Goals for Today

• Convey techniques for
  • How to model systems
  • How to generate tests
  • How to verify results

• Communicate a mindset

• Provide inspiration

• Foster discontent
Non-Goals for Today

• Specific tools

• Interfacing with an application
What are the Problems of Software Testing?

• Time is limited

• Applications are complex

• Requirements are fluid

© 2006 Harry Robinson, Google
What's wrong with manual testing?
Manual testing is ok sometimes ...
... but it can rarely go deep enough
What's wrong with scripting?
The Villain of this Piece

- Awe-inspiring
- Unchanging
- Indecipherable

© 2006 Harry Robinson, Google
Scripts are ok for some uses ...

Test case 1: Start Digital Stop Start Analog Stop
Test case 2: Start DblClk Stop Start DblClk Stop
... but they pile up quickly ...

Test case 1: Start Digital Stop Start Analog Stop
Test case 2: Start DblClk Stop Start DblClk Stop
Test case 3: Start Digital DblClk Stop Start DblClk Analog Stop
Test case 4: Start DblClk DblClk Digital DblClk DblClk DblClk Stop Start Analog Stop
Test case 5: …
... and what are you left with?

Test case 1: Start Digital Stop Start Analog Stop
Test case 2: Start DblClk Stop Start DblClk Stop
Test case 3: Start Digital DblClk Stop Start DblClk Analog Stop
Test case 4: Start DblClk DblClk Digital DblClk DblClk Stop Start Analog Stop
Test case 5: Start Digital Digital Stop Start Analog Stop
Test case 6: Start DblClk Stop Start Analog DblClk Stop
Test case 7: Start Digital DblClk Stop Start Digital DblClk Analog Stop
Test case 8: Start DblClk DblClk Digital Analog DblClk DblClk Stop
Test case 9: …
What is a model?

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

“Model-based testing is a testing technique where the runtime behavior of an implementation under test is checked against predictions made by a formal specification, or model.” - Colin Campbell, MSR

**In other words**

- A model describes how a system should behave in response to an action.
- Supply the action and see if the system responds as you expect.
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…

© 2006 Harry Robinson, Google
Model-Based Testing

... so do the tests.
Using Models to Test

- What type of model do I use?
- How do I create the model?
- How do I choose tests?
- How do I verify results?
Many Types of Models

• States
• Monkeys
• Sets
• Grammars
• Combinations
• Other
Creating a Model
We All Use Mental Models Already

if I am in the Analog display
and I execute the Digital action
I should end up in the Digital display
State Table Representation

StartState Action EndState
Analog        Digital        Digital

© 2006 Harry Robinson, Google
Exercise 1

Modeling a Website

Based on “Model-Based Testing: Not for Dummies” by Jeff Feldstein

© 2006 Harry Robinson, Google
State Table Representation

StartState  Action  EndState
HomePage    ImageTab  ImagePage
HomePage    NewsTab   NewsPage
ImagePage   HomeTab  HomePage
ImagePage   NewsTab   NewsPage
NewsPage   HomeTab  HomePage
NewsPage   ImageTab  ImagePage
Exercise 2

Modeling Notepad
Notepad Model

Start → Type A → Delete A → No → Close

Close → notepad.exe

Close → Cancel → Notepad

© 2006 Harry Robinson, Google
<table>
<thead>
<tr>
<th>StartState</th>
<th>Action</th>
<th>EndState</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotRunning</td>
<td>Start</td>
<td>MainWindow</td>
</tr>
<tr>
<td>WindowEmpty</td>
<td>TypeA</td>
<td>WindowDirty</td>
</tr>
<tr>
<td>WindowEmpty</td>
<td>Close</td>
<td>NotRunning</td>
</tr>
<tr>
<td>WindowDirty</td>
<td>Delete</td>
<td>WindowEmpty</td>
</tr>
<tr>
<td>WindowDirty</td>
<td>Close</td>
<td>SaveDialog</td>
</tr>
<tr>
<td>SaveDialog</td>
<td>Cancel</td>
<td>WindowDirty</td>
</tr>
<tr>
<td>SaveDialog</td>
<td>No</td>
<td>NotRunning</td>
</tr>
</tbody>
</table>

© 2006 Harry Robinson, Google
Exercise 3

Modeling Wordpad
Wordpad Model
# Wordpad State Table

<table>
<thead>
<tr>
<th>StartState</th>
<th>Action</th>
<th>EndState</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotRunning</td>
<td>Start</td>
<td>MainWindow</td>
</tr>
<tr>
<td>WindowEmpty1</td>
<td>TypeA</td>
<td>WindowDirty</td>
</tr>
<tr>
<td>WindowEmpty1</td>
<td>Close</td>
<td>NotRunning</td>
</tr>
<tr>
<td>WindowDirty</td>
<td>Delete</td>
<td>WindowEmpty2</td>
</tr>
<tr>
<td>WindowDirty</td>
<td>Close</td>
<td>SaveDialog1</td>
</tr>
<tr>
<td>WindowEmpty2</td>
<td>TypeA</td>
<td>WindowDirty</td>
</tr>
<tr>
<td>WindowEmpty2</td>
<td>Cancel</td>
<td>SaveDialog2</td>
</tr>
<tr>
<td>SaveDialog1</td>
<td>Cancel</td>
<td>WindowDirty</td>
</tr>
<tr>
<td>SaveDialog1</td>
<td>No</td>
<td>NotRunning</td>
</tr>
<tr>
<td>SaveDialog2</td>
<td>Cancel</td>
<td>WindowEmpty2</td>
</tr>
<tr>
<td>SaveDialog2</td>
<td>No</td>
<td>NotRunning</td>
</tr>
</tbody>
</table>
Exercise 4

Modeling Clock using Rules
Modeling Clock States

In our discussion of the Clock behavior, we only tracked a few data values in the application:

- **11:25 PM** was described as **Running Digital Framed**.
- Was described as **Running Analog Unframed**.
Modeling Clock Actions

We also tracked how our actions change those values:

Digital → Analog → Digital

11:25 PM – Running Digital Framed → Digital

© 2006 Harry Robinson, Google
So, we can replace this model ...
... with a state variable model
Rules for the Stop action

• If the Clock is Running, then the user can execute the 'Stop' action.
• The 'Stop' action puts you in Stopped mode

```csharp
static void Stop()
requires (AppStatus == AppValues.Running);
{
    AppStatus = AppValues.Stopped;
}
```
Rules for the SelectDigital action

• If the Clock is Running and Framed, then the user can execute the ‘SelectDigital’ action
• The ‘SelectDigital’ action puts you in Digital mode

static void SelectDigital()
{
    requires (AppStatus == AppValues.Running) && (FrameStatus == FrameValues.Framed);
    {
        ModeStatus = ModeValues.Digital;
    }
}
### A generated state table!

<table>
<thead>
<tr>
<th>STARTSTATE</th>
<th>ACTION</th>
<th>ENDSTATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stopped Analog Framed</td>
<td>Start</td>
<td>Running Analog Framed</td>
</tr>
<tr>
<td>Running Analog Framed</td>
<td>Stop</td>
<td>Stopped Analog Framed</td>
</tr>
<tr>
<td>Running Analog Framed</td>
<td>SelectAnalog</td>
<td>Running Analog Framed</td>
</tr>
<tr>
<td>Running Analog Framed</td>
<td>SelectDigital</td>
<td>Running Digital Framed</td>
</tr>
<tr>
<td>Running Analog Framed</td>
<td>DblClk</td>
<td>Running Analog Unframed</td>
</tr>
<tr>
<td>Running Digital Framed</td>
<td>Stop</td>
<td>Stopped Digital Framed</td>
</tr>
<tr>
<td>Running Digital Framed</td>
<td>SelectAnalog</td>
<td>Running Analog Framed</td>
</tr>
<tr>
<td>Running Digital Framed</td>
<td>SelectDigital</td>
<td>Running Digital Framed</td>
</tr>
<tr>
<td>Running Digital Framed</td>
<td>DblClk</td>
<td>Running Digital Unframed</td>
</tr>
<tr>
<td>Running Analog Unframed</td>
<td>Stop</td>
<td>Stopped Analog Unframed</td>
</tr>
<tr>
<td>Running Analog Unframed</td>
<td>DblClk</td>
<td>Running Analog Unframed</td>
</tr>
<tr>
<td>Stopped Digital Framed</td>
<td>Start</td>
<td>Running Digital Framed</td>
</tr>
<tr>
<td>Running Digital Unframed</td>
<td>Stop</td>
<td>Stopped Digital Unframed</td>
</tr>
<tr>
<td>Running Digital Unframed</td>
<td>DblClk</td>
<td>Running Digital Framed</td>
</tr>
<tr>
<td>Stopped Analog Unframed</td>
<td>Start</td>
<td>Running Analog Unframed</td>
</tr>
<tr>
<td>Stopped Digital Unframed</td>
<td>Start</td>
<td>Running Digital Unframed</td>
</tr>
</tbody>
</table>

© 2006 Harry Robinson, Google
A state diagram

1. Stopped Analog Framed
2. Running Analog Framed
3. Running Digital Framed
4. Running Analog Unframed
5. Stopped Digital Framed
6. Running Digital Unframed
7. Stopped Analog Unframed
8. Stopped Digital Unframed

© 2006 Harry Robinson, Google
Generating Tests from a Model
Generating Test Sequences

We can use the machine-readable model to create test sequences:

- Random walk
- All transitions
- Shortest paths first
- Most likely paths first
A sequence that hits all transitions
An all-transitions sequence

1. Start
2. Analog
3. Digital
4. Digital
5. Stop
6. Start
7. Double Click
8. Stop
9. Start
10. Double Click
11. Analog
12. Double Click
13. Stop
14. Start
15. Double Click
16. Stop
All paths of length 2
All paths of length 3
All paths of length 4
All paths of length < 5

1. Start, Stop
2. Start, Analog, Stop
3. Start, Analog, Analog, Stop
4. Start, Digital, Analog, Stop
5. Start, Double Click, Double Click, Stop
Exercise 5
Brainstorming Traversals
How would you test this?
Pathological
Closer in ...
Partial state table for the maze

<table>
<thead>
<tr>
<th>West end of long hall</th>
<th>South</th>
<th>maze of twisty little passages</th>
</tr>
</thead>
<tbody>
<tr>
<td>maze of twisty little passages</td>
<td>Up</td>
<td>twisty little maze of passages</td>
</tr>
<tr>
<td>maze of twisty little passages</td>
<td>NW</td>
<td>twisty maze of little passages</td>
</tr>
<tr>
<td>maze of twisty little passages</td>
<td>North</td>
<td>maze of little twisty passages</td>
</tr>
<tr>
<td>maze of twisty little passages</td>
<td>NE</td>
<td>twisting maze of little passages</td>
</tr>
<tr>
<td>maze of twisty little passages</td>
<td>West</td>
<td>maze of little twisting passages</td>
</tr>
<tr>
<td>maze of twisty little passages</td>
<td>East</td>
<td>little twisty maze of passages</td>
</tr>
<tr>
<td>maze of twisty little passages</td>
<td>SW</td>
<td>little maze of twisty passages</td>
</tr>
</tbody>
</table>

© 2006 Harry Robinson, Google
Executing the Test Actions

1. Start
2. Analog
3. Digital
4. Digital
5. Stop
6. Start
7. Double Click
8. Stop
Exercise 6

Modeling from a Spec

© 2006 Harry Robinson, Google
“Accessing an Account”

1. customer needs to log in to use the system
2. customer stays logged in to the system until she logs out
3. customer starts with no account
4. customer can create an account
5. customer can delete her account
6. customer can open an existing account
7. customer can close an existing account
Verifying the Outcomes
Oracles

• Crashes
• Prediction
• Checking
• Pre-oracleing
• Heuristics
• Filtering + Humans
• Assertions

© 2006 Harry Robinson, Google
Exercise 7

Monkeying an Input Field

Inspired by Noel Nyman’s article “Using Monkey Test Tools”
Monkey Models

Nothing I do should make this app crash…

1.2
a P, a P, ab P, ab P 2, abc P, P, P, P,
\scbuild1
Type 5
containing message laurie Bill"s October
? A
? v?-0Qrg+ 'cQ?_<$ ` Z`i7c} oV? E1X …
nov 31, 2000 Oct, 2000 dates Wednesday …
alike. In Word documents, for example …
3 M note
RNL in Office 10
Predicting the Outcome

StartState: Analog  Action: Digital  EndState: Digital
Exercise 8

Oracling the Square Root Function

\[ \sqrt{a} \]
\sqrt{a} \times \sqrt{a} = a
Exercise 9

Oracling the Sine Function

Inspired by Doug Hoffman’s article “Heuristic Test Oracles”

© 2006 Harry Robinson, Google
\[ \sin^2(x) + \cos^2(x) = 1 \]
\[ \cos(x) = \sin(x - \pi) \]
\[ \sin^2(x) + \sin^2(x - \pi) = 1 \]
Modeling with Sets
Exercise 10

Modeling Search

© 2006 Harry Robinson, Google
Software Testing Analysis Review (STARWEST 2005)
Desktop\051128 cleanup\proceedings\HTML\speakers.html - Open folder - 1 cached - Nov 16
Microsoft PowerPoint - W9

data-centric tests where each test does the same kind of thing to
different kinds of data. Available in different languages (Java, C++, C#,
Python, Perl, Ruby, FitNesse), a
Desktop\051128 cleanup\proceedings\pdf\W9.pdf - Open folder -
1 cached - Nov 16
Pragmatic Unit Testing

fun. For more information, as well as the latest Pragmatic titles, please visit us at: http://www.pragmaticprogrammer.com Copyright © 2003, 2004 The Pragmatic Programmers

Desktop\PragmaticUnitTestingInC#.pdf - Open folder - 1 cached - Oct 12
3513 – 12 = ?

+ruby
- pragmatic
3143 files

+ruby
+ pragmatic
12 files

-ruby
+ pragmatic
70 files

+ruby
3513 files

+pragmatic
82 files

© 2006 Harry Robinson, Google
Modeling with Grammars

Expr ::= IDENT
     | NUMBER
     | "let" Ident = Expr "in" Expr
     | Expr + Expr
     | Expr - Expr
How a Production Grammar Works

A → BB
B → CAC
C → DD
A mini-html grammar

\{seed\}:=<html>{body}</html>;

\{body\}:=\{body\}<p>{body}\mid<H1>{body2}</H1>|<H4>{body2}</H4>;

\{body2\}:=<font color="red"> \{body3\} </font>|<font color="blue"> \{body3\} </font>;

\{body3\}:=<U>{body4}</U>|{body4};

\{body4\}:=<I>{body5}</I> | {body5};

\{body5\}:=the quick brown fox;
Grammar Example 1

Modeling Arithmetic Expressions

100 * 5 / (6 - 5.01 * 3.14159)

Inspired by Pete TerMaat’s article “Adventures in Automated Testing”
Evaluate the following expressions

1. $10 \times 1.0 \times 0.1 \times 6.0 \times 1.16666666667 \times 1.0 \times 1.14285714286 \times 5.25$
2. $2 \times 2.5 \times 0.4 \times 4.0 \times 1.25 \times 1.0 \times 0.8 \times 5.25$
3. $4 \times 0.5 \times 3.5 \times 1.0 \times 0.285714285714 \times 3.0 \times 0.166666666667 \times 42.0$
4. $2 \times 5.0 \times 0.2 \times 1.5 \times 1.0 \times 2.66666666667 \times 1.125 \times 4.66666666667$
5. $5 \times 1.6 \times 1.0 \times 0.5 \times 1.75 \times 0.428571428571 \times 0.333333333333 \times 42.0$
6. $4 \times 2.0 \times 0.75 \times 1.5 \times 0.444444444444 \times 0.75 \times 2.66666666667 \times 5.25$
7. $6 \times 1.5 \times 0.777777777778 \times 0.428571428571 \times 1.333333333333 \times 0.25 \times 7.0 \times 6.0$
8. $2 \times 1.5 \times 0.333333333333 \times 10.0 \times 0.5 \times 0.8 \times 1.5 \times 7.0$
9. $5 \times 0.6 \times 1.66666666667 \times 0.2 \times 1.0 \times 1.0 \times 10.0 \times 4.2$
42

\[
5 \times 8.4
\]

\[
5 \times 1.0 \times 8.4
\]

\[
1 \times 5.0 \times 1.0 \times 8.4
\]

\[
1 \times 1.0 \times 5.0 \times 1.0 \times 8.4
\]

\[
10 \times 0.1 \times 1.0 \times 5.0 \times 1.0 \times 8.4
\]

\[
8 \times 1.25 \times 0.1 \times 1.0 \times 5.0 \times 1.0 \times 8.4
\]

\[
7 \times 1.14285714286 \times 1.25 \times 0.1 \times 1.0 \times 5.0 \times 1.0 \times 8.4
\]

© 2006 Harry Robinson, Google
Evaluate the following expression

\[ 10 \times 0.6 \times 1.5 \times 0.666666666667 \times 1.0 \times 1.0 \times 0.833333333333 \times 1.2 \times 1.5 \times 0.666666666667 \times 1.33333333333 \times 0.375 \times 1.666666666667 \times 1.0 \times 2.0 \times 1.0 \times 0.9 \times 0.777777777778 \times 1.0 \times 0.428571428571 \times 3.0 \times 0.666666666667 \times 1.5 \times 0.222222222222 \times 2.0 \times 0.5 \times 2.0 \times 0.4 \times 2.0 \times 0.5 \times 0.75 \times 1.33333333333 \times 0.75 \times 2.33333333333 \times 1.428571428571 \times 0.7 \times 1.0 \times 1.28571428571 \times 1.0 \times 0.33333333333 \times 2.0 \times 0.83333333333 \times 0.2 \times 6.0 \times 1.33333333333 \times 0.375 \times 0.666666666667 \times 1.0 \times 2.0 \times 2.25 \times 0.222222222222 \times 4.5 \times 0.888888888889 \times 0.125 \times 4.0 \times 2.0 \times 0.75 \times 1.0 \times 1.5 \times 0.33333333333 \times 2.33333333333 \times 0.428571428571 \times 3.0 \times 0.888888888889 \times 0.625 \times 0.2 \times 8.0 \times 0.375 \times 0.33333333333 \times 7.0 \times 1.428571428571 \times 0.1 \times 3.0 \times 3.33333333333 \times 1.0 \times 0.6 \times 0.33333333333 \times 4.0 \times 0.5 \times 2.5 \times 0.5 \times 1.4 \times 1.14285714286 \times 0.25 \times 0.5 \times 7.0 \times 0.428571428571 \times 2.0 \times 1.0 \times 0.5 \times 0.666666666667 \times 4.5 \times 0.33333333333 \times 2.33333333333 \times 0.571428571429 \times 1.75 \times 0.428571428571 \times 2.0 \times 0.5 \times 14.0 \]
Grammar Example 2

Modeling C Code

```
static void TestCase()
    while( Fun("L21") )
        L22: break; }
    else {
        L3:
            L31: goto L22;
            L32: goto L31; }
```
Grammar Model for a Subset of C

Statement :: if ( BooleanCondition ) { Statement } else { Statement }
Statement :: while ( BooleanCondition ) { Statement }
Statement :: { Statement Statement }
Statement :: break;
Statement :: goto Label;
Statement :: Label : Statement
Statement :: ExpressionStatement;
ExpressionStatement :: MethodCall
BooleanExpression :: MethodCall
MethodCall :: Fun(Label)

Based on “A High-Level Modular Definition of the Semantics of C#”
- Boerger, Fruja, Gervasi, Stark

© 2006 Harry Robinson, Google
Deriving Programs from the Grammar

{ Statement }

{ if ( Boolean ) { Statement } else { Statement } }

{ if ( MethodCall ) { Statement } else { Statement } }

{ if ( MethodCall ) { Statement Statement } else { Statement } }
Tests Generated from the Grammar

suite123.cs

```csharp
static void TestCase()
{
    L:
    if ( Fun("L1") )
    {
        L2:
        if ( Fun("L21") )
        {
            L22: goto L;
        }
    }
    else{
        L23: goto L;
    }
}
```

suite321.cs

```csharp
static void TestCase()
{
    L:
    if ( Fun("L1") )
    {
        L2:
        while( Fun("L21") )
        {
            L22:
            break;
        }
    }
    else{
        L3:
        {
            L31: goto L;
            L32: goto L31;
        }
    }
}
```

“Experiments on Semantics Based Testing of a Compiler” - Esin, Novikov, Yavorskiy

© 2006 Harry Robinson, Google
Exercise 13

Selecting Sandwiches

Inspired by an example in Lou Tylee’s book “Visual C# Express for Kids”

© 2006 Harry Robinson, Google
## Setting Up the Combinations

<table>
<thead>
<tr>
<th>Bread</th>
<th>Cheese</th>
<th>H</th>
<th>MU</th>
<th>MA</th>
<th>L</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>white</td>
<td>none</td>
<td>ham</td>
<td>mustard</td>
<td>mayo</td>
<td>lettuce</td>
<td>tomato</td>
</tr>
<tr>
<td>wheat</td>
<td>american</td>
<td>no_ham</td>
<td>no_must</td>
<td>no_mayo</td>
<td>no_lett</td>
<td>no_tom</td>
</tr>
<tr>
<td>rye</td>
<td>swiss</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using James Bach’s AllPairs program from www.satisfice.com

© 2006 Harry Robinson, Google
## Generating the Pairs

### TEST CASES

<table>
<thead>
<tr>
<th>case</th>
<th>Bread</th>
<th>Cheese</th>
<th>H</th>
<th>M</th>
<th>MA</th>
<th>L</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>white</td>
<td>none</td>
<td>ham</td>
<td>mustard</td>
<td>mayo</td>
<td>lettuce</td>
<td>tomato</td>
</tr>
<tr>
<td>2</td>
<td>white</td>
<td>american</td>
<td>no_ham</td>
<td>no_must</td>
<td>no_mayo</td>
<td>no_lett</td>
<td>no_tom</td>
</tr>
<tr>
<td>3</td>
<td>wheat</td>
<td>none</td>
<td>no_ham</td>
<td>mustard</td>
<td>no_mayo</td>
<td>lettuce</td>
<td>no_tom</td>
</tr>
<tr>
<td>4</td>
<td>wheat</td>
<td>american</td>
<td>ham</td>
<td>no_must</td>
<td>mayo</td>
<td>no_lett</td>
<td>tomato</td>
</tr>
<tr>
<td>5</td>
<td>rye</td>
<td>swiss</td>
<td>ham</td>
<td>mustard</td>
<td>no_mayo</td>
<td>no_lett</td>
<td>tomato</td>
</tr>
<tr>
<td>6</td>
<td>rye</td>
<td>swiss</td>
<td>no_ham</td>
<td>no_must</td>
<td>mayo</td>
<td>lettuce</td>
<td>no_tom</td>
</tr>
<tr>
<td>7</td>
<td>white</td>
<td>none</td>
<td>ham</td>
<td>no_must</td>
<td>no_mayo</td>
<td>no_lett</td>
<td>no_tom</td>
</tr>
<tr>
<td>8</td>
<td>white</td>
<td>american</td>
<td>no_ham</td>
<td>mustard</td>
<td>mayo</td>
<td>lettuce</td>
<td>tomato</td>
</tr>
<tr>
<td>9</td>
<td>wheat</td>
<td>swiss</td>
<td>ham</td>
<td>mustard</td>
<td>mayo</td>
<td>no_lett</td>
<td>no_tom</td>
</tr>
<tr>
<td>10</td>
<td>rye</td>
<td>none</td>
<td>no_ham</td>
<td>no_must</td>
<td>no_mayo</td>
<td>lettuce</td>
<td>tomato</td>
</tr>
<tr>
<td>11</td>
<td>rye</td>
<td>american</td>
<td>ham</td>
<td>mustard</td>
<td>no_mayo</td>
<td>lettuce</td>
<td>no_tom</td>
</tr>
<tr>
<td>12</td>
<td>white</td>
<td>swiss</td>
<td>no_ham</td>
<td>no_must</td>
<td>no_mayo</td>
<td>no_lett</td>
<td>tomato</td>
</tr>
</tbody>
</table>
But, did we miss any behavior of interest?
Exercise 14

Modeling the Triangle

Inspired by B J Rollison’s conference paper “Dissecting the Triangle Problem”
triangle test cases for the triangle program, that in no way guarantees
the correct detection of all equilateral triangles. The program could
contain a special check for values 3842,3842,3842 and denote such a
triangle as a scalene triangle. Since the program is a black box, the
only way to be sure of detecting the presence of such a statement is
by trying every input condition.
Exhaustion?

for ( a=0; a<=32767; a++)
  for ( b=0; b<=32767; b++)
    for ( c=0; c<=32767; c++)
      evaluate( a, b, c);
Exercise 15

Modeling Google Maps

© 2006 Harry Robinson, Google
| **Start address:** | 720 4th Ave  
|                  | Kirkland, WA 98033 |
| **End address:** | Kirkland, WA         |
| **Distance:**    | 0.9 mi (about 1 min) |
Trip #115
Trip start: 47.7945, -122.493
Trip end: 47.74753, -122.36585

Google route distance is 102.0 miles
Straightline distance is 6.7 miles

Ratio is 15.2 --------------------- SUSPICIOUS...
Start address: 47.794500, -122.492000
   +47° 47' 40.20", -122° 29' 31.20"

End address: 47.747530, -122.365850
   +47° 44' 51.11", -122° 21' 57.06"

Distance: 12.3 mi (about 2 hours 20 mins)

© 2006 Harry Robinson, Google
<table>
<thead>
<tr>
<th>Start address:</th>
<th>47.794500, -122.493000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+47° 47' 40.20&quot;, -122° 29' 34.80&quot;</td>
</tr>
<tr>
<td>End address:</td>
<td>47.747530, -122.365850</td>
</tr>
<tr>
<td></td>
<td>+47° 44' 51.11&quot;, -122° 21' 57.06&quot;</td>
</tr>
<tr>
<td>Distance:</td>
<td>102 mi (about 2 hours 25 mins)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Start address:</th>
<th>47.794500, -122.492000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+47° 47' 40.20&quot;, -122° 29' 31.20&quot;</td>
</tr>
<tr>
<td>End address:</td>
<td>47.747530, -122.365850</td>
</tr>
<tr>
<td></td>
<td>+47° 44' 51.11&quot;, -122° 21' 57.06&quot;</td>
</tr>
<tr>
<td>Distance:</td>
<td>12.3 mi (about 2 hours 20 mins)</td>
</tr>
</tbody>
</table>
Checking Driving Directions

<table>
<thead>
<tr>
<th>Maps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start address: CA 92564</td>
</tr>
<tr>
<td>End address: CA 90747</td>
</tr>
<tr>
<td>Distance: 79.8 mi (about 1 hour 21 mins)</td>
</tr>
</tbody>
</table>

© 2006 Harry Robinson, Google
Checking Driving Directions

Maps

<table>
<thead>
<tr>
<th>Start address</th>
<th>CA 90747</th>
</tr>
</thead>
<tbody>
<tr>
<td>End address</td>
<td>CA 92564</td>
</tr>
<tr>
<td>Distance</td>
<td>149 mi (about 2 hours 29 mins)</td>
</tr>
</tbody>
</table>

© 2006 Harry Robinson, Google
The Long Road Home

Start address: CA 92564
End address: Long Beach, CA 90813
Distance: 180 mi (about 3 hours 3 mins)

Start address: Long Beach, CA 90813
End address: CA 92564
Distance: 75.1 mi (about 1 hour 21 mins)
What Kinds of Bugs do Models Find?
The Incredible Shrinking Clock

Start
Maximize
Stop
Minimize
Stop
Start
Restore
Stop

© 2006 Harry Robinson, Google
That Was the Year that Wasn’t

Start
Minimize
Stop
Start
Restore
Date

© 2006 Harry Robinson, Google
What Kinds of Bugs do Models **NOT** Find?
Where is Model-Based Testing Heading?

1. Security Testing
2. Shortening Bug Repro Scenarios
3. Meaningful Regression Testing
4. Machine Learning?
Security Testing

“The testing method developed in the PROTOS project is uniquely practical .... Tests are conducted by bombing software with illegally formatted or unexpected input.”

Tekes - National Technology Agency of Finland

CERT Advisories

• CA-2001-18 Multiple Vulnerabilities in LDAP Implementations
• CA-2002-03 Multiple Vulnerabilities in SNMP Implementations
• CA-2003-06 Multiple Vulnerabilities in SIP Implementations
• CA-2003-11 Multiple Vulnerabilities in Lotus Notes and Domino

© 2006 Harry Robinson, Google
Shortening Repro Scenarios
The Motivation

“The fewer steps that it takes to reproduce a bug, the fewer places the programmer has to look (usually).

If you make it easier to find the cause and test the change, you reduce the effort required to fix the problem. Easy bugs gets fixed even if they are minor.”

- from Testing Computer Software
The Beeline Approach

A repro path is simply another traversal through the state model, so …

1. Choose any 2 nodes in the repro path
2. Find the shortest path between them
3. Execute the spliced ‘shortcut’ path
4. Evaluate the results and repeat
The repro path reduction problem
Random walk finds a bug

... but the repro path is inconveniently long
1. Choose any 2 nodes in the path
2. Find shortest path between them
3. Execute the spliced shortcut path

The bug repro’ed - this is the new shortest path
Continue trimming ...
... until you stop.
Why Use a Model for Reducing?

• The model can detect (and therefore reduce) both crashing AND non-crashing bugs.

• Finding a shortcut is simple in a model, so the reduction is more efficient.

• Finding bugs is good. Getting them fixed is better.
That Was The Year That Wasn’t

Start
Minimize
Stop
Start
Restore
Date

© 2006 Harry Robinson, Google
An 84-step repro sequence

invoke about ok_about no_title doubleclick seconds restore seconds doubleclick doubleclick date about ok_about restore gmt maximize doubleclick doubleclick date seconds date close_clock invoke close_clock invoke close_clock invoke close_clock invoke 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 close_clock invoke restore digital date minimize close_clock invoke maximize gmt digital restore doubleclick doubleclick about ok_about maximize digital digital digital seconds analog about ok_about maximize about ok_about minimize minimize close_clock invoke restore date

© 2006 Harry Robinson, Google
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

![Graph showing the number of repro steps over time for different shortcut attempts.](chart.png)
Useful Regression Testing
Q: What scenario does a developer use to test a fix?
A: The repro scenario you provided!
The Gawain* Approach

1. Assign the same weight to each arc in a graph
2. Choose a path through the graph
3. Assign a low weight to each arc in that path
4. Exercise paths in graph in weight-increasing order

* Graph Algorithm Without An Interesting Name

© 2006 Harry Robinson, Google
Assign the same weight to each arc
Choose a path through the graph
Assign a lower weight to each arc in that path

Weight of this path = 4
Execute all paths with total weight less than some amount “X”

E.g., weight of this path = 8
Weight of this path = 8
Weight of this path = 8
Weight of this path = 9
Weight of this path = 11
You end up “Cocooning” the regression path
Machine Learning?

This developer is new.

New bug fix here.

This feature is new.
Observations on Model-Based Testing
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
Economics of Model-Based Testing

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Cpu Cost (hrs/yr)</th>
<th>Tester Cost (hrs/yr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 tester, 1 cpu</td>
<td>2080</td>
<td></td>
</tr>
<tr>
<td>2 testers, 2 cpus</td>
<td>4160</td>
<td></td>
</tr>
<tr>
<td>1 tester, 10 cpus</td>
<td>52000</td>
<td></td>
</tr>
</tbody>
</table>

© 2006 Harry Robinson, Google
Metrics Issues

Should you count bugs you prevented?

Should you count how many test cases you’ve generated?
Benefits of Model-Based Testing

- Easy test case maintenance
- Reduced costs
- More test cases
- Early bug detection
- Increased bug count
- Time savings
- Time to address bigger test issues
- Improved tester job satisfaction

© 2006 Harry Robinson, Google
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
A Useful Resource

The Model-Based Testing Home Page

www.model-based-testing.org
Recommended reading
Thank you!