Approach Testing for Finding Defects that have Low Albedo Value

Saleem Al-Zoubi a

a IN- Ibid National University,P.O.Box:2600 Ibid 21110 Jordan

it@inu.edu.jo

**ABSTRACT**. The objective of this paper is to introduce and discuss why test cases should be treated as a product. The paper further introduces the practices that have evolved to design good test cases that consistently helped testers to design good test cases, deliver them as a product and meet the demands of test projects that testers undertake.

Apart from the generic discussion that one normally finds articles in test case design, Ross discusses a few practical problems of test teams that affect test design time and again. I have highlighted some of the examples below to keep this reference narrowly focused on only those variations.

**Keywords:** test, testers undertake, process model, Ross.

# 1 Introduction

Software Testing has become a self-governing and a very important profession over a period. As the software development process is becoming a complex activity day by day, the demand to continuously evolve the software testing practices and keeping them aligned to the needs of software engineering is becoming important as well.

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.[[1]](https://en.wikipedia.org/wiki/Software_testing#cite_note-Kaner_1-1) Software testing can also provide an objective, independent view of the [software](https://en.wikipedia.org/wiki/Software) to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding [software bugs](https://en.wikipedia.org/wiki/Software_bug) (errors or other defects).

It involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

* meets the requirements that guided its design and development,
* responds correctly to all kinds of inputs,
* performs its functions within an acceptable time,
* is sufficiently usable,
* can be installed and run in its intended [environments](https://en.wikipedia.org/wiki/Operating_environment), and
* achieves the general result its stakeholders desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding [software bugs](https://en.wikipedia.org/wiki/Software_bug) (errors or other defects). The job of testing is an iterative process as when one bug is fixed, it can illuminate other, deeper bugs, or can even create new ones.

Software testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors.[[1]](https://en.wikipedia.org/wiki/Software_testing#cite_note-Kaner_1-1)

Software testing can be conducted as soon as executable software (even if partially complete) exists. The [overall approach to software development](https://en.wikipedia.org/wiki/Software_development_process) often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an [Agile approach](https://en.wikipedia.org/wiki/Agile_software_development), requirements, programming, and testing are often done concurrently.

# 1.1 Related Work

Although testing can determine the correctness of software under the assumption of some specific hypotheses (see [hierarchy of testing difficulty](https://en.wikipedia.org/wiki/Software_testing#Hierarchy_of_testing_difficulty) below), testing cannot identify all the defects within software.[[2]](https://en.wikipedia.org/wiki/Software_testing#cite_note-2) Instead, it furnishes a criticism or comparison that compares the state and behavior of the product against [oracles](https://en.wikipedia.org/wiki/Oracle_%28software_testing%29)—principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, [contracts](https://en.wikipedia.org/wiki/Design_by_Contract),[[3]](https://en.wikipedia.org/wiki/Software_testing#cite_note-3) comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions.[[4]](https://en.wikipedia.org/wiki/Software_testing#cite_note-Kaner1-4) The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed.[[5]](https://en.wikipedia.org/wiki/Software_testing#cite_note-kolawa-5)

Every software product has a target audience. For example, the audience for video game software is completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it can assess whether the software product will be acceptable to its end users, its target audience, its purchasers and other stakeholders. Software testing is the process of attempting to make this assessment.

Cem and James Bach teach in their BBST course that to get the technical information from a piece of software, one must ask a question to the program and this question is nothing but a test case.

Some other definitions of test case are- “A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.” (IEEE)

“A test idea is a brief statement of something that should be tested. For example, if you're testing a square root function, one idea for a test would be ‘test a number less than zero’. The idea is to check if the code handles an error case.” (Marick)

Above discussion now takes us to the next point in our journey to understand test case as a product and quality of this product. First, let us see what quality, in general, means to us.

Quality is defined as-Fitness for use (Dr. Joseph M. Juran), Conformance with requirements (Philip Crosby)

Quality is value to some person (Gerald Weinberg).

Detailed discussion about above definitions is out of scope of this paper but these definitions form a good base for defining software product quality.

I tend to say that software product quality is a multi-dimensional measure of meeting requirements of users at an affordable cost.

If we apply above statement to quality of a test case as a product, we could say that it is, multi-dimensional measure of meeting requirements of information objectives at an affordable efforts and cost.

It is extremely difficult to draw a line through all aspects discussed above and come out with a central idea of quality of test case as a product. The practices that we evolve at our organization address our day-to-day needs of meeting the test project objectives and consistently satisfy our stakeholders. However, in the next section, I have attempted sharing my thoughts about how can we engineer test design activity in such a way that the end product is of high quality.

# 2 Test case Design Technique

Following are the typical design techniques in software engineering:

* 1. Deriving test cases directly from a requirement specification or black box test design technique. The Techniques include:
* Boundary Value Analysis (BVA)
* Equivalence Partitioning (EP)
* Decision Table Testing
* State Transition Diagrams
* Use Case Testing
	1. Deriving test cases directly from the structure of a component or system:
* Statement Coverage
* Branch Coverage
* Path Coverage
* LCSAJ Testing
	1. Deriving test cases based on tester's experience on similar systems or testers intuition:
* Error Guessing
* Exploratory Testing

# 2.1 Test Case Engineering

If we call a program as a function of data and operations on data, we treat a test case as function of carefully chosen set of data for the purpose of operations on it for a given information objective.

Test Case Engineering (TCE) starts by breaking down the software product into the three fundamental issues – the Technologies used, the Business domain (or the problem domain), and the Architecture of the product.



Fig 1. Product under Test

Once TCE identifies the three issues well, it grips the whole product with the help of practices in TCE that are depicted in the model below. The practices that one builds in an organization form the Test Case Engineering as a predictive activity to deliver expected quality of test cases.



Fig 2. TCE Practices

Above model clarifies how TCE is a careful thought process to design test cases that will encompass three distinct issues of any software product. The practices employed can be divided into following groups-

* Test management- leadership
* Knowledge Management
* Test designing practices
* Identifying information objectives
* Choosing test data
* Mapping to
* Development model
* Test types
* Test execution techniques- Manual or Automated testing
* Test methodologies- Functional or non-functional testing, White, Gray or Black-box testing
* Test case quality management processes
* Context-driven Reviews
* Test case management system
* Test case version control

TCE helps test teams achieve a balance between expectations, time in hand and quality of work. It helps them decide how and when to trade-off without compromising the objectives of the testing. Each of the practices described above needs a self-correction mechanism in place. The discussion about the self-correction mechanism is beyond the scope of this paper.

# 2.2 test cases

A [test case](https://en.wikipedia.org/wiki/Test_case) normally consists of a unique identifier, requirement references from a design specification, preconditions, events, a series of steps (also known as actions) to follow, input, output, expected result, and actual result. Clinically defined a test case is an input and an expected result.[[6]](https://en.wikipedia.org/wiki/Software_testing#cite_note-50) This can be as pragmatic as 'for condition x your derived result is y', whereas other test cases described in more detail the input scenario and what results might be expected. It can occasionally be a series of steps (but often steps are contained in a separate test procedure that can be exercised against multiple test cases, as a matter of economy) but with one expected result or expected outcome. The optional fields are a test case ID, test step, or order of execution number, related requirement(s), depth, test category, author, and check boxes for whether the test is automatable and has been automated. Larger test cases may also contain prerequisite states or steps, and descriptions. A test case should also contain a place for the actual result. These steps can be stored in a word processor document, spreadsheet, database, or other common repository. In a database system, you may also be able to see past test results, which generated the results, and what system configuration was used to generate those results. These past results would usually be stored in a separate table.

To be most effective and efficient, test cases must be designed, not just slapped together. The word "design" has a number of definitions:

1. *To conceive or fashion in the mind; invent: design a good reason to attend the STAR testing conference. To formulate a plan for; devise: design a marketing strategy for the new product.*
2. *To plan out in systematic, usually documented form: design a building; design a test case.*
3. *To create or contrive for a particular purpose or effect: a game designed to appeal to all ages.*
4. *To have as a goal or purpose; intend.*
5. *To create or execute in an artistic or highly skilled manner.*

Each of these definitions applies to good test case design. Regarding test case design, Roger Pressman wrote:

"The design of tests for software and other engineering products can be as challenging as the initial design of the product itself. Yet ... software engineers often treat testing as an afterthought, developing test cases that 'feel right' but have little assurance of being complete. Recalling the objectives of testing, we must design tests that have the highest likelihood of finding the most errors with a minimum amount of time and effort."

Well designed test cases are composed of three parts:

* *Inputs*
* *Outputs*
* *Order of execution*

Inputs

Inputs are commonly thought of as data entered at a keyboard. While that is a significant source of system input, data can come from other sources—data from interfacing systems, data from interfacing devices, data read from files or databases, the state the system is in when the data arrives, and the environment within which the system executes.

Outputs

Outputs have this same variety. Often outputs are thought of as just the data displayed on a computer screen. In addition, data can be sent to interfacing systems and to external devices. Data can be written to files or databases. The state or the environment may be modified by the system's execution.

All of these relevant inputs and outputs are important components of a test case. In test case design, determining the expected outputs is the function of an "oracle."

An oracle is any program, process, or data that provides the test designer with the expected result of a test. Beizer lists five types of oracles:

* *Kiddie Oracles - Just run the program and see what comes out. If it looks about right, it must be right.*
* *Regression Test Suites - Run the program and compare the output to the results of the same tests run against a previous version of the program.*
* *Validated Data - Run the program and compare the results against a standard such as a table, formula, or other accepted definition of valid output.*
* *Purchased Test Suites - Run the program against a standardized test suite that has been previously created and validated. Programs like compilers, Web browsers, and SQL (Structured Query Language) processors are often tested against such suites.*
* *Existing Program - Run the program and compare the output to another version of the program.*

Order of Execution

There are two styles of test case design regarding order of test execution.

* Cascading test cases - Test cases may build on each other. For example, the first test case exercises a particular feature of the software and then leaves the system in a state such that the second test case can be executed. In testing a database consider these test cases:
	1. *Create a record*
	2. *Read the record*
	3. *Update the record*
	4. *Read the record*
	5. *Delete the record*
	6. *Read the deleted record*

Each of these tests could be built on the previous tests. The advantage is that each test case is typically smaller and simpler. The disadvantage is that if one test fails, the subsequent tests may be invalid.

* Independent test cases - Each test case is entirely self contained. Tests do not build on each other or require that other tests have been successfully executed. The advantage is that any number of tests can be executed in any order. The disadvantage is that each test tends to be larger and more complex and thus more difficult to design, create, and maintain.

As Lee mentions about this book that, “It does, however, contain techniques that will make you more efficient and effective in your testing by helping you choose and construct test cases that will find substantially more defects than you have in the past while using fewer resources.”

# Ross Collard about Good Test case

Apart from the generic discussion that one normally finds in books and articles in test case design, Ross discusses a few practical problems of test teams that affect test design time and again. I have highlighted some of the examples below to keep this reference narrowly focused on only those variations.

After a detailed discussion about test case structure- test condition, test procedure, test results and test environment, he says,

“there is no standard or correct way to document the test cases. Multiple projects will use multiple formats, or a project will use multiple formats. Such variation is justified when test cases need different information or it might be just due to lack of coordination in test teams, which is often the case.”

Then Ross brings out a very interesting discussion about the level of details that a test case may have versus the experienced test team and non-experienced test team, who will be executing the test cases. Then he brings out another important aspect of good test design and test cases- collaboration of insiders and outsiders. According to him, the insiders have usually a very good knowledge of the system under test and they can develop better quality tests from this perspective however, they lack detachment from development team and hence they may lose the advantages of this good design by sheer mismatches due to human relationship dynamics and losing the objectivity of the project.

Whereas, though the outsiders have less knowledge of the system, due to their detachment, they may become highly successful in bringing out ‘breath of fresh air’. However, the main weakness of the outsiders is the depth into the system knowledge and he thinks that this is one of the areas, where they have to work hard to develop good quality test cases.

I have taken following references from Harry’s two papers- “Intelligent Test Automation” and “It’s Different in Test”. Though the papers that he wrote were to address different topic, I found a few very useful tips for test case designing, which I have reproduced below. I have highlighted the important tips in bold letters.

Harry Robinson is Test Architect for Microsoft's Enterprise Management Division. In addition to his day job, he teaches classes on advanced software test automation and is a driving force behind Microsoft's model-based testing initiative. He has been at Microsoft for five years and has a bachelor's and master's in electrical engineering

## Acknowledgments

The systematic approach of TCE can deliver high-quality test cases inherently and consistently if the test organization treats test cases as their product and apply the notion of quality to this very important aspect of the entire test management processes.

There's a saying that "If you all think alike, some of you are unnecessary." If that is true anywhere, it is certainly true among testers. There may be some justification for having a team of developers think in lockstep, but for testers it would be a catastrophe for everyone to think alike. One of the chief benefits testers bring to any group is a different viewpoint, and they should be encouraged to disagree with each other. I get a kick out of watching serious testers talk; it is usually at elevated volume levels.
Actively seek out a good mix of skills and backgrounds when filling out your test team. Do you have coders and non-coders on your team? Each will see different types of defects. How about both genders? Multilingual? Culturally diverse? Differently-abled? One of my favorite true stories is about the Tablet PC team demonstrating its handwriting recognition function to Bill Gates. The team was very excited and confident, but they had overlooked a big factor. The Tablet PC team was almost all right-handed, but Gates is a lefty. Guess what? The handwriting recognition didn't work well for him.
 **REFERENCES**

[Kaner, Cem](https://en.wikipedia.org/wiki/Cem_Kaner) (November 17, 2006). ["Exploratory Testing"](http://www.kaner.com/pdfs/ETatQAI.pdf) (PDF). Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL. Retrieved November 22, 2014.

[Software Testing](http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/) by Jiantao Pan, Carnegie Mellon University

Leitner, A., Ciupa, I., Oriol, M., Meyer, B., Fiva, A., ["Contract Driven Development = Test Driven Development – Writing Test Cases"](http://se.inf.ethz.ch/people/leitner/publications/cdd_leitner_esec_fse_2007.pdf), Proceedings of ESEC/FSE'07: European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering 2007, (Dubrovnik, Croatia), September 2007

[Kaner, Cem](https://en.wikipedia.org/wiki/Cem_Kaner); Falk, Jack; Nguyen, Hung Quoc (1999). Testing Computer Software, 2nd Ed. New York, et al: John Wiley and Sons, Inc. p. 480. [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [0-471-35846-0](https://en.wikipedia.org/wiki/Special%3ABookSources/0-471-35846-0).

Kolawa, Adam; Huizinga, Dorota (2007). [Automated Defect Prevention: Best Practices in Software Management](http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470042125.html). Wiley-IEEE Computer Society Press. pp. 41–43. [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [0-470-04212-5](https://en.wikipedia.org/wiki/Special%3ABookSources/0-470-04212-5).

IEEE (1998). [IEEE standard for software test documentation](https://en.wikipedia.org/wiki/IEEE_829). New York: IEEE. [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [0-7381-1443-X](https://en.wikipedia.org/wiki/Special%3ABookSources/0-7381-1443-X).

**saleem al\_zoubi** is an associate professor at the Ibid National University. He received his BS in Elec-Engineering \Communic from the University of Philadelphia in 1999 and MS degrees in Engineering/ Computing Technology from the University of Dnepropetrovsk 2001, respectively, and his PhD degree in Computing Technology and Control System in 2004. He is the author of more than 20 journal papers and has written three book chapters. His current research interests include data mining, artificial intelligence, and knowledge systems, Software Engineering.