Testing of software development using math and computational physic

Testing of software development using math and computational physic

J P Rodríguez[1], **M A Adarme**[1], **and O A Gallardo**[1]
[1] Universidad Francisco de Paula Santander, San José de Cúcuta, Colombia

E-mail: judithdelpilarrt@ufps.edu.co, madarme@ufps.edu.co

Abstract. In the perspective of math and physical-computational, software testing is conceived as a modeling of scenarios based on the different levels of abstraction that software is developed. These specifications allow marshaled representation of each software component as well as its intra - extra process interactions. This work analyzes in a systems approach under the perspective of computational physics how to identify, interpret, represent, and formally model functional software tests and how, through critical path representations, white box testing is developed. A test scenario is established for components built in the software called "Ferticacao," where white box testing is applied. Results show that the use of modeling techniques based on critical paths show the interaction of the code and its interoperability at the level of data structures and component calls that allow to quickly analyze the functionality of the software at the source code developing level.

1. Introduction

Since the beginning of computing, the goal of the first computers was to solve problems and equations impossible for humans. This relationship between mathematics and engineering is by no means new, but it has intensified considerably as the digital economy has become more specialized, and different disciplines have been perfected. Systems Engineering is the application of mathematical and physical sciences to develop systems that economically use the materials and forces of nature for the benefit of humanity. Thus, mathematics has a formative character for an engineer since it serves as an instrument to develop fundamental skills in his formation, such as writing, formalizing, acquiring skills to face new situations, precision, and constancy.

Mathematics has several useful properties for software developers. One is the physical situation, an object, or result of a Pressman [1] action. For a software engineer, the development of a specification is fundamental to employ specialized mathematics, just as an electrical engineer does. These specifications can be mathematically validated for contradictions and eliminate vagueness. A software engineer uses them to represent, in an organized way, degrees of abstraction in the specification of a system and model it, avoiding ambiguity. Using mathematical analysis is one of the options for software engineers since it allows them to know the behavior of products in the real world. However, mathematics as the discrete includes the knowledge of Mathematical Logic and its formal systems, brings together other aspects that, although essential for an engineer (for their training and mental structure, in the construction of criteria), contributes to the practice in students of generic skills such as a capacity for analysis and synthesis, capacity for planning and programming, oral and written communication, capacity for information management, problem-solving, decision-making, teamwork, critical reasoning, autonomous

learning, adaptation to new situations and creativity, motivation for quality and continuous improvement and capacity to apply knowledge to practice, Machinery-ACM & Society [2].

Thus, mathematics offers the engineer a high degree of reliability when using validation methods in software development. In other words, mathematics can be applied through models to demonstrate that a software design corresponds to a specification, and its source code is the correct reflex of the plan [1]. Based on this preliminary analysis, this research is focused on the design of a formal case to develop functional tests and how math has been developed as a science that has a relationship with necessary route tests, the white box testing technique initially proposed by Tom McCabe in 1976. To carry it out, it was applied, from a holistic perspective, starting from an example used to a project in which actors and academic interests are articulated, a method that allows the software engineer or designer of test cases to obtain a measure of the logical complexity of a procedural design that can be used as a guide to define a basic set of execution routes.

The article has been organized as follows: section 2 describes the theoretical framework underpinning this research; section 3 presents the advantages of using formal models applying software tests, section 4 shows the application a test case, and finally, section 5 establishes the conclusions based on the results obtained.

2. Theoretical framework

Software engineers did not apply rigorous mathematical methods to their products years ago. They resorted to empirical verification methods, processes that did not guarantee the absence of errors, and in which they allowed their software products to work and directly observe their behavior, making it tedious and generating more errors each time they were debugged (location and correction of defects). The reliability of the product increased throughout the development process. This work had to be developed in parallel with product development at each stage of the software lifecycle.

Gibbs [3] quoted by Pressman, "the ability of engineers to measure the reliability of their products is less than necessary in the quality approach, so it is desirable that engineers can mathematically demonstrate the correctness of their programs, in the same way that other branches of engineering can do". But what are formal methods? The concept of formal methods involves a series of logical techniques with solid mathematical bases, usually provided by an official language of specifications, with which information systems can be designed, implemented, and verified, Monin [4]. Formal methods use set theory notation and logic to create a clear statement (requirements), *i.e.*, a mathematical specification (math) is created and analyzed to improve (prove) its correctness and consistency; being the most rigorous formal methods, they apply techniques to check the arguments and justify the requirements, or other aspects to be dealt with in the design and implementation of the system.

They are properties of a formal specification (consistency, integrity, and unambiguousness, the first two are software quality metrics, while unambiguousness can be understood and interpreted in more than one way) are the objectives of all specification methods; however, in employing formal methods, the probability is much higher than these ideals. Applying formal methods produces a specification represented in a formal language; thus, formal methods use discrete mathematics as a specification mechanism, and apply logical tests to each function of the system to demonstrate that the specification is correct.

2.1. Formal methods and software testing

In software engineering, formal methods are used to: ensure security policies and properties such as reliability or data integrity. To describe the specification of system behavior, to develop matching tests between the specification and requirements (were manual or automatic rigorous tests), to create tests between the source code and the specification. This means that formal methods in software engineering are mathematically precise.

The cost of using formal methods is generally used in the development of critical high integrity systems. But formal methods are useful for software testing because they help avoid errors and can also provide a framework for testing, which requires the support of tools. One of the challenges for software

developers is to perform reliable testing, discarding preconceived notions of what is right, the engineer designs test and test cases; before delivering the product to the customer, it must be tested, which implies that it must have undergone reviews and other software quality assurance activities and discover and eliminate errors, but this is not enough. The software engineer must locate a large number of errors and apply tests systematically and design test cases using software testing techniques.

Myers [5] indicates that when we test software, it requires an added value to what we are checking, raise the quality and reliability, and this leads us to have to find and eliminate errors in the software. Researchers, such as Myers, propose separating software development from verification and validation; from this approach and following Deming [6], one option to fulfill this premise is to articulate the life cycle with quality improvement. This means that we do not have to test software to prove that it works, but we have to start from the assumption that the program is going to have errors.

Dijkstra [7], a computer scientist, defined the testing process as: "software testing can be an effective way to show the presence of errors, but they are inadequate to explain their absence. Every one of the definitions has in common that they focus on more or less error detection.

2.2. Advantages of using software testing in formal methods

Specification. Formally developing a specification requires detailed and precise knowledge of the system, which helps to expose errors and omissions, and therefore the greatest advantage of formal methods is given in the development of the Clarke & Wing specification. In the formalization of the description of the system, ambiguities and omissions are detected, and a formal specification can improve communication between engineers and clients. Formal methods were developed primarily to allow better reasoning about systems and software [8]. After designing a formal specification, it can be analyzed, manipulated and reasoned about.

Verification to ensure the quality of systems you need to test them, and to ensure that rigorous tests are developed requires an accurate and complete description of their functions, even when formal methods are used in the specification [9]. One of the most interesting applications of these is the development of tools that can generate complete test cases from the formal specification. Although a great number of tools to automate tests are available in the market, they automate only the simplest aspects such as: they generate the test data, enter that data into the system and report results. Defining the correct system response for a given set of input data is an arduous task that most tools cannot accomplish when the system's behavior is specified in natural language. Because the expected response of the system can only be determined by reading the specification.

Empirical measurements have been claimed to show that tests [10] generated with automatic tools provide good or better coverage than manually obtained, so engineers choose between producing more tests in the same time or reducing the number of hours needed to perform them.

Validation is a set of different activities that ensure that the software built corresponds to the customer's requirements. While verification can be performed semi-automatically and tests mechanically. A specific difference between verification and validation is that the first responds to whether "the product is being built correctly", and the second to whether "the right product is being built". From the set of requirements, it is possible to verify, formally or informally, whether the system implements them; however, validation is necessarily an informal process. Formal methods are applied especially in large and complex applications, such as modeling and simulation. Thus, when modelling requirements, there may be conflicts if they are not formally designed by not determining and exploring their properties [11].

3. Results

Results show that the use of modeling techniques based on critical paths show the interaction of the code and its interoperability at the level of data structures and component calls that allow to quickly analyze the functionality of the software at the source code developing level (Algorithm 1).

FERTIL-CACAO is a web application for the fertilization of cocoa that will provide different services to cocoa farmers, facilitating the processes of fertilization and registration of soil analysis and

can access the system at any time and place. This application will allow companies to keep a record of the fertilization process of the evolution of the plant through the fertilizer used (which is surplus or lacks nutrients). For the software web was necessary to apply various testing strategies and quality metrics for ensuring that the software is quality requirements.

In the field of software engineering a metric is any measure or set of measures intended to know or estimate the size or other feature of a software or an information system, usually to make comparisons or for planning development projects. one could say that the concept of software metric refers to the continuous application of measurements based on techniques for the development process of software and its products. Some utilities include estimating test cases, understanding productivity ranges, understanding project growth, calculating the real cost of the software, and estimating project cost, programming, and effort.

In research, white box or structural technique tests are applied, which are based on an exhaustive examination of the procedural details of the code to be evaluated [12]. It is necessary to know the logic of the program and then describe one of the basic route or coverage technique tests applied in the development and implementation of the FERTIL-CACAO nutritional monitoring and control platform. It is necessary to apply different tests in all the methods performed, in such a way as to correct the errors that occur during the software life cycle. The basic path technique is based on cyclomatic complexity measurement which is a software metric that provides a quantitative measurement of the logical complexity of a program.

Algorithm 1. Code source.

```
Función validarLogin ($usr,$password,$rol)
{
    1       If ($rol==1) {
            $usuario=new Usuario ();
            $usuarioDAO=new Usuario_DAO ();
            $usuario->setcedula($user);
            $usuario->setpassword($password);

    2       $resultado_$usuarioDAO->Login($usuario);
    3       if ($row_$usuarioDAO->getArray($resultado)){
                4       return $row;
            }
                        else {
                5       return false;
                        }
            }
            6           If ($rol==2) {
                        $administrador = new Administrador();
                        $administradorDAO = new Administrador_DAO();
                        $administrador ->setcedula($user);
                        $administrador ->setpassword($password);
            7           $resultado_$AdministradorDAO->Login($administrador);
            8           If ($row=$AdministradorDAO->getArray($resultado)) {
            9           return $row;
            }
                        else {
            10          return false;
                        }
            }
}.
```

It calculates the number of independent paths in the basic set and provides an upper limit for the number of cases necessary to execute all instructions at least once. Remembering a path is a sequence of sentences chained from program entry to program exit. Enough test cases are written for all or some of the paths in a program to be executed. You can have criteria such as: coverage of all roads, coverage of sentences, branches, predicates, basic route. The procedure to apply the technique is: represent the method in a flow chart, calculate cyclomatic complexity, determine the set of independent roads and derive test cases. Below, only describes the method validate login method, responsible for validating the login by a specific user type, for this case cocoa farmer and web system administrator complexity, determine the set of independent roads and derive test cases.

Once the code is developed, the nodes that make up the network are numbered, only the executable sentences are listed as shown in Algorithm 1 and Figure 1. Cyclomatic complexity is calculated, which is performed with the following formula in Equation (1), Equation (2) and Equation (3):

$$\mathrm{V(G)} = \text{Number of regions} \tag{1}$$

$$\mathrm{V(G)} = \mathrm{ARTISTS} - \mathrm{NODES} + 2 \tag{2}$$

$$\mathrm{V(G)} = \text{Number of predicates nodes} + 1 \tag{3}$$

According to the example given, we have whit: V (G) = Number of regions = 5, V (G) = SURGEN - NODES +2 = 15 – 12 + 2 = 5 and V (G) = Number of predicate nodes + 1 = 4 + 1 = 5. This value indicates the lower level of the number of tests that will be carried out to test at least all the existing paths passing at least once for each node and once for each edge of the graph made. In this case it would be five roads. Now we proceed to perform these five paths as shown in Table 1.

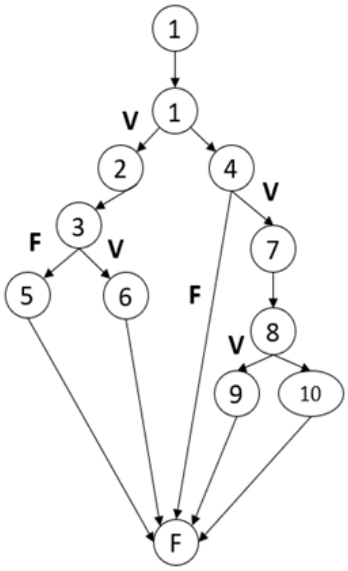

Figure 1. Control flow graph.

Table 1. Control flow graph specification.

Path	Input	Output
1, 2, 3, 4, F	Condition 1 = True Condition 3 = True	Return object $row
1, 2, 3, 5, F	Condition 1 = True Condition 3 = True	Return “false”
1, 6, 7, 8, 9, F	Condition 1 = False Condition 6 = True Condition 8 = True	Return object $row
1, 6, F	Condition 1 = False Condition 6 = False	Return “false”
1, 6, 7, 8, 10, F	Condition 1 = False Condition 6 = True Condition 8 = False	Return null

4. Conclusions

Software testing is directly related to the field of mathematics, and this is evidenced by the use of testing techniques that, through computational physics, offer mathematical models of validation through graph theory. Its interpretation and application make the Software Engineer has a piece of comprehensive knowledge in this area. Formal specification techniques and mathematics have been little used in software development companies; however, its applicability has advantages over less formal methods, that is, some tests can easily detect execution errors according to their requirements.

References

[1] Pressman R. 2010 *Ingeniería del Software: Un enfoque Practico, Séptima edición* (México: Mc Graw Hill)
[2] Association for Computing Machinery (ACM) and IEEE Computer Society 2013 *Computer Science Curricula 2013. Curriculum Guidelines for Undergraduate Degree Programs in Computer Science* (United States of America: Association for Computing Machinery and IEEE Computer Society)
[3] Gibbs W 1994 Chronic software crisis *Scientific American* **218** 72-81
[4] Monin J F 2003 *Understanging Formal Methods* (Nueva York: Springer-Verlag)
[5] Myers G 2014 *The Art of Software Testing* (Nueva Jersey: John Wiley & Sons, Inc.)
[6] Deming W, Medina J N, Gozalbes M 1989 *Calidad, Productividad y Competitividad* (España: Ediciones Diaz de Santos)
[7] Dijkstra E 1976 Structured programming *Software Engineering, Concepts and Techniques* ed Buxton J, et al. (Nueva Jersey: Van Nostrand-Reinhold)
[8] Hinchley M, Bowan J 1997 *Applications of Formal Methods* (New Jersey: Printice Hall)
[9] Murata T 1989 Petri nets: properties, analysis and applications *Proceedings of the IEEE* **77(4)** 541-580
[10] Farrell-Vinay P 2008 *Manage Software Testing, 1st edition* (Boca Raton: Auerbach Publication)
[11] Sommerville I 2002 *Ingenieria de Software, Sexta edición* (Mexico: Pearson Education)
[12] Pantaleon G, Rinaudo L 2015 *Ingenieria de Software*, *Primera edición* (Buenos Aires: Alfaomega Grupo Editor)