### SYSTEM TESTING

**1. Introduction**

System testing is a critical phase of the software development life cycle that evaluates the complete and integrated system to verify whether it meets the specified requirements. In the context of the Automated Vehicle Damage Localization and Severity Estimation Using Deep Learning project, system testing ensures that the integrated components (frontend, backend, machine learning model) work seamlessly together to provide accurate, consistent, and reliable results.

This document outlines the system testing approach, test strategy, test plan, and types of tests performed on the system to ensure quality, reliability, and correctness.

**2. Objectives of System Testing**

The key objectives of system testing are as follows:

* Validate Functionality: To verify that all system functionalities such as image/video upload, webcam input, damage detection, and severity estimation work as intended.
* Ensure Accuracy: To ensure that the damage detection model provides accurate results with a detection accuracy of 91%.
* Performance Testing: To assess the performance of the system under various conditions (e.g., different input sizes, multiple users).
* User Interface Testing: To check that the web interface is intuitive, responsive, and functions properly across different browsers and devices.
* Security Testing: To ensure that the system handles user data securely and prevents unauthorized access or malicious file uploads.

**3. Testing Strategy**

The testing strategy defines the overall approach for system testing. For this project, we employ a combination of manual and automated testing approaches across various testing phases. These phases include unit testing, integration testing, system testing, and acceptance testing. The system will be tested for accuracy, performance, security, and usability to meet the required standards.

**4. Test Environment**

**Hardware Requirements:**

* A system with a minimum configuration of 16GB RAM and a GPU for faster model inference.

**Software Requirements:**

* Python (for backend and model integration)
* Flask (as the web framework)
* YoloV8 model dependencies
* Browser for testing web interface (Google Chrome, Firefox, etc.)

**5. Types of Testing**

**5.1 Unit Testing**

Unit testing focuses on verifying the correctness of individual components in the system, such as the image and video upload functionalities, webcam feed, and the YoloV8 model's response to input data.

**Test cases include:**

* Test image, video, and webcam input validation.
* Test response handling for various damage categories (e.g., damaged door, damaged bumper).
* Test the behavior of the system when an unsupported format or invalid input is provided.

**5.2 Integration Testing**

Integration testing ensures that all individual components (frontend, backend, machine learning model) work together without issues. It verifies data flow between different parts of the system, such as input submission to the backend, model processing, and output display on the user interface.

**Test cases include:**

* Test the communication between the frontend (HTML forms) and backend (Flask server).
* Ensure that the YoloV8 model correctly processes inputs received from the frontend.
* Check if the results from the model are properly displayed back to the user interface.

**5.3 System Testing**

System testing focuses on validating the overall behavior of the complete system to ensure it meets the functional and non-functional requirements. The system is tested in real-world scenarios, simulating user inputs, different types of damage, and varying file sizes.

**Test cases include:**

* Uploading images and videos with varying resolutions and file sizes.
* Using the webcam-based detection functionality.
* Testing the accuracy of damage detection in different lighting conditions and vehicle angles.
* Assessing the performance of the system when multiple users access it simultaneously.

**5.4 User Interface Testing**

User interface testing evaluates the system’s frontend for usability, responsiveness, and cross-browser compatibility. It ensures that the system provides a user-friendly experience, is accessible, and displays the results appropriately.

**Test cases include:**

* Verify that the interface is responsive on different screen sizes (desktop, tablet, mobile).
* Ensure the layout, images, and input fields are properly aligned across browsers (e.g., Chrome, Firefox, Edge).
* Test the clarity of the results display and damage localization on the vehicle image.

**5.5 Performance Testing**

Performance testing assesses how the system performs under different conditions, including varying user loads, large image/video sizes, and different internet speeds.

**Test cases include:**

* Test the time taken to process high-resolution images and large videos.
* Measure response time when multiple users upload files or use the webcam simultaneously.
* Assess the system’s performance under normal and peak loads.

**5.6 Security Testing**

Security testing ensures that the system is secure and protects against potential vulnerabilities, such as unauthorized access and malicious file uploads.

Test cases include:

* Ensure the system only allows the upload of supported image and video formats.
* Test for protection against common web vulnerabilities (e.g., SQL injection, cross-site scripting).
* Verify that sensitive user data, such as images, is handled securely and not accessible to unauthorized users.

**Test Data**

The system will be tested with the following types of data:

* Valid Inputs: High-resolution images and videos of vehicles with various types of damage.
* Invalid Inputs: Unsupported file formats (e.g., text files), low-resolution images, and non-vehicle images.

**Test Results and Reporting**

Each test case will be executed and the results will be documented as Pass or Fail. A test report will be generated at the end of the testing phase, summarizing the performance, accuracy, and stability of the system. The report will also outline any issues or bugs identified during testing and their corresponding solutions or resolutions.

**Conclusion**

System testing is essential to ensuring the quality, reliability, and accuracy of the Automated Vehicle Damage Localization and Severity Estimation system. By performing unit, integration, system, and performance testing, we validate that the system meets its functional and non-functional requirements. The testing process guarantees that the system delivers accurate damage detection, handles multiple inputs efficiently, and provides a seamless user experience, making it a robust solution for vehicle damage assessment in industries such as automotive repair and insurance.

System testing is a critical phase in the software development life cycle that focuses on assessing the overall quality, functionality, and performance of a software system. It is a comprehensive and systematic process that aims to identify defects, ensure that the system meets specified requirements, and verify its readiness for deployment. System testing plays a crucial role in delivering reliable, robust, and high-quality software solutions.

*Importance of System Testing:*

System testing serves as the final gatekeeper before a software system is released to users. It helps identify and rectify defects, glitches, and inconsistencies that might have gone unnoticed during earlier testing phases. By rigorously testing the complete system, organizations can ensure that the software behaves as intended, performs well under various conditions, and meets user expectations.

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub- assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**TYPES OF TESTS**

**Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Unit testing is an essential practice in software development that involves testing individual units or components of a software application in isolation. Each unit, typically a small piece of code or a function, is tested to ensure that it functions correctly and produces expected outcomes. Unit testing plays a pivotal role in maintaining code quality, catching bugs early, and facilitating efficient debugging and maintenance.

*Importance of Unit Testing:*

Unit testing focuses on verifying the correctness of code at its smallest functional level. By isolating and testing individual units, developers can identify issues early in the development process, preventing defects from propagating through the entire application. This practice promotes better code quality, enhances software reliability, and simplifies the process of identifying and fixing defects.

**Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Integration testing is a critical phase in the software development lifecycle that focuses on testing the interactions and collaborations between different components or modules of a software application. This testing phase ensures that the integrated system functions as a cohesive whole, with various parts working seamlessly together. Integration testing identifies and resolves issues related to data exchange, communication, and inter-component dependencies.

*Importance of Integration Testing:*

Integration testing addresses the question: Do the different components of the software work together harmoniously? This phase verifies that the individual units, which have already been tested independently, can successfully collaborate and produce the desired outcomes when combined.

Integration testing plays a crucial role in ensuring that a software application's components collaborate seamlessly to deliver the intended functionality. By identifying and resolving issues related to interactions, dependencies, and data exchanges, integration testing contributes to the overall stability and reliability of the integrated system. A successful integration testing phase enhances confidence in the software's ability to perform as a unified whole and helps avoid integration-related problems in production environments.

**Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is a vital testing methodology in software development that focuses on verifying whether a software application's features and functionalities perform according to the specified requirements. This type of testing assesses the application's behavior in response to various inputs, user actions, and system interactions. Functional testing ensures that the software meets user expectations, delivers the intended outcomes, and aligns with the defined functional specifications.

*Importance of Functional Testing:*

Functional testing addresses the question: Does the software behave as expected? This testing phase helps ensure that the application's functionalities are reliable, accurate, and meet the defined business or user requirements. By validating that the software performs its intended tasks correctly, functional testing contributes to delivering a high-quality and user-friendly application.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

Functional testing is a fundamental aspect of software quality assurance that ensures the software's features and functionalities work as intended. By validating requirements, behaviors, and user interactions, functional testing provides insights into the software's reliability and alignment with user expectations. A successful functional testing phase contributes to delivering a functional, user-friendly, and high-quality application that meets both business goals and end-user needs.

**System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

White box testing, also known as structural testing or clear box testing, is a testing methodology that focuses on examining the internal logic, structure, and code implementation of a software application. Unlike black box testing that evaluates software functionalities from an external perspective, white box testing delves into the underlying code to ensure that all aspects of the codebase, including branches, conditions, loops, and data flows, are thoroughly tested. This methodology aims to uncover defects, vulnerabilities, and potential optimizations within the code.

*Importance of White Box Testing:*

White box testing addresses the question: Does the code behave as expected based on its internal structure? This testing approach is particularly useful for identifying issues that might not be apparent through external testing methods and for ensuring that code paths and decision points are adequately tested.

Key Objectives of White Box Testing:

Code Coverage:

White box testing aims to achieve high code coverage by testing all possible paths, branches, and conditions within the code. This ensures that all logical scenarios are tested, increasing the likelihood of identifying defects.

Error Detection:

By analyzing the code's internal logic, white box testing identifies issues such as incorrect calculations, logical errors, syntax errors, and issues related to variables and data manipulation.

Security Assessment:

White box testing can uncover security vulnerabilities that might be exploited by attackers. It helps identify potential weaknesses in code, such as inadequate input validation or improper handling of sensitive data.

Performance Optimization:

Through code analysis, white box testing can identify bottlenecks and inefficient code segments that impact the application's performance. This information helps in optimizing code for better efficiency.

**Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

Black box testing is a testing methodology that focuses on evaluating the functionality of a software application without examining its internal code, structure, or implementation details. Instead, this approach treats the software as a "black box," where the tester interacts with the application's inputs and examines its outputs to assess whether the desired functionalities work as expected. Black box testing emphasizes validating the software's behavior based on user specifications, requirements, and expected outcomes.

Importance of Black Box Testing:

Black box testing addresses the question: Does the software behave as expected from an end-user perspective? This methodology ensures that the software meets user requirements, functions correctly, and delivers the intended outcomes without requiring knowledge of its internal workings.

**Unit Testing:**

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Unit testing is a fundamental testing practice in software development that involves testing individual units or components of a software application in isolation. Each unit, which could be a function, method, class, or module, is tested to ensure that it behaves as intended and produces the expected outputs for a given set of inputs. Unit testing is a key element of the Test-Driven Development (TDD) approach and plays a critical role in maintaining code quality, preventing defects, and facilitating efficient debugging.

Importance of Unit Testing:

Unit testing addresses the question: Does each unit of code perform as expected on its own? This testing approach is essential for catching bugs early in the development process, isolating defects to specific units, and ensuring that individual components function correctly before they are integrated into the larger system.

**Test strategy and approach**

Field testing will be performed manually and functional tests will be written in detail.

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested**

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

# Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.