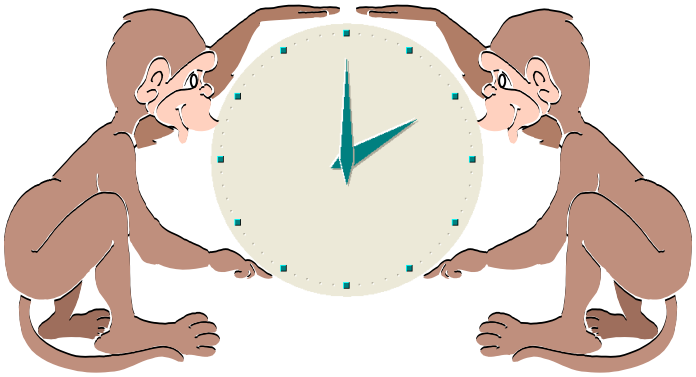
A Quick Intro to Model-Based Testing



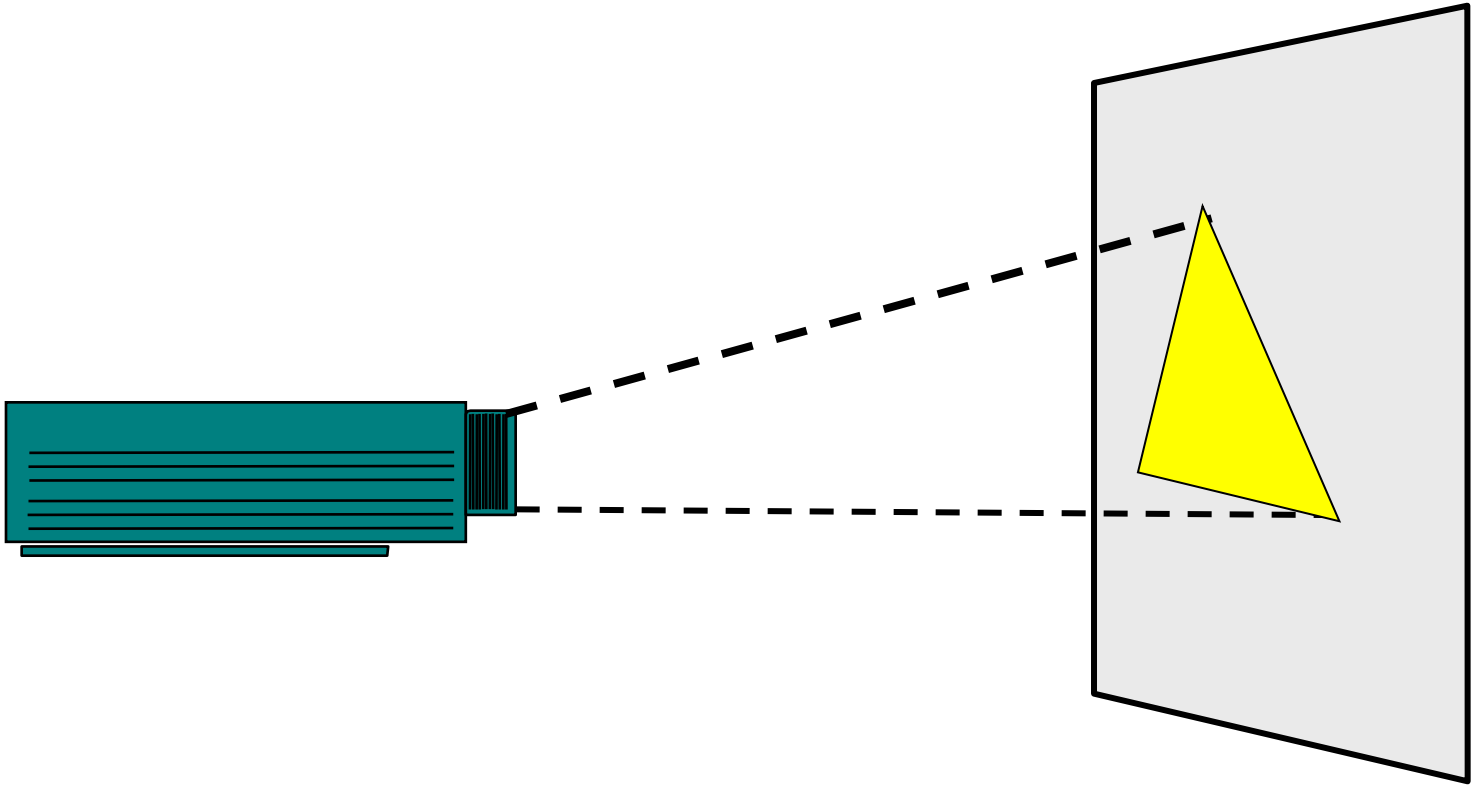
45,000 tests in 45 minutes or less!

The Villain of this Piece

```
CurrentWindow = WFndWndC("Calculator", "SciCalc")
WSetWndPosSiz(CurrentWindow, 88, 116, 260, 260)
WMenuSelect("@2\@2")
CurrentWindow = WFndWndC("Calculator", "SciCalc")
WSetWndPosSiz(CurrentWindow, 88, 116, 480, 317)
WButtonClick("@32")
WButtonClick("@27")
WButtonClick("@38")
WButtonClick("@58")
WMenuSelect("@2\@1")
CurrentWindow = WFndWndC("Calculator", "SciCalc")
WSetWndPosSiz(CurrentWindow, 88, 116, 260, 260)
WMenuSelect("@2\@1")
Play "{Click 330, 131, Left}"
WToolbarButtonClk("@2", "@6")
WToolbarButtonClk("@2", "@7")
WToolbarButtonClk("@1", "@1")
```

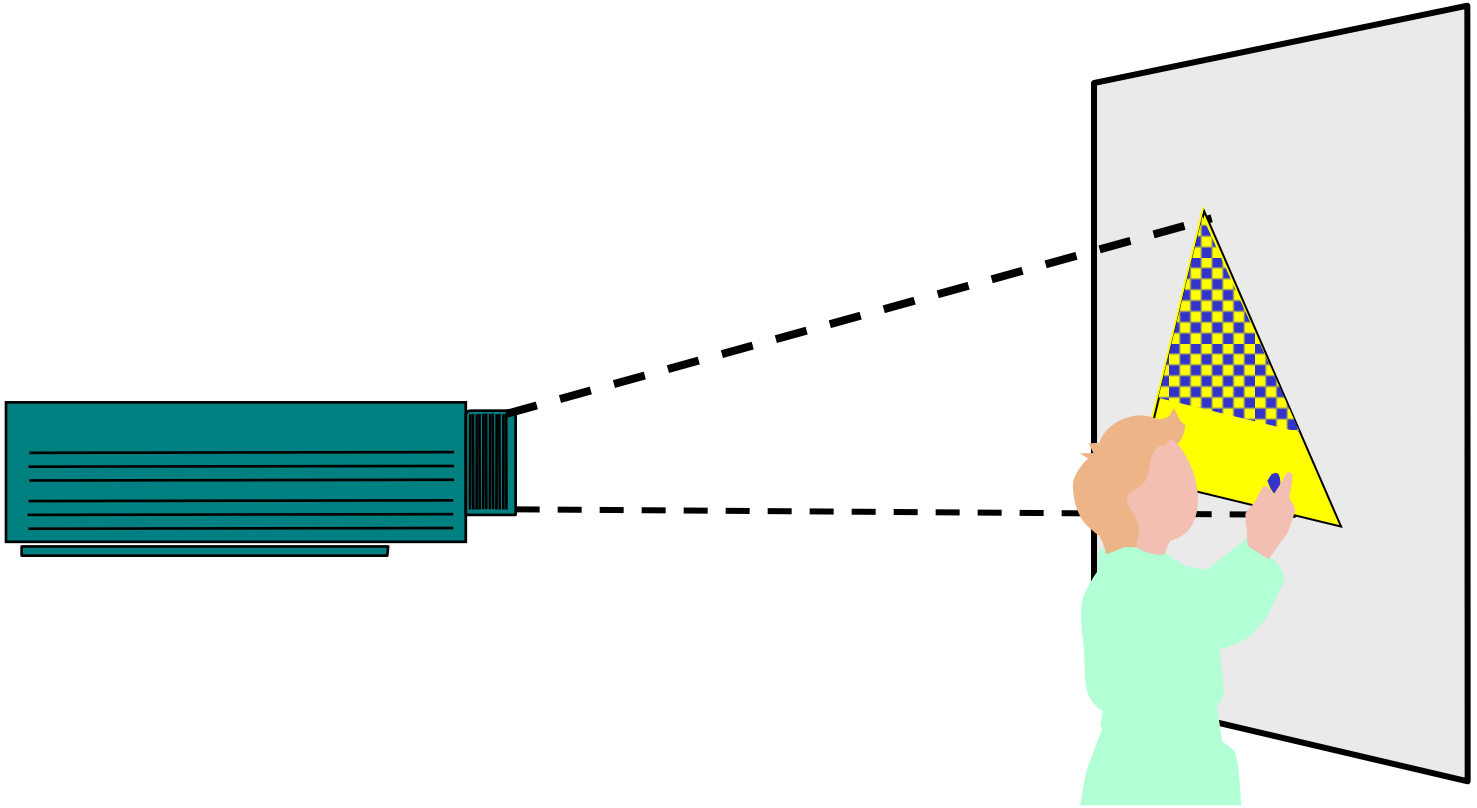
- Awe-inspiring
- Unchanging
- Indecipherable

Traditional Automated Testing



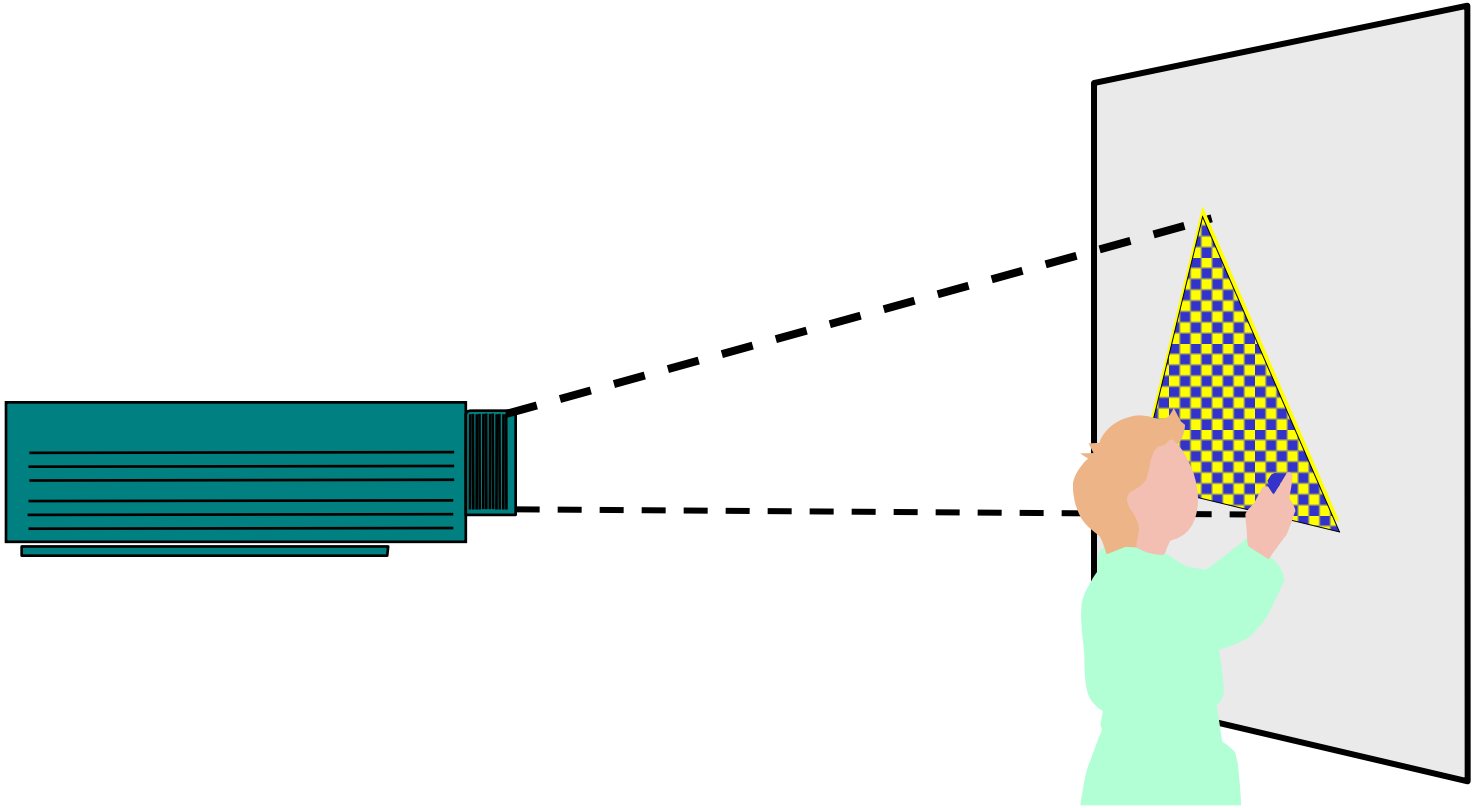
Imagine that this projector is the software you are testing.

Traditional Automated Testing



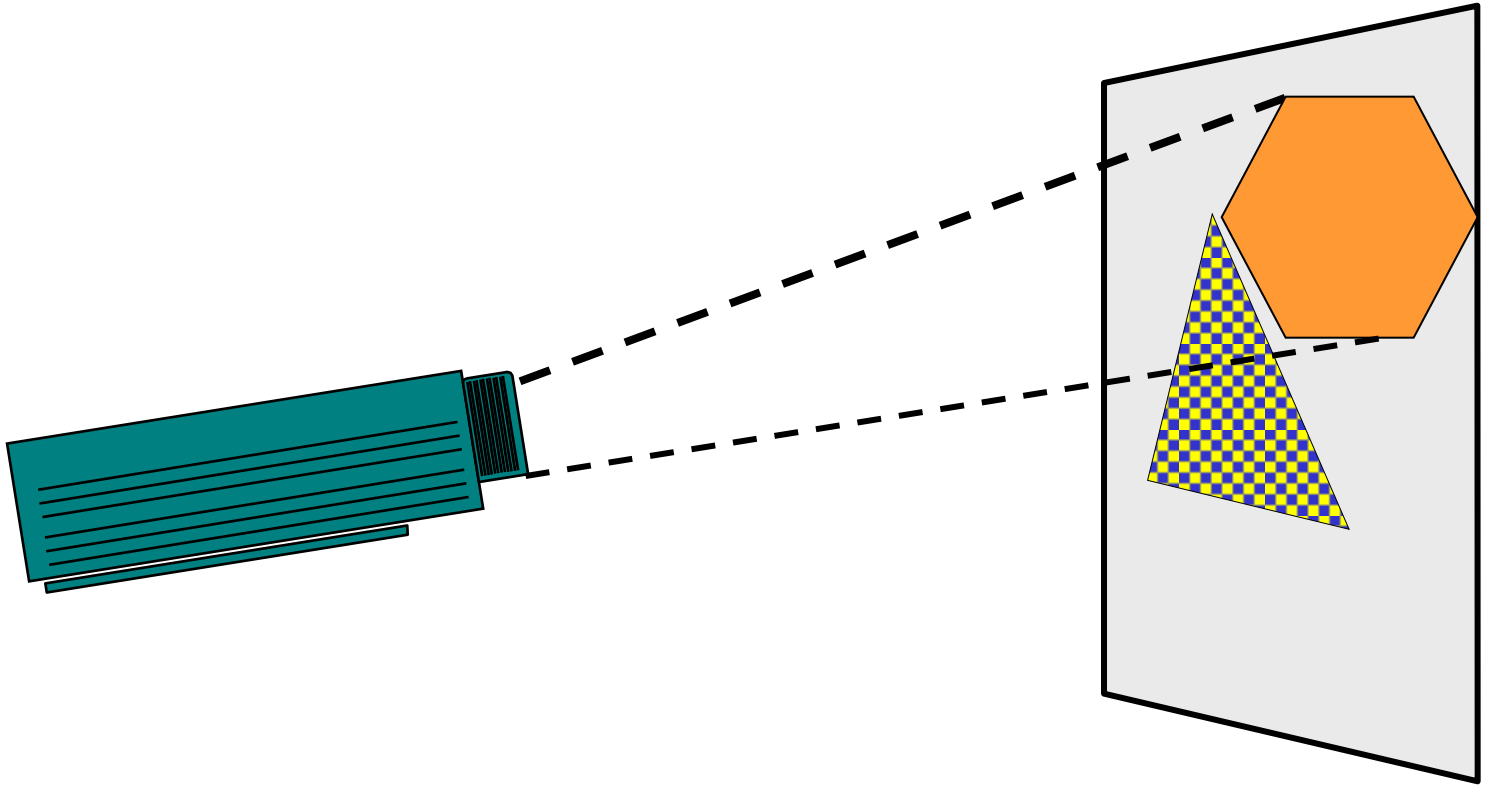
Typically, testers automate by creating static scripts.

Traditional Automated Testing



Given enough time, these scripts will cover the behavior.

Traditional Automated Testing

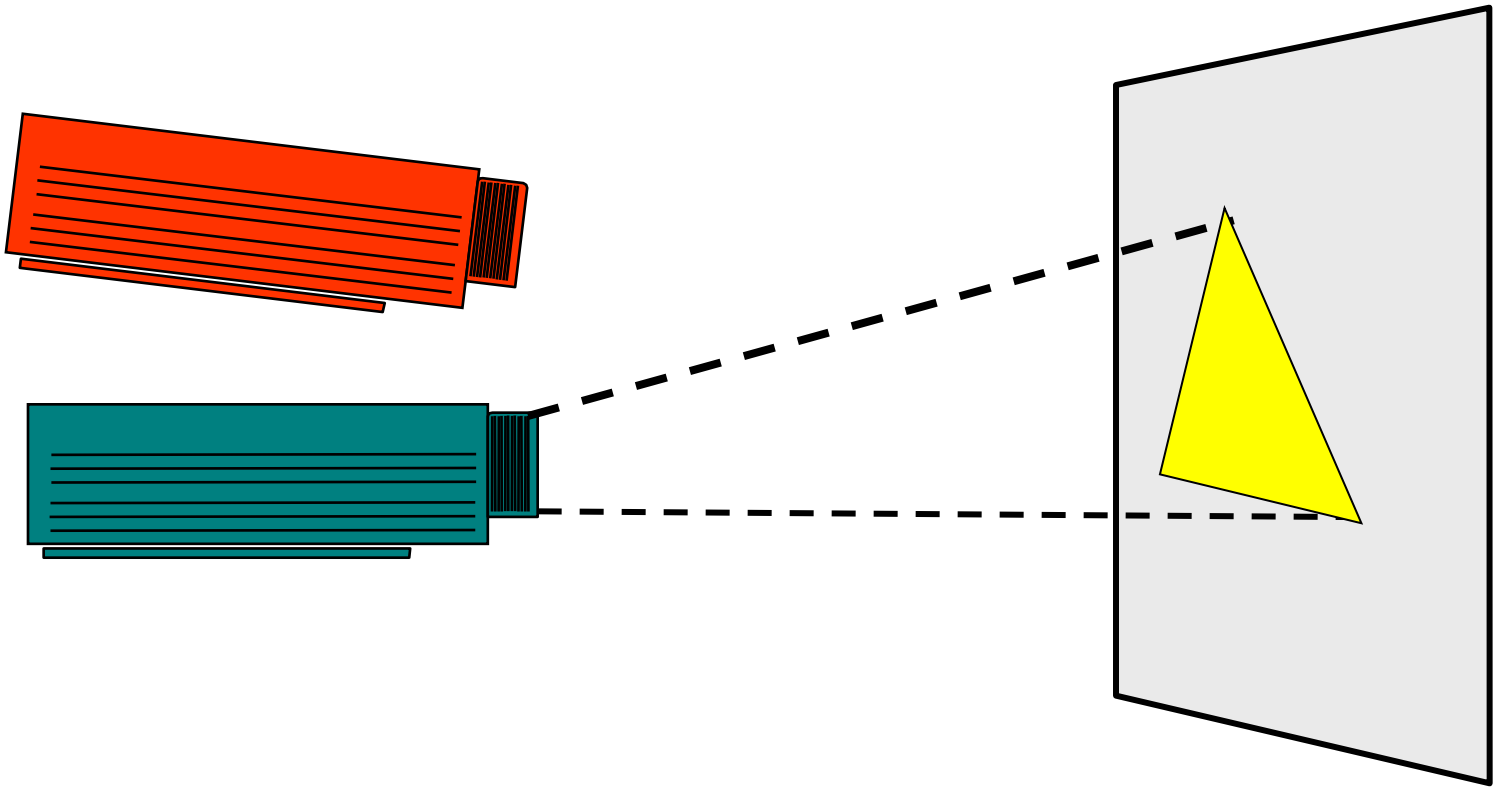


But what happens when the software's behavior changes?

So What's a Model?

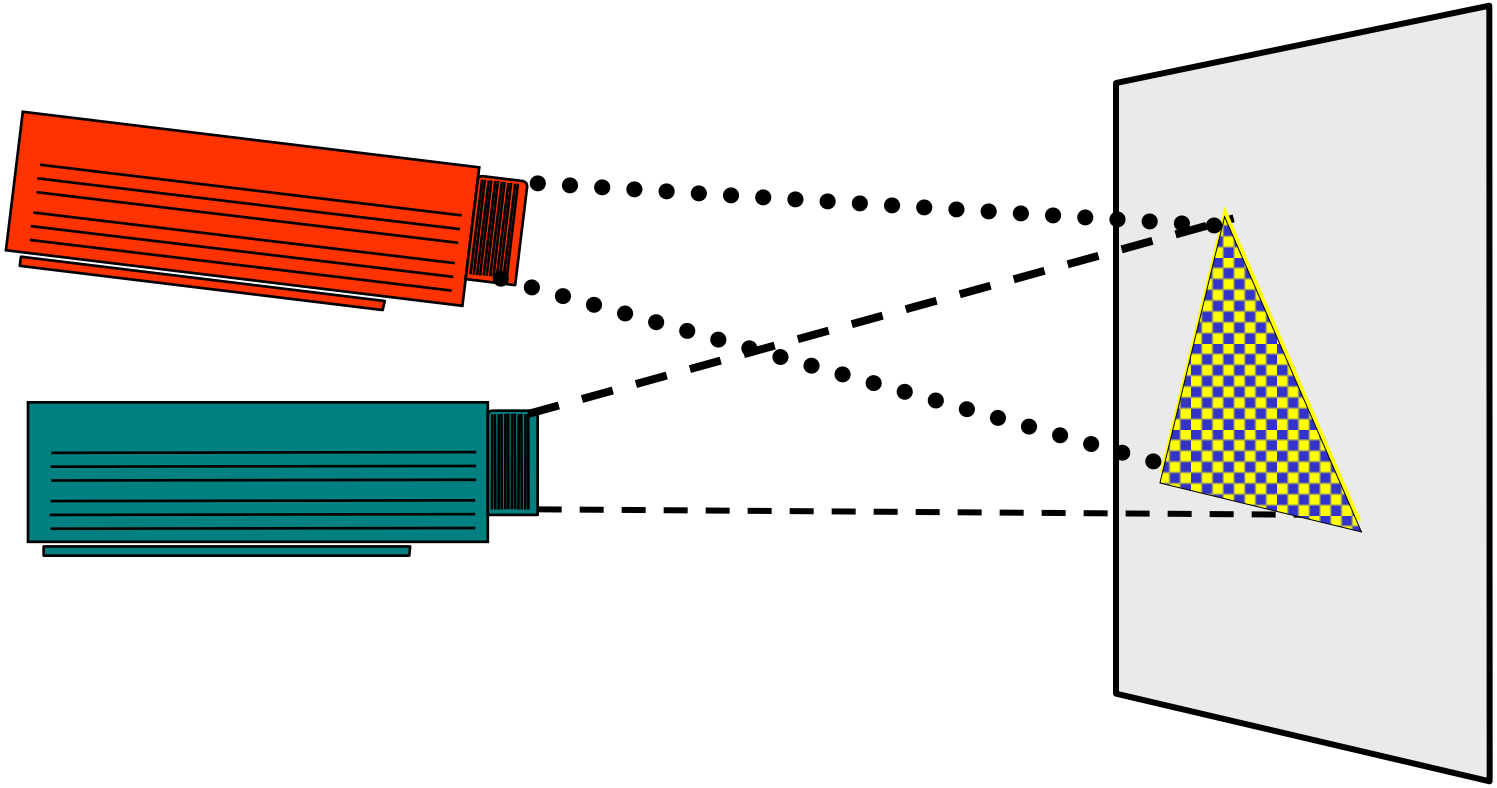
- A model is a description of a system's behavior.
- Models are simpler than the systems they describe.
- Models help us understand and predict the system's behavior.

Model-Based Testing



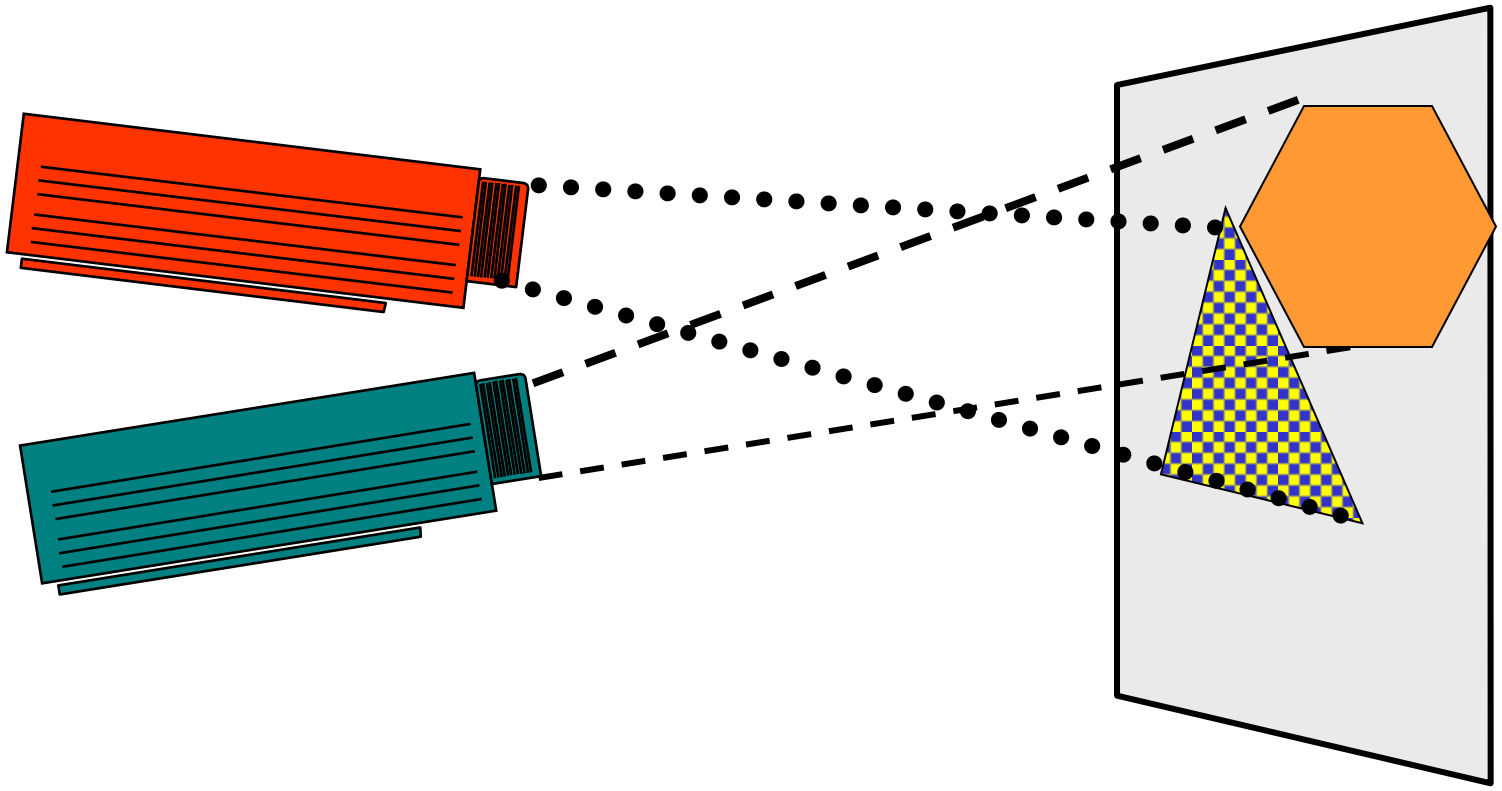
Now, imagine that the top projector is your model.

Model-Based Testing



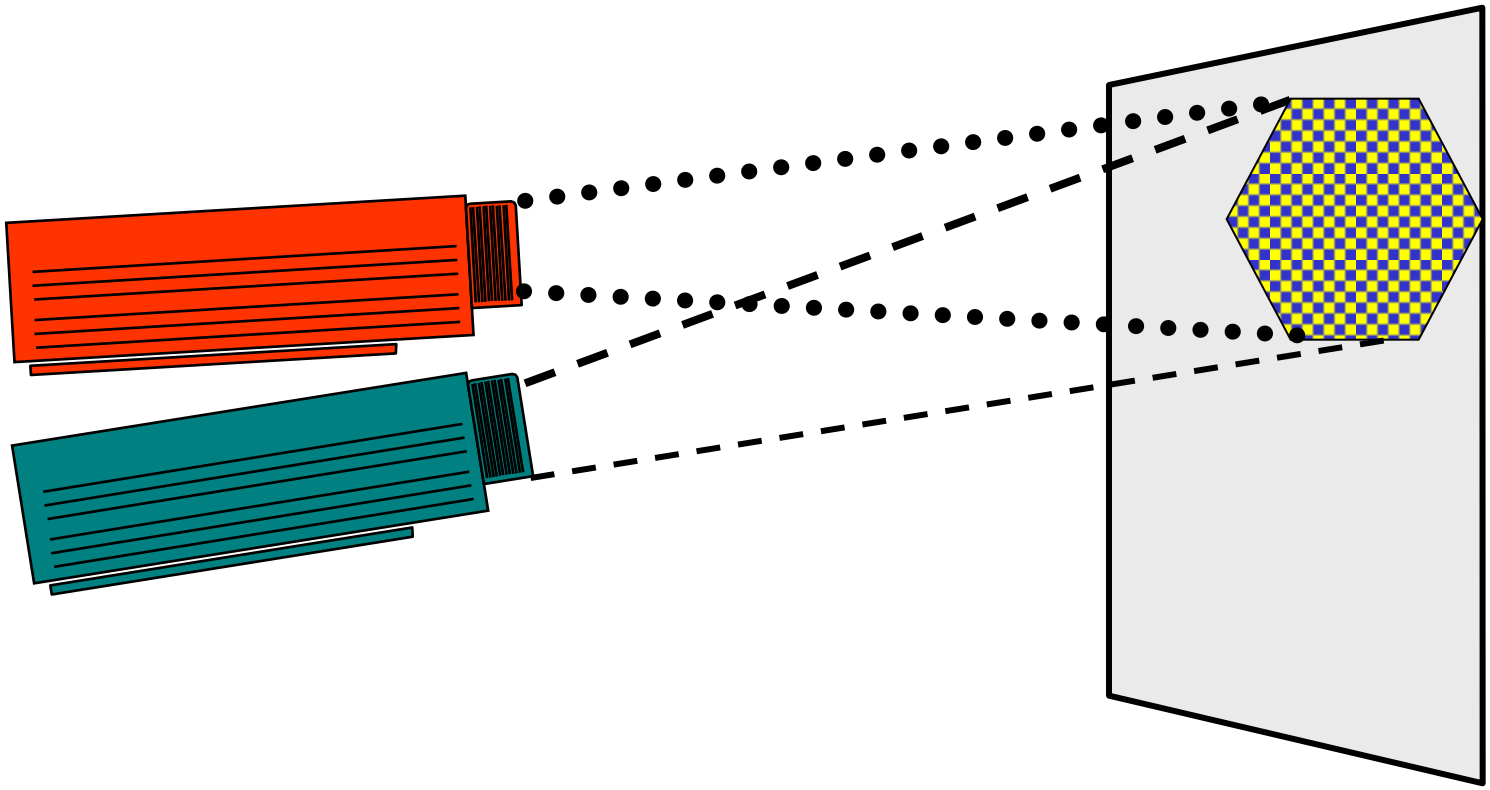
The model generates tests to cover the behavior.

Model-Based Testing



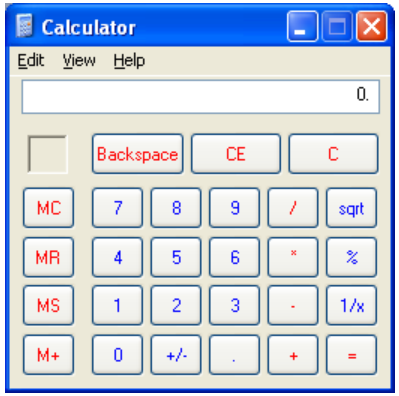
… and when the behavior changes…

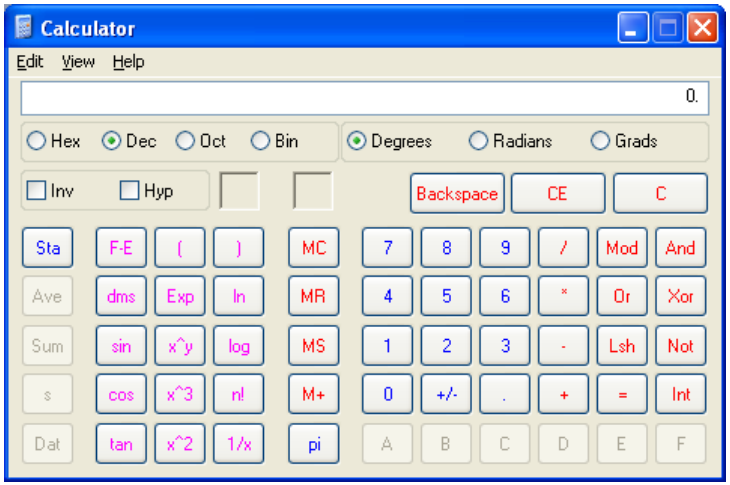
Model-Based Testing



… so do the tests.

Calculator





- Familiar enough
- Simple enough
- Complex enough
- Hard to test well

Exploratory Testing the Calculator

Start
Scientific
Enter a Number
Standard
…

hmm … let's see …

I **Start** the calculator, and the calculator starts running
- OK.

I select **Scientific**, and the calculator goes to Scientific mode.
- Good.

I **Enter a Number**, and the number appears in the display
- Yup.

I select **Standard**, and the calculator goes back to Standard mode
- OK.

…

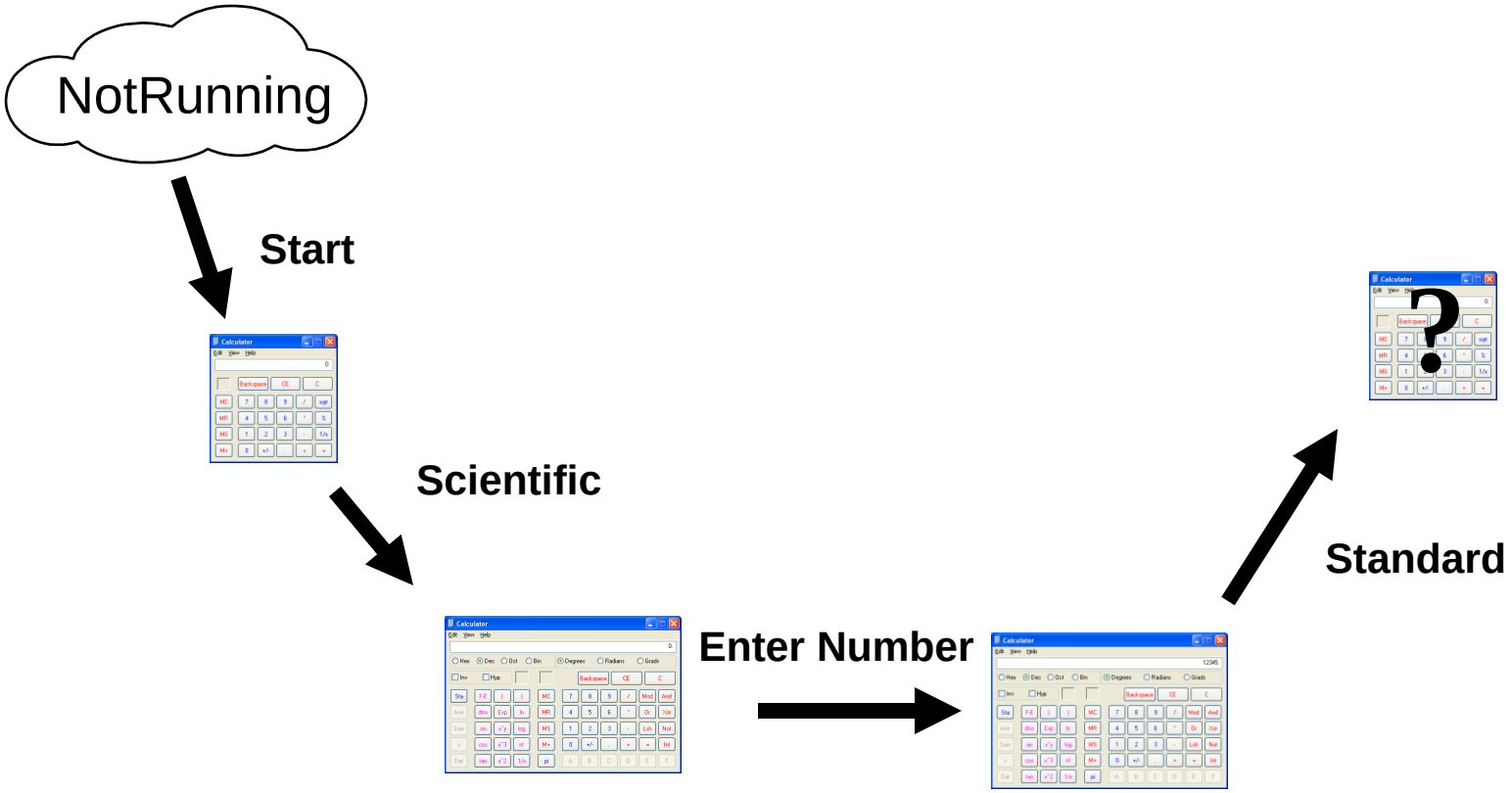
Scripting the Calculator Testing

Step	Action	Result
1	Start	Calculator should be Running
2	Scientific	Calculator in Scientific mode
3	Enter Number	Number in the display
4	Standard	Calculator in Standard mode
	...	

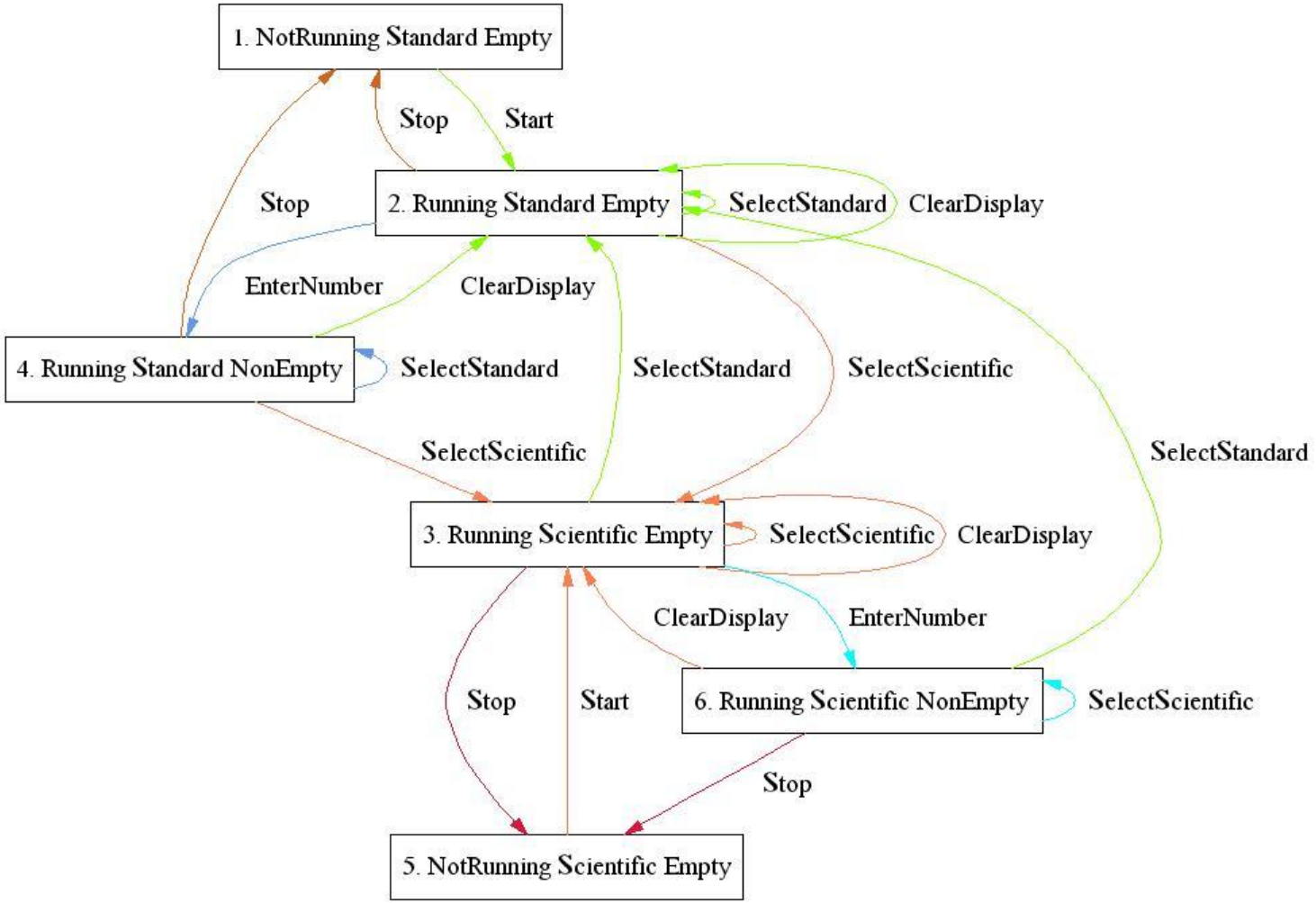
How did you think to try these actions in this order?

And how did you know you'd end up with these results?

You Have a Model in your Head



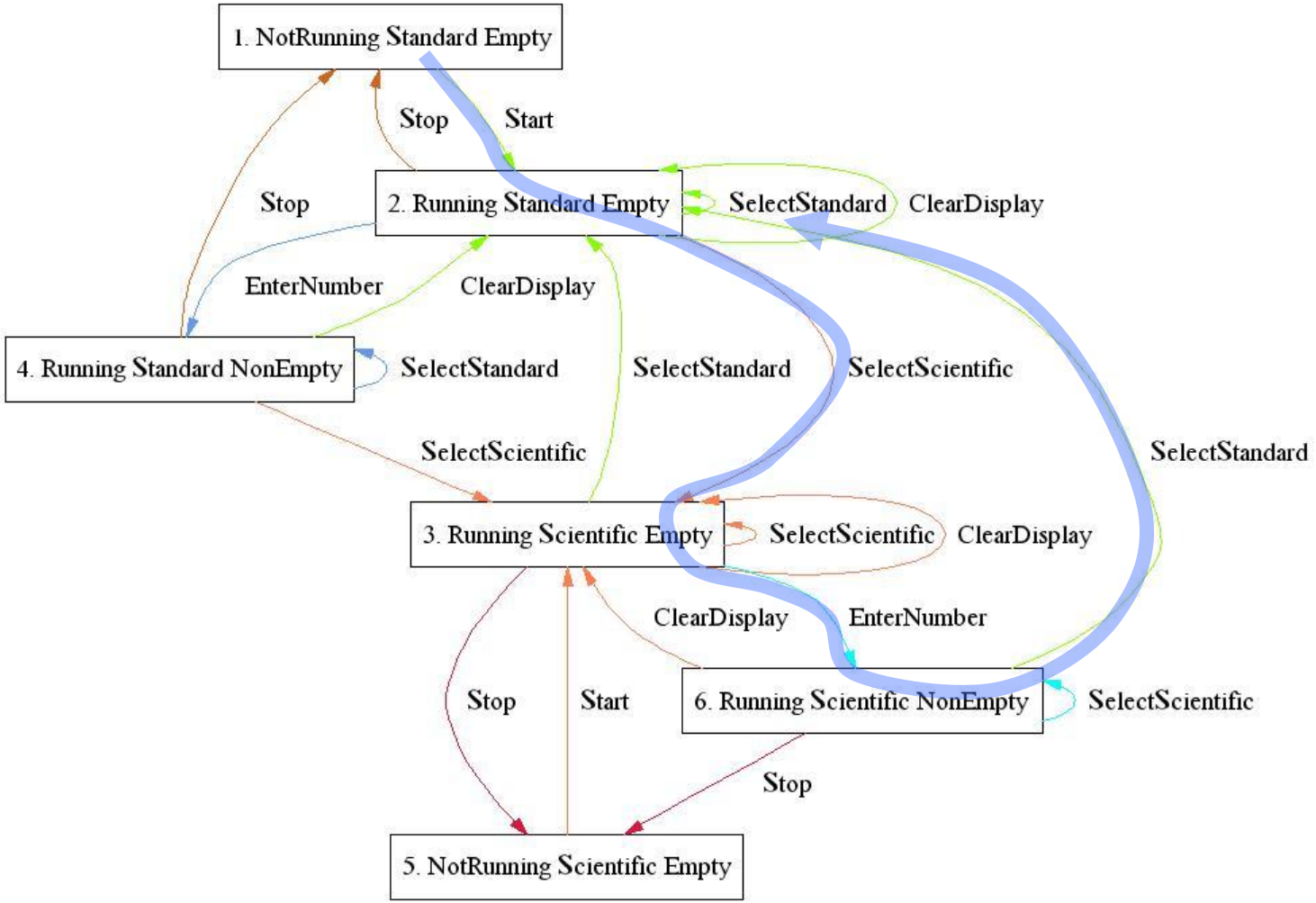
You Could Hand-Draw Your Model



But you wouldn't want to do that.

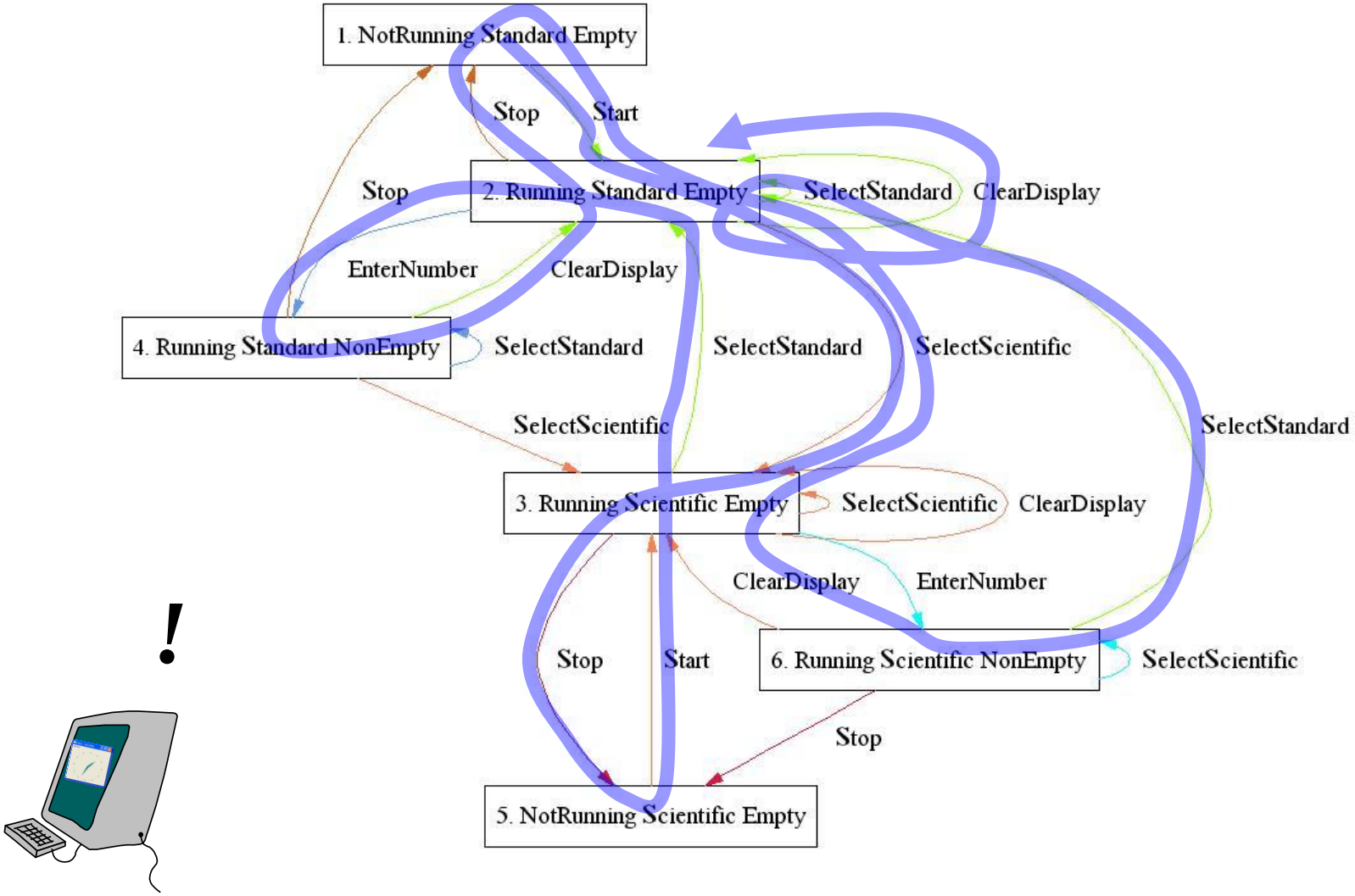
You Could Hand-Trace Test Sequences

1 Start
2 Scientific
3 Enter Number
4 Standard
5 …

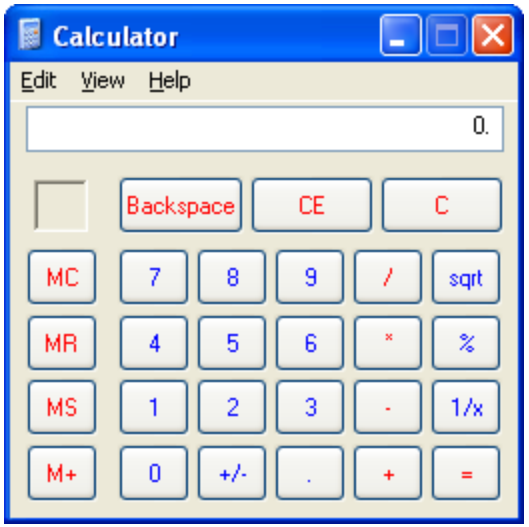


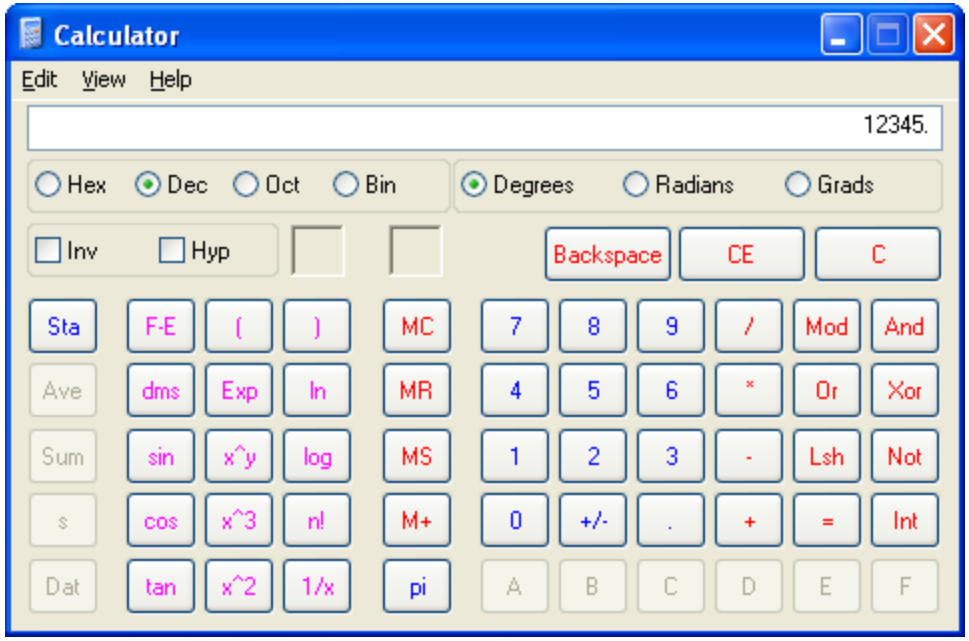
But you wouldn't want to do that either.

Let the Machine Do It for You



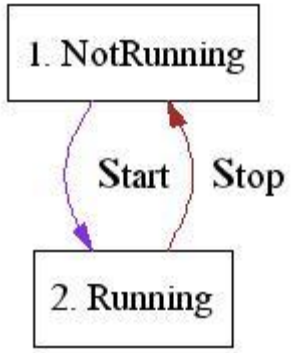
But first …



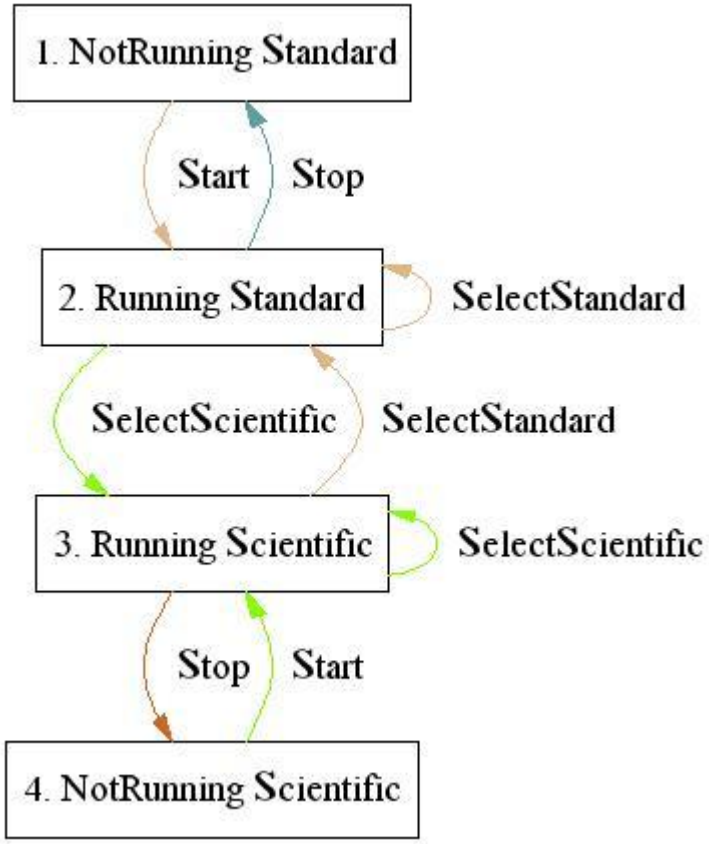


… let's talk about calculator's behavior.

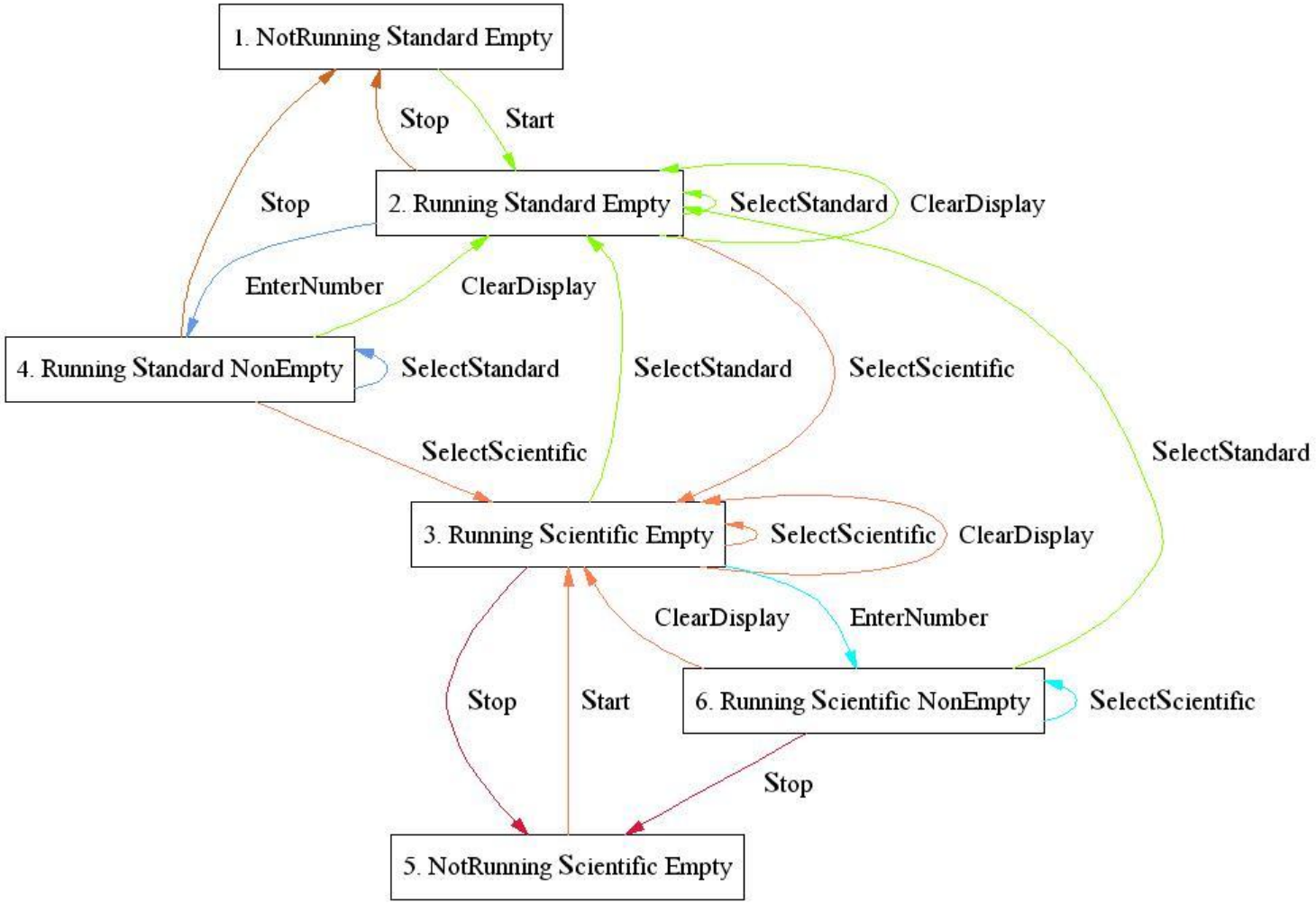
Let's Start Simply



And Get a Bit More Complex



And Even More Complex



What Does This Model Care About?

Application status:

- **Running**
- **Not Running**

Mode status:

- **Standard**
- **Scientific?**

Display status:

- **Empty**
- **Not Empty**

Find the Rules of your Model (Start)

Natural Language

If the Calculator is Not Running then the user can execute ‘Start’.

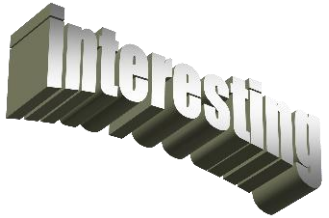
When the user executes ‘Start’, the Calculator goes to ‘Running’ mode

C#

```
if (AppStatus == AppStatusValues.NotRunning)
        a.Add("start")
```

```
AppStatus = AppStatusValues.Running;
```

Find the Rules of your Model (Standard)



Natural Language

If the Calculator is Running then the user can execute 'Standard'.

When the user executes 'Standard', the Calculator goes to 'Standard' mode and the display is cleared.

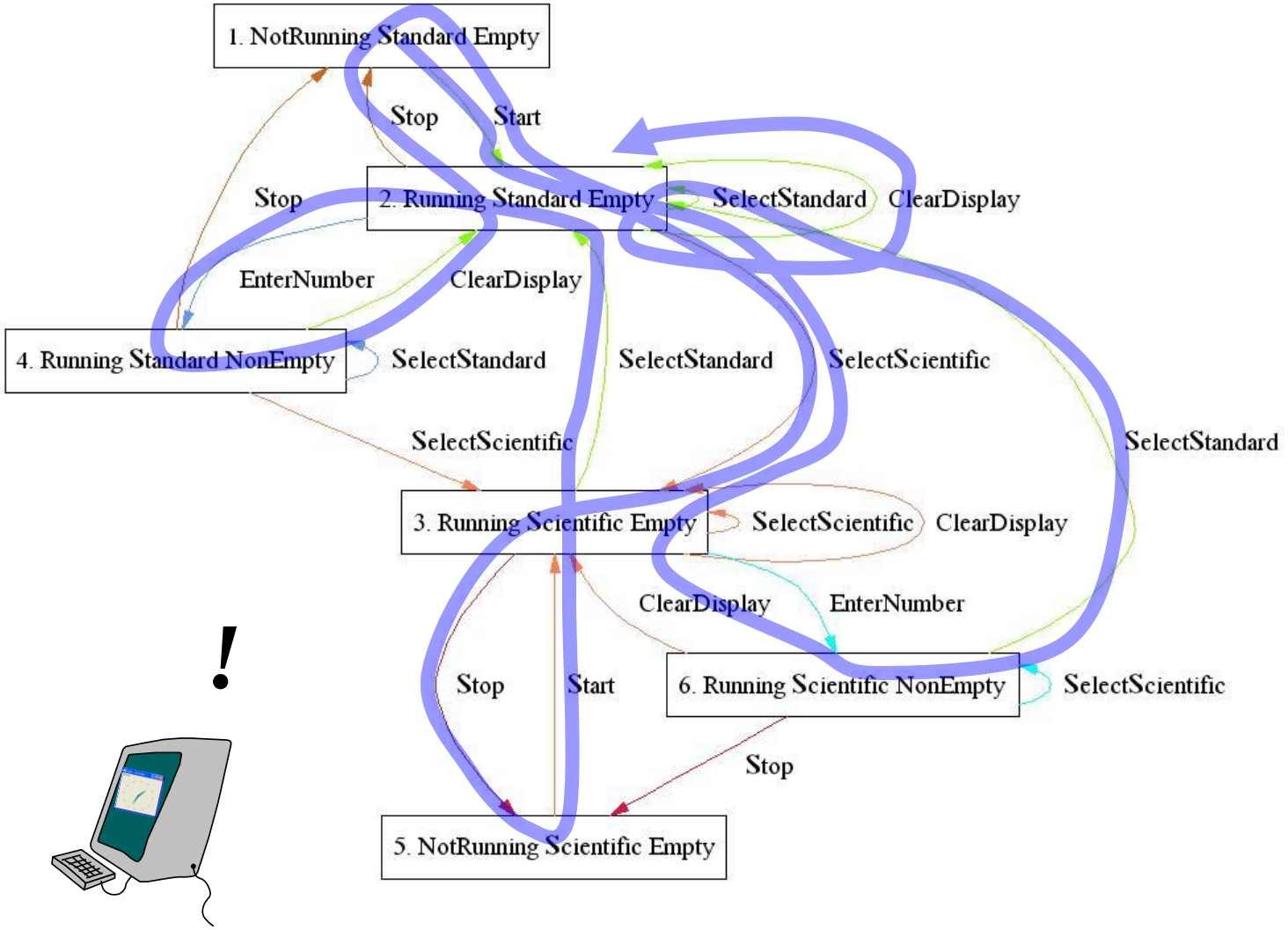
C#

```
if (AppStatus == AppStatusValues.Running)
        a.Add("standard");
```

```
if (ModeStatus == ModeStatusValues.Scientific)
   DisplayStatus = DisplayStatusValues.Empty;
ModeStatus = ModeStatusValues.Standard;
```

Let the Machine Do It for You

1	Start
2	Scientific
3	Stop
4	Start
5	Standard
6	Stop
7	Clear
8	Stop
9	Start
10	Scientific
11	Clear
12	Enter Number
13	Standard
14	Clear
15	…



demo

So let's generate a few thousand tests …

Executing the Test Actions, pt 1

```
while ( (strRecord = StreamToRead.ReadLine()) != null)
{
        string[] individualWords = strRecord.Trim().Split(whitespace.ToCharArray(),strRecord.Length);
        switch( individualWords[1])
        {
                <perform the actions>
        }

}
```

Executing the Test Actions, pt 2

<process the actions>

```
switch( individualWords[1])
{
        case "start":
        Process.Start("calc");
        break;

        case "stop":
        SendKeys.SendWait("%{f4}");
        break;

        case "standard":
        SendKeys.SendWait("%vt");
        SetForegroundWindow(FindWindow(null, "Calculator"));
        break;
…
```

A Brief Filk on Test Oracling

If you wish to succeed
As a tester you need
To consider all matters oracular.
Verifying is tough;
Monkey tests aren't enough;
(Though the crashes are often spectacular!)

Executing the Test Actions, pt 3

<oracle the result>

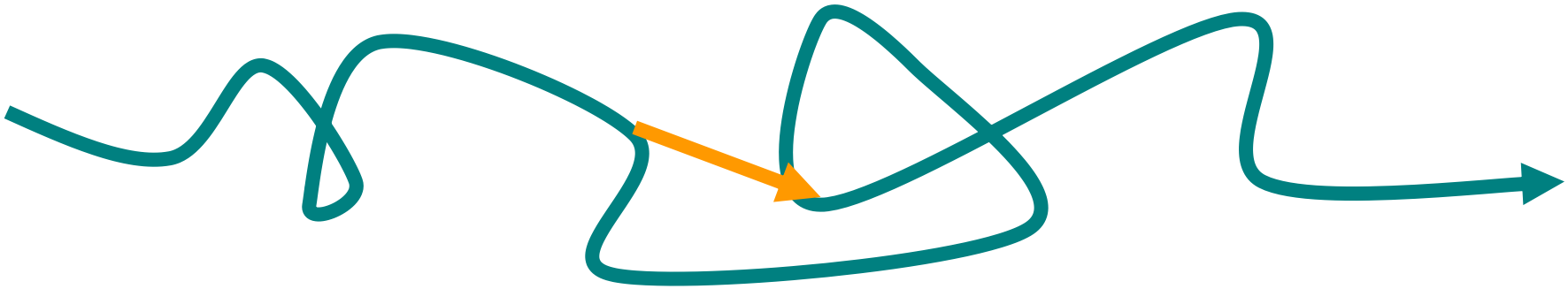
```
// copy display contents to the clipboard
SendKeys.SendWait("^c");

if ((individualWords[4] == "Empty")
&& (GetClipboardContent() != "0"))
{
        Console.WriteLine(" mismatch");
}

if ((individualWords[4] == "NotEmpty")
&& (GetClipboardContent() == "0"))
{
        Console.WriteLine(" mismatch");
}
```

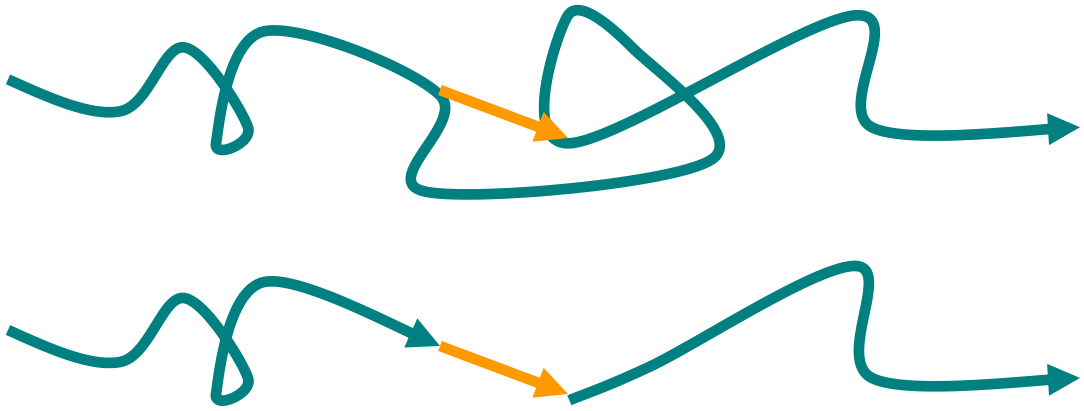
demo

So let's run a few dozen tests …



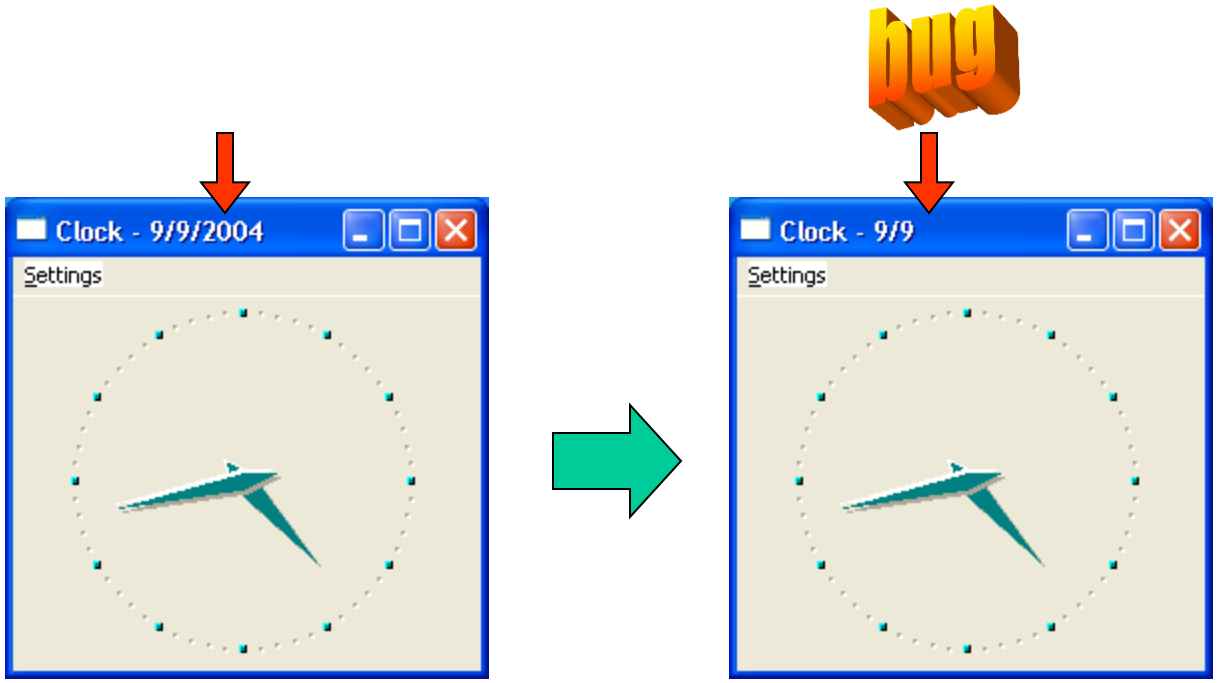
Doing Cool Stuff with Models

Beeline



1. Choose any 2 nodes in the path
2. Find the shortest path between them
3. Execute the spliced ‘shortcut’ path
4. Evaluate the results and repeat

That Was The Year That Wasn't



Start, Minimize, Stop, Start, Restore, Date

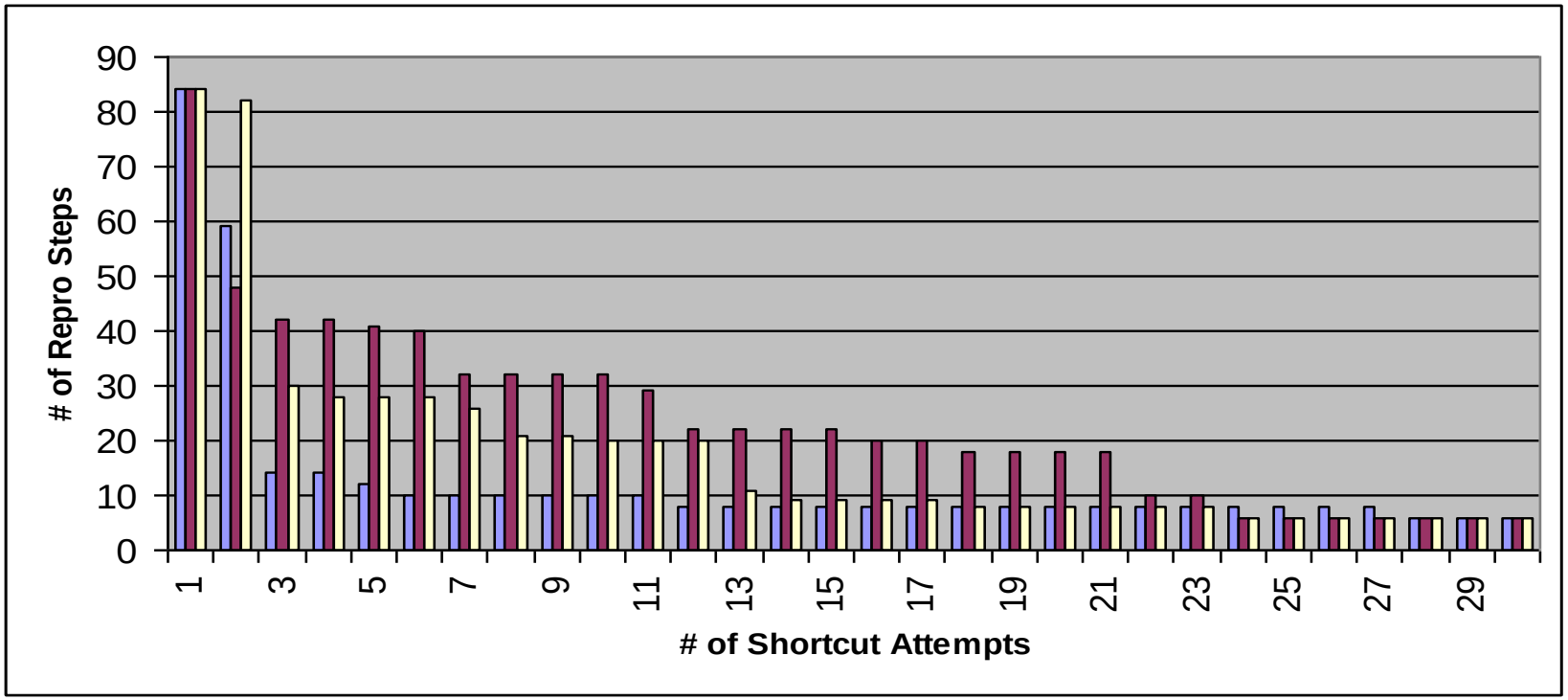
An 84-step bug repro sequence

start about ok_about no_title doubleclick seconds restore seconds doubleclick doubleclick date about ok_about restore gmt maximize doubleclick doubleclick date seconds date stop start stop start stop start seconds date restore about ok_about no_title doubleclick digital doubleclick doubleclick no_title doubleclick no_title doubleclick seconds restore restore doubleclick doubleclick gmt analog maximize date digital minimize restore **minimize stop start restore** digital date minimize stop start maximize gmt digital restore doubleclick doubleclick about ok_about maximize digital digital digital seconds analog about ok_about about ok_about minimize stop start restore **date**

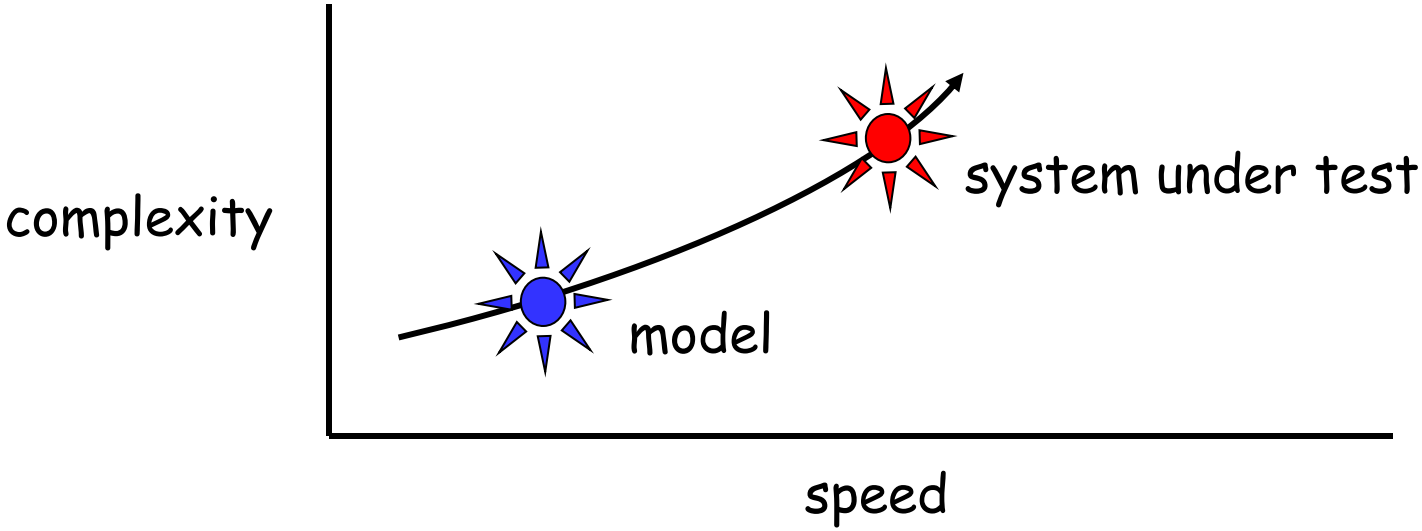
Reducing the Sequence:

- Initial path length: 84 steps
- Shortcut attempt 2 : repro sequence: 83 steps
- Shortcut attempt 3 : repro sequence: 64 steps
- Shortcut attempt 4 : repro sequence: 37 steps
- Shortcut attempt 5 : repro sequence: 11 steps
- Shortcut attempt 7 : repro sequence: 9 steps
- Shortcut attempt 20 : repro sequence: 8 steps
- Shortcut attempt 29 : repro sequence: 6 steps

Repro Steps Over Time



Why Does Model-Based Testing Work?



"… I think that less than 10 percent of most programs' code is specific to the application. Furthermore, that 10 percent is often the easiest 10 percent. Therefore, it is not unreasonable to build a model program to use as an oracle."

–Boris Beizer, Black Box Testing, p.63

Benefits of Model-Based Testing

- Easy test case maintenance
- Reduced costs/more tests
- Can run different tests on 1000s of machines
- Early bug detection
- Increased bug count
- Time savings
- Time to address bigger test issues
- Improved tester job satisfaction
- Start automated testing from Version 0.1

Obstacles to Model-Based Testing

- Comfort factor
  - This is not your parents' test automation
- Skill sets
  - Need testers who can design
- Expectations
  - Models can be a significant upfront investment
  - Will never catch all the bugs
- Metrics
  - Bad metrics: bug counts, number of test cases
  - Better metrics: spec coverage, code coverage

Tools

- Used in this talk:
  - C# (is free)
  - Notepad (is just about free)
  - WinSTDtoDOT (was written by a friend)

Acknowledgments

- Michael Corning
- Rory Clark
- Wolfram Schulte
- Mike Barnett
- Margus Veanes
- Wolfgang Grieskamp

To Learn More

- Model-based testing website:
  www.model-based-testing.org
- Books:

  "Black-Box Testing : Techniques for Functional Testing of Software and Systems" by Boris Beizer

  "Testing Object-Oriented Systems: Models, Patterns, and Tools" by Robert Binder

- A real model-based testing tool:
  - Spec#

thanks!