DocDuck Testing & Integration Plan

SWEng Group 1

May 27, 2024

Contents

1 Introduction **4**
1.1 Test Objectives . . . 4
1.2 Scope of Testing . . . 4
1.3 System Overview . . . 5
1.4 Definitions . . . 5

2 Approach **5**
2.1 Assumptions/Constraints, our deadlines both external and internal . . . 5
2.2 Coverage, how we keep track of tests . . . 5
2.3 Test tools, what tools we are using to test . . . 6
2.4 Test type . . . 6

3 Plan **7**
3.1 Test team . . . 7
3.2 Deliverables . . . 7

4 Modules to be tested **8**
4.1 Text Library . . . 8
4.2 Graphics library . . . 8
4.3 Parser and validator . . . 9
4.4 Login Page . . . 9
4.5 Database Library . . . 9
4.6 Server Storage Library . . . 10
4.7 Image Library . . . 10
4.8 Machine Status overview Page . . . 10
4.9 Calendar Page . . . 11
4.10 Machine Information Page . . . 11
4.11 Stock Page . . . 12
4.12 Admin Page . . . 12

5 Features Not in Spec **12**
5.1 Rationale for Exemption 12
5.2 Approach to Managing Potential Integration Issues 13

6 Integration Testing plan **13**
6.1 Objectives 13
6.2 Scope 13
6.3 Strategy 14
6.4 Tools and Technologies 14

Testing & Integration Version	**Release Date**	**Changes**	**Contributors**
0.1	19/02/24	Initial Version. Set up Document and Template	zm926 rw1834
1.0	22/03/24	Initial Release. Filled in sections 1-6. Filled in Intro. Edited grammar	jrb617 lw2380 htsm500 js2140 zm926

1 Introduction

1.1 Test Objectives

Testing of the DocDuck engineering guide will ensure that:

- Each Module is fully functioning as intended.
- Each module is fully compatible with each other.
- Each module successfully communicates with one another.
- That no matter how many extra features are added at its base it is still a fully functional module.
- Feature creep is prevented by giving a set of required targets to aim for first, Before adding extra features.
- There is a guide for the engineers in what to aim for, when creating the modules.

1.2 Scope of Testing

the testing scope of the project will cover:

- The Text Library
- The Graphics Library
- The Parser and Validator Library
- The Image Library
- The Database Library
- The Server Storage Library
- The Login Page
- The Machine Status Overview Page
- The Calendar Page
- The Stock Page
- The Admin Page
- Integration of the Text Library
- Integration of the Graphics Library
- Integration of the Parser and Validator Library
- Integration of the Image Library
- Integration of the Database Library
- Integration of the Server Storage Library
- Integration of the Login Page
- Integration of the Machine Status Overview Page
- Integration of the Machine Information Page
- Integration of the Calendar Page
- Integration of the Stock Page
- Integration of the Admin Page

The XML library additions will not be covered as they cannot be tested, and any additional none essential features will also not be covered in this plan.

1.3 System Overview

DocDuck is intended to provide businesses with an application that increases the efficiency of their maintenance engineering team by providing an easy and efficient way for engineers to access, edit, track and create documentation as well as provide easy and clear communication between admins engineers and operators. The application, DocDuck, is an all in one application for the documentation side of engineering, from tracking diagnostic and calibration dates to having clear and easy access to maintenance history and number of parts in stock all in one convenient place. The three main pillars for DocDuck are affordability, efficiency and usability. DocDuck will be networked on all PC's within a businesses system.

1.4 Definitions

2 Approach

2.1 Assumptions/Constraints, our deadlines both external and internal

Test Library	Start Date	End Date
Text Library	19/02/24	27/02/24
Graphics Library	20/02/24	27/02/24
Parser and Validator Library	20/02/24	27/02/24
Image Library	25/03/24	28/03/24
Database Library	26/03/24	28/03/24
Server Storage Library	03/04/24	08/04/24
Login Page	08/02/24	11/04/24
Machine Status Overview Page	12/04/24	16/04/24
Machine Information Page	16/04/24	19/04/24
Calendar Page	22/04/24	25/04/24
Stock Page	26/04/24	30/04/24
Admin Page	01/05/24	05/04/24

Table 1: Test Libraries Implementation Timeline

2.2 Coverage, how we keep track of tests

Maintaining high-quality software requires not only rigorous testing but also a comprehensive understanding of test coverage. Test coverage is a critical metric that helps us ensure that our testing efforts encompass as much of the source code as possible. To this end, we employ several tools and methodologies that enable us to monitor and improve the coverage of our tests effectively.

Understanding Test Coverage Test coverage refers to the percentage of our source code that is executed when our test suite runs. A higher percentage indicates a more extensive test suite that covers more possible use cases and code paths, reducing the likelihood of undetected bugs. However, it's essential to balance striving for high coverage with the quality of tests to ensure they meaningfully contribute to the application's reliability

Improving Test Coverage Identifying areas with low test coverage is only the first step; the next is to enhance our tests to address these gaps. This process involves writing additional tests for untested code paths and refining existing tests to cover more scenarios. We prioritize testing based on the criticality of code segments, focusing first on the core functionality and high-risk areas.

2.3 Test tools, what tools we are using to test

In the development of our application, ensuring the reliability and correctness of our code base is paramount. To achieve this, we have employed JUnit 5, a powerful and flexible testing framework designed for Java applications. JUnit 5 serves as the foundation for our testing strategy, enabling us to write and execute tests across all the libraries and modules within our project.

JUnit 5 JUnit 5 represents the evolution of Java testing frameworks, bringing forth an amalgamation of new features designed to enhance the testing experience. Key advancements include:

- **Versatility:** A modular architecture that not only promotes greater adaptability but also simplifies integration with a wide array of tools and frameworks, enhancing the versatility of our testing environment.
- **Variability:** The introduction of dynamic tests and improved mechanisms for parameterised testing, facilitate a broader and more nuanced assessment of our code under varying conditions.
- **Manageability:** Augmented assertion mechanisms and a suite of new annotations that streamline the process of test development, making tests more expressive and easier to manage.

TestFX TestFX provides a robust framework for automating the testing of JavaFX applications. It allows us to simulate user interactions with the GUI, such as clicking, typing, and navigating through the application, in a controlled test environment. This enables us to verify that the GUI responds correctly to user inputs and that visual elements behave as expected under various scenarios.

- **Comprehensive Coverage:** TestFX allows for detailed testing of all GUI components, ensuring every aspect of the user interface can be automatically tested for functionality and performance.
- **Repeatability and Reliability:** Automated tests with TestFX can be run repeatedly with consistent conditions, providing reliable results and helping to identify intermittent UI issues that may be difficult to replicate manually.
- **Efficiency:** Automating GUI testing with TestFX significantly reduces the time and effort required for manual testing, allowing for more frequent and thorough testing cycles.
- **Integration with JUnit 5:** TestFX seamlessly integrates with JUnit 5, enabling us to incorporate GUI tests into our existing test suites and workflows, further streamlining the testing process.

2.4 Test type

A robust testing strategy encompasses various types of tests, each targeting different aspects of the software to ensure comprehensive quality assurance. Our project employs a multi-tiered testing approach, leveraging the strengths of each test type to cover the full spectrum of software quality dimensions. Below, we detail the primary categories of tests utilized in our project and their respective roles in our testing ecosystem.

Unit Tests Unit testing forms the backbone of our testing strategy, focusing on verifying the smallest testable parts of the application in isolation (e.g., methods or classes). By using JUnit 5, we efficiently create and execute tests that validate each unit's correctness under various conditions. This granularity allows us to pinpoint defects at an early stage, facilitating swift resolution.

GUI Testing with TestFX An essential component of our testing strategy is GUI testing, for which we employ TestFX. This framework is specifically designed for testing JavaFX applications, enabling us to automate and validate user interactions with the graphical user interface. TestFX allows us to simulate clicks, keystrokes, and navigation through the application, ensuring that the UI behaves as expected in response to user actions. By incorporating TestFX, we can:

- Perform comprehensive testing of all graphical elements and user flows within the application.
- Ensure repeatability and consistency in tests, enhancing the reliability of our GUI testing process.
- Efficiently identify and rectify UI issues, improving the application's usability and user satisfaction.

3 Plan

3.1 Test team

The Testing Team consists of the Testing and Integration Manager, their Deputy and other members. Each member is assigned specific modules to create and perform the tests on. Their generate testing and integration reports which are checked and ratified by the Testing and Integration Manager.

Testing And Integration Manager: Zhihao Ma
Testing Team Members: James Stevenson and Noah Carter.

3.2 Deliverables

The testing of each module is documented in its own individual Testing and Integration Report. This consists of the list of tests performed and the results thereof. In the event of failures, the same test may be present multiple times. In the event of extra features being developed outside of this plan, they, and their associated tests, will be documented therein. This serves to clearly outline the tests performed on the module to ensure robustness and completeness of testing.

4 Modules to be tested

4.1 Text Library

Test #	Feature to Test	Methods to Test	Success Criteria
1	Draw default text box	TextBox Constructor	Default text box drawn with default parameters
2	Draw with defined text content	setContent method	Defined content drawn
3	Draw at defined position	setPostionX and setPositionY methods	Text box drawn at correct position
4	Draw with defined dimensions	setWidth and setHeight methods	Text box drawn with defined dimensions
5	Draw text with defined text colour	setFontColour method	Text drawn with defined font colour
6	Draw text with defined text size	setFontSize method	Text drawn at defined size
7	Draw text with defined font	setFont method	Text drawn with defined font
8	Draw text with defined line spacing	setLineSpacing method	Text drawn with defined line spacing
9	Draw text with defined character spacing	setCharacterSpacing method	Text drawn with defined character spacing
10	Draw text box with defined border width	setBorderWidth method	Text box drawn with defined border width
11	Draw text box with defined border colour	setBorderColour method	Text box border drawn with defined border colour
12	Draw for defined period of time	setDelay method	Text appears for designated time period

4.2 Graphics library

Test #	Feature to Test	Methods to Test	Success Criteria
1	Draw Circles	Circle Constructor	Circle is created with correct position, dimensions, colour, shading, border, duration.
2	Draw Rectangles	Rectangle Constructor	Rectangle is created with correct position, dimensions, colour, shading, border, duration.
3	Draw Regular Shape	Regular Shape Constructor	A regular shape is created with correct position, dimensions, number of sides, colour, shading, border, duration.
5	Draw Custom Shape	Custom Shape Constructor	Custom Shape is created with correct position, points position, number of points, number of sides, colour, shading, border, duration.
6	Draw Line Segment	Line Segment Constructor	Line Segment is created with correct points position, thickness, colour, shading, border, duration.

4.3 Parser and validator

Test #	Feature to Test	Methods to Test	Success Criteria
1	Open XML Files	Parser Class	Sets up the SAX parser API to read and parse XML files.
2	Validate XML Files	Parser Class	Setup SAX Validator to validate the XML files against the schema.
3	Parse XML Files	ParserHandler Class	Uses event handlers to read through the XML file and call events at each element in the file.
4	Store XML Data	ParserHandler Class	Uses a data structure to store all information in the XML file.
5	Parser & Validator Error Handling	ParserErrorHandler Class	Handles all SAX API errors and exceptions.

4.4 Login Page

Test #	Feature to Test	Methods to Test	Success Criteria
1	Correct XML	GUIBuilder	XML Elements are displayed correctly on the login page window, all properties of the elements such as width, height, etc. are correctly displayed
2	Username and Password Entry	TextField	Use of the TextFields for username and password passes the text on to other methods correctly.
3	Login / Signup / Forgot Password Button	Button EventHandler	Buttons pressed result in the correct actions taking place in the eventhandlers assigned for the buttons.
4	Username / Password Verification	Login Verification	Users are only allowed to log in when entered username / password is correct.
5	Incorrect Username / Password		

4.5 Database Library

Test #	Feature to Test	Methods to Test	Success Criteria
1	Stores User Data (Username & Password)	Create database & Store values in database	Successfully stores all data in a suitable format.
2	Encrypts Passwords	Database API methods	Successfully and securely encrypts all passwords and data, so it is unreadable without decryption.
3	Fetches data from database	API Interaction to retrieve data	Can retrieve the data for one user, as well as provide the ability to validate login details when logging in.

4.6 Server Storage Library

Test #	Feature to Test	Methods to Test	Success Criteria
1	Connects and interacts with remote server	SSH or FTP access API methods	Can connect and transfer and receive data from the server successfully with minimal delay.
2	Stores Images and Videos	API Methods to push and pull data	Library can successfully transfer media and other data onto the remote server for storage as well as return the data to the application when requested.
3	Cataloging and Organisation of files	Algorithms to sort/parse data	Can successfully rearrange and organise data in suitable formats for reading and displaying to the application users.
4	Backs up data to the server regularly	Schedular/Runnable Methods	Schedular task which runs every set time period to backup any data to the server which can then be fetched by other users.
5	Fetches data from the server regularly	Schedular/Runnable Methods	Schedular task which runs every set time period to fetch data from the server to refresh information for all users.

4.7 Image Library

Test #	Feature to Test	Methods to Test	Success Criteria
1	Display image	setImage()	The correct image is displayed on the label
2	Display image with defined dimensions	setWidth and setHeight	The label displaying the correct image is sized as defined
3	Display image at defined position	setPositionX and setPositionY	The label displaying the correct image is placed at the defined position
4	Set border around the image label	setBorder()	The border around the image is displayed with defined width and colour

4.8 Machine Status overview Page

Test #	Feature to Test	Methods to Test	Success Criteria
1	able to add new machine	create temporary machine with default settings	machine successfully added to page
2	able to add components to a machine	create a default machine and then add new component to machine	machine component successfully added on
3	able to add multiple notes to a machine and component	create a machine with a single component and apply three notes to each	notes are successfully created

4.9 Calendar Page

Test #	Feature to Test	Methods to Test	Success Criteria
1	display accurate dates	searching for a random date in the calendar	calendar is able to provide the correct date
2	assign events in the calendar	create an event for a date and manually change the calendar to that date to see if the even notification pops up	calendar is able to successfully recognise the date and the correct notification appears
3	calendar is able to repeat events	assign an event a specific day in the week and manually change the calendars day to the same day on each consecutive week 3 times	the same event notification pops up three times

4.10 Machine Information Page

Test #	Feature to Test	Methods to Test	Success Criteria
1	Display Image	setImage()	The correct image of the machine is displayed on the page.
2	Adjust Image Size	setWidth and setHeight	The displayed image is resized according to the defined dimensions.
3	Display Title	setTitle()	The title of the machine is correctly displayed on the page.
4	Display Description	setDescription()	The description of the machine is correctly displayed on the page.
5	Functionality of View Schematic Button	onClick()	Clicking on the View Schematic Button opens the schematic of the machine.
6	View Stock Page	Button/slider to view stock page	Clicking on the button/slider to view stock page navigates the user to the stock page where they can view available parts and their quantities.
7	Sort Machine Information	Sorting machine information	Machine information can be sorted based on different criteria (e.g., machine ID, status, last maintenance date) and the sorting functionality works as expected.
8	Search Machine Information	Searching for specific machines	The search functionality allows users to find machines based on various parameters (e.g., machine ID, status) and displays accurate results.

4.11 Stock Page

Test #	Feature to Test	Methods to Test	Success Criteria
1	Display Components	displayComponents()	All components within the machine are displayed on the stock page, including photos and text descriptions for each component.
2	Display Local Stock	displayLocalStock()	The local stock of each component is displayed accurately on the stock page, showing the stock number for each component.
3	Filter Components	filterComponents()	Users can filter components by types (e.g., electrical, mechanical) on the stock page, ensuring that only relevant components are displayed.
4	Filter Stock	filterStock()	Users can filter components by stock number (e.g., low stock, out of stock) on the stock page, enabling them to identify components needing replenishment.

4.12 Admin Page

Test #	Feature to Test	Methods to Test	Success Criteria
1	Create User Account	check that values entered are stored	values are stored correctly
2	assign permissions	create an account and assign it multiple roles	account successfully cycles through all roles
3	delete accounts	create a temporary account for deletion	account is successfully deleted
4	modify accounts	create an existing account and modify both the account username and password	successfully able to login with the newly modified account

5 Features Not in Spec

In the development of our application, we leverage a widely-used XML library for parsing, and manipulating XML files. This library forms a critical component of our system, facilitating the seamless handling of XML data, which is integral to various functionalities within our application. Despite its importance, the XML library is classified under ”Features Not in Spec” for direct testing due to several justifications outlined below.

5.1 Rationale for Exemption

The decision to exempt the XML library from our internal testing specifications is based on a comprehensive assessment of its reliability, performance, and the nature of its integration within our project. Key factors include:

1. **Established Stability and Reliability:** The XML library in use has been subjected to extensive testing and has demonstrated high levels of stability and reliability across numerous applications and platforms. Its maturity in the software development ecosystem underscores its resilience and performance consistency.
2. **External Validation:** The library benefits from ongoing development and testing by a dedicated community or organization. This external validation ensures that any bugs or vulnerabilities are promptly identified and rectified, thereby maintaining the library's integrity and security.
3. **Standard Compliance:** As a tool that adheres to well-defined XML standards and protocols, the library ensures compatibility and interoperability across different systems and applications. This compliance further reduces the necessity for redundant testing within our project scope.

5.2 Approach to Managing Potential Integration Issues

Acknowledging the exemption of the XML library from our testing spec does not imply a disregard for potential integration issues. To preemptively address any concerns and ensure smooth operation within our application, we adopt the following strategies:

- **Integration Testing:** While the library itself is not directly tested, its integration and interaction with our application are verified through comprehensive integration testing. This ensures that the library functions as expected within our specific use cases.
- **Monitoring and Feedback:** Continuous monitoring of system logs and feedback mechanisms are in place to quickly identify and address any anomalies or issues related to XML processing, ensuring minimal impact on application performance.
- **Version Control:** Regular updates and adherence to recommended versions of the XML library are practiced to leverage improvements and security patches, mitigating risks associated with outdated components.

6 Integration Testing plan

Integration testing is a crucial phase in our software development life-cycle, aimed at evaluating the combined functionality of interconnected modules within the main program. This section outlines our strategic approach to conducting integration tests, ensuring that all components work harmoniously together to achieve the desired outcomes.

6.1 Objectives

- To verify the data flow and interaction between modules are functioning as expected.
- To identify and resolve integration errors and interface mismatches.
- To ensure that integrated components meet the specified requirements.

6.2 Scope

The scope of our integration testing includes all critical modules and interfaces within the application. Specifically, we will focus on:

- Database connectivity and data retrieval mechanisms.
- Interactions between the user interface and business logic layers.
- External service integrations.
- Any modules that have undergone significant changes or refactoring.

6.3 Strategy

Our integration testing strategy encompasses several key elements, designed to methodically assess the interaction between various components:

1. **Top-Down Integration:** We will start by testing the higher-level modules, progressively integrating and testing lower-level modules. This approach facilitates early detection of issues in the major control or decision-making modules.
2. **Bottom-Up Integration:** In parallel, we will test the lower-level modules first, gradually integrating upwards. This is particularly useful for ensuring the reliability of utility and service modules.
3. **Continuous Integration (CI):** Throughout the development process, integration testing will be automated and run as part of our CI pipeline, enabling immediate feedback and early bug detection.

6.4 Tools and Technologies

For conducting integration tests, we will utilize:

- JUnit 5 for orchestrating the test cases.
- TestFX for testing GUI components and interactions.
- Mock frameworks (e.g., Mockito) for simulating external dependencies.