**5. COVERAGE AND USAGE TESTING**

**BASED ON**

**CHECKLISTS AND PARTITIONS**

**5.1 CHECKLIST-BASED TESTING AND ITS LIMITATIONS**

The simplest form of testing is to start running the software and make observations, in the hope that it is easy to distinguish between expected and unexpected behavior. Along the same line, software systems are sometimes tested in a similar way to see if some specific problems can be observed or if specific operational condition and input can be handled without resorting to some systematic method. We call these forms of simple and informal testing *ad hoc* testing. Ad hoc testing is also called *random* testing in some literature. However, we will avoid this use of the term random testing because of the possible confusion between it and usage-based statistical testing that is random testing based on specific operational profiles or distributions of likely operations or operational sequences.

When ad hoc testing is used repeatedly to test a software system, the testers then need to keep track of what has been done, in order to avoid wasting their time repeating the same tests. In addition, an informal “to-do” list is commonly used to track what needs to be done. Such to-do lists can be a physical list, an online list, or just a mental list. The use of these informal lists forms the rudimentary and implicit ***checklists,*** where each item can be checked off when corresponding testing was performed, until every item on the lists is checked off.

**Testing with checklists**

The idea of checklists can be and has been generalized to perform systematic testing for many software systems (DeMillo et al., 1987; Kaner et al., 1999; Binder, 2000). For example, a specification checklist, or a checklist based on product specifications with each major specification item as a checklist item, can be used to perform black-box testing. Similarly, checklists of expected programming features that are supposed to be implemented in a software product, or coding standards that are supposed to be followed in implementation, are examples of white-box checklists, which can be used to support various types of white-box testing. In fact, the commonly used testing strategy of statement coverage in unit and component testing, or component coverage in integration and system testing, is also white-box checklist based testing, where each element in the checklists corresponds to a specific statement or a specific component, respectively. Usage-based statistical testing can also be supported by a special form of checklists called operational profiles (OPs), in which each item is associated with an operation to be performed together with its probability of usage.

In using these checklists, a specific testing based on a specific checklist can stop when every item on it has been tested (or “checked off”). By been “tested” or “checked off’, we mean that the corresponding test case has been executed, and follow-up activities, such as fixing discovered problems, have been carried out and completed, which may also include rerunning the test case to verify that the problems have indeed been fixed. Some commonly used checklists for black-box or white-box coverage testing are listed below:

* Functional (black-box) checklists at different levels of abstraction and granularity, ranging from major high-level functions for the overall system to specific low-level functions of an individual unit or components.
* Checklists of system elements (white-box) at different levels of granularity, ranging from sub-systems and modules at the high level to individual statements or data items at the low level.
* Checklists of system elements (white-box) at different levels of granularity, ranging from sub-systems and modules at the high level to individual statements or data items at the low level.

**Table: A** high-level functional checklist for some relational database products

* abnormal termination
* backup and restore
* comnnunication
* co-existence
* file I/O
* gateway
* index. management
* installation
* logging and recovery
* locking
* migration
* stress
* Checklists about certain properties, such as coding standard, specific specification items, etc., which can be either black-box or white-box.

Other basic types of checklists are also possible. **A** common way to obtain usable checklists is to select items from several exhaustive checklists based on some criteria and to combine them. Many checklists can also be used together, to form some linked set of things to check during testing, as discussed below.

The above table gives a sample high-level checklist for some relational database products (Tian et al., 1995; Tian, 1998). In fact, each item corresponds to a specific high-level functional area or aspect important to the products as perceived by their users, which can be and was further refined into sub-areas using other checklists.

**From basic checklists to hierarchical and combined checklists**

The most commonly used form of multiple checklists is the hierarchical checklist, with each item in a higher-level checklist expandable to a full checklist at a lower level until we stop at a level of detail deemed enough by some criteria. In the checklist in Table, each high-level functional area represented by an individual item can be divided into sub-areas, and each sub-areas can be divided further. In fact, they form the hierarchical test suite associated checklists form a set of hierarchical checklists that can be used as the basis for coverage based testing for these large software systems.

In addition to the use of individual checklists and hierarchical sets of checklists above, various checklists can also be combined in other forms. For example, a coding standard checklist can be combined with a component or unit checklist to make sure that each component follows all the recommended practice in the coding standards. This combination of two checklists forms a two dimensional checklist, or a table with each of its entries to be checked off, as illustrated in Table.

**A** template for a two-dimensional checklist by combining a standards checklist and a component checklist component Standards Items

s1 s2**……………sn**

c1

c2

.

.

cm

Similarly, higher dimensional checklists can also be used. In addition, mixed checklists that mix the direct list combinations and hierarchies are also possible, but such mixed checklists should be used with care because of the possible confusion involved.

**Problems and limitations of general checklists**

One of the important characteristics of these checklists is that they are typically not very specific, particularly those high-level ones. For example, a high-level functional checklist typically lists the major functions to be performed by a software product, but the list items are hardly detailed enough for testers to start a specific test run. The translation from this testing model to the test cases and then to test runs is not a simple matter. It usually involves experienced testers setting up the system and testing environment to execute specific test cases.

In addition, repetition of the same test case in a later test run can only be guaranteed with this additional information about setup and environment, but not deduced from the checklist item itself. This would lead to difficulties when we try to rerun the failing execution to recreate the failure scenario for problem diagnosis, or when we need to re-verify the problem fixes. Therefore, additional information needs to be kept for these purposes. With the increased demand for more automation, service, and functionality, modern software systems also become larger and more complex. Such systems consist of many components that are interconnected and interact with one another. There are also many different functions provided or supported by the systems for many different users. Both the structural complexity and functional complexity make it hard to effectively use checklists for testing and quality assurance, because of reasons below:

* It would be hard to come up with checklists that cover all the functional (black-box) or structural (white-box) components from different perspectives and at different levels of granularity, resulting in missed areas of coverage. These areas are the “holes” in coverage commonly referred to by practitioners.
* In an attempt to provide good coverage, a lot of overlaps between different items in the checklists may be introduced, resulting in redundant testing effort when such checklists are used.
* Some complex interactions of different system components or among major system functions are hard or impossible to describe using checklists.

To deal with the third problem, we will introduced finite-state machines (FSMs) and other related models as the basis for formal and systematic testing. The first two problems can be resolved if we can derive a special type of checklists, called ***partitions,*** that can both achieve full coverage of some specifically defined entities or relationships and avoid overlaps. The formality and precision involved in defining these partitions would also help us obtain a more precisely defined set of test cases as compared to general checklist, thus making problem diagnosis, fix re-verification, and other tasks easier to perform.