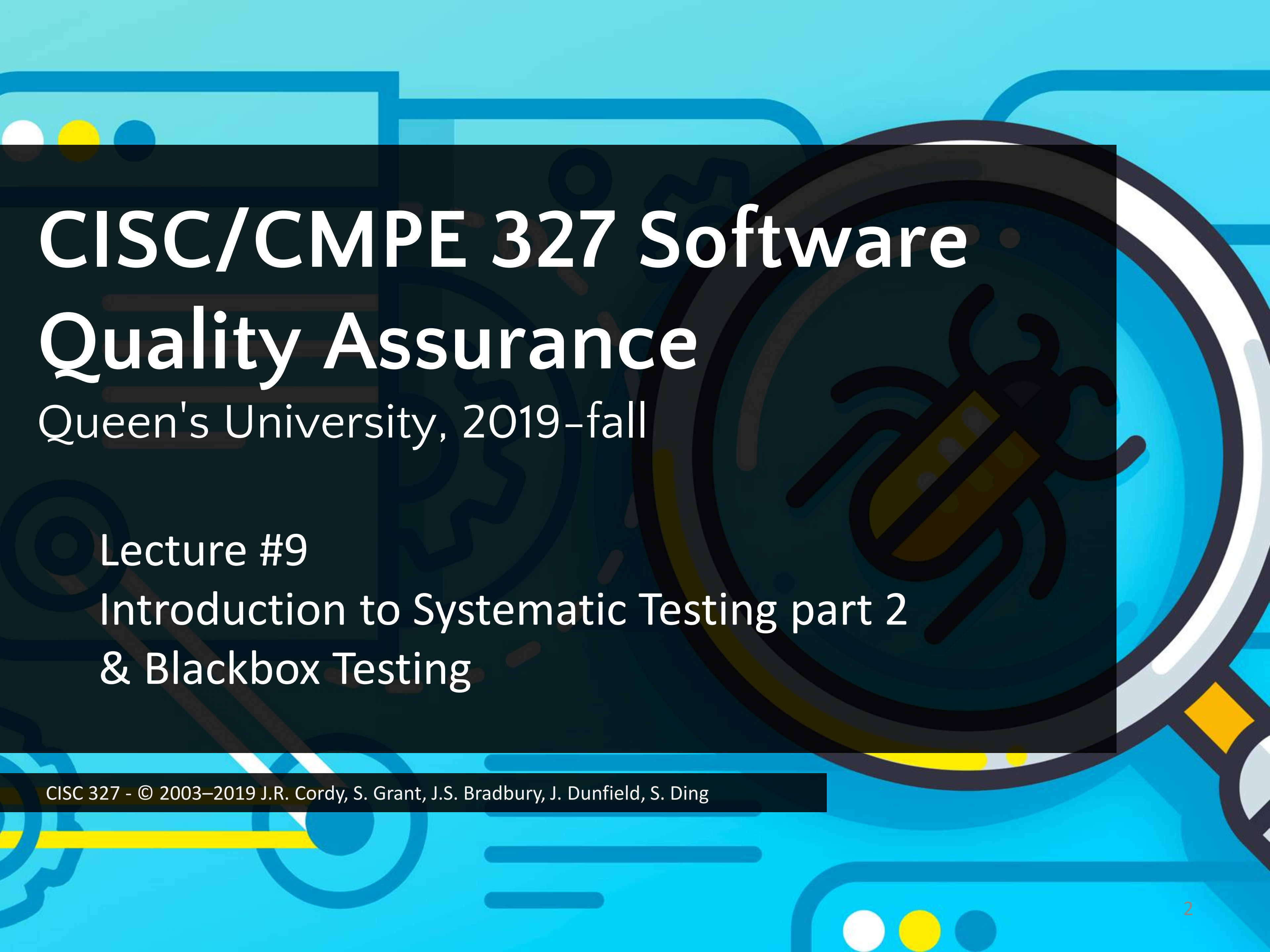
Assignment #1 Example



Front-End Requirement Test Cases

login:

Requirement	Test No.	Test Name	Purpose	Input	Input Files	Output	Output Files
No transactions before login	R1T1.	Idle logout	can't logout before logging in	`logout`	None	error prompt for user to log in	None
	R1T2.	Idle createacct	can't create an account before logging in	`createacct`	None	error prompt for user to log in	None
	R1T3.	Idle deleteacct	can't delete an account before logging in	`deleteacct`	None	error prompt for user to log in	None
	R1T4.	Idle deposit	can't deposit before logging in	`deposit`	None	error prompt for user to log in	None



CISC/CMPE 327 Software Quality Assurance

Queen's University, 2019-fall

Lecture #9
Introduction to Systematic Testing part 2
& Blackbox Testing

Me: I don't need to test this functionality...

Functionality: *breaks immediately*

Me:

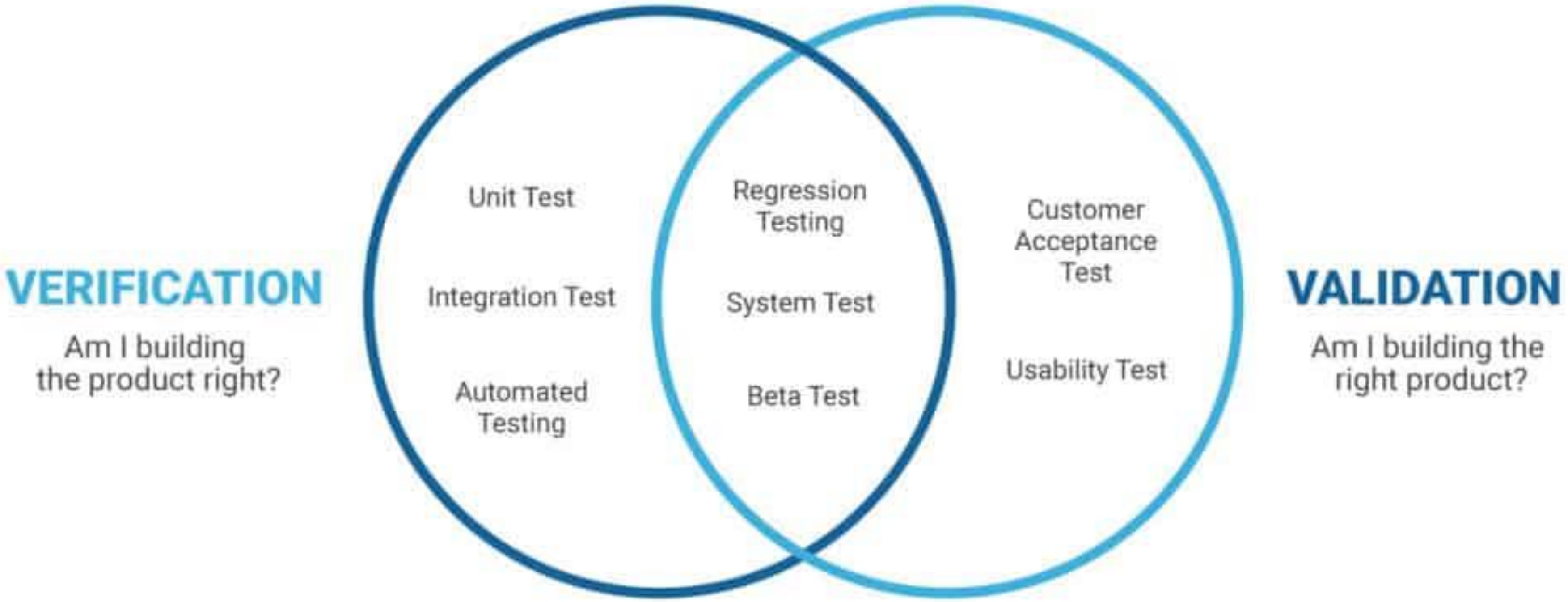
Introduction to Systematic Testing

- Outline
  - Definitions
  - Validation vs. verification
  - Role of specifications
  - Levels of testing
  - Today we continue with:
    - Testing in the life cycle
    - Test design and strategy
    - Test plans and procedures
    - Test results

What is Systematic Testing?

- An explicit discipline or procedure (a **system**) for
  - choosing and creating test cases
  - executing the tests and documenting the results
  - evaluating the results, ~~possibly~~ **automatically**
  - deciding when we are done (enough testing)

Validation vs. Verification

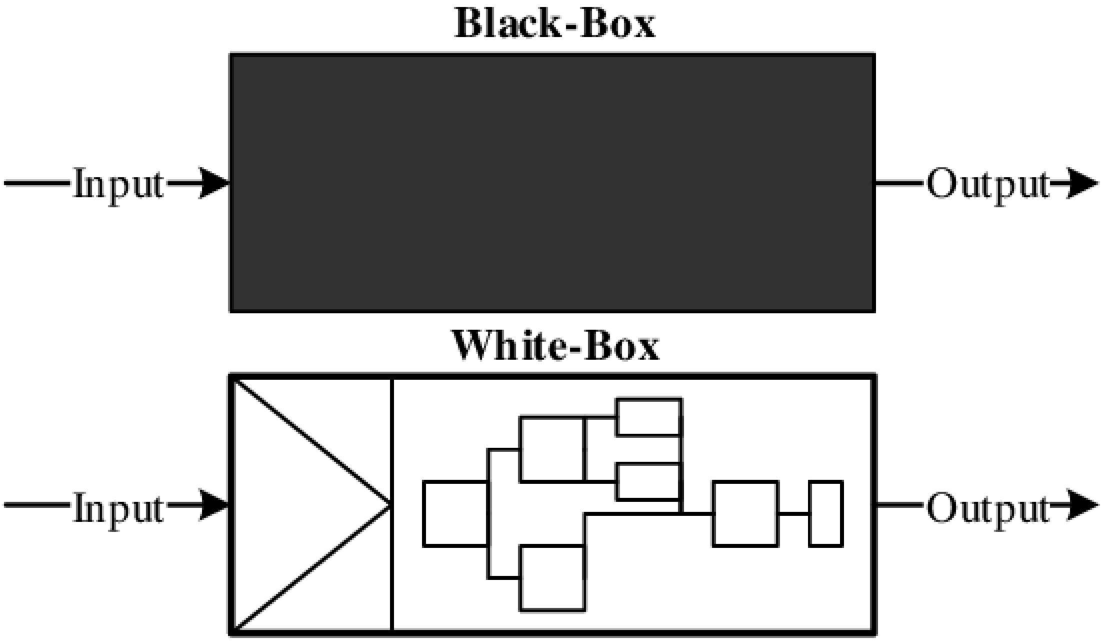


Testing in the Life Cycle

- Kinds of Tests
  - Testing done through software life cycle
    - development of code (unit testing)
    - the integration (integration testing)
    - the acceptance the system (system testing)

Testing in the Life Cycle

- **Black box** testing methods are based on the software's **specifications**
- **White box** (or glass box) testing methods are based on the software's **code**

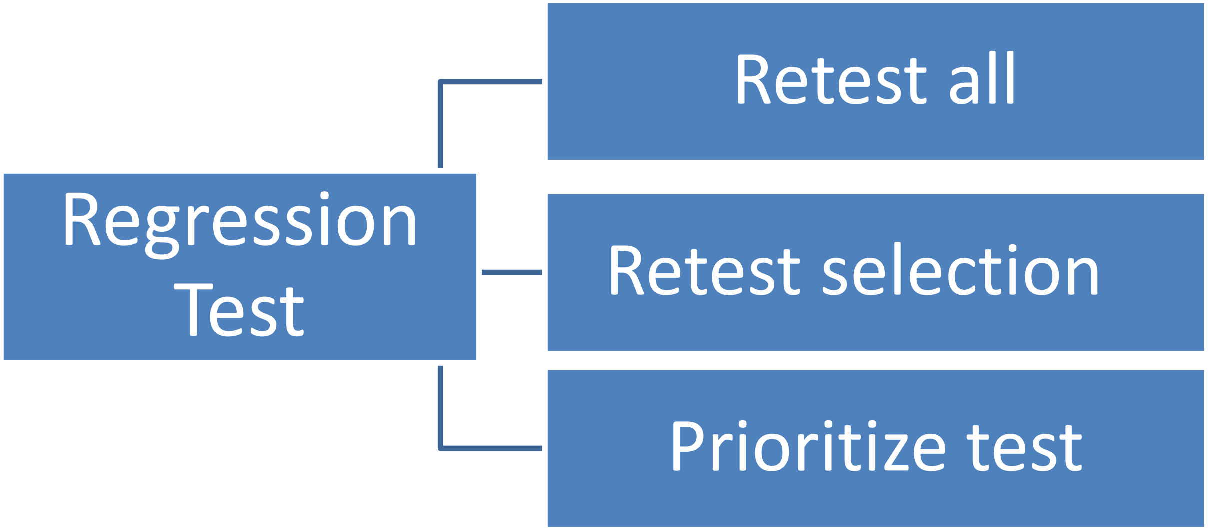


-ility Testing

- System characteristics for quality or testing
  - Capability:
    - The required functions?
  - Reliability:
    - Resist failure in all required situations?
  - Usability:
    - Easy to use?
  - Performance:
    - Fast? Responsive? Scalable?
  - Security:
    - secure?

Regression Test

- Codebase changed?
  - re-running functional and non-functional tests



Regression Test

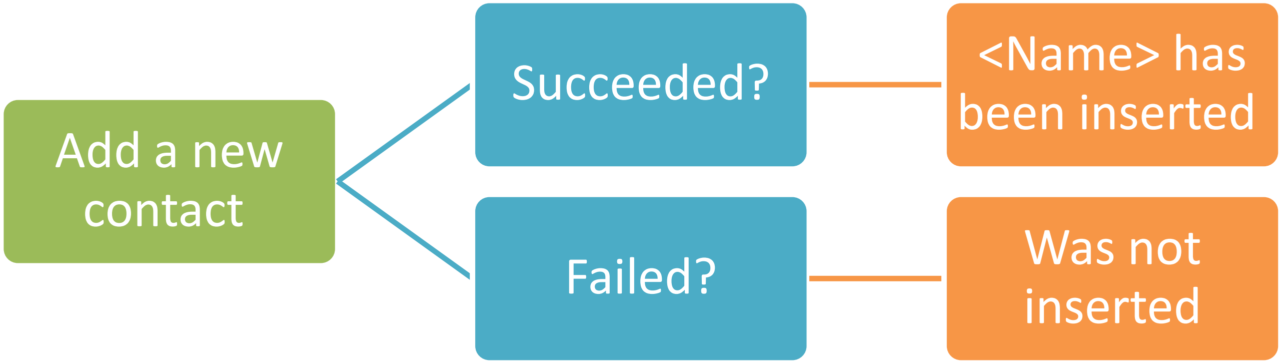
- Millions of test case…
- Frequent update
- Cost? $$$
- Maintenance?

Testing in the Life Cycle

- Failure Tests
  - Test known/discovered/fixed failures
  - Known observed inputs -> caused the past failures

Testing in the Life Cycle

- Failure Tests



Test Design

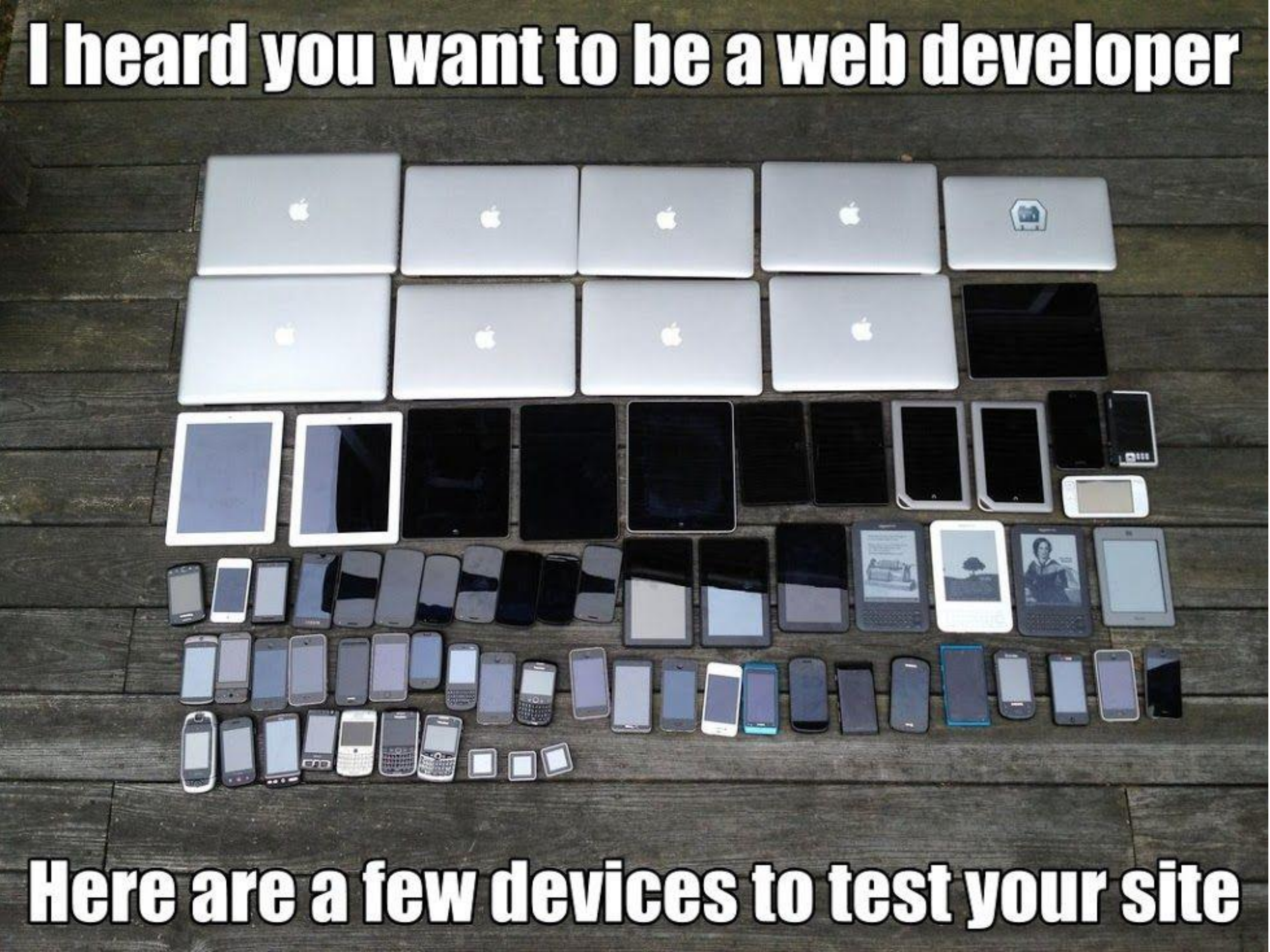
- Design of Tests
  - A difficult and tricky engineering problem
  - A set of stages
    - High level test -> detailed test procedures
  - Typical test design stages are:
    - test strategy
    - test planning
    - test case design
    - test procedure

Test Strategy

- Test Strategy
  - the overall approach to testing
  - Levels of testing?
  - Methods?
  - Techniques?
  - Tools?
  - Standards?
  - …
- overall quality plan, by PM, driven by business
- static

Test Plans

- Test Plans
  - how the test strategy will be carried out
    - the items to be tested
    - the level they will be tested at
    - the order they will be tested in
    - the test **environment**
    - Responsibility?
    - Coverage?
  - project-wide, or procedure-wise
  - By Test Lead or Test Manager

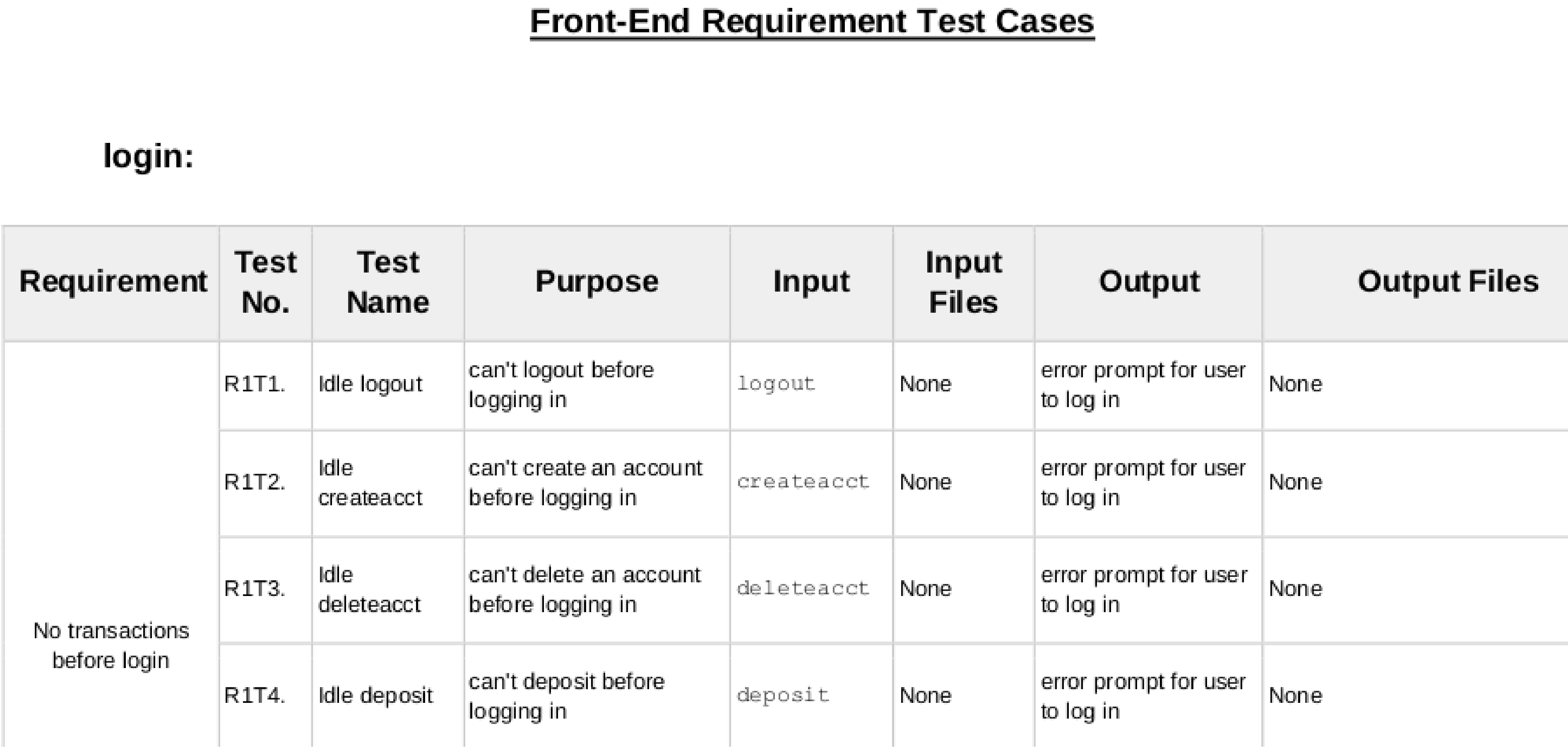

I heard you want to be a web developer
Here are a few devices to test your site

Test Case Design

- Test Case Design
  - a set of test cases for each item to be tested at each level
  - Each test case specifies
  - how the implementation is to be tested
  - how we will know if the test is successful
  - Input -> action[s]/event[s] -> expected response

Test Case Design

- What might a test case look like?



Front-End Requirement Test Cases

login:

Requirement	Test No.	Test Name	Purpose	Input	Input Files	Output	Output Files
No transactions before login	R1T1.	Idle logout	can't logout before logging in	`logout`	None	error prompt for user to log in	None
	R1T2.	Idle createacct	can't create an account before logging in	`createacct`	None	error prompt for user to log in	None
	R1T3.	Idle deleteacct	can't delete an account before logging in	`deleteacct`	None	error prompt for user to log in	None
	R1T4.	Idle deposit	can't deposit before logging in	`deposit`	None	error prompt for user to log in	None

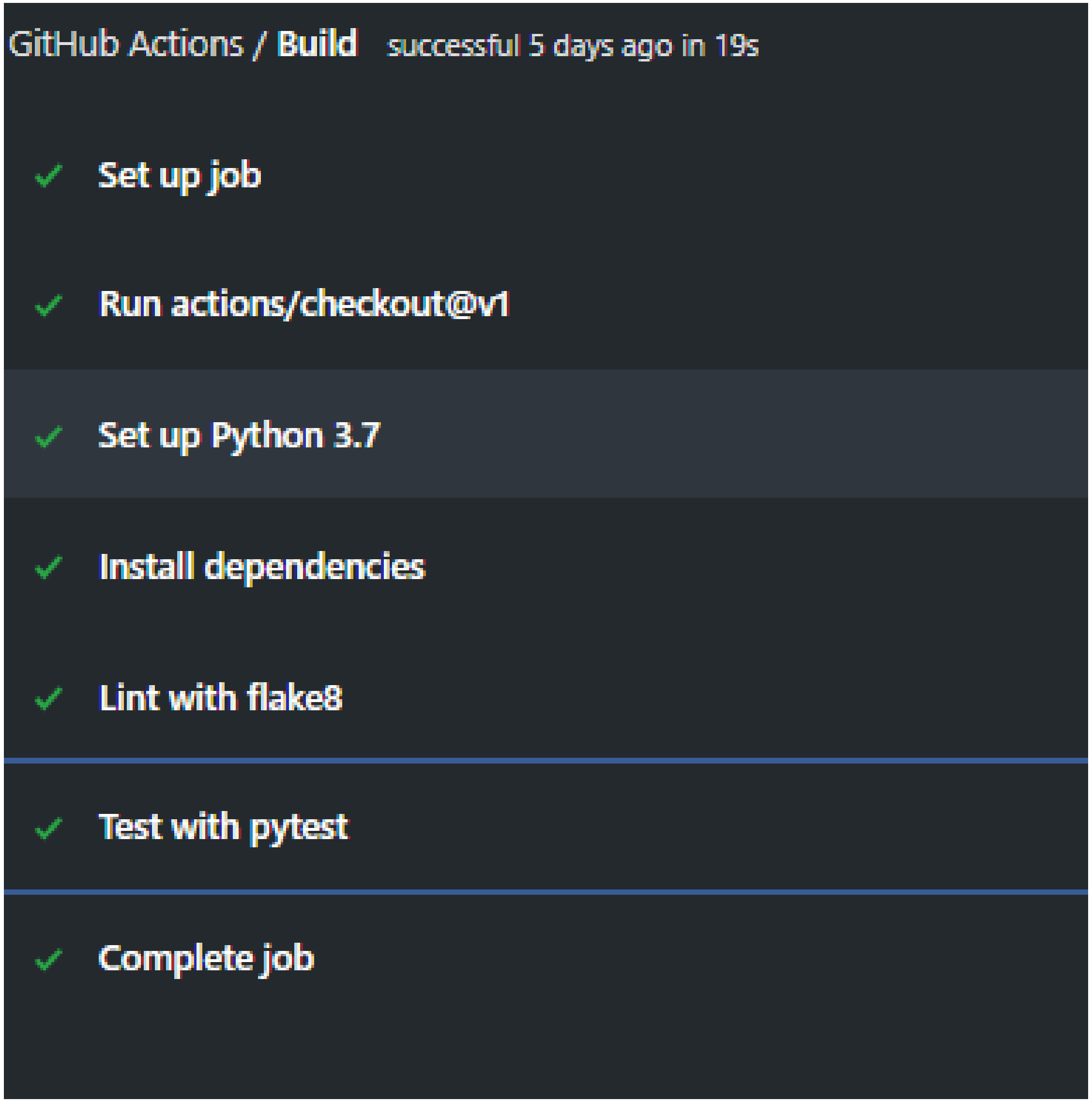
Test Case Design

- Test Case Design (continued)
  - positive testing (should do)
  - negative testing (shouldn't do)
  - separately by level: unit, integration, system, and acceptance

Test Procedures

- Test Procedures
  - the **process** for conducting test cases
  - For each level
    - the process for running and evaluating the test cases
      - test harnesses (run part of the system)
    - test scripts
    - testing tools (frameworks)
      - GitHub Actions

GitHub Actions



GitHub Actions / **Build** successful 5 days ago in 19s

- ✓ Set up job
- ✓ Run actions/checkout@v1
- ✓ Set up Python 3.7
- ✓ Install dependencies
- ✓ Lint with flake8
- ✓ Test with pytest
- ✓ Complete job

Test Reports

- Documenting Test Results
  - Output of test execution results file,
  - Summarized in a readable report
  - Concise, easy to read, and to clearly point out failures
  - A **standardized** form
  - With tools/framework
    - pytest xxxx --junitxml="result.xml"
    - There is an HTML option

Summary

- Introduction to Testing
  - Testing is not just a one time task, it is a continuous process that lasts throughout the software life cycle
  - Effective testing requires careful engineering, similar and parallel to the process for design and implementation of the software itself
  - An overall test strategy drives test plans, test case design, and test procedures for a project

Summary

- References
  - Sommerville, ch. 8, "Software Testing"
  - The Software Test Page (on the web)
- Next
  - Introduction to Black Box Testing
  - Assignment #1 due **next** Thursday

CISC/CMPE 327 Software Quality Assurance

Queen's University, 2019-fall

Lecture #9
Blackbox Testing

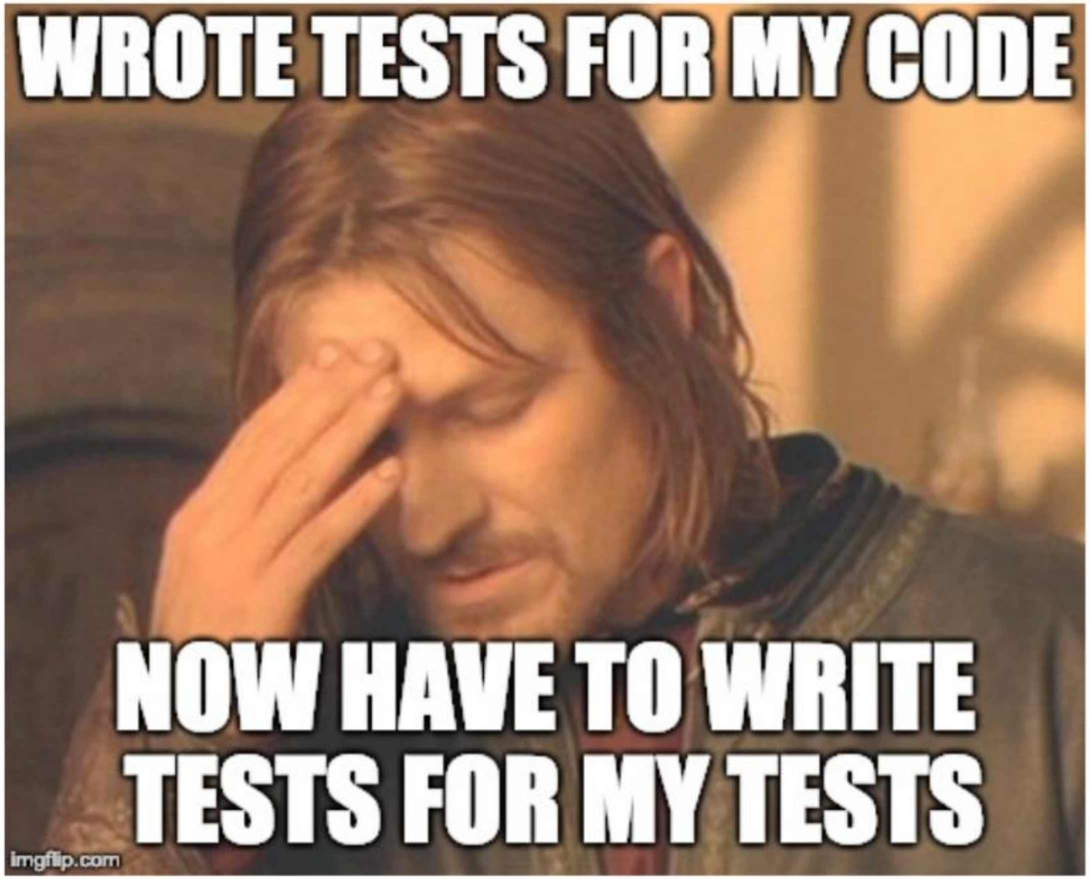
Black Box Testing

- Outline
  - Introduction to testing methods: black box and white box
  - Kinds of black box methods
  - Black box method 1: Systematic functionality testing

Systematic Testing Methods

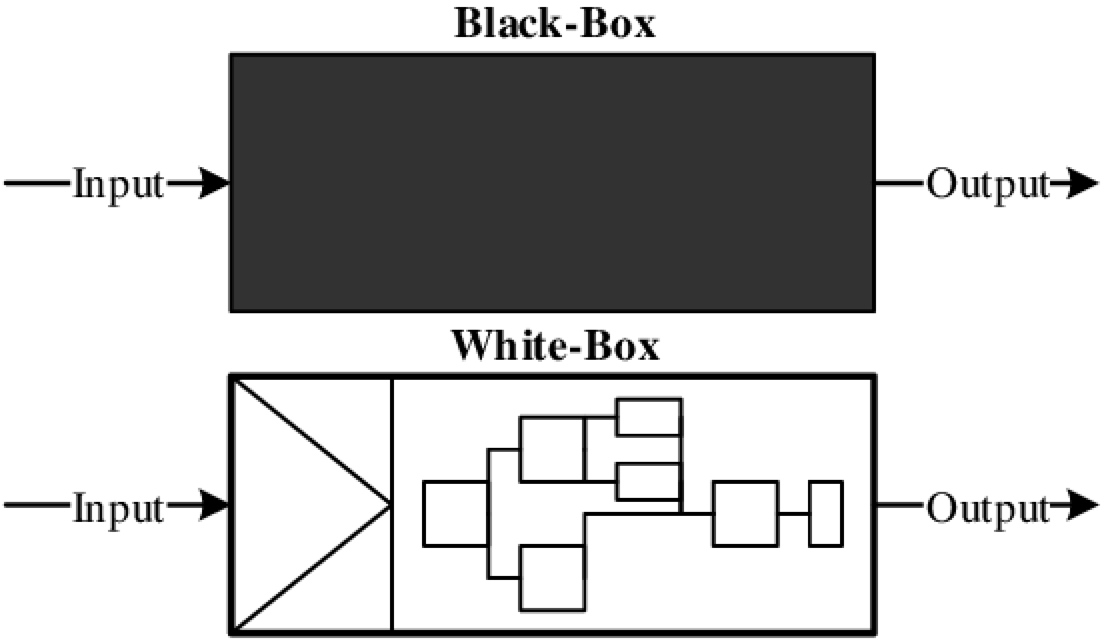
- systematic, it must have:
  - a system (rule) for creating tests
  - a measure of completeness
- What to test
- How to test
- When we're done

- Test Adequacy



Testing in the Life Cycle

- **Black box** testing methods are based on the software's **specifications/functionality**
- **White box** (or glass box) testing methods are based on the software's **code**



Black Box Methods

- Black Box Methods
  - Chosen based on requirements, specification, or (sometimes) design documents
  - Advantage: independently of the software
    - Parallel development of test cases.
  - Based on the functional specification (requirements) for the software system

Black Box Methods

- Functional Specifications
  - Formal (mathematical),
  - Informal (in a natural language)
  - at least three kinds of information:
    - the intended inputs
    - the corresponding intended actions
    - the corresponding intended outputs

Black Box Methods

- Three Kinds of Black Box Methods
  - Input coverage tests
  - Output coverage tests
  - Functionality coverage tests

Black Box Methods

- Input coverage tests
  - An analysis of the intended inputs
- Output coverage tests
  - An analysis of the intended outputs
- Functionality coverage tests
  - An analysis of the intended actions

Systematic Functionality Testing

- An example
  - partition the functional specification
    - into a set of small, separate requirements
  - Example: Suppose that the informal requirements for a program we are to write are as follows:

    - "Given as input two integers x and y, output all the numbers smaller than or equal to x that are evenly divisible by y. If either x or y is zero, then output zero."

Requirements Partitioning

- "Given as input two integers x and y"
  - R1. Accept two integers as input.
- "output ... the numbers"
  - R2. Output zero or more (integer) numbers.
- "smaller than or equal to x"
  - R3. All numbers output must be less than or equal to the first input number.
- "evenly divisible by y"
  - R4. All numbers output must be evenly divisible by the second number.
- "all the numbers"
  - R5. Output must contain **all** numbers that meet both R3 and R4.
- "If either x or y is zero, then output zero."
  - R6. Output must be zero (only) in the case where either first or second input integer is zero.

Test Case Selection

- Test Cases for **Each** Requirement
  - Each requirement: independent
  - Example: For the partitioned requirement:
    - "If either x or y is zero, then output zero."
      R6. Output must be zero (only) in the case where either first or second input integer is zero.

Test Case Selection

- Example: For the partitioned requirement:
  - "If either x or y is zero, then output zero."
    R6. Output must be zero (only) in the case where either first or second input integer is zero.

- We might choose the test cases:

R6T1.	0	0	(both zero)
R6T2.	0	1	(x zero, y not)
R6T3.	1	0	(y zero, x not)
R6T4.	1	1	(neither zero)

- Simplest possible and make no attempt to be exhaustive - more on this later

A Systematic Method

- Black Box Functionality Coverage
  - a system for creating functionality test cases
    - It tells us when we are done –covered all partitions
  - not the same as acceptance testing
    - a **separated** view of functional requirement
    - Cannot replace acceptance testing
    - It is a systematic method
    - it is only a partial test, like other systematic methods

Choosing & Organizing Tests

- An Experiment
  - experiment on the software system
  - hypothesis
    - software has certain properties,
  - method to test whether the hypothesis holds with our test cases
  - observe the results and draw conclusions

Choosing & Organizing Tests

- Experimental Design
  - Fundamentally - the isolation of "variables"
    - design the experiment
      - each possible cause that may affect the outcome (each experimental "variable") can be observed independently
    - Thus when an effect is observed, we can tell which cause is at work
  - The usual way to do this is to design the experiment in steps that only vary one "variable" (possible cause) at a time, keeping everything else constant

Choosing & Organizing Tests

- Test Plan Design
  - The experimental model of software testing gives us two important principles for our test plan:
    1. Test inputs should be chosen to carefully isolate different causes of failure (the experimental variables)
    2. Test cases should be ordered such that each test only assumes features to be working that have already been tested by a previous test

Guidelines for Choosing Test Inputs

- Choosing Inputs
  - For test inputs, this principle means that we should help isolate failure causes, by as much as possible
    1. Consistently choosing the simplest input values possible, in order not to introduce arbitrary variations
    2. Keeping everything constant between test cases, varying only one input value at a time
       (don't try to be "clever" introducing random input variations)
  - These principles hold for all systematic test methods, not just this one

Ordering Tests for QIES

- **T2** can log in in agent mode

  .

  .

  .

- **T14** createservice disallowed in agent mode

Ordering Tests for QIES

- **T14** createservice disallowed in agent mode

  .

  .

  .
- **T2** can log in in agent mode

If T14 doesn't produce the expected result, is it because *(a)* createservice is erroneously allowed, or *(b)* because agent-mode login doesn't work?

Don’t Duplicate Tests

- Some parts of the project requirements are redundant:
  - “after an agent login, only agent transactions and logout are accepted”
  - [createservice] “privileged transaction, only accepted in planner mode”
- This is a natural way of stating the requirements, not an inconsistency
- But don’t write two identical test cases here

Check your levels

- We are *not* doing acceptance testing here
- Instead, breaking down the requirements into pieces and testing each one
- **None** of the test cases you’re writing for A1 will be big enough to seem “realistic”

Assumptions

- Front End behaviour is deterministic:
  same input ⇒ same output.
- Can you think of a kind of testing that *isn't* deterministic?
- Aside: "voting" on airplanes
- Implementors (you!) are reasonable:
  - You won't write code that detects the specific case of 27 createservice transactions in a row so you can crash on purpose

Summary

- Black Box Testing
  - Two classes of systematic test methods, black box and white box
  - Black box methods include input coverage, output coverage, functionality coverage
  - Functionality coverage partitions the functional specification into separate requirements to test
  - Isolate causes by ordering tests by features used, keeping test input values simple, and varying one input value at a time

Summary

- References
  - Sommerville, ch. 8, "Software Testing”
- Next Time
  - More Black Box Testing - Input coverage methods