

# SOFTWARE TESTING

SMITA SINHA

Guru Gobind Singh Inderprastha University, Delhi  
smitasmita\_s@yahoo.com

*Abstract- One of the major problems within the software testing area is how to get a suitable set of test cases to test a software system. This set should assure maximum effectiveness with the least possible number of test cases. There are nowadays numerous testing techniques available for generating test cases. However, many of them are never used, while a few are used over and over again. Testers have little (if any) information about the available techniques, their usefulness and, generally, how suited they are to the project at hand. This lack of information means less tuned decisions on which testing techniques to use. This paper presents the study different software testing technique selection. When instantiated for a variety of techniques, the schema provides developers with a catalogue containing enough information for them to select the best suited techniques for a given project. The schema, and its associated catalogue, assure that the decisions developers make are based on grounded knowledge of the techniques rather than on perceptions, suppositions and assumptions. Selecting appropriate test cases is a crucial activity in software testing. In this paper, we give an overview and discuss existing methods and tools for test case selection.*

## 1.1 The Testing Process

The testing phase is the final frontier for software before it enters the market. Testing strives to ensure that products meet the market needs. Testing provides information or data that helps assure the quality of the following:

1. Product (finding faults)
2. Decision (identifying risks)
3. Process (finding root cause)

Testing is the process of comparing what a product or service is, to what it should be.

Based on the discussion so far, testing faces two conflicting needs. It needs to ensure the quality of the product, and do so in the shortest possible time, to maintain a competitive edge in the market. The scenario becomes more complex if we consider the software delivery challenges. There is another dimension to this scenario-the need to reduce the effort spent on producing the highest quality

product in the shortest possible time. The Testing team, acting as a custodian of the customer expectation, has to strike a balance between three critical parameters: **Quality, Resources, and Time**. None of these can be changed in isolation because a change always affects the other two parameters.

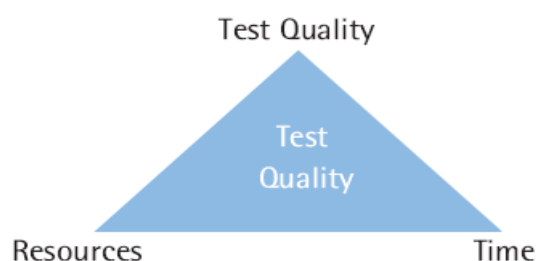


Figure:1

## 1.2 The Testing Market

The services industry is rapidly adopting software solutions. The use of commercial off-the-shelf products and third-party solutions is becoming common. In this scenario the IT group of a services company virtually becomes a system integrator. Testing becomes a vital component with multi-party development. Though it adds no functional value, it adds an immense business value. A testimony of the growing significance of testing is the increase in the Quality Assurance (QA) share of spends in the product life cycle, as shown in the chart below:

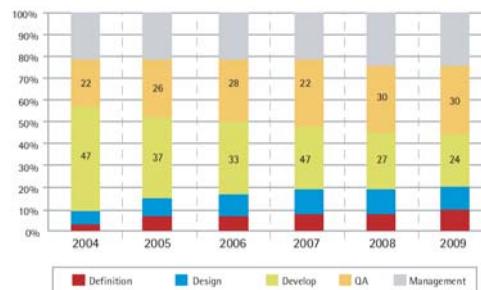


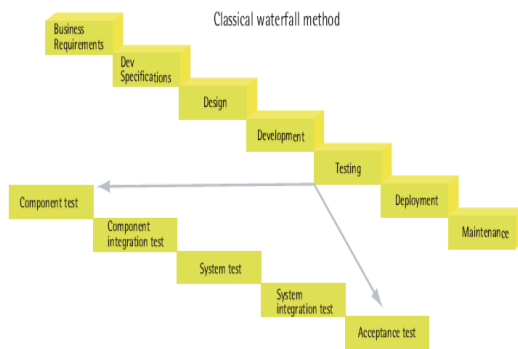
Figure:2 Growing significance of Testing

Testing is fast becoming a specialized service. The increase in the time spent is not a real indicator of

the increase in the spread of the testing effort. The testing coverage and importance has increased much more than what is indicated above. More and more automation tools are being used. but within a short period of time (2-6 cycles for a low-complexity product or application), the breakeven is achieved. Thereafter, the tools provide a much higher cover for a lower cost. Testing requires specialized skills.

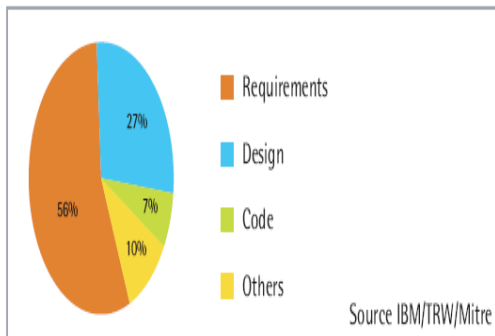
### 2.2 Conventional Testing in SDLC

Software Development Life Cycle (SDLC) defines the stages or phases of a project from inception through completion and delivery of the final project. In the conventional waterfall method, testing comes at the very end of the life cycle.



**Figure:3 water fall method**

The strength of the waterfall method lies in its thoroughness, and leads to an extremely robust code. But it has its drawback too---the time taken to reach the market. The testing stage, confined to the end, has its own problems as explained below. The chart shows the distribution of defects in the life cycle.



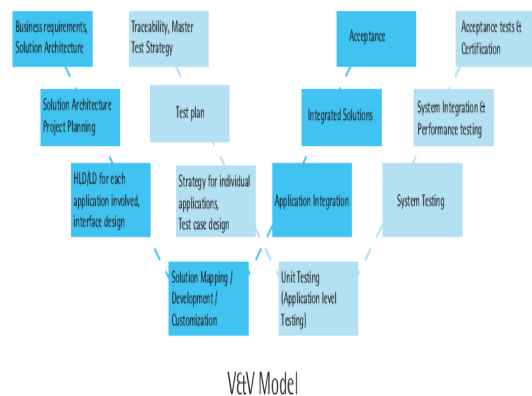
**Figure:4 Defects in life cycle**

As shown by a study, 56% of the defects result out of the requirements analysis phase, which if

tackled then, will not percolate down to subsequent phases, and hence will reduce time and effort required later to fix those defects.

### V & V MODEL

The test process rigidity and maturity in an organization comes with time and in different phases. The selection of the life cycle model is one aspect of the maturity. More often than not, the journey begins with waterfall model. The other aspect of maturity lies in the process being followed in bringing the operational segregation and division of labor.



**Figure: 5 V & V MODEL**

### 3.Criteria for Selecting a Testing Approach

A number of items must be considered when determining the approaches to the organization of the testing, the selection of the tests, and the methods to use when conducting the tests. Any approach must be compatible with the:

- Customer’s requirements,
- Development strategies,
- Testing objectives,
- System environment,
- Mandated methodologies.

#### 3.1.1 Customer Requirements

Review the customer's requirements before identifying the approach to testing. The amount of structure customers provide can vary. The customer's requirements may be very specific as to how testing is to be conducted, may provide some guidelines or parameters, or may allow strategies to be determined by the testers.

### 3.1.2 Development Strategies

The order of development must be compatible with the order in which testing is to be conducted. When organizing the order of testing, the development schedule must be considered. If possible, the testers should work with the development team to change the development order if it is critical to the logical organization of the testing.

In addition, development strategies, (e.g., use of traditional development methods or rapid application development) can influence the approach to testing.

### 3.2 Testing Objectives

The testing objectives can affect the selection of an appropriate approach. For example, if the objective is to provide a system for acceptance testing that is free of all errors, except those that are cosmetic in nature, then errors that are cosmetic in nature will be given a lower priority.

#### 3.2.1 System Environment

The type of project, (e.g., host-based versus client/server), affects the approach. For example, testing tools may be required to assist with the testing of the more complicated client/server environment. In addition, the system environment influences the way in which the testing environment is isolated, (e.g., the use of different regions versus the use of different servers).

#### 3.2.2 Mandated Methodologies

Choose approaches that are compatible with any mandated methodology. For example, the methodology may indicate that the design specification must be complete before the identification of tests can start.

## 4. Software Testing Techniques

The choice of the right techniques is critical to achieving a good return on the test investment. Some tests happen before we can even run the software. Some tests involve analyzing the structure of the system, while others involve analyzing the system's behavior. Each technique can involve special skills and particular participants, and might appropriately entail the use of tools—or not. The greater visibility of software systems and the cost associated with software failure are motivating factors for planning, through testing. It is not uncommon for a software organization to spend 40% of its effort on testing.

### 3.2 Testing objectives

A number of rules that act as testing objectives are:

- Testing is a process of executing a program with the aim of finding errors.
- A good test case will have a good chance of finding an undiscovered error.
- A successful test case uncovers a new error.

### 3.3 Static Testing

A static test is one that evaluates the quality of the system without the system actually running, called desk-checking, which is one kind of static test. Another kind of static test, in this case a social one, is a review meeting that evaluates requirements, design, or code. The benefit is clear once think about it: If we can find a problem in the requirements before it turns into a problem in the system, that will save time and money.

**Random Testing Techniques.** The random testing techniques family is composed of the oldest and intuitive techniques. This family of techniques proposes randomly generating test cases without following any pre-established guidelines. Nevertheless, pure randomness seldom occurs in reality.

### 3.4 White Box Testing

White box testing is a test case design approach that employs the control architecture of the procedural design to produce test cases. Using white box testing approaches, the software engineering can produce test cases that (1) guarantee that all independent paths in a module have been exercised at least once, (2) exercise all logical decisions, (3) execute all loops at their boundaries and in their operational bounds, (4) exercise internal data structures to maintain their validity.

#### 3.4.1 Basis Path Testing

Basic path testing is a white box testing techniques that allows the test case designer to produce a logical complexity measure of procedural design and use this measure as an approach for outlining a basic set of execution paths. Test cases produced to exercise each statement in the program at least one time during testing.

**3.6.2 Data Flow Testing Techniques.** Data flow testing techniques also require knowledge of source code. The objective of this family is to select program paths to explore. A path is a code sequence that goes from the start to the end of the program. sequences of events related to the data state. Again, the techniques in this family vary as

to the rigour with which they cover the code variable states.

### 3.4.3 Cyclomatic Complexity

McCabe's cyclomatic complexity is a software metric that offers an indication of the logical complexity of a program. When used in the context of the basis path testing approach, the value is determined for cyclomatic complexity defines the number of independent paths in the basis set of a program and offer upper bounds for number of tests that ensures all statements have been executed at least once. An independent path is any path through the program that introduces at least one new group of processing statements or new condition.

### 3.4.5 Graphical Matrices

The procedure involved in producing the flow graph and establishing a set of basis paths can be mechanized. To produce a software tool that helps in basis path testing, a data structure, called a graph matrix, can be quite helpful. A graph matrix is a square matrix whose size is the same as the identified nodes, and matrix entries match the edges between nodes.

#### Graph Matrix

In the graph and matrix each node is represented with a number and each edge a letter. A letter is entered into the matrix related to connection between the two nodes. By adding a link weight for each matrix entry the graph matrix can be used to examine program control structure during testing. In its basic form the link weight is 1 or 0. The link weights can be given more interesting characteristics:

- The probability that a link will be executed.
- The processing time expanded during traversal of a link
- The memory required during traversal of a link

Represented in this form the graph matrix is called a connection matrix.

### Loop Testing

Loops are the basis of most algorithms implemented using software. However, often we do consider them when conducting testing. Loop testing is a white box testing approach that concentrates on the validity of loop constructs.

Four loops can be defined: simple loops, concatenate loops, nested loops, and unstructured loops.

**Simple loops:** The follow group of tests should be used on simple loops, where n is the maximum number of allowable passes through the loop:

- Skip the loop entirely.
- Only one pass through the loop.
- Two passes through the loop.
- M passes through the loop where  $m < n$ .
- $n-1$ ,  $n$ ,  $n+1$  passes through the loop.

**Nested loop:** For the nested loop the number of possible tests increases as the level of nesting grows. This would result in an impractical number of tests. An approach that will help to limit the number of tests:

- Start at the innermost loop. Set all other loops to minimum values.
- Conduct simple loop tests for the innermost loop while holding the outer loop at their minimum iteration parameter value.
- Work outward, performing tests for the next loop, but keeping all other outer loops at minimum values and other nested loops to "typical" values.
- Continue until all loops have been tested.

**Concatenated loops:** Concatenated loops can be tested using the techniques outlined for simple loops, if each of the loops is independent of the other. When the loops are not independent the approach applied to nested loops is recommended.

**Unstructured loops:** This class of loop should be redesigned to reflect the use of the structured programming constructs.

### 3.5 Black Box Testing

Black box testing approaches concentrate on the fundamental requirements of the software. Black box testing allows the software engineer to produce groups of input situations that will fully exercise all functional requirements for a program. Black box testing is not an alternative to white box techniques. It is a complementary approach that is likely to uncover a different type of errors that the white box approaches.

**Black box testing tries to find errors in the following categories:**

- (1) Incorrect or missing functions
- (2) Interface errors
- (3) Errors in data structures or external database access
- (4) Performance errors, and
- (5) Initialization and termination errors.

By applying black box approaches we produce a set of test cases that fulfill requirements:

- (1) Test cases that reduce the number of test cases to achieve reasonable testing.
- (2) Test cases that tell use something about the presence or absence of classes of errors.

### 3.5.1 Equivalent Partitioning

Equivalence partitioning is a black box testing approach that splits the input domain of a program into classes of data from which test cases can be produced. An ideal test case uncovers a class of errors that may otherwise before the error is detected. Equivalence partitioning tries to outline a test case that identifies classes of errors.

Test case design for equivalent partitioning is founded on an evaluation of equivalence classes for an input condition. An equivalence class depicts a set of valid or invalid states for the input condition.

**Equivalence classes can be defined based on the following:**

- 1) If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
- 2) If an input condition needs a specific value, one valid and two invalid equivalence classes are defined.
- 3) If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined.
- 4) If an input condition is Boolean, one valid and invalid class are outlined.

### 3.5.2 Boundary Value Analysis

A great many errors happen at the boundaries of the input domain and for this reason boundary value analysis was developed. Boundary value analysis is test case design approach that complements equivalence partitioning. BVA produces test cases from the output domain also.

**Guidelines for BVA are close to those for equivalence partitioning:**

- If an input condition specifies a range bounded by values a and b, test cases should be produced with values a and b, just above and just below a and b, respectively.
- If an input condition specifies various values, test cases should be produced to exercise the minimum and maximum numbers.
- Apply guidelines above to output conditions.
- If internal program data structures have prescribed boundaries, produce test cases to exercise that data structure at its boundary.

**Advantages of Boundary Value Analysis:-**

1. **Robustness Testing** - Boundary Value Analysis plus values that go beyond the limits.
2. Min - 1, Min, Min +1, Nom, Max -1, Max, Max +1
3. Forces attention to **exception handling**.
4. For strongly typed languages robust testing results in run-time errors that abort normal execution.

### 3.5.3 Cause-Effect Graphing Techniques

In too many instances, an attempt to translate a policy or procedure stated in a natural language into a software causes frustration and problems. Cause-effect graphing is a test case design approach that offers a concise depiction of logical conditions and associated actions. The approach has four stages:

- Cause (input conditions) and effects (actions) are listed for a module and an identifier is allocated to each.
- A cause-effect graph is created.
- The graph is altered into a decision table.
- Decision table rules are modified to test cases.

### 3.5.4 Comparison Testing

Under certain situations the reliability of the software is critical. In these situations redundant software and hardware is often used to ensure continuing functionality. When redundant software is produced separate software engineering teams produce independent versions of an application using the same applications. In this context each version can be tested with the same

test data to ensure they produce the same output. These independent versions are the basis of a black box testing technique known as comparison testing. Other black box testing techniques are performed on the separate versions and it is assumed if they produce the same output they are assumed to be identical. However, if this is not the case then they are examined further.

### 3.6 Testing for Real-Time Systems

The specific characteristics of real-time systems makes them a major challenge when testing. The time-dependent nature of real-time applications adds a new difficult element to testing. Not only does the developer have to look at black and white box testing, but also the timing of the data and the parallelism of the tasks. In many situation test data for real-time system may produce errors when the system is in one state but to in others. Comprehensive test cases design methods for real-time systems have not evolved yet.

**However, a four-stage approach can be put forward:**

**Task testing:** The first stage is to test independently the tasks of the real-time software.

**Behavioural testing:** Using system models produced with CASE tools the behaviour of the real-time system and examine its actions as a result of external events.

**Intertask testing:** Once errors in individual tasks and in system behaviour have been observed testing passes to time-related external events.

**Systems testing:** Software and hardware are integrated and a full set of systems tests are introduced to uncover errors at the software and hardware interface.

### 4. Unit Testing

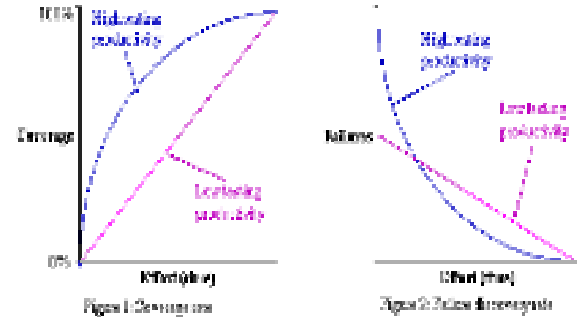
Unit testing concentrates verification on the smallest element of the program – the module. Using the detailed design description important control paths are tested to establish errors within the bounds of the module.

### 4.3 Integration Testing

Once all the individual units have been tested there is a need to test how they were put together to ensure no data is lost across interface, one module does not have an adverse impact on another and a function is not performed correctly. Integration testing is a systematic approach that produces the program structure while at the same time producing tests to identify errors associated with interfacing.

*Code coverage analysis* is the process of:

- Finding areas of a program not exercised by a set of test cases,
- Creating additional test cases to increase coverage, and
- Determining a quantitative measure of code coverage, which is an indirect measure of quality.



The sequence of coverage goals listed below illustrates a possible implementation of this strategy.

1. Invoke at least one function in 90% of the source files (or classes).
2. Invoke 90% of the functions.
3. Attain 90% condition/decision coverage in each function.
4. Attain 100% condition/decision coverage.

Notice we do not require 100% coverage in any of the initial goals. This allows we to defer testing the most difficult areas. This is crucial to maintaining high testing productivity; achieve maximum results with minimum effort.

### 5. Characterization schema:

One major issue in managing software engineering knowledge is the construction of information repositories for software development artefacts(techniques, products, processes, tools, etc.). What is the appropriate characterisation schema? This proposes an empirical and iterative process to identify the information that should be used to characterise a software engineering artefact, using both theoretical knowledge, practical experience, and expert opinion to generate a schema. The ultimate goal is to improve the schema and the package contents based upon it experience in their application. The proposed

process has been applied to define a characterisation schema for testing techniques.

### 5.1 Review The Testing Technique:-

Our aim is to review the empirical studies designed to compare testing techniques in order to identify the maturity level of their knowledge, based on the kind of empirical studies performed for getting that knowledge. We have grouped the empirical studies reviewed into several subsets taking into account which techniques they compare:

- **Intra-family studies:-** which compare techniques belonging to the same family to find out the best criterion, that is, which technique of all the family members should be used. We have found:

- Studies on the data flow testing techniques family
- Studies on the mutation testing techniques family

- **Inter-family studies:-** which study techniques belonging to different families to find out which family is better, that is, which type of techniques should be used. We have identified:

- Comparative studies between the control flow and data flow testing techniques families.
- Comparative studies between the mutation and data flow testing techniques families.
- Comparative studies between the functional and control flow testing techniques families.

### 5.3.1 Proposed Process for Discovering Relevant Information

Having detected the pitfalls of current characterisation schema construction processes, we propose a means of determining relevant information about any particular artefact type for inclusion in an experience repository.

Stages of the proposed characterisation schema construction process. This process can be divided into two parts: schema generation and schema testing.

- **Schema Generation.** Schema generation has been divided into four different stages. They explicitly state each source of information used to formulate the schema, and each stage aims to gather different information types. The generation stages are:

1. Development of a theoretical schema,
2. Development of an empirical schema,
3. Synthesis of perspectives and expert peer review.

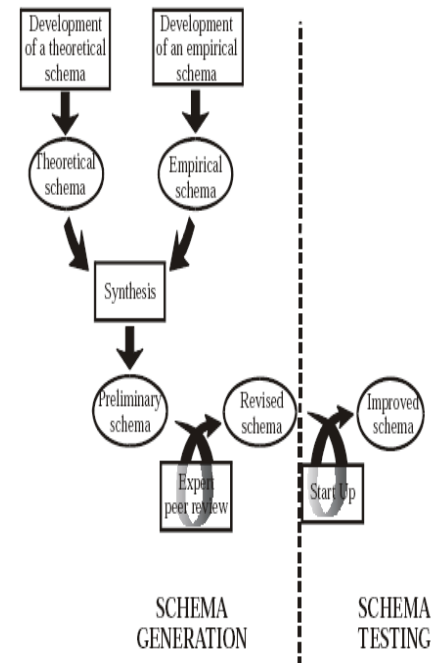
- **Schema Testing.** Schema testing or start up involves having two different population groups examine the schema and assess two different facets: population and use.

### 5.4 Theoretical Schema

Theoretical schema relates to what information it should contain. This is not an easy task, however, as the schema has to meet the information needs of a variety of people with different goals. More precisely, it must be:

- Useful for consumers when selecting the artefacts for their project situation.
- Possible for producers to fill in the information asked for in the schema.

Fig. 1. Proposed characterisation schema development process.



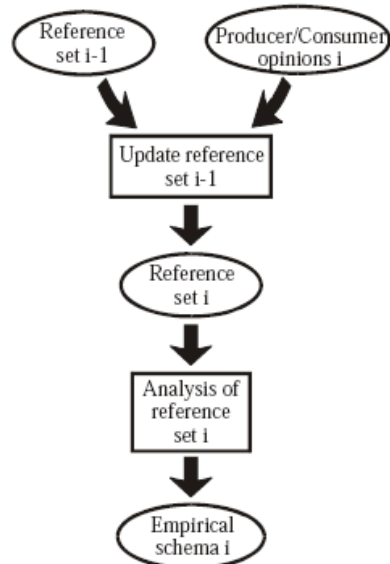
The schema obtained in the first iteration reflects the opinion of the schema designer on the information that can influence decision-making on which artefacts should be used in a given project. However, this schema does not necessarily respond, at least completely, to the consumers' opinion of selection. Therefore, the question is *What information does the consumer need to select an artefact from the experience base?*

### 5.5 Empirical schema

**Empirical schema** is developed incrementally. A set of opinions (questions or information) about the information required to completely select/define an artefact is gathered for each consumer/producer surveyed. The sets of questions/information obtained are analysed incrementally. This means that the producers/consumers are gradually incorporated, making it possible to cover the total set of possible producers/consumers according to their characteristics. The process is, therefore,

inductive, producing a schema containing the characteristics desired by producers and consumers. To be more precise, the iteration for running the analysis is as follows. Taking a reference set (originally empty) and the opinions of the producer/consumer, the reference set is updated to include any opinions not included before, and the respective empirical schema is obtained. The reference set can be updated in several ways: either by adding new opinions or reformulating others to make them more generic or more specific (never by deletion). Shows the activities to be performed to get the  $i$ -th empirical schema.

**Fig.** Activities to get the  $i$ -th empirical schema.



One interesting point is that the characteristics of the participants should be known, as it is important to be acquainted with what type of producers /consumers are represented in the schema. Another point (not as important as accounting for all producer/consumer types) is the number of people that have to participate in this stage. The number is not essential, as Glaser and Strauss state that the number of data collected during research is relevant for testing and not for generating the hypothesis. So, the number of individuals involved will be important at that point and, as such, will be taken into account later on. The stopping criterion for this activity is the stability of the characterization schema. It will not be possible to stop gathering information from different people until the rate of change of the schema is zero for at least the last 25% of subjects. Therefore, what we are examining at this stage is the evolution and change of the characterisation schema as new producers/consumers are incorporated.

## 5.6 Expert Peer Review

The schema obtained after the synthesis of the theoretical and empirical schemas reflects the viewpoint of the schema designer, consumers and producers concerning the selection problem. However, neither the consumers nor the producers have so far seen the schema (they were asked for their opinion on selection, but they were never shown what information had been input). It would, therefore, appear to be a good idea to get someone else to inspect and give an opinion on the schema. Also, according to the principles of some sciences (for example, medicine), it is advisable to get a second opinion about a complex problem. Therefore, a series of experts in the area the artefact belongs to, should be asked to give their verdict on the preliminary schema prior to start up. The goal of this expert peer review is to correct possible schema defects caused by the way in which it was derived. The typical defects of the schema obtained prior to the review by experts are as follows:

- **Defects of form.** Both producers and consumers have given their particular view of the information they believe to be relevant for selecting that particular artefact. However, the schema designer alone created the structure that reflects this information. It would not be amiss to get a second opinion on this structure.
- **Defects of substance.** The information for the preliminary schema is gathered indiscriminately. It may contain errors involuntarily introduced by the schema designer or by the people participating in the research. For example, there may be redundant information (dependencies between information contained in the schema), missing or unworkable information not detected by the designer. The preliminary schema will be modified on the basis of the analysis of the opinions of the experts to incorporate their suggestions, giving rise to a new, improved and almost final schema. The ideal number of experts for an expert peer review is as many as possible, and no less than three, so that discrepancies among experts can be handled.

## 5.7 Schema Levels

The software system testing process can be divided into the following stages:

1. Selection of the quality attributes that are to be tested, as well as the expected values for each attribute, when they are to be tested, the metrics to be used for the evaluation, and the parts of the system that will be affected by each test.
2. For each of the attributes identified in the previous stage, the tests identified above should be performed, which means: generate and execute the

test cases, and evaluate the results obtained, always considering the environment where the test is to take place. More formally, we have named these types of information as **tactical** and **operational** information and they correspond to two different levels.

The **scope of the test**, which can be defined by saying what part of the software system is to be tested, when the test is to be run and the components of the software system that are affected by the test. Depending on which test is run, it will affect different parts of the software, ranging from an algorithm, through an entire module, a group of modules that perform a system function, to a subsystem and even the entire system. Also, depending on how system development has been organised, the test will take place at one time or another within the process. We should also specify the part of the functionality offered by the system that needs to be tested. The scope, then, refers to the part of the system involved in the test.

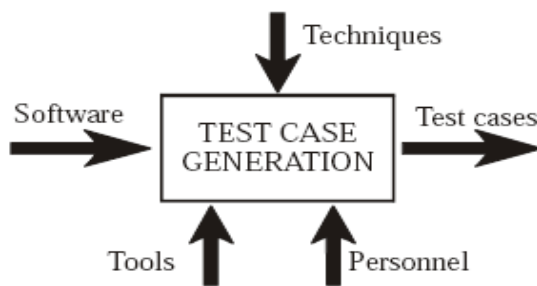


FIGURE: TEST CASE GENERATION TECHNIQUE

In other words, the test case generation technique that is applied to the software outputs a series of test cases within an environment that is determined by the tools available for performing the task and the personnel who carry out the task. Therefore, according to Fig. 19, it can be said that the information that the operational level of the characterisation schema should contain has to refer to:

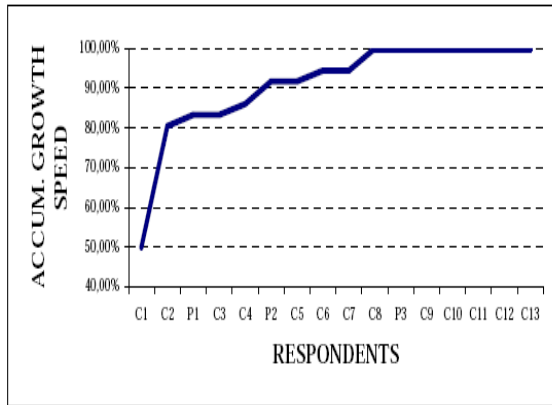
- The people who are to use the technique or *agents*. The characteristics of these people can lead to one or another technique being chosen. If the testing personnel are not very experienced in one technique and there is no time for training, another is likely to be selected.
- The *tools* that should or could be used. The fact that a company does or does not own a given tool that supports the use of a given technique can lead to the selection of one technique rather than another.

- The software (code) to be tested or *object*. The code has certain characteristics that can determine the use or rejection of a technique. For example, the type of programming language used, the code size, etc.
- The *technique*. Depending on the characteristics of the technique, a decision can be made on whether or not to use it at a given time. Characteristics like complexity, effectiveness, maturity, usability, etc., will be the key for deciding on its use.
- The generated test cases; that is, the *results* (and/or consequences) of using the technique. Some characteristics of the technique are environment dependent, and these are precisely the ones that reflect its behaviour. How good a technique is when applied can be ascertained from the generated test cases and not from the technique. Thus, some characteristics of these test cases will be of interest for selection purposes.

### 5.7. Development of an Empirical Schema

The tasks to be carried out to get the empirical schema include sending out two different questionnaires to respondents: a questionnaire that asks the consumer what information (s)he believes to be relevant for selection purposes, and another that asks the producer what information (s)he believes to be necessary to define a testing technique. The responses are then analysed to produce a characterization schema that will reflect the opinions of both consumers and producers about the selection problem. The empirical schema has been built incrementally, as described in earlier. That is, the first version of the empirical schema was generated with the information received from the first respondent and the schema version was updated as the information from successive respondents was analysed. When working with the empirical schemas, we tried to use the levels and elements of the theoretical schema as far as possible, because the respondents only supplied attributes. An important issue we had to deal with during this stage was the stability analysis of the empirical schema. This analysis was performed in order to find out when to stop gathering information. Fig. 20 shows the accumulated growth speed of the empirical schema. The x-axis shows the different people surveyed ordered according to time (C stands for consumer and P stands for producer) and the y-axis shows the size of the empirical schema as a percentage of its final size. It can be seen that the empirical schema reaches 50% of its final size with the first respondent. This figure increases to 80%

with the second respondent, and the schema reaches its final size with the tenth respondent. This means that the last six respondents did not add any new information to the empirical schema and, therefore, the empirical schema can be considered as stable at this point. **Fig.** Schema accumulated growth speed.



Another of the key tasks for designing the empirical schema was the selection of the respondents. The characteristics of the people involved in the construction of the empirical schema can have a significant influence on the resulting schema.

The people involved should be as heterogeneous as possible to assure that the schema does not reflect a unilateral viewpoint. For this purpose, an attempt was made to include respondents with a wide variety of characteristics: from a range of fields, with varying experience and of different nationality. As the set of participant subjects had to be as heterogeneous as possible, we looked for people who played different roles in the testing area. Also, respondents are asked for information starting with the ones that were most likely to give us more useful one. Table 2 shows the contents of the empirical schema. Note that the empirical schema provides us with some information that did not appear in the theoretical schema, since practitioners care about practical issues that are very often overlooked by theoreticians. The main differences of the empirical schema from the theoretical schema are:

- *Use Level.* It was not possible to associate the information contained in this level with any of the two levels in the theoretical schema. Therefore, a new level was created: the use level. The questions of which this new level is composed refer to the personal experiences of people who have used the technique. This level contains two elements:

- *Project.* The information covered in this element refers to the respondents interest in learning about and characterising software projects in which the technique has been applied in order to compare these earlier projects with the current situation.

- *Satisfaction.* The information covered in this element complements the above information on earlier projects. The respondents are also interested in knowing the results of using the technique in the project from the viewpoint of what impression it caused on the person who used the technique.

- *Tools element.* The information covered in the tools element refers to the characteristics of the tools that can be used when applying the technique. However, the inclusion of too much information can also lead to difficulties. Experts will play an essential role during peer review in dealing with this matter.

## Conclusions

The main problem met by software developers when choosing the best suited testing techniques for a software project is information. The information on testing techniques is, at best, distributed across many sources of information and, at worst, non-existent. The approach we have taken in this research to the problem of gathering relevant information about software testing techniques is called a characterisation schema. Using this schema, we can build a repository that contains the description of each technique of interest and describes all techniques according to the same pattern so that a decision can be made on whether or not to use a technique without having procedural knowledge of the technique. An empirical and iterative process has been followed to search for the information that such a characterization schema should contain. This process is empirical because it takes into account the opinions of a variety of people involved in software testing and iterative because it is gradually refined as new opinions are added to the schema. Finally, the generated schema has been validated in two ways. First, it has been instantiated for several testing techniques, by means of which we were able to test the schema for several techniques and check that it is possible to find the required information. Secondly, an experiment was run to check its behaviour against the use of other methods of selection, such as books.

## References

- [1] Althoff, K. D., Birk, A., Hartkopf, S., Müller W., Nick M., Surmann, D. and Tautz, C. . Systematic population, utilization and maintenance of a repository for comprehensive reuse. In Learning Software Organizations, Methodology and Applications, 11th International Conference on Software Engineering and Knowledge Engineering (SEKE'99). 2000. Springer-Verlag: Berlin. Pages. 25-50.
- [2] Basili, V. R., Lindvall M. Costa, P. Implementing the experience factory concepts as a set of experience bases. In *Proceedings of the 13th International Conference on Software Engineering and Knowledge Engineering*. Buenos Aires, Argentina. 13-15 June 2001.
- [3] Basili, V. R., Rombach, H. D. Support for comprehensive reuse. *Software Engineering Journal*, 303-316, September 1991.
- [4] Basili, V. R., Rombach, H. D., Caldiera, G. *The Experience Factory*. Encyclopedia of Software Engineering – 2Volume Set, pages 469-476 John Wiley & Sons, Inc. 1994.
- [5] Bass, L., Clements, P., Kazman, R. Bass, K. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison-Wesley, January 1998.
- [6] Beizer, B.. *Software Testing Techniques*. International Thomson Computer Press, Second Edition, 1990
- [7] Birk, A. Modelling the application domains of software engineering technologies. *Proceedings of the Twelfth International Conference on Automated Software Engineering (ASE)*. Lake Tahoe, California, November 1997.
- [8] Fenton, N, Krause, P. Neil, M. Software Measurement: uncertainty and causal modeling. *IEEE Software*. 2002. 19(4): p. 116-122.
- [9] Glaser, B., Strauss, A.. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine de Gruyter, 1967.
- [10] Henninger, S. Accelerating the successful reuse of problem solving knowledge through the domain lifecycle. *Proceedings of the Fourth International Conference on Software Reuse*, 124-133, Orlando, Florida, April 1996.
- [11] Komi-Sirvio, S., Mäntyniemi, A., Seppänen, V. Toward a practical solution for capturing knowledge for software projects. *IEEE Software*, 60-62, May/June 2002.
- [12] Maiden, N., Rugg, G. ACRE: Selecting methods for requirements acquisition. *Software Engineering Journal*. 11(3):183-192,1996.
- [13] Pfleeger, S. L. *Software Engineering: Theory and Practice*, Mc-Graw Hill. 1999
- [14] Prieto-Díaz, R. *Software Reusability*, Vol 1, Chapter 4. Classification of Reusable Modules, 99-123. Addison-Wesley, 1989.
- [15] Rus, I., Lindvall, M. Knowledge management in software engineering. *IEEE Software*, 26-38, May/June 2002.
- [16] Sommerville, I. *Software Engineering, 5th edition*. Pearson Education. 1998.
- [17] Vegas, S. *A Characterisation Schema for Selecting Software Testing Techniques*. PhD Thesis. Facultad de Informática. Universidad Politécnica de Madrid. February 2002.
- [18] von Wangenheim, C. G.. REMEX- A Case-Based approach for reusing software measurement experienceware. In *Case-Based Reasoning Research and Development (LNAI 1650)*. K.D. Althoff, R. Bergmann and L.K. Branting, Editors. 1999. pp 173-187.
- [19] von Wangenheim, C. G., Althoff, K. D., Barcia, R. M. Goal-oriented and similaritybased retrieval of software engineering experienceware. In *Learning Software Organizations: Methodology and Applications (LNCS, 1756)*. G. Ruhe, F. Bomarius, editors. 2000. Springer-Verlag: Berlin. Pp 118-141.