**4. ACTIVITIES, PEOPLE, AND MANAGEMENT**

In this section, we examine people’s roles and responsibilities in specific test activities as well as the related management issues.

**People’s role in informal vs. formal, testing**

Informal software testing and some types of formal testing could involve minimal prior knowledge of the software products or systems. One simple way to test the software is to just run it and observe its behavior. Some obvious problems can be easily recognized by people with almost no prior knowledge of computer software and software products.

Some formal forms of testing, such as usability testing, can be performed with little prior knowledge as well. For example, to test some user-friendly, “plug-and-play” software products, novice users are often asked to start using the products. Their behavior and their difficulties in using the products are observed and related information is recorded for usability assessment and improvement. In this scenario, the testing involves the actual novice users as testers, but it may also involve experienced testers who observe and record the testing information. With automated information recording, the role of the experienced tester in this situation can be eliminated.

Because of the above situations, many people have the wrong perception that testing is “easy”, and any “warm body” can perform testing. This misconception also contributes to various problems related to software management, where the least experienced and skilled people are assigned to testing groups. This problem can be corrected by a good knowledge of the technical skills and experience involved in testing, and through some organizational initiatives, such as creating a well-established and well respected career path for testers (Weyuker et al., 2000).

For the large and complex software systems used in society today, any hope of assured quality needs to be supported by testing beyond informal ad hoc testing. We need to model the software systems, their operational environment, their users and usage scenarios, sequences, and patterns, so that systematic testing can be performed to ensure that these systems satisfy their customers’ quality expectations. Test cases can be derived from these models and used systematically to detect and fix defects and to ensure software quality and reliability. All these activities are performed by individual testers or testing teams.

Various other development personnel also need to be involved in testing activities. For example, as part of the follow-up activity to testing, problems detected during testing need to be resolved by the people who are responsible for the creation of the product design or code. Therefore, software developers, particularly those designers and programmers whose code is tested, also need to be involved in testing, although mostly indirectly to follow up on failure observations and defect fixing.

Sometimes, people can play the dual role of developers and testers, when they test their own code, such as in the unit or component testing sub-phases. However, for the overall system, professional testers are typically employed to testing the integration of different components and the overall operation of the system as a whole in the integration, system, and acceptance testing sub-phases.

**Testing teams: Organization and management**

The test activities need to be managed by people with a good understanding of the testing techniques and processes. The feedback derived from analyses of measurement data needs to be used to help with various management decisions, such as product release, and to help quality improvement.

Test managers are involved in these activities. Testers and testing teams can be organized into various different structures, but basically following either a horizontal or a vertical model:

* ***A vertical model*** would organize around a product, where dedicated people perform one or more testing tasks for the product. For example, one or more teams can perform all the different types of testing for the product, from unit testing up to acceptance testing.
* ***A horizontal model*** is used in some large organizations so that a testing team only performs one kind of testing for many different products within the organization. For example, different products may share the same system testing team.

Depending on the demand for testers by different projects, staffing level may vary over time. In the vertical model, as the product development shifts from one phase to another or as the development focus shifts from one area to another, project personnel could be reassigned to perform different tasks. One common practice in industry is to use programmers to perform various testing tasks when testing phase peaks. This practice may create various problems related to staffing management: If not done carefully, it may also lead to project delays, as in Brooks’ famous observation that adding people to a late project will make it later (Brooks, 1995). The mismatch between people’s expertise and their assignments may also result in more defects passing through the testing phase to cause additional in-field problems. This fact is part of the reason for people to adopt the horizontal model where staffing level variations can generally be better managed due to the different schedules and demands by different projects.

In reality, a mixture of the two is often used in large software organizations (Tian, **1998),** with low-level testing performed by dedicated testers or testing teams, system testing shared across similar products, and general project support provided by a centralized support unit for the entire organization. The general project support includes process, technology, and tool support necessary for formal development and testing. This centralized support unit resembles the so-called experience factory that also packages experience and lessons learned from development for more effective future use (Basili, 1995). The idea of experience factory is connection to defect prevention based on process improvement.

**External participants: Users and third-party testers**

Besides the above internal participants, external participants or users may also be involved in testing. The concept of users can also be expanded to include non-human users of software as well, such as other software and hardware environments that the software product in question interacts with (Whittaker, 2001). This extended user concept is particularly relevant to embedded systems or heterogeneous systems with extensive software components.

In general, the users’ views and perspectives, their usage scenarios, sequences, and patterns, and the overall operational environment need to be captured in some models and used in testing to ensure satisfactory performance and reliability for the software products. This is particularly true for usage-based statistical testing, where active user participant is essential in model construction. Sometimes, the users can even serve informally as testers, such as in the usability testing example earlier.

For certain types of software systems, such as those used in defense industry or government, independent verification and validation (IV&V) model is extensively used, where software systems are independently tested or verified and validated using various techniques by third-party participants. This model has gained popularity for various other types of high-assurance software systems, where high reliability, high integrity, or other properties are required, resulting in the so-called certification model or certification pipeline (Voas, 1999).

Another reason for IV&V’s popularity is the increasing use and focus on software development using COTS (commercial-off-lhe shelf) components and CBSE (component based software engineering, or CBSD - component-based software development). In such paradigms, independent testing and certification of software components or reusable parts are key to the possible selection, use, and adoption of software components, parts, or subsystems.

**5. TEST AUTOMATION**

Test automation aims to automate some manual tasks with the use of some software tools. The demand for test automation is strong, because purely manual testing from start to finish can be tedious and error-prone. On the other hand, long standing theoretical results tell us that no fully automated testing is possible. Even most of the major sub-activities, such as result checking or the oracle problem, are undecidable.

However, some level of automation for individual activities is possible, and can be supported by various commercial tools or tools developed within large organizations. The key in the use of test automation to relieve people of tedious and repetitive tasks and to improve overall testing productivity is to first examine what is possible, feasible, and economical, and then to set the right expectations and goals. Various issues related to test automation include:

* specific needs and potential for automation;
* selection of existing testing tools, if available;
* possibility and cost of constructing specific test automation tools;
* availability of user training for these tools and time/effort needed;
* overall cost, including costs for tool acquisition, support, training, and usage;
* impact on resource, schedule, and project management. We examine test automation in connection with the major test activities and people’s roles and responsibilities in them.

**Automation for test execution**

Among the three major test activities, preparation, execution, and follow-up, execution is a prime candidate for automation. In fact, this is the area in which the earliest test automation tools found some unequivocal successes. For example, various semi-automatic debugging tools or debuggers allow testers to set and reset variable values and execution states during execution and observe the dynamic execution behavior at different observation points. These tools are semi-automatic because testers are still involved in test execution intervention.

Many of the modern test automation tools can be considered as enhanced debuggers that work for larger products, automate more individual testing activities, and are generally more flexible and more tailorable than earlier debuggers. Various automated task sequencing tools for job transfer from one test run to another work in much the same way as job dispatcher/scheduler in various operating systems. In fact, most such test run sequencing tools are platform-specific, and are often constructed within testing organizations using some system utilities or APIs (application program interfaces).

An additional functionality for many of the test automation tools is to allow information recording and collection. For example, in the testing of some commercial software product in IBM (Tian et al., 1995; Tian, 1998), an internal test automation tool called T3 was used to generate workload, monitor the execution, and record various execution details for a subset of test scenario classes listed in Table 7.1. The specific measurement data that need to be collected are dictated by the specific analyses to be performed. Therefore, we cover test measurement tools in conjunction with analysis tools later.

**Automation for test planning and preparation**

In test planning and preparation, the potential for automation is different for different sub-activities. The overall planning part can only be carried out by experienced personal with expertise in planning and management as well as a good understanding of testing and development technologies. Not much automation can be achieved in these sub-activities, nor is there a high demand for automation here. Similarly, test procedure planning is primarily done by experts, although the planned procedure can be later enforced and automated during actual test execution with the help of various test execution automation tools.

Test case preparation is the area where there is some realistic potential for automation. For example, in testing of legacy products, various automated analysis can be performed to compare the current version of the product with its previous versions, and to screen the existing test suites to select the ones for regression testing. For construction of new test cases, automation is also possible. For example, in the T3 tool we mentioned above for test execution support, a script can be provided to generate different workload for testing, which effectively generates test cases and related test runs dynamically from test script. However, the test scripts, which are high-level descriptions of what to test, need to be constructed in the first place by the experienced testers. These test scripts are usually much simpler and shorter, thus much less costly to generate than actual test cases. Consequently, a semi-automated test case generation is supported in this case.

In general, test scripts or test cases are based on some formal models. The model construction for different test techniques requires high levels of human intelligence and expertise, and is therefore not easily automated. However, some individual steps in model construction can be automated, such as some automated data gathering, graphical or other aids for modeling, etc. For small-scale programs, some tools can be used to generate certain models and test cases directly, much like using compilers to generate object code from source code. However, these tools cannot scale up to large software systems. In addition, in most of the models, various decisions need to be made and parameters need to be selected for specific model variations, which can only be carried out by people with proper expertise.

Once such a model is constructed, various tools can be used to generate test cases automatically. Sometimes, even if a tool is not directly available, the testing model is typically associated with some algorithms that can be at least partially implemented for automatic generation of at least some test cases. For example, once an underlying usage model in the form of a Markov chain is constructed, several algorithms can be used for usage based statistical testing to cover frequently used usage patterns, frequently visited states, and call-pairs (Avritzer and Weyuker, 1995). If there is a commercial tool or an existing tool within the organization available, the key in its adoption for test case generation is to understand what kind of model is supported and how difficult it is to construct models of this type, in order to match it with our purpose of testing.

**Automation for test measurement, analysis, and follow-up**

In terms of analyses of test results and follow-up actions, the situation is similar to test planning and preparation. Most of the follow-up actions involve problem fixing and various other remedial and improvement initiatives, very little of which can be automated. However, specific analysis activities can be supported by various analysis and modeling tools. For example, many of the reliability analysis activities described in Tian (1998) were automated. This was achieved after many rounds of studies that converged on the appropriate models and data to use. Many popular tools were discarded because they were found to be unsuitable for the type of commercial products froni IBM. This experience told us that automated analysis tools should not be indiscriminately applied, but rather based on intelligent choice based on one's own specific environment and experience. **A** general tool support strategy for QA and development process measurement, data analysis, and follow-up.

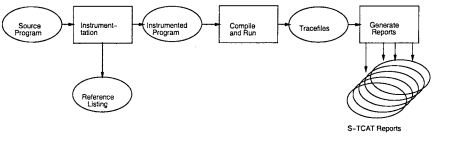
Closely related to test result analyses a.re coverage analysis for coverage-based testing and reliability analysis for usage-based testing. For traditional reliability analysis, we typically need results for individual test ru.ns and related timing information (Lyu, 1995a). Sometimes, some additional information, such as test input, environment, and personnel information can also be used to link input states to reliability or to identify problematic areas for focused reliability improvement (Tian, 1995). These data can usually be automatically collected using various test execution tools or dedicated data collection tools.

Coverage analysis usually involves the use of more detailed information and measurement data than that for reliability analysis. But, fortunately, various coverage analysis tools exist to collect coverage information during test execution. For example, several popular commercial test tools collect and analyze coverage information to provide feedback and quality assessments to the testing process or the overall development process, including:

* McCabe Test from McCabe and Associates provides control flow coverage related information and analysis.

S-TCAT (System Test Coverage Analysis Tool) from SRI (Software Research, Inc.) provides function-level coverage and call-pair information. S-TCAT can also be integrated into a tool suite called Testworks from SRI for various other testing purposes. ATAC (Automatic Test Analysis for C) from Telecodia is a data flow coverage analysis tool for white-box testing.

To use these tools for coverage analysis, the source code is usually instrumented to build an instrumented test driver. When this instrumented code is run in testing, information related to coverage in the form of raw data is collected. Later on, the raw data are analyzed for coverage. Figure 7.1 illustrates these steps with the use of S-TCAT for test coverage analysis. Each of these steps is usually automated by the tools themselves.



**Figure.** Test coverage analysis with S-TCAT

One interesting fact to notice is that, although these tools are designed for coverage-based testing, they can also be used sometimes to support usage-based testing, such as the use of S-TCAT to collect customer usage information for some IBM products (Lu and Tian, 1993b).