Announcements

- GCP Assignment Due Today
- Quizizz for participation and extra credit today.
- Project Design Assignment due 6/22
- CATME Midterm Evaluation Due 6/22
- Midterm Survey due 6/22

CS3300 A: Introduction to Software Engineering

Lecture 9: Black-Box Testing

Dr. Nimisha Roy ▸ nroy9@gatech.edu

Slides adapted from Alessandro Orso

Black- Box Testing

Advantages

- Focus on the domain
- No need for the code
  - Early test design
  - Prevents the highly occurring scenario of no-time-for-testing
- Catches logic defects
- Applicable at all granularity levels

Slide adapted from Alessandro Orso

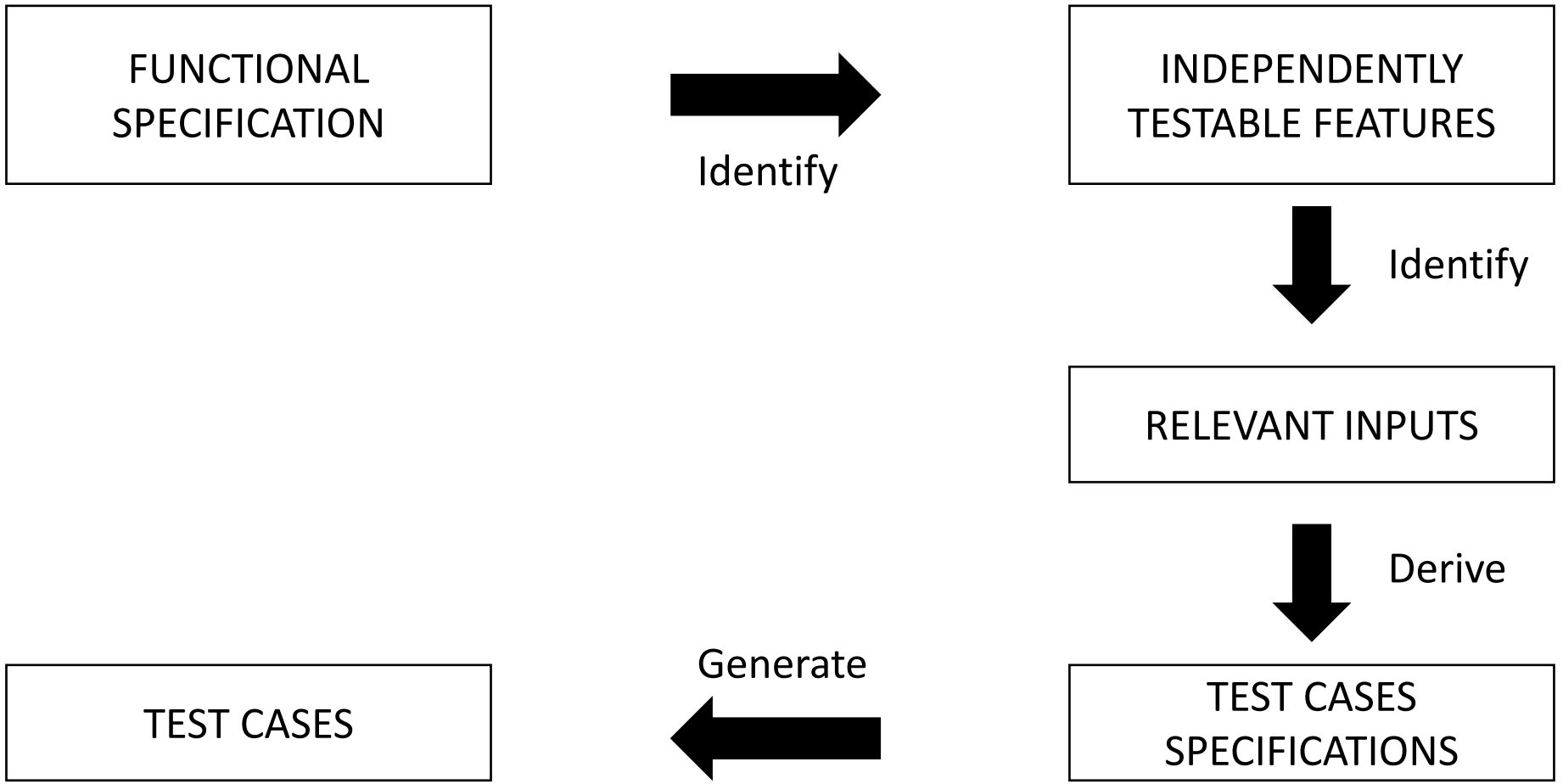
From Specifications to Test Cases

FUNCTIONAL SPECIFICATION

TEST CASES

Slide adapted from Alessandro Orso

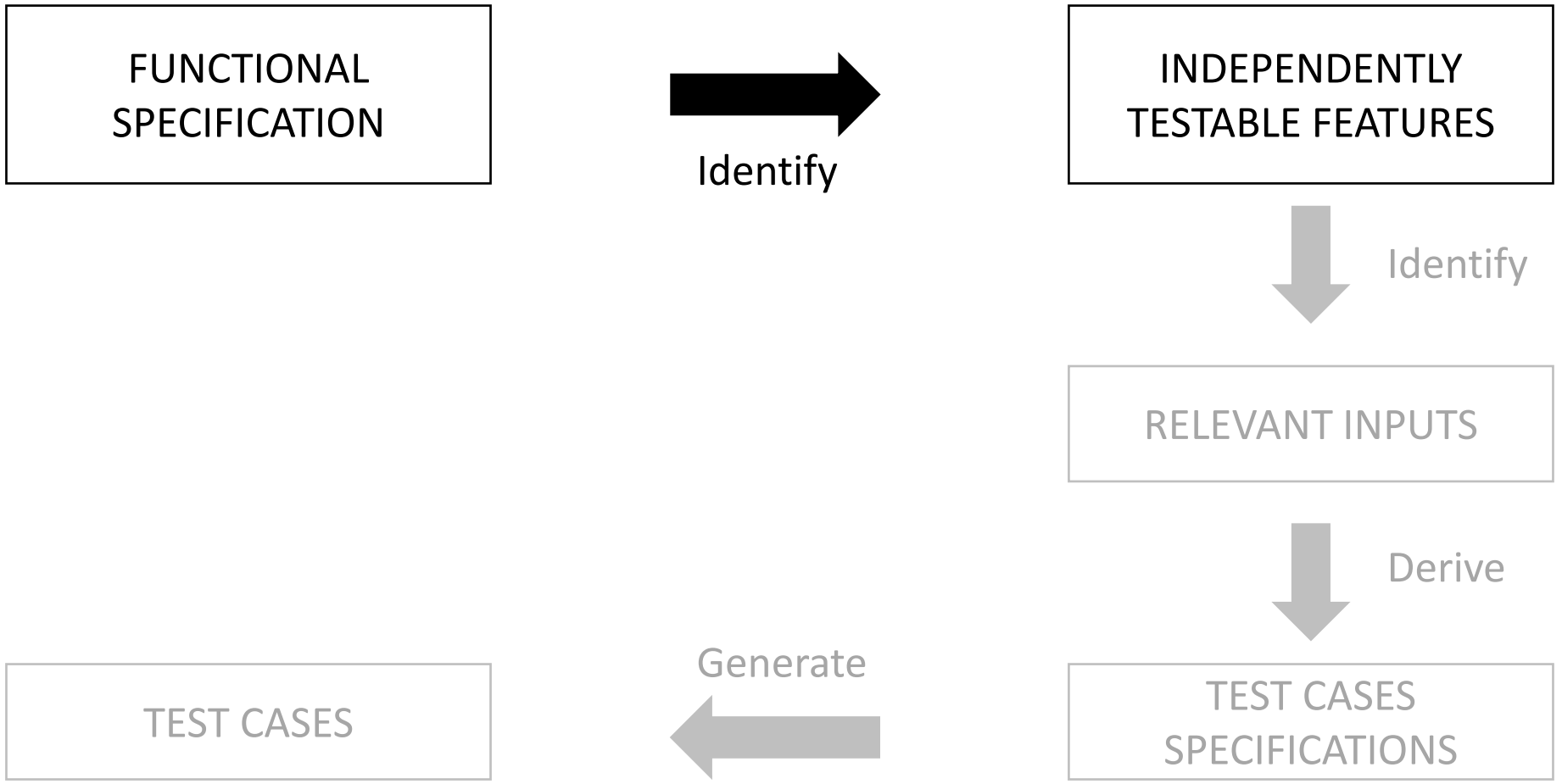
A systematic Functional-Testing Approach



Decoupling; Automated Sub-tasks; Monitor testing process

Slide adapted from Alessandro Orso

A systematic Functional-Testing Approach



Slide adapted from Alessandro Orso

Identifying Testable Features

printSum (int a, int b)

How many independently testable features do we have here?

[✓] 1

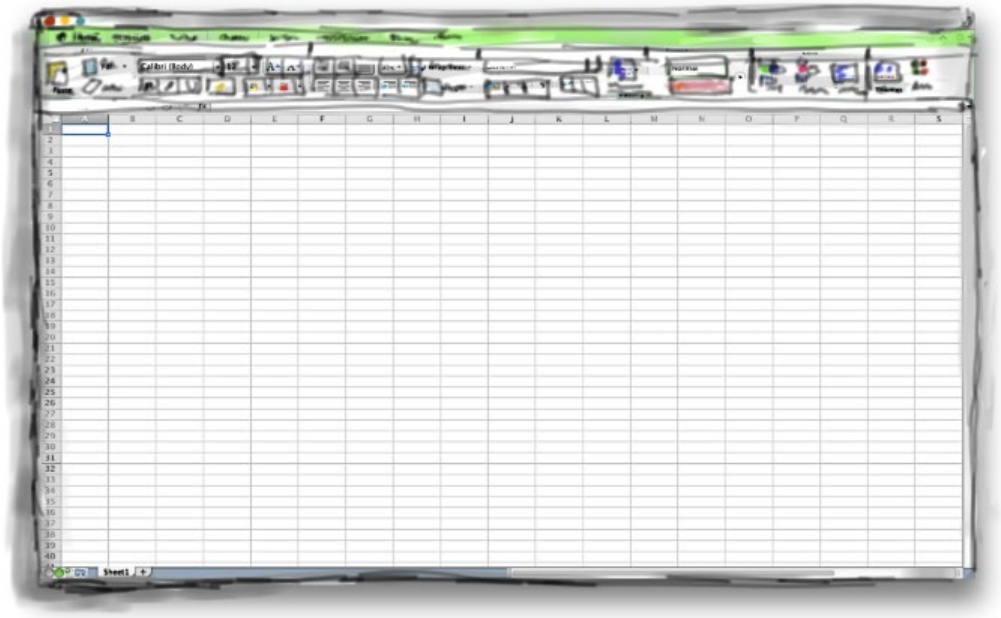
[ ] 2

[ ] 3

[ ] 4

Identifying Testable Features

Identify 3 possible independently testable features for a spreadsheet

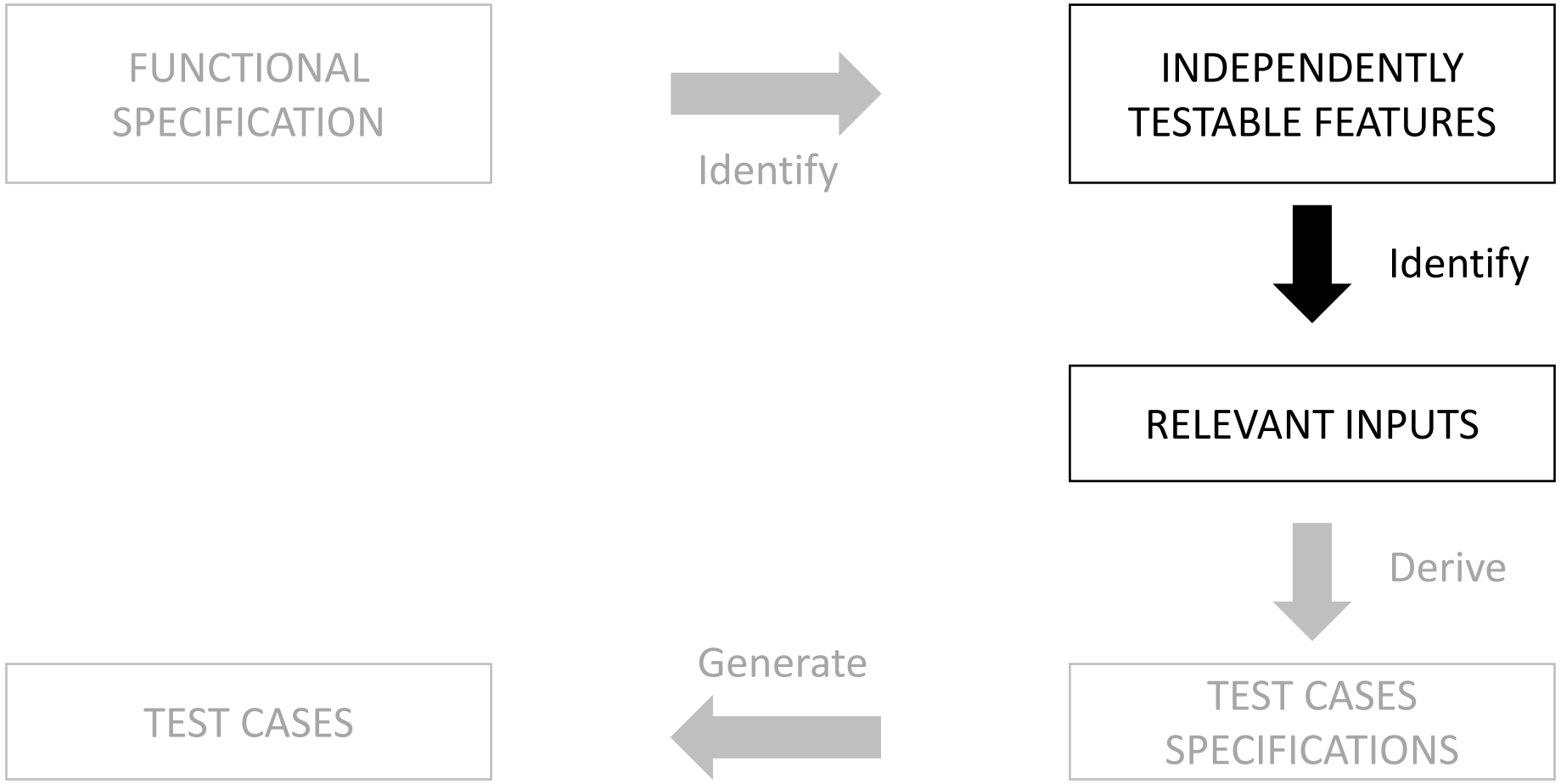


[Statistical Functions]

[Cell Merging]

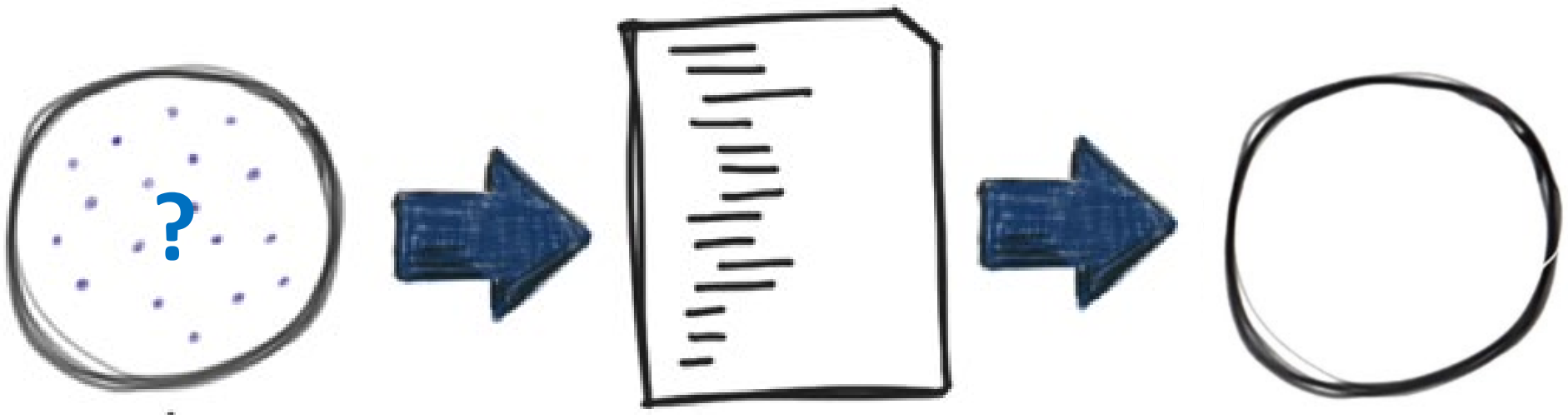
[Chart creation]

A systematic Functional-Testing Approach



Slide adapted from Alessandro Orso

Test Data Selection



Input Domain D

Software

Output Domain O

How to select meaningful set of inputs and corresponding outputs?

Powerful machines, why not exhaustive search?

Straw-Man Idea: Exhaustive Testing!

How long would it take to exhaustively test the function printSum(int a, int b)?

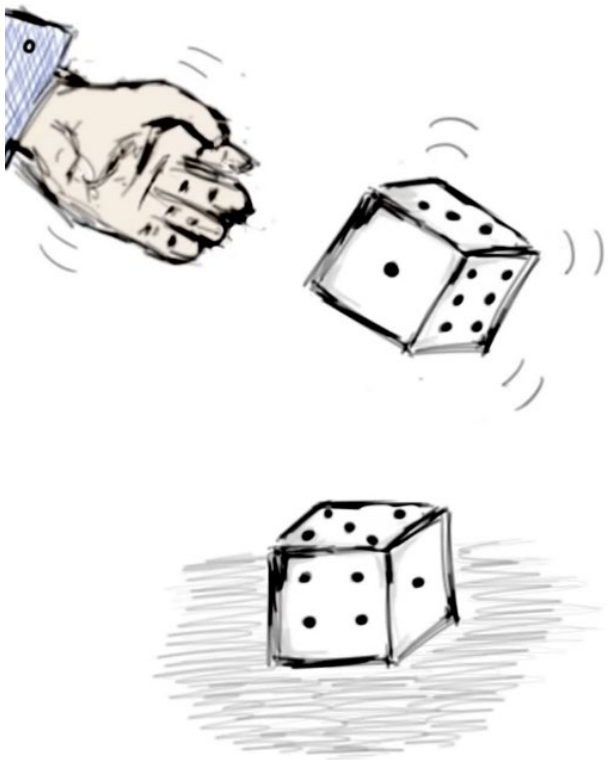
$2^{32} * 2^{32} = 2^{64} \sim= 10^{19}$ tests

1 test per nanosecond

10^9 tests per second

10^{10} seconds overall ~ 600 years

Random Testing



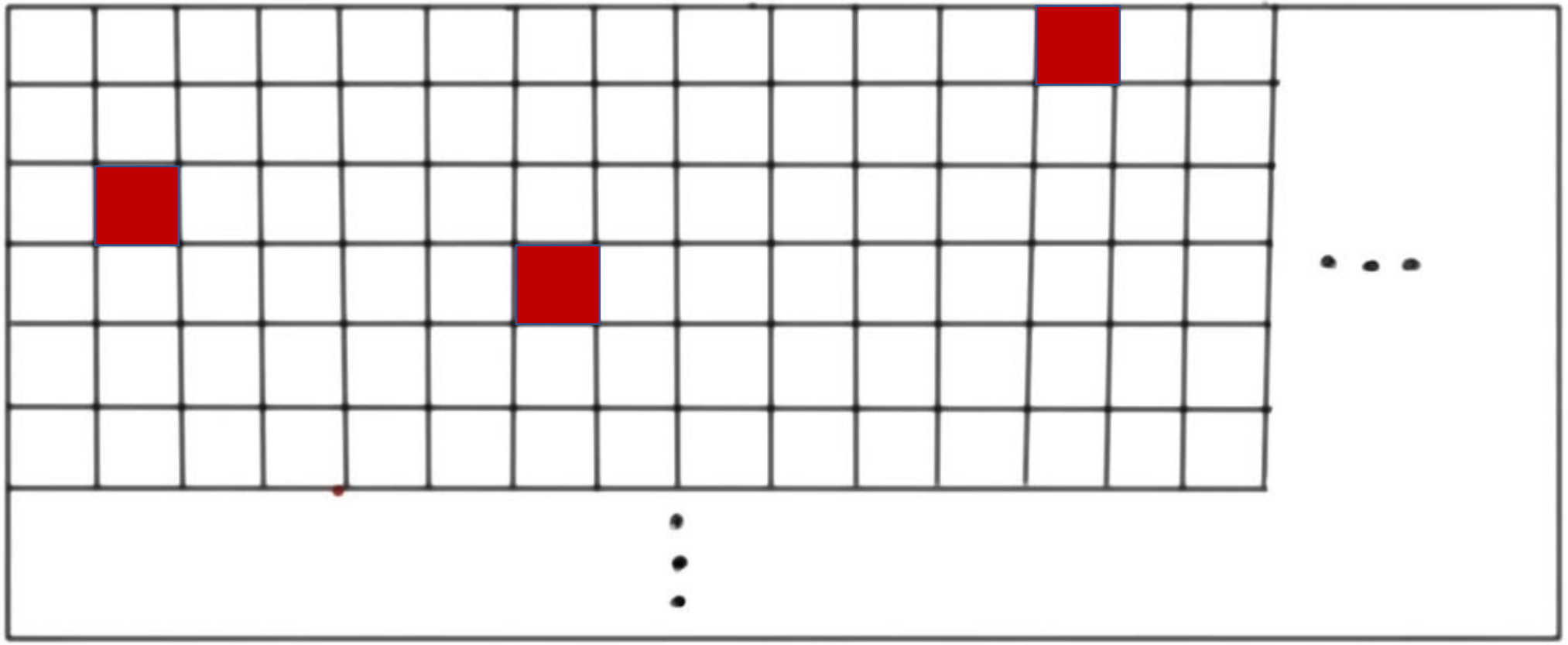
Advantages

- Pick inputs uniformly
- All inputs considered equal
- No designer bias (developer may develop code based on an assumption, test cases may also be biased)

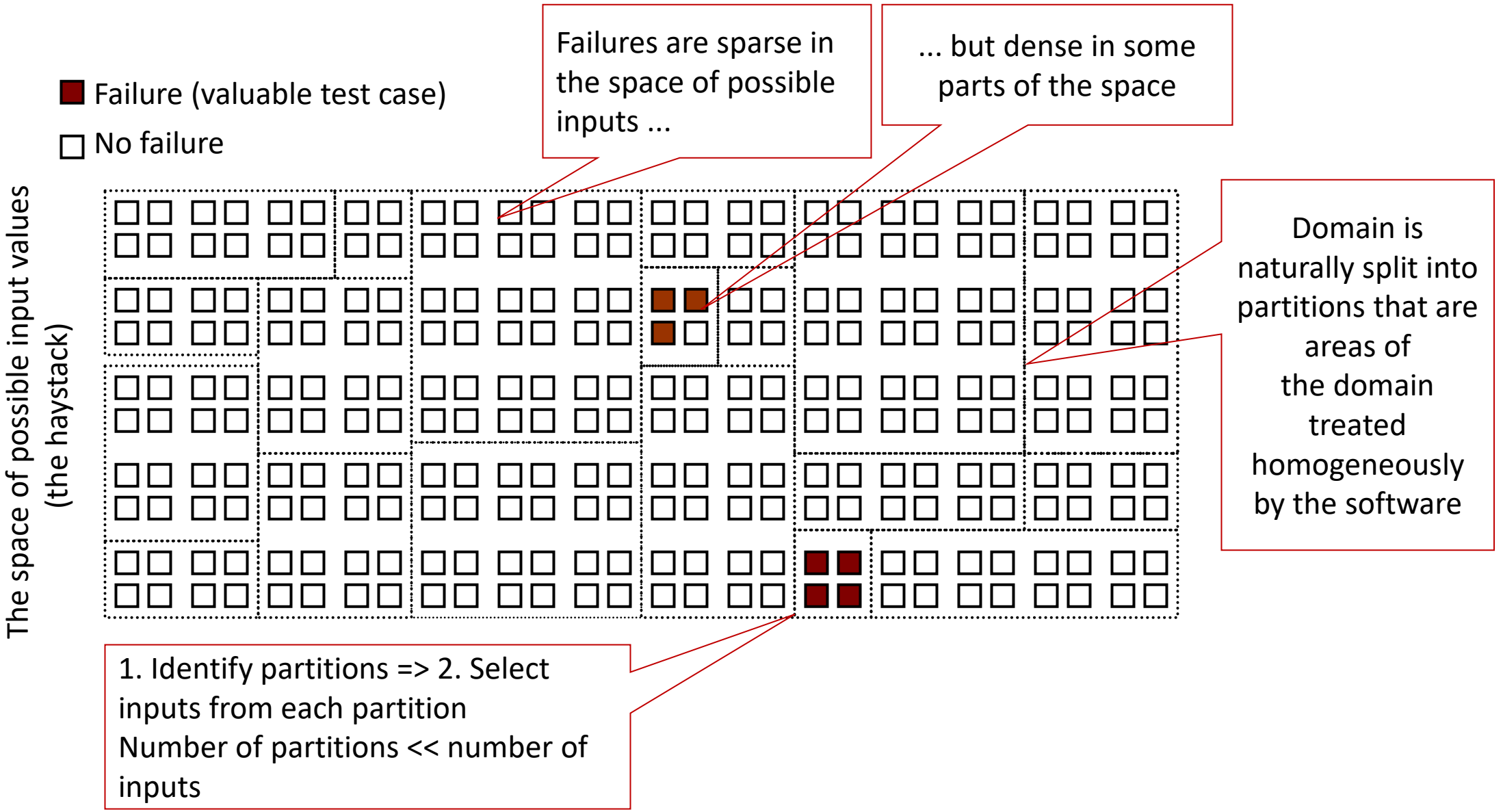
So why not random?

Same as finding many needles in a haystack

So why not random?



Systematic Partition Testing



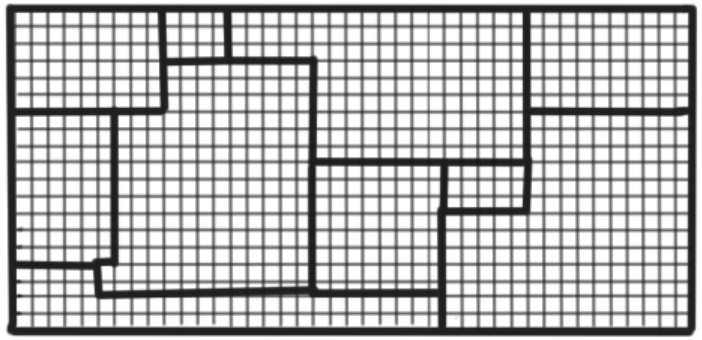
Example

split (string Str, int Size)

1. Identify partitions:

- Size < 0 (Designer bias might let you not pick this partition)
- Size = 0
- Size > 0
- Str with length < Size
- Str with length in [Size,Size*2]
- Str with length > Size*2
- …

Boundary Values



2. Select **interesting** Inputs from each partition

Basic Idea: Errors tend to occur at the boundary of a sub-domain

=> Select inputs at these boundaries

Select interesting Inputs from each partition: Example

split (string Str, int Size)

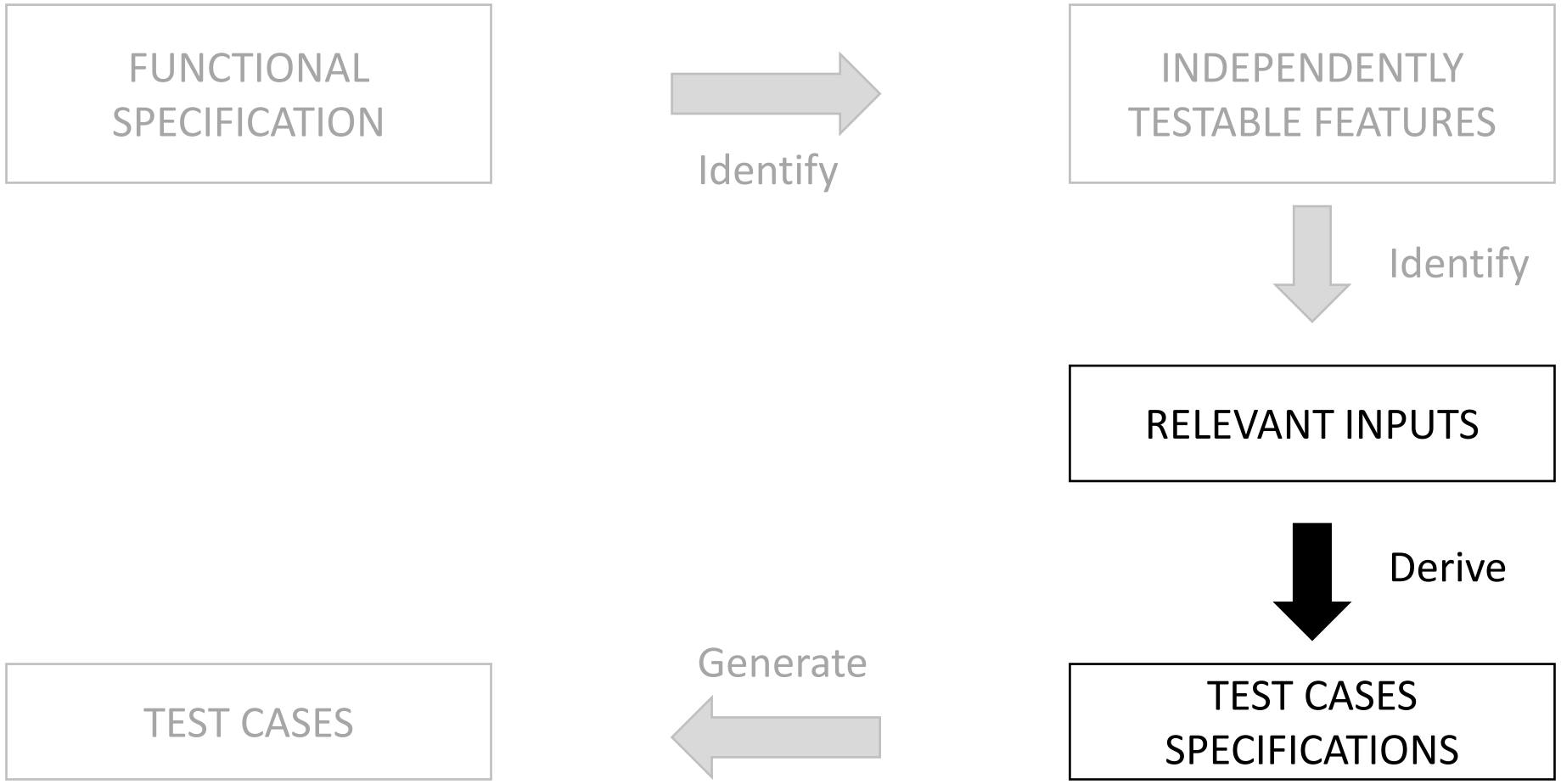
Some possible partitions:

- Size < 0
- Size = 0
- Size > 0
- Str with length < Size
- Str with length in [Size, Size*2]
- Str with length > Size*2

Some possible inputs:

- Size = -1
- Size = 1
- Size = MAXINT
- Str with length = Size- 1
- Str with length = Size
- …

A systematic Functional-Testing Approach



3. Generate Test Case Specifications: Example

split (string Str, int Size)

Some possible inputs:

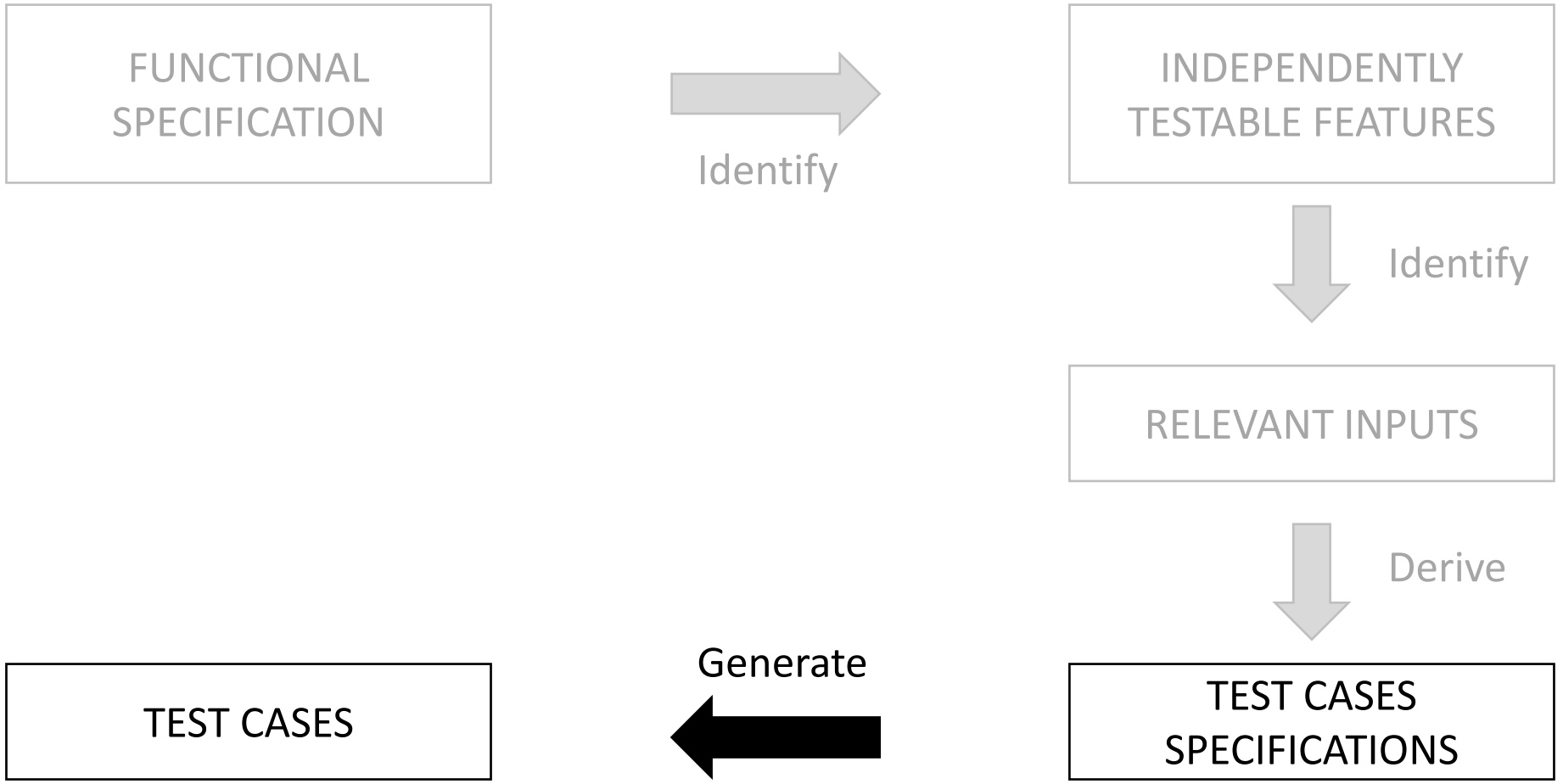
- Size = -1
- Size = 1
- Size = MAXINT

- Str with length = Size- 1
- Str with length = Size
- …

Test Case Specifications: (combine input values)

- ~~Size = -1, Str with length = -2~~
- ~~Size = -1, Str with length = -1~~
- Size = 1, Str with length = 0
- Size = 1, Str with length = 1
- …

A systematic Functional-Testing Approach



Slide adapted from Alessandro Orso

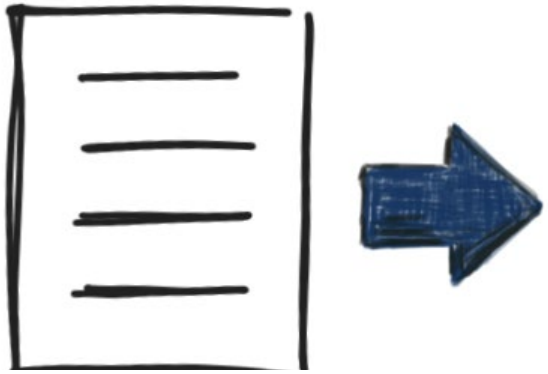
A Specific Functional Testing Black-Box Approach
The Category-Partition Method

[Ostrand & Balcer, CACM, June 1988]

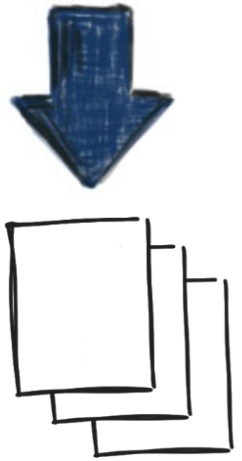
Specification

Test Cases

The Category-Partition Method



1. Identify independently testable features
2. Identify Categories
3. Partition Categories into choices
4. Identify constraints among choices
5. Produce/Evaluate test case specifications
6. Generate test cases from test case specifications



Test Cases

Identify Categories

Characteristics of each input element

split (string Str, int Size)

Input Str

- Length
- Content

Input Size

- value

Partition Categories into choices

Interesting cases (subdomains) – boundary values

split (string Str, int Size)

Input Str

- Length
  - 0
  - Size-1
- Content
  - Only Spaces
  - Special characters

Input Size

- Value
  - 0
  - >0
  - <0
  - MAXINT
  - …

Identify Constraints among choices

To Eliminate meaningless combinations & To reduce number of test cases

Three types: PROPERTY---- IF, ERROR, SINGLE

Input Str

- Length
  - 0 PROPERTY zerovalue
- Content
  - Special characters If !zerovalue

Input Size

- Value
  - <0 ERROR
  - MAXINT SINGLE

Produce And Evaluate Test Case Specifications

Can be automated

Produces test frames

Example (specify the characteristic of the inputs for that test)

Test frame #45
- Input Str
  - length: size -1
  - content: special characters
- Input Size
  - value: >0

Produce and evaluate test case specification
-how many test frames?
-add additional constraints to reduce the number if required

Generate Test Cases from Test Case Specification

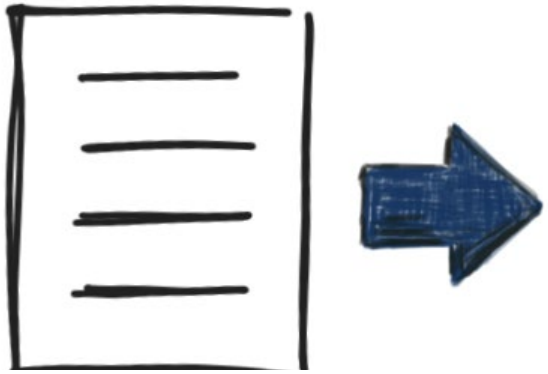
Simple Instantiation of frames

Final result: Set of concrete tests

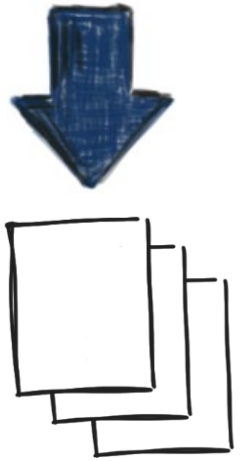
Example (specify the characteristic of the inputs for that test)

Test case #45
Str = “ABCC!\n\t”
Size = 10

The Category-Partition Method



1. Identify independently testable features
2. Identify Categories
3. Partition Categories into choices
4. Identify constraints among choices
5. Produce/Evaluate test case specifications
6. Generate test cases from test case specifications



Test Cases

Category Partition DEMO TIME

- Use category partition to generate test frames from a specification file (with categories, partitions, and constraints)
- Tool called TSLgenerator is used: Developed by team at UC Irvine, Oregon State, and Georgia Tech
- Download from: https://github.com/alexorso/tslgenerator/tree/master/Binaries
- run the code from command prompt: **./TSLgenerator-win8.exe**
- For help: **./TSLgenerator-win8.exe –manpage**
- To get number of test cases and write the test frames against your specification file: **./TSLgenerator-win8.exe -c *filename***

How can we use AI tools?

1. Using LLM for Test Case Generation

- **Description:** GPT/Llama/Gemini/Claude can generate coherent, contextually relevant text based on prompts. This capability can be utilized to create detailed test cases from a set of functional requirements written in plain English, significantly speeding up the test design process.
- **Example:** For a feature that allows users to book flights, you might have a requirement: “Given --- project description, one of the requirements is “The user should be able to select a departure and return date using a calendar widget.“ Generate test cases for this requirement”. From this, LLM can generate a series of test cases, such as:
  - Test the functionality of the calendar widget for date selection.
  - Check the behavior when no dates are selected.
  - Verify the system's response when past dates are selected.

How can we use AI tools?

2. Selenium with AI Extensions

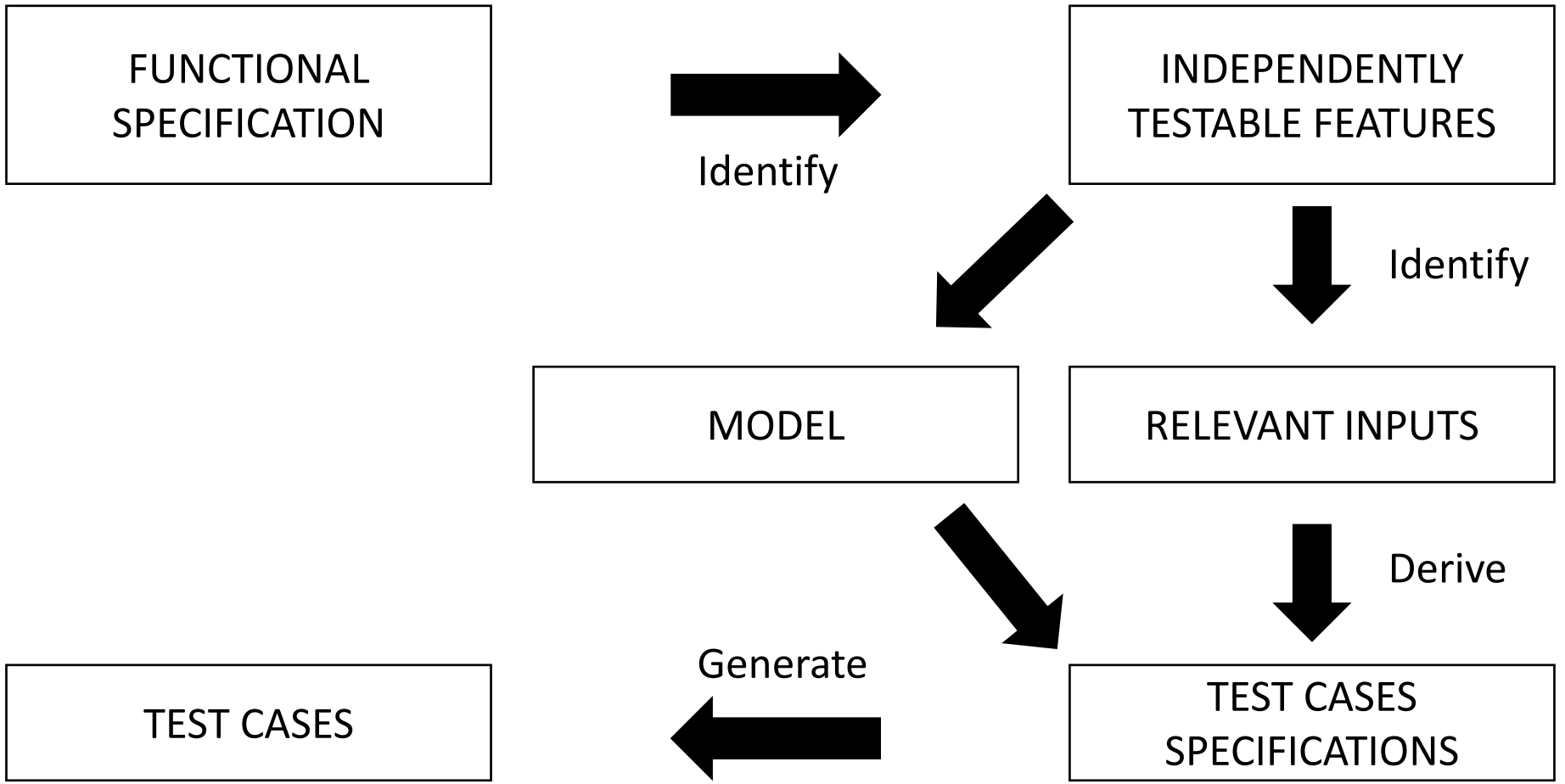
- **Description:** Selenium is a tool for automating web browsers, allowing it to mimic user actions on a webpage. AI extensions (test.ai, testRigor, Selenium Sage, Parasoft Selenic) can enhance Selenium by predicting UI changes and optimizing test flows based on previous test runs, reducing test maintenance.
- **Example:** For a web application's login page:
- Typical Selenium Test:
  - Enter valid credentials and verify successful login.
  - Enter invalid credentials and check for error messages.
- With AI Extensions:
  - The AI identifies frequent UI changes, like button relocations or text field adjustments, and dynamically updates the Selenium selectors before running tests.
  - Predicts potential fail points like heavy load times for login button clicks during peak hours and adjusts test parameters dynamically.

How can we use AI tools?

3. Postman and Postbot for API Testing

- **Description:** Postman is a popular tool for API testing that allows users to construct complex HTTP requests, examine the responses, and automate testing through scripts. Postbot, Postman’s AI assistant, can generate test scripts based on the user's API schema or past test scripts.
- **Example**: For an API endpoint that retrieves user profiles:
  - API Endpoint: GET /api/user/{userID}
  - Typical Test Cases Using Postman:
    - "Validate response with a valid userID.“
    - "Check response for a userID that does not exist.“
  - AI-Generated Test Cases by Postbot:
    - "Test for SQL injection vulnerabilities by inputting SQL code as the userID.“
    - "Verify the handling of unusually long user ID strings."

Model-Based Testing



Slide adapted from Alessandro Orso

Model-Based Testing

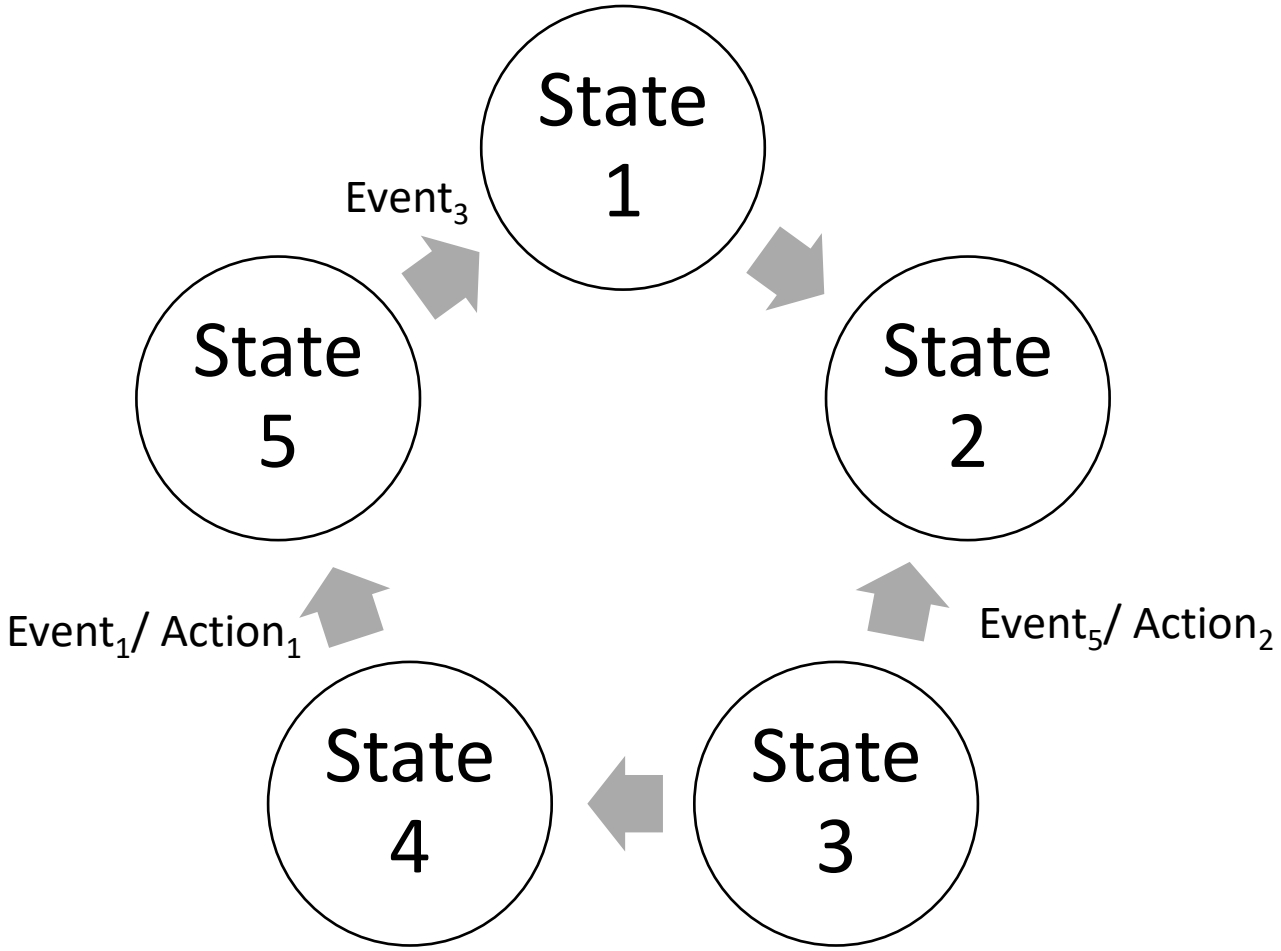
Specification

Model
(Abstract representation
of the software)

Test Cases

Finite State Machines (FSM)

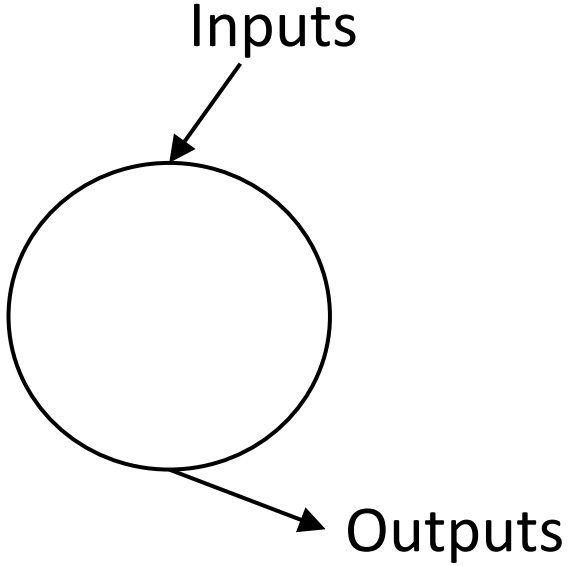
Nodes = States
Edges = Transitions
Edge Labels = Events/Actions



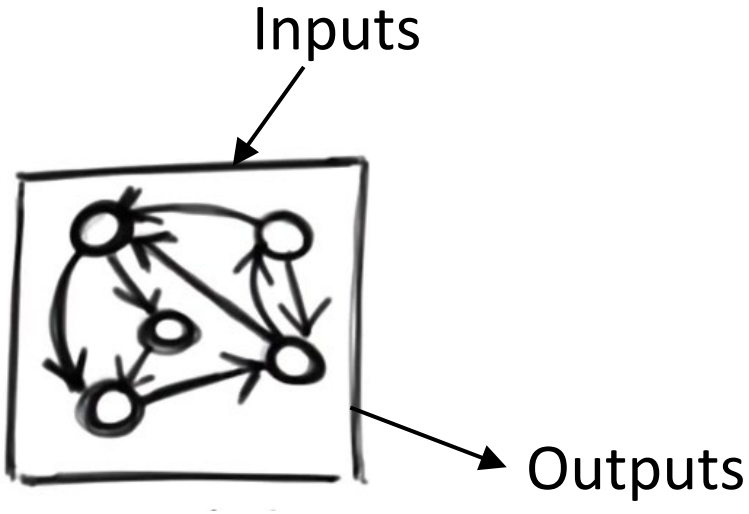
Building an FSM

Specification

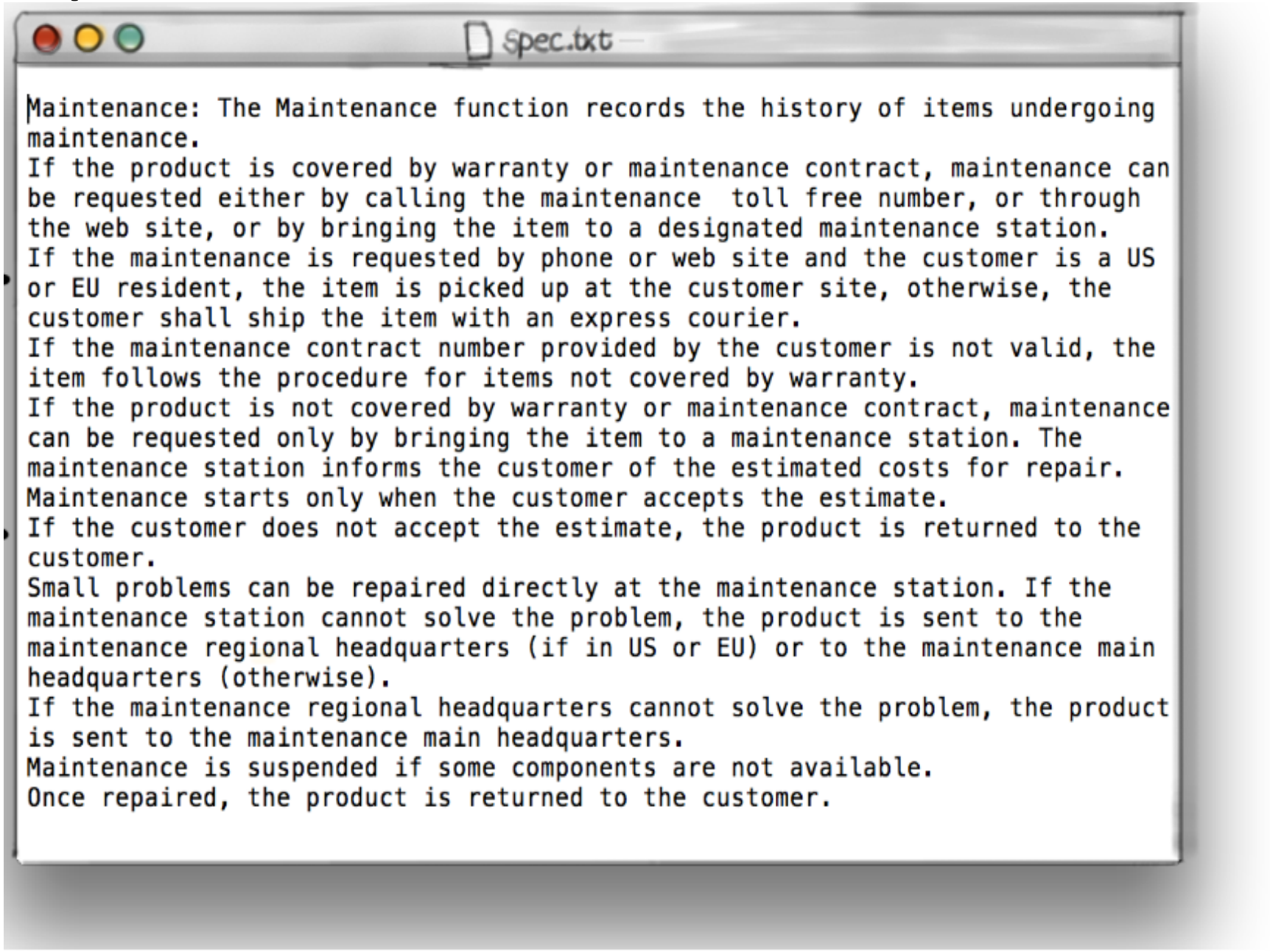
Identify System's Boundaries, and Input and Output



Identify relevant states and transitions

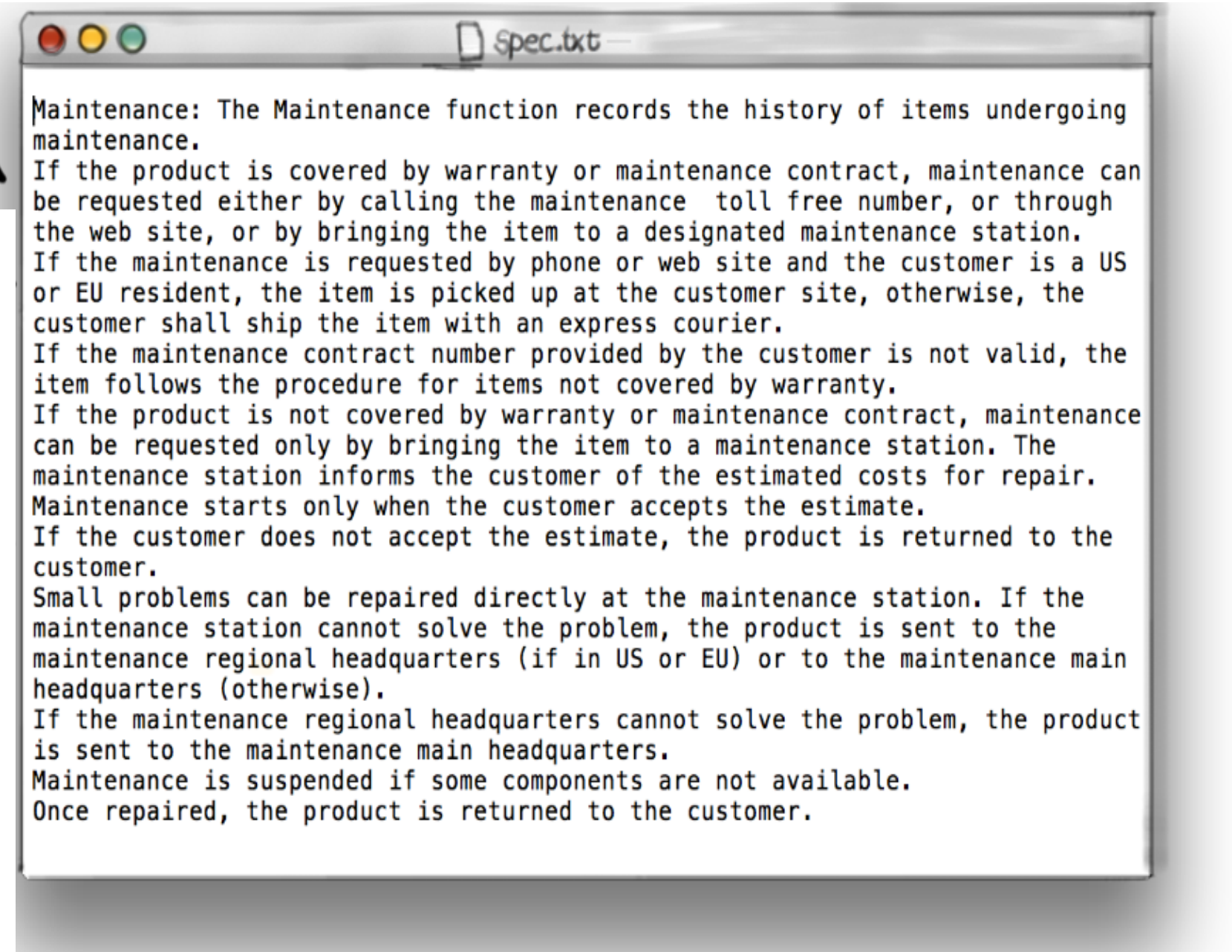


From an Informal Specification…

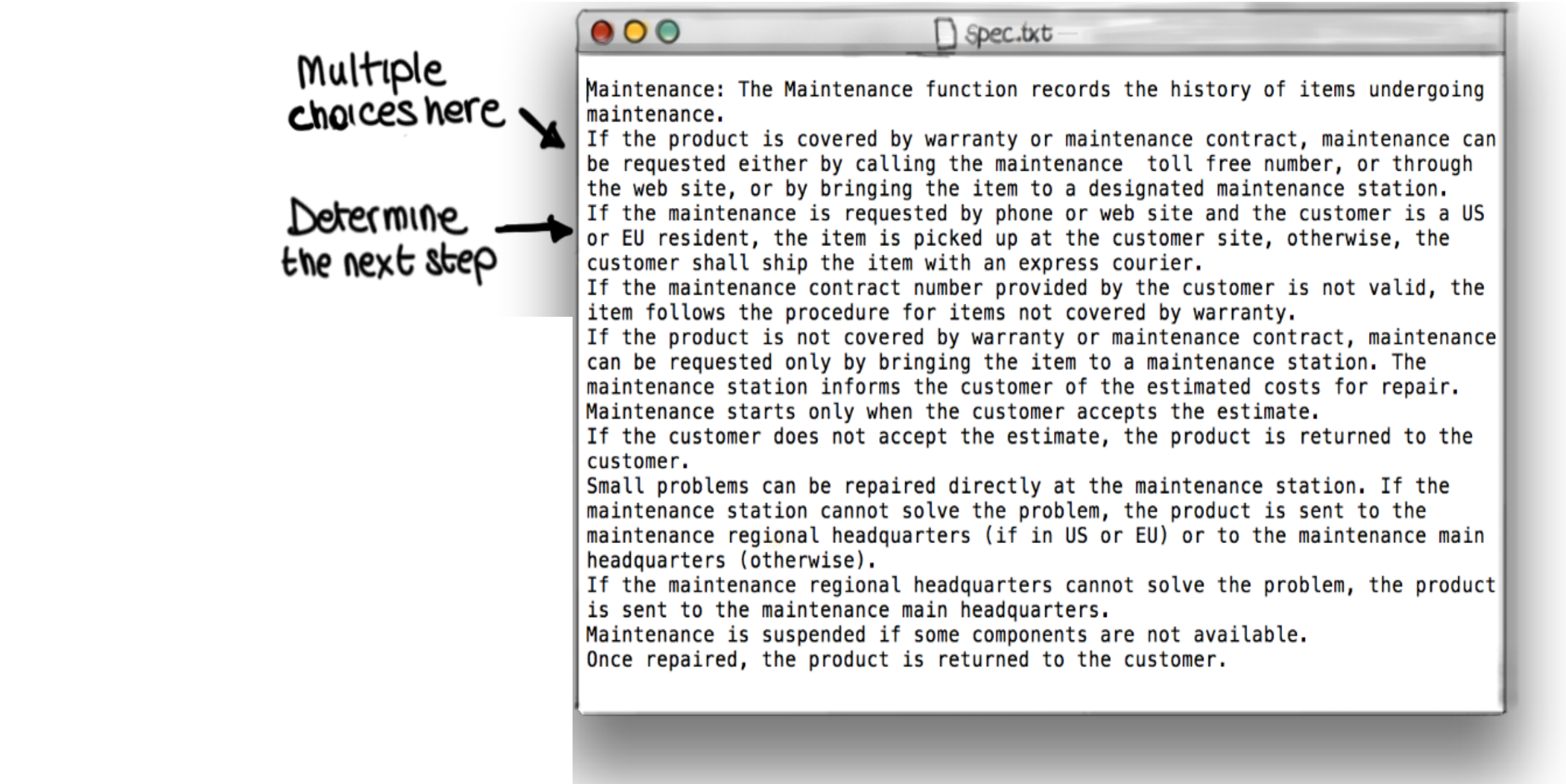


From an Informal Specification…

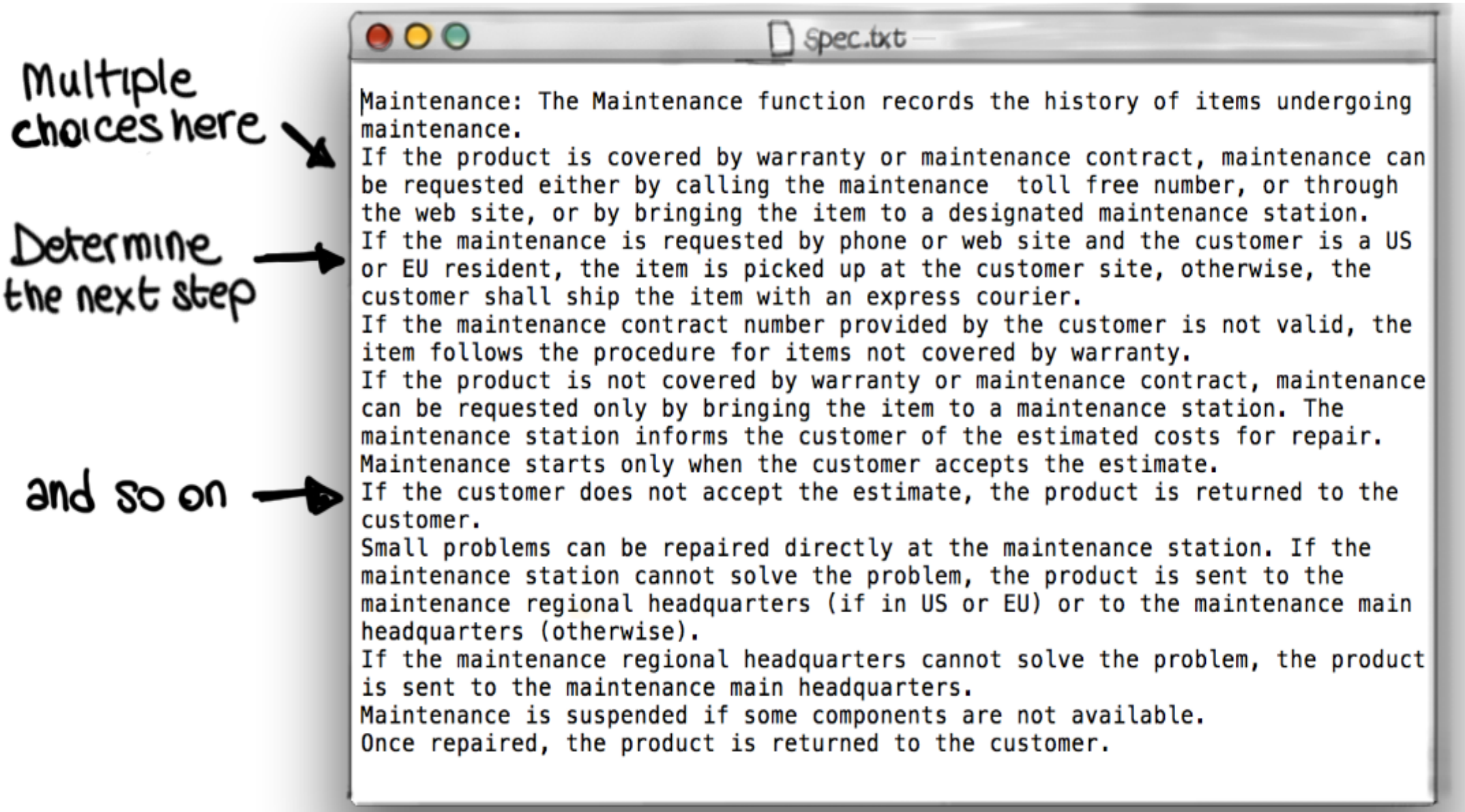
Multiple choices here



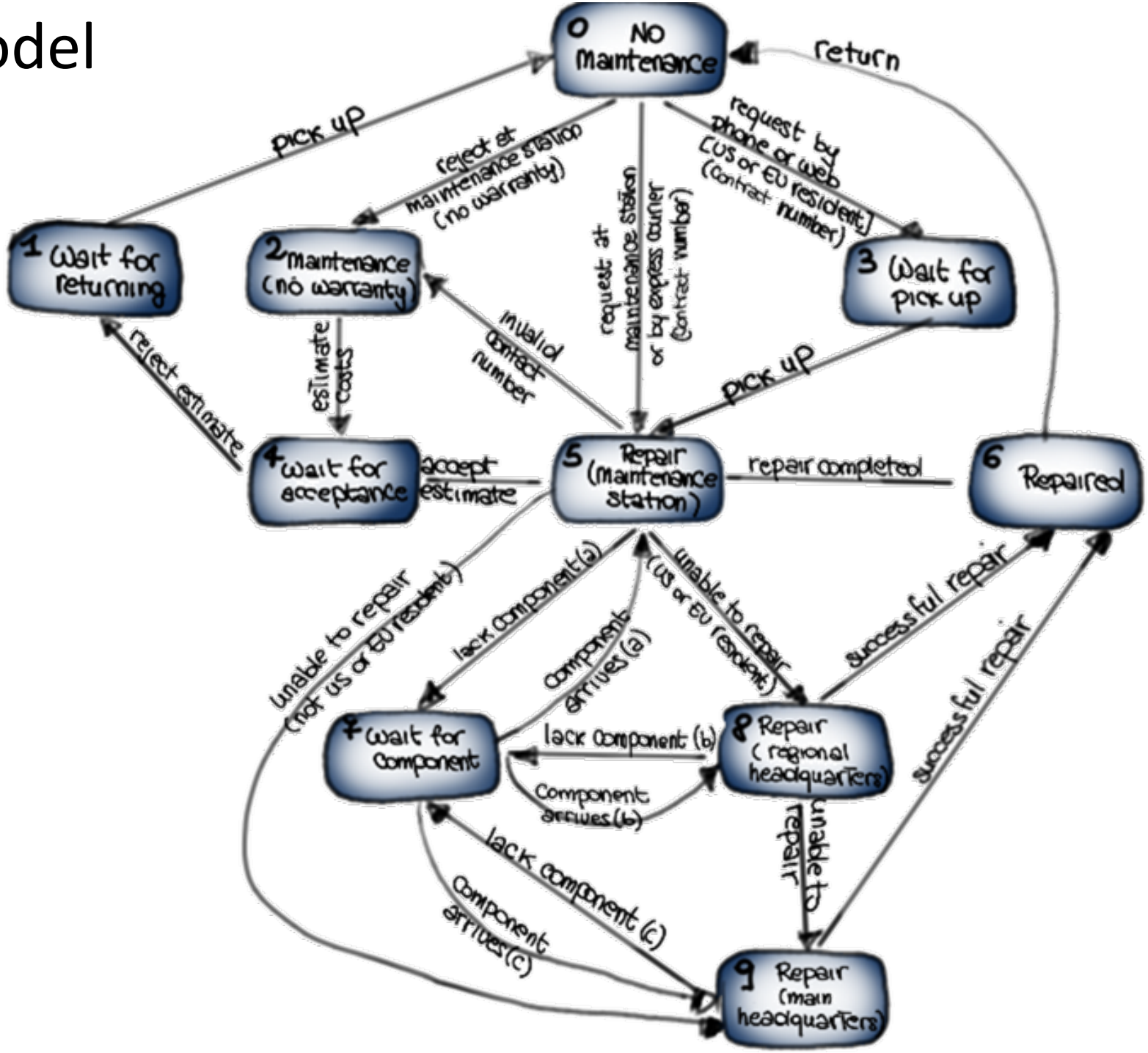
From an Informal Specification…



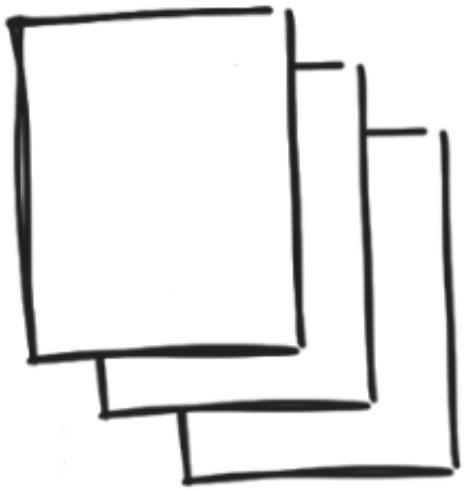
From an Informal Specification…



… To a Finite State Model



Finite State Model to a Set of Test Cases

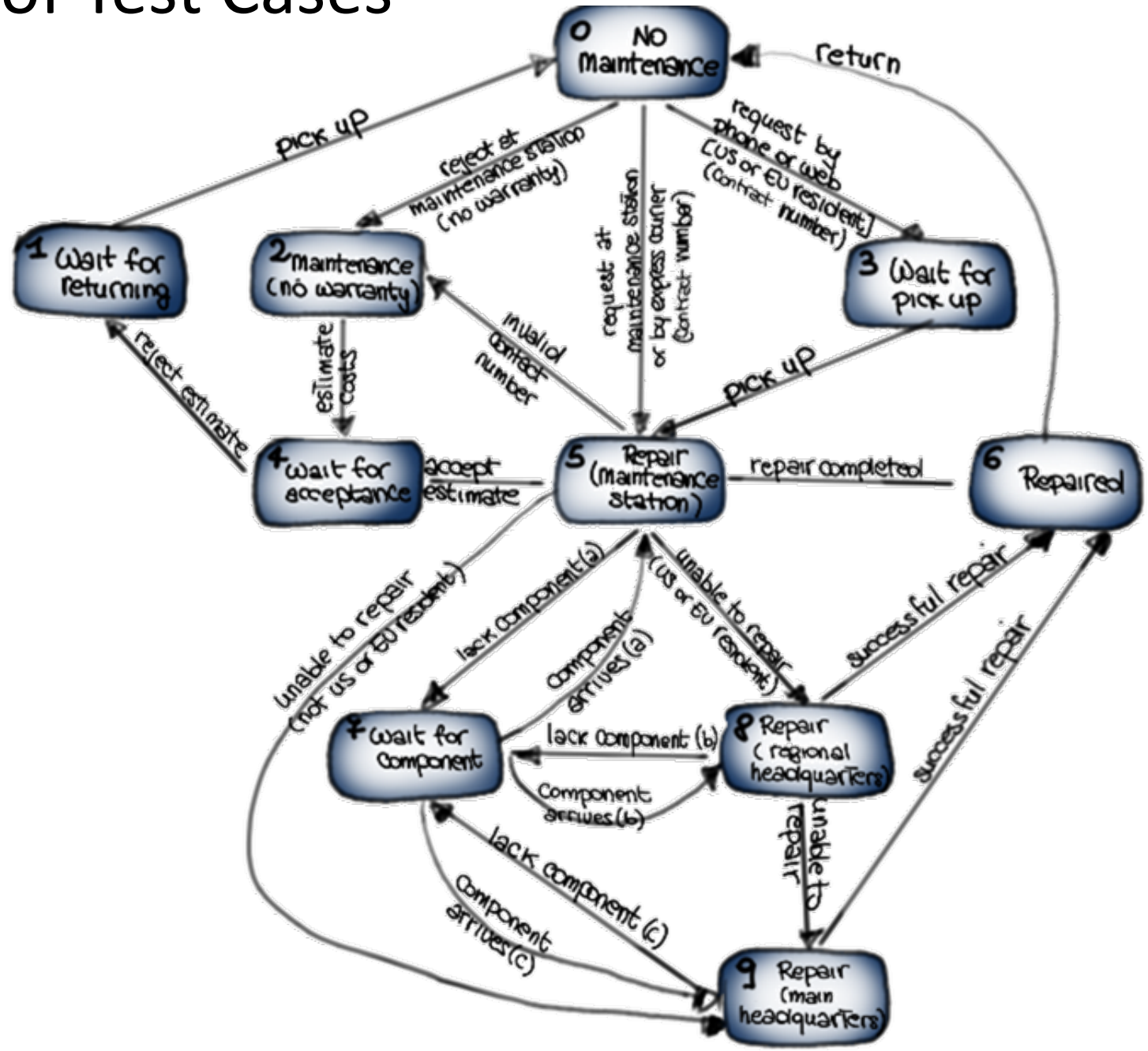


Test Cases

Finite State Model to a Set of Test Cases

Cover the behaviors represented by the state machine.

- Cover all the states
- Or Identify paths in state machine that go through all states in the machine

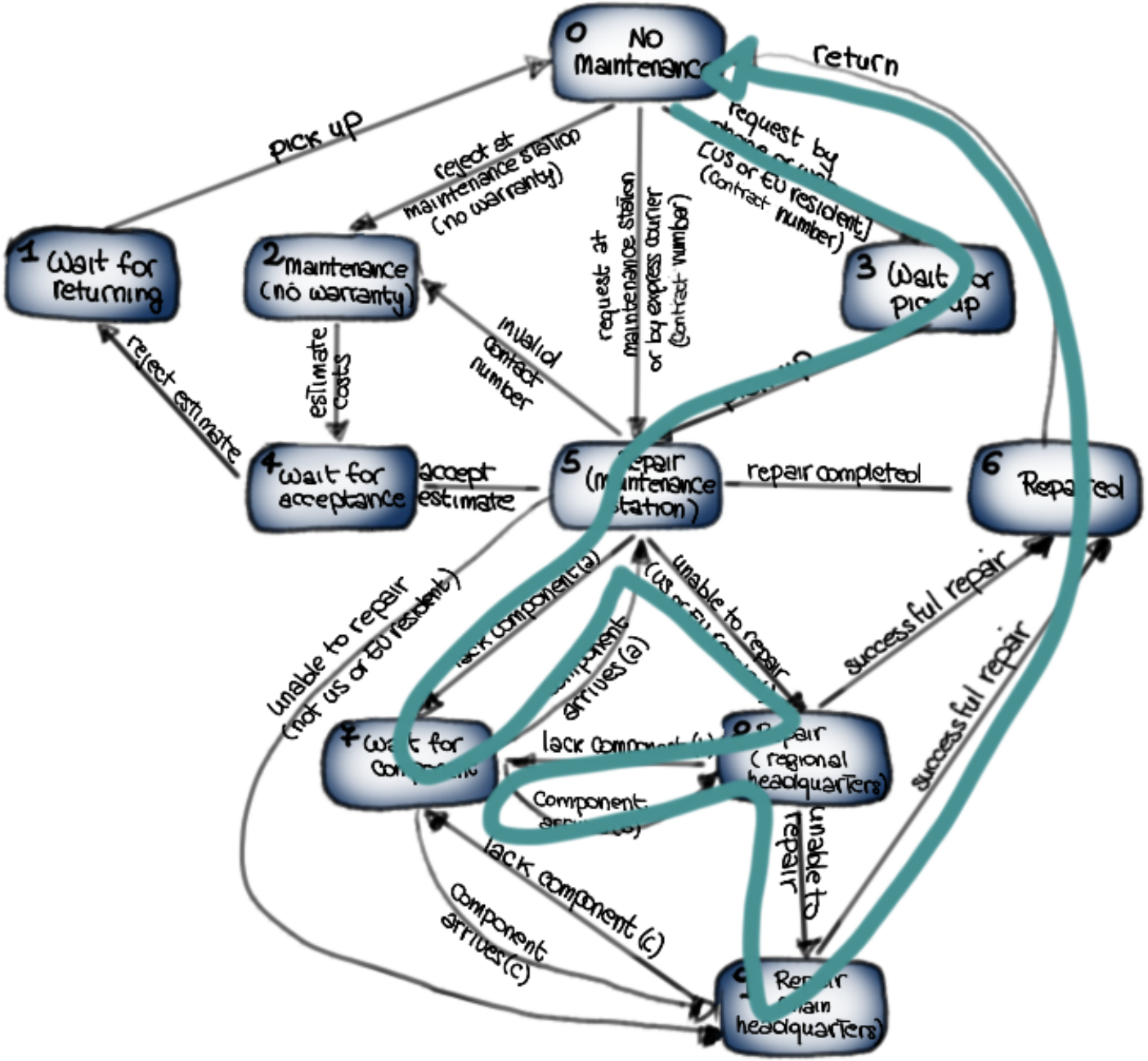


Finite State Model to a Set of Test Cases

Cover the behaviors represented by the state machine.

- Cover all the states
- Or Identify paths in state machine that go through all states in the machine

TC1: Φ, 3, 5, 7, 5, 8, 7, 8, 9, 6, Φ



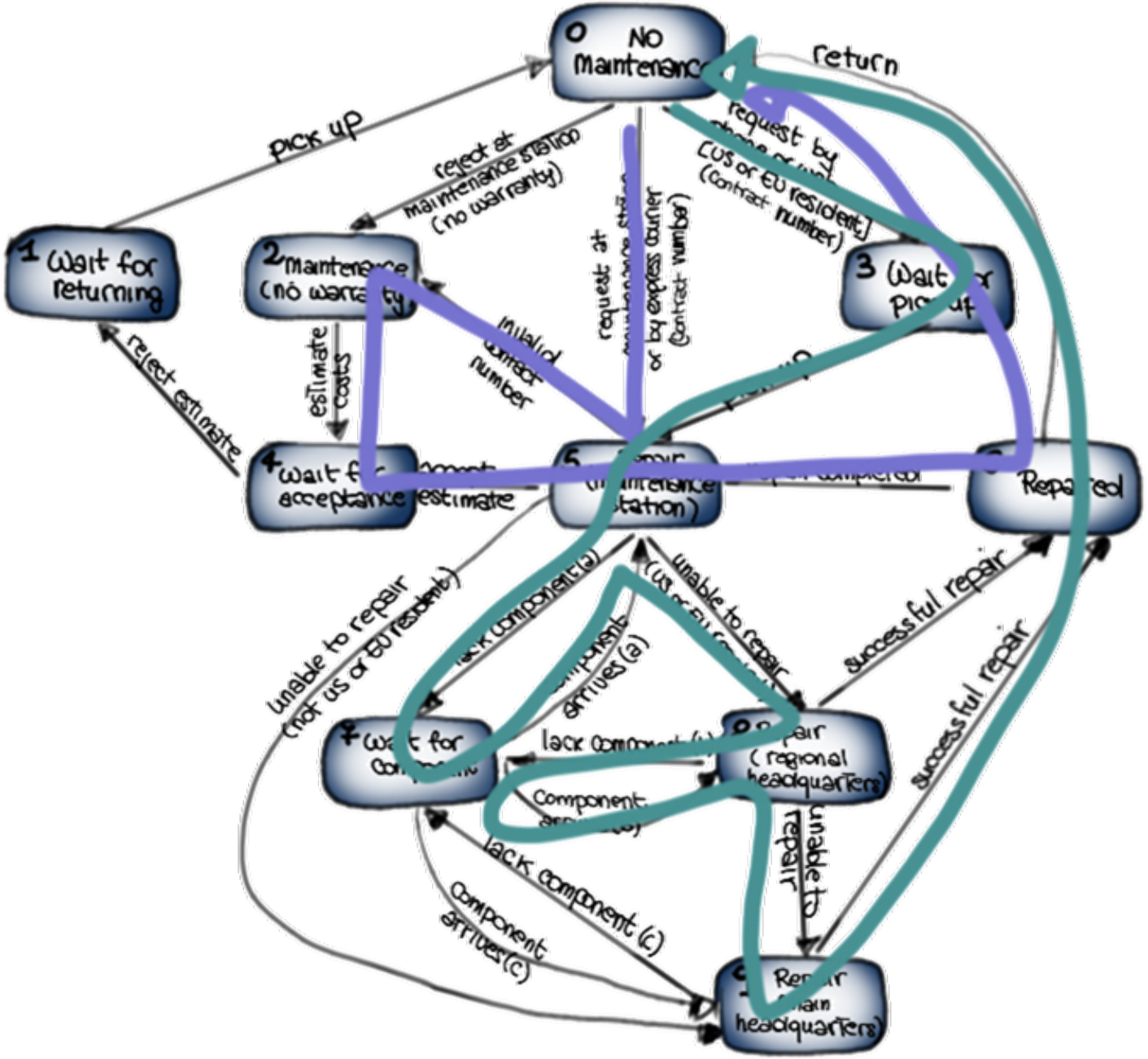
Finite State Model to a Set of Test Cases

Cover the behaviors represented by the state machine.

- Cover all the states
- Or Identify paths in state machine that go through all states in the machine

TC1: Φ, 3, 5, 7, 5, 8, 7, 8, 9, 6, Φ

TC2: Φ, 5, 2, 4, 5, 6, Φ



Finite State Model to a Set of Test Cases

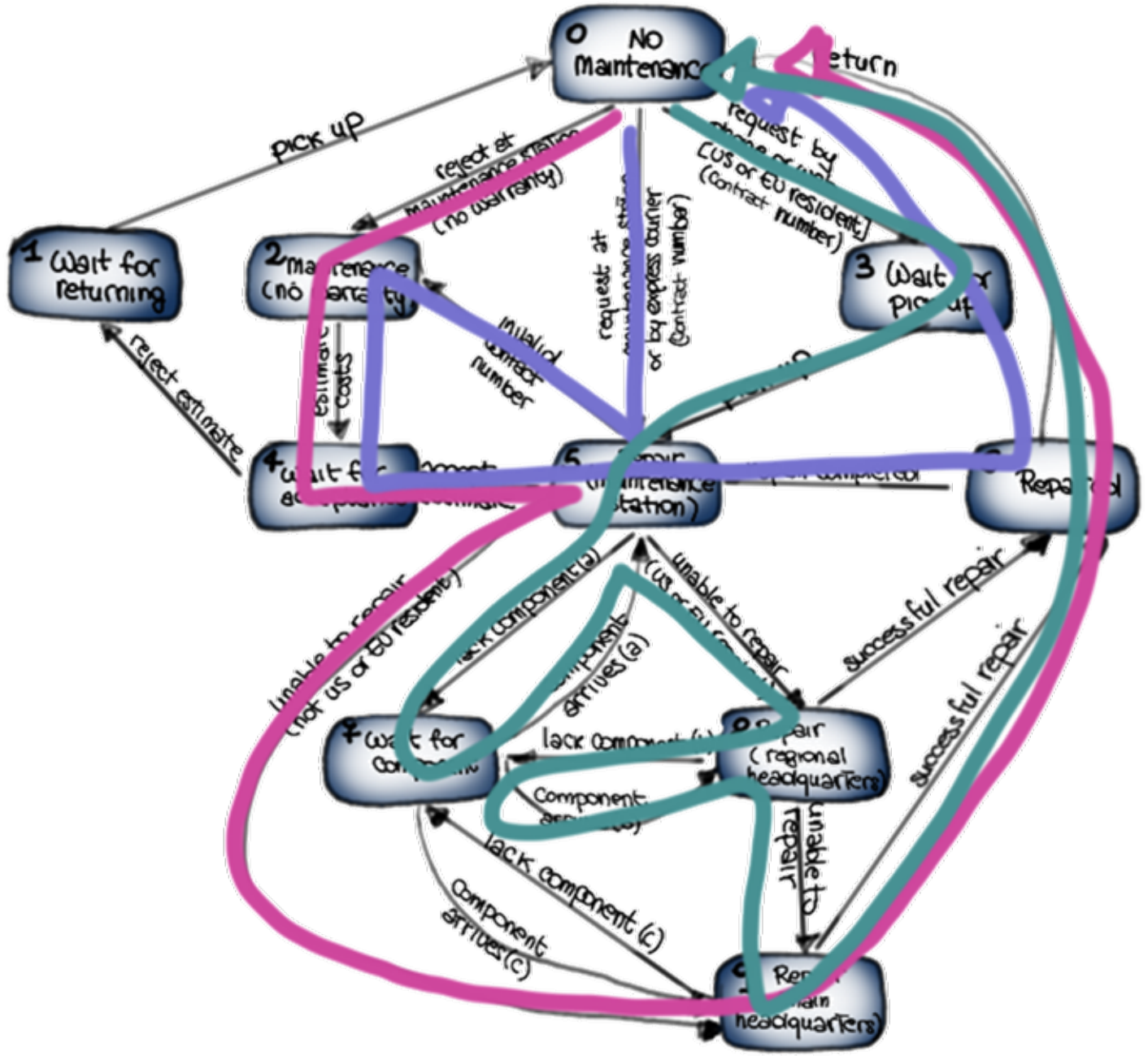
Cover the behaviors represented by the state machine.

- Cover all the states
- Or Identify paths in state machine that go through all states in the machine

TC1: Φ, 3, 5, 7, 5, 8, 7, 8, 9, 6, Φ

TC2: Φ, 5, 2, 4, 5, 6, Φ

TC3: Φ, 2, 4, 5, 9, 6, Φ



Finite State Model to a Set of Test Cases

Cover the behaviors represented by the state machine.

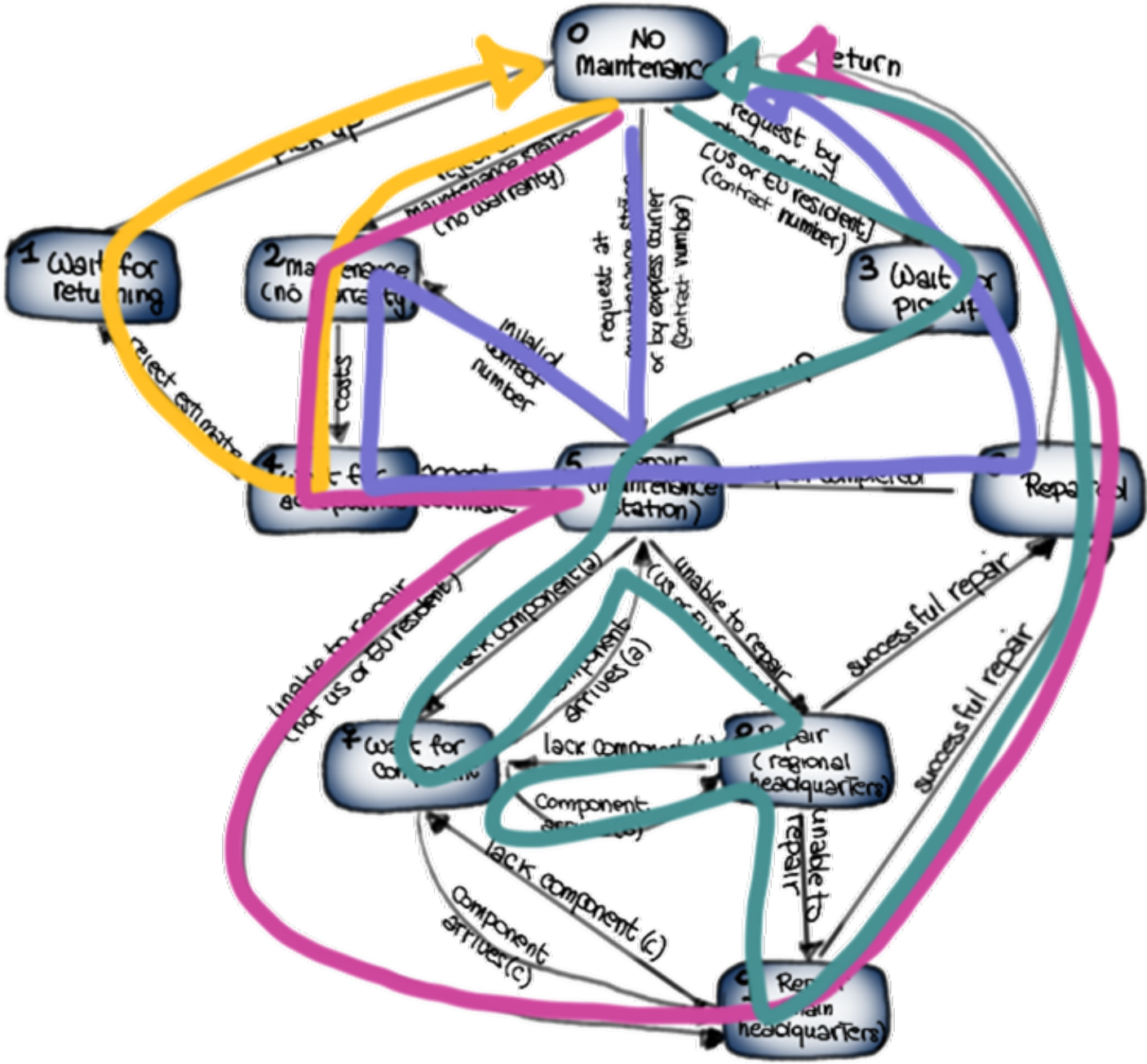
- Cover all the states
- Or Identify paths in state machine that go through all states in the machine
- Or cover all transitions

TC1: Φ, 3, 5, 7, 5, 8, 7, 8, 9, 6, Φ

TC2: Φ, 5, 2, 4, 5, 6, Φ

TC3: Φ, 2, 4, 5, 9, 6, Φ

TC4: Φ, 2, 4, 1, Φ



Finite State Model to a Set of Test Cases

Cover the behaviors represented by the state machine.

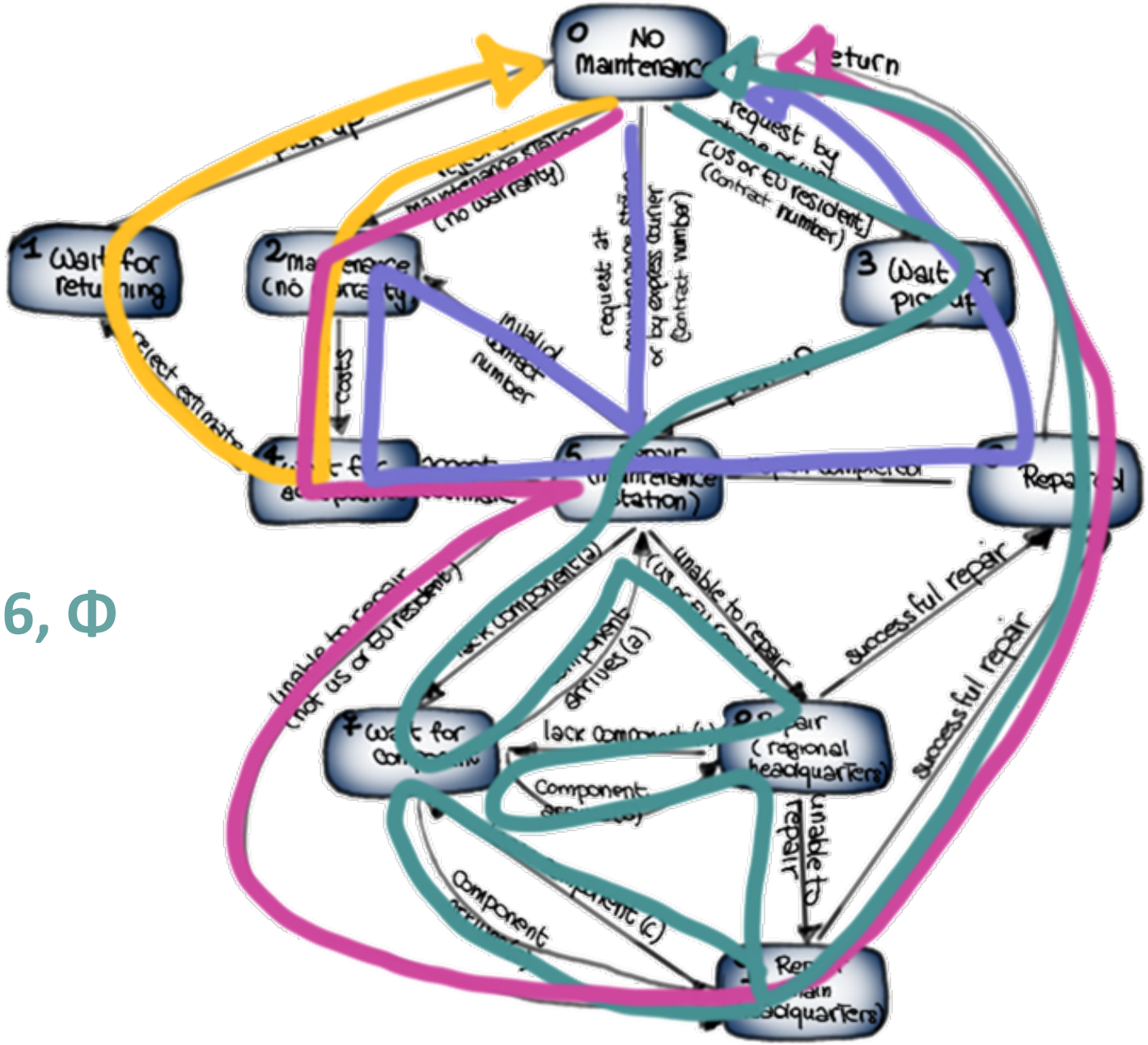
- Cover all the states
- Or Identify paths in state machine that go through all states in the machine
- Or cover all transitions

TC1: Φ, 3, 5, 7, 5, 8, 7, 8, 9, 7, 9, 6, Φ

TC2: Φ, 5, 2, 4, 5, 6, Φ

TC3: Φ, 2, 4, 5, 9, 6, Φ

TC4: Φ, 2, 4, 1, Φ



Finite State Model to a Set of Test Cases

Cover the behaviors represented by the state machine.

- Cover all the states
- Or Identify paths in state machine that go through all states in the machine
- Or cover all transitions

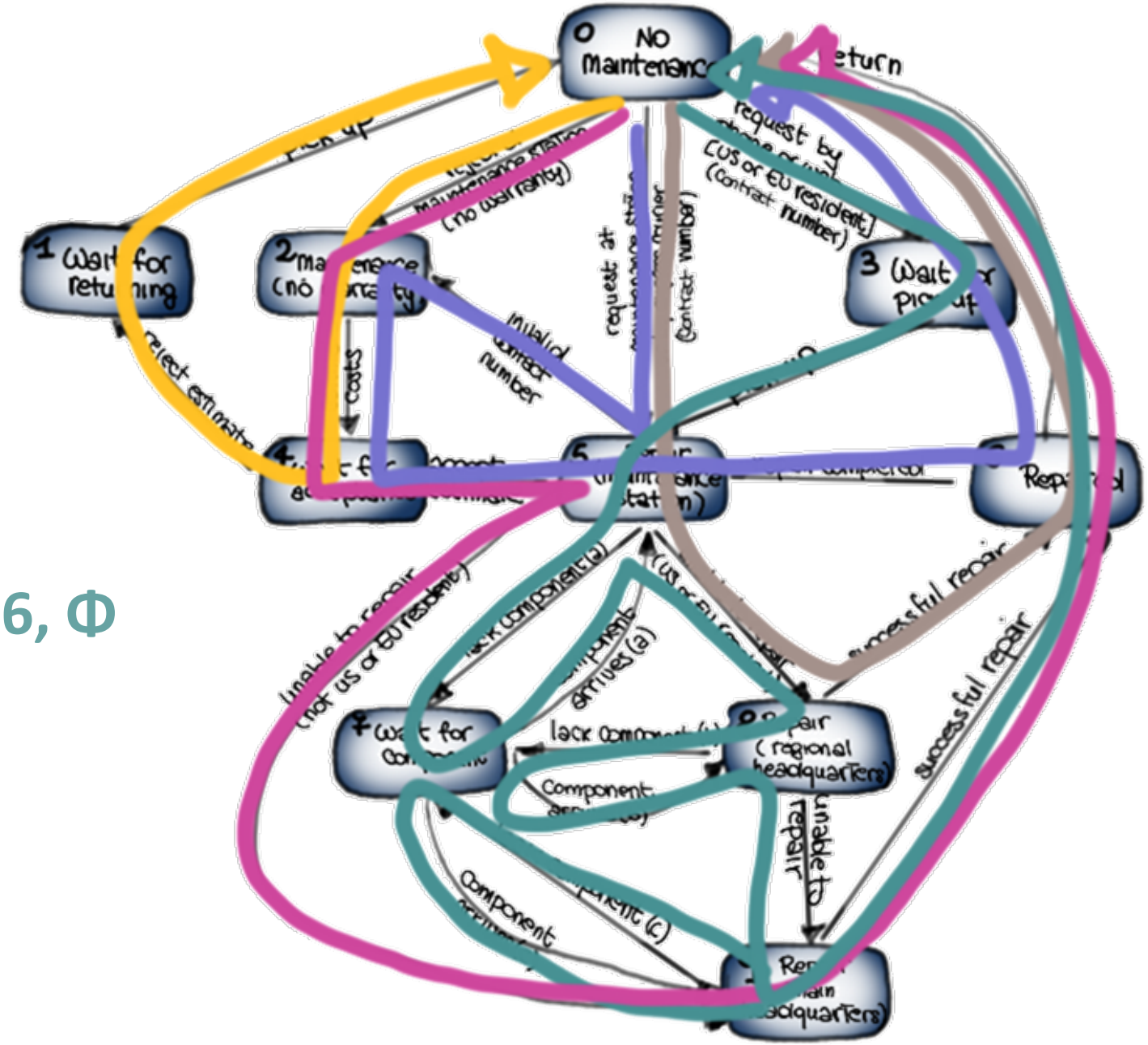
TC1: Φ, 3, 5, 7, 5, 8, 7, 8, 9, 7, 9, 6, Φ

TC2: Φ, 5, 2, 4, 5, 6, Φ

TC3: Φ, 2, 4, 5, 9, 6, Φ

TC4: Φ, 2, 4, 1, Φ

TC5: Φ, 5, 8, 6, Φ



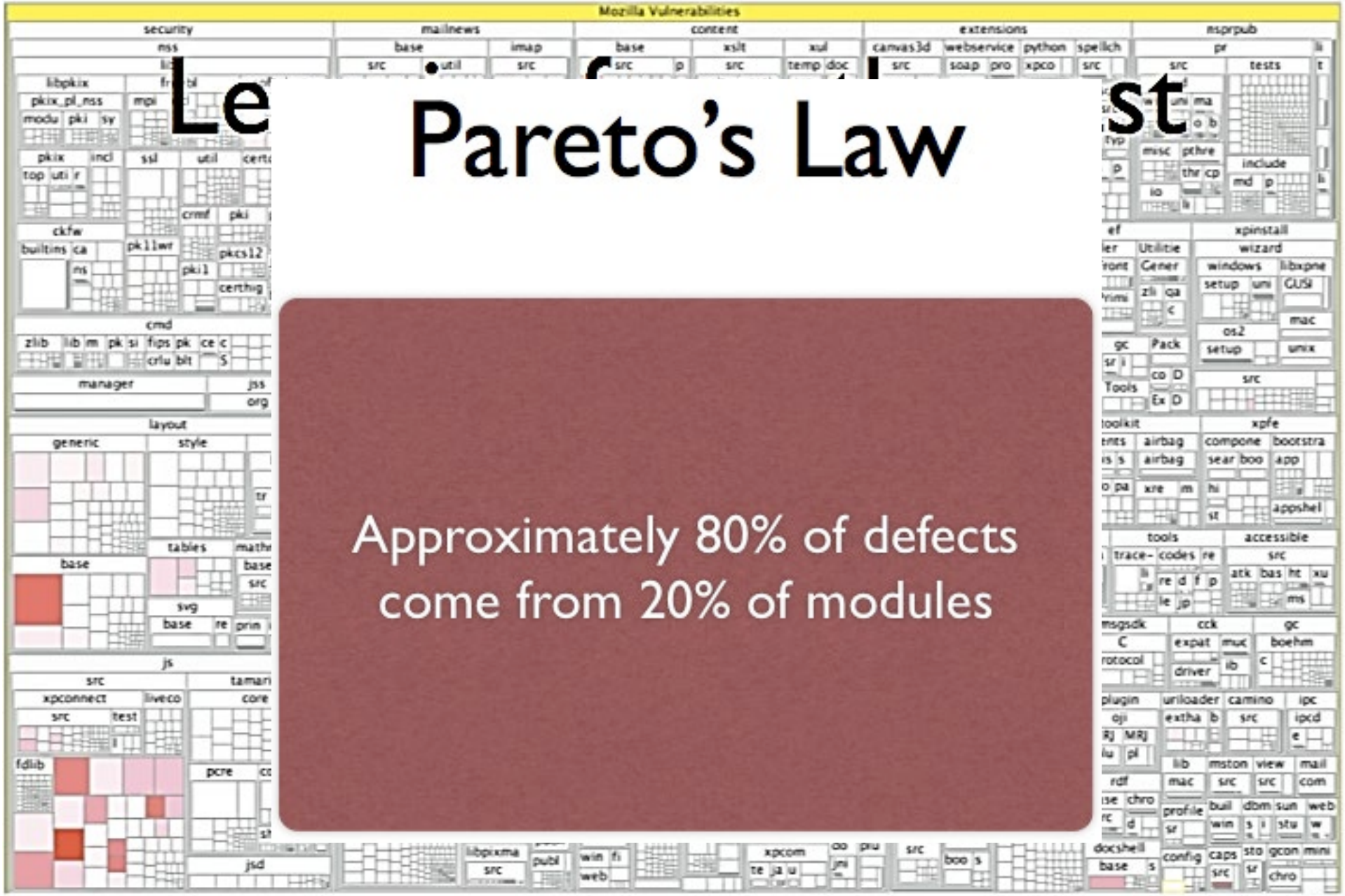
Some Considerations

Applicability
- Very General Approach
- In UML, state machine are readily available

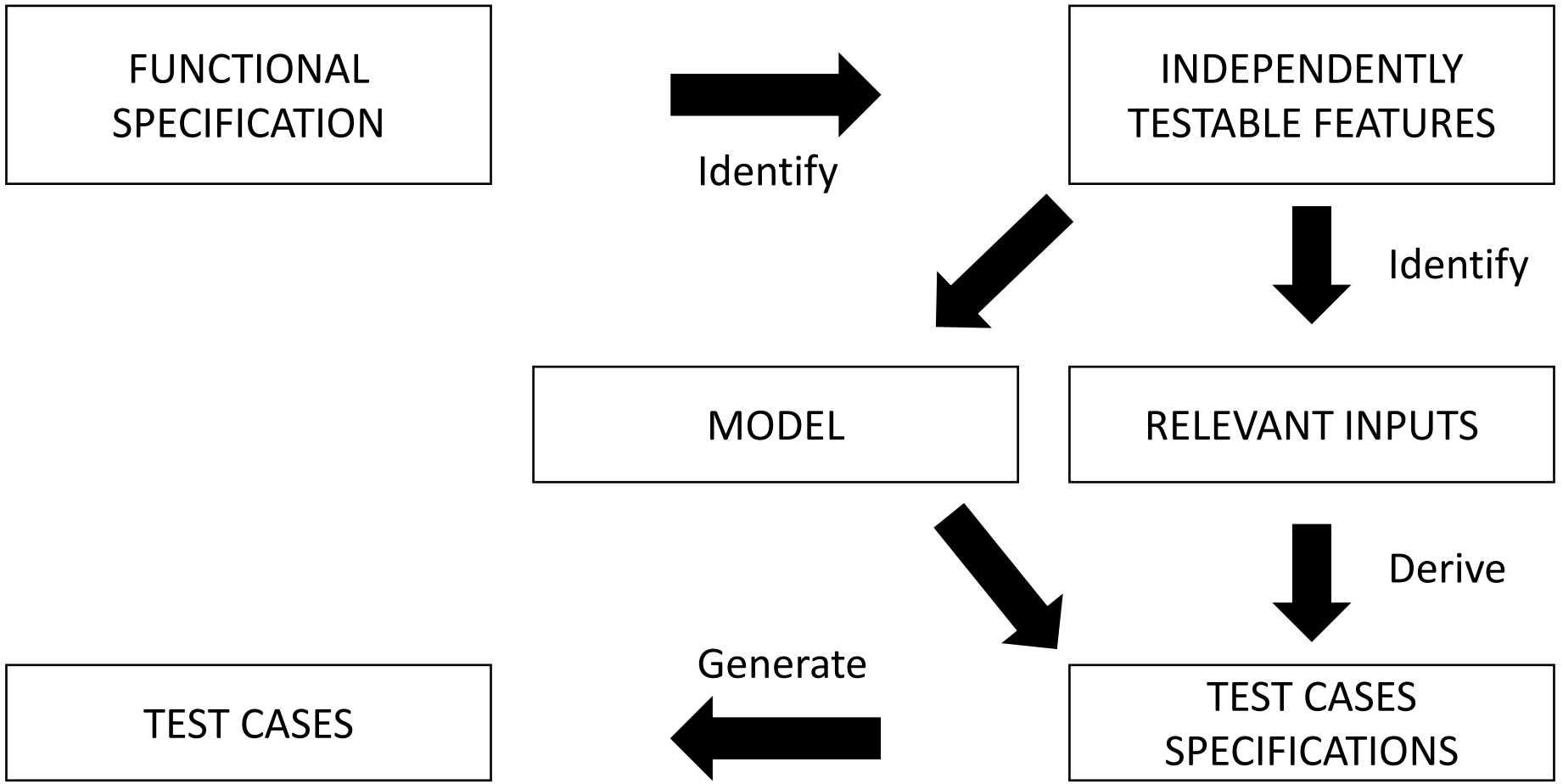
Abstraction is key – right balance between abstracting and expensive testing

Many other approaches
- Decision tables
- Flow graphs
- Historical Models
- …

Historical models – Distribution of security vulnerabilities in Firefox



Black-Box Testing



Decoupling; Automated Sub-tasks; Monitor testing process

Slide adapted from Alessandro Orso

Industry Standards Today

- Many techniques of Black Box Testing
  - 2 important techniques discussed in class- Category partition & Model based testing
- Automating Black-Box Testing is very important
- Tools popular in Industry:
  - UFT (Unified Testing Tool)
    - designed for automated regression, and functional testing.
    - widely used by many big companies like Microsoft, IBM, Apple
    - difficult to use on GUI-rich browser-based applications.
  - Selenium WebDriver
    - designed as a language-independent automated black-box testing tool (open source)
    - most popular open-source solution for web application functional and regression testing
    - recording and playing back scripts, exporting and importing test cases from one language to another, debugging capabilities, and creating custom commands
  - Watir (Ruby), Ranorex Studio (commercial solution), SilkTest, Squish
  - Postman
- AI in Testing – Extensions with most standard tools available - dynamic element detection, intelligent test case generation, and predictive failure analysis.

Example demo: Postman + Postbot