SUPPLEMENTING YOUR TESTING

CONTENTS

I. Automation Testing and Test Tools

1. The Benefits of Automation and Test Tools

1. Check that the bugs you found in previous test runs were indeed fixed and that no new bugs were introduced.
2. This process of rerunning your tests is known as regression testing.
3. If a small software project had several thousand test cases to run, there might be barely enough time to execute them just once.
4. Running them numerous times might be impossible, let alone monotonous.
5. Software test tools and automation help solve this problem by providing a better means to run your tests than by manual testing.

The principle attributes of tools and automation are

(Question: Explain the principle attributes of tools and automation in software testing. = 4 marks)

- **Speed**: Think about how long it would take you to manually try a few thousand test cases for the Windows Calculator. You might average a test case every 5 seconds or so. Automation might be able to run 10, 100, even 1000 times that fast.
- **Efficiency**: While you're busy running test cases, you can't be doing anything else. If you have a test tool that reduces the time it takes for you to run your tests, you have more time for test planning and thinking up new tests.
- **Accuracy and Precision:** After trying a few hundred cases, your attention span will wane and you'll start to make mistakes. A test tool will perform the same test and check the results perfectly, each and every time.
- **Relentlessness:** Test tools and automation never tire or give up. They're like that battery-operated bunny of the TV commercials—they can keep going and going.

II. Test Tools

(Question: Explain the various test tools of software testing? = 8 marks)

As a software tester wide range of testing tools are available. For using the test tools, don't always need to be an expert in how they work or exactly what they do to use them.

1. Viewers and Monitors

1. A viewer or monitor test tool allows you to see details of the software's operation that you wouldn't normally be able to see.
2. A code coverage analyzer is an example of a viewing tool. Most code coverage analyzers are invasive tools because they need to be compiled and linked into the program to access the information they provide.
3. The tool communication analyzer allows seeing the raw protocol data moving across a network or other communications cable.
4. It simply taps into the line, pulls off the data as it passes by, and displays it on another computer.

5. If testing such a system, the tester could enter a test case on Computer #1, confirm that the resulting communications data is correct on Computer #3, and check that the appropriate results occurred on Computer #2.
6. You might also use this system to investigate why a bug occurs.
7. By looking at the data pulled off the wire, you could determine if the problem lies in creating the data (Computer #1) or interpreting the data (Computer #2).
8. This type of system is non-invasive to the software as shown in Figure 1.

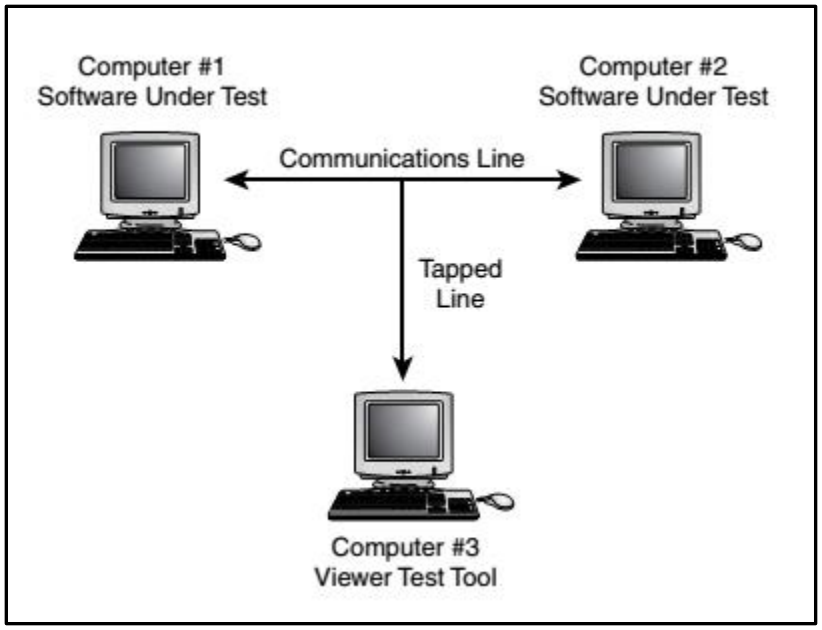

Figure 1

2. Drivers

1. Drivers are tools used to control and operate the software being tested.
2. One of the simplest examples of a driver is a batch file, a simple list of programs or commands that are executed sequentially. In the days of MS-DOS, this was a popular means for testers to execute their test programs.
3. They'd create a batch file containing the names of their test programs, start the batch running, and go home.
4. With today's operating systems and programming languages, there are much more sophisticated methods for executing test programs.
5. For example, a complex Perl script can take the place of an old MS-DOS batch file, and the Windows Task Scheduler can execute various test programs at certain times throughout the day as shown in Figure 2.

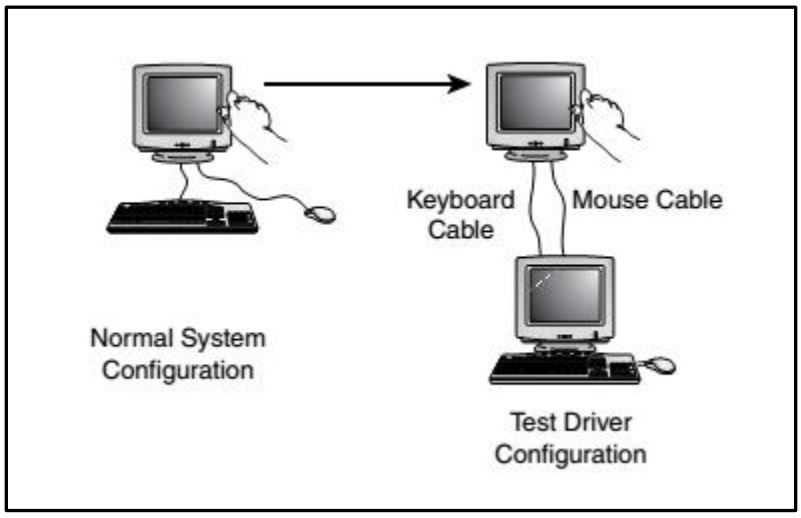

Figure 2

3. Stubs

1. Stubs, like drivers, are white-box testing techniques.
2. Stubs are essentially the opposite of drivers in that they don't control or operate the software being tested; they instead receive or respond to data that the software.
3. The Figure 3 show the general view of stub configuration.

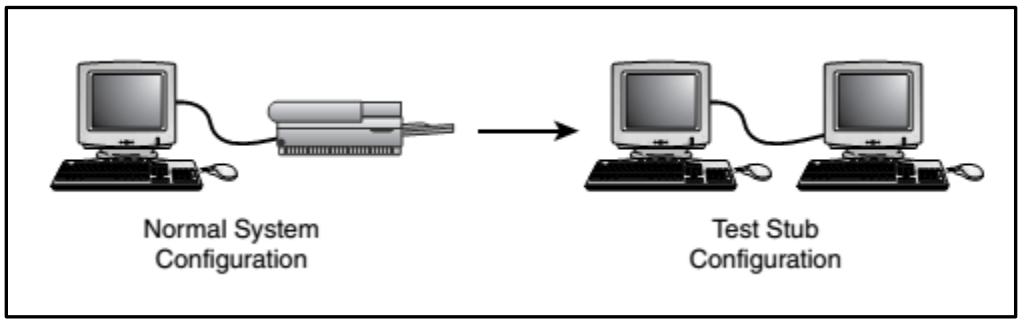

Figure 3

4. Stress and Load Tools

1. Stress and load tools induce stresses and loads to the software being tested.
2. A word processor running as the only application on the system, with all available memory and disk space, probably works just fine.
3. But, if the system runs low on these resources, you'd expect a greater potential for bugs.
4. You could copy files to fill up the disk, run lots of programs to eat up memory, and so on, but these methods are inefficient and non-exact.
5. A stress tool specifically designed for this would make testing much easier.

5. Interference Injectors and Noise Generators

1. Another class of tools is interference inject or sand noise generators.
2. They're similar to stress and load tools but are more random in what they do.
3. The Stress tool, for example, has an executor mode that randomly changes the available resources.
4. A program might run fine with lots of memory and might handle low memory situations, but it could have problems if the amount of available memory is constantly changing.
5. The executor mode of stress would uncover these types of bugs as shown in Figure 4.

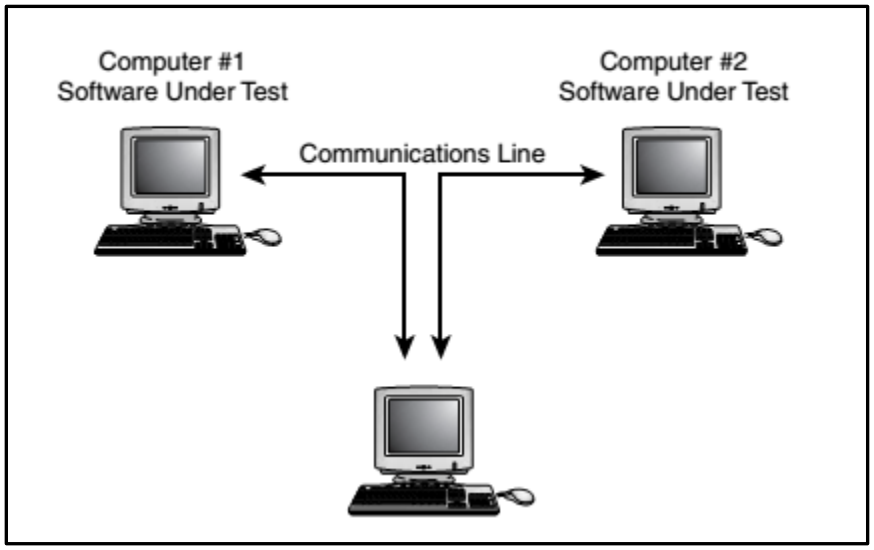

Figure 4

6. Analysis Tools

1. You might call this last category of tools analysis tools,a best-of-the-rest group.
2. Most software testers use the following common tools to make their everyday jobs easier.
3. They're not necessarily as fancy as the tools discussed so far.
4. They're often taken for granted, but they get the job done and can save you a great deal of time.
   - Word processing software
   - Spreadsheet software
   - Database software
   - File comparison software
   - Screen capture and comparison software
   - Debugger
   - Binary-hex calculator
   - Stopwatch
   - VCR or camera

III. Software Test Automation

Test automation is just another class of software testing tools, it's one that deserves special consideration. The software test tools that you've learned about so far are indeed effective, but they still must be operated or monitored manually.

1. Macro Reading and Playback

(Question: Explain the Macro reading and playback in software test automation = 8 marks)

1. The most basic type of test automation is recording your keyboard and mouse actions as you run your tests for the first time and then playing them back when you need to run them again.
2. If the software you're testing is for Windows or the Mac, recording and playing back macros is a fairly easy process.
3. Macro recorders and players are a type of driver tool. As mentioned earlier, drivers are tools used to control and operate the software being tested. With a macro program you're doing just that—the macros you record are played back, repeating the actions that you performed to test the software.

The Macro Setup Wizard allows you to set the following options for your macros:

- **Name**: Giving the macro a name provides a way to identify it later. Even for a small software project you could write hundreds of macros.
- **Repetitions:** Repetition testing is a great way to find bugs. You can set the number of times the macro will repeat or loop when it runs.
- **Triggers:** You can set how the macro is started. This can be by a hot key (for example, Ctrl+Shift+T), by a set of typed-in characters (maybe run macro 1), by clicking a shortcut, when a certain window is displayed (whenever Calculator is started, for example), or when the system has idled unused for a certain length of time.
- **What's captured:** You can select to capture (record) just keystrokes or both keystrokes and mouse actions such as moving and clicking.
- **Playback speed:** The macro can play back from up to 20 percent slower to 500 percent faster than how you originally recorded it. This is important if your software's performance can vary. What would happen if the software you're testing became a bit slower and the button the macro was to click on wasn't yet onscreen?
- **Playback position:** This option determines if the mouse movements and clicks should be absolute or relative to a certain window onscreen. If you're testing an application that might change onscreen positions, making your movements relative to that application is a good idea; otherwise, the mouse may not click where you would expect.

2. Programmed Macros

1. Programmed macros are a step up in evolution from the simple record and playback variety.

2. Rather than create programmed macros by recording your actions as you run the test for the first time, create them by programming simple instructions for the playback system to follow.

3. Fully Programmable Automated Testing Tools

1. Automated testing tools such as Visual Test provide the means for software testers to create very powerful tests.
2. Many are based on the BASIC programming language, making it very easy for even non-programmers to write test code.
3. If you wanted to try typing the string Hello World!10,000 times, you'd write a few lines of code such as this:

```
FOR i=1 TO 10000
PLAY "Hello World!"
NEXT I
```

4. If you wanted to move your mouse pointer from the upper left of your 640×480 screen to the lower right and then double-click, you could do it like this:

```
PLAY "{MOVETO 0,0}"
PLAY "{MOVETO 640,480}"
PLAY "{DBLCLICK}"
```

5. The most important feature that comes with these automation tools is the ability to perform verification, actually checking that the software is doing what's expected.
6. There are several ways to do this:
   - **Screen captures:** The first time you run your automated tests, you could capture and save screen images at key points that you know are correct. On future test runs, your automation could then compare the saved screens with the current screens. If they're different, something unexpected happened and the automation could flag it as a bug.
   - **Control values:** Rather than capture screens, you could check the value of individual controls in the software's window. If you're testing Calculator, your automation could read the value out of the display field and compare it with what you expected. You could also determine if a button was pressed or a check box was selected. Automation tools provide the means to easily do this within your test program.
   - **File and other output:** Similarly, if your program saves data to a file—for example, a word processor—your automation could read it back after creating it and compare it to a known good file. The same techniques would apply if the software being tested sent data over a modem or a network. The automation could be configured to read the data back in and compare it with the data that it expects.

IV. Random Testing:

(Question: Explain the concept of random testing. 1 mark each = 4marks)

1. Monkeys and Gorillas

1. Using tools and automation will help find bugs; while the tools are busy doing regression testing, tester will have more time to plan new tests and design new and interesting cases to try.
2. Another type of automated testing, though, isn't designed to help run or automatically run test cases. Its goal is to simulate what your users might do.
3. That type of automation tool is called a test monkey.

2. Dumb Monkeys

1. The easiest and most straightforward type of test monkey is a dumb monkey.
2. A dumb monkey doesn't know anything about the software being tested; it just clicks or types randomly.
3. Listing shows an example of Visual Test code that will randomly click and type 10,000 times.

```
1: RANDOMIZE TIMER
2: FOR i=1 TO 10000
3: PLAY "{CLICK "+STR$(INT(RND*640))+", "+STR$(INT(RND*480))+" }"
4: PLAY CHR$(RND*256)
5: NEXT i
```

4. Line 1 initializes the random numbers. Line 2 starts looping from 1 to 10,000 times.
5. Line 3 selects a random point onscreen between 0,0 and 640,480 (VGA resolution) and clicks it.
6. Line 4 picks a random character between 0 and 255 and types it in.

3. Semi-Smart Monkeys

1. Dumb monkeys can be extremely effective.
2. They're easy to write and can find serious, crashing bugs.
3. They lack a few important features, though, that would make them even more effective.
4. Adding these features raises your monkey's IQ a bit, making him semi-smart.
5. The solution is to add logging to your monkey so that everything it does is recorded to a file.
6. When the monkey finds a bug, you need only to look at the log file to see what it was doing before the failure.

4. Smart Monkeys

1. Moving up on the evolutionary scale is the smart monkey.
2. A monkey takes the effectiveness of random testing from his less-intelligent brothers and adds to that an awareness of his surroundings.
3. He doesn't just pound on the keyboard randomly—he pounds on it with a purpose.
4. A true smart monkey knows
   - Where he is
   - What he can do there

- Where he can go
- Where he's been
- If what he's seeing is correct

5. Realities of Using Test Tools and Automation Beta Testing

1. The software changes. Specifications are never fixed. New features are added late. The product name can change at the last minute. What if you recorded thousands of macros to run all your tests and a week before the product was to be released, the software was changed to display an extra screen when it started up? All of your recorded macros would fail to run because they wouldn't know the extra screen was there. You need to write your automation so that it's flexible and can easily and quickly be changed if necessary.
2. There's no substitute for the human eye and intuition. Smart monkeys can be programmed to be only so smart. They can test only what you tell them to test. They can never see something and say, "Gee, that looks funny. I should do some more checking"—at least, not yet.
3. Verification is hard to do. If you're testing a user interface, the obvious and simplest method to verify your test results is capturing and comparing screens. But, captured screens are huge files and those screens can be constantly changing during the product's development. Make sure that your tools check only what they need to and can efficiently handle changes during product development.
4. It's easy to rely on automation too much. Don't ever assume that because all your automation runs without finding a bug that there are no more bugs to find. They're still in there. It's the pesticide paradox.
5. Don't spend so much time working on tools and automation that you fail to test the software. It's easy and fun to start writing macros or programming a smart monkey, but that's not testing. These tools may help you be more efficient, but you'll need to use them on the software and do some real testing to find bugs.
6. If you're writing macros, developing a tool, or programming a monkey, you're doing development work. You should follow the same standards and guidelines that you ask of your programmers. Just because you're a tester doesn't mean you can break the rules.
7. Some tools are invasive and can cause the software being tested to improperly fail. If you use a tool that finds a bug, try to re-create that bug by hand without using the tool. It might turn out to be a simple reproducible bug, or the tool might be the cause of the problem