UNIT III

EMBEDDED PROGRAMMING

PROGRAM VALIDATION AND TESTING

- ❖ Complex systems need testing to ensure that they work as they are intended. But bugs can be subtle, particularly in embedded systems, where specialized hardware and real-time responsiveness make programming more challenging.
- ❖ Fortunately, there are many available techniques for software testing that can help us generate a comprehensive set of tests to ensure that our system works properly.
- ❖ In this section, we concentrate on nuts-and-bolts techniques for creating a good set of tests for a given program. The first question we must ask ourselves is how much testing is enough.
- ❖ Clearly, we cannot test the program for every possible combination of inputs. Because we cannot implement an infinite number of tests, we naturally ask ourselves what a reasonable standard of thoroughness is. One of the major contributions of software testing is to provide us with standards ofthoroughness that make sense.
- ❖ Following these standards does not guarantee that we will find all bugs. But by breaking the testing problem into sub problems and analyzing each sub problem.

- we can identify testing methods that provide reasonable amounts of testing while keeping the testing time within reasonable bounds. The two major types of testing strategies:

  Black-box methods generate tests without looking at the internal structure of the program. **Clear-box** (also known as **white-box**) methods generate tests based on the program structure.

In this section we cover both types of tests, which complement each other by exercising programs in very different ways

Clear-Box Testing

- The control/data flowgraph extracted from a program's source code is an important tool in developing clear-box tests for the program. To adequately test the program, we must exercise both its control and data operations.
- In order to execute and evaluate these tests ,we must be able to control variables in the program and observe the results of computations, much as in manufacturing testing. In general,we may need to modify the program to make it more testable. By adding new inputs and outputs, we can usually substantially reduce the effort required to find and execute the test. Example 5.11 illustrates the importance of observability

Example 5.11 illustrates the importance of observability and controllability in software testing. No matter what we are testing, we mustaccomplish the following three things in a test:

- Provide the program with inputs that exercise the test we are interested in.
- Execute the program to perform the test.
- Examine the outputs to determine whether the test was successful.

Being able to perform this process for a large number of tests entails some amount of drudgery, but that drudgery can be alleviated with good program design that simplifies controllability and observability. The next task is to determine the set of tests to be performed.We need to perform many different types of tests to be confident that we have identified a large fraction of the existing bugs.

Black-Box Testing

Black-box tests are generated without knowledge of the code being tested. When used alone black-box tests have a low probability of finding all the bugs in a program. But when used in conjunction with clear-box tests they help provide a well-rounded test set, since black-box tests are likely to uncover errors that are unlikely to be found by tests extracted from the code structure.

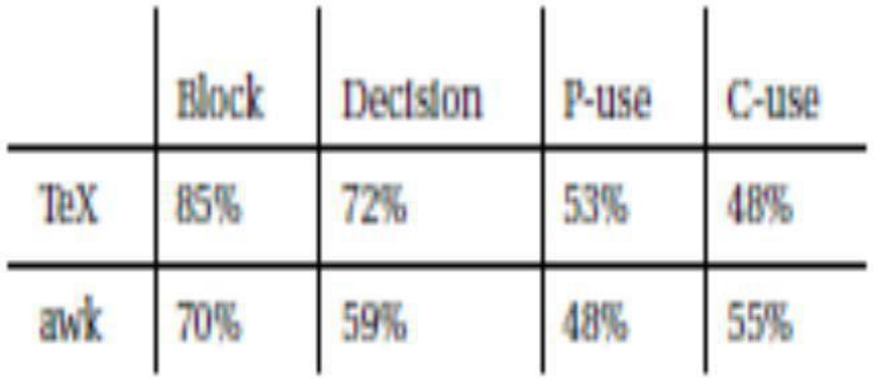
- Black-box tests can really work. For instance, when asked to test an instrument whose front panel was run by a microcontroller, one acquaintance of the author used his hand to depress all the buttons simultaneously.
- The front panel immediately locked up. This situation could occur in practice if the instrument were placed face-down on a table, but discovery of this bug would be very unlikely via clear-box tests. One important technique is to take tests directly from the specification for the code under design.
- The specification should state which outputs are expected for certain inputs. Tests should be created that provide specified outputs and evaluate whether the results also satisfy the inputs.
- We can’t test every possible input combination, but some rules of thumb help us select reasonable sets of inputs. When an input can range across a set of values, it is a very good idea to test at the ends of the range. For example, if an input must be between 1 and 10, 0, 1, 10, and 11 are all important values to test.
- We should be sure to consider tests both within and outside the range, such as, testing values within the range and outside the range. We may want to consider tests well outside the valid range as well as boundary- condition tests.

- **Random tests** form one category of black-box test. Random values are generated with a given distribution. The expected values are computed independently of the system, and then the test inputs are applied. A large number of tests must be applied for the results to be statistically significant, but the tests are easy to generate. Another scenario is to test certain types of data values. For example, integer valued inputs can be generated at interesting values such as 0,1,and values near the maximum end of the data range. Illegal values can be testedas well.
- **Regression tests** form an extremely important category of tests. When tests are created during earlier stages in the system design or for previous versions of the system, those tests should be saved to apply to the later versions of the system.
- Clearly, unless the system specification changed, the new system should be able to pass old tests. In some cases old bugs can creep back into systems, such as when an old version of a software module is inadvertently installed. In other cases regression tests simply exercise the code in different ways than would be done for the current version of the code and therefore possibly exercise different bugs. Some embedded systems, particularly digital signal processing systems, lend themselves to numerical analysis. Signal processing algorithms are frequently implemented with limited-range arithmetic to save hardware costs. Aggressive data sets can be generated to stress the numerical accuracy of the system. These tests can often be generated from the original formulas without reference to the source code

Evaluating Function Tests

- How much testing is enough? Horgan and Mathur [Hor96] evaluated the coverage of two well- known programs, TeX and awk. They used functional tests for these programs that had been developed over several years of extensive testing. Upon applying those functional tests to the programs, they obtained the code coverage statistics.
- The columns refer to various types of test coverage: block refers to basic blocks, decision to conditionals, p-use to a use of a variable in a predicate (decision), and c-use to variable use in a non predicate computation. These results are at least suggestive that functional testing does not fully exercise the code and that techniques that explicitly generate tests for various pieces of code are necessary to obtain adequate levels of code coverage.
- Methodological techniques are important for understanding the quality of your tests. For example, if you keep track of the number of bugs tested each day, the data you collect over time should show you some trends on the number of errors per page of code to expect on the average, how many bugs are caught by certain kinds of tests, and so on. We address methodological approaches to quality control in more detail. One interesting method for analyzing the coverage of your tests is **error injection**.

- ❖ First, take your existing code and add bugs to it, keeping track of where the bugs were added. Then run your existing tests on the modified program. By counting the number of added bugs your tests found, you can get an idea of how effective the tests are in uncovering the bugs you haven't yet found. This method assumes that you can deliberately inject bugs that are of similar varieties to those created naturally by programming errors.
- ❖ If the bugs are too easy or too difficult to find or simply require different types of tests, then bug injection's results will not be relevant. Of course, it is essential that you finally use the correct code, not the code with added bugs.



	Block	Decision	P-use	C-use
TeX	85%	72%	53%	48%
awk	70%	59%	48%	55%

Platform level performance Analysis

Bus based systems add another layer of complication to performance analysis platform level performance involve much more than the CPU we often focus on the CPU because it processes instructions but any part of the system can affect total system performance. More precisely the CPU provides an upper bound on performance but any other part of the system can slow down the CPU merely counting instruction execution times is not enough

Consider the simple system we want to move data from memory to the CPU to process it to get the data from memory to the CPU we must

- Read from the memory
- Transfer over the bus to the cache
- Transfer from the cache to the CPU

The time required to transfer from the cache to the CPU is included in the instruction execution time , but the other two times are not .

Bandwidth as performance

❖ The most basic measure of performance we are interested in is bandwidth the rate at which we can move data ultimately if we are interested in real time performance we are interested in real time performance measured in seconds but often the simplest way to measure performance is in units of clock cycles however different parts of the system will run at different clock rates. We have to make sure that we apply the right clock rate to each part of the performance estimate when we convert from clock cycles to seconds

Bus bandwidth

❖ Bandwidth questions often come up when we are transferring large blocks of data for simplicity let's start by considering the bandwidth provided by only one system component the bus consider and image of 320 pixels with each pixel composed of 3 bytes of data this gives a grand total of 230 400 bytes of data if these images are video frames, we want to check if we can push one frame through the systemwithin the 1/30 sec that we have to process a frame before the next one arrives.

- ❖ Let us assume that we can transfer one byte of data every microsecond which implies a bus speed of 1 Mhz. in this case we would require 230400 us =0.23 sec to transfer one frame that is more than the 0.033 sec allotted to the data transfer we would have to increase the transfer rate by 7xto satisfy our performance requirement
- ❖ We can increase bandwidth in two ways we can increase the clock rate of the bus or we can increase the amount of data transferred per clock cycle for example if we increased the bus to carry four bytes or 32 bits per transfer we would reduce the transfer time to 0.058 sec if we could also increase the bus clock rate to 2 Mhz. then we would reduce the transfer time to 0.029sec , which is within our time budget for the transfer

THANKYOU