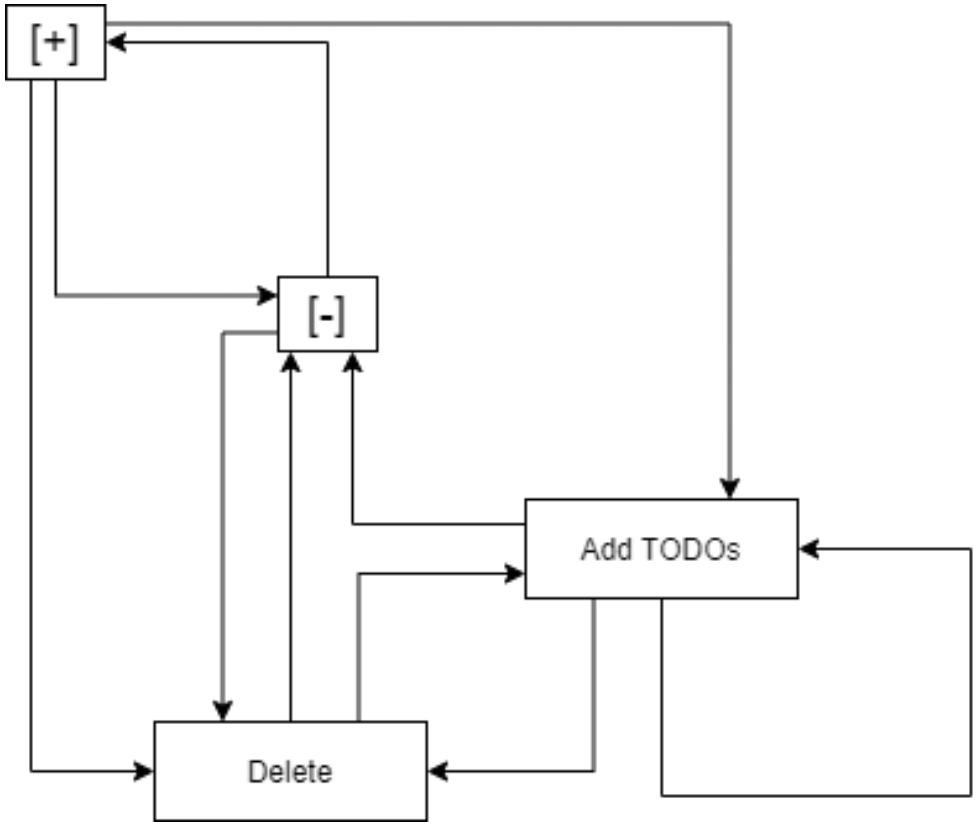
Ans 1)



Q3) Briefly explain the trade-offs involved in stopping at event coverage for this program, rather than pursuing event interaction coverage or length-n event sequence coverage. What kinds of tests are these simpler coverage criteria not forcing you to write? What are the advantages of not writing them (think about how many tests you'd need for the stronger criteria…).

Ans)

Event flow graphs focus on representing all possible interactions among the events whereas event interaction coverage focuses on representing all possible events and their interactions once any event e is performed. Likewise, Length-n event sequence coverage focuses on representing all event sequences of a given set of length n. Event flow graphs produce tests that cover all functionalities with minimal effort.

However, with the event interaction and length-n event sequence coverage, producing tests to cover all the functionalities take a lot more time and effort. This is because of the exponential increase in cases with the length of the sequences of the events. Although event flow graphs cover all the functionalities, hidden states are not covered or could be discovered with these tests compared with the other two strategies, hidden states can be discovered.

Also, with many large applications, determining all events sequences are just not possible at the initial stage so achieving good coverage with the event interaction

coverage or length-n event sequence coverage is not feasible. That's why writing tests with event coverage would give developers a better understanding of the system. Moreover, with the new changes, it would be even difficult to add/change written tests in the case of event interaction or length-n event sequence coverages. In the case of event flow coverage, tests can be easily added to represent new changes.

Q4) Part 1 requests an event flow graph, which has a fixed number of nodes. As you likely noticed when designing the event flow graph, this page's UI does *not* have a fixed number of UI elements. So you have derived tests from an abstraction that mimics some aspects of the program and hides others. Please consider the consequences of this mismatch. If this event flow graph is the basis for all of your tests, what sorts of problems are unlikely to be discovered? Conversely, are there any ways in which the fixed-size assumption helps simplify testing in a good way? Does it encourage you to write tests for impossible scenarios? Would any of these issues be affected by working with a larger event flow graph (say, up to 5-element TODO lists)?
Ans)

This mismatch could result in missing any hidden internal state that is not easy to observe. This abstraction is driven through the reasoning of the developer so very much dependent on the developer to include all interactions between the fixed nodes. It became hard to follow by other developers as event-flow graphs with this mismatch look inconsistent and inaccurate at first look. Likewise, this problem also arises with the event-interaction or length-n event sequence coverage due to the overlapping of many event sequences and a gigantic number of interactions.

It is hard to discover any bad state of the system raised due to a specific event sequence from fixed-size assumptions. This happens all the time in the system where unknown event sequences break the system to produce unprecedented results. The magnitude of this problem gets even bigger in the case of more elements such as a 5-element TODO list due to an exponential increase in the test cases.

In large-scale applications, fixed-size assumptions help in simplifying the interactions between the different events. It helps in recognizing the coupling between different events helping developers to check the need for any particular interaction. In many systems, it is non-feasible to identify all event sequences making productions of tests using that strategy non-feasible as well. However, in the case of small applications, we should write tests to cover impossible scenarios to solve the above-mentioned problems.