# Novel Testing Rules : The Operation Of Defects

Prakash Kawar Asawat[#1], Krishan Kant Lavania [#2], Deepak Dembla[#3]

*Arya Institute of Engineering and technology , Jaipur, Rajasthan[#1,2,3]*

Prakash.asawat@gmail.com [1] , k@lavania.in [2] ,deepak_dembla@yahoo.com [3]

*Abstract*— **Current scenario of research shows that testing is a valuable area for every project to guarantee its quality and performance at actual behaviour. Before starting to test a system we first known literature about the remaining problem in project, due to this problem the performance and quality of project may not capture its planned area. So we first search out few of common error in the project. For that we need to clarify the concept of defect, error, fault, failure and other relevant to affect the system performance. In this paper we proposed simple and valuable stages for testing to capture common error and increase the performance of system. The paradigm of actual defect helps to classify the fault type. At last we summarize the paper and concluded with features scope.**

***Keywords:- defect, testing, error, fault, principals.***

## I. INTRODUCTION

Now-a-days, all the output affected by defects & other terms relate to defects. In medical minor defects in CT scan made the changes in doctors decision a big mistake may occurred. May be patent advised to take a harmful medicine. Similarly in system, a defect makes changes in output performance of real time system and they may crash. A lot of money with human body may affected from this crashed. It means the minor defect may create a big problem & mostly person affected from it. It means, it's important to create a framework for defects, so that peoples can easily understand them and may save the harm. Therefore we create a framework and focused to procedure with UML diagram for defects.

Computer system is affected by defects. The other terms are also involved to create the problems in system performance that is fault, failure, error etc. The actual 'mistake' in the program code is known as fault & the variation from expected behaviour observed by the user as a result of the error is call a failure. Error is the bad state into the system that results from the fault. The definition of these term are varies according to situation. The IEEE gave standard definition of these terms as, Failure- External behaviour is incorrect. It is the inability of a system or component to perform required function according to its specification. Fault- Discrepancy in code that causes a failure. It is a condition that causes the software to fail to perform its required function. And Error- Human mistake that caused fault.

### a. Means of Defect Density

Defect density means the percentile amount of conformation to detect number of defects (with compared to complete/ size of) into software component or configured item during predefined time duration of building process.

Figure 1: Defect density [2]

$$\text{Defect Density} = \frac{\text{Number of Defects}}{\text{Size}}$$

DD = M / (M + N) [3]

M = number of bugs detected by the test team

N = number of defects detected by client or end user

Size of Project can be Function points, feature points, use cases, KLOC etc. In general terms we can say that, Defect Density shows the comparative results of number of defects to the size (actual size) of the Project.

## II. HISTORICAL PERSPECTIVE

Defect is the dieses into system. The role of doctors is performed by the tester. It means defect play a major role to change the system performance. Now-a-days more than 40 research laboratories do research to manage the defect problem.

**Arpita Mittal & Sanjay kumar Dubey [1]**

In this research paper authors have studied about the various types of defect techniques and then undergone through the survey of COQUALMO cost constructive model which is a two-step software defect prediction model for improving the software quality. They have studied three techniques of Defect handling i.e. Defect Detection Technique, Second Defect Analysis Technique, and Defect Prediction Technique.

**Ghazia Zaineb and Dr. Irfan Anjum Manarvi [2]**

Zaineb etal., research presents the actual percentage of bugs rejection based on data collected from bug tracking system. Their paper provides a list of reasons behind bug rejection, their relation with severity level and possible threats that can affect software testing efficiency with reference to the life of a rejected bug. The major problem areas causing bug rejections are bug reports and insufficient knowledge of tester over the developed software.

**Sakthi Kumaresh and Baskaran Ramachandran [3]**

The articles provide a general framework of defect with its defect prevention measures suggested in order to enhance quality culture establishment in an organization. Implementation of defect prevention measures in subsequent projects would result in better performance, rapid and sustained improvement in the product quality as is evident from the example.

**Ruihua Chang, Xiaodong Mu and Li Zhang [4]**
In this paper, authors proposed a novel approach to resolve the problem of software defect prediction. The method is classified using Non-Negative Matrix Factorization (NMF). NMF algorithm is not only used for extracting external features but also as a powerful way for classification of software defect data. And the results show that it outperforms the state of the art techniques tested for this experiment. Finally, they suggest that it can be a useful and practical way addition to the framework of software quality prediction.

**Summary:** After studying of these papers, we conclude that recently research work is done on the defect & all stages of system problems. Suited models are used to estimate and prediction of defects, but no single model is sufficient for it. So, construction of a new model is very important.

**Defect Predictors [5]**
In software development, every change induces a risk. What happens if code changes again and again in some period of time? In an empirical study on Windows Vista, we found that the features of such change bursts have the highest predictive power for defect-prone components. With precision and recall values well above 90%, change bursts significantly improve upon earlier predictors such as complexity metrics, code churn, or organizational structure.

Software development can be seen as a sequence of changes—a constant stream of activities that add new value to software, adapt it to a changing environment, delete features no longer required, or improve its structure for better maintenance. All of these activities are ultimately conducted by humans, and as humans make mistakes, it is unavoidable that some of these changes will induce defects.

COMPARING PREDICTORS FOR DEFECT-PRONE VISTA COMPONENTS

| Predictor | Precision | Recall |
|---|---|---|
| Pre-Release Defects | 73.8% | 62.9% |
| Test Coverage | 83.8% | 54.4% |
| Dependencies | 74.4% | 69.9% |
| Code Complexity | 79.3% | 66.0% |
| Code Churn | 78.6% | 79.9% |
| Organizational Structure | 86.2% | 84.0% |
| Change Bursts *(this paper)* | 91.1% | 92.0% |

In this paper, our conjecture is that over the development time of a system, such multiple attempts would manifest themselves in consecutive code changes over a period of time. Such change bursts could be indicators for various problems, including those traditionally detected by earlier predictors:

- Incomplete or changing requirements. Requirements may only become stable after multiple implementation

attempts—for instance, because of conflicting organizations involved.

- Hairy bugs. Defects may only be tentatively fixed without knowing the exact cause, making them re-occur

again and again—that is, the code or task is overly complex.

- Insufficient quality assurance. Quality assurance may not detect all issues in the first place, thus requiring

constant fixing of newly discovered defects—improving test coverage over time.

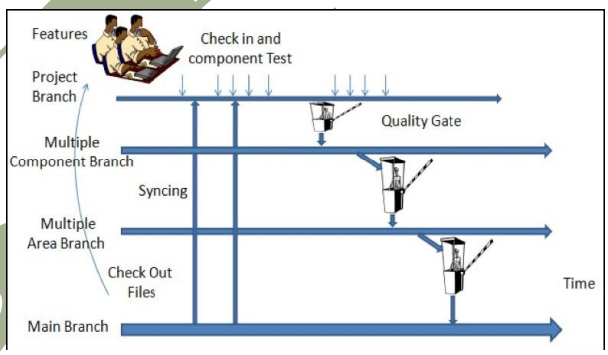



Figure 1. How the Windows development process works. Changes are first committed in project branches, and then subsequently merged and integrated into the Windows main branch.

**Comparing characteristics of Firefox and Internet Explorer regarding defects [6]**

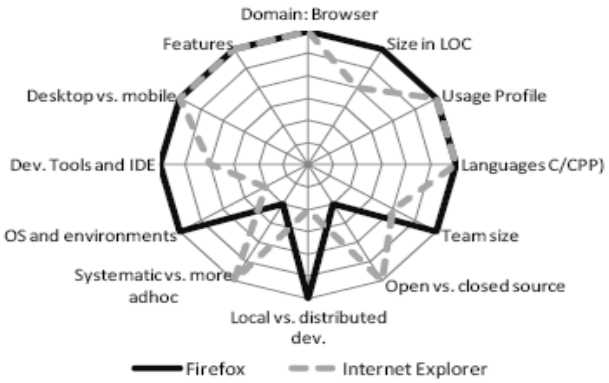



Figure 2: Comparing Characterisitcs of Firefox & IE

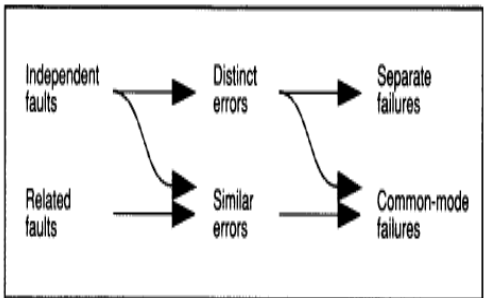



Figure 3: Classes of Faults, errors & failures

## III. TESTING TECHNQUE & RULES

*Phase Detected of defects:* Phase Detected indicates the phase in the software development lifecycle where the defect was identified.

- *Test coverage in unit testing*

- Breadth of functional coverage
- Percentage of paths, branches or conditions that were actually tested
- Percentage by criticality level: perceived level of risk of paths
- The ratio of the number of detected faults to the number of predicted faults.
    - Unit Testing
    - Integration Testing
    - System Testing
    - Acceptance Testing

***System Test Triggers***

System test activities deal with system wide and cross-system issues, including hardware and software environment implications as well as cyclic and sometimes demanding workload volumes.

***Function Test Triggers***

There are several terms used to describe the testing of the functional aspect of a product. Depending on the size and scope of the project, any or all could be applied under the heading of function test. Unit test, for example, is an effort to validate the ability of the code written to execute successfully, independently from other influences such as interfaces with other products or functions. Function test takes a broader view, ensuring not only that the function executes successfully, but that interfaces are handled correctly, and that the function provides expected results. Component test is a term applicable to a large product which consists of multiple elements (components). This additional 'function test' ensures that all of the functions within a component perform satisfactorily, and the components of a product interface correctly with each other.

**DEFECT TYPES**

There are various ways in which we can classify. Below are some of the classifications:

**Severity Wise:**

- **Major:** A defect, which will cause an observable product failure or departure from requirements.
- **Minor:** A defect that will not cause a failure in execution of the product.
- **Fatal:** A defect that will cause the system to crash or close abruptly or effect other applications.

**Work product wise:**

- **SSD:** A defect from System Study document
- **FSD:** A defect from Functional Specification document
- **ADS:** A defect from Architectural Design Document
- **DDS:** A defect from Detailed Design document
- **Source code:** A defect from Source code
- **Test Plan/ Test Cases:** A defect from Test Plan/ Test Cases
- **User Documentation:** A defect from User manuals, Operating manuals

**Type of Errors Wise:**

- **Comments:** Inadequate/ incorrect/ misleading or missing comments in the source code
- **Computational Error:** Improper computation of the formulae / improper business validations in code.
- **Data error:** Incorrect data population / update in database
- **Database Error:** Error in the database schema/Design
- **Missing Design:** Design features/approach missed/not documented in the design document and hence does not correspond to requirements
- **Inadequate or sub optimal Design:** Design features/approach needs additional inputs for it to be completeDesign features described does not provide the best approach (optimal approach) towards the solution required

- **In correct Design:** Wrong or inaccurate Design
- **Ambiguous Design:** Design feature/approach is not clear to the reviewer. Also includes ambiguous use of words or unclear design features.
- **Boundary Conditions Neglected:** Boundary conditions not addressed/incorrect
- **Interface Error:** Internal or external to application interfacing error, Incorrect handling of passing parameters, Incorrect alignment, incorrect/misplaced fields/objects, un friendly window/screen positions
- **Logic Error:** Missing or Inadequate or irrelevant or ambiguous functionality in source code
- **Message Error:** Inadequate/ incorrect/ misleading or missing error messages in source code
- **Navigation Error:** Navigation not coded correctly in source code
- **Performance Error:** An error related to performance/optimality of the code
- **Missing Requirements:** Implicit/Explicit requirements are missed/not documented during requirement phase
- **Inadequate Requirements:** Requirement needs additional inputs for to be complete
- **Incorrect Requirements:** Wrong or inaccurate requirements
- **Ambiguous Requirements:** Requirement is not clear to the reviewer. Also includes ambiguous use of words – e.g. Like, such as, may be, could be, might etc.
- **Sequencing / Timing Error:** Error due to incorrect/missing consideration to timeouts and improper/missing sequencing in source code.
- **Standards:** Standards not followed like improper exception handling, use of E & D Formats and project related design/requirements/coding standards
- **System Error:** Hardware and Operating System related error, Memory leak
- **Test Plan / Cases Error:** Inadequate/ incorrect/ ambiguous or duplicate or missing - Test Plan/ Test Cases & Test Scripts, Incorrect/Incomplete test setup
- **Typographical Error:** Spelling / Grammar mistake in documents/source code
- **Variable Declaration Error:** Improper declaration / usage of variables, Type mismatch error in source code

**Status Wise:**

- Open
- Closed
- Deferred
- Cancelled

## I. SIMPLE PARADIGM MODEL FOR STAGES OF TESTING

a. Literature of historical perspective
   (i) Search the relevant topic relate to problem
   (ii) Idea stored from ontological technique/ approaches
   (iii) Summarized the work and their goal that may previously have done.
   (iv) Improve the knowledge to clear appropriate type of testing apply to which type of problem.

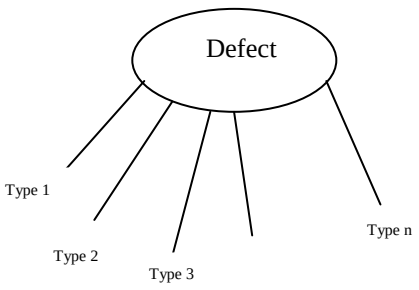
b. Selected the nearest and appropriate required area of testing
   (i) Create the list of common problem in software with their appropriate testing technique
   (ii) Arrange the problem in an order with testing technique

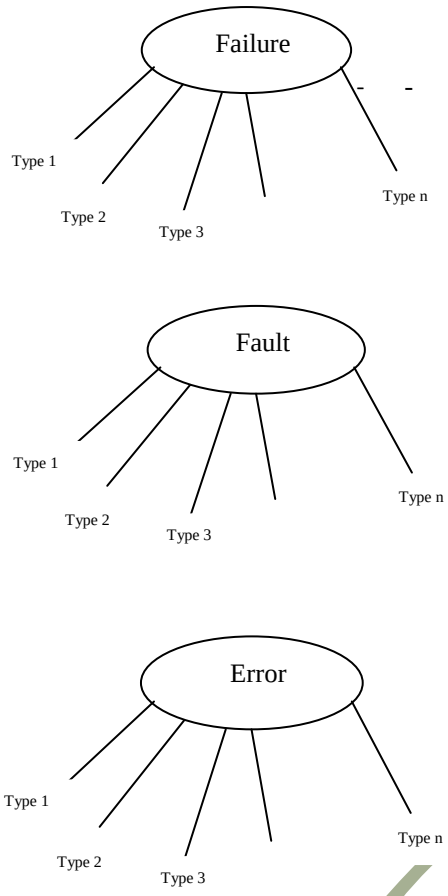
c. Apply problem searching technique
   (i) list of common problem
   (ii) Search there may be a problem that match with list of common problem
   (iii) If problem is differ from general problem then, detection technique applied

d. Capture problem

e. Classify the category of problem
   (i) Classify the class of Problem. They may be belong to the class of fault, failure, error & defect etc,
   (ii) Further classification of those categories. For example: a problem may be in defect categories then classify the defect( in which problem are matched)

(iii)






## APPLICATION

This model is Applicable in the area of examination of software and hardware. Other most common use of this model is listed as below:

(i) Design Electronics tools
(ii) Software development life cycle (software testing)
(iii) Examine Mechanical tools

## IV. CONCLUSION AND FUTURE SCOPE

Here we have created a descriptive research on defects. We clarify the basics of testing techniques to control the defects in different prospective. Through the help of these, we designed the novel rules for testing. There graphical representation of this rules is known as the smple paradigm model for stages of testing [SPST]. Through this model, we can easily improved the quality of product.

.

## ACKNOWLEDGMENT

Authors are thankful to all audience who support dicetly& indiactrely.

## REFERENCES

[1] Arpita Mittal, Sanjay kumar Dubey, " Defect Handling In Software Metrics", International Journal of Advanced Research in Computer and Communication Engineering Vol. 1, Issue 3, May 2012.

[2] Ghazia Zaineb and Dr. Irfan Anjum Manarvi, "IDENTIFICATION AND ANALYSIS OF CAUSES FOR SOFTWARE BUG REJECTION WITH THEIR IMPACT OVER TESTING EFFICIENCY", International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.4, October 2011.

[3] Sakthi Kumaresh and Baskaran Ramachandran,"DEFECT PREVENTION BASED ON 5 DIMENSIONS OF DEFECT ORIGIN", International Journal of Software Engineering & Applications (IJSEA), Vol.3, No.4, July 2012.

[4] Ruihua Chang, Xiaodong Mu and Li Zhang, "Software Defect Prediction Using Non-Negative Matrix Factorization",JOURNAL OF SOFTWARE, VOL. 6, NO. 11, NOVEMBER 2011.

[5] Sannella Change Bursts as Defect Predictors Nachiappan Nagappan_ Andreas Zellery Thomas Zimmermannz Kim Herzigx Brendan Murphy.

[6] Forman, "Cross-project Defect Prediction: A Large Scale Experiment on Data vs. Domain vs. Process", Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger & Brendan Murphy, *ESEC/FSE'09,* August 24–28, 2009, Amsterdam, The Netherlands. Copyright 2009 ACM 978-1-60558-001-2/09/08...$10.00.

TABLE 1: Difference between system problems

| S. No | System Problem Types | | | | |
|---|---|---|---|---|---|
| 1. | Defect | Error | Bug | Fault | Failure |
| 2. | Mismatch between the requirements | A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. | A fault in a program which causes the program to perform in an unintended or unanticipated manner. | An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner. | The inability of a system or component to perform its required functions within specified performance requirements. |

TABLE: Defect metrics for a week of operation of a system that runs 24 hours a day

| S. No. | Date | Time of Defect | Defect Severity | Time since last defect |
|---|---|---|---|---|
| 1. | START of Recording, Monday, July 2, 0000 | N/A | N/A | N/A |
| 2. | Monday, July 2 | 0900 | 1 | 9 hr |
| 3. | Monday, July 2 | 1600 | 4 | 5 hr |
| 4. | Tuesday, July 3 | 0700 | 1 | 18 hr |
| 5. | Wednesday, July 4 | 1800 | 1 | 33 hr |
| 6. | Thursday, July 5 | 1300 | 2 | 17 hr |
| 7. | Saturday, July 7 | 1300 | 1 | 8 hr |
| 8. | End of Recording, Sunday, July 8, 2400 | N/A | N/A | (8 hr with no defect) |
| **Total** | | **6 defects** | **6 failure plus 1 defect** | |

The above metrics were collected over a week of July 2012. There were 6 defects in 5 days, or an average of 0.833 days between defect, or 16.33 hours between defects. There were 6 failures in 6 days, for an average of 1 days between failure, or 24 hours between failure.
Initially,

$$9 + 5 + 18 + 33 + 17 + 8 = 90 \text{ hours}$$

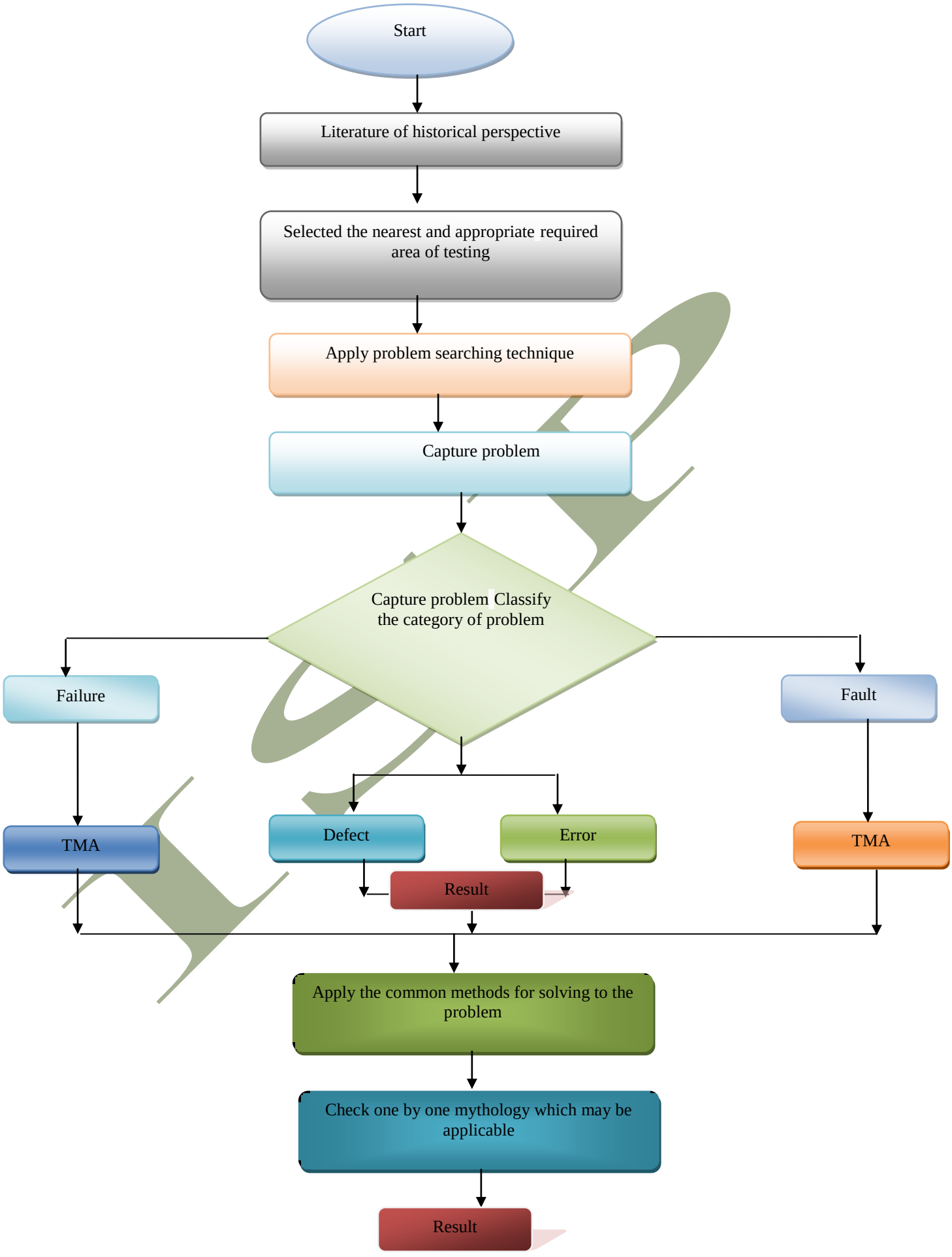
Addition of time with the 8 hours with no defect

$$90 + 8 = 98 \text{ hours}$$

Divided by 6 defects

$$= 98 / 6$$
$$= 16.33 \text{ hours}$$

Similarly failures can be calculated. We didn't need to record the exact time each defect or failure is encountered in order to compute MTTD or MTTF. We just need the total number of defects or failures encountered and the total amount of time the system was running or tested.


Start
Literature of historical perspective
Selected the nearest and appropriate required area of testing
Apply problem searching technique
Capture problem
Capture problem Classify the category of problem
Failure
Fault
TMA
Defect
Error
TMA
Result
Apply the common methods for solving to the problem
Check one by one mythology which may be applicable
Result