State Diagrams - A Case Study

I found this image on Google (https://thecentergame.com/state-machine-vending-machine/state-machine-vending-machine-magnificent-state-diagrams-the-vending-machine-shown-in-figu-chegg) and I thought I would try to illuminate more on the subject matter in an attempt to demonstrate a higher fluency in the subject. This is an image for a vending machine, whose main goal is to achieve $0.20 in order to disperse some candy, or it could be pop, whatever it is – it costs $0.20. This diagram shows multiple loops and has a diagram located above it that helps to shed some light on what it's trying to accomplish.

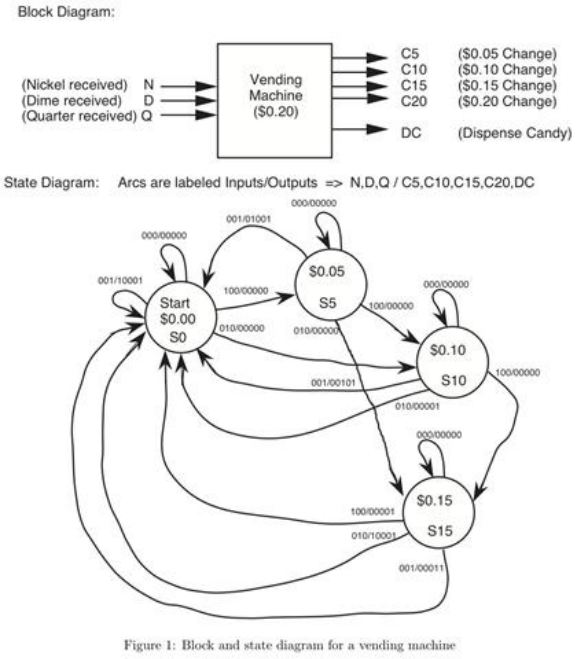

Figure 1: Block and state diagram for a vending machine

2

We know that it will take $.20, but it should give changes back when the time comes. So, with the coded diagram listed above, it has whenever someone puts in a quarter(Q) the output is DC which means distribute candy(DC), and (C) $0.05 change back to the customer. Now since dimes and nickels are still in circulation, and the machine doesn't accept any other currency, realistically there are only 3 inputs, however there are a total of 5 outputs, meaning 5 different ways to get your changes back depending on what sorts of coins you use, and in what order you are depositing the tender. Putting in one coin at a time, I've made a smaller chart below that demonstrates the different orders one could insert their change. The diagram that I created is more sequential based, as opposed to this 2-chart paradigm that we see above. My chart unveils **new data** about the above chart, in the sense of total number of steps that could ever potentially be taken given the parameters of how much the item is, and what sorts of legal tender it will accept.

As you can see below – The most possible coins that you could fit into the vending machine before it would reset is 4, specifically 4 nickels. The smallest amount of coins that could ever be used is 1 quarter. The most change you could ever have the machine spit out is $0.20, and the amount of different kinds of coins that you could input into the machine are 3 different ypes – 1 nickel, 1 dime, and 1 quarter. Additionally, there are 10 possible paths that could ever occur from these combinations of coins. This data is relevant to the software engineering elements that we work with because it shows us data in new ways we might not have thought possible previously. These can be great tool sin planning for potential outcomes and preparing boundaries within our programs and parameters so we know what to expect. For example, if the manufacturer had not thought about the 2 different ways that one could get $0.20 back in change from the machine, it may have broken the machine if there were no tests

setup in place to catch the errors. While it's probably not very likely someone would put in a dime, a nickel, and a quarter, or 3 nickels and a quarter on purpose, it could still happen, and preparing for those redundancies is crucial to the development process.