TED UNIVERSITY

CMPE492/SENG492

Computer Engineering Senior Project

Software Engineering Senior Project

PyroGuard Fire Detection System

Test Plan Report

05.05.2024

Team Members:

Ecem Sıla Gök (1118505616)

Ece Selin Adıgüzel (1988345282)

Zeynep Beyza Uçar (1004263330)

Doruk Aydoğan (1354737307)

Table of Contents

1. Introduction

PyroGuard Fire Detection System is an end-to-end fire detection system that detects the presence of fire as well as finding the nearest fire extinguisher for rapid response. The scope, methodology, resources and schedule of PyroGuard Fire Detection System testing activities are described in this test plan document. It serves as a detailed guide that ensures that every aspect of the system has been thoroughly examined in relation to both functional and non-functional needs. This document summarizes our testing objectives in many areas and underlines our commitment to using stringent validation procedures to ensure the effectiveness and reliability of the system.

Scope of Testing Activities

In Scope Testing Scope:

- Real-time detection and analysis of fire incidents through video feeds:
  - Testing the accuracy and reliability of fire detection algorithms in real-time scenarios.
  - Evaluating the efficiency of data processing for timely detection of fire incidents.
  - Verifying the responsiveness of the system to promptly analyze and respond to fire incidents captured by video feeds.
- Accurate location identification of detected fires to facilitate rapid response:
  - Testing the precision and accuracy of the location identification subsystem.
  - Verifying the synchronization between fire detection and location identification for prompt emergency response.
- Automatic notifications to emergency responders and system administrators upon fire detection:
  - Testing the reliability and timeliness of automatic notification generation upon fire detection.
  - Evaluating the effectiveness of notification delivery to emergency responders and system administrators.
- Development of a user-friendly interface for system monitoring and management:
  - Testing the functionality and usability of the user interface.

    - Evaluating interface responsiveness and effectiveness in system monitoring and management tasks.

Out of Scope Testing Scope:

- Integration with external fire fighting systems and other third-party emergency services:
    - Not within the scope of testing, as it involves integration with external systems outside the project's control.
- Long-term maintenance and operational adjustments post-deployment:
    - Not within the scope of testing, as it pertains to activities beyond the initial testing phase, such as ongoing maintenance and operational adjustments after system deployment.

2. Test Methodology

The testing method chosen for this project was iterative development, guided by the agile principles of flexibility, adaptability and continuous improvement. In iterative development, the project is broken down into smaller, manageable pieces and iterated in successive cycles, allowing for incremental improvements and refinements throughout the development process.
Iterative testing works well with agile development cycles, as with each change to the project testing processes are repeated which allows for a reliable and functional system overall.

2.1 Fire Detection Algorithm Subsystem

The Fire Detection Algorithm Subsystem serves as the core intelligence of the PyroGuard system, responsible for analyzing live video feeds from cameras to detect the presence of fire and smoke. This subsystem utilizes advanced deep learning models, based on YOLO (You Only Look Once), to perform real-time object detection. Furthermore, this subsystem works closely with the Camera subsystem to scan and examine incoming frames to look for items connected to fire.

Test Objectives

The objectives of testing the Fire Detection Algorithm Subsystem are as follows:

- Functionality Testing: Assess the algorithm's ability to accurately identify fire and smoke objects within video frames under varying environmental conditions.
- Performance Assessment: Evaluate the performance of the deep learning model in terms of detection accuracy, speed, and resource utilization.
- Integration Testing: Ensure seamless interaction and data flow between the Fire Detection Algorithm Subsystem and other subsystems, including the Camera Subsystem, Alarm Activation Subsystem, and Notification System.

2.1.1 Functionality Testing

Input Validation:

The testing process begins by providing a variety of test scenarios encompassing different lighting conditions, camera angles, and fire/smoke intensities to validate the algorithm's robustness. For instance, simulations of daytime conditions with direct sunlight, shaded areas, and artificial lighting, as well as nighttime scenarios with low light levels, will be conducted. Additionally, variations in fire and smoke intensities will be simulated to assess the algorithm's performance under different conditions. These scenarios will be complemented by controlled fire experiments conducted within the university premises using the school's IP cameras. This approach allows for real-time data collection and analysis, providing insights into the algorithm's behavior in authentic fire situations. By conducting controlled experiments, we can evaluate the algorithm's performance under controlled conditions while simulating both day and night scenarios.

Output Verification:

After subjecting the fire detection algorithm to diverse test scenarios, the next step is to compare its detection results with ground truth annotations. This comparison allows us to assess the algorithm's accuracy and reliability in identifying fire and smoke in various environments. By analyzing the discrepancies between the algorithm's outputs and the ground truth annotations, we can evaluate its effectiveness in accurately detecting fire incidents.

Error Handling:

In addition to validating the algorithm's accuracy, functionality testing also evaluates its ability to handle edge cases and unexpected scenarios effectively. This includes assessing its performance in situations such as occlusions, partial visibility of fire or smoke, and occurrences of false positives/negatives. The algorithm's robustness can be evaluated and opportunities for enhancing its error handling and detection capabilities can be identified by putting it through these difficult conditions.

2.1.2 Performance Evaluation

Performance evaluation centers on assessing the efficiency and effectiveness of the deep learning model. This includes:

Speed Testing:

One crucial aspect of performance evaluation involves measuring the time taken by the algorithm to process each frame and detect fire/smoke objects. This assessment ensures that the algorithm can respond in real-time to potential fire incidents, enabling timely alerts and action. By analyzing the processing speed under various conditions, such as different frame resolutions and complexities, we can determine the algorithm's responsiveness and efficiency.

Resource Utilization:

Monitoring system resource usage, including CPU and memory utilization, during algorithm execution is essential to identify any performance bottlenecks or resource constraints. By tracking resource consumption, we can optimize the algorithm's implementation to ensure optimal utilization of available resources while maintaining efficient operation. This analysis helps prevent system slowdowns or crashes due to resource exhaustion, ensuring smooth and reliable performance.

Scalability Testing:

Another critical aspect of performance evaluation involves testing the algorithm's scalability, particularly its ability to handle multiple concurrent video streams from different cameras. By simulating real-world deployment scenarios with varying numbers of cameras and video feeds, we can assess the algorithm's performance under different loads and identify any scalability issues. This testing ensures that the algorithm remains effective and responsive even in environments with high data throughput, enabling seamless integration into large-scale fire detection systems.

2.1.3 Integration Testing

Integration testing verifies the seamless interaction between the Fire Detection Algorithm Subsystem and other subsystems. This involves:

Data Exchange:

One primary focus of integration testing is to validate the transmission of video frames from the Camera Subsystem to the Fire Detection Algorithm Subsystem. This validation ensures that the data exchange process maintains data integrity and synchronization, preventing loss or corruption of critical information during transmission. Through the verification of the smooth transfer of video data between subsystems, we can guarantee that the fire detection algorithm is provided with precise and timely input for analysis.

Alert Triggering:

Integration testing also involves verifying the generation and transmission of alerts to the Alarm Activation Subsystem and Notification System based on fire/smoke detection events. This validation ensures that the subsystem effectively communicates detection results to downstream components, enabling timely alerting and response to potential fire incidents. By confirming the proper triggering and delivery of alerts, we can ensure that the fire detection system functions as intended, providing reliable notification of fire threats to stakeholders.

Compatibility Testing:

Another crucial aspect of integration testing is ensuring compatibility with different camera models, video formats, and communication protocols commonly used in surveillance systems. This testing ensures that the fire detection algorithm can seamlessly integrate with diverse hardware and software environments, enabling broad deployment across various surveillance setups. By validating compatibility with a range of configurations, we can ensure that the fire detection system remains versatile and adaptable to different operational contexts, maximizing its utility and effectiveness.

Test Environment

The test environment includes:

- Simulated Fire Scenarios: Controlled test scenarios simulating various fire and smoke conditions, including different fire types, sizes, and environmental factors.
- Real-world Video Feeds: Incorporation of actual video feeds from surveillance cameras installed in diverse environments to validate algorithm performance under authentic conditions.

Risks

Failure to thoroughly test the Fire Detection Algorithm Subsystem may lead to:

- Missed Detection: Inaccurate or missed detection of fire and smoke objects, resulting in delayed or inadequate response to fire incidents.
- False Alarms: Excessive false positive detections, leading to unnecessary alerts and potential disruption to normal operations.
- System Instability: Performance issues or system crashes due to resource exhaustion or algorithmic errors under high load or adverse conditions.

2.2 Camera Subsystem

Camera subsystem includes all the cameras installed in the monitored area to continuously capture and transmit video footage to the Fire Detection Algorithm Subsystem. This includes various types of camera hardware that is connected to the Fire Detection Algorithm Subsystem through live-video footage and to the database where the according hardware and maintenance information are kept.

The testing of this Camera Subsystem focuses on integration with the Fire Detection Algorithm Subsystem as well as the database, ensuring reliable, real time data transmission to the model.

Integration testing will be implemented, confirming capture quality and accurate data transmission to the Fire Detection Algorithm Subsystem in order to make sure the video capture is consistent, accurate, and without fault or loss.

System Testing for the camera hardware will be done for verifying power management, reliable image and video capture functionalities, zoom settings, focus, color, and exposure accuracy, testing connectivity, battery performance, continuous shooting, memory cards.

Maintenance testing for the alarm subsystem is crucial to ensure its functionality and reliability over time, as the camera hardware and software are expected to perform together. This includes regular checks for physical wear, such as connection cables, sockets, logs and accordingly updating the database system in such conditions.

In the case a problem occurs in the performance of this subsystem, there may be delays in real-time transmission and camera hardware failures. To eliminate such cases, the calibration of camera and validation of network configurations will be tested and verified.

2.3 Alarm Activation Subsystem

The Alarm Activation Subsystem is responsible for an alarm to be activated when a potential fire event is detected. This subsystem is responsible for triggering the relevant alarm to warn the necessary personnel about the fire hazard. It includes the alarm hardware, manual override controls, and security PIN controls. It communicates with the Fire Detection Algorithm subsystem to take fire detection alerts and activates the alarms accordingly.

Integration testing will be used to test functionality and activation responsiveness in this test section. Alarm triggering accuracy when a fire is detected by the Fire Detection Algorithm subsystem is dependent on the accuracy of the data being received, so for this subsystem to be functional, the two subsystems need to be perfectly in sync. Simulated fire events are used for the testing environment to test the alarm system, both within means of a trigger and means of hardware. Additionally, integration with the maintenance and configuration subsystem is also extremely important as the hardware information needs to be up-to-date for accurate detection.

System testing for the alarm hardware includes verifying functionality of alarm such as alarm triggering, notification process, responsiveness of the alarm, connectivity of the alarm, compatibility with the main program, and manual override functionality of an active alarm. To simulate an alarm system, we built a simple Arduino system as a prototype to use during the testing operations.

In the case that this subsystem fails to pass these tests or is not implemented properly, false alarms, delayed or failed alarm activation may occur, causing the fire to grow unnoticed and cause extreme harm.

2.4 Location Identification Subsystem

The Location Identification subsystem is responsible for identifying the approximate location of the fire using image analysis and camera perspective data combined with the coordinate information of the camera that monitors the area. This module works together with the Camera subsystem and the Fire Detection Algorithm to pinpoint the fire.

The testing of this subsystem focuses on the accuracy of location identification, as well as the integration of this subsystem with the Fire Detection Algorithm subsystem, and Fire-Extinguishing Stations subsystem, utilizing accuracy testing and integration testing.

Accuracy testing is used to assess how well the subsystem is able to locate the fire, by comparing the actual position of the fire and the identified location. QGIS (Quantum Geographic Information System) is used to map and analyze the geospatial data to get the actual location of the fire, and then to compare that with the approximated placement of the fire.

A controlled testing environment will be created to evaluate the performance of this subsystem, consisting of pre-determined fire locations so that the outcome can be compared accordingly. Additionally, real-world data will be used to verify the subsystem's efficacy in real-world scenarios.

Integration testing is used to ensure that the information is passed accurately and without loss in between subsystems, as the collaboration of these is crucial. The integration testing will be implemented through triggering events and actions and observing how the rest of the system is affected, as well as creation of test-cases, and monitoring the data flaw.

If this subsystem is not implemented correctly and tested thoroughly, the risks include delayed or failed responses, possibly causing damage and loss in property as well as safety hazards.

2.5 Fire-Extinguishing Stations Subsystem

Once the location of the fire is detected using the Location Identification subsystem, a Euclidean distance algorithm is used to calculate the nearest fire extinguishing station in order to allow for a quick intervention. Once the closest station is determined, this information is used by the Notification subsystem to inform the necessary personnel.

The testing of this subsystem focuses on integration with the Location Identification Subsystem, ensuring accurate distance calculations. To achieve this integration testing and performance testing will both be implemented, confirming the necessary data can be drawn efficiently and correctly, while also evaluating the accuracy and performance of the calculations performed.

Performance testing is used to measure the accuracy and efficiency of the distance calculations to detect the nearest fire extinguishing station. Similarly to the Location Identification subsystem, QGIS is used to map the actual placements of both the fire and the fire extinguishing stations to compare with the output of the system to detect the accuracy. Additionally, since this is a time-critical calculation, execution times in various scenarios are considered in means of efficiency.

Unit testing is utilized through test-cases to test the function that calculates the distance between the fire and the fire extinguishing stations and returns the closest one.

Integration testing is used to verify that the location data for fire is being received reliably from the Location Identification subsystem, as well as the location data for the fire extinguishing stations from the database, achieving a seamless system that transmits data accurately and without loss.

Database management systems are used to manage and update the fire extinguishing station data, when necessary, through the management and configuration subsystem.

By accessing a sample database and creating sample scenarios to simulate real-life fire situations both the integration of subsystems and the performance of the calculations can be tested.

In the case a problem occurs in the performance of this subsystem, there may be delays in responding to the fire.

2.6 Notification System

Notification subsystem is responsible for informing the administrative and security personnel of the facility immediately after fire detection. It includes communication modules, contact databases, and alert dissemination software. It basically receives information about fire detection and location from the program, and sends notifications to relevant people.

To test this subsystem comprehensively, the functionality and timeliness of notifications will be observed and tested. The parameters to test these properties are alert dissemination speed and reliability of communication modules. Since this subsystem is connected to the Fire Extinguishing Stations subsystem, as well as the database, integration testing is also an important part of the testing process of this subsystem.

Integration testing is used to ensure the contact information of the users are kept up-to-date and are functional, so that the notifications can be sent accordingly. It also interacts with the Maintenance and System Configuration subsystem for customization of notifications, such as what type of notifications the users will receive. The messages being sent needs to include the information of the nearest fire extinguisher provided by the Fire-Extinguishing Stations subsystem. All these connections need to be tested through monitoring the data, and ensuring it is transmitted accurately and without loss.

System testing is used to test the system as a whole, to make sure all parts are connected smoothly and without fault.

Unit testing is also employed in various different contexts, such as verifying that the notifications are triggered accurately based on fire detection time stamps or conditions. Tests will be held to see whether the notifications are delivered without any issues through different connection components of the system. Additionally, tests will be performed to confirm that information of sent notifications is logged correctly. Mocks or stubs may be used for testing dependencies.

Risks of not having a functional Notification subsystem include delayed notifications, and failure to reach relevant personnel.

2.7 User Interface and Control Subsystem

To make sure it's functional and easy to interact with, the User Interface and Control Subsystem undergoes detailed testing. We really look into how well it works and easy it is for users to handle, especially focusing on the speed of the interface and how easy the setup tools are to use. There are two main types of tests involved which are; System Testing, which digs into how the subsystem performs under different scenarios and settings, and User Acceptance Testing, which checks if the end users feel comfortable and satisfied with the interface.

Accurate testing conditions are ensured by the use of a specific test environment that closely resembles the production environment. This time range, which is planned for Weeks 6-7 of the testing phase, permits extensive testing while adhering to the project's overall schedule. Control techniques include rigorous interface usability testing using controlled scenarios, user interactions, as well as frequent user input gathering to identify usability faults and opportunities for improvement.

Roles and duties are defined clearly. For example, system administrators are in charge of managing the test environment, conducting system testing, and making sure that requirements are met, while user experience designers optimize the interface for improved usability and user experience.

A few issues with the system that we are aware of might provide difficulties for users. For example, the interface could be tricky or hard to navigate, and setting things up can be complicated, potentially preventing users from tweaking it to their liking. To ensure of everyone can use the system comfortably, it's crucial to conduct thorough testing and continuously work on improvements. We are aware of a few issues with the system that might provide difficulties for users.

Test Scenario: Web-Based User Interface Test

Objective: To verify the user-friendliness and functionality of the PyroGuard Fire Detection System's web-based interface.

Preconditions:

- A modern web browser is installed on the computer used for testing.
- The test user must be logged into the system.

Steps:

1. Login:
   a. The user logs into the system with their username and password.
   b. Expected Result: The system should provide smooth access when the correct credentials are entered.
2. Viewing Fire Alarm Status:

The user navigates to either the "Current Alarms" or "Fire Alerts" section on the dashboard.

   a. Expected Result: The system should show the current statuses of fire alarms in real time.

3. Configuration Settings:
   a. The user goes to the "Settings" or "Configuration" menu to adjust alarm settings, including sound levels and alert options.
   b. Expected Result: The changes should be saved correctly, and the new settings should take effect immediately.
4. System Status Check:
   a. The user clicks on the "System Status" or "Health Check" tab.

    b. Expected Result: The system should display clear status and performance indicators for all system components.
5. Review Notifications and Alerts:
    a. The user opens the "Notifications" tab and selects a specific date range to view past notifications.
    b. Expected Result: The system should accurately list all notifications from the chosen date range in full detail.
6. Logout:
    a. The user clicks on the "Logout" or "Sign Out" button.
    b. Expected Result: The user should be securely logged out and redirected to the login page.

Final Evaluation:

Record any errors or glitches encountered during the test process.

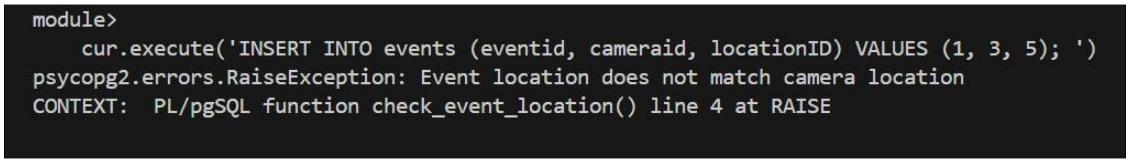
Evaluate the fluidity and response time of the user interface.

2.8 Maintenance and Configuration Subsystem

The Maintenance and Configuration Subsystem allows for an environment to keep the specific data for different locations such as update intervals and maintenance logs to ensure routine updates and maintenance are in order by interacting with all other subsystems, providing an efficient and reliable system. Additionally, it checks for any entries of inconsistencies in newly entered or altered data in the system with database triggers. These triggers can be set to automatically perform actions in certain situations, such as validating data or initiating maintenance tasks.

It is important that the maintenance tools are functional and effective for the system to be robust and reliable. The testing of this subsystem includes testing the functionality of the mentioned database triggers, as well as the ability to manage location-specific data for customization and to perform and schedule maintenance tools. This is mainly done by system and integration testing.

System testing is used to check the functionality of update modules and maintenance scheduling, through the use of configuration settings. Test-cases and boundary testing are especially important tools in testing the functionality of the system, as we can create situations with extreme and invalid conditions as well as usual ones. Since this subsystem is not expected to be used heavily and all at once, the performance of it is not a matter of concern.



```
module>
    cur.execute('INSERT INTO events (eventid, cameraid, locationID) VALUES (1, 3, 5); ')
psycopg2.errors.RaiseException: Event location does not match camera location
CONTEXT:  PL/pgSQL function check_event_location() line 4 at RAISE
```

Figure 1: Screenshot of the RaiseException from the database for an inconsistent entry.

Automized systems such as database triggers are especially useful for catching inconsistent or invalid entries, as can be seen in Figure 1. This allows for a reliable data entry system and helps keep testing conditions relatively simpler.

Integration testing is considerably the most important phase of testing for this subsystem, as the integration of maintenance is required for all subsystems to be kept up-to-date and for the overall system to be functional. Especially the Configuration subsystem is crucial for the Notification subsystem to be customizable and functional. The Maintenance subsystem needs to be integrated with the Camera subsystem, the Alarm subsystem, the Fire Extinguishing Stations subsystem, and the User subsystem as these modules need to be updated as changes happen in the system to ensure all subsystems are compatible with each other.

If not implemented correctly, update failures may happen and inaccurate information can be kept in the system as a result, later causing unreliable outcomes or the inability to detect fire locations.

3. Testing Conditions and Operations

3.1 Test Tools & Environment

Test Tools & Environment	Description
Test Tools	- PyTest, JUnit, Selenium (for automated testing) - Apache JMeter, New Relic (for performance monitoring) - MySQL, PostgreSQL (for database management)
Hardware	- Computers with specified configurations for testing purposes - Hardware setup including servers, cameras, and network devices to mimic real-world deployment scenarios. - Availability of surveillance cameras and network infrastructure for real-time video feed analysis. - Dedicated testing environment with necessary hardware setup
Software	- Windows 8 and above - Python 3.9 (for development and testing purposes) - Version control systems (e.g., Git) for managing codebase revisions
Environment	- Test environment mimicking production conditions without impacting live operations - Availability of surveillance cameras and network infrastructure for real-time video feed analysis - Controlled testing environment for replicating various fire scenarios

Table 1: The table details the tools and environments used for testing a software system.

Test Tools	Description
QGIS (Quantum Geographic Information System)	Employed for accuracy testing of the Location Identification Subsystem by mapping and analyzing geospatial data.
Database Management Systems	Utilized for managing and updating data related to fire extinguishing stations, camera configurations, etc.
Monitoring Tools	Used to track system resource usage, including CPU and memory utilization, during testing.
Communication Modules	Tested for reliable transmission of notifications to emergency responders and system administrators.

Table 2: The table lists various testing tools used in a software system and their specific applications.

3.2 Test Schedule

Test ID	Test Description	Start Date	End Date	Members
TS-001	Fire Detection Algorithm Subsystem Functionality Testing	2024-05-15	2024-05-18	Ece Selin Adıgüzel
TS-002	Camera Subsystem Integration Testing	2024-05-19	2024-05-21	Beyza Uçar
TS-003	Alarm Activation Subsystem System Testing	2024-05-22	2024-05-24	Beyza Uçar
TS-004	Location Identification Subsystem Accuracy Testing	2024-05-25	2024-05-28	Ecem Sıla Gök
TS-005	Notification System Reliability Testing	2024-05-29	2024-05-31	Beyza Uçar & Ecem Sıla Gök
TS-006	User Interface and Control Subsystem Testing	2024-06-01	2024-06-03	Doruk Aydoğan
TS-007	Database Management System Testing	2024-06-04	2024-06-06	Ecem Sıla Gök

Table 3: The table summarizes the schedule and team members involved in various subsystem tests for a software project.

3.3 Control Procedures

Control procedures for testing the PyroGuard Fire Detection System include:

- Regular Review Meetings: Regular review meetings will be conducted to discuss testing progress, identify issues, and adjust testing strategies as needed.
- Documentation and Reporting: Comprehensive documentation and reporting of test results, including test plans, test cases, test logs, and defect reports, will be maintained throughout the testing process.
- Version Control: Version control mechanisms will be implemented to track changes to the system and ensure consistency across testing environments.
- Change Management: Change management processes will be established to manage any changes or updates to the system during testing, ensuring proper documentation and validation of changes before implementation.

3.4 Roles and Responsibilities

Roles and responsibilities for testing the PyroGuard Fire Detection System are defined as follows:

Test Manager: Responsible for overall test planning, coordination, and execution. Ensures that testing activities align with project objectives and timelines.

Test Engineers: Responsible for designing, executing, and reporting test cases. Collaborate with development teams to identify and resolve defects.

System Administrators: Responsible for setting up and maintaining testing environments, including software installations, configurations, and updates.

AI & Image Processing Specialist: Responsible for developing and optimizing the AI models and image processing algorithms used in the fire detection system. This includes training, testing, and refining the models to improve accuracy and performance.

Frontend and Backend Developers: Responsible for developing the user interface (frontend) and backend systems that support the PyroGuard Fire Detection System. Ensure seamless integration between the user interface and underlying system components.

Embedded Software Engineers: Responsible for designing and implementing embedded software systems that control hardware components within the PyroGuard Fire Detection System. This includes firmware development for sensors, actuators, and communication modules.

Algorithm Engineers: Responsible for developing and refining algorithms used in fire detection and location identification subsystems. Collaborate with AI & Image Processing Specialists to integrate algorithms into the overall system architecture.

Database Specialists: Experts responsible for the design, setup, and maintenance of database systems.

User Experience Designers: Responsible for evaluating the user interface's functionality and usability. Identify areas for improvement to enhance user experience.

Stakeholders: Responsible for providing feedback and validation of system requirements and functionality throughout the testing process.

4. References

Son, Hannah. "How To Create A Test Plan (Steps, Examples, & Template)." TestRail Blog. Published August 21st, 2023. Accessed May 4, 2024. URL: https://www.testrail.com/blog/create-a-test-plan/#one-page-test-plan-template-1

Hamilton, Thomas. "Software Testing Methodologies: QA Models." Guru99. Last updated April 12, 2024. Accessed May 4, 2024. URL: https://www.guru99.com/testing-methodology.html