Assignment 4

In this assignment we performed the **Model-based Software Testing** technique. For this purpose, we used the QF-Test tool, this tool enables us to capture a sequence of steps in our program, and later analyze its states and create tests based on these states transitions.

Use Cases

1. `Create Task and Mark It Done`

The user should be able to create and edit the fields of a task, possibly marking it done, and then save it in file to read it later. This is probably the most important use case for the application that deals with a core functionality of the program.

In this use case, the user should first create a task, filling the required fields, such as the description and date, and then mark it as done.

2. `Visualize tasks with specific project or context`

The user should be able to tag a task, defining a specific project or context to it. Then, the user should be able to filter by the tag and visualize only the relevant tasks.

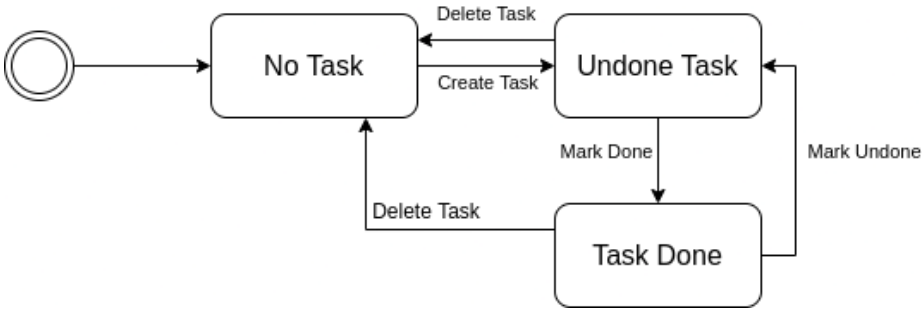
3. `Sort tasks by priority`

The user should be able to sort the tasks by priority, in ascending or descending order. This use case could also be implemented using other parameters besides the priority. The priorities are defined by letters from A, highest priority, to Z, lowest priority.

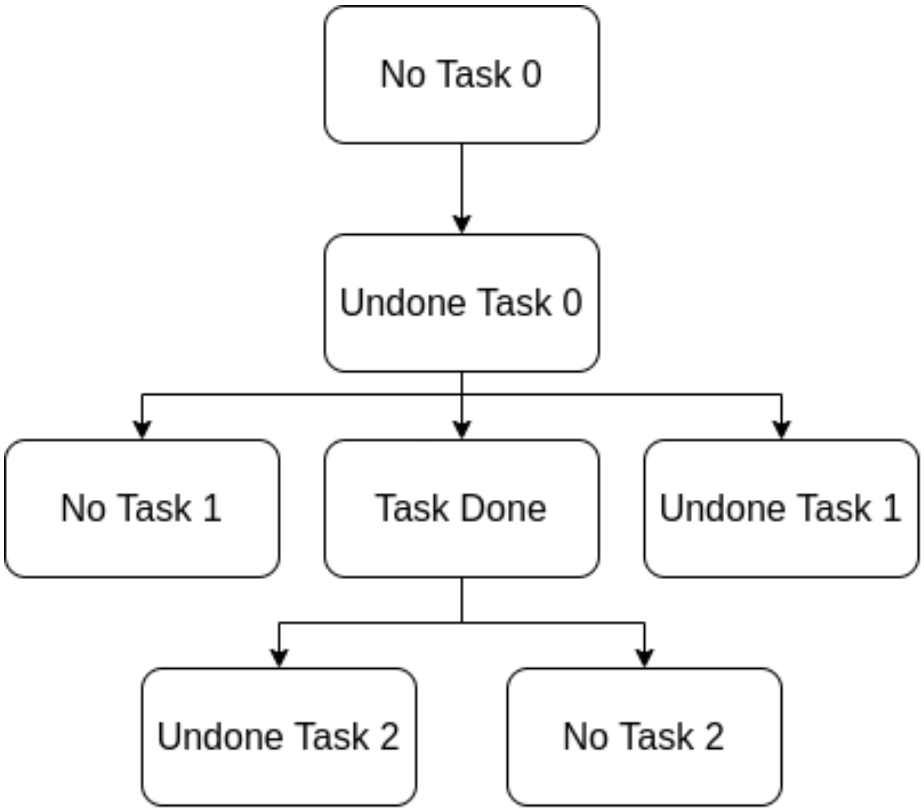
State Machines

Use case 1

State Machine It is composed of 3 states, in the initial state, the task is not yet created *No Task*, then it is possible to create a new task *Undone Task*, or mark it done *Task Done*:



Transition Tree The transition tree creates 4 main paths, from the root node to its leaves.



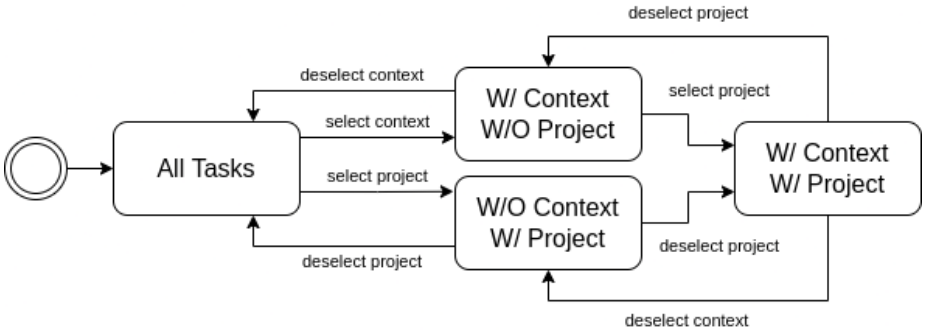
Transition Table

States Events	Create Task	Delete Task	Mark Done	Mark Undone
No Task	Undone Task	———	———	———-
Undone Task	———	No Task	Done Task	———-

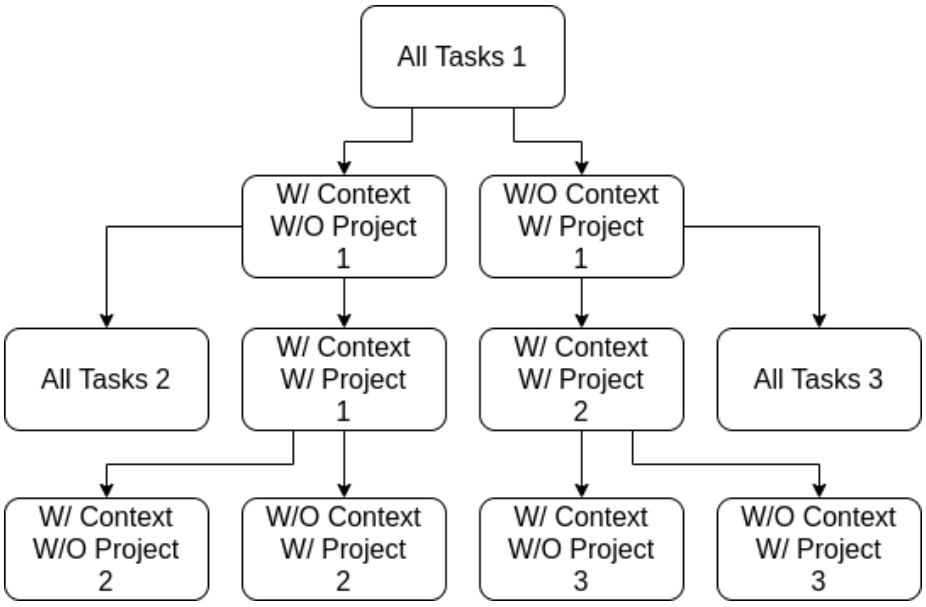
States Events	Create Task	Delete Task	Mark Done	Mark Undone
Done Task	———	No Task	———	Undone Task

Use case 2

State Machine In the initial state *All Tasks*, all tasks creates can be visualized, after the events of selecting a context or a project, only the tasks within this context/project are shown. If a project and context are selected, then only tasks having these two are shown:



Transition Tree The transition tree, from the first state, can follow 6 distinct paths:

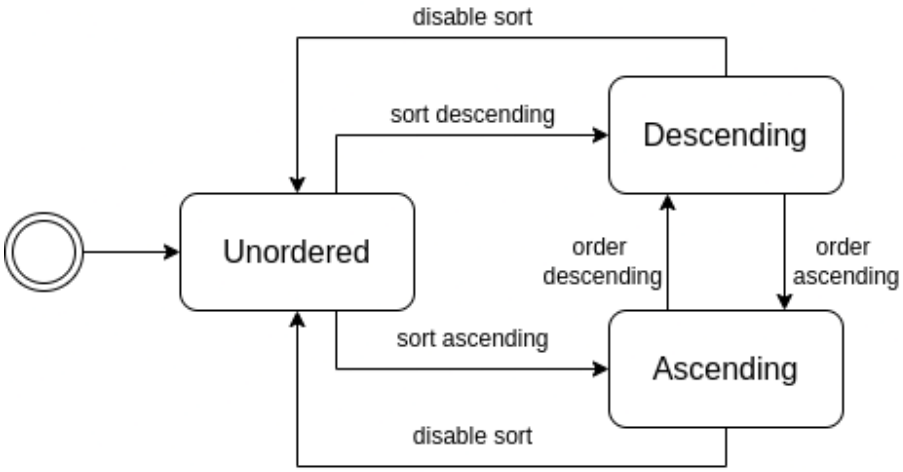


Transition Table

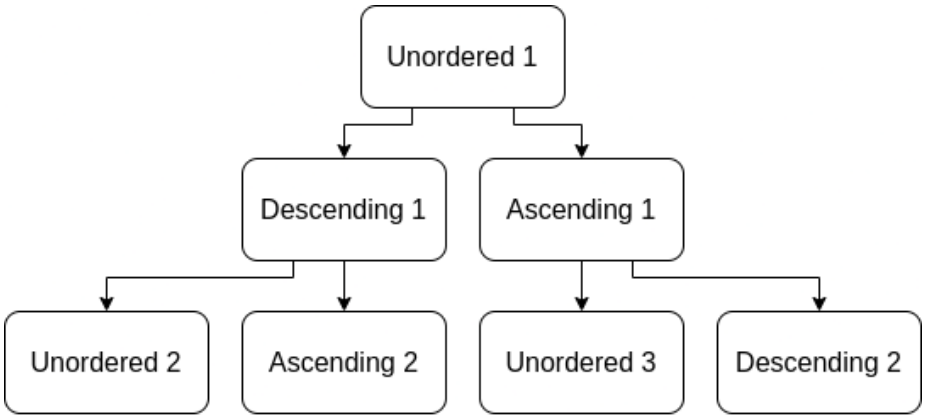
States Events	Select Context	Deselect Context	Select Project	Deselect Project
All Tasks	W/ Ctx W/O Proj	———	W/O Ctx W/ Proj	———-
W/ Context W/O Project	———	All Tasks	W/ Ctx W/ Proj	———-
W/O Context W/ Project	W/ Ctx W/ Proj	——-	———	All Tasks
W/ Context W/ Project	———	W/O Ctx W/ Proj	———	W/ Ctx W/O Proj

Use case 3

State Machine From an unordered set of tasks, the user can sort it both in descending or ascending order of priority:



Transition Tree From any state, the user is able to go to one of the other two:



Transition Table

States Events	Sort descending	Sort Ascending	Disable sort
Unordered	Descending	Ascending	———
Descending	———-	Ascending	Unordered
Ascending	Descending	———	Unordered

Tests

The QF-Test file is stored in the root in the path `/qftest/qftest_a4.qft`

Use case 1

Test: `Create a Task and Mark It Done` In this test, the user creates a task with the description "test application", and, after clicking in the *add* button, marks the checkbox to set it Done. The test asserts for the description field and the checkbox. The test passed, and was able to successfully follow this use case flow.

Test: `Delete Task` The test first creates a new task and then deletes it, then check for the nonexistence of the component in the graphical interface.

Test: `Create a Done Task` (Sneak Path) This task creates an already Done task. This test failed, as it was possible to create an already Done task. But it could also be argued that this could be an expected feature, however, our state machine did not predict this path.

Test: `Deletes when no task exists` (Sneak Path) This task should not create any side effect in the program. It Passed, as still no task existed and no exception was thrown.

Use case 2

Test: `Visualize tasks of a context` When selecting a specific context, then only tasks within this context should be shown. Passed, only the right tasks could be visualized.

Test: `Visualize tasks of a project` When selecting a specific project, then only tasks within this project should be shown. This test passed, and only tasks of the select project were presented.

Test: `Visualize tasks of a project with a specific context` When selecting a specific context and project, then only tasks within this context and project should be shown. Passed, and only a specific task with a context and project was presented.

Use case 3

Test: `Sort Descending` From unordered tasks, order them by descending priority. Passed, the tasks were correctly ordered.

Test: `Sort Ascending` From unordered tasks, order them by ascending priority. Passed, the tasks were ordered in ascending priority.

Test: `Sort Descending from Ascending` From descending tasks, order them by ascending priority. Passed, the tasks had their order inverted.

Test: `Sort Ascending from Descending` From ascending tasks, order them by descending priority. Passed, the tasks had their order correctly inverted.

Test: `Sort descending while already sorted this way` (Sneak Path) This test should not create side effects and should maintain the order of the tasks. Passed, the tasks remained ordered.

Use of the QF-Test tool & Conclusions

The QF-Test tool appeared to be promising when automating tests of user interfaces. One problem that is not directly related to the tests creation, but we ran into, was that with high DPI monitors, some workarounds were needed to run the application decently.