Final Project Part 4 Test Planning

SSW 567

Joseph Carbonell, Prateek Singh Chauhan, Brianna Garland, Veronika Myshkina,
& Neel Hiteshkumar Savani

Introduction:

Overall System and High-Level Goals:

- The overall goal for this system is to make software that is able to take information read from a hardware device and use that information to be able to compare that information against data in a database to ensure that the criteria match on all levels and be able to report mismatches. From there using the software created, the following main goal is to use what we learned over the course of the semester to adequately test the system and report on the findings—specifically, requirement testing, unit testing, performance testing, and test planning.

SDLC Approach:

- As a team, we have worked together and decided on a combination of both a plan-driven and agile approach. In terms of the setup, We all worked together to go through and make a plan of all of the steps of the project using the Gantt chart. Week to week we are working in sprints with plan-driven user stories. Each week each team member is responsible for completing their activities within the suggested time. At the end of each week we had a sprint scrum meeting and essentially went through the work each team member was able to accomplish and what they will work on within the next sprint.

Test Strategy:

- As provided in the assignment, we already have a very structured method for testing this system. At a high level, we are doing four rounds of testing: Requirements Testing, Test Planning, Unit Testing, and Performance Testing. The goal of doing all of these tests is to apply what we have learned and do homework together for one project. We are choosing to do the assignment in this order because after requirements testing before any testing is done, Test Planning should be done before everything else. Within Requirements Testing, we make sure that all of the requirements are clear and that we have all of the information needed to establish clear goals and tasks for the timeline for the project. Next with test planning, we went through and answered all the questions needed to plan for the last two parts. Unit Testing was the first testing on the system we completed. This included doing static coverage testing as well as mutation testing. Lastly is performance testing. To do this we had to create different separate programs using a python library to see how the software created actually performs.

Development Impact on Testing:

- The approach our team has chosen affects our testing process in several ways. The order in which we chose to complete the steps, affects the timeline in which we can get certain steps of the project completed. Our team has chosen to create the implementation and then write and complete the tests. This affects testing since it shows we are not doing test-driven development but are closer to automated testing. This allows us to keep our project closer to agile as well as make sure the tests keep the project from having as many bugs. Also by choosing to do the development in a way where we can split the work between teammates, it allows the entire team to have a full understanding of the system as a whole because the part they developed may not necessarily be the part they complete the testing for.

Test Scope:

Components under Testing:

- We are testing whether the MRZ that was scanned is authentic or not. This is done by testing the scanned check digits against the check digits generated by the system, using unit tests. We are testing the encoding and decoding of the MRZ numbers. We are testing the database information retrieval through a mock function. We are testing the coverage of the system and performing mutation testing. We are also checking the performance of the software system through performance testing.

Components not considered for Testing:

- We are not testing the hardware components of the system. We are not performing usability testing.

Prioritizing Test Criteria:

- To prioritize tests, we assessed their criticality, their value to the performance of the system, and their dependence on the implementation of the system's functions. The tests that had the highest criticality for the system, those that did not require the system to be completed, and those that were the most important for verifying the basic functions of the system had a higher priority.

Test Approach:

Key Factors:

- Factors to be considered:
    - Test plan,
    - Requirements validation, and verification
    - Test case,
    - Defect documentation,
    - Status reports

Key Risks:

- Understanding of requirements changes
- Delays with implementation
- Issues with Python Timer Library due to inexperience
- Incorrect reading of data
- Incorrect interpretation of inputs

Success Criteria:

- Accurately read and interpret MRZ data
- Successfully encode user data into MRZ strings with accurate check digits
- Confirm accuracy of check digits compared to user data

Contingency:

- If the integration fails, we will review the implementation as a team and work to fix it before resuming testing
- If the environment is not stable, the development will be stalled until the environment is stabilized.
- If performance testing shows results below the expected baselines, testing will be suspended and usage/response charts will be generated in order to inform performance improvements.

Pass/Fail Criteria:

- In this part we going to check whether the entered data are correct or not. If the entered data are not exactly what it has then we will consider it as a fail case. Sometimes there are possibilities that many data are blocked by our system then it will consider a blocked case. If the data are correct and data is validated then we will consider it as a pass case. Like we have entered input: **"P<REUMCFARLAND<<TRINITY<AMITY<<<<<<<<<<<<<<;Q683170H11REU6403131M6904133UK128819I<<<<<<9"** and we get output: **['P','REU','MCFARLANDTRINITYAMITY'],['Q683170H1','REU','640313','M','690413','UK128819I'], [1, 1, 3, 9]**
- These three arrays than our criteria is considered as a pass case.

Entry/Exit Criteria:

For Entry Criteria, we will check what the criteria are going to be. Scenarios such as:

- If the entered details correctly?
- Is it what we are looking for to get as input?
- It is a valid passport or it has valid numbers that are what we exactly going to check?

For Exit Criteria, we will decide whether is it correct data or not and that will be our exit criteria.

- We take a string as input and three arrays as output, so our entry criteria are string and exit criteria are three arrays which we are getting from this string.

Testing Criteria and Check Points:

Testing Stage	Criteria	CheckPoints
Requirements testing	• Verify and validate each requirement.	• Each project requirement should have at least one associated requirement for the test.
Unit Testing	• All functionality needs to be covered. • Code Coverage, Branch coverage and Conditional Coverage >=90% • No failed unit test case.	• Software modules must pass all automated unit tests before integration. • Static code Coverage above 90%. • Code rate >= 8.0 out of 10
Integration Testing	• Test suites to ready before testing starts. • All integration test suites should be successful. • Environment and configuration variable needs to be versioned, ready and stable. • No build and deployment failures.	• Environment needs to be ready for testing. • All integration suites should pass before going to the system/performance test. • No failures, Faults/Exception for functionality <3 acceptable. • No build and deployment failures.

Performance Testing	• Load testing and Stress testing need to be completed with at least the below configurations:
	  ○ Users >= 1000
	  ○ Iteration = 3
	  ○ Ramp up = 1 sec
		• Response Time per 1000 user load <= 1 sec
		• Active Threads at max load should be <= 85% of total CPU threads.
		• Heap memory usage, CPU memory usage <= 80%

Test Deliverables:

- Test Deliverables will include the following reports and documentation in stages:

Stage	**Deliverables**	**From**
Deliverables required before testing	Documentation: • Test Plan • Test Design • Release Criteria	Test Manager, Product Owner
Deliverables required during testing	Meetings: • Triage calls with development, Product Manager, and Tester for resolution and discussion Documentation: • Test suites/scripts • Simulators or Emulators used • Test Data used Reports: • Weekly Bug Reports • Code Coverage(Static Code analysis) Reports • Error and Execution logs • Test Play Book Documentation • Testing Progress/Status Reports	Test Manager, Test Team, Development Team (Code Coverage Reports)
Deliverables required after testing	Documentation: • Release Notes • Resolved/Unresolved Bugs List • Runbook/Playbook for	Test Team, Test Manager, Product Manager/Owner

	support team Reports: ● Code Coverage Reports ● Performance Testing Reports ● Integration testing reports ● Security Testing Reports ● Usability Testing Reports ● Release Criteria Reports ● OAT Testing Reports ● Open Bug Reports	

Test Budget:

The testing budget is depending the scope of the project and type of test being done. It also depends on direct and indirect costs. Different types of testing licenses, testing tools, and testing staff will be our direct costs. Indirect costs like training to employees, travel expenses(if any), and other expenses.

Test Tools:

Stage	Tools Required
Configuration Management	Chef (Its code-driven approach means greater flexibility and control of configurations)
CI/CD	Jenkins (Popularly used in corporate for CI/CD for real life implementation)/ Circle CI (For project)
Bug Tracking tools	Atlassian JIRA (Bug Tracking and Task Management)
Unit Testing Tool	Pytest (Stable version: 1.7.3)
Automation Testing	Cucumber Automation testing tool (for BDD) and Pytest (For unit and integration testing in TDD)
Performance Testing Tool	Apache J-meter (real-world Application), Python-timer lib (For project Purposes)
Documentation Management Tool	Atlassian Confluence

Automation Strategy:

- For automation, Jenkins will be used for CI/CD, integrated with CHEF DevOps tool for configuration Management, and Cucumber Automation testing tool for Behavior Testing.

Test Platform:

Test	Platform
Static Code Analyzer	Pylint
Coverage Analysis	Coverage py
Unit Test	Python Unit Test
Performance Test	Python Library Timer

Test Techniques and Types:

- **Coverage testing** to make sure the system works with all manner of input cases to avoid failures caused by outliers
- **Unit testing** to ensure that the code implementation functions as expected throughout the development period. This test also includes **Mutation Testing and Mock Testing.**
- **Integration Testing** to ensure all modules are integrated with the environment as per the design and successfully.
- **Performance testing** to make sure the program is not too slow when working on larger data sets
- **Reliability testing** to ensure that the system will not fail part of the way through a larger data set, and operates consistently

Reports and Metrics to be used:

Stage	Measurement/Metrics	Reports
Requirements Testing	Verification Successful/Failed, Validation Successful/Failed, Trade-Off analysis for requirements	Verification and Validation Reports, Trade-Off Reports
Unit Testing/ Static Code Analysis	Code Coverage, Branch Coverage, Conditional Coverage, Passed/Failed Tests, Code Rating	Static Code Analysis Reports, Test successful/failed reports, Daily Bug reports
Integration Testing	Continuos Integration to build success/failures, Test suites/scripts Execution success/failure counts, Continuous Deployment success/failures	Daily CI/CD reports, Test Suite Execution Reports
Smoke test	Success/Failure of Smoke scenario test	Smoke test report

Performance Testing	Response Time per 1000 user load, Average Response Time over time, Active threads over time	Performance Test Reports
Pre-Production Functional Test (Alpha and Beta Test)	All Functionalities need to work with Production Load and Configuration.	Production Ready Reports, Production Change Request report

Release Criteria:

Release Criteria needs to be met before the Production release is decided.

Reliability:

- <= 2 failures per day expected in operation
- Response Time per 1000 user load <= 1 sec

System Test Complete:

- >= 99% tests executed
- >=98% tests passed
- <= 5 open defects

Static Code Analysis:

- Code Coverage: >= 95%
- Branch Coverage: >=90%
- Condition Coverage: >=90%
- Code rating: >=8 out of 10

Approvals:

- **Product development/test leads, Test team** – They are responsible for implementing the tests, receiving the test results, and taking actions like fixing bugs based on the results.
- **System and Database designers and architects** – They design the software, receive test results, and take action based on test results.
- **Business and marketing analysts** – They define the product features and their expected quality. They also contribute in defining test coverage, analyzing test results, and taking decisions on the basis of those results.
- **Senior-level management, Product managers and Project sponsors** – They contribute to defining test coverage, analyzing test results, and taking decisions on the basis of those results.
- **Project Managers/Owners** – They lead their projects to successful completion while achieving the objectives, balancing required quality, and maintaining schedules and as per budgets. They are also responsible for acquiring the necessary resources for testing activities and working in coordination with Test Managers in planning, monitoring, and controlling test activities.
- **Change Management Team** – Change Management Team will play a vital role in the approval of Test results before going to production. The team weighs the risk of any changes done to the Production Environment based on the test results and decides whether or not new features or changes will be deployed to Production.
- **Technical, and Support staff Leads** – They provide support to the customers and end-users who use the delivered software and have extensive knowledge of product functionality to support incidents. Hence, their approval is needed for test completion and before the feature goes to production so that the Support team can start training and going through all open bugs and Runbook/Playbook documentation before it goes to production. Hence, the Technical Support Lead approval is needed.

Planned Schedule:

Assumption: SDLC method assumed SCRUM Agile where each Sprint is of 15 days leading to 2 Sprints/month. Additionally, the timeline has 1 Sprint Buffer time for any blocker or Major Failure. All Reports will be generated by Scrum Master after every Sprint and Phase Transition.

SDLC Phases/Task	Sprint Accommodation	Start Date	End Date	Approvals/Implementation team
Requirements Elicitation and Analysis				
- Task 1: Domain Analysis	Sprint 1	01/01	01/15	Domain Expert, Business Stakeholders, Development Managers, Requirements Engineer
- Task 2: Requirements Elicitation	Sprint 2	01/16	01/30	Domain Expert, Requirements Engineer, Product Owner,
- Task 3: Requirements Validation and Verification	Sprint 3	01/31	02/14	Domain Expert, Requirements Engineer, Business Stakeholders, Development Manger, Test Manager, Product Owner,
Design				
- Design Planning	Sprint 4	02/15	03/01	Product Owner, Software Architect, Requirements Engineer, Development Team, Test Manager, DevOps Manager
- System Architecture and Design Plan	Sprint 5	03/02	03/16	Product Owner, Software Architect, Requirements Engineer, Development Team, Test Manager, DevOps Manager
- Decide Configs and Bechmarks	Sprint 6	03/17	03/31	Product Owner, Software Architect,

				Requirements Engineer, Development Team, Test Manager, DevOps Manager
- Test Design and Plan	Sprint 6	03/17	03/31	Product Owner, Software Architect, Development Team, Test Manager, DevOps Manager
Development				
- Code Implementation (Modular)	Sprint 7	04/01	04/15	Software Architect, Development Team,
- Resolve Unit test Case Defects	Sprint 9	05/01	05/15	Development Team, Test Team
- Resolve Integration test Failures	Sprint 10	05/16	05/30	Development Team, Test Team
- Resolve Smoke Test Defects	Sprint 11	05/31	06/14	Development Team, Test Team
- Resolve Performance Issue	Sprint 13	07/01	07/15	Development Team, Test Team
Environment Configuration and System Configuration				
- Environment Configuration	Sprint 7	04/01	04/15	Software Architect, DevOps Team
- Resolve Integration Environment Configuration	Sprint 10	05/16	05/30	Software Architect, DevOps Team, Test Team
- Security Configurations and Development (Services and Cloud)	Sprint 8	04/16	04/30	Security Team, DevOps Team
- Resolve Security Configuration and	Sprint 10	05/16	05/30	Software Architect, DevOps Team, Test Team

Development Issues				
- Resolve Performance Issue	Sprint 13	07/01	07/15	Software Architect, DevOps Team, Test Team
Test				
- Static Code Analyses	Sprint 8	04/16	04/30	Development Team, Test Team
- Improve SCA Code Rate	Sprint 8	04/16	04/30	Development Team, Test Team
- Unit Test	Sprint 9	05/01	05/15	Test Team
- Integration Test	Sprint 10	05/16	05/30	Test Team
- Smoke Test	Sprint 11	05/31	06/14	Test Team
- Security Test	Sprint 10	05/16	05/30	Test Team
- Performance Test	Sprint 12	06/15	06/30	Performance Test Team
Pre Production Test (Alpha test)				
- Alpha Test (Functionality test)	Sprint 14	07/16	07/30	Aplha test Users
Production Deployments				
- Change Request/Implementation Approvals	Sprint 15 (7 days)	07/31	08/06	Change Management Team, Scrum Master, Product Owner
- Submission of test reports for CR	Sprint 15	07/31		Scrum Master, Product Owner, Test Manager, Development Manager, DevOps Manager
- Production Env Config Setup	Sprint 14	07/16	07/30	DevOps Team, L3 Support Team
- Production Deployment (Only if CR is approved)	Sprint 15	07/06	08/14	Devlopment Team, L3 Support Team
- Beta Test:	Sprint 15	07/31	08/14	Beta Testers (Users),

Production Env Test after successful Prod Deployment				Product Owner
- Production Ready (Declaration of all changes being live and ready for use)	By End of Sprint 15	08/14		Change Management Team, Product Owner
Buffer Sprint (for any blocker or Major Failure)	Sprint 16	08/15	08/31	

References:

- MRZ Specification Documentation:

https://www.icao.int/publications/Documents/9303_p3_cons_en.pdf

- Pylint:

https://pylint.pycqa.org/en/latest/

- Coverage:

https://coverage.readthedocs.io/en/6.5.0/

- Mutation Testing:

https://medium.com/analytics-vidhya/unit-testing-in-python-mutation-testing-7a70143180d8

- Python Timer:

https://realpython.com/python-timer/

- SDLC Agile:

https://www.interviewbit.com/blog/agile-model/

- Test Planing By Browser Stack:

https://www.browserstack.com/guide/test-planning#:~:text=A%20Test%20Plan%20refers%20to,properly%20%E2%80%93%20controlled%20by%20test%20managers.