Bulletin of Business and Economics, 13(1), 465-474
https://bbejournal.com
https://doi.org/10.61506/01.00229

Failures and Repairs: An Examination of Software System Failure

Shoaib Ur Rahman[1], Nouman Arshid[2], Zulfiqar Ali Ayaz[3], Sadia Watara[4], Muhammad Waseem Iqbal[5], Saleem Zubair Ahmad[6], Riasat Ali[7]

Abstract
The central theme of the article is to provide a better knowledge of software system failures and how to assure, maintain, and provide the support software systems that are in production. It includes the results of our search study. We conducted a qualitative analysis of thirty cases: fifteen from public incident reports and fifteen from in-depth interviews with engineers. Understanding and classifying failures as well as their identification, investigation, and mitigation were the main goals of our study. Furthermore, we obtained important analytical insights that are pertinent to the condition of practice as it is now and related problems. It is common for engineers to be unaware of the scaling limitations of the systems they support until those limits are exceeded, and failures have the potential to cascade across a system and cause catastrophic outages. We argue that the difficulties we've discovered may lead to changes in how systems are designed and supported.
Keywords: software failures, incident response, software monitoring, empirical research

1. Introduction
Engineers (*Site Reliability Engineering [Book]*, n.d.) are notified when a software system has an interruption or a decline in functionality or performance so they may look into the problem and fix it. Such an event is referred to in this work as an incident, and the engineering reaction in this context is called an incident response. Organizations may choose to conduct a postmortem investigation and publish an incident report following the mitigation of an incident. Although incidents and incident response activities have not received much attention from academics, they are an important aspect of software engineering, particularly when it comes to maintaining and evolving systems that are already in production and when an outage or degradation needs to be quickly addressed because there could be financial or other consequences involved (Collier et al., 1996).
Assume that a non-call engineer receives notification from the monitoring system when a system's availability—that is, the percentage of successful system queries—drops below a certain threshold. The engineer then investigates the cause of the drop using a range of instruments. Assume that in this hypothetical situation, she learns that the decrease in availability is a result of a system component that has been deployed in a new version, and She starts the deployment's rollback as a mitigating measure. The engineer then kept an eye on the system using the same instruments that she had used to identify the issue. The work of engineers in the following context is referred to as incident response once the problem has been resolved. This is how we define incident response Following the mitigation of an occurrence, companies may choose to publish an incident report and conduct a postmortem investigation. Though incidents and incident response practices have not received much attention from academics, they nonetheless constitute significant software engineering work, particularly when it comes to maintaining and evolving systems that are already in production and having to respond quickly to an outage or degradation because of possible financial or other costs. Before the version can be released, further work may need to be done to find and fix bugs.
It is possible to see work in a variety of software engineering research domains as an attempt to avoid or lesser the chance of events occurring during the evolution of software that is already in production (Huang et al., 2017).
By improving While some work focuses on reducing the time it took to mitigate events as they occur, other efforts focus on needed to monitor the following software systems and evaluate system behavior and status (Shimari et al., 2019). Reducing time to mitigation and preventing problems would both benefit from engineers having a richer grasp of events and incident response procedures as they currently occur. The thirty occurrences that are the subject of this paper's qualitative analysis include fifteen from incident reports that are made available to the public and fifteen from in very deep study of interviews with software engineers who are experienced in incident response. Our analytical findings shed light on the emergence, detection, investigation, and mitigation of problems, all of which have an impact on software systems that support them.

1.1. Software Quality
Nowadays, software quality is being prioritized, and there is a strong emphasis on developing high-quality software solutions. The purpose of software engineering is to reduce development costs while also improving software product quality. Nowadays, developing a functional software program is a demanding undertaking. Before developing a good software product, several software quality characteristics must be identified. When dealing with quality assurance (SQA), develop software development strategies, tools, procedures, and methodologies. The majority of software program is the result of years of collaboration between numerous developers and designers. Nobody knows what occurred. If quality assurance is not satisfied, the project will fail (Salahat et al., 2023).
QA approaches are primarily concerned with the performance phase and related assessment duties (Rasool et al., 2023). This paper is separated divided into two sections: software quality assurance (SQA) and software modification control (SMC), along with a few fundamental tools to aid in the execution of each iteration. "Software development" describes, tracks, and evaluates the

[1] Department of Information Technology, Superior University, Lahore, 54000, Pakistan, msit-f21-007@superior.edu.pk
[2] Department of Information Technology, Superior University, Lahore, 54000, Pakistan, msit-f21-002@superior.edu.pk
[3] Department of Information Technology, Superior University, Lahore, 54000, Pakistan, msit-f21-006@superior.edu.pk
[4] Department of Computer & Mathematical Sciences, New Mexico Highlands University Las Vegas, USA, swatara@live.nmhu.edu
[5] Department of Software Engineering, Superior University, Lahore, 54000, Pakistan, waseem.iqbal@superior.edu.pk
[6] Department of Software Engineering, Superior University, Lahore, 54000, Pakistan, saleem.zubair@superior.edu.pk
[7] Department of Computer Science, Superior University, Lahore, 54000, Pakistan, mscs-f21-016@superior.edu.pk

process of creating software products as they are developed. The authors describe the desired result and demonstrate how the suggested modification gears may help with installation, goal formulation, and a more accurate evaluation of the results.

1.2. Testing

The term "software" refers to the full collection of programming, methods, and processes associated with computer system functioning. Recent papers contain suggestions, approaches, and tools for optimizing software-based processes. However, because we don't fully understand what test case design comprises, these concepts, procedures, and tools must be applied ambiguously during real testing. Rather than concepts, the testing phase focuses on processes and rules. Several elements influencing the evaluation process are mentioned in the literature. One of these aspect progressions is integration testing, as are composite organizational testing, linkages between states and testing, and the utilization and execution of software component tests (Bhatti et al., 2023).

Software testing seeks to analyze a product's qualities or capabilities and decide whether or not it meets the required requirements, and if not, how to improve them. Software testing is the process of running a programmer to find vulnerabilities and generate defect-free software. Software testing is a well-researched topic that has experienced a lot of development work. This field will grow in importance in the future.

A program, set of data, or instructions used to operate a machine and carry out certain activities is called software. It is the opposite of hardware, which refers to a computer's actual hardware. Software is the term for programs, scripts, and applications that are executed on a device. It is a computer's variable component; hardware, on the other hand, is its constant component.

There are two main types of software: application software and system software. An application is a piece of software that accomplishes a specific goal or resolves a specific issue. System software's job is to control a computer's hardware. and give programmers a workspace. System software is software for computers that is designed to run hardware and application applications simultaneously. The system software serves as a conduit between hardware and user applications within the computer system's layered architecture (Hamid, Muhammad, Basit, Hamza, et al., 2022) (Hamid & Iqbal, 2022) (Hamid, Ayub, et al., 2024).

2. Literature review

The PC revolution in the 1990s, which revolutionized the philosophy of software creation, the globalization of software began (Jacob & Prasanna, 2016). Software development is no longer limited to giant corporations, but also to small businesses (Hamid, Iqbal, Muhammad, Basit, et al., 2022) (Iqbal et al., 2024). The unstoppable trend of globalization has increased competitiveness in the software business. However, because of restricted access to trained labor and a long while to market, numerous Software companies started looking for international partners and setting up development facilities. Software development is now as a result of this shift (Hamid, Muhammad, et al., 2023). Depending on the business, employees in a variety of roles and titles—such as some people maybe others—perform incident response work. Give an overview of site reliability engineers' responsibilities and highlight a few of the following topics/topics that we have studied, including system monitoring and incident response, from the viewpoint of a single organization (Hamid, Ibrar, et al., 2024). Because incidents are unpredictable, incident response is briefly discussed in field studies of system. On the other hand, our study's objective was to gather (qualitative) information about many occurrences that occurred in different businesses.

2.1. Software Testing Outsourced

When it comes to large projects, several organizations have already used offshore as one of their key techniques. Companies are ready to contract out projects once the time-to-market approaches grow and competition in the evidence technology sector intensifies, ensuring that new offshore initiatives are launched in regularly. Outsourcing of software for testing to offshore software businesses has risen in popularity to minimize Inefficiencies and postponements associated with multisite growth. Organizations discovered an optimal solution by outsourcing the entire project abroad. Development steps, including coding and testing, have begun to be outsourced to a third-party offshore provider. Fixing software defects remains an exhausting activity, and the efficiency with which existing resources are used has to be enhanced. Before delivering software for commercial use, a software team works to detect flaws in it; alternatively, the cost associated with software errors grows dramatically. Outsourcing testing duties to overseas contractors has become a common strategy for developing high-quality software in recent years (Smite & Wohlin, 2011). Outsourcing onshore tests is replacing traditional ways in the field of software development.

2.2. Software

Software is the collective term for all programmers, techniques, and procedures involved in running a computer system. Current publications include suggestions, approaches, and tools for optimizing software-based processes. However, since we lack a solid understanding of what test case design includes, this is how ambiguous these concepts, approaches, and tools must be implemented in practical testing. The testing phase is more concerned with methods and rules than theory. Several aspects that influence the evaluation process are mentioned in the literature. Integration testing, for example, is one of these aspects progressions, as are composite organizational testing, linkages to the state between growths and testing, and the utilization and execution of software component tests (Bhatti et al., 2023).

2.2.1. Software Testing

Software testing aims to evaluate a product's attributes or capabilities and determine if it satisfies the needed criteria or not, and if not, to enhance them. Software testing is the process of running a programmer to identify flaws and produce software with no defects. Software testing is a highly researched field that has seen a significant amount of development work. This field will become increasingly important in the future.

Software is a group of programmers, data, or instructions used to operate machines and do specific tasks. It is the opposite of hardware, which is the term for the actual parts of a computer. The term "software" refers to any program, script, or application that runs on a device.

2.2.2. Some examples

System software is software that serves as a foundation for other programs. Operating systems (OS), such as Microsoft Windows, Linux, Android, and macOSExamples of system software include industrial automation, software-as-a-service applications, search and game engines, computational scientific software, and industrial automation. Software testing is expensive, but the cost of not testing it is far higher. Product testing is a crucial part of Software Quality Assurance, and some companies allocate as much as 40–50% of their development resources to it. Other types of software include middleware, which acts as a mediator between driver software, which manages computer peripherals and devices, and programming software, which offers the programming tools required by software developers, and system software and applications (Hamid, Muhammad, Iqbal, Bukhari, et al., 2022).

2.2.3. Testing Models

- **Before 1956, the Debugging Process Model**

At that point, the time distinction between testing and debugging was not evident. Both were interchangeably modified. During that time, program checkout, debugging, and testing ideas were not easily distinguished. They used the terms "testing" and "debugging" interchangeably. Alan Turing authored two essays at the time, answering certain problems and defining an operational test for intelligent behavior.

- **Demonstration Process Model (1957-58)**

By that time, the distinction between testing and debugging was evident, and the concepts of detection, identification, locating, and fault repair were in use.

- **Evaluation Process Model (1983-87)**

The National Bureau of Standards Department for Computer Science and Technology developed rules with the goal of targeting. FIPS is a security standard that is employed for verification, validation, and testing. The test activities, analysis, and reviewing come together to give product evaluation with the application lifecycle outlined (Hamid, Iqbal, Muhammad, Fuzail, et al., 2022).

- **Prevention Process Model (Since 1988)**

This time, in terms of mechanism, this differs from the preceding model in that greater emphasis is placed on test design and plan.

- **Describe the same definitions In Software processes**

i. Verification

Verification is a type of testing that is performed at each stage of a product's or software's development.

Verification Static testing is a type of testing that is used to determine whether or not we are building the proper product and whether or not our software meets the needs of the client. Here are some of the actions associated with verification.

- Examining Inspections
- Walkthroughs
- Desk checks

ii. Validation

Validation is a sort of testing that is performed at the end of the software development process.

Validation Testing is also known as Dynamic Testing, and it examines whether we designed the product correctly as well as the client's business demands. Here are some of the actions associated with Validation.

- Testing in the Dark
- Unit Testing Integration
- Testing White Box Testing

iii. Error

It is a circumstance in which the programmer fails to accomplish its intended function. The developer's terminology is incorrect [10]. You will encounter several software faults as a tester throughout the testing process. However, there are so many distinct forms of software faults that you may feel overwhelmed attempting to categorize them appropriately.

iv. Fault

It's a situation that prevents the code from serving the intended function. Any program's undesired behavior on the computer can be attributed to an incorrect step in any process or data description. Software or hardware bugs or defects can result in errors. One definition of an error is a system component that leads to the failure of the system. A program mistake signifies that failure has occurred or must occur. If the system has several components, faults in the system will result in component failure. Because the system's components interact with one another, the failure of one component may be accountable for causing one or more problems in the system (Hamid, Iqbal, Aqeel, Rana, et al., 2023).

2.3. Type of Faults

2.3.1. Algorithm failure

This sort of failure happens when a component algorithm or logic fails to produce the correct output for the supplied input owing to incorrect processing steps. It is simple to delete simply reading the program i.e. disc checking.

Failure is the accumulation of several errors that, in turn, cause software to malfunction and data to be lost in critical modules, making the system inoperable. The issue is discovered by testers and rectified by developers during the SDLC development process. Human error leads to fault.

2.3.2. Computational faults

Arise when a fault disc implementation is incorrect or unable to compute the appropriate result, for example, mixing integer and floating point variables may yield unexpected results.

A computational error occurs when the program is unable to compute the right result. This is typical when using values of multiple data kinds in the same expression.

2.3.3. Syntax problem

This sort of problem arises when incorrect syntax is used in the program. We must utilize the correct syntax for the programming language we are using.

Syntax faults are flaws in the source code that lead to the compiler producing an error message. Examples of these problems include misspellings, inappropriate labeling, and so on. These are shown in a separate error window along with the type of error and line number so that the edit window can correct them.

2.3.4. Documentation Error

The program's documentation describes what it does. As a result, it can arise when a program does not match the documentation.

2.3.5. Overload Fault

For memory-related reasons, we used data structures like an array, queue stack, and so on in our programs. Our software encounters an overload error when they are used to their fullest extent and then some.

2.3.6. Timing Fault

When the system does not respond when a malfunction occurs in the program, this is referred to as a timing fault.

2.3.7. Bug

Bug is Tester terminology. A software bug is a defect, mistake, or an issue with a program. The programs behave in an unforeseen or unanticipated way as a result of this problem, such as crashing or delivering incorrect results (zafar et al., 2023) (Memon et al., 2023).

2.4. Strategy for Testing

One way to create a strategy for the Software Testing Life Cycle (STLC) is via a test strategy. It helps QA teams determine the scope and coverage of their testing. It gives testers a constant, clear perspective of the project. There is very little chance of missing any test activity when a competent test strategy is in place.

2.4.1. Detection of Faults

Fault Detection is only recognizing a problem that has happened; the cause does not need to be removed at this time. There are several quantitative and qualitative strategies for detecting flaws (Hamid, Waseem Iqbal, Abbas, Arif, et al., 2022). The two primary approaches to fault detection are:

- Model-driven reasoning
- Pattern recognition, fault signatures, and classifiers Recognition of patterns

2.4.2. Tentative Design

The tentative design is a well-organized comprehensive strategy to investigate all experimental possibilities conditions or cases. This is carried out to predict the outcome or predicted outcome of any circumstance that may touch or impede the same action. It essentially assists you in avoiding surprises that a programmer may intend to ("O," 2009) (*Software Process Improvement*, n.d.). The test design centers on the tests themselves, including how many must be performed, the test circumstances, and the strategy for testing. The ISTQB blog states that creating and writing test suites for software testing is also a part of test design; nevertheless, this will require accuracy and specificity.

2.5. Principles for Testing Software

These are the established guidelines. Numerous Tests of these values are:

- **Testing Has Been Completed**

It must come to an end someplace. When the danger is under control, it can be stopped. In written English, the phrase 'testing has been finished' is proper and acceptable. It can be used to describe a circumstance in which a certain set of tests or assessments has been completed. For instance: "The team has worked hard all week, and now the testing has been completed".

- **Testing needs to be done at different levels:**

Various testing kinds must be conducted at various levels and utilizing various testing methodologies.

- **Valid data and an invalid test:**

It is also necessary to look over test cases for erroneous or unexpected situations. Regression testing of a sort.

A valid test determines how a system reacts to valid data. It generally tests the program's main goal. Unsupported files or instructions are measured by an invalid test. It examines how a program reacts to incorrect inputs, including the message it displays to the user.

- **Bugs in a Cluster**

This indicates that the highest amount of testing-related errors is present in a small number of modules. In software testing, defect clustering refers to a uneven assignment of problems across the course of the course. Rather, it focuses on a few essential areas of the application. This is particularly true for large systems, when the quantity and complexity of developer errors increase.

- **Test to invalidate the Code:**

Testing is done by running the program to identify faults. Negative test cases are created by defining scenarios in which the system or application should not function as expected and testing it with erroneous, unexpected, or wrong inputs to ensure that it handles these circumstances appropriately.

2.5.1. Beginning testing

As early in the SDLC as feasible, testing efforts must begin.

- Strategy should be tested

Your objective is to be as efficient as possible.

- Plan for testing

A testing strategy is developed for your organization's needs.

- Case Studies

Test cases are created as the programs are being written.

- Data from a test
- Environment for testing.

2.5.2. Phase testing

- Analyze the requirements.
- Test preparation.
- Design and creation of test cases.
- Configure the Test Environment.
- Execution of the test.
- Closure of the test.

2.5.3. Aims of Evaluation

Discovering flaws that the developer may have introduced during development.

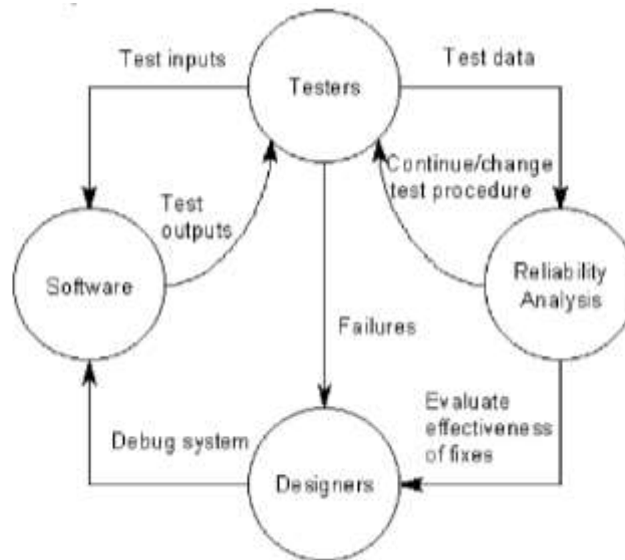

Figure 1: Software Testing

- An increase in self-assurance and quality.
- Ensuring that the product fits both consumer and company criteria.
- To determine if the system performs as intended.
- Making sure that the BRS (Business Requirement Specification) is satisfied (Stratila et al., 2024).

2.5.4. Strategies for Software Testing

A successful product test is achieved by integrating a numerical test case designing approach into a software testing strategy. There are typically four different sorts of strategies. A testing of software the plan is a collection of different actions that must be taken to ensures that the finished result of the following is greatest possible best quality. It is a plan of action that an in-house QA department or an outsourced QA team takes to achieve the quality level that you specify.

- **Black box testing** involves evaluating how well the software works without looking at the core code structure.
- **White box testing** investigates the main logic and code structure of the software.
- Software **components or individual** units are checked to make sure they are functioning as planned.
- **Integration testing** assesses how well many software components are integrated to make sure they function as a cohesive unit.
- Making sure the functional requirements of the software are met is the task of **functional testing**.
- **System testing** comprises evaluating the software system as a whole to make sure it satisfies the specifications.
- Testing the application to make sure it meets the needs of the client or end user is known as **acceptance testing**.
- **Regression testing** is the process of evaluating the program to make sure that no new vulnerabilities have been created after updates or alterations have been done.
- **Performance testing** is testing the programs to determine their performance characteristics, such as stability, scalability, and speed.
- **Security testing** is testing the program to look for bugs and make sure it complies with security requirements.

2.6. Unit Testing

Unit testing uses the test cases that have been generated to run a module in isolation while comparing the expected and actual results of module design. The developer does this testing, and adequate program design understanding is needed. It is the initial level of testing that contributes to the overall system of software testing. It is typically regarded as a strategy for white-box testing.

2.6.1. Advantages

- It is a highly cost-effective method
- It helps in reaching a high degree of internal code coverage
- Error discovery is simple because a single module is tiny.
- Individual pieces may be tested quickly and simply without having to wait for the availability of the other parts.

3. Integration Testing

Integrating testing is carried out to ascertain the accuracy of the interfaces, i.e., inside the modules. It is used to determine whether or not the parameters match on both sides.

There are three types

3.1. Top Down Integration

Is it a progressive approach to creating the framework of a system and adding components? Top-down integration descends the hierarchy, adding one module at a time until full tree integration is attained, negating the need for drivers.

3.2. Bottom-Up Integration

It operates similarly, beginning from the bottom, and a counterfoil is not necessary.

3.3. Integration of a sandwich

This approach involves going from top to bottom simultaneously, resulting in a Centre meeting point of advantages drivers are removed, and the cluster is tested.

3.4. Black box testing

Black box testing is also called functional testing because the black box's operation is only understood in terms of inputs and outputs. Its fundamental objective is to ensure the input is accepted properly and the output is produced correctly. Both the requirements and the functionality are tested.

- **Pros**

Access without a code is required.
Quite effective for long code.
A tester and a coder are made independent of one another.

- **Cons**

A small amount of input can be examined at once.
Designing the test cases requires a clear specification.
Testing can occasionally be ineffective owing to the tester's lack of expertise.

Examples of testing

Analysis of Boundary Values
Robustness Evaluation
Testing for Equivalence Classes
Use of a cause-and-effect graph

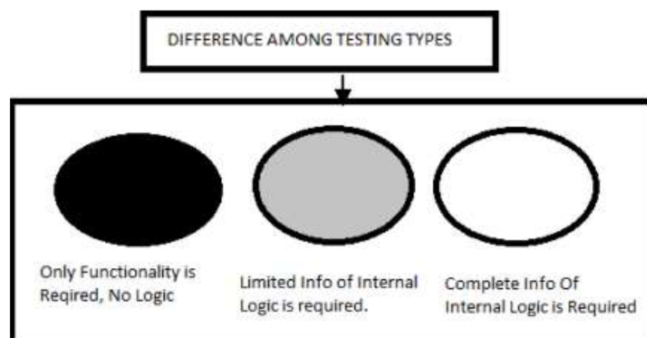

Figure 2: Types of Testing

4. White Box Testing

A black box technique is complemented by white box testing. Another name for it is structural testing. It enables analysis of the code's intrinsic logic; the specifications are ignored in this case. To get the intended outcome, a process of supplying input values and watching how the system works is involved. Unit, integration, and system levels can all be used for white box testing. This kind is effective at both identifying and resolving issues since faults may be found before they result in issues.

Pros

- Code optimization is accomplished.
- Large coverage may be attained during testing thanks to the tester's understanding of the programmer.

Cons

- The price is significantly greater
- Finding hidden faults is exceedingly tough, which is why it occasionally fails.

Example of White Box Testing:

- Path Analysis

- Manage Flow
- Data Stream

5. Gray Box Testing

It uses algorithms to create test cases that are larger than the white box and smaller than the black box.

Pros

- In this case, the tester relies on interfaces and standards rather than internal access.
- The perspective of the user is used during testing.

Cons

- There is not much testing coverage.
- It could take a long time.

H: Acceptance Testing

It is a formal testing process that evaluates whether a system satisfies acceptance criteria and gives users, customers, or other authorized entities the option to accept or reject the system based on user requests, needs, and business procedures.

5.1. Types

- **User Acceptance Testing (UAT)**

UAT is employed to evaluate if a product is meeting the needs of the user appropriately. For testing purposes, specific needs that consumers frequently use are mostly selected. Another term for this is end user testing.

- **Business Acceptance Testing (BAT)**

BAT is employed to determine whether a product to meets the goals and objectives of the firm. BAT's primary goal is firm profits, which are challenging to attain in light of evolving market dynamics and new technological advancements. As a result, updating the current implementation may be necessary, incurring additional costs.

- **Contract Acceptance Testing (CAT)**

A CAT is a contract that specifies that the acceptance test has to be finished within a specific amount of time and pass all acceptance use cases after the product goes live. A contract known as a Service Level Agreement (SLA) specifies that payment will only be made in the event that the Product services fulfill all requirements, signifying that the terms of the agreement have been met. Occasionally, this contract is signed before to the product's release. A clear contract should outline all relevant details, such as testing duration, testing sites, terms for issues found later in the process, payments, and so forth.

- **Regulations Acceptance Testing (RAT)**

RAT is employed to determine whether a product violates the guidelines established by the country's government when it is introduced. Even if this is unintentional, it will negatively impact the company. Generally speaking, the product or application that is going to be introduced to the market has to be within RAT because different countries or regions have different standards and guidelines that are outlined by their regulatory organizations. The product will not be made available in any nation or territory if any laws or regulations are broken there. Only the product's sellers will be held personally accountable if the product is released in spite of the security breach.

- **Operational Acceptance Testing (OAT)**

OAT is non-functional testing is out to confirm a product's operational readiness. Recovery, compatibility, maintainability, and reliability testing are its key topics. Before the product is put into production, OAT makes sure it is stable.

Advantages

- This testing allows the project team to immediately learn about the users' future requirements since it incorporates the users in the testing process.
- Test execution is automated.
- It gives clients engaged in the testing process confidence and satisfaction.

Disadvantages

- Users may occasionally decline to participate in the testing process
- Users should have a basic understanding of the product or application.

5.2. Functional Testing

Black box testing is generally included in functional testing, which can be done automatically or manually. The purpose of functional testing is to:

- **Each application function should be tested:**

Functional testing examines each application function by giving acceptable input and comparing the output to the program's functional requirements.

- **Examine the principal entry function:**

The tester performs functional testing on each entry function of the program to ensure that all entry and exit points are functioning.

- **Flow of the GUI screen testing**

The flow of the GUI screen is examined during functional testing to ensure that the user can navigate across the program.

Verifying the functionality of the application being tested is the aim of functional testing. It is concentrated on:

Basic usability testing is part of functional testing to determine whether the user can freely navigate around the screens without trouble.

- **Mainline functions**

This entails testing the main functionalities and features of the program.
Determining the system's usefulness for the user is the task of accessibility testing.

- **Error Conditions**

Functional testing entails determining whether or not the necessary error messages are presented in the event of an error condition.
The following stages are included in functional testing:

- **Identify test input:** This stage entails determining the functionality that must be tested. This might range from evaluating usability and primary functionalities to error circumstances.
- **Execute test cases:**

This stage entails carrying out the defined test cases and noting the results.

6. Methodology

While the state of practice has been the primary focus of data collection and analysis, a wide range of academic disciplines can benefit from our categorized analytical results and eleven noteworthy observations. Fault tolerance, for instance, is not a particularly new idea. Our analysis reveals a recurring theme: failures can cascade, making the difference between short-lived, localized incidents and large-scale outages. Our study shows that the existing state of practice is far from flawless. One common lesson from postmortem investigations is to "consider cascading failure scenarios more carefully." In a similar vein, testing and verification are well-established study areas, despite ongoing challenges in creating tests and testing environments. To specifically list some of the ramifications of the four findings, we finish this study with a brief discussion of three potential avenues for further investigation.

6.1. Inferring System

Determining the System State The complexity and scope of the following systems that require maintenance increase, making manual notification management less feasible. Many glitch finding techniques based on automated log analysis have been proposed (Hamid, Iqbal, Aqeel, Liu, et al., 2023) and may eventually replace manually generated notifications, even though they haven't been used frequently in the events we've looked at. A lot of earlier work on anomaly identification takes a generic approach, labeling an event as anomalous. Nonetheless, the best monitoring system can be one that can accurately determine the state of components and systems. In order to achieve that, these inference systems would have to understand or deduce relationships between elements and correlate measurements..

6.2. Automated Mitigation

While human response times may be surpassed by automated mitigation systems in certain situations, mitigating actions may exacerbate the problem. Research on self-healing systems (Hamid, Iqbal, Nazir, Muhammad, et al., 2022) and autonomous computing, which aim to provide system support software that can handle systems on its own, is typically linked to automated mitigation. As previously mentioned, by enabling engineer-program rule-based mitigation measures, enhanced state inference may help us move toward more fully autonomous mitigation. Although a thorough investigation of this option is outside the purview of this study, the following is a list of hypothetical states that could be assumed, along with hypothetical automated actions that could be taken in response to those states. Rotate and remove logs more frequently if the fleet's disc use is high and growing.

- Lower the maximum number of concurrent connections if a transient high load is the cause of the fleet's high CPU burden.

6.3. Incident-Specific Dashboards

System dashboards were crucial in the incident investigation process, enabling responders to identify additional symptoms that were either more precise or closer to the underlying cause than the initial symptom that prompted the report. According to our respondents, dashboards covered a particular system in addition to particular information types and sources. They also offered charts with a pre-selected set of metrics and were needlessly incomplete. We saw instances when data was present but not readily available, requiring a more thorough and laborious investigation before it could be determined. Responders would benefit more from an automatically generated dashboard that is event-specific, integrates data from numerous systems, and correlates it. The least amount of help needed would be an algorithm that combines and links anomalous data with a first symptom.

7. Result and Discussion

Every functional transaction must be examined to detect potential failure circumstances. Identified failure states must be addressed by specifying the data integrity criterion to be queried to determine a failure state, as well as the actions to be done when a certain condition emerges to finish the data processing operation.
Every functional transaction must be analyzed for potential failure scenarios. Identified failure states must be handled by describing the data integrity criterion to be queried to determine a failure state, as well as the actions to be taken when a certain condition appears to complete the data processing procedure. Software FMEA should be used to identify probable failure modes, analyze their implications on a system or business process operation, and create reaction mechanisms that avoid failure or lessen the impact of the failure on operational performance. While it may be impossible to anticipate every conceivable failure mode, the development team should create a comprehensive list of likely failure modes in the following manner:

- Create software product specifications that reduce the possibility of probable operational failures.
- In the software performance and post-development phases, evaluate the requirements collected from stakeholders to ensure that they do not add difficult failure states or situations.
- Identify design features that aid in failure detection and reduce failure propagation across a data processing session.
- Create software test scenarios and methods that put the software behaviors related to failure detection, isolation, and recovery to the test.

- Identify, track, and manage any design risks to guarantee product dependability is predictable and supported by software testing.
- Ensure that any potential failures do not result in personal injury or have a significant impact on the system's or operational processes' operation.

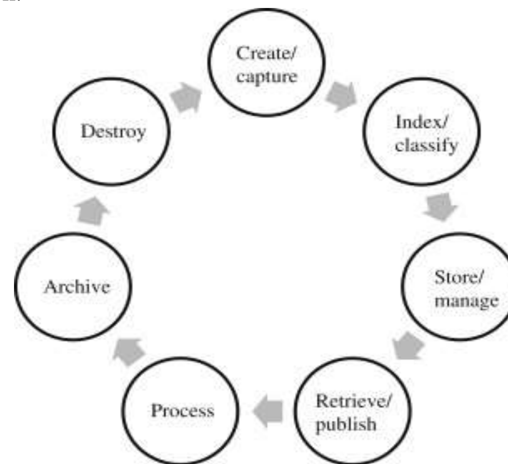

Figure 3: Data Cycle

7.1. Controlling the Statistical Process

Instead of attempting Corrective action is part of Critical Process Control, which compares the output of a technology or service to a standard. while discussing one of the two. We suggest that the largest proportion of the coding rate differences across all classes can be explained by the optimal linear discriminative representation for high-dimensional multi-class data. the data set plus the total of all sub-sets. We show how a multi-layer net model that has characteristics with modern bottomless networks can be automatically reduced using the fundamental iterative slope ascent approach with a customizable optimization rate.

7.2. Offline Quality Control

Offline quality control approaches employ measurements to choose customizable procedures and product variables from which the process or product's output may be deduced. The standard will be decreased, and it will be accompanied by an experimental design that is addressed in the approach and is developed within capital and manufacturing environment limits, such as meat production needs.

8. Conclusion

Based on an investigation of thirty occurrences in this post, we have discussed the current state of practice regarding the occurrence, discovery, investigation, and mitigation of incidents. We have discussed the challenges engineers encounter when creating test environments and test suites that are appropriate for identifying code and configuration errors that result in incidents; the difficulties they encounter when anticipating, testing, and tracking system behavior as scale increases; the expense of manually maintaining monitoring systems, which can have flaws that cause late or missed detections; and the cognitive difficulties involved in analyzing incident data in light of complex architectural designs and diagnostic information. We anticipate that these results will influence future research and development as well as the upkeep and support of software systems already in use.

References

Bhatti, S., Hamid, K., Bashir, A., zafar, zishan, raza, ahmad, & Iqbal, M. waseem. (2023). *Solutions, Countermeasures, And Mitigation Methods For The Rise Of Automotive Hacking*. *56*, 77–99.

Collier, B., DeMarco, T., & Fearey, P. (1996). A defined process for project post mortem review. *IEEE Software*, *13*(4), 65–72.

Hamid, K., & Iqbal, M. waseem. (2022). Topological Evaluation of Certain Computer Networks by Contraharmonic-Quadratic Indices. *Computers, Materials and Continua*, *74*, 3795–3810.

Hamid, K., Ayub, N., Delshadi, M. A., Ibrar, M., Rahim, N. Z. A., Mahmood, Y., & Iqbal, M. W. (2024). Empowered corrosion-resistant products through HCP crystal network: A topological assistance. *Indonesian Journal of Electrical Engineering and Computer Science*, *34*(3), Article 3.

Hamid, K., Ibrar, M., Delshadi, A. M., Hussain, M., Iqbal, M. W., Hameed, A., & Noor, M. (2024). ML-based Meta-Model Usability Evaluation of Mobile Medical Apps. *International Journal of Advanced Computer Science and Applications (IJACSA)*, *15*(1), Article 1.

Hamid, K., Iqbal, M. W., Aqeel, M., Rana, T. A., & Arif, M. (2023). Cyber Security: Analysis for Detection and Removal of Zero-Day Attacks (ZDA). In *Artificial Intelligence & Blockchain in Cyber Physical Systems*. CRC Press.

Hamid, K., Iqbal, M. waseem, Aqeel, M., Liu, X., & Arif, M. (2023). *Analysis of Techniques for Detection and Removal of Zero-Day Attacks (ZDA)* (pp. 248–262).

Hamid, K., Iqbal, M. waseem, Muhammad, H., Basit, M., Fuzail, Z., † Z., & Ahmad, S. (2022). *Usability Evaluation of Mobile Banking Applications in Digital Business as Emerging Economy*. 250.

Hamid, K., Iqbal, M. waseem, Muhammad, H., Fuzail, Z., & Nazir, Z. (2022). Anova Based Usability Evaluation Of Kid's Mobile Apps Empowered Learning Process. *Qingdao Daxue Xuebao(Gongcheng Jishuban)/Journal of Qingdao University (Engineering and Technology Edition)*, *41*, 142–169.

Hamid, K., Iqbal, M. waseem, Nazir, Z., Muhammad, H., & Fuzail, Z. (2022). Usability Empowered By User's Adaptive Features In Smart Phones: The Rsm Approach. *Tianjin Daxue Xuebao (Ziran Kexue Yu Gongcheng Jishu Ban)/Journal of Tianjin University Science and Technology*, *55*, 285–304.

Hamid, K., Muhammad, H., Basit, M., Hamza, M., Bhatti, S., & Aqeel, M. (2022). *Topological Analysis Empowered Bridge Network Variants By Dharwad Indices*.

Hamid, K., Muhammad, H., Iqbal, M. waseem, Bukhari, S., Nazir, A., & Bhatti, S. (2022). Ml-Based Usability Evaluation Of Educational Mobile Apps For Grown-Ups And Adults. *Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)*, *41*, 352–370.

Hamid, K., Muhammad, H., Iqbal, M. waseem, Nazir, A., shazab, & Moneeza, H. (2023). Ml-Based Meta Model Evaluation Of Mobile Apps Empowered Usability Of Disables. *Tianjin Daxue Xuebao (Ziran Kexue Yu Gongcheng Jishu Ban)/Journal of Tianjin University Science and Technology*, *56*, 50–68.

Huang, Q., Xia, X., & Lo, D. (2017). Supervised vs Unsupervised Models: A Holistic Look at Effort-Aware Just-in-Time Defect Prediction. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 159–170.

Iqbal, M. W., Hamid, K., Ibrar, M., & Delshadi, A. (2024). Meta-Analysis and Investigation of Usability Attributes for Evaluating Operating Systems. *Migration Letters*, *21*, 1363–1380.

Jacob, P. M., & Prasanna, M. (2016). A Comparative analysis on Black box testing strategies. *2016 International Conference on Information Science (ICIS)*, 1–6.

O: Economic Development, Technological Change, and Growth. (2009). *Journal of Economic Literature*, *47*(3), 930–945.

Rasool, N., Khan, S., Haseeb, U., Zubair, S., Iqbal, M. waseem, & Hamid, K. (2023). Scrum And The Agile Procedure's Impact On Software Project Management. *Jilin Daxue Xuebao (Gongxueban)/Journal of Jilin University (Engineering and Technology Edition)*, *42*, 380–392.

Salahat, M., Said, R. A., Hamid, K., Haseeb, U., Abdel Maguid Abdel Ghani, E., Abualkishik, A., Iqbal, M. W., & Inairat, M. (2023). Software Testing Issues Improvement in Quality Assurance. *2023 International Conference on Business Analytics for Technology and Security (ICBATS)*, 1–6.

Shimari, K., Ishio, T., Kanda, T., & Inoue, K. (2019). Near-Omniscient Debugging for Java Using Size-Limited Execution Trace. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 398–401.

Site Reliability Engineering [Book]. (n.d.). Retrieved April 25, 2024.

Smite, D., & Wohlin, C. (2011). Strategies Facilitating Software Product Transfers. *IEEE Software*, *28*(5), 60–66.

Software Process Improvement. (n.d.). Retrieved April 25, 2024,

Stratila, S., Glasberg, D., & Mălăel, I. (2024). Performance Analysis of a New Vertical Axis Turbine Design for Household Usage. *Engineering, Technology & Applied Science Research*, *14*(1).

Zanden, J. L. van . (2023). Examining the Relationship of Information and Communication Technology and Financial Access in Africa. *Journal of Business and Economic Options*, *10*(3), 29-39.