**Problems:**

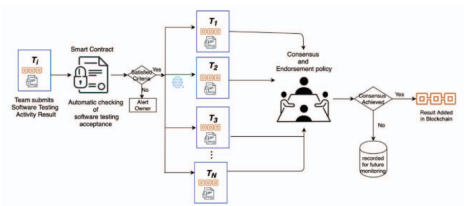
1. Team T produces results for software testing as per assignment in the smart contract. If the result R, generated by Tk meets the acceptance criteria, R is added to the blockchain as a blockchain transaction. If R generated by T does not meet the acceptance criteria, T is alerted by

the blockchain that R is not successful and added to the blockchain with the rejection message.

All these unsuccessful results when added to the blockchain lead to the problem of **Storage Overload.**

2. Every transaction that comes out of the consensus and endorsement policy is added to the blockchain directly if it satisfies the consensus criteria otherwise it will be recorded for future reference.

But the rejected transaction is being stored for future reference. This may again create a problem of **Storage Overhead.**

****

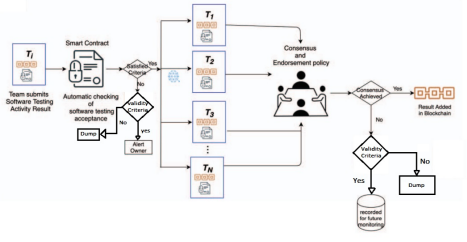
**Solution:**

**The proposed solution is that we introduce a validity criteria before we decide to store it somewhere.**

**It can be introduced after checking the satisfaction criteria in the Smart contract and also after checking the consensus criteria.**

**The way this will work is that a certain x% will be set as the validity criteria by the owner. Only the transactions fulfilling this will be considered desirable to be stored for future reference. The transactions failing to do so are assumed to be of no use in future reference and hence are dumped.**

**This will reduce the Storage Overhead significantly and also increase the data processing speed, as low data processing speed is due to the consensus protocols used for recording each transaction in the blockchain for data ownership, auditability, and the prevention of injection attacks in the software testing process.**

**The change in the current Flowchart diagram that we suggest:**

**Determining the value x used in the validity criteria**

**For every smart contract C(i,k), Team Tk, produces Rt ,1<= t <= Z , results.**

**We will generate all the possible numbers of transactions for a given smart contract (suppose N). Out of these transactions we have assumed that the bottom p% of them are considered not important and hence dumped.**

**Here, the number P will be predetermined by the owner of that smart contract. It’ll be already mentioned inside the attributes of the smart contract. Now the lowest P% of the N transactions will be analyzed and the highest fault percentage will be the value of X.**

**Here, the fault percentage for every transaction will be calculated as follows:**

**FP = (Output Received) / (Ideal Output Mentioned In C(i,k) ) 100**

**Pseudocode**