Entrega 4 - Model-Based Software Testing

Model-Based Software Testing

É uma técnica de teste de software em que o comportamento em runtime é avaliado e comparado a previsões feitas por um modelo. Um modelo é a descrição do comportamento do sistema, e pode ser feito através de sequências de eventos, ações, condições, output ou transferência de dados.
Esta técnica descreve o comportamento de um sistema em resposta a uma ação determinada pelo respetivo modelo.
Model-Based Testing difere das outras técnicas estudadas ao longo do semestre, até agora, na medida em que faz mais parte do processo de desenvolvimento de software do que scripts independentes realizados *a posteriori,* como por exemplo os realizados em JUnit. Assim, permite desenvolver melhor software, fazendo a equipa pensar sobre os modelos a implementar.
Contudo, tendo em conta que é um novo tipo de teste, “foge” ao paradigma dos testes habituais desenvolvidos por *testers* profissionais, obrigando assim a uma fase de aprendizagem da técnica.
Para realizar os testes ao programa, utilizámos a ferramenta QF-Test.

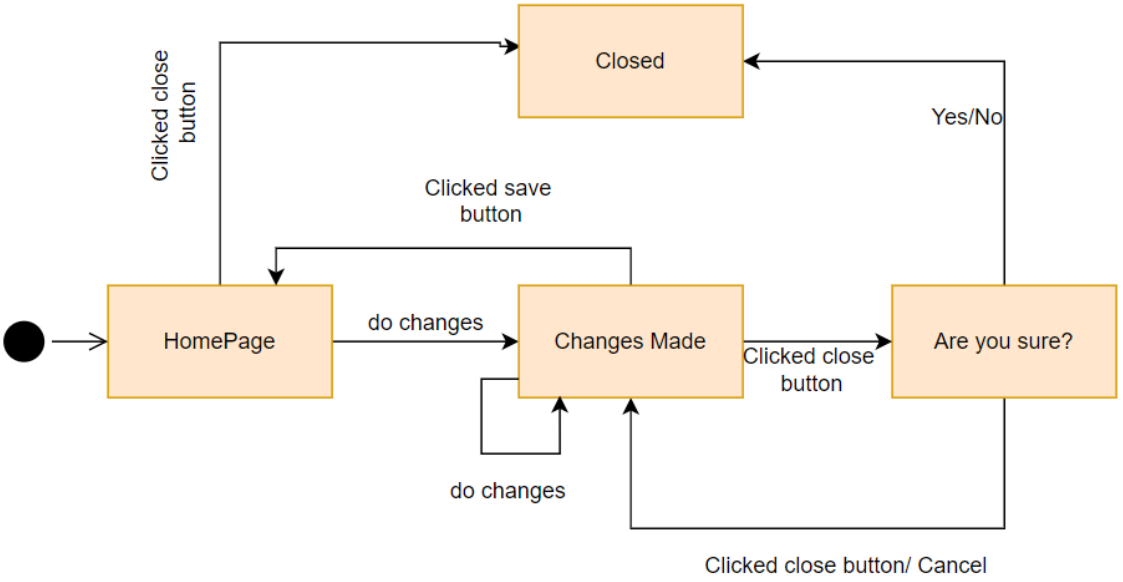
QF-Test

Esta ferramenta é usada para automatizar testes funcionais para aplicações Java ou web que tenham uma interface gráfica.
Para poder testar o programa jdotxt, tivemos que comentar os testes, das outras técnicas de black-boxing, que lançassem excepções ou erros, e de seguida fazer build, recorrendo ao maven. O resultado deste processo é um jar localizado na pasta “target”, chamado “jdotxt-0.4.9-SNAPSHOT-jar-with-dependencies”. Iniciámos um test-suite, utilizando o jar do projeto.
Os testes gerados foram feitos pré-execução.

Caso de Uso - Editar tarefa, Guardar e sair do programa

Escolhemos este caso de uso para ver como é que o sistema se comporta, quando uma tarefa foi editada e o utilizador quer sair do programa. Para um utilizador poder sair do programa, só em alguns estados do sistema é possível. Por exemplo, é esperado que um utilizador não possa sair do programa com alterações em curso, oferecendo a possibilidade de descartar ou guardar as mesmas, primeiro.

State Machine:



Descrição:

Esta máquina de estados representa o comportamento do sistema, durante o processo de alteração de tarefas, guardá-las e fim do mesmo.
O sistema começa na **HomePage** que apresenta as tarefas até ao momento guardadas (O sistema acede a um ficheiro local chamado todo.txt e faz upload para o sistema das tarefas lá presentes). Deste estado, o utilizador pode:

- Sair do programa, clicando no “X” presente no canto superior direito da interface;
- Pode editar o nome de alguma das tarefas.

Caso opte pela primeira opção, o sistema parte para o estado **Closed**. Caso opte pela segunda opção, parte para o estado **Changes Made**.
O estado Changes Made representa o momento da execução em que houve alterações às tarefas, contudo ainda não foram guardadas localmente. Deste estado o utilizador pode:

- Continuar a fazer alterações, mantendo-se no mesmo estado;
- Guardar, clicando no botão “Save”, e movendo o sistema para o estado inicial, **HomePage**;
- ou tentar fechar o programa, clicando no “X” presente no canto superior direito da interface, partindo para o estado **Closed**.

Tentando fechar o programa, com alterações em curso, resulta numa caixa de diálogo a notificar o utilizador de que as suas alterações não foram guardadas. Assim, ele tem 3 opções:

- Clicar *Yes*, e guarda as alterações e termina o programa, transitando para o estado **Closed**;
- Clicar No, e as alterações são descartadas e termina o programa, transitando para o estado **Closed**;
- Clicar Cancel, em que termina a caixa de diálogo, e recua o sistema para a transição anterior, neste caso a **Changes Made**.

Nesta máquina de estados, não está contemplado guardar ou descartar as alterações, quando o utilizador clica *Yes* ou *No*, respetivamente, visto que o objetivo deste modelo é mostrar o comportamento da interface gráfica e não a disposição de dados.

Transition Table:

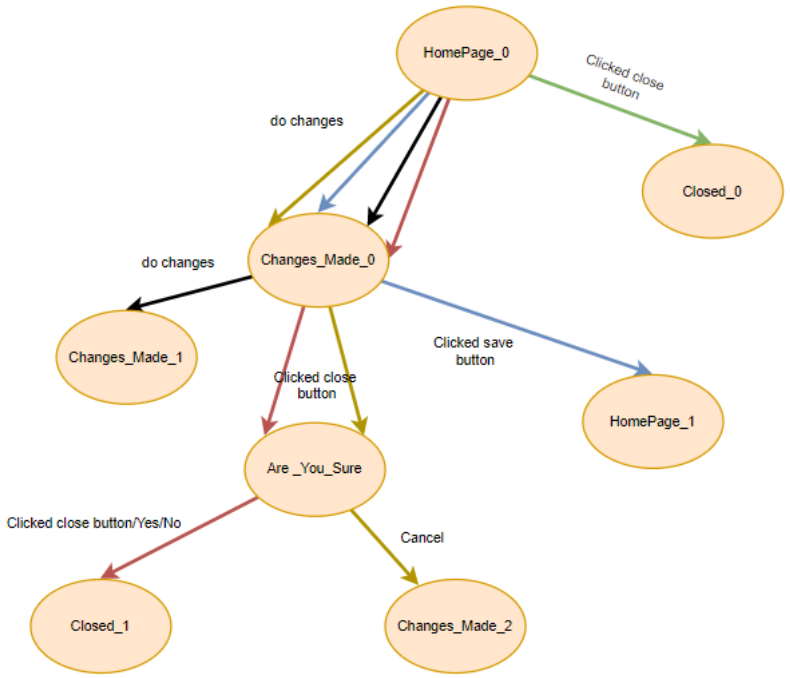
Estado	Eventos					
	Do changes	Clicked close button	clicked save button	Cancel	Yes	No
HomePage	Changes Made	Closed				
Changes Made	Changes Made	Are you sure ?	HomePage			
Are you sure ?		Closed		Changes Made	Closed	Closed
Closed						

Descrição:

Esta tabela tem como propósito representar qual o resultado de determinadas transições a partir de determinados estados.
Muito semelhante à *Transition Machine*, dá para observar o comportamento do sistema quando o utilizador realiza determinados inputs. Por exemplo:

- Partindo da página inicial (***HomePage***), efetuando *Do Changes,* transição responsável pela alteração do nome de uma tarefa, atinge-se o estado **Changes Made**, estado do qual há alterações em curso não guardadas.
- Partindo da página inicial (***HomePage***), não existe nenhuma interação possível na interface gráfica que possibilite ao utilizador guardar, clicar *Yes/No/Cancel* (opções só disponibilizadas pela caixa de diálogo no estado ***Are you sure?****;*
- É de notar também que o estado ***Closed*** representa o programa terminado, daí não haver transições possíveis.

Transition Tree:



Descrição:

A *transition tree* acima é composta por 4 estados principais (HomePage, Changes Made, Closed e *Are You Sure)* e 7 eventos diferentes. Esta árvore permite ver quais as interações possibilitadas ao utilizador quando este está na interface gráfica do programa.
Os estados estão representados por “bolas” e as transições por setas.
Analisando a árvore dá para observar que há 11 caminhos possíveis para o utilizador seguir, cada um colorido diferentemente.
Os nós encontram-se numerados para representar diferentes estados em diferentes momentos da execução do sistema. Os nós “Zero” representam o comportamento possível em cada estado, enquanto que os “não-Zero” são uma repetição dos anteriores e por isso não há necessidade de repetir o comportamento e tornar a árvore infinita.
Transition Tree têm como objetivo minimizar e selecionar o número de testes necessários para validar a state machine (definida anteriormente). Tendo em conta que há 5 nós-filho, serão feitos 5 testes.

Testes realizados:

Para este caso de uso, foram realizados 5 testes, que correspondem às transições definidas na tabela de transições acima. De forma a mostrar as interações na GUI de forma perceptível e pausada, cada evento terá um *delay* de 3000ms.

- O primeiro teste corresponde ao utilizador estar na página inicial e terminar o programa, clicando no botão no canto superior direito. Chama-se **HomePage_Clicked_Close_button**
- O segundo teste corresponde a fazer uma alteração a uma tarefa, previamente inserida. Chama-se **Changes_Made_Change_Task_Name**.
- O terceiro teste é para testar se o utilizador terminar o programa sem salvar as suas alterações, têm possibilidade de as guardar ou descartar. Neste caso, o utilizador clicaria no botão “Cancel” que tem como funcionalidade cancelar o término do programa e, por isso, retornar ao estado ***Changes Made.*** Este teste chama-se **Are_You_Sure_Cancel.**
- O quarto teste é semelhante ao anterior, na medida em que é disponibilizada a caixa de diálogo, contudo o utilizador pode clicar *Yes* ou *No* (que têm o mesmo efeito na GUI, apesar de um deles guardar as alterações), terminando assim o programa. A este teste chamámos **Are_You_Sure_Yes.**
- Ao teste final chamámos **Changes_Made_Clicked_Save_Button,** que corresponde ao conjunto de transições que leva o utilizador da página principal a alterar uma tarefa e, de seguida, guardá-la.
- Para a realização dos testes que guardam alterações, foi necessário criar uma tarefa *a priori* e, no final do mesmo, apagá-la.

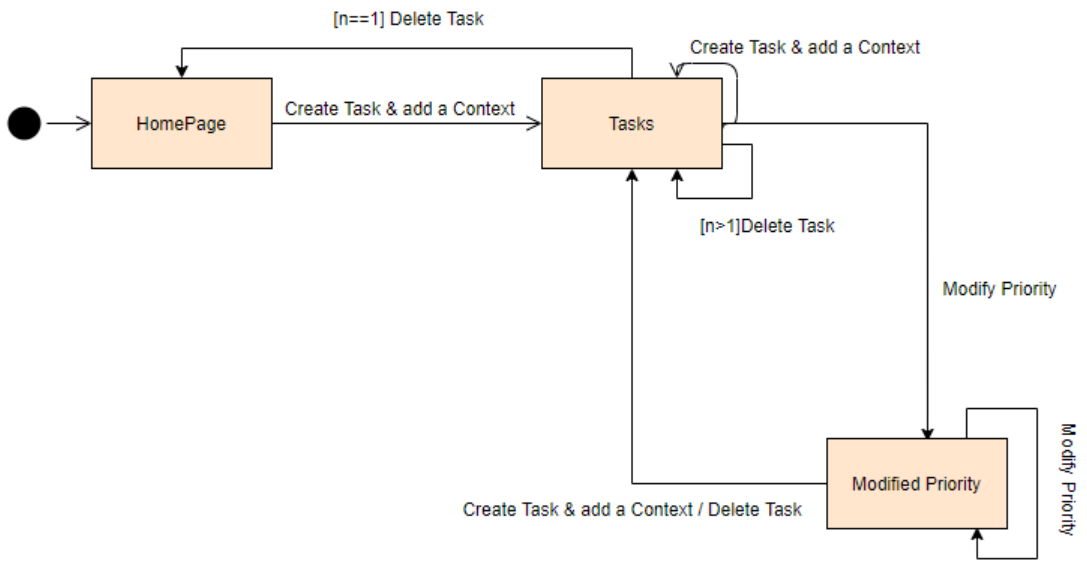
Sneak Path:

O *sneak path* que decidimos analisar é uma transição de estados que pode permitir ao utilizador guardar dados, sem ter feito alterações.
Recorrendo à **Transition Table**, pode-se deduzir que o utilizador encontra-se no estado **HomePage** e tenciona ativar o evento “clicked save button”. A junção destas células, na tabela, corresponde a uma célula vazia, e por isso, é um comportamento desconhecido. Criando um test-set que iniciali o programa e um test-case que contém a interação do utilizador a clicar no botão de salvar.
Ora, o botão encontra-se inativo, logo, quando o programa inicia e é feito o teste, resulta numa **DisabledComponentException,** excepção que indica que um componente não se encontra disponível para uma determinada interação, neste caso é um clique.
Assim, pode-se concluir que não é possível o utilizador estar no estado **HomePage** e realizar o evento “clicked save button”.

Caso de Uso - Adicionar Tarefas a contextos, e mudar as suas prioridades:

Caso de uso escolhido de forma a ser possível verificar o comportamento do programa aquando a situação de adição de tarefas a contextos fornecidos pelo utilizador, alterando no final a ordem de prioridade destas tarefas, de modo a alterar a ordem de demonstração das mesmas.

State Machine:



Descrição:

Neste caso de uso, o sistema começa na **HomePage**, que irá apresentar o estado atual do sistema (as tarefas, projetos e contextos guardados até ao momento).
Em seguida, será adicionada uma tarefa a um contexto (**Por exemplo:** Fazer jantar @Alimentacao), para adicionar a tarefa "Fazer jantar" ao conceito de alimentação, ficando assim no estado **Tasks,** que consiste num estado que têm no mínimo uma tarefa.
O mesmo é feito para se adicionar novas tarefas, permanecendo no estado **tasks**, adiciona-se uma nova tarefa ao mesmo contexto (**Por exemplo:** Preparar almoços para a semana @Alimentacao).
Para terminar este caso de uso, o utilizador altera a prioridade de uma tarefa adicionada, de modo a alterar a ordem de demonstração dos mesmos por parte do programa.

Transition Table:

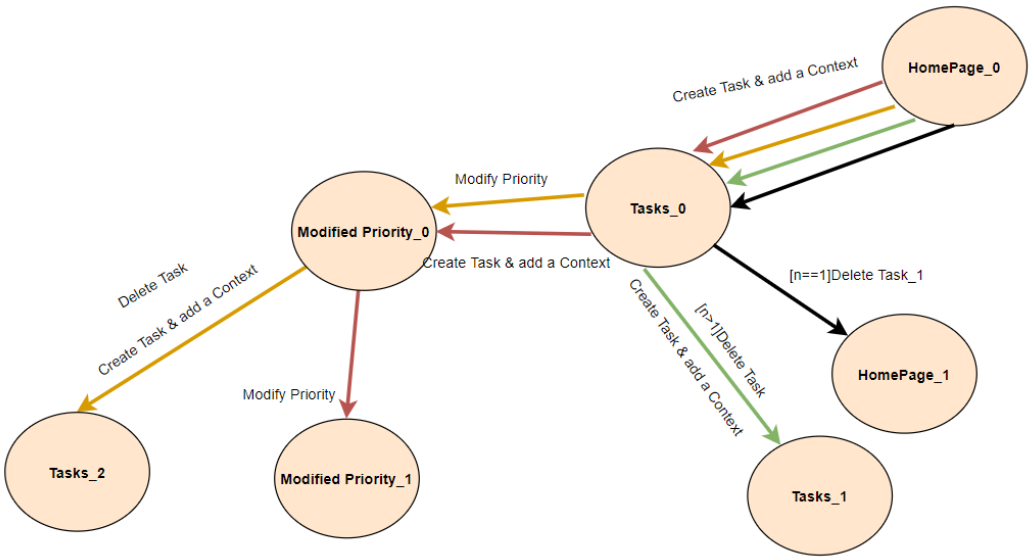
Estado	Eventos		
	Create Task & Add Context	Delete Task	Modify Priority
HomePage	Tasks		
Tasks	Tasks	HomePage/Tasks	Modified Priority
Modified Priority	Tasks	Tasks	Modified Priority

Descrição:

Acima encontram-se as transições entre estados encontrados para este caso de uso.
Os estados acima descrevem:

- Através da página inicial, é possível criar tarefas e adicionar-lhes um contexto, passando sempre para o estado **Tasks** como um resultado dessa acção;
- A partir desse estado, é possível tomar três caminhos, apagar a tarefa já existente, e retornar ao estado **Homepage**, adicionar uma nova tarefa, permanecendo no estado **Tasks** ou modificar a prioridade da tarefa, mudando para o estado **Modify Priority**.

Transition Tree:



Descrição:

Esta árvore de transição é composta pelos principais estados do caso de uso a que pertence (**HomePage, Tasks, Modified Priority**).
Esta árvore pode ser utilizada para verificar quais as transições possíveis entre estados com base nas interações do utilizador com o programa.
É possível observar que há 10 caminhos para percorrer, todos coloridos respetivamente. Serão feitos testes com base nestes nós.

Testes realizados:

Para testar este caso de uso, foram produzidos vários test-sets usando a ferramenta QF-Test, uma para cada transição da tabela;

- Para começar os testes:
    - Inicialmente, é adicionado uma nova tarefa com um contexto;
    - Existindo já uma tarefa, é adicionada uma nova tarefa para se manter o estado **Tasks**;
    - Após serem adicionadas ambas as tarefas, é modificada a prioridade da 2ª tarefa adicionada, que após processada, altera a ordem das tarefas no programa, movendo para o estado **Modified Priority**;
    - Após isto, são removidas as duas tarefas, retornando ao estado original, **Homepage**;

Sneak Paths:

Dentro deste caso, é possível delimitar alguns “Sneak Paths”:

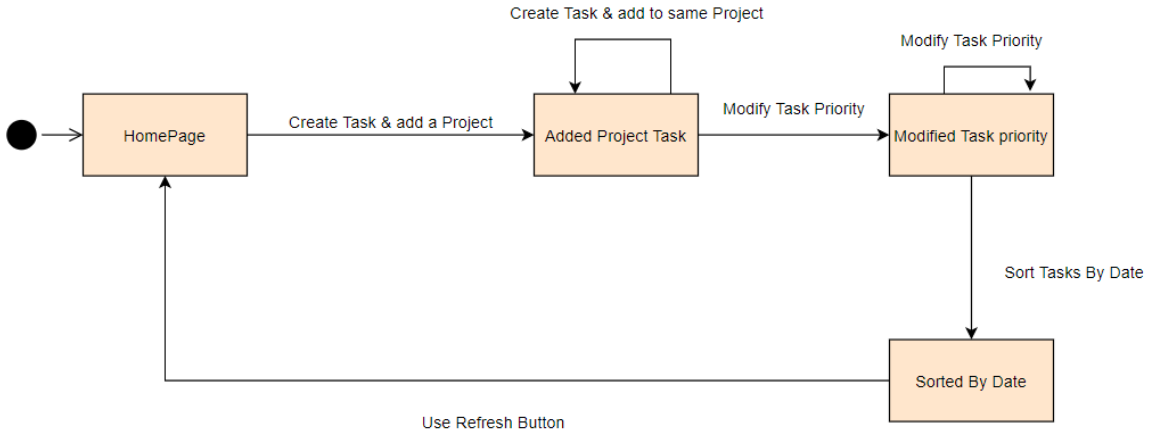
- Após adicionar a 1ª tarefa **(Tasks)**, é possível seguir para o estado **Modify Priority** alterando a prioridade da tarefa já existente.
- Após adicionar a 2ª tarefa **(Tasks)** é possível apagar essa mesma tarefa, retornando ao estado **(Tasks).**

Nenhum destes sneak paths deveriam ser barrados, porque embora inválidos para o caso de uso a ser testado, são dois caminhos que utilizam as funcionalidades base do programa, sendo necessários para outros casos de uso.

Caso de Uso - Adicionar Tarefas a Projetos, modificar as suas prioridades e alterar a ordem (sort) das tarefas:

Caso de uso escolhido de forma a ser possível testar o comportamento do programa aquando a situação de adição de tarefas a projetos fornecidos pelo utilizador, alterando a prioridade das mesmas, de forma a alterar a sua ordem, alterando, em seguida, a função base de sorting utilizada pelo programa, de modo a alterar a demonstração das tarefas com base na data de criação das mesmas.

State Machine:



Descrição:

Neste caso de uso, o sistema começa na **HomePage**, que irá apresentar o estado atual do sistema. Será, em seguida, adicionada uma tarefa a um projeto, adicionando um novo projeto. **(Por exemplo:** Escrever relatório #VVS), para adicionar a tarefa “Escrever relatório” ao projeto VVS, ficando assim no estado **Added Project Task**.
O mesmo é feito diversas vezes para adicionar novas tarefas ao projeto, até terminar de adicionar as tarefas pretendidas, assim que isso acontecer e a pessoa decidir alterar as prioridades das tarefas, muda para o estado **Modified Task Priority**, podendo aí alterar as prioridades com base na sua preferência.
No final deste caso de uso, altera-se a função de sorting, alterando para ordenar as tarefas com base na data de criação, contrariamente à função que estava a ser utilizada, com base nas prioridades.
No final, utilizando o botão de refresh, é possível retornar ao estado **HomePage**.

Transition Table:

Estado	Eventos			
	Create Task & Add to project	Modify Task Priority	Sort Tasks By Date	Use Refresh Button
HomePage	Added Project Task			
Added Project Task	Added Project Task	Modified Task Priority		
Modified Task Priority		Modified Task Priority	Sorted By date	
Sorted By Date				Homepage

Descrição:

Acima encontram-se as transições entre estados encontrados para este caso de uso.
Os estados acima descrevem:

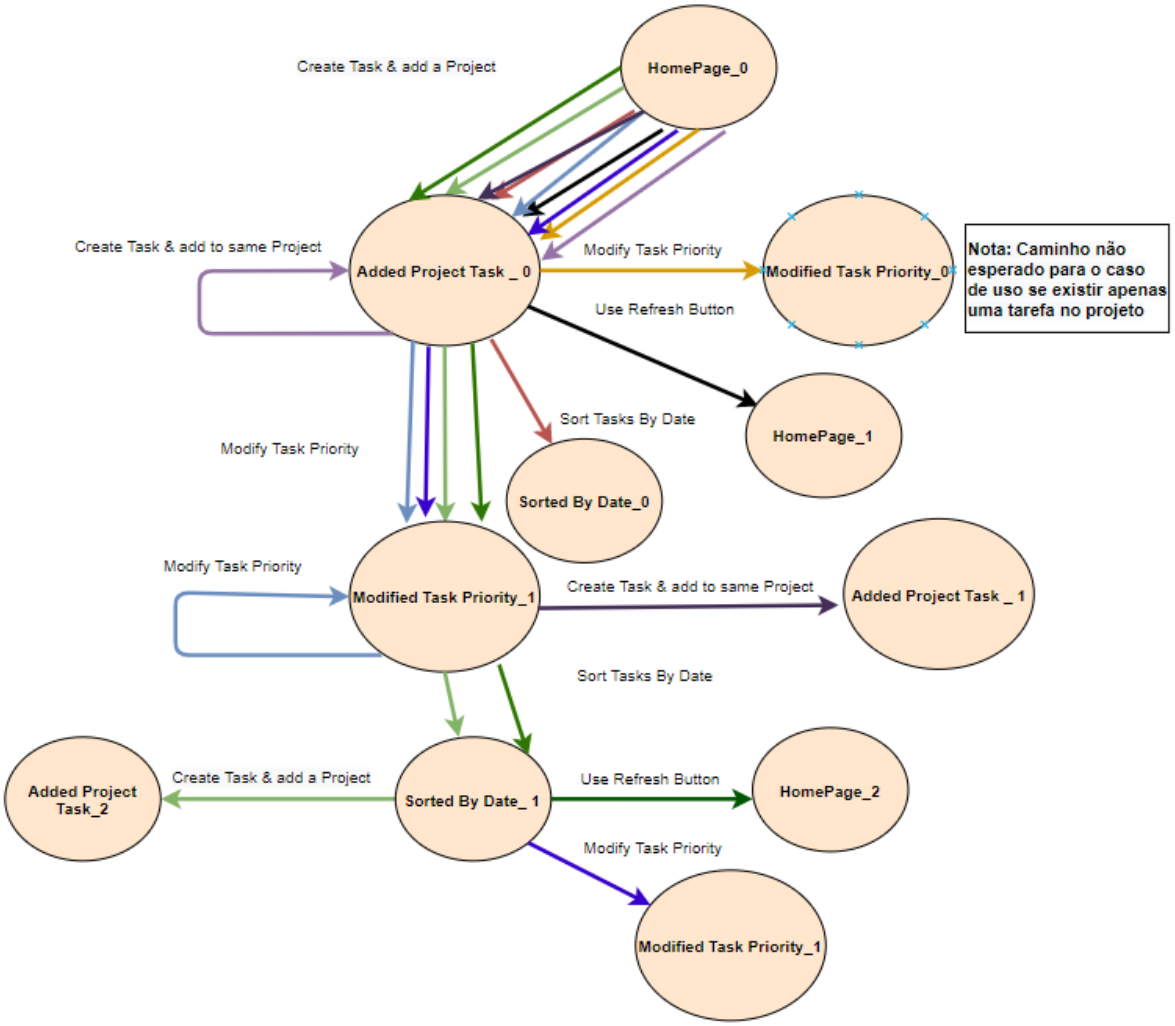
- Através da página inicial, é possível criar tarefas e adicionar-lhes um projeto, passando para o estado **Added Project Task** quando tal é feito, permanecendo no mesmo estado enquanto tarefas do projeto continuarem a ser adicionadas.
- Após terminar de adicionar tarefas ao projeto, poderá passar para o estado **Modified Task Priority** ao mudar o nível de prioridade de uma tarefa. Poderá permanecer neste estado ao alterar a ordem de prioridade de qualquer uma das tarefas já existentes.
- Em seguida, é possível utilizar o botão de *sort* para alterar o que prioritizar na demonstração das tarefas, mudando para o estado **Sorted By Date**, que demonstra as tarefas por ordem de data de criação.
- Para terminar o caso de uso**,** é possível utilizar o botão de refresh, levando o programa de volta para o estado **HomePage.**

Testes realizados:

Para testar este caso de uso, foram produzidos vários test-sets usando a ferramenta QF-Test, com um test-set para cada transição da tabela;

- Para começar os testes:
    - Inicialmente, é adicionado uma nova tarefa com um novo projeto, movendo para o estado **Added Project Task**;
    - Existindo já um projeto com uma tarefa inserida, insere-se mais uma tarefa (O mínimo para ser possível utilizar funções de sort);
    - Após inseridas as tarefas, alteram-se as prioridades de ambas as tarefas de forma a alterar a sua ordem, passando para o estado **Modified Task Priority**, podendo alterar quantas prioridades forem necessárias.;
    - Em seguida, é alterada a função de sorting, utilizando a data de criação da tarefa como argumento da função, passando para o estado **Sorted By Date;**
    - Como último teste, é utilizado ao botão de refresh, para garantir que todas as mudanças foram apagadas, retornando ao estado **HomePage**;

Transition Tree:



Descrição:

Esta árvore de transição é composta pelos principais estados do caso de uso ao qual pertence (**HomePage, Added Project Task, Modified Task Priority, Sorted By Date**). Esta árvore pode ser utilizada para verificar quais as transições possíveis entre estados com base na interação do utilizador com o programa. Existem vários caminhos possíveis entre os estados, que acabaram por ser testados utilizando o QF-Test, e demonstraram vários sneak paths.

Sneak Paths:

Dentro deste caso, é possível delimitar alguns “Sneak Paths”:

- Após adicionar a primeira tarefa ao projeto, ficando no estado **Added Project Task**, é possível mover para os outros estados:
    - Usando o botão de refresh, é possível retornar para o estado **HomePage;**
    - Usando a função de Sort, por data de criação é possível tentar uma passagem para o estado **Sorted By Date**, uma transição que retorna uma exceção no programa. **(ArrayIndexOutOfBoundsException);**
    - É também possível alterar para o estado **Modified Task Priority**, alterando a prioridade da tarefa logo após adicioná-la (Algo que só seria parte do percurso normal do caso de uso se acontecesse após a inserção de todas as tarefas);
- Após alterar a prioridade da primeira tarefa, ficando no estado **Modified Task Priority**, é possível mover para outros estados:
    - Usando o botão de refresh, é possível retornar para o estado **HomePage;**
    - Usando a função de Sort, por data de criação é possível uma transição para o estado **Sorted By Date** (Algo que só poderá ser aceite pelo programa caso exista mais do que uma tarefa);
    - É possível retornar para o estado **Added Project Task** (Algo que não deve ser proibido, sendo uma parte crucial do programa);
- No final, após alterar a função de sort a ser utilizada sobre as tarefas:
    - É possível retornar ao estado de **Modified Task Priority**, alterando uma das prioridades das tarefas (Algo que não deve ser proibido, sendo também uma funcionalidade core do programa);
    - É possível retornar ao estado **Added Project Task**, adicionando uma nova tarefa ao projeto **(**Algo que não deve ser proibido, sendo também uma funcionalidade core do programa);

QF-Test Feedback:

For first time users, we consider that this tool served its purpose. Yet, we had several difficulties:

- There weren’t explicit tutorials to explain how the component’s interface worked or what it required to implement.
- We didn’t find any useful information about how we could change files in before and after executing tests.
- While testing one of the sneak paths on the program, QF-Test launched the exception DisabledComponentException, yet didn’t show it on the terminal. We had to search on the logs. We don’t know if it is on purpose.
- The UI is not totally obvious for newer users. The difference between Test-case and Test-Set is one that had to searched online, and the fact that Test-Sets could be chained with Test-Cases didn’t help to explain their difference;
- Procedures remained unused, we were somewhat confused as to their purpose.