Participant 1

Day 1:
Working on two projects.
Refers to test plan, design estimates that include a list of updates to be made.
Also, designs test plan based on the design estimates.

Day 2:
Visited design documents and requirement documents to generate a list of test cases along with clear test plan.
Checks design estimates to understand expected changes based on timeline.
Design testplan based on these expected changes.
Checks for time-intervals, to see if issues raised were fixed, with respect to the response expected. Design testplan to include the time-interval information.

Day 3:
UI development → Checks UI Screens, checks the previous source code to understand/replicate part of the similar screen. Adds in new icons on the screen one by one.
Visits the UI screens for text and places the test on the icons through source code.

Day 4:
Verify the testcases created by the team by comparing with the test plan.
Cross verified using the requirements document.

Day 5:
Lists out the requirements based on previous discussions with the clients. Creates usecases accordingly. Refers to previous requirements document to ensure a common format is being followed.

Day 6:
Visit requirements document to understand what is expected of a new feature being added and design the new testplan accordingly. Adds testcases that test the normal expected flow. Also introduces testcases that correspond to instances where wrong or unexpected inputs are fed - makes a list of unexpected inputs based on the format discussed in the requirements document. Refers to requirements document to understand what output is to be returned in case of unexpected inputs.

Day 7:
Designs testcases for unexpected inputs based on discussion with the clients. This information is not clearly logged in the requirements document. Writes the new testcase to input unexpected values and adds this information in the requirements document, but also marks it as to be discussed with the team members.

Day 8:
Visited design documents to spot if any new changes were made, based on the requirements of a new feature. Tallies with the existing test cases and observes no new testcase requirement based on the design document.

Day 9:
Adds timelines to the new testplan for each test case corresponding to the new feature based on the production timeline agreed upon by the team and the client.

Day 10:
Cross verified each of the test cases across all the testplans created by fellow team members with the requirements document. Approved majority of the testplans. One of the test plan had a minor discrepancy with the requirements document - the participant edited this test plan and informed the concerned team member in person.

Participant 2

Day 1:
Referred to user stories and discussed them in the scrum call to understand how many and how much of the user stories have been tested.
Refers to the test plan to execute functionalities listed in them.

Day 2:
Referred to the test plan and the list of tasks discussed during the scrum meeting. Keeps referring to the testplan to design testcases. Checks previous testplans to see if there are any overlaps.
Overlapping instances with the previous testcases are taken and modified in consultation with the testplan.

Day 3:
Tasks for the day are discussed in the scrum meeting. Identifies similarities with the previous test plan. Create test cases by modifying the ones associated with similar test plan.

Day 4:
Ready to release product is being tested. Found an unexpected response. Visits commit history to see the list of changes and the commits made since the last test instance and tries to identify committer who could have changed a file that caused the issue. Reached out to the committer for quick response.

Day 5:
Tests based on the testcases designed. Logs outputs of each testcase execution. Tallied the output with expected output of the testcase.

Day 6:
Visited the testplan created for new set of features and created test cases for two new features. Tested two of the previously logged features based on the test cases.

Day 7:
Discussed user stories in the scrum meeting to make sure that the testplan is complete. Visits older testplan to compare previous features and the completely new ones added.

Day 8:
Tested a set of testcases and found unexpected behavior for two testcases, one of them displaying a different error message than the one expected and the other redirecting to a different page than expected. Logs these two cases as issues in their issue tracker.

Day 9:
Tested a set of testcases and observed that the application behaves as expected. Logs the status of the testcases in the testplan as passed and alerts the team members.

Day 10:
Retested the previously failed testcases and updated the status as passed in the testplan.

Participant 3

Day 1:
Needs to add a new feature and requires migration to a different technology platform using API. Interacts with the team to collect any previous knowledge in this regard.
Tried implementing and when error occurred, visits ChatGPT for resolution.

Day 2:
Frequent team interactions about new feature. Checks older releases (changelogs and corresponding requirements) of previous projects.
Checks design documentation of previous projects and tries to see if there are any similarities in the previous projects with expected ones.

Day 3:
Starts implementation of a new feature. Lists all modules based on the design documents. List connections to be established between the modules. Starts with a base module and lists the features to be implemented in the base module based on the requirements document.

Day 4:
Issue reported for wrong values being displayed. Directly checks the source code associated and goes to the part that extracts and displays values to the user. Spots the source file from which data is being extracted. Discusses with the team to confirm that the source file is correct.

Checks the file structure to see if there is an issue in the file path. Spots the problem and changes to the right path.

Day 5:
New feature of the project → aware that a similar feature has been implemented previously. Checked the commit history, reading through the commit messages to reach the file with similar functionality. Vague commit messages so could not filter out easily. Goes to source code and views the method names to identify the required functionality.

Day 6:
While implementing the new feature, observes a need ensure that the feature should also work on mobile devices. Explores ways to not rewrite the feature, but reuse it for mobile application. Checks if the libraries being used in the method are compatible with mobile application - searches on google, and learns that they are compatible.

Day 7:
Tries to build a small working POC of the feature and tries to include it in the mobile application. Runs into dependency issues with the library versions. Discusses with team members about the issues being encountered. Tries to downgrade versions of two libraries after which the feature works on the mobile application.

Day 8:
Implements a new feature of the project for the mobile application. He is aware that a similar feature has already been implemented for the web version of the application and hence goes through the filenames to anticipate which file could have this feature in it.

Day 9:
Searched through about 20 files looking at the methods in them the previous day, but could not find any leads. So, started from the file discussing landing page of the application, then navigates to the files linked to each other through the source code and finally arrives at the file containing method that implements the desired feature. Gets the method implementation and replicates the same for mobile application.

Day 10:
The participant has updated about 4 new features to support migration to mobile platform. The participant demonstrated these features on the mobile application to his teammates to get their opinion on whether the features are same as expected and also requested them to test these features (as a part of unit testing), to ensure that he did not miss out on any related cases.

Participant 4

Day 1:
Works on CI/CD pipeline. First thing done when a task is assigned is to look for already implemented similar functionalities in the project. Goes through previous commits and source code to understand the naming conventions and the deployment process or the steps involved.

Day 2:
Reaches out to the team for discussions. Contacts concerned team member in-person and discusses the folder structure of where certain files and data are stored and the corresponding naming conventions.

Day 3:
In an instance where the concerned team member is not found in-person, looks through the discussion history on communication platforms of the team, pings the team member on communication platforms.
Also, looks at changelog to understand which files were updated by the concerned team member, who has previously implemented a similar functionality.
Then, checks the logs associated with a specific set of files to understand how and what is modified. Tries to understand existing implementations and get an idea of the changes to be made.

Day 4:
Going through the discussion forums in the organization for information related to modifications to be made in the code and the location of the modifications.

Day 5:
Need to change the processing library being used and the model, hoping for better results. Starts with the model. A trained model is received from the team. Need to use this model and realises it needs preprocessing based on the input, test and train data as discussed by the team. Looks into the source code that deals with pre processing and navigates to the code through discussion with the team.
Implements the library.

Day 6:
Implementation of a new pre-processing library is complete. Checks for output expected on large real-time data. Finds that the processing takes a lot of time and does not converge. No results are obtained. Checks resource utility logs and observes that the code uses up 6GB of the GPU. Discusses with the team about this and mentions that it was about 3GB previously.

Day 7:
Identifies issue with model size, causing high resource consumption. Reports the same to the team. Asked to check for a lighter model already available with the team. Implements the lighter

model and it works well. Checks the resource utilization log which now uses 3GB as in previous case. Reports the same to the team.

Day 8:
Given a new, updated model to optimise and deploy. First time working on optimization of a machine learning model. Spent most of the time in learning various ways to automate - google. Tried out optimising models available online to arrive at a basic understanding of optimisation.

Day 9:
Follows steps learnt, to optimise the model shared by the team. Runs into errors that say the model format is not supported for optimisation using the optimisation libraries, due to libraries that were used to build the model. Checks for other libraries that can optimize the models.

Day 10:
Searches for library compatibility - libraries that can optimize models vs libraries that were used to build the model. Comes across discussion forums on the web and observes that model developed using a certain library are not compatible with optimisation libraries. Observes through the discussion forums that the workaround to make the model compatible with optimisation libraries is to convert it to a different format by rewriting parts of the model and loading it with previous model's weights. Conveys the same to the team members.

Participant 5
(Game Development)
Day 1:
Checks the history of the project to get to related information and then looks through commits and navigates to the respective source code. - to see if the participant could find similar or opposite implementations of the functionality to be implemented.
Checks for list of files that would cause a change for feature upgrade or those that would be affected by changing the function - how many files would be affected and which ones?

Day 2:
Looks at list of files affected by changing one file. Checks for opposite implementations in previous change logs.
Reads through the last four changelogs and spots one changelog that mentions the implementation of exact opposite feature[On press, move left]. Replicates it to opposite functionality [On press, move right].

Day 3:
Refer to design documents to understand all the components to be added to the game. For one component of increasing player score on killing an enemy, which is two times more powerful than those in previous levels. Visits the previous code to access the database path based on the commits that mentions the integration of player actions to be reflected into the database. Extracts the path and stores this path in the new method in source code.

Day 4:
Based on the path of the database stored, writes code to update the player score to the path. Then writes code to render the series of changes that should occur based on the double increment. This series of changes are listed in the design documents, which is referred during this implementation.

Day 5:
Need to implement a new feature that performs the opposite action of an existing one. Player interacting with an object should go down in strength by 2 levels. Checks project history by exploring the release documents (changelog) and navigates to commits, source code and spots the opposite functionality, where interaction with an object increases the player's strength by 2 levels. Takes this function and modifies the value to be changed to -2 and the object ID to the desired object.

Day 6:
Need to implement a new feature, not previously present in any level or screen of the game. Explores the web to see how this feature is implemented. Starts implementing the feature based on the understanding of the search results. Tries out for one element in the game. Performs partly as expected.

Day 7:
Working on implementing the new feature. Modifies attributes of the action for the element to check if it performs as expected. Tries out various combinations of the attributes and checks the performance.

Day 8:
Implemented the new feature for one element in the game. Checks for all elements that need to exhibit this new feature by referring to the design documents.

Day 9:
Checks the performance of the new feature and tests the whole game till the current point to ensure consistency. Checks for possible conflicts, did not find any, pushes the code to the project repository.

Day 10:
Worked on updating a player action in a new level with respect to the number of points earned on completion of a new level-specific target. Checks the files that contained path to the database and extracts the path and stores the path in the new method to reflect player actions onto the database.

Participant 6

Day 1:
Checks naming conventions and locations to store new files. Refers to the flow/component diagram.
Looks for changes through the commit messages - but did not find necessary information - commit messages were not clear and not descriptive. So, goes through code modified in the commits and checks for the changes.
Checks resource utilization of implementing a specific feature.

Day 2:
Checks resource utilisation without any feature addition. Makes changes in animations of the game elements. Implements the desired animation.
Only one animation that is common for all elements on the screen.

Day 3:
Checks resource utilisation for the animation added. Observes very high usage of the resources.
So, reduces animations for all the elements and adds to only 7 elements out of the 32 elements on the screen, by checking each of the elements one by one and deciding whether the animation is essential for the element.
For instance, trees waving animation and animals moving animation is preserved. But, rocks waving animation is removed.

Day 4:
Plays the scene sequence after reducing the animations to see if the scene is intact. Checks resource utilization after reducing the animations which has significantly reduced from the previous. Contacts team member to confirm if the resource utilisation is fine.

Day 5:
Refers to tasks assigned by the team lead to add animations to all the elements of the game for a lively experience. Adds interesting movements which are very clear. Checks Resource Utilisation Logs and finds that the RAM usage to render this scene was about 1.6GB. Retraces back to before these changes were made and checks RAM usage. The usage was less that 1GB (between 0.8GB to 0.9GB). Tries to figure out ways to reduce the RAM usage, while preserving the liveliness of the scene.

Day 6:
Due to high resource utilization, checks for options to reduce the resource utilization. Reduced the graphics in terms of number of actions and movements and their frequencies, to all the in-game elements, except the protagonist of the game.

Day 7:
Now, the game scene looks lively where the elements move, but they move very slow and the movements are very light. It looks close to real-time scenario, but not exactly the same. Checks the resource utilization logs, which is now updated to 0.97GB. Committed the change and pushed it to the project repository.

Day 8:
Refer to design diagrams to understand different modules to be communicated.
Check file structure to extract path and the names of the modules and add these paths.

Day 9:
Started working on creating a game based on the requirements of the firm's investor. He checked the naming conventions and the architecture of previously developed games to understand the naming and location to store new files. He referred to the flow diagram and the component diagram to understand how to divide and organise the functionalities and for implementing the functionalities.

Day 10:
Created a basic template for the game that included placing of elements required, based on the requirements discussed and created a skeleton of the files that would be needed, with respective file names, stored in appropriate folders.

Participant 7

Day 1:
Assigned the task of deploying the developed tools and games on devices with different environments, while keeping a check on the resource utilisation. He explores minikube dashboard associated with the projects to understand deployment process and to track if there are any applications running already. Monitors minikube dashboard.

Day 2:
Troubleshooting based on minikube dashboard. Application stops unexpectedly - checks for list of applications running on CPU selected and the amount of space being used.

Day 3:
On minikube dashboard, monitors the CPU usage during the scenario described, where the application stops. Observes that a set of actions in the game use up too much memory on RAM, causing it to stop unexpectedly. Reports the same to other team members and requests to initiate a discussion on the same.

Day 4:
Team discussions revealed that the application stopped unexpectedly could be due to multiple repeated interactions on the backend. Participant visited source code and traced the functions

that are repeatedly executed. Identified the specific function being called in a loop and re-writes the part of the code to ensure that there are no infinite loops.

Day 5:
Another issue reported by the testing team has been assigned. Discussion with the team revealed that a similar issue has been faced in a different game context. Checks for issue history. Reads through the issue titles to see if a similar issue has been mentioned. Spots the similar issue and checks the issue body. Confirms similarity and checks the solution (commit) that solved the issue and follows similar steps to resolve the issue by modifying parts of the solution to fit the new usecase. The issue gets resolved.

Day 6:
A new feature is added to the game. Starts playing the game to test the resource utilization, while monitoring the minikube dashboard. Keeps track of the resource utilization logs for every action performed in the game.

Day 7:
Creates a document containing the action performed and the amount of resources utilised for that action throughout the game and shares the same with the team.

Day 8:
The game freezes unexpectedly and this issue is reported to the participant. The participant checks the minikube dashboard to check if there are any resources being overused, leading to the game freezing. Reports the CPU usage statistics to the team and suggests the need to optimise the functionality in the game.

Day 9:
Rewrites part of the functionality in the game causing high resource consumption, by reducing the number of interactions with the database and only creates interactions at the beginning and end of the functionality, while updating the specific variable in the code itself.

Day 10:
An issue reported by testing team has been assigned. Explores previous issues, spots a similar issue and resolves the issue by following the steps reported in previously occurred similar issue.

Participant 8

Day 1:
Clones existing similar functionalities based on previous implementations - identifies similar previous implementations by comparing the outputs with the expected outputs to understand similarities and differences.

When errors occur during programming, compares the expected vs obtained and navigates from one file to another.

Day 2:
Understand the functionality to be implemented. Looks through commit history to identify similar functionality mentions in the commit messages. Could not find any such mention in the messages. Goes through changelog, but that is also very crisp. Could not understand changelog or commit history. Discussed with team member about how to address this issue.

Day 3:
Goes to file structure to understand where to insert the new file. Starts implementing the functionality. Goes back to file structure and checks for naming conventions and names the file based on the naming conventions. Clones existing functions with similar and nearest functionalities.

Day 4:
Need to modify access permissions as a part of the new feature. Goes to changelog across 5 releases until a mention of the permissions being added has occurred.

Day 5:
Visits the commit history of the change log that has been spotted and reads through the commit messages. Spots the commit that discusses permissions in the message and goes to the source code affected by the commit. Extracts the function that grants permission and replicates that for new set of users.

Day 6:
New feature to implement → Checks naming convention of the target csv file. Need to implement a function that processes data and stores into a target csv. Checks for existing source code files to identify where this function is written. Navigates through the file structure one by one to find the source of this utility function. Identifies path to arrive at the utility function and imports the function from this path.

Day 7:
Needs to implement a new feature. Identifies the elements on the application that deal with this new feature by comparing the current functionality of the application and the elements used in the source code. Starts writing the code to add the new feature.

Day 8:
Writes part of the code, that implements part of the new feature and tests it to see if it performs as expected. Finds the performance to be satisfactory, and then goes ahead to complete the new feature.

Day 9:
Faces errors in the process of compiling the new feature, due to scope of certain variables in used in the new method. Goes to the existing source code to see how and where certain variables are declared. Compares the functionality of those variables to that of the variables in the new feature and adjusts the scope accordingly.

Day 10:
Tests the working of new feature added, by running the application and entering relevant details that trigger the new feature. The application works as expected. Logs the updates, checks for conflicts with the existing code and finally pushes the updated code to the project repository.

Participant 9

Day 1:
Started a new project and is exploring setting up the system for the project. Checks the required environment and associated dependencies. Checks issues that occurred with similar dependencies, with respect to versions and other required sources. Ensured that the requirements to install and the dependencies are intact.

Day 2:
An error occurred during the setup – identified conflicting versions based on the error log and the installation log and then searched for resolving these on Google, stack overflow.

Day 3:
Goes through design documents to identify different modules required for new project. Identified the environment requirements for each of the modules. List out libraries that support these modules from previous code → changelog → check for instances where it is implemented.

Day 4:
Go to commits associated with selected changelog and identify commit that implemented this instance → list out the libraries imported for the same.

Day 5:
Import the libraries as listed yesterday. Starts implementing new feature. For some libraries that gave warnings about versions and dependencies during installation, checked the list of issues reported, related to these libraries to understand whether these warnings could really result in problems. Spotted one issue that mentioned conflicting dependencies - versions of another library, for two libraries. Lists out such issues.

Day 6:
Based on the issues identified in the issue history with respect to library dependencies, checks for alternatives for libraries. Goes through the design documents to see number of modules that could use these libraries. Checks pros and cons of each library and takes a call accordingly.

Checks common dependencies across more important libraries and installs those dependency environments. Checks alternatives for other libraries compatible with these environments.

Day 7:
Started implementing part of a module identified previously. Created dummy database, with structure that would be analogous to the database to mimic database access. Implemented the reads and writes to the database part of the module, along with the processing that happens on the data read from the database.

Day 8:
Faced issue with a method of a library, where the method has been deprecated in the latest version of the library imported, but has been previously used for similar purposes in previous projects of the firm. Explores the documentation of the library to see if there is an alternative method discussed in the latest version of the library. Also searches through the web to find discussions on this issue.

Day 9:
Comes across a discussion forum that discusses a work around to replicate the functionality of the deprecated method, using the latest version of the library imported. Follows the solution discussed and implements the same. After multiple errors and resolutions, finally arrives at a solution that can perform the action of the deprecated method, using two new methods in the latest version of the library.

Day 10:
Writes code to complete one module identified. Replaces the dummy database with the copy of the actual database and tests the working of the new module, by passing varied arguments to the module and assess the values written back to the database and the messages returned on the console.

Participant 10

Day 1:
Works on UI development. Looks at the UI screens shared to her by the team management, and tries to understand the elements to be included and their types.

Day 2:
Refers to design documents to understand the destination to which the action of elements should correspond to.
UI Screen → Design Documents → list target → UI Screen.

Day 3:
Links UI screen elements with target files. Based on the lists made, for every icon on the UI, the target file to be taken is linked with the actions like “on-click”, “on mouse-over” and so on.

Refers to design document to validate this list of actions.

Day 4:
Consults design documents for new UI features and got the target component. Passes the data from UI to the target component for a new feature that requires a specific data flow on an event from the feature.

Day 5:
Issue assigned on UI. Aware that the part of the product was working fine previously . So checked the commit history of the files that could have changed the product. Reads through the commit messages and spots the file that should be modified to resolve the issue.

Day 6:
Given a task to add a new element on the UI that takes in data from the database and displays it on the UI when the user clicks on a button. The element to be added is a table, that also facilitates filtering based on the users' choice. First time working on table implementation in the UI. So, explores the web to check ways to include tables in the UI and to make them interactive.

Day 7:
Reads through the previous commit messages where she had integrated UI with the database to extract the credentials and path to the database. Spots the file that checks for login details of the user and obtains the details required.

Day 8:
Starts implementing the table element - without filter functionality, only to display data from the database.

Day 9:
Finished implementation of the table along with the filtering option. Checks the functionality thoroughly for all the possible user choices - exploring all permutations and combinations. The functionality works as expected, but the look of the table is different from the required one. So, refers to the UI Screens provided, identifies the color codes for the heading rows and the table and further decides the dimensions of the table based on the UI Screens.

Day 10:
Updates the table display with appropriate icons, dimensions and color codes as per the UI Screens, checks for conflicts with the existing code and pushes the UI code to the project repository.

Participant 11

Day 1:
Tasked to work with APIs. To arrive at a new feature, explores similar features in other applications, chalked out how to implement and then created a database structure based on this understanding.
Frequently referred to the expected outcomes of the tasks and compares with obtained. Fills the data onto the platform and looks at the response returned. Add new fields one by one.

Day 2:
Tasks are listed from the team lead.
Explores database structure and then identifies if there is a need to modify the structure. Discussed in-person with their teammates to get a second opinion on the database structure. Added two new fields since additional information is needed. Also, discussing on adding a new table for easy access of information.

Day 3:
Referred to part of requirements document that lists the input and output. This is listed by the team lead during their scrum meetings.
Checked the required collection name in the database and that collection is called into API implementation.
Called the API partly to see if it returns desired value.

Day 4:
Updates the database structure by referring to the requirements document.

Day 5:
Add a new function to the API. Check the target file location based on the code associated. Checked the input and output parameters of the existing functions in the API. Checked output parameters of the functions dealing with desired target location.

Day 6:
Checks the fields in source and target files, their names, their key types in the database. Identifies the primary and foreign keys for each of these. Extracted the data to the APIs based on these keys. Checks for the extracted data and modifies the code. Changes the query to obtain the desired value.

Day 7:
Tasks listed by the team lead - need to create a new API that takes input from the application and returns a result based on the input. Visits the source code of the existing APIs and observes that the new API functionality can be added as a new path in the existing source code itself. Updates the main file accordingly and other utility files and source files in the associated folders to take the input and pass it along.

Day 8:
Used FastAPI to build the API. Runs the API and navigates to the new path to test the functionality (using SwaggerUI). Passes inputs as suggested in the requirements document and checks the output. Repeats this for 7 to 8 examples and then shares the curl call with the team.

Day 9:
The team member using the API in the application gets back to the participant and asks to update the API to take in inputs as request body, rather than as parameters of the url. Explores ways to take input as request body on the web. Observed a few examples and implemented the code to take the input in request body.

Day 10:
Checks the updated API that takes in input as request body by testing on the Swagger UI. It performs as expected. Hence, updates all other paths as well to fetch input as request body.

Participant 12

Day 1:
Needs to query the database for a certain requirement. Views the database structure and checks for the list of columns required, in the database collection.
Identifies the names of the columns required and the respective variable types. Found the common column names across two tables in the database and identifies the primary and foreign keys in the table.
Often refers to chatgpt for code - makes changes based on the project organization.

Day 2:
Explores existing files to get the details of the database credentials, from the files that have previously dealt with database querying task.
Checks the env file to get paths required. Visits source code to check the path to the data directory - this data directory is the source to extract database details.
Discussed with team member for clarification when errors occur.

Day 3:
Checks the env file - identifies the list of all paths required to identify the location where data is stored. Goes to commit messages to see if there is a mention of accessing desired information. Filters out a few commit messages based on the content written. However, not all the commit messages were clear, so goes through the “diff” and the source code associated, for better clarity and spots the source code files that access the desired information. Replicates the same methodology.

Day 4:
A new feature to be added is manually communicated by the lead. No requirements document. Interacts with the lead to get clarity on the expected outcome and identifies the required files

accordingly. Starts implementing required features and generates a sample output to discuss with the lead.

Day 5:
Tests a part of the product and spots an unexpected behavior based on the test plan. Checks the requirements document to confirm that the obtained output is not expected. Need to log this as an issue for his teammates to resolve. Recently took up the responsibility of testing the product. Not aware of the issue logging practices and hence goes through the project issue history to understand how issues are logged. Spots a similar issue logged and replicates this issue.

Day 6:
Tests the product based on the test plan - test plan had 20 testcases. Some of the testcases were unclear - lacking clear details of what information to be entered on to the application, but they rather mentioned “fill in the details of the user”. Communicates with the team member to gain further clarity on the details and the location where these details can be found.

Day 7:
Updates some of the testcases in the test plan to add in more clarity on the inputs to be fed and the expected outputs for respective inputs. Also, discusses with the team member about certain inputs that should ideally be not accepted (dummy random inputs - for instance, all 9s in the phone number). Brings such cases to the team’s notice.

Day 8:
The testcases in the testplan and the application are updated based on the participant’s suggestions and the participant is asked to test the application with the latest updates. The participant starts testing against each of the testcase. Tested 17 testcases and did not find any issue.

Day 9:
Tasked to update a previously implemented new feature (not logged in the requirements) by adding new checks on the input and to return error message on an unacceptable input. Updates the source code accordingly by adding if-else loops in the source code.

Day 10:
Needs to adopt a streamlined pattern to return the error message - the text and the template. Visits the commit history to check if there is a mention of “returns an error message” or similar wordings. Finds a commit message that mentions “throw error” and explores the associated source code to obtain the format of the error message and implements the same.

Participant 13

Day 1:
Works on adding security to the application. Added authentication and authorisation mechanism from scratch.
Adds a new code and also modifies the existing code. Faces inconsistencies in the commits and observes conflicts with the existing code.
Walkthrough of the existing code to understand and identify locations of these conflicts.

Day 2:
Checks requirements document to check the list of users with respective access permissions. Creates a list accordingly. Explores ways to implement the authorization and authentication by searching on the web.

Day 3:
An issue occurred in the application ready for production, which was directly reported to the participant. Visits the source code based on prior knowledge, and sees that a part of the code has been changed and is causing the issue. Tries to modify the part of the code identified.

Day 4:
Issue still not fixed, so visits all the files that use this snippet of the code and sees that the expected parameters have been changed all across.
Tracks down the set of commits that resulted in the changes and reverts back to one commit. (Stalls production before making these changes)

Day 5:
Works on the issue occurred. Reverting to the previous commit resulted in the loss of latest changes. Checks for commit messages of the next changes made and re-implements each of these changes.

Day 6:
Decides to add support to OTP through email and sms to authenticate the user for a specific functionality. Explores multiple applications that can support this feature on the web and checks the pricing for each of these applications.

Day 7:
Starts implementing OTP through email functionality. Decides to use a specific library supported by springboot. Explores the web to get an idea of the source code that is used to send the otp. Implements the code and checks with a test email to assess the OTP authentication feature. The feature works as expected.

Day 8:
Updates the source code to consider the otp entered by the user before allowing the user to access the specific functionality. Checks for inconsistencies in the code, and for any conflicts. Pushes the code since no conflicts were identified.

Day 9:
Decides to use Twilio to send otp on Whatsapp of the user to authenticate the user. Explores ways to integrate twilio on the web and tries out basic implementation for a test mobile number. Implements the code and the feature works as expected.

Day 10:
Updates the source code with otp to whatsapp of the user and pushes it since no conflicts were observed. Retests the otp authentication through email and whatsapp for a set of email ids and mobile numbers.

Participant 14

Day 1:
Checks UI design Screens. Implements the screens and then tallies with previous UI.
Gets list of files related to the UI and the files that communicate with each other.
Links the files accordingly in javascript and tally the variables being passed, while comparing with the expected results.

Day 2:
Issue on the UI screen → causing continuous refresh of the host page, while it should only refresh the chat application. Checks commit history to identify last set of commits that caused the change.
Goes to the source code of the respective commit that implements response from chat application and reverts this commit.
[Long standing developer of the organisation → so did not have to explore much and directly visited the commit that changed a specific source code file]

Day 3:
Adding a new UI block. Details to be added were discussed during a scrum meeting and the list of these details was created. No proper requirements document was created. Based on the list, each of these items were added into the UI. Based on the previous knowledge, the target files were linked. Had to visit the source code to copy the path of the target and source files.

Day 4:
Bug reported by the stakeholder claiming that the values expected are not being returned. When a part of the paragraph has to be highlighted, the whole para is being highlighted by the application and also the paragraph being highlighted is incorrect.
Visits the changelog to identify the list of changes made that could have caused the bug. Goes to commits that discuss updation of or integration with a new library.

Day 5:
Visits the source code that integrates the new library. Tests the input and output being returned by the new library with a sample input. Observes that the library does not give output in the expected format. Modifies this format using pre-processing and post-processing scripts which solves that bug reported and then commits the code.

Day 6:
Gets a requirement to add three dropdowns on the screen. Adds the first dropdown field in the UI. Needs to extract the data to be filled in dropdown from the data available on the application server - this is stored in the database. So, extracted the list of items from the database.

Day 7:
The two dropdowns are dependent on the users' choice in the first dropdown. So, the participant has written code to extract subsequent elements, based on the users' choice in the first dropdown. The dropdowns are based on the category of the items in the first dropdown (which is not included as a part of the dropdown). The participant extracted the category field from the database and stored it as another attribute for the item, without displaying it in the UI.

Day 8:
All the three dropdowns are added and the relation between them is written in the source code based on comparison with the item categories (in the form of if-else loops). The data corresponding to the dropdown choices is then obtained from the database and displayed to the user.

Day 9:
The participant checks for compatibility of the new dropdowns with the UI design and color theme and updates the dropdowns accordingly, while also updating their position on the UI. Then checks for conflicts, pushes and merges the code with the UI.

Day 10:
Participant worked with the latest updated dropdowns - tested various input selections in the first dropdown and assessed if the next dropdown items are updated accordingly, followed by the appropriate display of the respective results from the database.