Test Plan – Box delivery Project

1. **Introduction**

**The project to be tested is a program that decides pathways for delivery trucks. The it works is it uses sets of arrays to make a map and the same for routes. Upon success the maps and routes are merged as one. For the testing each function should be tested individually as a bare, then afterwards they should be integrated one at a time to each other, adding the getRoute functions one at a time. The expectations is that the output printed out during each tests will correspond with the right symbols as designated by the legend that’s commented out in the mapping.h file.**

1. **Scope**

**The functions to be tested are the getYellowRoute, getGreenRoute, getBlueRoute, addRoute and finally the printMap function. The getRoute functions functionality tests will be tested out to make sure they print the right path and will be tested in conjunction with the printMap function. This will allow us to see that the bare basics of routes are printed out. Also, they will allow us to see how well each function is to integrate with eachother. On top of that, as they integrate with eachother, we will be able to see how well the program can modify the structure variables. What won’t be tested is its load capacity since only one user can use the program so theres no need to yet and security won’t be an issue for now either so it also won’t be tested.**

1. **Test Strategy**

**When testing this program out I believe testing each part individually should be done to test all the positive test cases and to get a bare basic background of the program. This will also allow us to make negative test cases that can branch off to test other aspects of the program. Then from that, requirements can be decided in which they will deem the program be able to pass the acceptance tests and at least 85% of the tests should pass. After the requirements have been decided a traceability matrix made on excel can be formed or on Microsoft word in which we incorporate both the requirements and the tests together.**

**For positive testing, first the printMap function will be tested out on its own in conjunction with the populateMap function. Then the individual getRoute functions will be tested. First to be tested is the getYellowRoute function on its own without the getBlueRoute nor the getGreenRoute interfering. The getYellowRoute function will be tested in conjunction with the printMap function but then it will be tested with getBlueRoute function. Afterwards the same will be repeated with the getGreenRoute function instead of the getBlueRoute. Then the getGreenRoute function will be tested in the same manner but only with the getBlueRoute function. Afterwards, once we know the basics of this programs functionality works we will begin making test cases.**

**The test cases should measure how well the program handles negative test cases in which numbers that surpass the entries are handled and from these negative test cases we can also incorporate other elements to test out other aspects of the program such as stress/volume tests, integration tests or end to end testing. From these tests finally we can decide on what requirements are needed for the test cases and go over them as a team to see which ones are reasonable which ones can be discarded based on their traits so we don’t test similar data more than once. Once the test cases and requirements are made the traceability report can be done as well.**

**Through the traceability matrix which will allow us to see which tests can go with which requirements, shorten our test time by integrating multiple tests that align with the same requirement together and test them as one.**

1. **Environment Requirements**

**The testing environment for the mapping code should meet the following hardware and software requirements:**

* **Hardware:**
* **Laptops or computers capable of running Visual Studio for code development and execution.**
* **Software:**
* **Git and Jira repositories should be set up to manage version control and issue tracking.**
* **Visual Studio: The primary development environment required for coding, compiling, and debugging the mapping code.**
* **Test Data:**
* **The code may require specific test data to validate different scenarios, such as different sizes, weights, and destinations. This test data will be created and added to the repository on GitHub for testing purposes.**
* **Test Tools:**
* **Visual Studio will be utilized, along with its built-in testing tools, for conducting unit testing and analyzing code coverage.**

1. **Execution Strategy**

The execution strategy for the mapping code involves the following steps:

* 1. **Development and Compilation:**
* Example: Ensure that the code is developed and compiled successfully without any critical errors. For instance, using the Visual Studio environment, the code is written, debugged, and compiled to ensure its correctness and functionality.
  1. **Test Case Coverage:**
* Example: Create a wide range of test cases to cover different scenarios. For instance, test cases can include shipments with varying weights (e.g., 20 kg, 200 kg, 500 kg), box sizes (e.g., 0.5 cubic meters, 1.0 cubic meters), and destinations within the city map (e.g., 12L, 8Y). Test cases should cover both positive scenarios (valid inputs) and negative scenarios (invalid inputs or edge cases).
  1. **Defect Severity Classification:**

Example: Classify defects based on severity levels to prioritize resolution efforts. For example:

* Critical: Defects that cause the system to crash or produce severe anomalies, rendering the code unusable. An example could be a situation where the code fails to handle invalid inputs, leading to a program crash.
* High: Defects that result in a lack of program functionality and may require workarounds. An example could be the code failing to consider weight limitations properly, leading to incorrect package assignments.
* Medium: Bugs that degrade the quality of the system but can often be worked around to achieve the desired functionality. An example could be the code not considering optimal routes, resulting in slight diversions for package delivery.
* Low: Minor errors, unclear messages, or other issues with minimal impact on functionality. An example could be the code displaying ambiguous error messages for invalid inputs.
* Feature Request: Suggestions for improving the user interface or enhancing functionality without affecting core system behavior. An example could be adding a feature to optimize truck allocation based on available space and proximity to the destination.
  1. Test Reporting:

Example: Generate reports to track testing progress and code quality. For instance:

* Test Execution Report: Provide a summary of the execution of test cases, including the number of tests conducted, passed, and failed. This report can highlight the progress of testing and the overall stability of the mapping code.
* Defect Report: Outline the identified defects during testing, including their severity levels and steps to reproduce them. This report helps the project manager, development team, and quality assurance team understand the issues and prioritize bug-fixing efforts.
  1. Collaboration between Quality Assurance (QA) and Development Team:

Example: Adopt effective collaboration between QA and the development team for defect resolution. For instance:

* QA team members report defects with detailed steps to reproduce and share them with the development team.
* The development team reviews the reported defects, prioritizes them based on severity and impact, and assigns developers to fix the bugs.
* QA and the development team collaborate to verify the fixes, ensuring that the defects are resolved in a timely manner.

1. **Test Schedule**

**The test schedule for the mapping code involves planning and estimating the time required for testing activities. The schedule aims to provide a timeline for testing completion and deliverables. Here is an example of a test schedule:**

**1. Test Planning Phase (2 days):**

* + **Define the testing objectives, requirements, and strategies.**
  + **Identify and prioritize test cases based on functionality, edge cases, and critical scenarios.**
  + **Prepare the necessary testing resources, such as test data and testing tools.**

**2. Test Case Preparation (2 days):**

* + **Develop and document test cases covering various scenarios and requirements.**
  + **Create positive and negative test cases for weight, box size, and destination inputs.**
  + **Design test cases for edge scenarios, such as the maximum weight or volume capacity of trucks.**
  + **Review and validate test cases with the development and QA teams.**

**3. Test Environment Setup (1 day):**

* + **Configure the required hardware and software environment for testing, including Visual Studio and testing frameworks.**
  + **Set up test data and repositories for storing test cases and test results.**
  + **Ensure that all necessary tools and dependencies are installed and functioning correctly.**

**4. Unit Testing (3 days):**

* + **Execute unit tests for essential functions and algorithms within the mapping code.**
  + **Validate the accuracy and correctness of individual units or components.**
  + **Ensure that all unit tests pass successfully, indicating the code's functional integrity.**

**5. Integration Testing (5 days):**

* + **Conduct integration testing to verify the interaction and compatibility between different modules or components of the mapping code.**
  + **Test the integration of truck routes, package assignment, distance calculations, and route optimization.**
  + **Execute test cases that simulate real-world scenarios and validate the overall system behavior.**

**6. Defect Resolution (Ongoing):**

* + **Collaborate with the development team to address and fix any defects identified during testing.**
  + **Prioritize and assign developers to resolve critical and high-severity defects promptly.**
  + **Verify and retest the fixes to ensure they have been effectively resolved.**

**7. Test Reporting and Documentation (2 days):**

* + **Generate test reports summarizing the testing activities, including test execution status, pass/fail results, and defect metrics.**
  + **Document the identified defects with detailed descriptions, steps to reproduce, and severity levels.**
  + **Share the reports with project managers, development teams, and quality assurance teams for review and decision-making.**

1. **Control Procedures**

* Reviews
* Regular code reviews:
* Regular code reviews will be conducted by team members.
* The purpose of code reviews is to ensure the quality of the codebase and adherence to coding standards. It helps identify potential issues, improve code readability, maintain consistency, and promote collaboration among team members.
* As a group, we schedule regular code review sessions where team members review each other's code. This can be done individually or in pairs, depending on the complexity and size of the codebase.
* During the code review sessions, team members can provide feedback, suggest improvements, identify bugs or potential issues, and share best practices.
* The code review process can also be facilitated using tools like pull requests in version control systems (e.g., Git) to track and manage code changes and comments.
* Test Case Reviews:
* Test case reviews ensure the completeness and accuracy of test cases developed by the designated group members.
* As a group, we organize test case review sessions where team members collectively review and validate the test cases.
* During the test case review, focus on verifying that test cases cover all the required scenarios, adequately test the software functionality, and provide clear steps and expected results.
* Team members can provide feedback on the test cases, suggest improvements, and ensure that the test cases align with the project requirements and objectives.
* It's essential to document any changes or updates made to the test cases during the review process.
* By conducting regular code reviews and test case reviews as a group, you foster a collaborative and quality-focused environment. It allows team members to learn from each other, identify and address issues early in the development process, and ensure that the software meets the desired quality standards. Additionally, these reviews help improve the overall codebase and test coverage, leading to more robust and reliable software.
* Test Environment
* A dedicated test environment will be set up to simulate the production environment for testing purposes.
* The test environment will be regularly monitored and maintained to ensure its stability and availability.
* For this project, the test environment would consist of the following:
* Software:
* Visual Studio IDE: The integrated development environment where you write, build, and debug your code.
* Testing frameworks: Any testing frameworks or libraries planned to be used for unit testing or integration testing.
* Bug tracking tool: A tool to track and manage reported defects.
* Hardware:
* Laptops or computers: The hardware on which you will run the software and perform testing.
* Network:
* Local network: The network connectivity required for testing communication protocols, if applicable.
* While the test environment in this project may be relatively simple compared to a larger-scale production environment, it is still important to ensure that the necessary software, hardware, and network configurations are properly set up and maintained for testing. This includes ensuring that all team members have the required software and hardware specifications to run the software and perform their testing tasks effectively.
* Additionally, depending on the specific requirements of your project in the following weeks, we may also consider setting up virtual environments or using tools to simulate different system configurations, operating systems, or network environments.
* The main goal of having a dedicated test environment is to provide a controlled and consistent environment for testing, allowing to identify and reproduce issues, validate software functionality, and ensure that your software performs as expected across different scenarios and configurations.
* Bug Review Meetings and Progress Monitoring
* Bug review meetings
* Bug review meetings are scheduled sessions where the team discusses and prioritizes reported defects or bugs.
* During these meetings, team members come together to review the reported bugs and assess their impact on the software.
* The purpose of bug review meetings is to analyze and understand the nature of the reported bugs, determine their severity and priority, and decide on the appropriate actions to address them.
* The meeting participants can discuss the root causes of the bugs, potential fixes, and any additional information needed to reproduce or investigate the reported issues.
* Bug review meetings also provide an opportunity for knowledge sharing, collaboration, and decision-making regarding bug resolutions. Defects will be tracked using a bug tracking tool, and their status and progress will be monitored regularly.
* Progress Monitoring:
* Progress monitoring involves keeping track of the status and progress of the testing activities throughout the software development lifecycle.
* This includes monitoring the execution of test cases, tracking the identified defects, and ensuring that testing milestones and objectives are being met.
* The progress monitoring process involves regular updates and communication among team members to stay informed about the testing progress, any challenges or delays encountered, and the overall health of the testing effort.
* Test execution progress can be tracked using test management tools or spreadsheets, where the status of test cases (e.g., pass, fail, in progress) and the number of defects found are recorded.
* By monitoring the progress, the team can identify bottlenecks, prioritize testing efforts, and take necessary actions to address any issues that may arise during the testing phase.
* Progress monitoring also helps in providing team members with accurate and timely updates on the testing progress and any potential risks or concerns that may impact the project timeline.
* Change Request
* Change requests related to the testing activities will be managed through a defined change control process.
* The impact of change requests on test planning, test cases, and test execution will be assessed and incorporated accordingly.
* Defect Reporting
* Defects will be reported in a standardized format by a designated group member.
* Clear and detailed information, including steps to reproduce the defect, will be provided for effective bug resolution.
* The following steps will be followed for defect reporting:
* Defect Identification:
* Defects will be identified during test execution or any other phase of the software development lifecycle.
* Testers and team members will actively monitor and identify anomalies, deviations, and issues.
* Defect Documentation:
* Testers will document the defects using the designated bug tracking tool or system.
* Sufficient information will be provided, including steps to reproduce the defect, observed behavior, and expected behavior.
* Defect Prioritization and Classification:
* Reported defects will be prioritized based on their impact, severity, and urgency.
* A classification scheme will be used to categorize defects, such as critical, high, medium, and low priority.
* Defect Assignment and Ownership:
* Reported defects will be assigned to respective team members responsible for defect resolution.
* Clear ownership will be established to ensure accountability and timely resolution.
* Defect Tracking and Monitoring:
* The bug tracking tool or system will be used to track and monitor the status and progress of defects.
* Regular updates will be provided, including the current status, assigned resources, and estimated resolution time.
* Defect Resolution and Verification:
* Assigned team members will work on defect resolution, addressing the reported issues.
* After fixing a defect, the resolution will be verified through appropriate testing and validation.
* Defect Closure and Documentation:
* When a defect is successfully resolved and validated, it will be marked as closed in the bug tracking tool.
* Closure information, including the fix details and verification results, will be documented.
* Effective defect reporting ensures that identified issues are properly documented, assigned, and resolved in a timely manner. It facilitates communication between team members and enables efficient tracking and monitoring of defect resolution progress.
* Test Execution Schedule
* A test execution schedule will be created, outlining the test phases, milestones, and timelines.
* Test execution progress will be monitored against the schedule, and any deviations or delays will be addressed promptly.
* The following elements will be included in the test execution schedule:
* Test Phases:
* The test execution schedule will define different phases of testing, such as unit testing or integration testing.
* Each phase will have specific objectives and criteria for test completion.
* Milestones:
* Key milestones will be identified and marked in the test execution schedule to track progress and assess the overall testing timeline.
* Milestones can include completion of specific test phases, major deliverables, or important events in the testing process.
* Test Activities:
* The schedule will outline the specific activities to be performed during each test phase, such as test case creation, test execution, defect reporting, and regression testing.
* It will provide a clear sequence of tasks and their estimated durations.
* Resource Allocation:
* The schedule will identify the resources required for each test phase, including testers, test environments, test data, and tools.
* Resource allocation will be planned to ensure availability and avoid conflicts or bottlenecks in the testing process.
* Dependencies and Constraints:
* Any dependencies between test activities or external factors that may impact the testing schedule will be identified and addressed.
* Constraints such as availability of test environments, dependencies on external teams or components, or time limitations will be considered.
* Timelines and Deadlines:
* The schedule will include estimated start and end dates for each test phase and activity.
* Deadlines for completing specific milestones or achieving testing objectives will be defined and communicated to the team.
* Monitoring and Reporting:
* Progress against the test execution schedule will be regularly monitored and tracked.
* Deviations or delays from the planned schedule will be identified, analyzed, and appropriate corrective actions will be taken.
* The test execution schedule serves as a roadmap for the testing activities, ensuring that testing is conducted in an organized and structured manner. It helps in managing resources effectively, meeting project timelines, and providing visibility into the progress of testing activities.
* Documentation
* Test artifacts, including the test plan, test cases, test scripts, and test results, will be appropriately documented.
* Test reports summarizing the testing activities, test coverage, and defects will be generated and shared with the professor.
* The following aspects will be mainly addressed in the documentation process:
* Test Plan:
* A detailed test plan will be documented, outlining the objectives, scope, and approach of the testing activities.
* It will include information about the test environment, test data, test schedule, and the roles and responsibilities of the testing team.
* Test Cases:
* Test cases will be documented to provide step-by-step instructions for executing specific test scenarios.
* Each test case will include a description, preconditions, test steps, expected results, and any necessary test data or inputs.
* Test Scripts:
* If automated testing is employed, test scripts will be documented to provide instructions for automated test execution.
* Test scripts will include the necessary code or commands to perform specific test actions and validate the expected results.
* Test Results:
* Test results will be documented to record the outcome of test executions, including both successful and failed tests.
* For failed tests, detailed information about the encountered issues or defects will be documented, including steps to reproduce the problem.
* Test Reports:
* Test reports will be generated to summarize the testing activities, progress, and key findings.
* The reports will provide an overview of the executed test cases, test coverage, defects identified, and their status (open, resolved, closed).
* Risk Management
* Risks associated with testing will be identified, assessed, and documented. Each part would be addressed by designated members based on their strength.
* Mitigation strategies and contingency plans will also be established to address identified risks by group members.
* Effective risk management ensures proactive identification and mitigation of potential risks, reducing the likelihood of project disruptions, delays, or failures. By addressing risks early in the testing process, the project team can make informed decisions, allocate resources effectively, and ensure the successful completion of the testing activities within the allocated time and resources.
* Communication
* Regular in-person team meetings and WhatsApp communication channels will be established to discuss progress, address any challenges or issues encountered during testing, and ensure effective collaboration among team members. The focus of communication will be on internal team discussions and updates.

1. **Functions To Be Tested**

On this part of the project (milestone 2), the following functions and features need to be tested:

* populateMap(): This function which creates and populates a map with the position of all buildings should be tested to ensure that the map is correctly initialized and populated.
* getBlueRoute(), getGreenRoute(), and getYellowRoute(): These functions return routes for different colored trucks. They should be tested to verify that the correct routes are generated.
* addRoute(const struct Map\* map, const struct Route\* route): This function adds a route to a map. It should be tested to ensure that the route is correctly added to the map.
* printMap(const struct Map\* map, const int base1, const int alphaCols): This function prints the map. It should be tested to verify that the map is printed correctly after any modification, including the symbols representing different elements.
* Additionally, the interaction between these functions should be tested to ensure the correct flow of data and functionality. This includes verifying that the routes are added to the map correctly and that the final map is printed as expected.

1. **Resources and Responsibilities**

|  |  |  |
| --- | --- | --- |
| Team Member | Role | Responsibilities |
| Mostafa H.. | Project Lead / test coordinator | Overall responsibility for managing the testing activities and project coordination.  Coordinating with other project members and managing communication related to testing.  Collaborates with Wilson to understand testing objectives and requirements.  Works with Lebna to ensure test plans and documentation meet project standards.  Coordinates with Sasawat to assign testing tasks and monitor progress.  Communicates with Samin to address defect management and prioritize bug fixes.  Collaborates with Radmehr to ensure appropriate test data is available. |
| Wilson Sum | Test Planner | Analyzing project requirements and identifying testable features.  Creating a test plan and defining the testing strategy.  Designing test cases and test scenarios.  Collaborates with Mostafa to define the testing strategy and objectives.  Works with Sasawat to design effective test cases and test scenarios.  Collaborates with Lebna to document the test plan and test design.  Communicates with Radmehr to identify test data requirements. |
| Sasawat Yimleang | Test Executor | Executing test cases and scripts based on the test plan.  Analyzing test results and reporting defects.  Providing insights and recommendations for test improvements.  Collaborates with Mostafa to understand assigned testing tasks.  Works with Wilson to clarify test case expectations and requirements.  Collaborates with Samin to report defects and provide detailed defect information.  Communicates with Radmehr to ensure relevant test data availability.  Provides insights and feedback to Wilson for test design improvements. |
| Samin Sorayya | Bug Tracker / Defect Reporter | Tracking reported defects and issues.  Documenting and reporting defects in a standardized format.  Assisting in the defect management process and keeping track of defect status.  Collaborates with Mostafa to prioritize reported defects and establish defect management processes.  Works with Sasawat and Wilson to gather additional information for reported defects.  Communicates with Lebna to ensure proper documentation of defects. |
| Radmehr Behzadfar | Test data Manager | Identifying and managing test data requirements.  Generating or acquiring relevant test data for testing purposes.  Ensuring the availability and quality of test data for different test scenarios.  Collaborates with Mostafa to understand test data requirements.  Works with Wilson and Sasawat to identify relevant test data for different test scenarios.  Communicates with Sasawat to provide appropriate test data for testing.  Collaborates with Samin and Wilson to analyze test data-related issues. |
| Lebna Noori | Documentation Specialist | Creating and maintaining project documentation, including the test plan, test cases, and test reports.  Ensuring that all testing activities and outcomes are well-documented.  Collaborates with Wilson to ensure accurate documentation of test plans.  Works with Sasawat to gather test execution results and update test reports.  Collaborates with Samin to document and track reported defects.  Communicates with Mostafa to ensure documentation meets project standards. |

This collaborative approach ensures effective communication and coordination among team members throughout the testing process. It promotes shared responsibilities, knowledge exchange, and smooth workflow, leading to successful testing outcomes.

1. **Deliverables**
   * + - * MS1

* Completed team contract.
* Fully initialized Git repository
* Fully setup Jira project
  + - * + MS2
* An analysis of the problem (no written artifacts produced),
* A series of data structures created as header files and stored in the repository,
* A test plan stored in the repository.
  + - * + MS3
* A set of function specifications stored in the repository,
* A set of blackbox tests as test documents with test data for the functions.
* Start writing blackbox test code and store in repository.
* Start implementing functions and store in repository.
* A function-test matrix added to the repository.
* Updated Jira project to show activities and progress.
  + - * + MS4
* Implemented Functions
* Implemented blackbox tests (store in repo), executed (results in Jira) and debugged,
* whitebox tests written and stored in repository.
* whitebox tests implemented (store in repo), executed (results in Jira) and debugged.
* Updated function-test matrix stored to the repository.
  + - * + MS5
* integration tests written and stored in repository,
* integration tests written (store in repo), executed (results in Jira) and debugged.
* acceptance tests written and stored in repository.
* Updated function-integration-requirements-test matrix stored to the repository.
  + - * + MS6
* Execute acceptance tests(results in Jira), and debug.
* Updated function-test matrix stored to the repository.
* Final Testing report listing tests conducted, bugs fixed and the final test passed.

1. **Suspension / Exit Criteria**

* **The condition for temporary suspension is when the program has bugs from the test cases (fail the test) that cannot identify the point, the coding team needs to find those bugs, and the testing team need to create more test cases on that specific point to make sure there will not be the problem for continue developing the program.**
* **The condition for completing the program is when the program is working properly with passed all the coverage test cases and the objectives have been accomplished satisfactorily. The team needs to confirm with the Professor about the coverage of test cases.**

1. **Resumption Criteria**

* **After the testing team informed the bug and the leader analyzed that is the critical problem for continue developing the program. The team needs to make a meeting on this bug topic to find the possible root of this bug. After that, the coding team needs to fix the bug and the testing team needs to create more test cases on this specific bug. Afterward, the team will be meeting again to conclude the coverage test and decide together to keep focusing on this bug or continue developing the program. In the case that the project is running out of timeframe, the teams need to consult with the professor about the status of the project and modify the planning of the project.**

1. **Dependencies**  
   13.1 Personnel Dependencies:

There are many roles involve in the software testing projects. The amount of roles mainly depends on the amount of work which it has to be done. Here there are some of the main roles in software testing which each one of them can be divided to many other roles depend on the condition.

a. Testers: Each Software project needs tester in order to find the bugs or issues involved in the program or identify the exact place of the issue through the codes and point that problem to other personnel in the project and clarify the reason of that bug for them.

b. Developers: Developers are responsible to fix the bugs or completely change the code in order to be ran in the desired way. Developers will be aware of the bug and address of that problem and attempt to fix that part of code.

c. Software Programming managers: Managers are mainly responsible in order to provide the technical needs for other people in the project; they are also in charge of leading the group and assign each one to their specific part. Managers are those how give the roles and they are those who know how to conduct the whole project. They also have to show the way how other people in the group must conduct to each other.

13.2 Software Dependencies:

a. Integrated Development Environment (IDE): Applications like Visual Studio or Visual Studio Code which provide us a condition to code or even somehow debug our code. Testers and mainly developers need these kinds of tools through their procedures.

b. Packaging, Communication and deployment Software: All the programmers and testers need these kind of applications in order to communicate to each other or send the files or folders to each other. Softwares such as Github can help a lot in order to have the same data base part which also reveals the clear idea of the whole program and how far it is completed. There are many other communicating programs which might be needed.

c. Test Automation Tools: These are the Programs which can help us a lot in order to accelerate our speed in the testing procedure and it also can give us a better idea how the wrong code leads to the certain problem. Applications such as Junit or Selenium.

d. Test Management Tools: These are the software that can help us through the whole procedure of testing and debugging and they will give us a clear idea of how we should conduct the test and how we have to fix them all. Applications such as: TestRail or qTest.

13.3 Hardware Dependencies:

There are several Hardware equipment’s that can be involved in the software testing project which depends on the amount and attributes of the test.

1. Basic Hardware: Hard wares such as computers and servers which are naturally needed for each IT project no matter what that is. All these projects needs some network infrastructures, devices and computers.
2. Target Devices: We also need to have the target hardware which we want to run our program on it. Imagine we want to run a program for Auto-Drive Vehicles, so for sure we need that car in order to check and test the program.

13.4 Test Data & Database:

a. Data: For sure if we want to find the bug of some program or we want to develop some program we do need to have the access to the source code or the data of that software.

b. Information: We need to have the scheme of the what we are spouse to do. We need to know which part and how must be fixed and other information related to previous experiences if there is any.

**14. Risks**

14.1. Schedule:

a. Delays : There can be the risk of delay in each part of software testing. It can be in testing part as well as debugging and etc. Delay can be the consequence of many reasons which mostly is related to management part of the job.

b. Wrong time assignment: If the management assign insufficient amount of time for each part, that can cause many problems; For example the program wont be ready at the assigned time and that can cause problems with the clients. Another thing which can happen is that testers and developers may rush in order to release the program and that will increase amount of mistakes.

14.2. Technical :

a. Compatibility issues: Compatibility problems with hardware, operating systems, or other software components can affect the execution and accuracy of tests. Imagine a situation which your program will be run only on certain devices while it can not support others.

b. Lack of tools: Lack of both software and hardware infrastructure can cause technical issues. These problems are mainly irrelevant to our testing and debugging project but it can shut the whole project down.

14.3. Management:

1. Lack of clear Idea: If the management wont be able to give the clear idea to people engaged in the project, then we might face some problem when we want to merge the project together. Because each person is not clarify what is the final target they may do the project in the way which is also right but it may cause a problem in future steps or it can be incompatible.
2. Bad Coordination: Bad coordination can also cause the problems in different scales. In big scale they don’t know how close the project is to end. While in small scale programmers are confused and have no idea what they are expecting next.

14.4. Personnel :

1. Lack of skills : testers or programmers might not be skillful enough to do the project
2. Lack of Personnel: Some times we do need more expertise programmers whit special skills or we simply want more tester or programmers in order to conduct the program.

14.5 Requirements :

* 1. Ambiguous or changing requirements: If the requirements wont be clear enough for the group then it can cause confusion or the team may deliver the program in the way which is not desired; If requirements change at the middle of the project then It also can cause concussion or they might miss some parts which are no more compatible.

1. **Tools**
   1. In this section of the test plan template, you would list the tools or software that will be used during the testing process. These tools can include test management tools, error tracking systems, frameworks for automated testing, code coverage tools, performance testing tools, etc. It identifies the specific tools that will be used to simplify the testing activities and make sure proper communication and reporting.
      1. **Test Management Tool**: To simplify test planning, test case creation, test execution, and test reporting, a test management tool will be used. This program will help with keeping track of test objects, reviewing test progress, and producing accurate test reports.
      2. **Automation Testing Framework**: Repeated and regression test cases will be executed with an automation testing framework. This will help in lowering employment requirements, increasing test coverage, and raising overall test performance. The framework of choice will allow the coding and execution of automated test cases and give strong data reporting capabilities.
      3. **Code Coverage Tool**: this tool will be used to measure the success of the testing procedures in terms of code coverage. The analyzes the code and shows which sections of the code have been run during testing. It will make sure that important parts of the program went through enough testing, raising the software's overall quality.
      4. **Performance Testing Tool**: this tool will be used to review the program's capacity and efficacy. Different load scenarios will be created with this program, which will additionally track reaction times and look for any performance problems. The testing team will be able to confirm that the program can successfully handle the workloads that are expected.
2. **Documentation**
   1. The documentation section of the test plan template refers to the may documents that will be made during the testing process. This can include test cases, test scripts, test data, test logs, test summary reports, and any other relevant documentation. To guarantee that the testing process is thoroughly documented, and that the necessary information is available, it is important to know the documentation requirements.
      1. **Test Cases**: Detailed test cases which explain the processes to be followed, in order to get the expected results, or any necessary requirements will be documented. These test cases will test every function that needs to be tested while considering a few situations, making sure full coverage.
      2. **Test Scripts**: Matching test scripts for automated test cases will be created. The necessary instructions to run the tests using the automation testing framework will be included in these scripts. They will be created to effectively validate the program's expected behaviors.
      3. **Test Data**: will keep updated records of the datasets and test data used for testing. This includes the input data, expected results, and any special setups needed for different test scenarios. Documenting test results is important for consistency and repeatability.
      4. **Test Logs**: creates detailed logs during testing that record the actual results, encountered problems, and system data. These logs help analyze and investigate any defects that arise.
      5. **Test Summary Reports**: generates summary reports that provide an overview of the testing operations. These reports include the number of tests conducted, passed, and failed, key findings, identified issues, and suggestions for further development.
3. **Approvals**
   1. Specifying the people or roles in charge of approving the test plan is done in the approvals part of the test plan template. This is often done to make sure the test plan is reviewed and approved by the right people before the start of testing operations. It's important to identify the people or organizations who must give their consent, such as project managers, or other decision-makers. The approval procedure helps in making sure that everyone is on the same page with the testing strategy and goals established in the test plan.
   2. Project Managers: The project managers will review and approve the test plan to ensure it fits the project's goals and requirements.
   3. Development Team: The development team will provide feedback on the test plan to make sure it accurately reflects the program's functionality and requirements.