UNIVERSITY OF LEICESTER

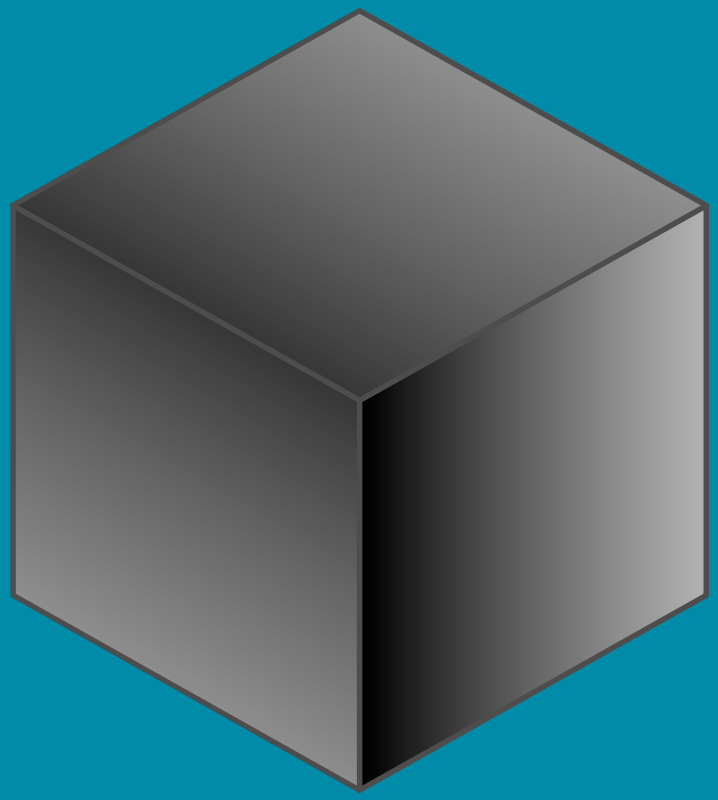
Black-Box Testing

CO3095, CO7095, CO7508

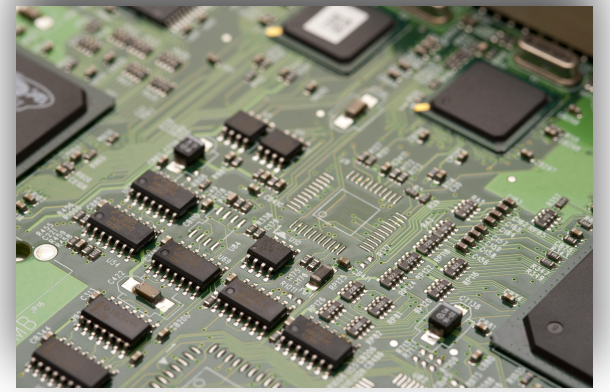
Neil Walkinshaw

Black-Box Testing

- No access to “internals”.
  - *May* have access, but don’t want to.
- We know the **interface**.
  - Parameters.
  - Possible functions / methods.
- We may have some form of specification document.



Many Scenarios

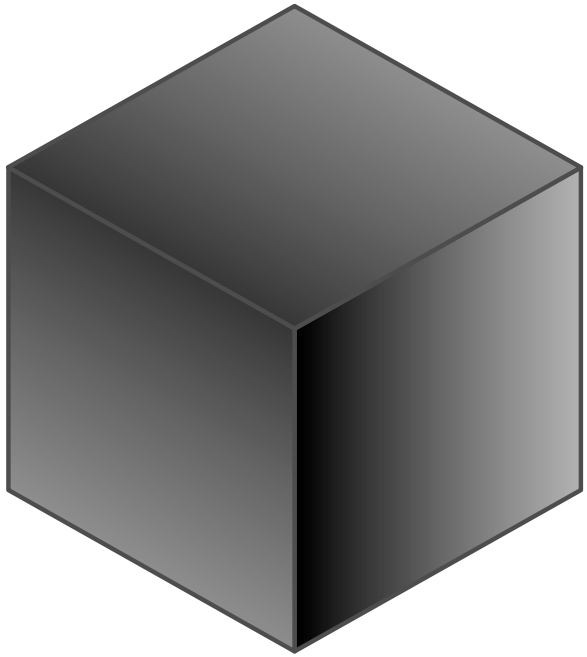


a cryptographic device, but its exact nature is to be determined. There is one special situation that can occur in such an experiment that is worthy of note. The device being experimented on may explode, particularly if it is a bomb, a mine, or some other infernal machine. Since the experimenter is presumably intelligent enough to have anticipated this possibility, he may be assumed to have conducted his experimentation by remote control from a safe distance. However, the bomb or mine is then destroyed, and nothing further can be learned from it by experimentation.

Early scenario suggested by Moore, 1956

Two Families of Techniques

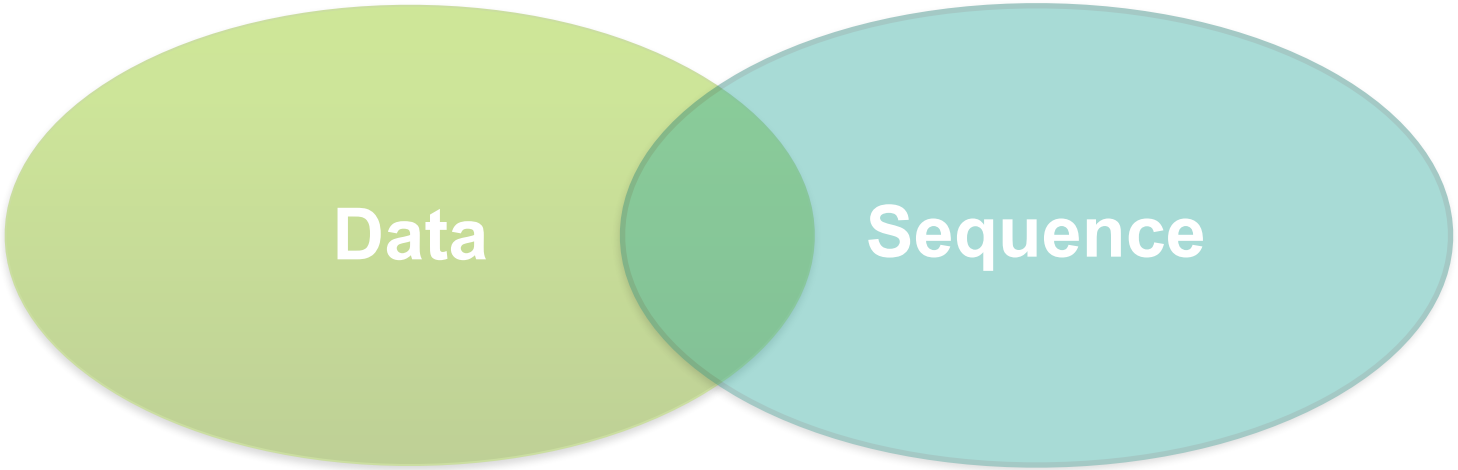
- **Specification-based Testing**
  - We know what behaviour we seek to verify.
  - Have idea of how to trigger it.
- **Random**
  - Very little information available.
  - Some high-level properties to be verified.



Specification-Based Testing

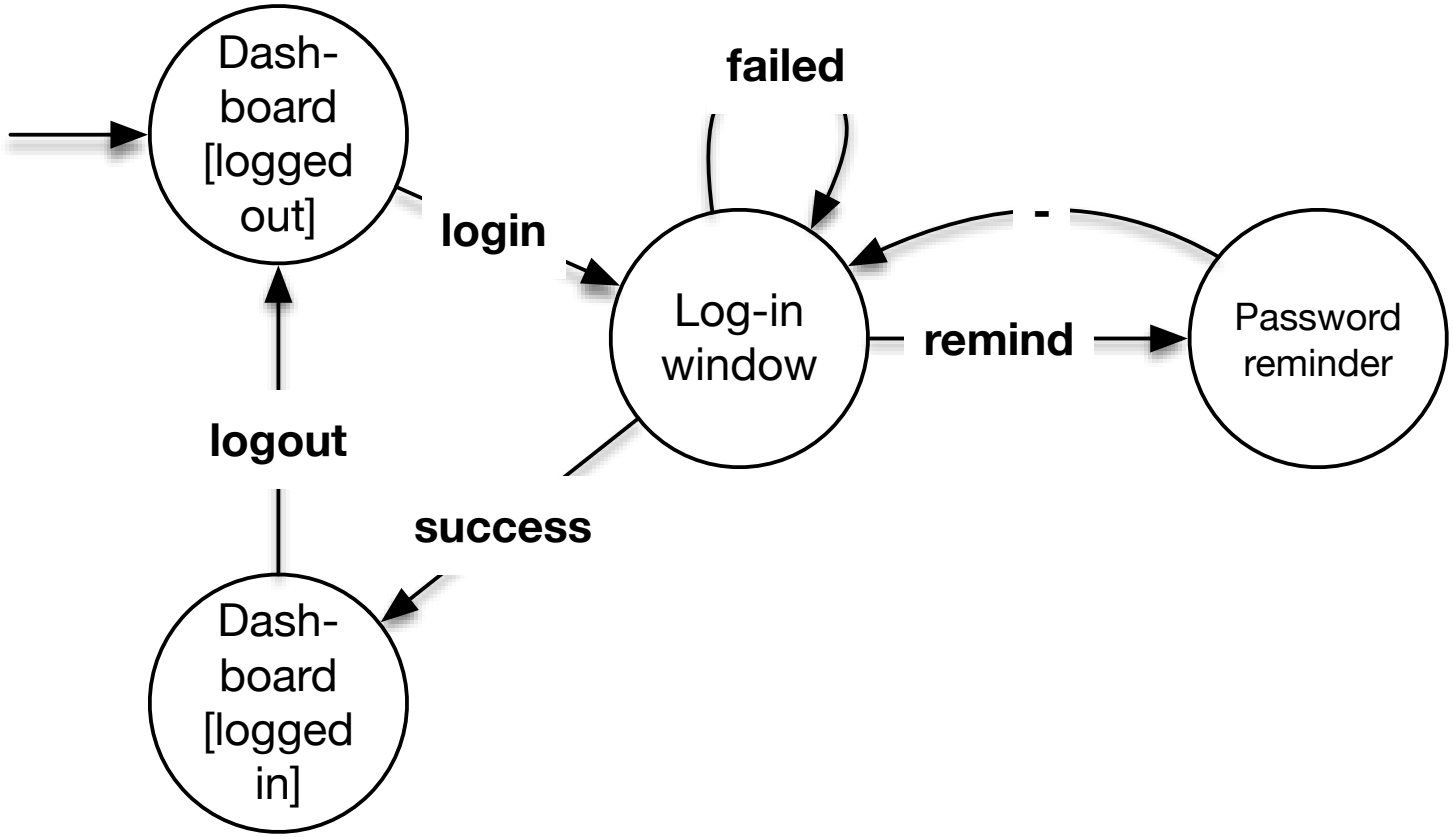
Specifications

- Contain information about intended behaviour.
  - Recall first testing lecture.
- Come in a variety of forms.
  - Z-Specifications, simple assertions, state machines, etc.
- Can be split into two broad types:



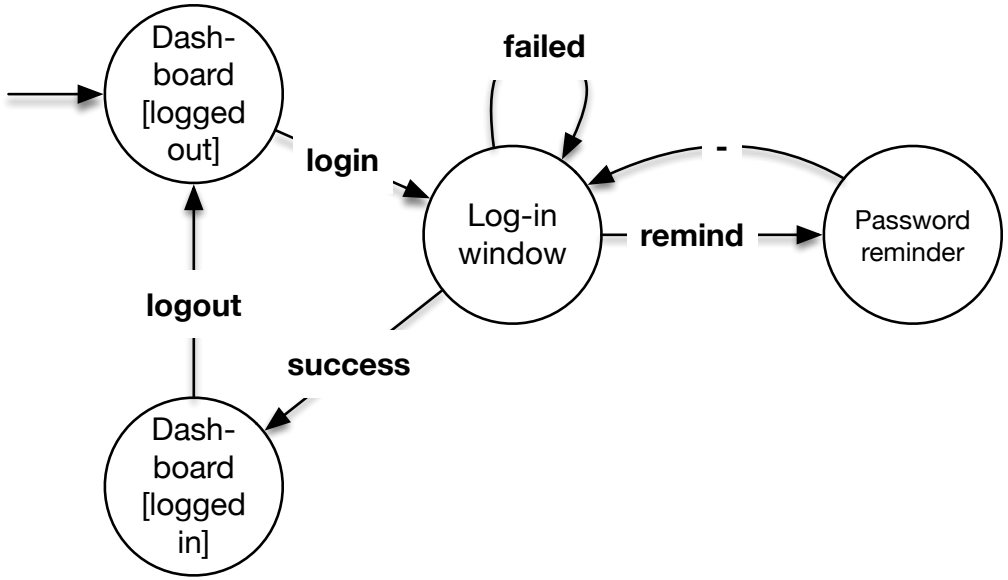
State Machine Testing

Modelling Sequential Behaviour with State Machines



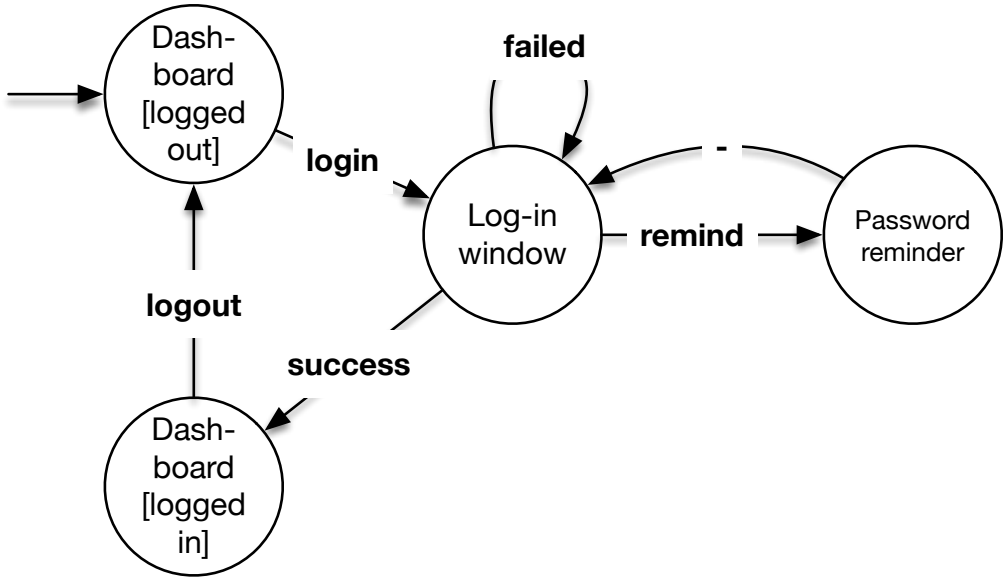
State Coverage

- Set of tests cover each state at least once.



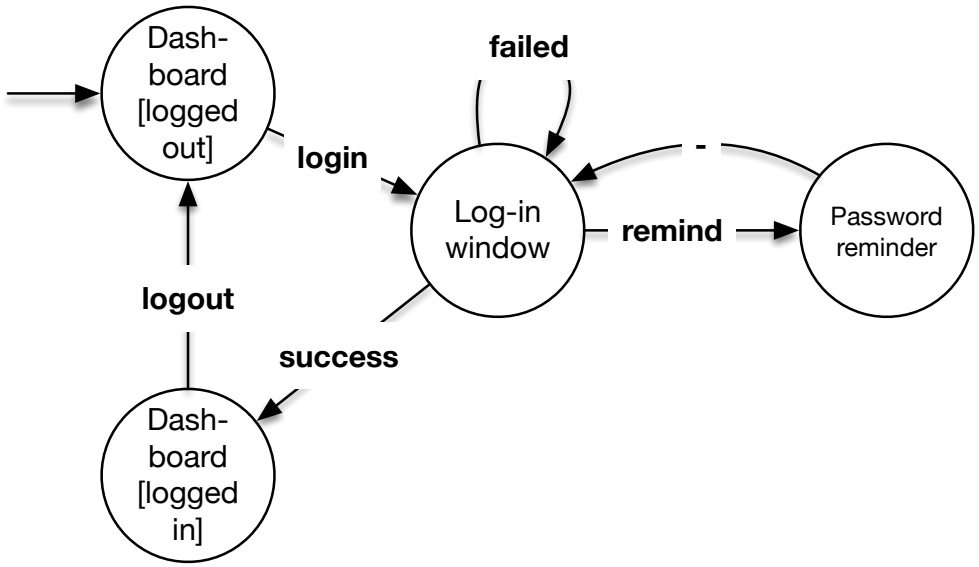
State Coverage

- Set of tests cover each state at least once.
- E.g.
  1. login
  2. remind
  3. -
  4. success



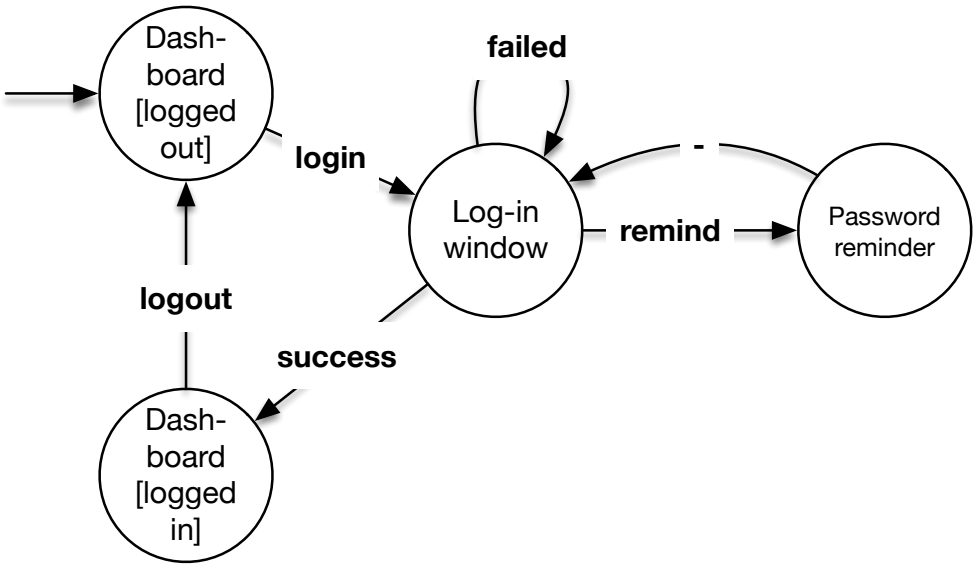
Transition Coverage

- Set of tests cover each transition at least once.



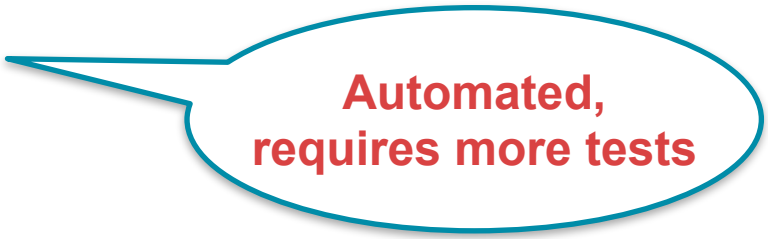
Transition Coverage

- Set of tests cover each transition at least once.
- E.g.
  1. login
  2. remind
  3. -
  4. failed
  5. success
  6. logout



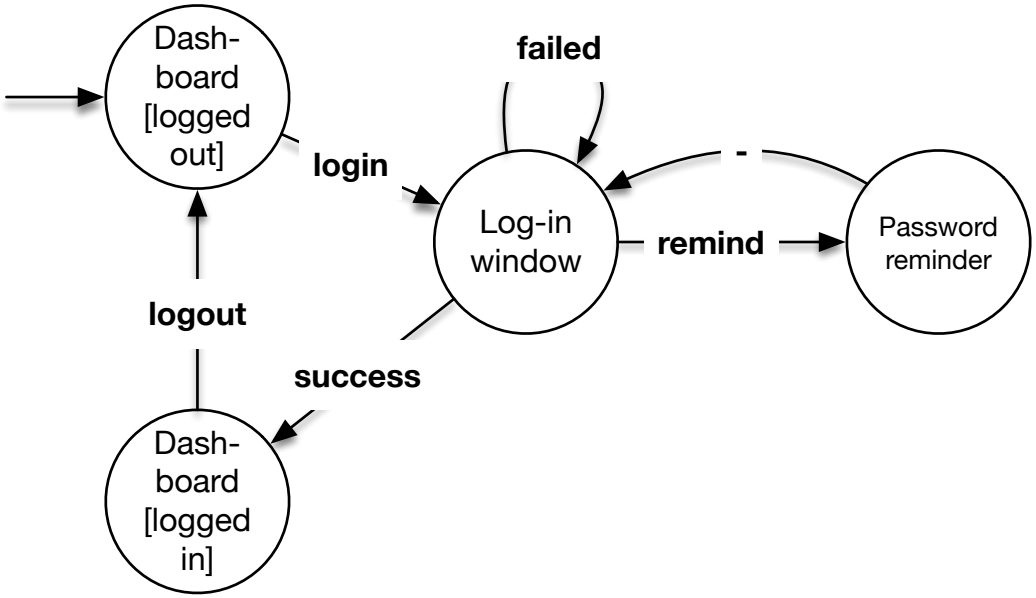
What about the Oracle?

- **Strong:** Can check externally that SUT is in the correct state?
  - E.g. Human, checking each GUI screen on a phone
- **Weak:** Only able to check that sequences are possible.
  - E.g. SUT refuses certain inputs.



Approach for Weak Oracles

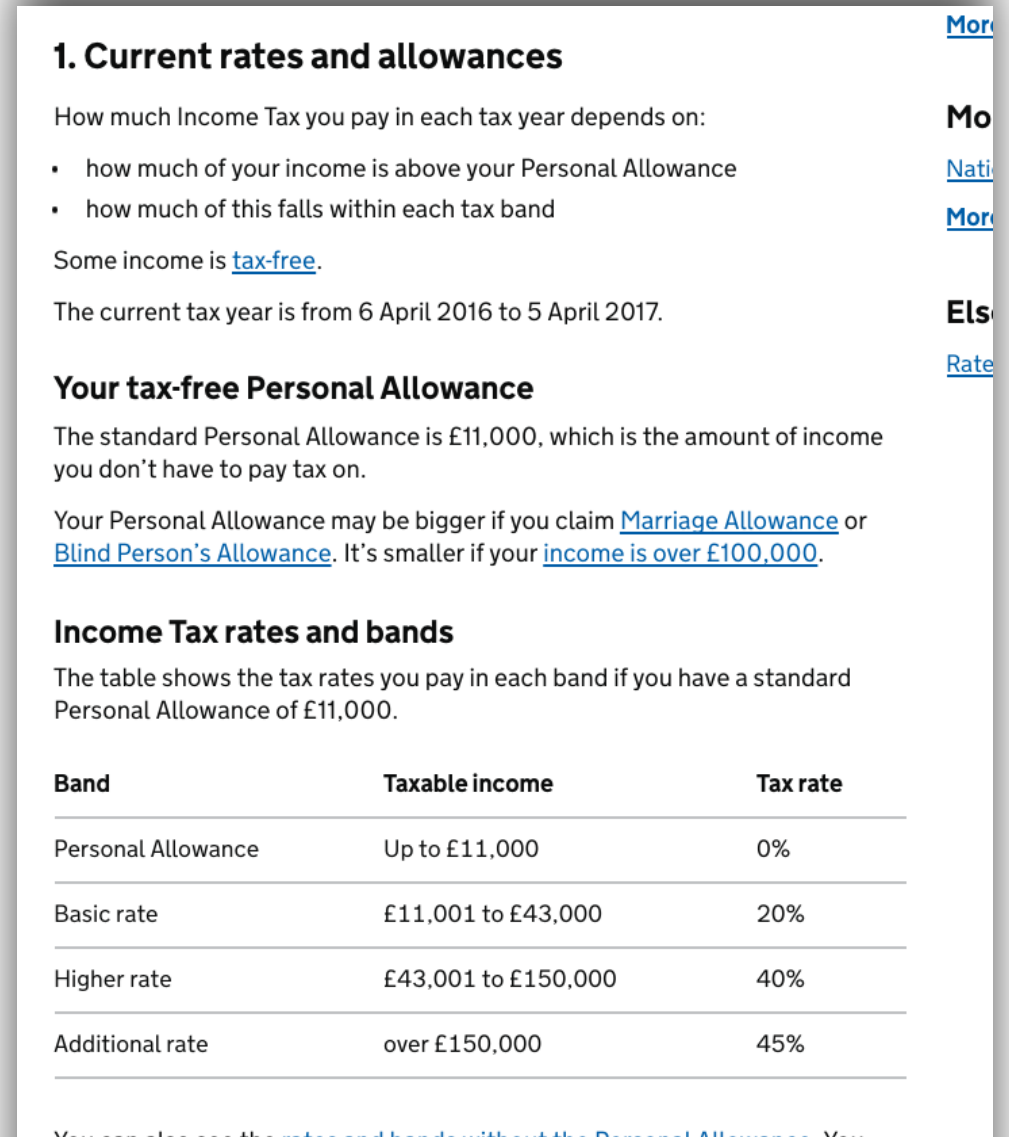
- Devise several test sequences for each state.
- For each state:
  - Find shortest path to state.
  - Attempt every input from that state.
  - Record whether it should be possible or not



Data-Driven Testing

taken from https://www.gov.uk/income-tax-rates/current-rates-and-allowances

Example: Testing a UK Tax Allowance Calculator



1. Current rates and allowances

How much Income Tax you pay in each tax year depends on:

- how much of your income is above your Personal Allowance
- how much of this falls within each tax band

Some income is tax-free.

The current tax year is from 6 April 2016 to 5 April 2017.

Your tax-free Personal Allowance

The standard Personal Allowance is £11,000, which is the amount of income you don't have to pay tax on.

Your Personal Allowance may be bigger if you claim Marriage Allowance or Blind Person's Allowance. It's smaller if your income is over £100,000.

Income Tax rates and bands

The table shows the tax rates you pay in each band if you have a standard Personal Allowance of £11,000.

Band	Taxable income	Tax rate
Personal Allowance	Up to £11,000	0%
Basic rate	£11,001 to £43,000	20%
Higher rate	£43,001 to £150,000	40%
Additional rate	over £150,000	45%

UNIVERSITY OF LEICESTER

taken from h

Examp

1. Current rates and allowances

How much Income Tax you pay in each tax year depends on:

- how much of your income is above your Personal Allowance
- how much of this falls within each tax band

Some income is tax-free.

The current tax year is from 6 April 2016 to 5 April 2017.

Your tax-free Personal Allowance

The standard Personal Allowance is £11,000, which is the amount of income you don't have to pay tax on.

Your Personal Allowance may be bigger if you claim Marriage Allowance or Blind Person's Allowance. It's smaller if your income is over £100,000.

Income Tax rates and bands

The table shows the tax rates you pay in each band if you have a standard Personal Allowance of £11,000.

Band	Taxable income	Tax rate
Personal Allowance	Up to £11,000	0%
Basic rate	£11,001 to £43,000	20%
Higher rate	£43,001 to £150,000	40%
Additional rate	over £150,000	45%

Mor owances

Mo culator

Nati

Mor

Els

Rate

UNIVERSITY OF LEICESTER

taken from

Examp

vances

culator

Additional rate	over £150,000	45%

You can also see the rates and bands without the Personal Allowance. You don't get a Personal Allowance on taxable income over £122,000.

If you're employed or get a pension

Check your Income Tax to see:

- your Personal Allowance and tax code
- how much tax you've paid in the current tax year
- how much you're likely to pay for the rest of the year

Savings and dividends allowances

You have tax-free allowances for both:

- savings interest
- dividends, if you own shares in a company

You pay tax on any interest or dividends over your allowance.

Paying less Income Tax

You may be able to claim Income Tax reliefs if you're eligible for them.

If you're married or in a civil partnership

You may be able to claim Marriage Allowance to reduce your partner's tax if your income is less than the standard Personal Allowance.

If you don't claim Marriage Allowance and you or your partner were born before 6 April 1935, you may be able to claim Married Couple's Allowance.

Testing Challenge

- Many different types of input.
- Lots of different ways in which input choices can affect output (tax allowance)
- An almost infinite number of possible inputs & combinations.

Category Partition Method

- Identify tests by analysing the program interface
- Operate as follows:
    1. Decompose program into “functional units”
    2. Identify inputs / parameters for these units.
    3. For each input
        - 3.1. Identify its limits and characteristics.
        - 3.2. Define “partitions” - value categories
        - 3.3. Identify constraints between categories
        - 3.4. Write test specification

Step 1: Decomposing Behaviour

- Dividing into smaller units is good practice
  - Possible to generate more rigorous test cases.
  - Easier to debug if faults are found.
- Example:
  - Dividing a large Java application into its core modules / packages.
  - In our tax-calculator example, this is not necessary.

Step 2: Identify the inputs and outputs

- For some systems this is straightforward
  - E.g. the Triangle program:
    Input: 3 numbers, ***output:*** 1 String.
  - Our tax calculator:
    Input: Age, income, married, blind, …, ***output:*** number
- For others less so. Consider the following:
  - A phone app.
  - A command such as grep.
  - A web-page with a flash component.

Step 3: Identify the `Categories'

- Significant value ranges / value-characteristics of an input.

Band	Taxable income	Tax rate
Personal Allowance	Up to £11,000	0%
Basic rate	£11,001 to £43,000	20%
Higher rate	£43,001 to £150,000	40%
Additional rate	over £150,000	45%

3. Income over £100,000

Your Personal Allowance goes down by £1 for every £2 that your adjusted net income is above £100,000. This means your allowance is zero if your income is £122,000 or above.

If you don't claim Marriage Allowance and you or your partner were born before 6 April 1935, you may be able to claim Married Couple's Allowance.

Possible Categories

Category	Condition	
A	In Personal Allowance	$pay \leq 11,000$
B	In Basic Rate	$11,000 < pay \leq 43,000$
C	In Higher Rate	$43,000 < pay \leq 150,000$
D	In Additional Rate	$pay > 150,000$
E	Above 100k, sub additional rate	$100,000 < pay < 150,000$
F	Blind	$blind = true$
G	Not blind	$blind = false$
H	Married	$married = true$
I	Not Married	$Married = false$

Generating Tests by Combining Categories

Category	Condition	
A	In Personal Allowance	$pay \leq 11,000$
B	In Basic Rate	$11,000 < pay \leq 43,000$
C	In Higher Rate	$43,000 < pay \leq 150,000$
D	In Additional Rate	$pay > 150,000$
E	Above 100k, sub additional rate	$100,000 < pay < 150,000$
F	Blind	$blind = true$
G	Not blind	$blind = false$
H	Married	$married = true$
I	Not Married	$Married = false$

Generating Tests by Combining Categories

Category	Condition	
A	In Personal Allowance	$pay \leq 11,000$
B	In Basic Rate	$11,000 < pay \leq 43,000$
C	In Higher Rate	$43,000 < pay \leq 150,000$
D	In Additional Rate	$pay > 150,000$
E	Above 100k, sub additional rate	$100,000 < pay < 150,000$
F	Blind	$blind = true$
G	Not blind	$blind = false$
H	Married	$married = true$
I	Not Married	$Married = false$

Several categories cannot be combined with others.

Generating Tests by Combining Categories

Category	Condition	
A	In Personal Allowance	$pay \leq 11,000$
B	In Basic Rate	$11,000 < pay \leq 43,000$
C	In Higher Rate	$43,000 < pay \leq 150,000$
D	In Additional Rate	$pay > 150,000$
E	Above 100k, sub additional rate	$100,000 < pay < 150,000$
F	Blind	$blind = true$
G	Not blind	$blind = false$
H	Married	$married = true$
I	Not Married	$Married = false$

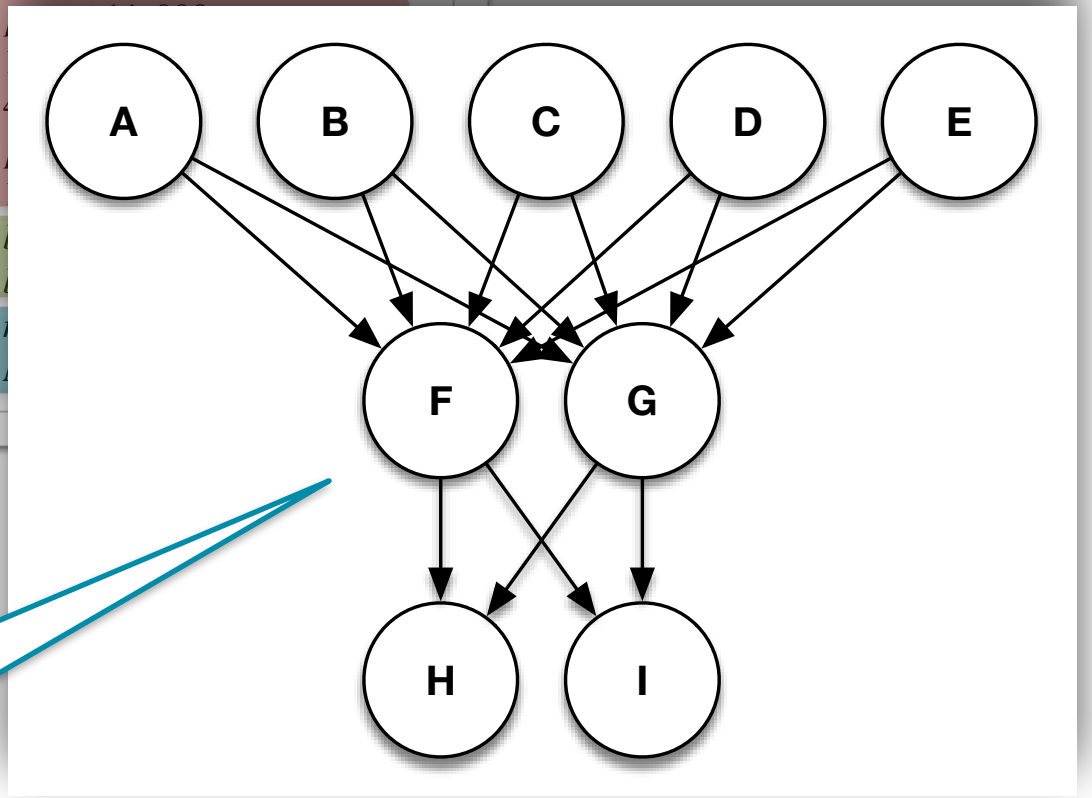
Several categories cannot be combined with others.

Every Possible Combination

Category	Condition	
A	In Personal Allowance	$pay \leq 11,000$
B	In Basic Rate	$11,000 < pay \leq 43,000$
C	In Higher Rate	$43,000 < pay \leq 150,000$
D	In Additional Rate	$pay > 150,000$
E	Above 100k, sub additional rate	$100,000 < pay < 150,000$
F	Blind	$blind = true$
G	Not blind	$blind = false$
H	Married	$married = true$
I	Not Married	$Married = false$

Every Possible Combination

Category	Condition
A	In Personal Allowance
B	In Basic Rate
C	In Higher Rate
D	In Additional Rate
E	Above 100k, sub additional rate
F	Blind
G	Not blind
H	Married
I	Not Married



Valid combinations: Every path through this network.

Command:
find

Syntax:
find <pattern> <file>

Function:
The find command is used to locate one or more instances of a given pattern in a text file. All lines in the file that contain the pattern are written to standard output. A line containing the pattern is written only once, regardless of the number of times the pattern occurs in it.

The pattern is any sequence of characters whose length does not exceed the maximum length of a line in the file. To include a blank in the pattern, the entire pattern must be enclosed in quotes ("). To include a quotation mark in the pattern, two quotes in a row ("") must be used.

Examples:

```
find john myfile
```
displays lines in the file **myfile** which contain **john**

```
find "john smith" myfile
```
displays lines in the file **myfile** which contain **john smith**

```
find "john"" smith" myfile
```
displays lines in the file **myfile** which contain **john" smith**

```
Parameters:
    Pattern size:
        empty
        single character
        many character
        longer than any line in the file

    Quoting:
        pattern is quoted
        pattern is not quoted
        pattern is improperly quoted

    Embedded blanks:
        no embedded blank
        one embedded blank
        several embedded blanks

    Embedded quotes:
        no embedded quotes
        one embedded quote
        several embedded quotes

    File name:
        good file name
        no file with this name
        omitted


Environments:
    Number of occurrences of pattern in file:
        none
        exactly one
        more than one

    Pattern occurrences on target line:
    #  assumes line contains the pattern
        one
        more than one
```

```
Parameters:
    Pattern size:
        empty
        single character
        many character
        longer than any line in the file

    Quoting:
        pattern is quoted
        pattern is not quoted
        pattern is improperly quoted

    Embedded blanks:
        no embedded blank
        one embedded blank
        several embedded blanks

    Embedded quotes:
        no embedded quotes
        one embedded quote
        several embedded quotes

    File name:
        good file name
        no file with this name
        omitted


Environments:
    Number of occurrences of pattern in file:
        none
        exactly one
        more than one

    Pattern occurrences on target line:
    #  assumes line contains the pattern
        one
        more than one
```

1944 possible combinations.
I.e. 1944 possible tests.

```
Parameters:
    Pattern size:
        empty
        single character
        many character
        longer than any line in the file

    Quoting:
        pattern is quoted
        pattern is not quoted
        pattern is improperly quoted

    Embedded blanks:
        no embedded blank
        one embedded blank
        several embedded blanks

    Embedded quotes:
        no embedded quotes
        one embedded quote
        several embedded quotes

    File name:
        good file name
        no file with this name
        omitted


Environments:
    Number of occurrences of pattern in file:
        none
        exactly one
        more than one

    Pattern occurrences on target line:
    #   assumes line contains the pattern
        one
        more than one
```

1944 possible combinations.
I.e. 1944 possible tests.

Several contradictory requirements

```
Pattern size : empty
Quoting : pattern is quoted
Embedded blanks : several embedded blanks
Embedded quotes : no embedded quotes
File name : good file name
Number of occurrences of pattern in file : none
Pattern occurrences on target line : one
```

Add Constraints

```
# Test Specification for find command
# Modified:  property lists and selector expressions added

Parameters:
    Pattern size:
        empty                                [property Empty]
        single character                     [property NonEmpty]
        many character                       [property NonEmpty]
        longer than any line in the file     [property NonEmpty]

    Quoting:
        pattern is quoted                    [property Quoted]
        pattern is not quoted                [if NonEmpty]
        pattern is improperly quoted         [if NonEmpty]

    Embedded blanks:
        no embedded blank                    [if NonEmpty]
        one embedded blank                   [if NonEmpty and Quoted]
        several embedded blanks              [if NonEmpty and Quoted]

    Embedded quotes:
        no embedded quotes                   [if NonEmpty]
        one embedded quote                   [if NonEmpty]
        several embedded quotes              [if NonEmpty]

    File name:
        good file name
        no file with this name
        omitted


Environments:
    Number of occurrences of pattern in file:
        none                                 [if NonEmpty]
        exactly one                          [if NonEmpty] [property Match]
        more than one                        [if NonEmpty] [property Match]

    Pattern occurrences on target line:
        #  assumes line contains the pattern
        one                                  [if Match]
        more than one                        [if Match]
```

Add Constraints

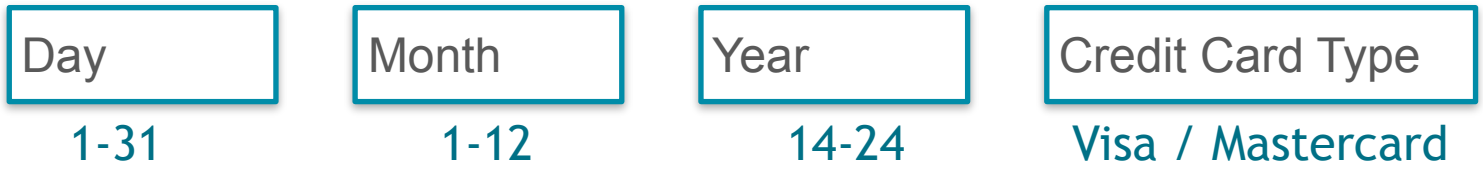
```
# Test Specification for find command
# Modified:  property lists and selector expressions added

Parameters:
    Pattern size:
        empty                                [property Empty]
        single character                     [property NonEmpty]
        many character                       [property NonEmpty]
        longer than any line in the file     [property NonEmpty]

    Quoting:
        pattern is quoted                    [property Quoted]
        pattern is not quoted                [if NonEmpty]
        pattern is improperly quoted         [if NonEmpty]

    Embedded blanks:
        no embedded blank                    [if NonEmpty]
        one embedded blank                   [if NonEmpty and Quoted]
        several embedded blanks              [if NonEmpty and Quoted]

    Embedded quotes:
        no embedded quotes                   [if NonEmpty]
        one embedded quote                   [if NonEmpty]
        several embedded quotes              [if NonEmpty]

    File name:
        good file name
        no file with this name
        omitted


Environments:
    Number of occurrences of pattern in file:
        none                                 [if NonEmpty]
        exactly one                          [if NonEmpty] [property Match]
        more than one                        [if NonEmpty] [property Match]

    Pattern occurrences on target line:
        #  assumes line contains the pattern
        one                                  [if Match]
        more than one                        [if Match]
```

678 possible test cases.

Pairwise (or All Pairs) testing

Category Partition can still produce lots of test cases.

Consider this GUI:

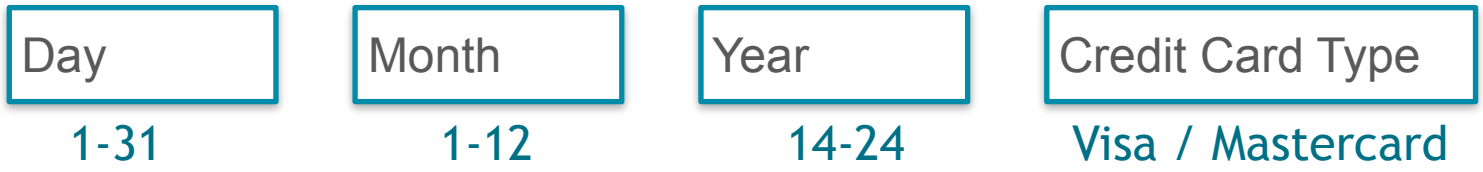


How many possible combinations are there?

Pairwise (or All Pairs) testing

Category Partition can still produce lots of test cases.

Consider this GUI:



How many possible combinations are there?

31 * 12 * 10 * 2 = 7440

Pairwise Testing

Suggests that most of these 7440 tests are **irrelevant**.

Usually a fault does not occur because of a *specific combination* of all inputs.

Faults tend to occur because of one single input, or an *interaction between two inputs*.

Procedure

1. Partition the input space into categories
2. Create a table where variable names are the column headers
3. Repeat each possible value in the first column for the number of possible values in the second column.
4. Fill in the other values, ensuring that every pair appears at least once.

Example

Parameters	Category	Value
day	Max+1	32
	Max	31
	Min	1
	February	28
	Leap February	29
	Zero	0
month	Max+1	13
	Max	12
	Min	1
	Zero	0
year	Max	13
	Min	03
type	visa	visa
	mastercard	mastercard

96 possible combinations of test cases

Day
Max + 1
Max + 1
Max + 1
Max + 1
Max
Max
Max
Max
Min
Min
Min
Min
February
February
February
February
Leap February
Leap February
Leap February
Leap February
Zero
Zero
Zero
Zero

Day	Month
Max + 1	Max + 1
Max + 1	Max
Max + 1	Min
Max + 1	Zero
Max	Max + 1
Max	Max
Max	Min
Max	Zero
Min	Max + 1
Min	Max
Min	Min
Min	Zero
February	Max + 1
February	Max
February	Min
February	Zero
Leap February	Max + 1
Leap February	Max
Leap February	Min
Leap February	Zero
Zero	Max + 1
Zero	Max
Zero	Min
Zero	Zero

Day	Month	Year
Max + 1	Max + 1	Max
Max + 1	Max	Min
Max + 1	Min	Max
Max + 1	Zero	Min
Max	Max + 1	Max
Max	Max	Min
Max	Min	Max
Max	Zero	Min
Min	Max + 1	Max
Min	Max	Min
Min	Min	Max
Min	Zero	Min
February	Max + 1	Max
February	Max	Min
February	Min	Max
February	Zero	Min
Leap February	Max + 1	Max
Leap February	Max	Min
Leap February	Min	Max
Leap February	Zero	Min
Zero	Max + 1	Min
Zero	Max	Min
Zero	Min	Max
Zero	Zero	Min

Day	Month	Year	Type
Max + 1	Max + 1	Max	Visa
Max + 1	Max	Min	MasterCard
Max + 1	Min	Max	Visa
Max + 1	Zero	Min	MasterCard
Max	Max + 1	Max	Visa
Max	Max	Min	MasterCard
Max	Min	Max	Visa
Max	Zero	Min	MasterCard
Min	Max + 1	Max	Visa
Min	Max	Min	MasterCard
Min	Min	Max	Visa
Min	Zero	Min	MasterCard
February	Max + 1	Max	Visa
February	Max	Min	MasterCard
February	Min	Max	Visa
February	Zero	Min	MasterCard
Leap February	Max + 1	Max	Visa
Leap February	Max	Min	MasterCard
Leap February	Min	Max	Visa
Leap February	Zero	Min	MasterCard
Zero	Max + 1	Min	Visa
Zero	Max	Min	MasterCard
Zero	Min	Max	MasterCard
Zero	Zero	Min	Visa

Why is it Useful?

Category Partition Method

- Reduces 7440 test cases to 96.

Pairwise Testing

- Reduces 96 to 24

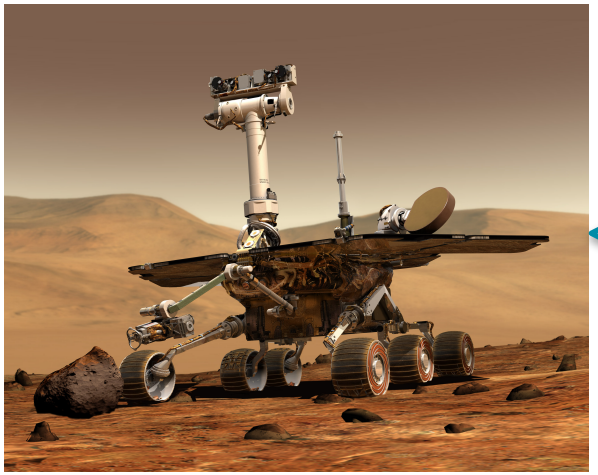
Extensively used in the industry

- E.g. testing webpages
- Can use the Microsoft PICT tool to generate Pairwise tests.

Random Testing

Scenario:

- No details / models showing how to execute functions.
- Very limited oracle, e.g:
  - Set of obvious rules specified by a developer.
  - No rules at all, limited to observing obvious problems such as crashes / exceptions.
- Popular (because of its few requirements)



Used by NASA on its Mars rovers.

Useful Properties of “Randomness”

- Good at attempting unexpected inputs.
- *Every* input that reveals a bug has at least a small probability of occurring.
  - **IF** you could run an infinite number of tests…
    … you would expose *every* bug.
- Can use **probability theory** to estimate **test adequacy**.

Useful Properties of "Randomness"

- Good at attempting unexpected inputs.
- *Every* input that reveals a bug has at least a small probability of occurring.
  - **IF** you could run an infinite number of tests…
    … you would expose *every* bug.
- Can use **probability theory** to estimate **test adequacy**.

Infinite Monkey theorem: A monkey, given infinite time, will eventually type out Shakespeare's Hamlet.

Picking Random Values

Demo:

Generate Random Inputs for the BMI program (white-box testing example)

Constraining the Input Space

- Crucial:
  - Want to ensure that most inputs hit interesting behaviour.

Constraining the Input Space

- Crucial:
  - Want to ensure that most inputs hit interesting behaviour.

Last Updated: Friday, 9 May, 2003, 12:28 GMT 13:28 UK

Email this to a friend Printable version

No words to describe monkeys' play

A bizarre experiment by a group of students has found monkeys cannot write Shakespeare.

Six monkeys took part in the experiment at Paignton Zoo

Lecturers and students from the University of Plymouth wanted to test the claim that an infinite number of monkeys given typewriters would create the works of The Bard.

A single computer was placed in a monkey enclosure at Paignton Zoo to monitor the literary output of six primates.

But after a month, the Sulawesi crested macaques had only succeeded in partially destroying the machine, using it as a lavatory, and mostly typing the letter "s".

Constraining the Input Space

- Crucial:
  - Want to ensure that most inputs hit interesting behaviour.

Last Updated: Friday, 9 May, 2003, 12:28 GMT 13:28 UK

Email this to a friend Printable version

No words to describe monkeys' play

A bizarre experiment by a group of students has found monkeys cannot write Shakespeare.

Six monkeys took part in the experiment at Paignton Zoo

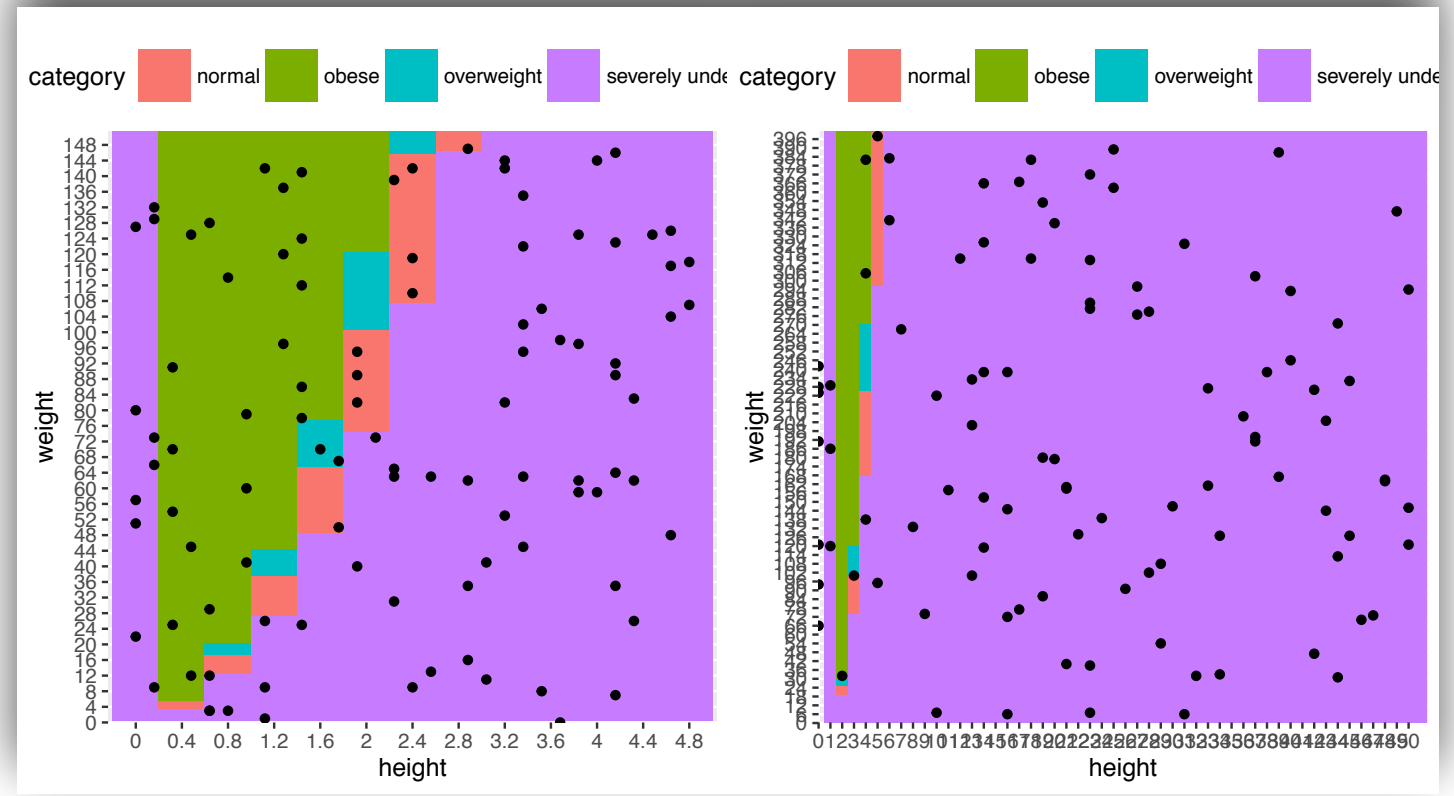
Lecturers and students from the University of Plymouth wanted to test the claim that an infinite number of monkeys given typewriters would create the works of The Bard.

A single computer was placed in a monkey enclosure at Paignton Zoo to monitor the literary output of six primates.

But after a month, the Sulawesi crested macaques had only succeeded in partially destroying the machine, using it as a lavatory, and mostly typing the letter "s".

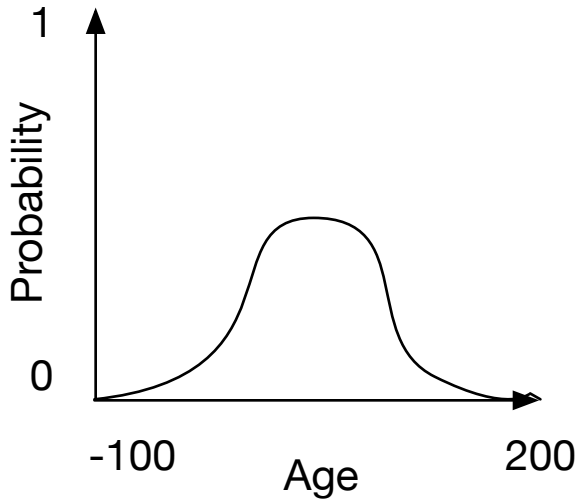
Constraining the Input Space

- Crucial:
  - Want to ensure that most inputs hit interesting behaviour.



Typical Constraining Techniques

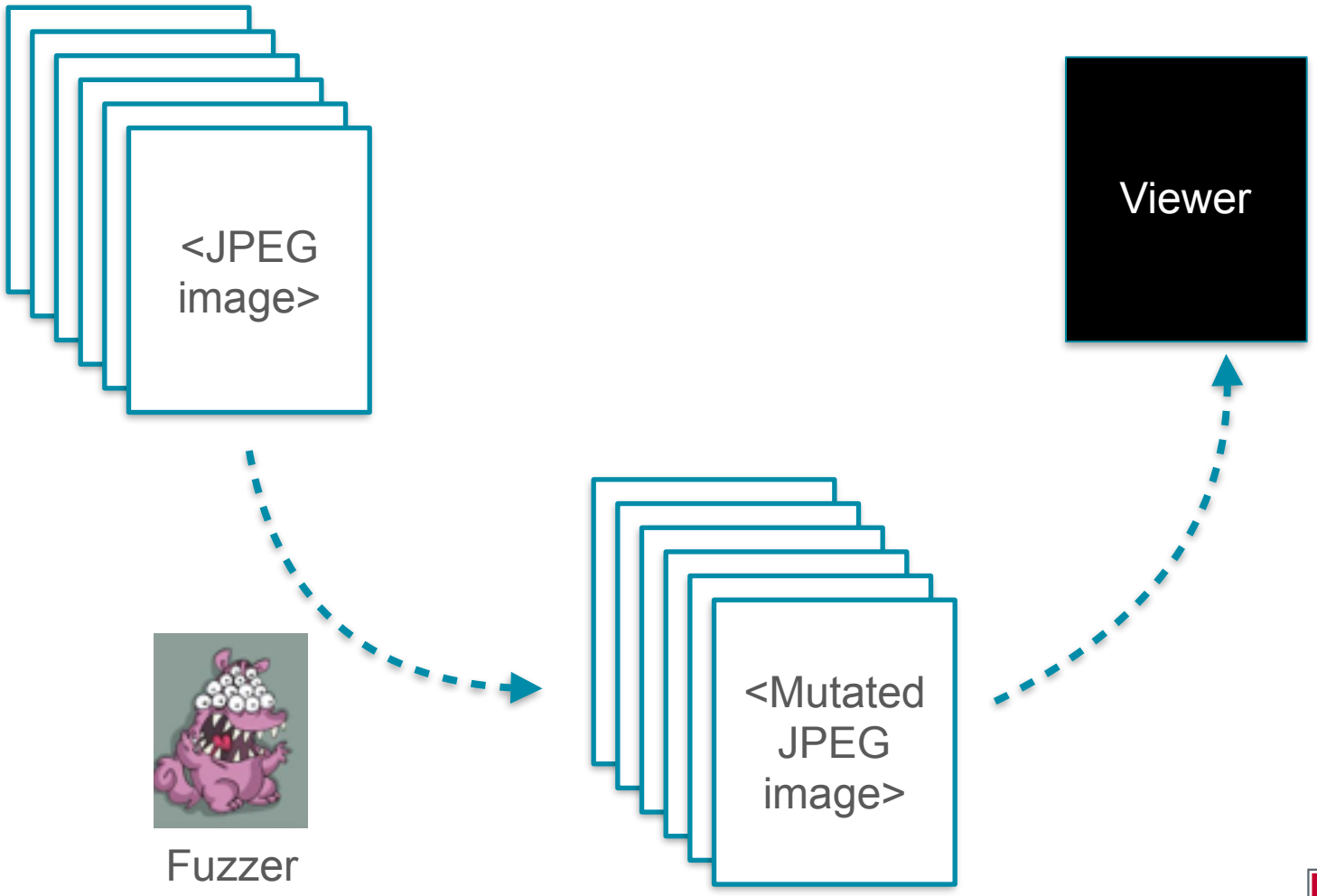
- For numerical inputs, constrain range (min, max).
- Add probability distributions
- Add “input generators” for complex inputs.
  - A random date generator.
  - Random webpage generators to test web browsers.
  - Random source code to test parsers & compilers…



Finding Good Test Inputs

- (Relatively) easy for numerical parameters.
- Very difficult for more complex input structures.
  - What if the program you're trying to test is a PDF viewer?
  - Or a web browser?
  - Or Microsoft Word?
  - Or the controller for a self-driving car?

Fuzz Testing



Using Probability to Quantify Reliability

- How *reliable* is a software system?

Using Probability to Quantify Reliability

- How *reliable* is a software system?

θ — Probability that the SUT will fail.

Using Probability to Quantify Reliability

- How *reliable* is a software system?

θ — Probability that the SUT will fail.

$1 - \theta$ — Probability that the SUT will pass.

Using Probability to Quantify Reliability

- How *reliable* is a software system?

θ — Probability that the SUT will fail.

$1 - \theta$ — Probability that the SUT will pass.

$(1 - \theta)^N$ — Probability of no failure in N tests.

Using Probability to Quantify Reliability

- How *reliable* is a software system?

θ — Probability that the SUT will fail.

$1 - \theta$ — Probability that the SUT will pass.

$(1 - \theta)^N$ — Probability of no failure in N tests.

$1 - (1 - \theta)^N$ — Probability that at least one test will fail in N tests.

Example

Example

- How *reliable* is (Neil's copy of) Panopto for uploading?
  - Has failed to upload 3/18 times:

Example

- How *reliable* is (Neil's copy of) Panopto for uploading?
  - Has failed to upload 3/18 times:

$$\theta = \frac{3}{18} = 0.167$$

Example

- How *reliable* is (Neil's copy of) Panopto for uploading?
  - Has failed to upload 3/18 times:

$$\theta = \frac{3}{18} = 0.167$$

  - Probability that upload will succeed:

$$(1 - \theta) = 1 - 0.167 = 0.8333$$

Example

- How *reliable* is (Neil's copy of) Panopto for uploading?
  - Has failed to upload 3/18 times:

$$\theta = \frac{3}{18} = 0.167$$

  - Probability that upload will succeed:

$$(1 - \theta) = 1 - 0.167 = 0.8333$$

Example

- How *reliable* is (Neil's copy of) Panopto for uploading?
  - Has failed to upload 3/18 times:

    $$\theta = \frac{3}{18} = 0.167$$
  - Probability that upload will succeed:

    $$(1 - \theta) = 1 - 0.167 = 0.8333$$
  - Probability of no failed uploads for CO3095/CO7095 next term? (24 recordings)

    $$(1 - \theta)^N = (1 - 0.167)^{24} = 0.8333^{24} = 0.013$$

Example

- How *reliable* is (Neil's copy of) Panopto for uploading?
  - Has failed to upload 3/18 times:

    $$\theta = \frac{3}{18} = 0.167$$
  - Probability that upload will succeed:

    $$(1 - \theta) = 1 - 0.167 = 0.8333$$
  - Probability of no failed uploads for CO3095/CO7095 next term? (24 recordings)

    $$(1 - \theta)^N = (1 - 0.167)^{24} = 0.8333^{24} = 0.013$$

Example

- How *reliable* is (Neil's copy of) Panopto for uploading?
  - Has failed to upload 3/18 times:

    $$\theta = \frac{3}{18} = 0.167$$
  - Probability that upload will succeed:

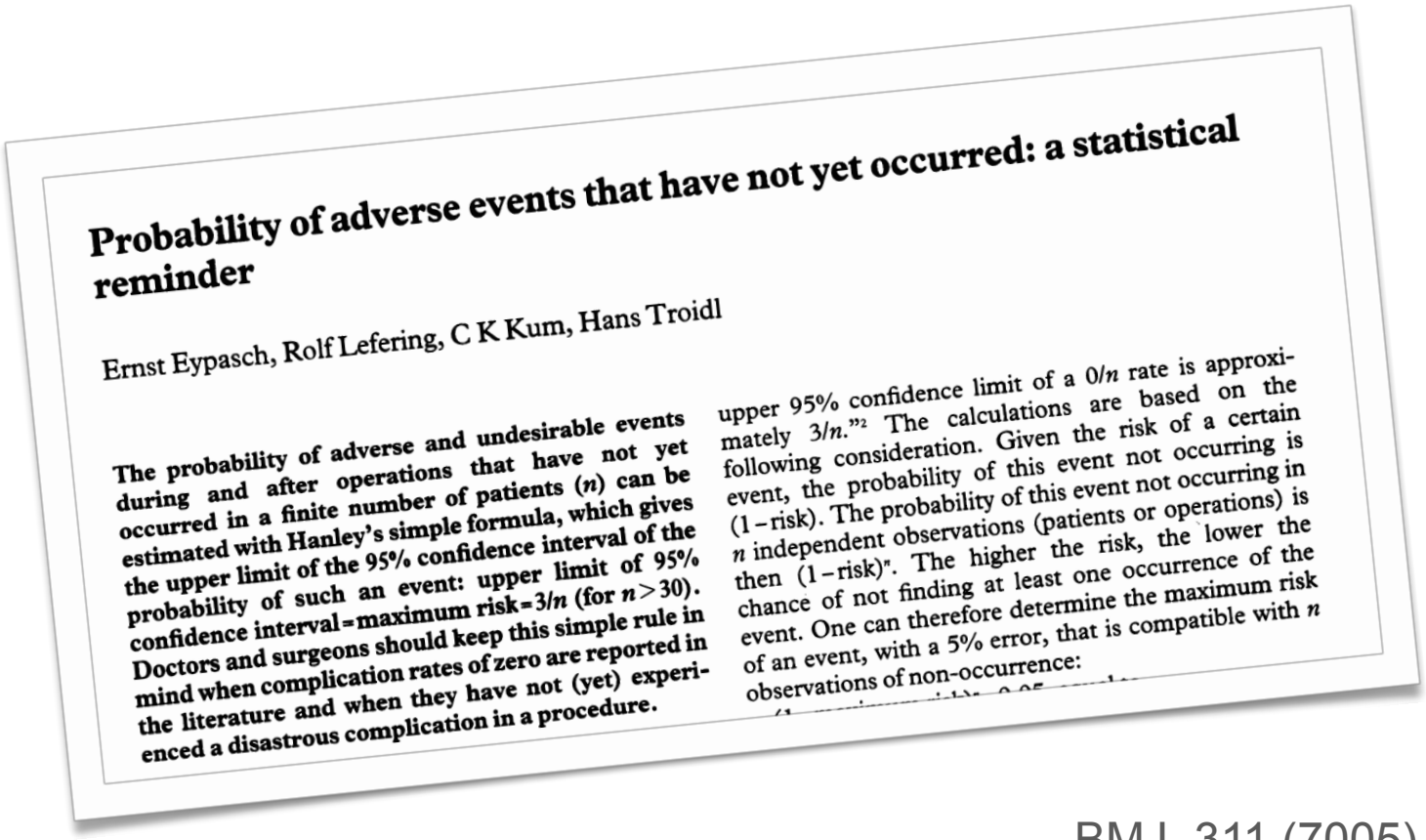
    $$(1 - \theta) = 1 - 0.167 = 0.8333$$
  - Probability of no failed uploads for CO3095/CO7095 next term? (24 recordings)

    $$(1 - \theta)^N = (1 - 0.167)^{24} = 0.8333^{24} = 0.013$$
  - Probability that at least one recording will fail to upload?

    $$1 - ((1 - \theta)^N) = 1 - ((1 - 0.167)^{24}) = 1 - 0.013 = 0.987$$

What if no failures have been observed?

- No basis for estimating failure rate (θ)



Probability of adverse events that have not yet occurred: a statistical reminder

Ernst Eypasch, Rolf Lefering, C K Kum, Hans Troidl

The probability of adverse and undesirable events during and after operations that have not yet occurred in a finite number of patients (n) can be estimated with Hanley's simple formula, which gives the upper limit of the 95% confidence interval of the probability of such an event: upper limit of 95% confidence interval = maximum risk = $3/n$ (for $n > 30$). Doctors and surgeons should keep this simple rule in mind when complication rates of zero are reported in the literature and when they have not (yet) experienced a disastrous complication in a procedure.

upper 95% confidence limit of a $0/n$ rate is approximately $3/n$."[2] The calculations are based on the following consideration. Given the risk of a certain event, the probability of this event not occurring is (1 – risk). The probability of this event not occurring in n independent observations (patients or operations) is then $(1 - \text{risk})^n$. The higher the risk, the lower the chance of not finding at least one occurrence of the event. One can therefore determine the maximum risk of an event, with a 5% error, that is compatible with n observations of non-occurrence:

BMJ. 311 (7005), 1995

Rule of Three

- Probability of an adverse event can be estimated by $\frac{3}{N}$
  - Where N is number of events (tests)
  - N>30
  - Probability is with a 95% confidence interval.

Example

- This year Neil's smug colleague successfully all 24 of their lectures using Panopto.
- What is the probability that at least one failure will occur next semester?

Example

- This year Neil's smug colleague successfully all 24 of their lectures using Panopto.
- What is the probability that at least one failure will occur next semester?

$$\frac{3}{24} = 0.125$$

Summary

- Black box testing techniques can be broadly split into specification-based testing and random testing.
- Specification-based testing techniques tend to focus either on sequences or data.
- State machine testing techniques focus on sequencing.
- Techniques such as Category Partitioning focus on data.
- Random testing is good at eliciting unexpected faults.
- Requires careful calibration.
- Reliability can be estimated by probabilistic rules.
- Rule of Three applies when prior failures have not been observed.