|  |  |
| --- | --- |
| Use case | Manage Models |
| Test objective | To successfully test the functionality of the creation/update/deletion of model which further includes creation of entities, creation of properties for the corresponding entities and linkage of entities. |
| Verification method | Testing, Inspection, Analysis |
| Specifics of the methods | 1. **Analysis:** Perform Static Analysis to find syntactic and basic semantic defects such as null pointer exception, unclosed connection object checking, etc., in the code.    1. Style checking    2. Pattern Matching    3. Control Flow Analysis    4. Data Flow Analysis    5. Formal Verification 2. **Inspection:** Code review with a checklist can be done. The checklist may contain the basic checks like for checking for code modularity, etc. 3. **Testing:** In testing we suggest both the Black Box and White Box Testing to test the functionality of Manage Model function.    1. **Black Box**:       * For instance, the user creates the model, saves the same and reloads to check if the model is created.       * Use a classification tree to identify the equivalence classes from the test aspects. Some examples of the test aspects could be “number of associated properties”, “duplicate model name” etc. You can then come up with the test cases from these equivalence classes.    2. **White Box**:       * To ensure that all the valid basis paths are traversed so that the execution paths have been tested at least once.       * To test for 100% branch and statement coverage.       * White box testing needs to perform when the model is committed to the data base and the program checks if the model is valid before storing it in the database. |
| Testing environment | Development environment |
| Tools required | * Testona * ACTS * JUnit * SonarLint * Code review facilities of GitHub * CheckStyle * Control Flow Graph * Factory * FindBugs |
| Parameters of interest | * Basis Path Coverage * Combinatorial Coverage * Inspection checklist |
| UI Modules | * EntityManager * LinkManager * PropertiesManager * RolesManager * UserManager * RightClickPane * NotificationPane |
| Service Modules | * ProjectListService * SaveProjectService * RetrieveStdEntityService * FetchModelService * SaveModelService |
| Database Modules | * Entity Collection * Project Collection * Role Collection |
| Unit Testing | |
| Entry Criteria | * Use cases and scenarios associated with the given code artifact have been created and approved by the client. * The specific code modules have been completed * The code module has passed the static analysis |
| Exit Criteria | * All the generated test cases have been executed in both 3-way testing and white-box basis path testing. |
| Testing Approach | * Generate classification tree to model the input domain using Testona * Using ACTS, enter the parameters from the classification tree and generate the test specification for a 2-way combination. * Filter out the infeasible combinations through constraints. * Generate the test cases for the given specification * Then, for white box testing, generate the control flow graph from the code artifact using Control Flow Graph Factory tool * Using the CFG, calculate the number of basis paths and generate the test cases to traverse all the basis paths. * Execute the test cases and record the test summary results. * Correct the defects and re-run the test suite until adequacy measure |
| Measure of adequacy | * Faults per test case is not significantly large. (<0.0005) * 100% test coverage |
| Responsibilities | * Developer |
| Integration Testing | |
| Entry Criteria | * The code modules have passed unit testing * Interface documentation which shows how two components which are to be integrated communicate with each other. |
| Exit Criteria | * All the generated test cases have been executed. |
| Testing Approach | * Here the data flow between the components being integrated are to be tested. * Generate the test cases to test the communication between two components. * Execute those test cases and verify if the communication between the components is what is expected. When integrating a front-end component with a service component, verify if the data which is passed from the front end to the service is in the expected format. Also, the communication between the service and the database, ie., what is finally being stored or retrieved from the database is to be verified. * Techniques like Control Flow coverage, N-1 Switch state-transition coverage and mutation testing can be used. |
| Measure of adequacy | * 100% Node coverage (every pair is at least exercised once) * 100% Edge Coverage (Every edge is at least exercised once) |
| Responsibilities | Developers, testers |
| System Testing | |
| Entry Criteria | * The integrated system which needs to be tested should be completed. * Integration testing has passed. |
| Exit Criteria | * The functionality as described by the acceptance criteria in the form of use cases have been executed. |
| Testing Approach | This will be a black box test to test the functionality of the system. Walkthrough the use cases present and execute them in the system. |
| Adequacy Measure | * Conforms with acceptance criteria in SOW |
| Responsibilities | Testers |
| Soak Testing | |
| Entry Criteria | * The integrated system which needs to be tested must be completed. * Integration testing has been passed. |
| Exit Criteria | Test report generation and analysis are complete. |
| Testing approach | * Determine the workflow of a normal user of the system. * Using this estimate the number and the type of requests made per user session. * Estimate the number of users on normal usage hours per day * Feed this information to JMeter and execute the tests constituting this workflow for an extended period of time say 8 hours * Generate report * Analyze the report to ensure |