# Four Exercises for Teaching Exploratory Testing

James Lyndsay, Workroom Productions Ltd.
jdl@workroom-productions.com
London, 2006

## Abstract

This paper describes four exercises designed to help teach frameworks for exploration of working software, including test design and judgement of problems.

Practical tuition in exploratory software testing needs software for students to actively explore. To allow genuine exploration, the software must be new to the students. A full sized system is often taken as the test subject for some courses, and for on-the-job experience. In contrast, these exercises make use of small systems constructed to reinforce the techniques taught.

## Terms

**Exercise**: a hands-on exploration of known software to allow experience of a framework.

**Framework**: a repeatable process to guide exploration.

**Machine**: Test subject for an exercise. In these exercises, a Flash 'movie' file that can be executed with appropriate player software.

## 1. Background

Received wisdom says that most commercial testers make use of unscripted techniques. Their unscripted approaches are most often undisciplined, and hidden from individuals outside the immediate team. This understanding matches my own experience – and I have also found that an individual tester typically makes use of a single style of unscripted testing, or focuses on a single type of target. This second characteristic can potentially be addressed with explicit exposure to a wider range of exploratory techniques.

In '*An Introduction to General Systems Thinking*' [1], Weinberg describes some abstract machines. Their purpose is initially unclear, but can be deduced. An investigator makes observations, some based on stimulus. The observations are analysed, and a hypothesis reached and tested. Different observers reach different hypotheses.

The exercises below are inspired in part by Weinberg's machines. In the exercises, observation leads to a model, which can be tested by further stimulus.

These exercises have been used in public and private commercial training since mid 2002. Aside from their immediate use in training testers to explore systems in a variety of ways, the exercises have proved particularly helpful in generating discussion about fundamental ideas of analysis. The teaching notes, which were developed two years after the machines, reflect some of these discussions.

The exercises have been used by third parties as part of candidate assessment. It should be noted that the exercises have no 'right answer'.

## 2. Typical use

The exercises have been used in public and private commercial classes of one or two days duration. Class sizes are limited to twelve students. Students work singly, or in pairs – so there should be at least half as many computers as students. Most students are experienced testers, with a year or more hands-on testing in their recent past.

The exercises and associated machines have also been used in larger and shorter classes, as demonstrations, and with non-testers.

## 3. Exercises

### 3.1. In / Out (Machine A)

This exercise introduces a framework that helps in building a model of the subject. Students are asked to study the machine and list its inputs and outputs. Once this is complete, the students are asked to study and describe the possible linkage between the recognised inputs and outputs.

There is no clear space in the framework to list the functionality of the machine. Students who concentrate on the functionality should be guided to stay within the framework: it is designed in part to encourage students to work in a potentially novel way, and to feel the discipline in controlled exploration.

'Input' and 'output' are not defined. This allows the exercise to lead to an important class discussion of what might characterise an input or an output. This can in turn lead to a broader list of possible inputs and outputs.

Please see the attached teaching notes for details and further information.

The model developed is used in later exercises (against Machine D).

The learning objectives of this exercise are:

- Experience of a basic modelling framework
- Experience of a disciplined approach
- Improved understanding of inputs and outputs
- Stimulus to imagination (possible inputs /outputs) to help improve analysis

### 3.2. Event / Behaviour (Machine B)

This exercise introduces a second framework that helps in building a different model of the subject. Students are asked to identify any events and their effects, and to list and group the subject's behaviours.

Once again, students who concentrate on 'what the machine is doing' without placing their enquiries or statements within the context of the framework should

be guided to restate their information or re-direct their exploration.

The machine stops working after one inevitable event. While students can affect when this happens, they cannot avoid it – or trigger it – directly. This encourages students to move away from the mistaken understanding that they are in control of the machine. Once stopped, the machine cannot be reset without reloading the file. This encourages students to think of test activities outside the functionality of the machine.

'Event' and 'behaviour' are not defined. This allows the exercise to lead to an important class discussion of what might characterise an event or a behaviour. This can in turn lead to an understanding of states and state transitions.

This machine has an intentional bug, typically seen in by one or two people in the class, early in testing. It is audible to all, but not initially easy to reproduce. Drawing a state diagram to model the system helps make the reproduction simpler, providing a positive learning experience.

Please see the attached teaching notes for details and further information.

The learning objectives of this exercise are:

- Experience of a second basic modelling framework
- Experience of an event that is not under tester control
- Improved understanding of states
- Improved understanding of use of modelling and analysis techniques in exploratory testing

Framework based in part on 'active play' ideas from Hendrickson [2].

## 3.3. Judgement: Cultural expectations (Machine C)

This exercise introduces a framework that helps in discovering, classifying and judging potential problems. The framework involves identifying 'inconsistencies', 'absences' and 'extras' between the test subject and other artefacts; the students may have already been introduced to this framework in an exercise that uses a physical text. In this exercise, they are asked to apply the framework to a countdown timer, in comparison with their cultural expectations.

The exercise is typically time-limited – students have a few minutes to test, while the timer has a default 20-minute countdown. The exercise can be improved for some classes by working through a short risk assessment before starting the hand-on testing. Some testers will need to be guided to explicitly consider the influence of these two contexts on their test design.

Once testing has completed, the class should take the opportunity to discuss the choices they made before and during testing. The machine is not return good results from the In/Out and Event/Behaviour frameworks within the short time available, and students will typically return to their usual approaches when testing this machine. For instance, some will attack the input, some will look for usability errors, and some will simply let their 20-minute timer count down for the duration.

There is no specification for the machine, although some students will discover a partial specification in the help text. Some students may feel unable to log any bugs, while others may choose to identify wide range of characteristics as problems. Class discussion allows the students to compare their judgement with that of their peers, and may help them to refine their understanding.

Please see the attached teaching notes for details and further information.

The learning objectives of this exercise are:

- Experience of a basic judgement framework
- Improved understanding of effect of context and experience on test design
- Stimulus to imagination (possible problems) to help improve judgement

## 3.4. Judgement: Inconsistency (Machine D)

This exercise introduces a different context for the judgement framework used in the exercise above. In this exercise, students are asked to apply the framework to differences between Machine A and Machine D. The students are asked to imagine that these machines are two different versions of the same product.

To avoid judgement based on a perception of some functionality being more desirable than another, there is no indication of error correction; the students are not told which machine might be an earlier version. The exercise can be improved for some classes by making reference in the earlier exercise to characteristics such as the different behaviour of the buttons, or the linearity of the dial's response to the slider.

Students typically take one or two of the following approaches to discover the differences:

- Using equivalent tests on both machines side-by-side
- By comparison with the model built in the first exercise
- By comparison with their memory of Machine A

Students should be encouraged to discuss the differences and relative merits of their approaches. This discussion can happen after testing is complete, but should be kept separate from discussion of the judgement of the differences found.

As in the previous exercise, different students will have different results, and it helps to have a short discussion is to allow general agreement on the differences that exist. Once these are recognised, students have clear targets to judge. Some differences are more likely to be bugs than others; students should be asked which they would log and under what circumstances, and should be encouraged to justify their judgement. In particular, attention can be focussed on further (perhaps hypothetical) tests that could add weight, or introduce alternatives, to their judgement.

Please see the attached teaching notes for details and further information.

The learning objectives of this exercise are:

- Improved understanding of effect of test technique on bug discovery
- Improved understanding of use of diagnostic tests to judge problems found.
- Improved understanding of effect of context on judgement.

# 5 Results from teaching

Feedback on the exercises during the usual 1-2 day course is generally positive. However, 10-20% of students feel at least initially uncomfortable with the incomplete definitions in the first two exercises. A smaller proportion of students are uncomfortable testing or judging bugs without a specification.

Feedback gathered at the end of each course is positive, but the effects of the exercises cannot be clearly distinguished from the rest of the course material and delivery.

No information has yet been collected about retention or use of the frameworks after the course, or the effects of this training on the effectiveness of testers.

# 6 Further progress

Exercises and frameworks for mapping, attacking and the effects of notation on observation have been developed, but are not yet publicly available.

There is a clear need to study the effectiveness (or otherwise) of these exercises in teaching exploratory testing.

# 7 Conclusion

It is hoped that the exercises described in this paper will complement existing teaching approaches, and help expose testers to different exploratory approaches.

# 8 Acknowledgements

The author would like to acknowledge the contributions of the numerous reviewers and students who have helped refine the exercises and their teaching notes. Particular credit should go to Alan Richardson for planting the initial thought, and to Robert Sabourin and James Bach for their involvement and comments.

# 9 References

[1] G.M. Weinberg, *An Introduction to General Systems Thinking*, (Dorset House, 2001), Ch. 4, 87-101

[2] E. Hendrickson, *Discovery Zone: A guide to Software Bug Habitats*, proceedings of STAREast 02

[3] Worldwide Ubiquity of Macromedia Flash by Version: *http://www.macromedia.com/software/player_census/flashplayer/version_penetration.html*

# 10 Appendices

## 10.1 Choice of Technology

The machines are have been developed in Flash. Although this requires investment in a proprietary development environment, the files produced have the following advantages:

- Execution is within a 'player':
  - Reliably cross-platform
  - Code to handle I/O, exceptions, screen drawing etc. is not part of executable
  - Does not install .dlls
  - Can be run from read-only media – does not need prior installation
  - Small size
- Attractive interface encourages interaction
- Interactions are (mostly) limited by default

The machines are written to work in players of version 5 or later (version released in July 2000). The flash player is installed on most personal computers (penetration 97% ±2% [3]).

Explicit code is written in ActionScript; excerpted listings below.

## 10.2 Potential for failure

Problems have been observed in use. Most commonly, the machines do not run because the Flash player has been uninstalled, or is prevented from running by security software.

Older computers may show slow response, particularly low power laptops built before 2000. It was sometimes possible to observe a progressive slowdown on these low-power computers. This issue was traced to a coding error in the Flash machines; releasing a slider did not properly relinquish control. This was thought to have triggered a memory leak under the Flash 5 player. The error has been addressed in the current machines.

It is possible to write Flash files which stress or exploit the player application, and so can trigger problems in the browser and the operating system. The machines for these four exercises are designed to avoid these hazards. One browser crash has been observed, while using a machine that shipped with the error referred to above.

## 10.3 Machine Architecture

The machines are all built on the same architecture.

- Flash files are movies. Each movie has a frame rate; typically 20fps for the machines used in these exercises. Once initialised, the movie loops on the same frame, running the main subroutine 20 times a second.
- Buttons, dials etc. are objects. These objects have variable linked to some aspect of their appearance; user interaction with the appearance of the object can change the variable, and the variable can change the appearance of the object.

- The main subroutine collects variables from on-screen objects are into an array. The array is transformed, and some on-screen objects receive new values for their variables, changing their appearance.

The mouse-driven interface means that only one point on-screen can be interacted with at a time. This limitation is important to some exercises, but can be avoided by introducing objects that react to key presses and other events.

All machines built on this architecture have by default a session timer, help text and a commercial / licensing message. Machines are 'published' to run on the Flash 5 player or better.

Although Machine C is constructed within the same working architecture, the working code is distributed between a dozen or more interlinked objects. This code has not been included here.

## 10.4 Code

### Shared engine

```
onClipEvent (enterFrame) {
	put_output_info(translate(get_input_info()))
}
```

### Machine A

```
function translate(input) {
	output = new Array(12);
	output[0] = input[0];
	output[1] = input[1];
	output[2] = input[2];
	output[3] = input[3];
	return output;
}
//
function get_input_info() {
	input = array(12);
	input[0] = inp_mov_01.var1;
	input[1] = inp_mov_02.var1;
	input[2] = inp_mov_03.var1;
	input[3] = inp_mov_04.var1;
	input[4] = inp_mov_05.var1;
	input[5] = inp_mov_06.var1;
	return input;
}
//
function put_output_info() {
	out_mov_01.var1 = output[0];
	out_mov_02.var1 = output[1];
	out_mov_03.var1 = output[2];
	out_mov_04.var1 = output[3];
}
```

### Machine B

```
function translate(input) {
	output = new Array(12);
	output[0] = input[0];
	output[1] = input[1];
	output[2] = input[2];
	if ((input[4] <> 1) && (input[5] <> 1))
	{
		output[0] = input[0] + 1;
		if (input[3] <> 0)
		{
			output[1] = input[1] + 1;
		}else{
			output[2] = input[2] + 1;
		}
	}
	if ( (input[6] == 1) && (input[7] == 1) )
	{
		output[3] = 1;
	}
	return output;
}
//
function get_input_info() {
	input = array(12);
	input[0] = out_mov_01.var1;
	input[1] = out_mov_02.var1;
	input[2] = out_mov_03.var1;
	input[3] = inp_mov_01.var1;
	input[4] = out_mov_04.var1;
	input[5] = out_mov_05.var1;
	input[6] = out_mov_02.var2;
	input[7] = out_mov_03.var2;
	return input;
}
//
function put_output_info() {
	out_mov_01.var1 = output[0];
	out_mov_01.start_at = 0;
	out_mov_01.range = 120;
	out_mov_02.var1 = output[1];
	out_mov_02.start_at = 0;
	out_mov_02.range = 100;
	out_mov_03.var1 = output[2];
	out_mov_03.start_at = 0;
	out_mov_03.range = 100;
	out_mov_06.var1 = output[3];
	if (out_mov_02.var1 > out_mov_02.range)
	{
		out_mov_04.var1 = 1;
		out_mov_07.var1 = 1;
	}

	if (out_mov_03.var1 > out_mov_03.range)
	{
		out_mov_05.var1 = 1;
		out_mov_07.var1 = 1;
	}
}
```

### Machine D

```
function translate(input) {
	output = new Array(12);
	if ((input[0] == 1) && (input[2] <>1))
	{
		output[0] = 1;
	}else{
		output[0] = 0;
	}
	output[1] = input[1];
	output[2] = input[2];
	output[3] = (input[3] * input[3])/100;
	output[4] = input[3];
		return output;
	}
//
function get_input_info() {
	input = array(12);
	input[0] = inp_mov_01.var1;
	input[1] = inp_mov_02.var1;
	input[2] = inp_mov_03.var1;
	input[3] = inp_mov_04.var1;
	input[4] = inp_mov_05.var1;
	input[5] = inp_mov_06.var1;
	return input;
}
//
function put_output_info() {
	out_mov_01.var1 = output[0];
	out_mov_02.var1 = output[1];
	out_mov_03.var1 = output[2];
	out_mov_04.var1 = output[3];
	out_mov_05.var1 = output[4];
}
```

## 10.5 Rights



## 10.6 Teaching notes

Attached below, from pre-prepared .pdf.

# Getting a Grip on Exploratory Testing: Exercises

Many competent testers use one or two exploratory approaches, but are not comfortable working outside that range.

In the course *Getting a Grip on Exploratory Testing*, I teach a variety of exploratory techniques. By exposing participants to a range of techniques and disciplines, I have tried to put testers in a position where they gain an appreciation of their own style, and of the range of options available.

To teach the techniques, I have developed an number of interactive machines. Each of these machines has been designed primarily to help teach a specific technique – although other trainers may find different uses.

I have decided to make the machines available through testingeducation.org. This document is part of that distribution, and contains teaching notes for each machine. Exploration is a process of learning, so if you have received this document as part of a class in exploratory testing, you need to know that you will get much more from the exercises if you put this document aside.

The notes for each machine contain a brief description of the exercise I use to teach a technique or discipline, and an example of work that a keen student might produce. Separate sections cover what you should know about the deeper structures of the machine, and what you might want to look out for while teaching to assess the progress of a class.

If you still plan to read the teaching notes without doing the exercises, I have to assume that you are the kind of person who does yesterdays crossword by looking at the answers in today's paper.



My thanks to the people – particularly Alan Richardson, James Bach and Robert Sabourin – who have encouraged me to develop these exercises, and to the colleagues and course participants who have taught me how to teach them.

## Technology

The machines have been developed in Flash 5, allowing:

Compatibility across browser, OS and platform.
Use without installation - can be run from CD or network
Small size

## Using the exercises

Open index.html in a browser
Use the test machine in the support section to check that you have the flash plugin.
Follow the links for the four exercise machines A-D

## Role of the trainer / coach

You need to present the technique or discipline, coach participants as they test during the exercise, and facilitate discussion after the exercise.

Different exercises puzzle different people. You have to pay close attention and decide whether to intervene, or let to let a participant arrive at a solution themselves.

## Contact

| | |
|---|---|
| | James Lyndsay |
| Company: | Workroom Productions Ltd. |
| Website: | http://www.workroom-productions.com |
| Email: | jdl@workroom-productions.com |
| AIM: | workroomprds |
| Tel: | +44 (0) 20 7372 6986 |
| Mobile: | +44 (0) 7904 158 752 |

# Machine A: Input / Output / Linkage

Summary: Disciplined exploration of an abstract machine.
Takes: Between 10 and 20 minutes, including discussion.

Introduce idea of active, systematic exploration - and a framework to help build a simple model. Help re-consider concepts of input and output. Encourage imaginative extensions of diversity of input / output, types of dependency. Share different approaches to GUI discovery.

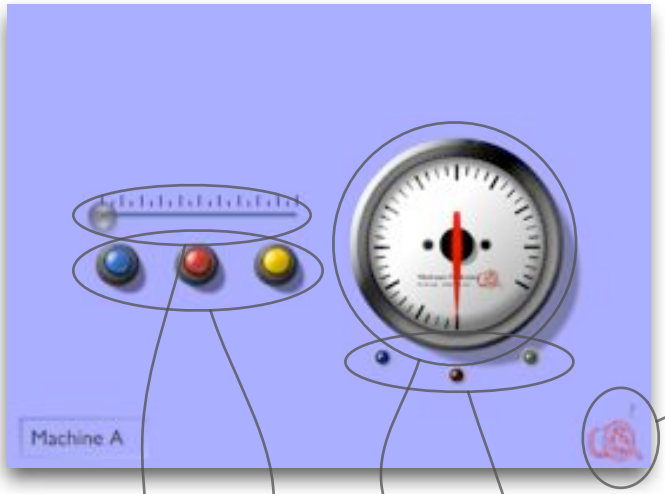
## Assess and assist group progress:

Simple ideas of 'an input is a button' should give way to more complex concepts.

On analysis, 'random' clicking will crystallise into techniques. Different individuals will have different approaches – notice these and encourage participants to try novel techniques.

Participants may not believe you about the logo / '?'. Encourage them in this!

The group may need help to model the linkage. Challenge them to increase their certainty.




## Teacher Awareness

The machine should be simple to explore. The buttons, lights, slider and dial are clear and act independently. There are no hidden tricks.

The blue button is a toggle, the red stays down as long as it is pressed, and the yellow is transient.

The lights correspond to the state of the picture button, not to the state of the mouse button.

The blue button is the only one that allows its state to be fixed for further action/testing. Its state (and by implication that of the blue light) has no effect on any other input or output.

Paired testing is highly effective

The '?' and red logo in the bottom right corner respond to rollover/mouseclick. They have no direct effect on the machine (and are a common feature). *I like to let people find these before I tell them – it helps them think of different surprises.*

There are no known bugs

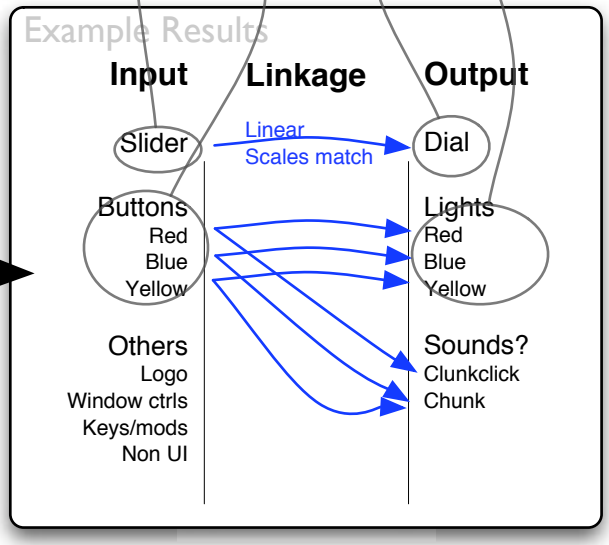
## Suggested exercise:

Identify 'inputs' and 'outputs' (3 minutes).

*Discuss what makes an input, what makes an output. What others might there be?*

Identify links and dependencies between identified input and output. (3 minutes)

*Discuss types of linkage - one-to-one, multiple dependencies, linearity*




## Extending:

*Discuss the differences between input if seen as information / stimulus, and input if seen as something that can be stimulated*

*Introduce resizing, ctrl-click, keypresses, platform etc.*

*Discuss APIs and automation.*

*Discuss discovery of input/output during exercise, and effectiveness (or not) of systematic approaches.*

# Machine B: Event / Behaviour / Information

Summary: Disciplined exploration of an abstract machine.
Takes: Between 20 and 30 minutes, including discussion.

Introduce a second exploratory framework. Highlight that testers/users are not always the direct cause of an observed effect. Different behaviour / response indicates different state. Imagining underlying system – making a model modelling and testing cause / effect. Use state model to assist exploratory testing.

## Assess and assist group progress:

Once the machine stops for the first time, some delegates may think they have broken the machine. Some, perhaps feeling they have proved their prowess, will go no further. Gently ask them to reproduce the bug, and to describe the actions that they took – further investigation may lead them to question their initial judgement.

Can the group tell you about their theories about what the machine might be doing, and why it stops?

Ask the group about the lights – how is the green one lit?


## Teacher Awareness

The machine starts as soon as it is opened, and the central dial spins until the machine stops. While the machine is runnning, either the left or right dial spins – the blue button toggles between them. The machine stops when either of the outer dials reaches its clockwise maximum.

There is no 'reset' button: Once the machine stops, the user needs to take external action (i.e. reload in browser) to start the machine again

The machine has been designed to be a poor subject for input/output/linkage – and to also introduce testers to the idea that they must observe events that are not triggered by their own actions.

There is a (noisy) bug that can be observed if the blue button is pressed as one of the dials reaches the end of its travel. This bug is hard to reproduce – modelling the state transitions can help focus attention and allow it to be observed more reliably.

Note: there are two ways of leaving a state, and they have non-exclusive triggers. The bug appears when the two exits happen close together – the state model shows this potential.

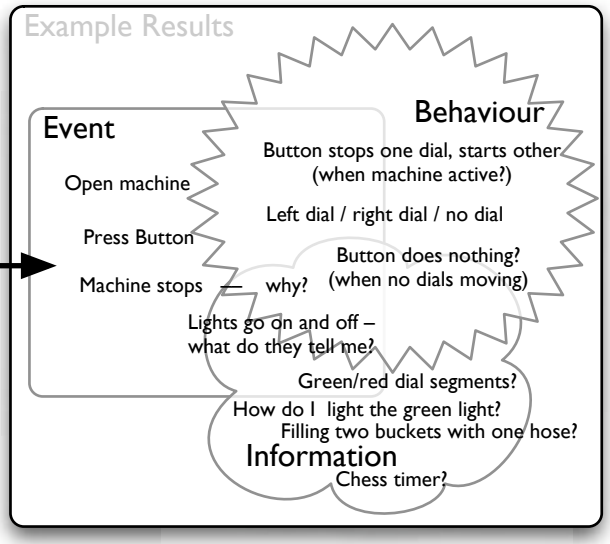
## Suggested exercise:

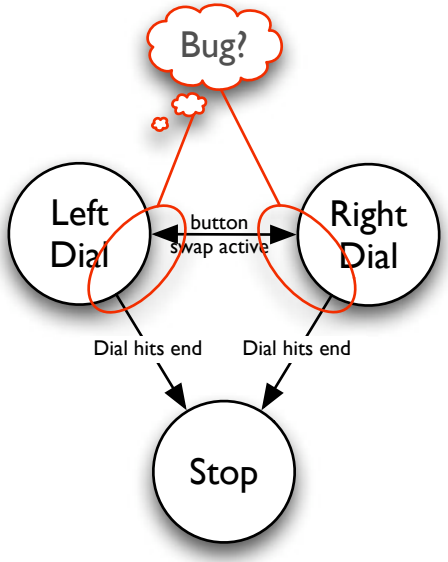
Identify the events that affect the machine, the behaviours it displays, and any information that gives you clues / seems important (6 minutes).

*Discuss events that are not triggered by testing / testers*

Use information (unanswered questions, models etc.) to imagine links between events and behaviours. (3 minutes)

*Discuss similarities between behaviour and state. Draw state diagram and hunt bugs. Discuss different state diagrams that could be used.*



# Machine C: Testing against external expectations

Summary: Disciplined exploration of something with a known function.
Takes: Between 15 and 30 minutes, including discussion.

Finding bugs in something that has an accepted way of working. Ways that planning and focussed error-guessing can help find bugs, and hinder the discovery of others. Experience of observing unexpected problems and following leads. Judging faults.

## Assess and assist group progress:

Some people may spend the entire period attacking the input. Others may not change the timer to an appropriate value for the short time available for testing. Encourage delegates to take diverse approaches, and to design tests fo fit the situation.

Judgement of a bug is key to exploration – without judgement, it is hard to consider which of many paths to follow. Testers that concentrate entirely on ambiguous characteristics may need to be challenged to find more valuable faults.


## Teacher Awareness

The logo and ? have useful information – you may want to reveal this to the class.

At this stage, delegates should be thinking of test design as well as exploration. You may want them to consider risks, likely faults, or particularly significant errors. They should also consider the constraints of the test parameters; a few minutes will not be enough to expose some issues, what can they say about coverage?

As individuals find bugs, others in the group may be distracted from their own paths (particularly if you're using pair testing with ebullient testers). Encouraging competition can motivate discovers to keep their discovery secret until the period of discovery is over.

Alternatively, you may wish to split the group into two parts – the away group thinks about risk and methods without testing, while the discovery group think about the principles that might help guide the others to discover bugs more quickly.

Changing the time on the PC Is a interesting attack...

## Suggested exercise:

Find bugs – and justifications about why the characteristics you have identified are indeed bugs (5 minutes)

*Discuss the methods used to discover the bugs – were bugs found because of carefully-aimed tests, or perceptive observation?*

*Discuss the discovery of the bugs; what bugs were found first? Did different people find different bugs?*

*Discuss the bugs themselves – are they all bugs?*

## Known bugs

The circular timer runs 10% slow [this can be seen by comparison with your watch – or by looking at the numerical timer. If you set the timer to a minute, you can see this problem in the first 15s]

'Tick/Tock' is layered in front of the logo / ? mark text . Note – 'paused' is behind.

Start/Step – should be Start/Stop [is this a typo? If 'Step' is intentional, is it still a bug]

No ambient/aural indication when timer reaches 0. [also missing from production version]

## Potential usability issues

Reset works as a button – but without the graphic. No explanation of elements – particularly input text box. Cursor appears in text box. Inconsistent response to tab. Nasty pink.

## By design

Can't reset the timer while it's going [to avoid accidental reset]. Pausing the timer doesn't stop the numerical timer [measures elapsed time]. New timer appears after timer counts to 0 [requirement to show time since timer reached 0]. Can't stop count-up timer [ask why this might be necessary].

# Machine D: Discover and judge inconsistencies

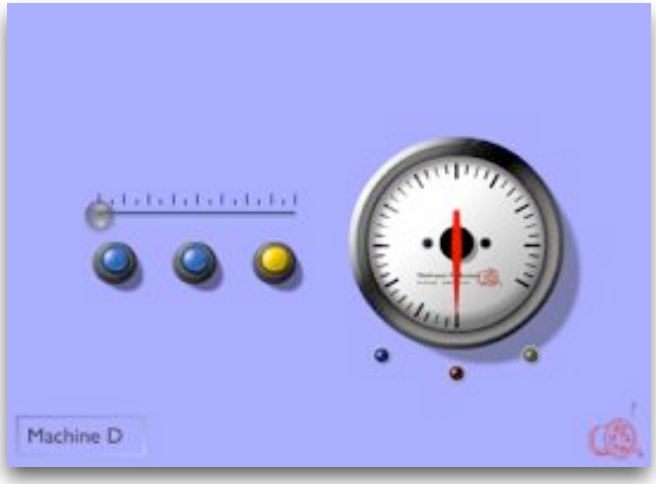
Summary: Compare two similar machines
Takes: Between 15 and 30 minutes, including discussion.

Re-use and extension of existing method. Modelling failure / difference, designing tests to verify model. Judgement of differences / bugs. Ways that repeat testing is influenced by what has gone before – use of prior test results to guide test design.

## Assess and assist group progress:

What methods are in use to explore the machine? Are people making another map of input/output/linkage? Comparing machines, or maps? Are they working from memory, or do they have their subjects open side-by-side?

In judging the differences, people will have to build models of the internal logic/connections. How are they building these models? Can they be drawn/articulated? What tests can be devised to expose the differences?




## Teacher Awareness

The class needs to know that Machine A and Machine D are different versions of the same machine. However, there is no information about which is the earlier version. You might want to discuss the influence that this information would have on judgement of bugs etc.

Non-linear response is not visible at the boundaries of slider travel. However, it's easy to see the difference in the middle, or while moving. How does this relate to BVA / ECP?

The machine is limited by possible interactions. If it had a physical interface, or if its code/ circuitry/mechanics were exposed, different tests would be available. You might want to encourage the class to think of these tests.

Having two machines open at the same time has no designed effect – but are people considering it?

Actions that didn't work on Machine A may not even be tried in Machine D – has anyone tried hitting keys, resizing the window etc.? Perhaps there are more differences? Try tab...

## Suggested exercise:

Identify differences between the machines. For each of the differences, make a judgement as to whether the difference is a bug. Justify that decision. (6 minutes)

*Discuss the methods used to discover differences.*

*Discuss whether new tests were needed to justify assessment of differences as bugs. What influenced the test design?*

*Discuss the judgements made.*

## Differences

| | |
|---|---|
| Slider scales differ | May be a bug – dial scales are the same in A and D. In A, there is a correspondence between the dial and slider - in D, that correspondence is broken. |
| D Dial has a non-linear response to slider | Unknown – there is no evidence to indicate whether this is required or consistent behaviour. |
| Middle button is red in A, blue in D | Bug. Button works like a red button, but is blue. The two blue buttons work differently in D. Is it a red button that is the wrong colour, or a blue button that works wrongly? Given the button/light colour correspondence, which remains the same for blue and yellow, it is likely to be a red button that is the wrong colour. |
| Yellow button affects blue light in D, but not in A | Unknown – not enough information to judge. Simple models of the interaction might be: a) Only one light can be on at a time; b) the yellow button inverts the blue light; c) the yellow button disconnects the blue light; d) the yellow and blue lights can't be on at the same time. (a) and (b) can be disproved – but it is not possible to manipulate the machine directly to examine (c) or (d). |