International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 12, Number 3 (2019), pp. 321-332


# Testing the correctness of Educational Software System Based on Testmatica Model to explore its impact on productivity gains.

**Eze Nicholas Ude, Obichukwu Peter Uzochukwu*, Ibezim Nnenna Ekpereka**

*Department of Computer and Robotic Education, University of Nigeria, Nsukka.*

**Abstract**

**Objectives**: *This study evaluates various technical developments in testing that are currently advanced as potential breakthroughs in determining the correctness of software system and its correlation with productivity gains.*

**Methods/Statistical analysis**: *TESTMATICA testing Model was developed for testing the correctness of the entire structure of software using phase-to-phase approach. Thereafter, quantitative research design technique was adopted where 5 software firms across Africa and G-8 countries were randomly selected to determine the impact of the model on productivity gains. Questionnaire was used for data collection and lone hypothesis guided the study.*

**Findings:** *Findings reveals that the test based on the design using testmatica testing model when evaluated against other testing approaches can be a more powerful tool for checking if all the entire structures at the design level of many software systems are correct. On productivity gains, the findings reveal that testmatica testing model improves customer satisfaction, speeds up the development process and improves productivity of software development teams.*

**Novelty:** *The use of testmatica model is very unique and novel because it deals with the entire structure of the design on a phase-wise basis; and not just only the control structures of software as reported in literature.*

**Keywords:** *Correctness proofs, Productivity gains, Software system, Testing, Testmatica model.*

## I. INTRODUCTION

Research in software engineering have shown that the greatest threat to the traditional method of software development endeavor is the high amount of errors detected when testing is conducted, [1, 2]. This is because the method by which software is developed often determines the time and how testing is carried out. For example, in a traditional waterfall model of software development, testing is carried out only after the design of the entire software, [3]. But, under an agile programming [4] approach, requirements, programming, and testing are often conducted at the same time, [5].

However, because testing is conducted only during the final stage of software development in a traditional waterfall process, problem is bound to occur especially when it takes a very long period of time before an error that occurred at the earlier phases of the process model is detected, [3, 6]. More so, since project managers are always overanxious in placing targets on task duration times, implementation of projects cannot be concluded without a problem when the software is finally released, [7] as software test coverage are always low and only few testing are conducted with several faults detected, [8].

Even when there is enough time to test the software, not every faults are detected when the entire code base are examined [9, 10]. This is because all the test codes are examined at once and no one will know how faulty the implemented code is until the date and time it will be released [11]. Then when the functionality of the software is tested, a latent and reasonable number of wrong results may be detected and this could lead software developers into heavy loss in terms of increased cost, depreciated profit, delayed delivery of software product, clients dissatisfaction and project deviating from earlier budget, [12, 13].

However, the entire problem is not unsolvable [14, 15, 16, 11]. For example, it is far too late to test a software product only at the time of delivery to the client without carrying step-wise check of every step taken to develop it. This is because if the software developer makes a mistake while eliciting and documenting the requirements, then the remaining phases of software development (the specifications, the design, the code and so on), will all be affected [17].

Faults should therefore be detected as earlier as possible because the earlier a fault is detected; the cheaper it is to fix [17]; though it depends on what has to be done to fix the fault. Let's say, if a problem that is supposed to be discovered earlier at the requirement or the specification stage (as requirements, analysis, and design stage faults constitute over 60% of all faults [18, 19] is found only at the post delivery stage, then it would consume huge amount of resources (money, time, etc) to correct than if the fault had been detected at the earlier stages.

Testing and proofing of correctness of software is therefore very important in software development not only because it help to discover faults earlier but it also help to reduce the cost and time of delivery of software products. Secondly, software are developed by humans who because they are not perfect could make mistakes. Without proper testing and proofing of correctness of software, there will also be of less value in terms of cost, time, profit, customer satisfaction etc for any software development group to discover after several months of developing a software that a single mistake they made at the

*Corresponding Author: Obichukwu Uzochukwu Peter

early phases of the development process will cause them to redesign the entire software.

Because of this, phase-wise testing is so touted in this study as it requires over 70% of software development effort in other to achieve software that one can rely on, [20]. To achieve this reliability will require that the requirement must be tested; the specification document must be tested; the design also must be tested; and so on, [21]; and each of these phases must be certified error free by the software quality assurance team before moving to the next phase (see Figure 4 in section II).

However, based on the literature reviewed so far, it is so glaring that because of when and how testing is conducted, there are always errors when the software is finally released [11]. Trying to correct such errors could lead to increased cost and budget and clients not being satisfied because there is always delay in delivering software products. All these problems have persisted and what researchers are only interested on is getting empirical data to know if software testing strategies are reliable or not, [14]. This is the rational for this study since the existing problems have remained without addressing it.

To start addressing the problem, this study first dealt with phase-wise testing of the design structures of software and how the testing techniques and methods suggested can be used in every phase/iteration of software development to achieve error free software. These techniques when applied will go a long way in solving the problems. Secondly the study introduces a new and better model – (the Testmatica Model) for proofing the correctness of the control structures of software. This is to support Brooks's in his article, [7] 'no silver bullet', that a major potential silver bullets in software development, is to prove its correctness.

Our discussion then proceed to an examination of the perceptions of testing by different groups on productivity gains and we conclude by reporting their experiences in applying testing methods to determine the productivity gains in a business context.

## II. LITERATURE SURVEY

This section reviews previous research from different scholars on software testing in correlation with productivity gains and consequences of lack of it. The aim is to enable the readers gain more insight on testing and its impact on productivity gains. We then examine the various testing methods and techniques employed at each phase of software development. We further introduce a model for proofing the correctness of a software product based on TESTMATICA MODEL.

### AN OVERVIEW OF PRIOR RESEARCH

So many success stories have been heard of software development. Of these are that software development has reduced radically the rate at which firms across the globe fail in business. Despite these, a huge amount of software products are still not delivered within the stipulated time frame and when it is delivered, a lot of errors are detected, [12]. A very good example is a Standish Group study involving 8,380 development projects which was completed in 2006 with 365 respondents on the adoption of agile methodology in software development process **a**s summarized in Figure 1 [22].

The major findings of this study was that, of a hundred percent success rate with respect to timely completion and within budget successful delivery of projects, only a meager 35 percent of these projects were successfully completed and delivered within stipulated time and budget. The rest 65 percent were not even channeled on completing and delivering the entire projects; rather 19 percent of these projects were terminated at one point while the product is still being developed or were never implemented at all. Then only 46 percent of the software product were completed and handed over to the client.

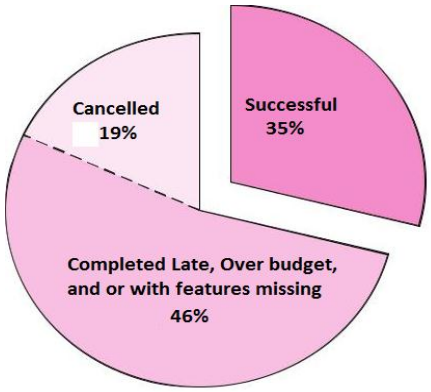


**Fig. 1 The outcome of over 8,000 development project completed in 2006. [Rubenstein, 2007]**

What this means was that the initially specified features and functionality of these projects were no longer maintained as projects were seen delivered over budget and late. Hence, in 2006, the success rate of software development endeavor was just one third with over 50percent of the projects showing so many symptoms of the software crisis which was due to improper software testing.

The financial implications of these failures and crises are horrendous on both the firms and the customer. For example, research has shown that late delivery of a software product could lead to serious legal action which is capable of sapping huge amount of resources from clients and firms, [23]. However, Cutter Consortium in 2002 reported that:

: A surprising 78 percent of software development firms have settled their disputes in court.

: 67 percent of the issues lies in the fact that software developers failed to measure up with the functionality and performance of the software products as delivered.

: In 56 percent of those cases, the actual dates for delivering of the product were not met.

: In 45 percent of those cases, there were so many errors on the software that could not be managed hence not usable.

All these reports are not healthy for successful software development considering its many success stories as they come with heavy cost implications.

Corroborating, a study conducted by National Institute of Standards and Technology (NIST) in 2002, also shows that one of the greatest threat to the U.S. economy is software bugs. According to the report, the United State loses a whopping $59.5 billion annually as a result of this. The study however suggested that better software testing could save more than a third of this cost[24].

The above suggestion is in agreement with the result of the study as reported in [25], which stated that while developing an operating system, proper testing can bring down the amount of error by 85 percent, [26].

The Jet Propulsion Laboratory (JPL) also corroborated when they reported that over $25,000 is saved per inspection of software as every little inspection reveals at least 4 major faults and 14 minor faults [27].

However, an Australian group, [28], conducted a survey involving 131 respondents of software development groups and firms that used testing methodologies in their software development projects and their report shows that, productivity was improved by 93%; development cost was reduced by 49%; 88% quality and customer satisfaction was achieved; and 83% business satisfaction was equally great.

Further research also reveals that conducting a thorough testing on software can lead to an increase in the overall quality of the product and decrease the cost of the product, [29].

A comparative study was conducted between 1997 and 2007 to determine the perceived influence of testing methods and techniques used by different software development firms on productivity gain in the G8 countries, [30]. Their findings as shown in Figure 2; is the discovery that companies that use some degree of testing techniques have more satisfied employees and clients. The perception of University students is that use of testing methods avails them more experience and relevant training. On productivity gains, the study found that there is a greater increase in productivity gain if testing is thoroughly used while developing a software product.

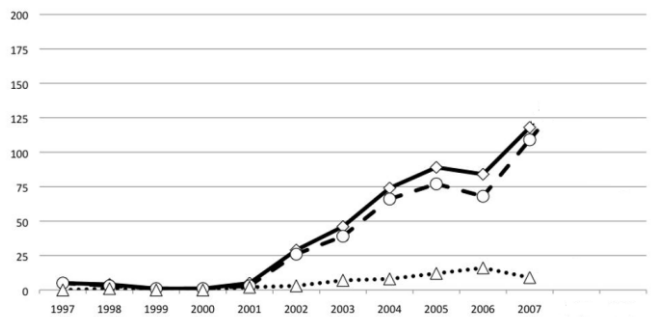


**Fig. 2 Customer satisfaction total numbers (top), employee satisfaction (middle) and other user's satisfaction (bottom).**

Summing up the literature review, testing is a nice tool for software development. This is because it comes with a lot of benefits for the software development team, as well as business benefits to the client, [7].

Secondly, testing should not be seen as a separate phase of software development. For example, consider Figure 3.

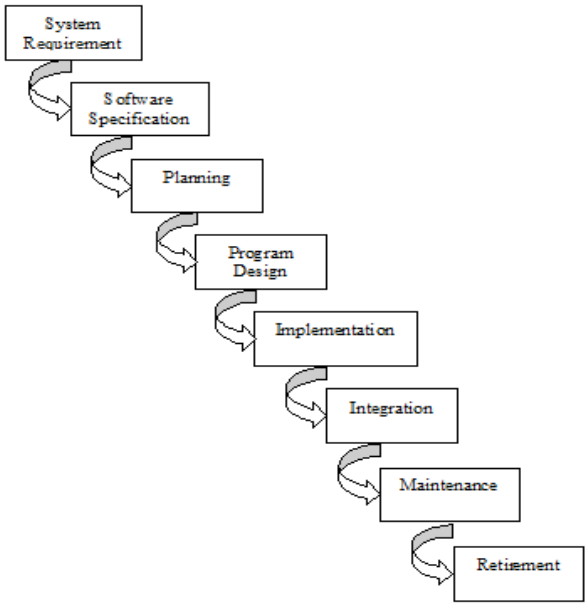


**Fig. 3. Software Development Life cycle**

In the preceding list of phases of software development in Figure 3, notice there is no separate testing phase. This omission is deliberate for the purpose of this study. This study is advocating that testing rather should be an activity that should take place all the way through software production, [21, 5] (see Figure 4)


Fig. 4 Software Testing Diagram

More emphases should be placed on testing more than other activities in software development. This normally should take place at the end of every phase or iteration (verification); and also before the product is finally handed down to the client (validation), [31, 6]. This is to verify if system behaves according

to specification, and to check the system correctness to ascertain if the clients' requirement as specified actually met the desired output.

Note that this verification could be achieved in two ways, 1) Test Mode, and 2) walk- through mode [32]. Under the test mode, we first generate the correct operational sequences of the design according to the requirement specification and semantics of operations. Then the true input responses are generated and compared with the correct version. Example see Figure 5.

```
1:    ignore;  1
*:    empty-ab; I: ignore
 *:   acc-ab;
 4): acc-ab;
 Case input of  * /empty-ab
(*:   acc-ab)  2 3
(1:   acc-ab)  *  acc-ab
```

**Figure 5. Input response in comparison with correct version**

In walkthrough, input sequences from the design and their corresponding responses can be taken as "path programs" where its correctness could be achieved by a walk-through procedure based on the specification by applying the testmatica model as will been seen later in this study. During this step, one may discover that there may be ambiguity or incompleteness in the specification; and also not all specification errors can be revealed. But the application of the testimatica model must establish that the input specification satisfies the output specification.

However, although there are times when testing predominate, there should never be times when no testing is carried out else it will have a high negative impact on productivity gain [11].

For these reasons, the testing of each phase of software development process must be certified error free by software quality assurance team of the firm,[33] before moving on to the next phase. This will make software development 'more agile' which is the best for addressing the problems of the traditional waterfall, incremental and spiral model of software development, [4].

## STRATEGIES FOR TESTING SOFTWARE

To achieve good software testing, one needs to get acquainted with the best testing strategies capable of integrating various software test case design methods into a well organized series of steps. Our earlier review of literature shows that software testing strategies are necessary for testing and these are developed by project managers, Software Quality Assurance Engineers and individuals who are specialist in testing. This study has identified and employed four software testing strategies in other to draw inference from reports across firms that use testing techniques as an agile method of software development. They include:

- **Unit testing**

This type of testing is conducted only at the smallest level often referred to as "unit", "iteration", "module", or "component". Here, the entire program is broken down into separate, smaller sections called modules, units, subroutines, subprograms as they are interchangeably referred to. Each module has a specific job to do and is relatively easy to test each module to verify the functionality of a particular section of the code. This type of testing is generally grouped into a white box test class. However, in an object-oriented programming environment, this is usually at the class level in which the constructors and destructors constitute the unit test, [34].

- **Integration Testing**

In integration testing, program codes to be tested are organized within various control structures and then merged as one bigger entity which has direct interface with the control structures. Testing is then conducted on the interfaces in other to discover faults in the interfaces and interaction between the merged components (modules). The testing and integration of the larger groups of software components corresponding to elements of the architectural design continues gradually until the software works as a system [35] (See section C for more details).

- **System Testing**

In system testing, both the functional and requirement specifications including the behavioral properties of the entire system are tested so as to know if the product functions correctly, [36]. The SQA group first look at these properties, in parts, run it with known input data, and examines the output, [37]. The SQA simply does this by inputting intentionally erroneous data into the system to check the functionality of the product or if the mechanism for detecting faults is still working perfectly to detect faults incase bad data is inputted into the system. After this, the internal consistency of the product source code will be tested to know if the new inputted product will have any form of effect on the client's existing computer operations.

- **Acceptance Testing**

Acceptance testing precedes Integration testing as it is the testing that stands to give complete assurance that the entire system is perfectly working without any residual fault, [38]. This type of testing is done at the time when the complete design artifact is to be handed over to the client by the developer. In other words, without a software product passing through its acceptance testing, it can never be said to be correct or meet its specification, [11].

However, in the course of this testing the SQA must ensure that the portability of the system as well as working as expected should protect the systems operating environment from damage to avoid causing other processes within the environment to become inoperative [39].

## THE TESTING PROCESS OF SOFTWARE DEVELOPMENT

In this section the seven phases of the software life cycle, are carefully analyzed and the role played by software quality assurance team during testing of each phase to improve software quality by detecting software bugs were explored. The first phase is the requirement phase.

### Requirement Phase Testing

Every software development organization has Software Quality Assurance (SQA) group [33]; who ensures that the quality of a software product that is delivered to the client met with the specification of the client. This is so because; the quality and the correctness of any software product depend on the extent to which it meets its specification [37]. In that case, the SQA is set up to enforce these standards and must play a role right from the beginning of the software development process. In particular, it is important that clients get satisfied when the product is finally handed over to them by first crosschecking with the client to know if the rapid prototype of the software reflects their current needs.

Nevertheless, no matter how meticulously this is done, there is always the possibility that forces beyond the control of the development team will cause changes to happen during the design process. Further development will then have to be put on hold until the necessary modifications have been made to the partially completed products.

- **Specification Phase Testing**

Before the specification phase can be deemed to be finished, the SQA group must carefully check the specifications, looking for contradictions, ambiguities, and any sign of incompleteness, [40]. In addition, the SQA group must also certify the capacity of the specified hardware and online disk storage in handling the new products.

If a specification document is to be testable, then one of the properties it must have is traceability where every statement in the output specification document is traced back to every statement in the input specification document. If the requirement has been methodically presented, then the SQA group will have fewer jobs tracing through the specification documents. If rapid prototyping has been used in the requirement phase, then the relevant statements of the specification document should be traceable to the rapid prototype.

However, a way out for checking the specification document is by means of a review where walkthrough or inspection could be used. During this, both the SQA team and the client meet to determine the correctness of a specification document. The specification documents are reviewed, ensuring that there are no misunderstandings about the documents.

- **Planning Phase Testing**

In this phase, a Software Project Management Plan (SPMP) is drawn and carefully checked by the SQA team. The SPMP contains detailed plan on delivery date and an estimate of the cost for developing the software.

To test the planning phase, software development firms obtain more than one independent quotes of both delivery date and cost at the commencement of the planning phase. If there is any difference in the two quotes, such differences is reconciled through the process of a review similar to the review of the specification document. This review will make it possible for software development cost and delivery date not to slip away.

- **Design Phase Testing**

Testing the design phase of software requires traceability where every aspect of the design are traced to a statement in the specification document. This design review is similar to the specification reviews as it helps the SQA team to check and know if the actual design conforms to what is specified or that whatever is in the specification document reflects with what is in the design.

During the design review, clients need not be physically present because of the technical nature of the design. Here software development team and the SQA team work through the entire design and each separate module, looking for logic faults, interface faults, etc all with the intent of achieving a correct and perfect design. In addition it is important that the review team should know that some specification faults were not detected during the previous phase before the design and so pay more attention to discover such faults during the design phase.

- **Implementation Phase Testing**

The modules should be desk checked by the programmer during implementation and also tested immediately they have been implemented and run against test cases. After this informal testing, the SQA team then performs methodical testing on the modules.

Code review could be employed in detecting programming faults by the programmer and the SQA representatives. This procedure is similar to reviews of specifications and design described previously.

- **Integration Phase Testing**

Integration phase testing ensures that various units or modules are combined correctly in other that the entire product meets with its specification. Here module interfaces are meticulously tested and checked by the compiler and the linker to ensure that number, order, and types of formal arguments matches with number, order, and types of actual arguments.

The SQA team carries out product testing [37] at the completion of the integration testing during which the entire product is

tested against its specification. In particular, the constraints listed in the specification documents must be tested to ascertain if the correct specifications have been implemented. Only after this are the test cases drawn up.

In some cases, early versions (alpha version) of the complete product are sent to the client for testing on site. After correction, this corrected alpha version, now called the beta version is intended to be close to the final version. Alpha testing and beta testing are particularly important during software development [7].

- **Maintenance Phase Testing**

From the foregoing, once the software developer has satisfied that the desired changes have been implemented correctly, then the software product must undergo regression testing, [3] to ascertain if the functionality of the rest of the product has been compromised or not. Here all the previous test cases are retained, together with the results of running those regression tests. If it is discovered that the product has been compromised, then fault could be easily traced and corrected.

## A. TESTING THE CORRECTNESS OF SOFTWARE SYSTEM

According to Dijkstra, "software testing can show the presence of faults, but definitely not the total absence of it," [10]. What this means is that, when a product undergoes some testing and the output did not meet the specification, then that product is wrong. On the other hand if a product is tested and the output met the specification, the product may likely still have some faults in it.

For this reason, there must be some behavioral attribute of the software which needs to be tested to ascertain whether the program still shows the presence of fault or not, [37]. These behavioral attributes ranges from reliability, performance, robustness, utility, and correctness. However, for the purpose of this study, the behavioral attribute of software that will be considered is correctness.

- **Correctness**

A product is simply said to be correct [37], if the input specification satisfies the output specifications. This definition has big worrisome implications. For example, there is no law that says a product must be accepted simply because the product has been successfully tested against a broad range of test data, [32]. If a product is correct and should be accepted, that simply means that the output specifications were met. But what if the specifications themselves are incorrect? Consider the following illustrations

| | |
|---|---|
| *Input specification:* | p : array of n integers, n 0. |
| *Output specification:* | q :array of n integers such that<br>q[0] q[1] ··· q[n 1] |

FIGURE 6 Incorrect specifications for a sort.

```
void trickSort (int p[ ], int q[ ])
{
    int i;
    for (i    0; i    n; i    )
        q[i]   0;
}
```

FIGURE 7 Method trickSort, which satisfies the specifications of Figure 6

| | |
|---|---|
| Input specification: | p : array of n integers, n0 |
| Output specification: | q : array ofn integers such that q[0]q[1]-q[n1]<br>The elements of array q are a permutation of the elements of array p, which are unchanged. |

FIGURE 8 Corrected specifications for the sort

Illustrating the above difficulty, one may think that the specification in Figure 6 is correct simply because, may be when the product was tested against a given set of data, it runs correctly. But it is not. The specification have only created a specification gap by showing that the input to the sort is an array p of n integers, and the output is another array q sorted in non decreasing order; without recourse that the element of q and the output array are all a permutation of the element of the input array p.

The second method in Figure 7, tricksort then capitalizes on this specification fault and tries to correct it by setting all n elements of array q to 0 as in Figure 8. Figure 8 is then the corrected specification for the sort. This example has set the record straight that it is no meaning claiming that a product is correct when its specification is incorrect.

In other words, it is better to look at what accounted for the correctness of a product rather than just showing that the product is correct. Therefore it is necessary to prove that a product specification is correct. This proof which is a mathematical technique should be carried out in conjunction with step by step coding and the design [10], after which it is tested to know if the coding and design is equivalent to the specification and therefore is correct [11].

To prove the correctness of software, [32] proposes the automata theory for testing just the control structures of software. Their result only showed that the method is correct. However, in furtherance of their study, we propose here a similar method based on TESTMATICA MODEL where the entire structure of the design is thoroughly checked; and tested for errors while satisfying some reasonable assumptions. The test based on the design is evaluated against the specification and comparison was made with other testing approaches to compare their error detecting capabilities.

The result shows that our proposed method "TESTMATICA MODEL" TESTING STRATEGY can be a powerful testing tool for checking if all the control structures at the design level of many software systems are correct. In achieving this, we identified the following steps; 1) the highest number of states

in the design is estimated; for example, see an informal specification of a comment printer below:

**Comment Pointer:**
**Specification**: input consists of characters n{1,2,3}
print only comment. A comment is an
input sub-sequence enclosed by{*
on the left and *} on the right. (it may
contain other {*'s but not *{'s )
now looking at the specification;

```
int k, s;
int y[n];
k  0;
s  0;
while (k  n)
{
  s  s  y[k];
  k  k  1;
}
```

**Fig. 9 informal specification for a comment printing system**

2) According to the coding and the design, the test sequence is generated; and finally 3) the test sequence generated in step 2 is then checked.

The flowchart equivalent of Figure 9 is shown in fig 10 below.

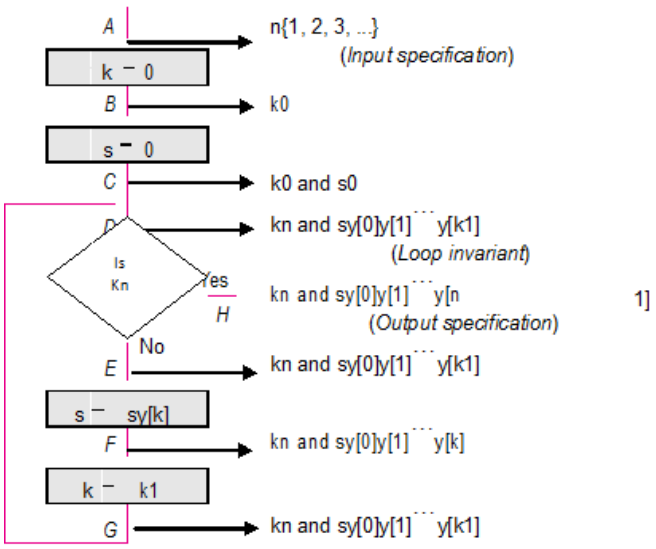


**Fig. 10. A representation of fig 9 with input specification, output specification, loop invariant, and assertions added.**

To prove that the code fragment and the corresponding flowchart in Figures 9 and 10 respectively is correct, we set, the variable s to contain the sum of the n elements of the array y after the code has been executed.

However, observe that Figure 10 contains a certain claim of a mathematical attribute from statement A to H; that is at the beginning and at the end of each statement. Having observed that, then each statement can be proven as thus.

The input specification, just at the beginning of the code execution holds at A and initializes n value as a positive integer; i.e,

$$A: n \in \{1, 2, 3, \ldots\} \quad \textbf{(1.1)}$$

The output specification to this is that, when control gets to *H* , then the addition of the n value will be stored in the s value which will all be stored in array y, that is,

$$H: s \;\; y[0] + y[1] + \ldots + y[n-1] \quad \textbf{(1.2)}$$

Then a very strong output specification can only prove correct the code fragment as shown:

$$H: \;\; k \;\; n \text{ and } s \;\; y[0] + y[1] + \ldots + y[n - 1] \quad \textbf{(1.3)}$$

After declaring the input and output specification, we move to prove the mathematical expression that holds at point D not minding if its loop invariant has been executed several times or not. That is;

$$D: k \# n \text{ and } s\ y[0] + y[1] + \ldots + y[k - 1] \quad \textbf{(1.4)}$$

Next is to prove that the code fragment is correct by showing that the output specification in (1.3) holds at point H; and the input specification in (1.1) holds at point A. Then the assignment statement k R 0 where the control is at B is first executed and this is what holds:

$$B: k \;\; 0 \quad \textbf{(1.5)}$$

That is to say that at point *B*, the statement holds as k 0 and $n \in \{1, 2, 3, \ldots\}$ thus making the input specification in (1.1) to hold at all points in the flowchart after which k 0 and $n \in \{1, 2, 3, \ldots\}$ is removed from the prove.

Then at point *C*, as a result of the statement s R 0, seen in the second assignment, the statement below holds true:

$$C: k \;\; 0 \text{ and } s \;\; 0 \quad \textbf{(1.6)}$$

Next is to prove by induction the correctness of the loop invariant in (1.4). While considering that statement (1.6) holds that C: k 0, and s 0; then k 0 by statement (1.6) and n $ 1 from input specification (1.1), will require k # n. Now since k 0, in statement (1.6) holds; then $k - 1 - 1$, will empty the sum in (1.4) and s 0 as required thereby proving that the Loop invariant (1.4) is true.

Next, the inductive hypothesis will be performed to ascertain if k is equivalent to some value $k_0$, $0 \# k_0 \# n$, and execution is at point *D.* then the statement that holds is

$$D: \;\; k_0 \# n \text{ and } s \;\; y[0] + y[1] + \cdots + y[k_0 - 1] \quad \textbf{(1.7)}$$

Next if k $_0$ $ n, and $k_0$ # n by hypothesis, it means that $k_0$ n. Going by the inductive hypothesis in (1.7), the following statement implies

$$H: \; k_0 \quad n \text{ and } s \quad y[0] + y[1] + \cdots + y[n-1] \qquad \textbf{(1.8)}$$

The above assertion is the output specification as in (1.3). Now, supposing the test $k_0$ $ n? did not hold, then control will be passed from a point *D* to a point *E.* Again *since* $k_0$ *is neither* ≥ n, $k_0$ n, then the statement implies.

$$E: k \quad n \text{ and } s \quad y[0] + y[1] + \ldots + y[k-1] \qquad \textbf{(1.9)}$$

O 0

Now that the statement s R s + y[$k_0$] is executed, then from statement (1.9), at point *F,* the following statement must hold:

$$k_0 \quad n \text{ and } s \quad y[0] + y[1] + \cdots + y[k_0 - 1] + y[k_0]$$

$$y[0] + y[1] + \cdots + y[k] \qquad \textbf{(1.10)}$$

0

After that, $k_0$ R $k_0$ + 1 will be executed. Now if we set the value of $k_0$ to 19 say before the statement is executed, then the last term in the sum in (1.10) will be y[19]. Notice that the value of $k_0$ is increased by 1 to 20 with the sum s unaltered, hence the last term in the sum still remains y[19], thereby bringing it to y[$k_0$ − 1]. Again, the incremental value of $k_0$ by 1 at point *F*, $k_0$ n presupposes $k_0$ # n should the inequality be hold at point *G. Now the outcome of* increasing $k_0$ by 1 is that at point *G* the following statement must hold:

$$G: \; k_0 \# n \text{ and } s \quad y[0] + y[1] + \cdots + y[k_0 - 1 \qquad \textbf{(1.11)}$$

Statement (1.11) that holds at point *G* above reveals its similarity with assertion (1.7) meaning that suppose (1.7) holds at *D* for k $k_0$, it will also hold at *D* with k $k_0$ + 1 proving the statement that the loop invariant in (1.4) holds for k 0 which also stand for all values of k, 0 # k # n.

Next is to prove the termination of the loop by observing that the loop adds 1 to the value of k at every iteration each time the statement k R k + 1 is executed. Remember also that statement (1.6) sets k to be equal to 0. This loop addition per iteration will therefore continue to increase by 1 until k reaches the value n thereby causing the exit of the loop and setting the value of s as in assertion (1.8), thereby satisfying output specification (1.3).

To conclude the review, observe that the input specification is given in (1.1), and that loop invariant in (1.4) holds irrespective of the number of times executed, which thereafter terminates after n iterations through an incremental value of 1. When this happens, the values of k and s will be seen to satisfy the output specification as in (1.3) hence proving correct the code fragment of Figure 9 and the flowchart in Figure 10.

The above process will give the programmer confidence that the software is correct since the correctness of any software simply means that input specification must satisfy the output specification. It also means that numbers of faults are reduced to the barest minimum and productivity gains are obvious.

## B. BENEFITS OF TESTING AS A FACTOR OF AGILE DEVELOPMENT METHODS.

Thus far, having reviewed the literature and explored the testing techniques applied in software development, this section discusses the benefits of testing as a component of the agile software development method.

### *1) Improves Quality*

Testing improves quality. When testing and review is conducted regularly at every phase of software development, quality is improved since every faults detected will be exposed and fixed at once. This is done more easily and quickly when the entire project is broken down into units or modules. Then it will allow the project development team to focus on developing high quality products.

### *2) Allows for change*

During the process of software development, the developers tries to constantly shape and reshape the overall product backing so as to meet customers demand. However, this process can introduce new or changed backlog items in the process thereby providing the opportunity for making or creating change.

### *3) Predictable cost and schedule*

Testing allows for cost of developing a product to be predicted. Owing to the fact that each scheduled time box is a fixed duration, the cost can be predictable. It will make clients to be well informed of the approximate cost which the product will take and also help to achieve good decisions on which feature should be considered first.

### *4) Early and predictable Delivery*

By using time box where jobs are scheduled to be delivered within a particular spate of time, testing makes it easier and possible for product delivery date and time to be highly predictable. Those faults that could have delayed the development process are detected earlier and fixed thereby making way for speedy development and delivery.

### *5) Transparency*

Testing guarantees transparency because it includes the client in the development process so as to make input. This gives the client a good impression that they are seeing their work in progress. Clients follow the development process and tend to be satisfied with what they are observing.

## III. METHODOLOGY

- **Design of the Study**

This study adopted a quantitative study in which the unit of analysis was individuals who were expected to provide their perception regarding the benefits of testing and testing techniques employed in executing software projects in their organizations. The rational for using quantitative research approach is that it allows the researcher to examine the relationship between two variables. In this study, the two variables are software testing and productivity gains. Insight from this study can be used to look for cause and effect relationships and to make predictions.

- **Area of Study**

The area of study is limited to selected G-8 countries and African countries. The countries that constitute G-8 are the United States, Canada, France, Germany, Italy, Japan, Russia and the UK; whereas the independent countries in Africa are 54.

- **Sample and Sampling Technique**

In a research with a large population of this kind, it is uneconomical to involve all the members (firms) of the population, [20]. Therefore a simple random sampling technique was used to select 5 software development firms across 5 countries randomly selected.

These firms were selected on ground that they recently shifted to agile method of software development and so are employing testing techniques in their software development process. Table 1 shows a distribution of the firms and their countries.

**Table I.** Distribution of firms and their countries

| S/N | Firms | Countries |
|---|---|---|
| 1 | Microsoft | USA |
| 2 | Symbiotic Application Services | South Africa |
| 3 | Net Solutions | United Kingdom |
| 4 | Microtelesoft | Japan |
| 5 | Tenece | Nigeria |
| | | |

- **Instrument for Data Collection**

Data for this study were collected using newly developed questionnaire by the author. The content was aimed at investigating the day-to-day experience associated with testing that reduces cost, risk, improves customer satisfaction and bring return on investment. Although the study used a convenient sample, precautionary measures were taken to ensure that all respondents were software professionals who work at these software development firms, and who have recently implemented (less than eighteen months) a software project.

- **Method of Data Collection**

The questionnaire was sent through e-mail to the firms sampled for the study. However, I received usable feedback from only 20 respondents in the following order, **Microsoft (USA) 01; Symbiotics Application Services, (South Africa) 04; Net Solutions (UK) 03, Microtelesoft (Japan) 05 and Tenecee (Nigeria) 07.** (see distribution in Table2).

My analysis therefore will be based on these pieces of data. However, my hypotheses is that indeed, testing in software development makes the software development process more agile and reduced cost, risk and this is in line with my research.

**Table II:** Showing Analysis of the rate of questionnaire returned

| S/N | Firms | Countries | |
|---|---|---|---|
| 1 | Microsoft | USA | 01 |
| 2 | Symbiotic Application Services | South Africa | 04 |
| 3 | Net Solutions | United Kingdom | 03 |
| 4 | Microtelesoft | Japan | 05 |
| 5 | Tenece | Nigeria | 07 |
| | **Total** | | **20** |

- **Data Analysis**

The data collected from the respondents will be analyzed using mean and standard Deviation. However, the results of data analysis will be presented with reference to the research hypotheses posed to determine whether the observed experience with testing matched my research, figure 3 Shows the Data Analysis Using mean and Standard Deviation.

**Table 3:** Showing data Anlysis using Mean and standard Deviation

| S/N | Items | Mean (n = 32) | SD (n =32) |
|---|---|---|---|
| 1 | The use of testing method improves software quality | 4.68 | 1.46 |
| 2 | The use of testing method reduces project cycle time | 5.05 | 0.95 |
| 3 | The use of testing method reduces development cost | 4.45 | 1.3 |
| 4 | The use of testing method improves the productivity of teams | 5.18 | 1.14 |
| 5 | The use of testing method improves customer satisfaction | 5.55 | 1.26 |
| 6 | I personally like extreme programming | 4.68 | 1.13 |
| 7 | I believe extreme programming speeds up the development process | 5.18 | 1.01 |
| 8 | I believe using testing methods improves the quality of code | 5.09 | 1.02 |
| | **Total (Average)** | 4.98 | 1.16 |

- **Discussion**

Indeed, the study found that (at the time of the present research), all the firms surveyed use testing process throughout its entire development cycle. The respondents maintained that team members spend more time on projects under testing process with a strong intent on achieving customers' desires. Due to these experiences, it would seem that team work of software developers has led to high productivity gains and customer satisfactions, [37].

- **Conclusion and Future scope**

Testing has been widely used as a way to help engineers develop high-quality software. Above result indicates that the top benefits of testing are customer satisfaction, speeding up the development process and improved productivity of teams. Based on similar norms and firm sizes, it is expected that other software development firms in the G-8 countries, Africa and the world could experience the same benefits.

However, shortfalls are also there. For example based on the small sample size and the fact that the study was conducted using few selected firms, my results may not give any surprising points which is different from what other researchers produced for generalization. But with time, and given a high response rate from employees and firms that employs testing methods, more data may be collected, analyzed and substantial results may be found to produce a more quantitative fact on the impact of testing on productivity gains.

- **Recommendation for Further studies**

Our first recommendation suggests that the way software is produces should be changed in other to achieve comparable future breakthroughs. To improve cost and productivity gains, software should be developed incrementally to enable it to be constructed phase-wise with testing carried out at each phase instead of trying to build the product as a whole. Some teams must begin phase-to-phase testing as necessary.

Great and potential software designers should be encouraged to undergo more training on software development to be up-to-date with recent trends in software development. This will bring about greatest hope considering that Brooks's opined that great designers should be considered first if we wish to improve software production and cost.

## ACKNOWLEDGMENT

I would like to thank all respondents who have participated in this study.

## REFERENCES

[1] Goodenough B, Gerhart SL. **Toward a theory of test data selection.** *IEEE Transactions on Software Engineering*. 1975 June, 1(2), pp. 156-173.

[2] Stephenson W. An Analysis of the Resources Used in Safeguard System Software Development. *Proceedings of the 2nd international Conference on Software Engineering, Califonia,* 1976, pp 312-321

[3] Myers G, Glenford J. The Art of Software Testing. 2nd edn. John Wiley and Sons: New York. 1979.

[4] Beck K. Extreme Programming Explained: Embrace Change. 1st edn. Addison-Wesley Longman Publishing Co.: USA, 2000.

[5] Dustin E. Effective Software Testing. 1st edn. Addison-Wesley Professional: New York. 2002.

[6] Software Testing Lifecycle. etestinghub. Testing Phase in Software Testing. Retrieved: 13/01/2012.

[7] Brooks F. No Silver Bullet. In: Information Processing. Kugler (ed.), Elsevier North-Holland.: New York. 1986; reprinted in IEEE Computer, April 1987), pp. 10–19.

[8] Eze N. **Development Process for Controllable Software**. *International Journal of Education and Research.* 2017 July, 5(7), pp. 37-52.

[9] Pan J. Evaluation of Software Testing tools (coursework). http://myassignmenthelp.info/assignments/master-the

sis-title-evaluation-software-testing-tools-87710/. Retrieved 21/11/2017.

[10] Dijkstra E. **The Humble Programmer**. *Communications of the ACM.* 1972 Oct, 15(10) pp. 859–66.

[11] Schach SR. Object Oriented and Classical Software Engineering. 10th edn. McGraw-Hill.: New York, 2010, pp. 18–23.

[12] Johnson RA. **The Ups and Downs of Object-Oriented System Development**. *Communications of the ACM. 2000 October,* 43(10) pp. 69–73.

[13] Certified Tester Foundation Level Syllabus. International Software Testing Qualifications Board. https://www.bcs.org/upload/pdf/ct-foundation-syllabus.pdf. Date Accessed: 31/03/2011.

[14] Howden WE. **Reliability of the path analysis testing strategy**. *IEEE Transaction on Software Eng*. 1976 Sept, vol. SE-2, pp. 208-215.

[15] Raymond ES. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. 2nd edn. O'Reilly & Associates: Sebastopol, CA, 2000.

[16] Dorota H, Adam K. Automated Defect Prevention: Best Practices in Software Management. 1st edn. Wiley-IEEE Computer Society Press. USA, 2007.

[17] Mcconnell S. **The Nine Deadly Sins of Project Planning**. *IEEE Software.* 2001 January, 18(1), pp. 5–7.

[18] Boehm BW. Software Engineering: R & D Trends and Defense Needs. *ICSE '79 Proceedings of the 4th international conference on software engineering*, *Germany, 1979,* pp 11-21.

[19] Kelly JC, Sherif JS, Hops J. **An Analysis of Defect Densities Found during Software Inspections**. *Journal of Systems and Software.* 1992 January, 17(2), pp. 111–17.

[20] Welman C, Kruger F, Mitchell B. Research Methodology. 3rd edn. Oxford University Press: London, 2005.

[21] Smith RK, Hale JE, Parrish AS. **An Empirical Study Using Task Assignment Patterns to Improve the Accuracy of Software Effort Estimation.** *IEEE Transactions on Software Engineering*. 2001 March, 27(3), pp. 264–71.

[22] Rubenstein D. Standish Group Report: There's Less Development Chaos Today. www.sdtimes.com/content/article.aspx?ArticleID=30247. Date accessed: 01/03/ 2007.

[23] Highsmith J. Cutter Consortium Reports: Agile Project Management: Principles and Tools 4. 2nd edn. Cutter Consortium, Arlington, MA, 2003.

[24] The Economic Impacts of Inadequate Infrastructure for Software Testing. National Institute of Standards and Technology, May 2002. Retrieved Date Accessed: 19/12/2017.

[25] Ackerman AF, Buchwald LS, Lewski FH. **Software Inspections: An Effective Verification Process.** *IEEE Software.* 1989 May, 6(3), pp. 31–36.

[26] Fowler PJ. **In-Process Inspections of Work products at AT&T.** *AT&T Technical Journal. 1986 March-April,* 65(2), pp. 102–12.

[27] Bush M. Improving Software Quality: The Use of Formal Inspections at the Jet Propulsion Laboratory. *Proceedings of the 12th International Conference on Software Engineering*, Nice, France, 1990, pp. 196–99.

[28] Shine Technologies. Agile Methodologies Survey Results. http://www.shinetech.com/download/attachment/98/ShineTechAgileSurvey. Date accessed: February 2012.

[29] Maria P, Lassenius C. How Does an Agile Coaching Team Work?: A Case Study. *Proceedings of the 2011 Sixth IEEE International Conference on Global Software Engineering.* New York, 2011, pp 29-38.

[30] Dyba T, Dingsoyr T. **Empirical studies of agile software development: A systematic review**. *Information and Software Technology.* 2008 August*, 50(9)*, pp. 833-859.

[31] Tran E. Verification/validation/certification (coursework). Carnegie Mellon University. (1999) Date Accessed: 13/08/2008.

[32] Chow TS. **Testing Software Design Modeled by Finite-State Machines**. *IEEE Transaction on Software Engineering*. 1978 May, Vol. SE-4(3), pp. 178-187.

[33] Gregory J, Crispin L. More Agile Testing. 1st edn. Addison-Wesley Professional. 2014, Pp. 23-39.

[34] Robert VB. Testing Object-Oriented Systems: Objects, Patterns, and Tools. 1st edn. Addison-Wesley Professional. USA, 1999.

[35] Beizer B. Software Testing Techniques. 2nd edn. Van Nostrand Reinhold. New York, 1990.

[36] A Glossary of Software Engineering Terminology. Institute of Electrical and Electronic Engineers, New York, 1990.

[37] Goodenough JB. A Survey of Program Testing Issues. In: Research Directions in Software Technology. P. Wegner (ed). The MIT Press, Cambridge, MA, 1979, pp. 316–40.

[38] Chauhan RK, Singh I. **Latest Research and Development on Software Testing Techniques and**

**Tools**. *International Journal of Current Engineering and Technology.* 2014 August, 4(4), pp.2368-72

[39] Whittaker JA. **What Is Software Testing? And Why Is It So Hard?** *IEEE Software*. 2000 Jan-Feb. 17(1), pp. 70–79.

[40] Miller GA. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information: The Psychological Review 63. 1956 March, pp. 81–97; reprinted in: www.well.com/user/smalin/miller.html.