Volume 3, Issue 8, August 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# Analysis of Robustness of A Developed E-Learning System

[1]Afolabi A.O , [2]Adagunodo E.R
[1]*Ladoke Akintola University of Technology,Ogbomoso. Nigeria*
[2] *Obafemi Awolowo University, Ile Ife*

***Abstract: Among the reasons why programs fail is logic errors in the code, and exception failures, Exception failures can account for up to 2/3 of system crashes. [1], Traditional approaches to reducing exception failures, such as code reviews, walkthroughs and formal testing, while very useful, are limited in their ability to address a core problem: the programmer's inadequate coverage of exceptional conditions. Therefore ,this work is an attempt to analyze the degree of robustness of the developed E-Learning Systems by the use of software robustness metrics and fishbone analytical tool.The result shows that exception failures in the developed E-Learning System could be identified and the application of exception handlers can increase the robustness of the system.***

***Keywords:***

## 1.1 APPROACHES TO ANALYSIS

This section chapter addresses the problem of reducing the number of exception failures caused by inadequate exception handling in code. The approach taken is to regard instances of inadequate exception handling as errors in coding; such errors are effectively design errors or, ultimately, human errors. [2] would appear to support this position when he says, citing that "approximately two thirds of system failures are due to design faults in exception handling (or recovery) algorithms." His use of the term "design fault" raises the following kinds of questions. What kinds of errors do programmers make when they fail to cover exception conditions? Their main concern is programming-language constructs for handling exceptions. [2], in a thorough survey of the literature, suggests one reason for the poor state of exception handling: "In operational computer software systems often more than 2/3 of the code is devoted to detecting and handling exceptions" Yet, since exceptions are expected to occur rarely, the exception handling code of a system is in general the least documented, tested and understood part and there are various exception categories as discussed below.

## 1.2 EXCEPTION CATEGORIES

From a theoretical perspective, a mnemonic would be effective, because the mnemonic would facilitate the initial recall of the categories, and then each category would provide a context for semantically extending the elements of the category to include all or most of the category members (a process that cognitive scientists call priming. If the mnemonic was linked to a physically salient graphic structure, memory would be even further enhanced. An example of such a mnemonic is *children*. "Everyone knows exceptional *children*." The letters of the word *children* can be used to remember the list of exception categories shown in the Table below.

| NO OF CATEGORIES | CATEGORIES OF EXCEPTIONAL CASES |
|---|---|
| 1 | **C omputational problem** |
| 2 | **H ardware problem** |
| 3 | **I /O and file problems** |
| 4 | **L ibrary function problem** |
| 5 | **D ata input problem** |
| 6 | **R eturn-value problem** |
| 7 | **E xternal user/client problem** |
| 8 | **N ull pointer and memory problems** |

Fig 1 The exceptional *children* mnemonic.

Such mnemonic devices could address the matter of exception coverage; the issue of *correctly* handling exceptions is a separate matter.

## 1.3 Fishbone Analysis

[2] provides guidance on the perception of graphical structures, one graphical technique that facilitates recall and provides relational structure is the so-called fishbone diagram, also known as a cause-and-effect diagram or an Ishikawa diagram, after its founder, Kaoru Ishikawa [7]. Fishbone diagrams are widely used in quality control. Figure 2 shows an example, roughly in the shape of a fish that depicts exceptions that could be encountered in a software system. At the "head" of the fishbone is the phenomenon to be avoided: exception failures. The ribs are labeled with categories of events that cause exception failures, and the events within each rib are examples of specific causes. For instance, the rib labeled "computational problem" lists divide-by-zero as an exemplar. The fishbone in the figure is an attempt to lay out a fairly comprehensive set of exception causes covering most programs. The exemplars were obtained via hazard analysis, everyone knows exceptional children.

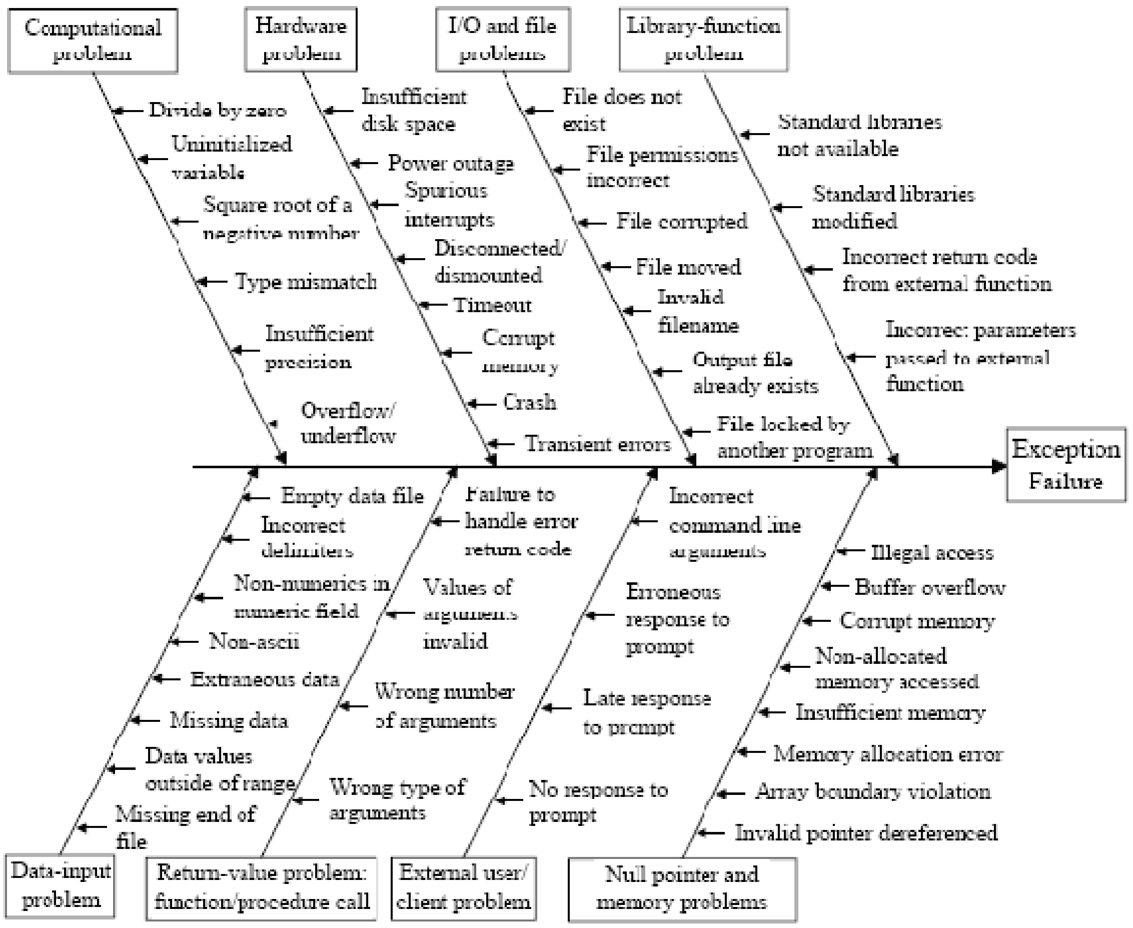

**Figure 2:** Fishbone diagram showing exception types and exemplars.

The first letters of the rib labels spell the mnemonic children, an ideal dependability case might be little different from a safety case, although it's unclear that all of the elements of a safety case are necessary if one's goal is simply to improve coverage of software exceptions. The term "safety case" is suggestive of safety-critical software, and may not be taken seriously by programmers working on such projects as spreadsheets which, at first blush, do not appear to be safety critical; so, the term "dependability case" is introduced here in an effort to broaden the appeal to all programmers.

## 1.4 Exception Types in developed E-Learning System

Using the fishbone diagram to analyze the exception types in e-Learning system it was found out that the categories of events that can cause exceptions are depicted by the diagram above in Fig 3. While the fish bone diagram shows the case of events in eight categories with 49 cases of events, in the E-Learning system developed the categories of event identified is also 8 with 26 exception cases that could affect the robustness of the system. For the system to be robust there is the need of provision of exception handlers within the system that guarantees robustness against the number of events identified within the system. Notice that the first letters of the rib labels spell the word *children*, the mnemonic that typifies exceptional cases that could be encountered within the System.

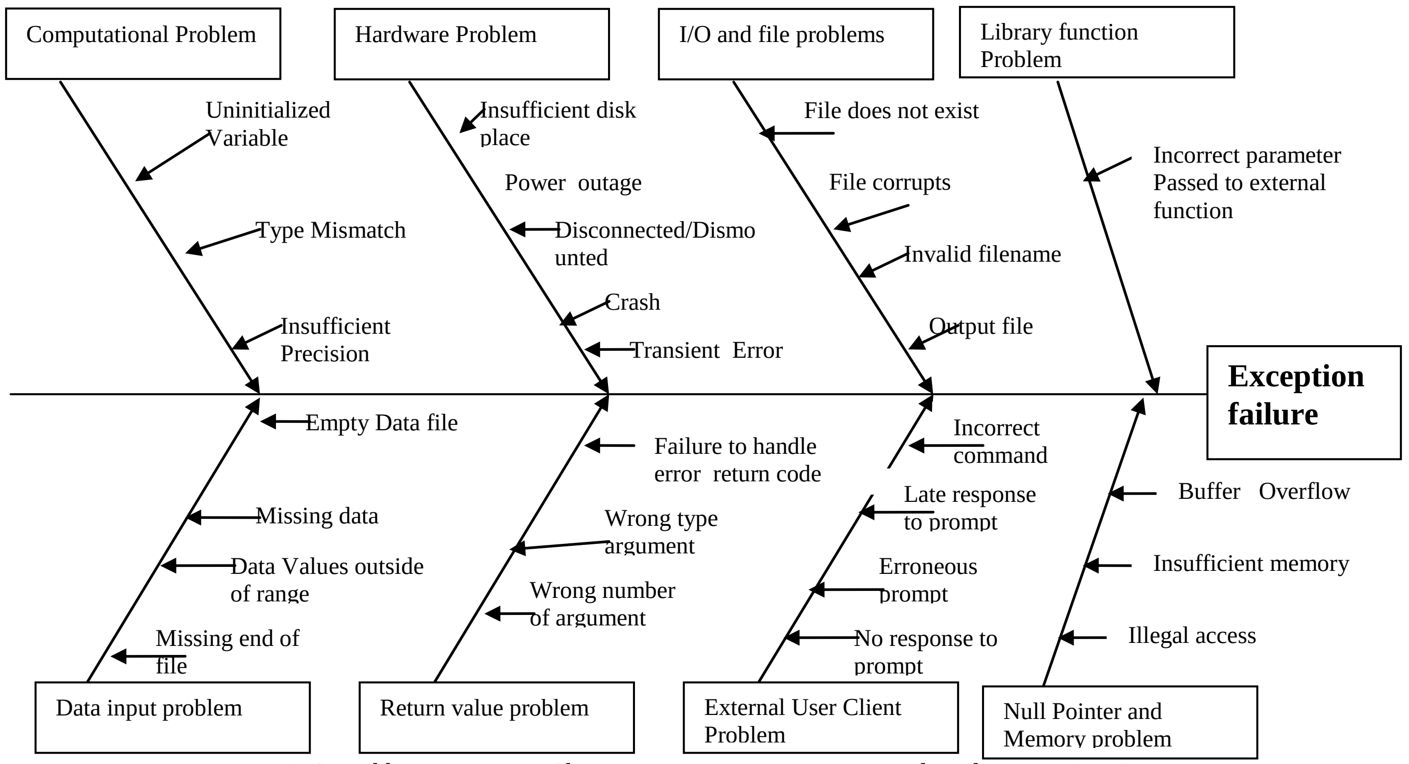

Fig 3 Fishbone Diagram Showing Exception types in Developed E-Learning System

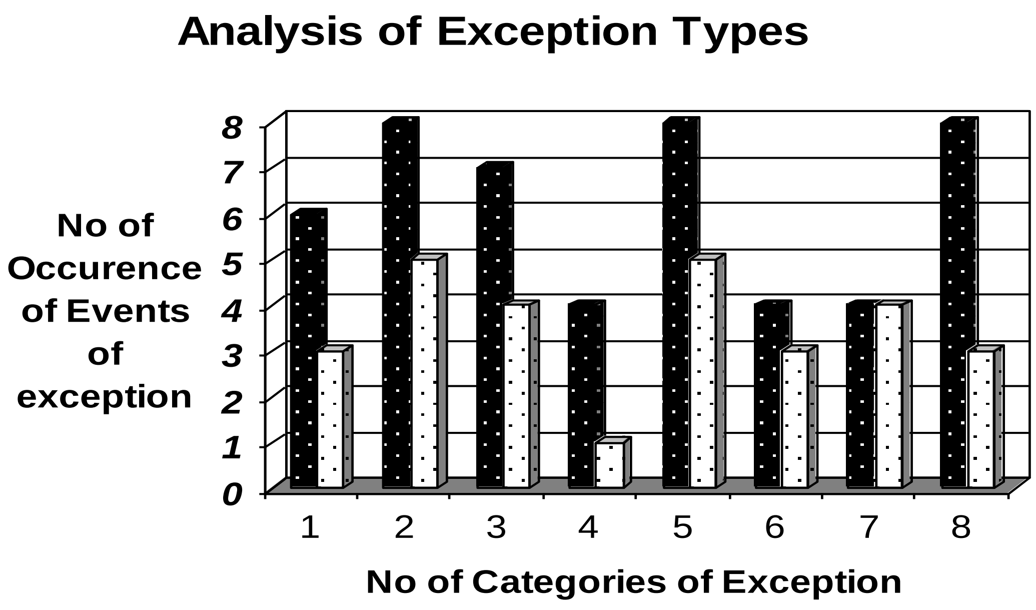

Fig 4 Exception Analysis using the mnemonic 'CHILDREN'
*Exception Types in E-Learning Fish bone diagram – White*

## 2.0 Robustness of Software

Robustness measures the degree of protection of a system against potential outage events. In this work, robustness is referred to as a system's ability to detect and handle fault, maintenance and system-external events, and the resulting degree to which it remains available in the face of these events. There are three types of events a system must protect against, the

first one is hardware and software faults. Past research has primarily focused on measuring a system's robustness against faults through fault injection. The second type of events a system must protect against is maintenance events. Maintenance events can also result in a system outage, and the system's robustness against maintenance events is a key differentiator among vendors. However, a system's robustness against maintenance events has not received much attention in past research. In this work we propose a method to encapsulate maintenance events into robustness benchmarking. Both fault and maintenance events are *system internal events*. The third type of event a system must protect against is system-external events, such as environmental events and human error. [2] investigates issues in benchmarking robustness in the presence of human operator interaction with the system. In this work, we focus our discussion around the first two types of events, since methods for benchmarking robustness involving human and other external factors are still in their infancy stage.
The fault and maintenance space can be quite large so it is impractical to measure the impact of each fault and maintenance event on availability. Fortunately, systems are designed to handle certain types of fault and maintenance events in a consistent matter. For example, in a given system, certain types of faults will always cause the system to crash, while others may be totally transparent to users. Hence, for the purpose of benchmarking, it seems logical to group fault and maintenance events into a small number of classes.

Many of the previous robustness studies have proposed their own classification systems. For example, [2] defines a 5-point CRASH system, and Brown has four classes of availability. Once fault and maintenance events are grouped into a small set of classes, a method is needed to formulate a robustness benchmark score so different systems can be compared. [7] proposed such a method for benchmarking robustness of systems. In this framework, we use fault classification and maintenance classification schemes that are designed to be general, in order to be applicable to a wide range of systems as well as at different levels of a system. In conjunction with the fault and maintenance classifications, we use a method by [7] to calculate robustness benchmark scores.

### 3.8.1 Fault and Maintenance Classification

The most obvious classification of fault events is a two-class system based on a system's availability in the face of these events: system available and system unavailable. While this classification has the obvious advantage of having a direct relationship with system fault tolerance definition, only a small portion of fault and maintenance events will result in extended outages that require human intervention. The ability to return to service quickly is an integral part of the strategy for achieving high availability for general purpose computers. It is our view that a third class is needed to extend the traditional fault tolerant concept to general purpose computers. We define fault events as falling into one of three classes:

**Class 1 (Non-recoverable) Fault :** A fault that causes a system outage and forces the system to remain unavailable *until the fault is removed*. An example of this is a non-redundant component essential to system operation, such as the main interconnect of the system.

**Class 2 (Recoverable) Fault:** A fault that causes an outage, but the fault can be circumvented to allow the system to return to service. An example of this is a CPU fault in a multi-CPU UNIX system. The fault can be circumvented with firmware blacklisting and restart features.

**Class 3 (Transparent) Fault:** A fault that does not cause a system outage. An example of this is a failure of a redundant component (e.g., a power supply module).
Similarly for maintenance events, it would be too simplistic an approach to define a two-class system based on whether or not the system is up. Quite often a system is down only during a certain part of a maintenance operation. For example, a system is available during a software patch installation, but a reboot is subsequently needed in order for the patch to take effect. We define maintenance events as falling into one of three classes:

. **Class 1 (Non-recoverable) Maintenance:** A maintenance event that requires the system to be unavailable for the entire duration of the maintenance activity. The replacement of the main system interconnect may be an example of a class 1 maintenance activity.

. **Class 2 (Recoverable) Maintenance :** A maintenance event that requires the system to be unavailable *during part, but not all, of the maintenance activity*. The addition of a software patch above is an example of a class 2 maintenance activity.

. **Class 3 (Transparent) Maintenance:** A maintenance event that does not require a system outage. The replacement of a hot plugable, redundant component (e.g., a power supply) would be a class 3 maintenance event.
There is a natural interaction between fault events and maintenance events. Figure 4 provides an illustration of the relationships among the different classes of fault and maintenance events. Class 1 faults, by definition, will always require a class 1 maintenance event. Class 2 and 3 faults, howe ver, may require or allow for maintenance handling as a class 1, 2, or 3 maintenance event.

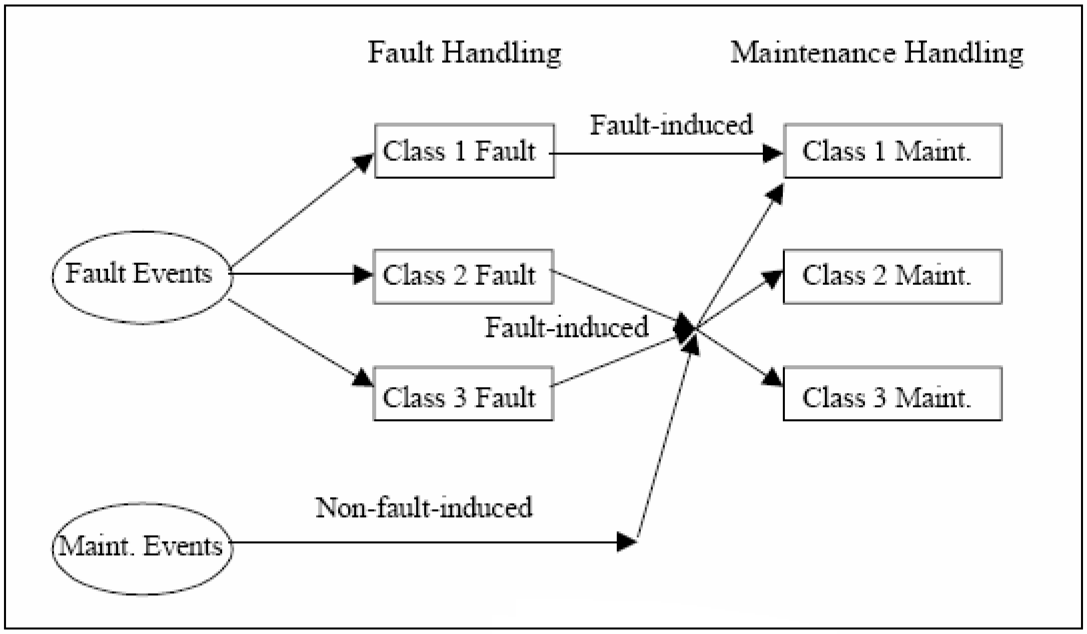

Fig 5 Fault and Maintenance Events

The "Maint. Events" node in Figure 4 depicts non-fault induced maintenance which falls into 1 of the 3 maintenance categories.

**3.8.2 Robustness Benchmark Metrics and Their Calculation**

In order to calculate robustness benchmarks that account for system protection against both fault and maintenance events, there is the need to define the following three robustness benchmark metrics:

**Fault Resilience Index** (*FRI*) measures a system's robustness against fault events.

**Maintenance Resilience Index** (*MRI*) measures a system's robustness against maintenance events.

**Outage Resilience Index** (*ORI*) measures a system's robustness against both fault and maintenance events. *ORI* is a function of *FRI* and *MRI*.

**Fault Resilience Index (*FRI*).** For each fault class, we assign a numeric value called the Fault Class Factor (*FCF*). *FCFi* is numeric representations of the fault class i. Higher FCF values imply better robustness. *FRI* is the weighted average of the three fault class factors, with the weights being the percentage of faults in the corresponding fault class. *PFCi* refers to the percentage of faults in fault class *i*. The sum of *PFCi* over the three possible values of *i* is taken to be 100%. FRI can be calculated using the formula:

$$FRI = \Sigma(FCF_i * PFC_i) \quad \text{where } i=1,2,3$$

*eqn 1.0*

When assigning a value to a given fault class, the absolute value of *FCF* is not important; *consistency* in assigning the value is. Otherwise, an *FRI* from a system can not be compared to an *FRI* from another system. For general purpose computers, we suggest using the following values: *FCF1*=1, *FCF2*=10, *FCF3*=100. These values represent the large disparity in downtimes for the different classes of faults. A class 1 fault tends to result in an outage that is in the range of several hours, whereas the outage duration associated with a class 2 fault is typically measured in minutes or tens-of minutes.

The values are designed to reflect that class 3 faults have an order-of-magnitude better
availability than class 2 faults, which in turn, yield an order-of-magnitude better availability over class 1 faults. For example, if a system has 20% of its faults in class 1, 30% of its faults in class 2, and the rest (50%) of its faults in class 3, the *FRI* of the system is calculated as follows: *FRI* = 1 * 0.20 + 10 * 0.30 + 100 * 0.50 = 53.20

Given the total set of faults occurring in a system under consideration, their categorization into the three fault classes (and hence the determination of the *PFCi*) can be done through fault injection tests. For most systems, finding the complete set of faults is intractable. However, a representative sample that accurately reflects the types of faults in a majority of situations can be used as a close-to-optimal methodology for the determination of the *PFCi*.

**Maintenance Resilience Index (*MRI*).** *MRI* is similar to *FRI*, but applied to maintenance events. Hence, following the discussion of *FRI* above, we have *MRI* being the weighted average of Maintenance Class Factors (*MCFi* ), with the weights being the percentage of maintenance events in each maintenance class.

Let *PMCi* refer to the percentage of maintenance events belonging to maintenance class *i*. The sum of *PMCi* over all possible values of *i* (*i* =1, 2, 3) is always 100%.
*MRI* can be calculated using the following formula:

$$MRI = \Sigma(MCF_i * PMC_i) \quad \text{where } i=1,2,3$$

*eqn 1.2*

Let *MCF1*=1, *MCF2*=10, *MCF3*=100, as we did with Fault Class Factors. Therefore, *MRI* can be calculated once we know the values for *PMCi* .
Unlike the fault event space, which is very large, the event space for maintenance events tends to be comparatively small. Taking the example of system hardware, computer systems are designed to be serviced in Field Replaceable Units (*FRU*), which are modules that can be replaced from the system individually. Typical mid- to high-end UNIX systems consist of *FRUs* ranging from dozens to a few hundreds in quantity. Regardless of what component fails in an *FRU*, the maintenance scenario for the
*FRU* is the same, which is replacing the entire *FRU.* Its categorization into the three classes can be achieved via simulation of this event on the system in question. Since there is a limited number of *FRUs* in a given system, this means that quantifying *PMCi* can be done relatively easily.

**Outage Resilience Index (*ORI*).** *ORI* is a function of *FRI* and *MRI*. Moreover, *ORI* must also account for the potential difference in the impact of unplanned disruption (caused by fault events ) versus planned disruption (caused by maintenance events). In many instances, unplanned downtime tends to be more harmful than planned downtime. We account for this in the *ORI* formula by allowing weights to be associated with the the fault and maintenance robustness indices.
We express *ORI* as the weighted average of *FRI* and *MRI*:
*ORI* = (*wf***FRI*)+(*wm***MRI*) Here *wf* and *wm* are weights associated with fault and maintenance indices respectively, and the sum of *wf* and *wm* is always 1. A higher value can be assigned to *wf* over *wm* to reflect the impact of fault (unplanned) outages versus maintenance (planned) outages, or an equal weighting, where *wf* = *wm* = 0.5, results in
*ORI* representing a simple average of *FRI* and *MRI*. The values selected for fault and maintenance weights must also be the same across the systems being evaluated.
*FRI* and *MRI* have a value between 1 and 100, with 1 being the least robust and 100 being the most robust. Thus, *ORI* will be a value between 1 and 100, with 1 representing the least robust, and 100 representing the most robust. In all cases, larger values denote better robustness.

## Application of Robustness Benchmark Metrics on the Secured E-Learning System

The above metrics can be used to measure the robustness of the E-learning portal that is being developed in this project. The front end of the developed system is done by the use of Java language and some types of exception were handled and these are servelet exception, Class not found exception, SQL exception, Input and Output exceptions, the
robustness of the system developed was measured by the use of the various indexes of the metrics mentioned above.
Fault Resilience Index (*FRI*) measures a system's robustness against fault events.
Maintenance Resilience Index (*MRI*) measures a system's robustness against maintenance events.
Outage Resilience Index (*ORI*) measures a system's robustness against both fault and maintenance events. *ORI* is a function of *FRI* and *MRI*. Fault Resilience Index (*FRI*) measures a system's robustness against fault events.
Maintenance Resilience Index (*MRI*) measures a system's robustness against maintenance events.
Outage Resilience Index (*ORI*) measures a system's robustness against both fault and maintenance events. *ORI* is a function of *FRI* and *MRI*. From the analysis through the fishbone model various indices are measured by the use of the number of occurrence of cases as depicted by the events.

**Fault Resilience Index (FRI)**

$$FRI = \Sigma(FCF_i * PFC_i) \quad \text{where } i=1,2,3$$

*equation 1.3*

From the given value above FCF (Fault Class Factors) FCF 1=1, FCF 2=10 and FCF 3=100 and $\sum$ PFC $_I$ =100% using the equation 1.3 the results in table 3.2 are found.

Table 2 FRI (Fault Resilience Index)

| i | 1<br>Non Recoverable Fault | 2<br>Recoverable Fault | 3<br>Transparent Fault |
|---|---|---|---|
| Benchmark value | 25% | 50% | 25% |
| E-Learning value | 25% | 25% | 50% |

From this results the Fault Resilience Index of the E-Learning system can be calculated :
i.e FRI=1*25%+ 10*25%+100*50%
= 0.25+ 2.5+ 50 = 52.75

**For Maintenance Resilience Index (MRI)**

*MRI* can be calculated using the following formula:

$$MRI = \Sigma(MCF_i * PMC_i) \quad \text{where } i=1,2,3$$

*eqn 1.4*

From the given value above MCF (Maintenance Class Factors) MCF 1=1, MCF 2=10 and MCF 3=100 and $\sum$ PFC $_I$ =100% using equation 1.4 the results in table 3.3 are calculated

Table 3.3 Maintenance Resilience Index

| i | 1<br>Non Recoverable Fault | 2<br>Recoverable Fault | 3<br>Transparent Fault |
|---|---|---|---|
| Benchmark value | 25% | 62.5% | 12.5% |
| E-Learning value | 25% | 75% | - |

From this results the Maintenance Resilience Index of the E-Learning System can be calculated : i.e MRI=1*25%+10*75%
= 0.25+7.5 = 7.30

**Outage Resilience Index (*ORI*)**

We express *ORI* as the weighted average of *FRI* and *MRI*:
*ORI* = (*wf***FRI*)+(*wm***MRI*) Here *wf* and *wm* are weights associated with fault and maintenance indices respectively, and the sum of *wf* and *wm* is always 1
For the Benchmark:
ORI = (0.5* 30.25) + (0.5* 19)
ORI = 15.12+9.5
ORI= 24.62
For the E-Learning System
ORI = (0.5* 52.75) + (0.5* 7.30)
ORI=26.40+3.65= 30.05

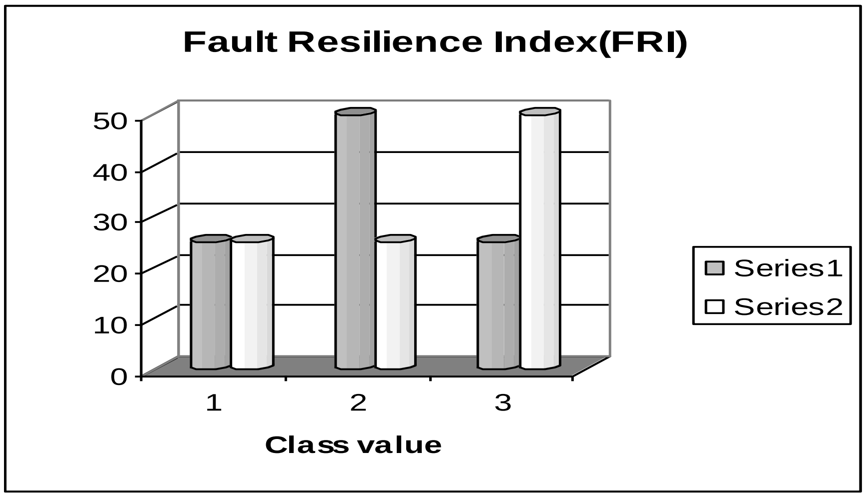

Fig 6 Comparison of Fault Resilience Index

## 3.11 Analysis of the Results

In order to do the analysis of the robustness of the E-Learning system developed we used Robustness Benchmark Metrics on the Secured E-Learning System and another system as the benchmark of the analysis.
In order to find the robustness there is the need to know the value of Fault Resilience Index (FRI) and Maintenance Resilience Index (MRI ) the values of the two were used in determine the value of the Outage Resilience Index according to Maindera et al 2001.

Fig 1 and 2 respectively shows the fault analysis of both the general system and E-Learning and these are the details of exceptional cases that give rise to lack of robustness. The fault analysis first suggested the various exceptional cases to handle within the framework of the development. Then , Fault Resilience Index and Maintenance Resilience index of both system were calculated in order to find the Outage Resilience Index that shows the robustness of the system .For the general system with those eight categories of exception cases the Outage Resilience Index is 24.62 from the calculation and for the E-Learning System it is 30.05. The comparison of this result is shown by fig 4 from Miller 1990 Outage Resilience Index will be a value between 1 and 100, with 1 representing the least robust, and 100 representing the most robust. In all cases, larger values denote better robustness,

Therefore, the significance of this is that the value of the Outage Resilience Index shows that the E-Learning System is more robust than the benchmark since 1 representing the least value and 100 representing the most robust. Fig 5 and.6 shows the variations in the value of the Resilience Index for both Faults and Maintenance in their various classes.

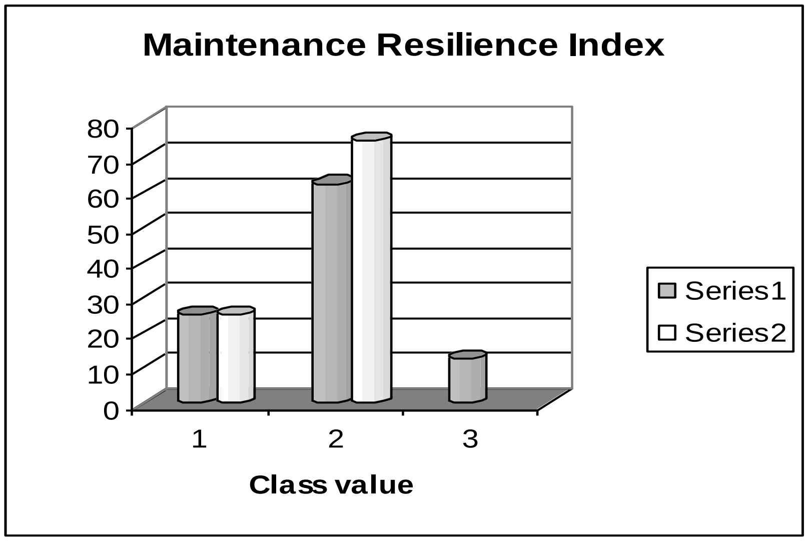

Fig 6 Comparison of Maintenance Resilience Index

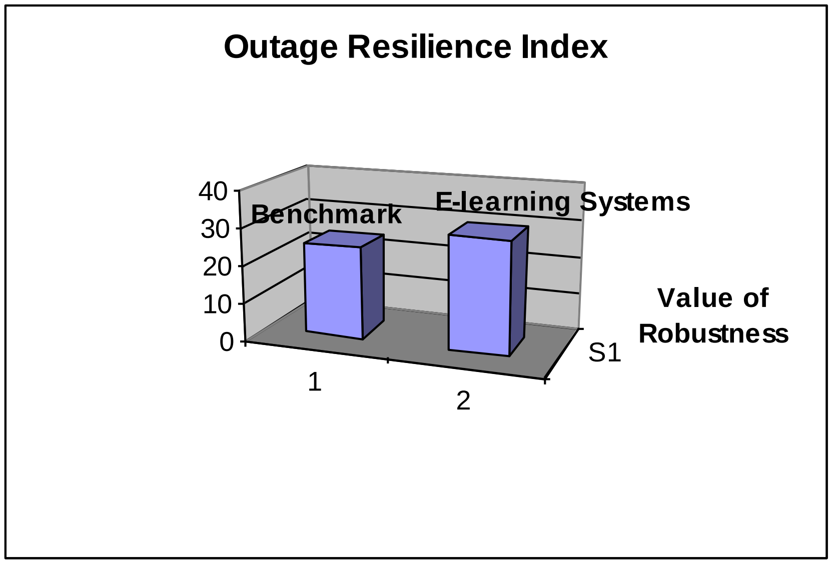

Fig 3.7 Comparison of Outage Resilience Index

**Conclusion**

This work is an attempt to analyze the robustness of a developed E-Learning portal with the use of metrics of robustness and analytical tool that is fishbone analysis.It is however found out that the exception cases that makes software to be less robust could be minimized by the use of exception handling tools also the exception types that occurred within the portal developed are well analyzed and comparison was made with the benchmark.From the work it could be said that E-Learning can only be robust when the exception cases identified in this work are taken care of in the development of the system.

Re**ferences**

[1] Bletchley. O (2002) Data Encryption infohttp://www.bletchleypark.net/

[2] Koopman P , (1997) "*Comparing Operating Systems Using Robust Benchmarks*", Proceedings of the 16th Symposium on Reliable Distributed Systems, pp. 72-79,

[3] *Learning Educational Researcher* Vol. 26, No2, March, pp. 27-23. PGP!

[4] Mukherjee A and D. P. Siewiorek, "*Measuring Software Dependability by Robustness Benchmarking*", IEEE Transactions on Software Engineering, Vol. 23, No. 6, June 1997.

[5] Oweston, R.D. (1997) *The World Wide Web: a Technology to Enhance Teaching and Platforms*". http:// 723-765www.Landahule.Co.UKb pp. 242-259

[6] Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems (IOPADS Proceedings, UNIX Security Symposium III.

[7] Siewiorek D. P.(2003) "Development of a Benchmark to Measure System Robustness",

[8] Proceedings of the 2003 International Symposium on Fault-Tolerant Computing, pp. 88-97.