STUDY OF A MANUFACTURING FACILITY

SYSC 4005

Sara Shikhhassan 101142208
Sam Al Zoubi 101140949
Peter Tanyous 101127203

Table of Contents

Milestone 1

1. Problem Formulation

This study aims to evaluate the performance of a manufacturing plant. The facility assembles three distinct products, P1, P2, and P3, each of which is made up of one or more component kinds. C1, C2, and C3 are the three separate components utilised in the creation of the items. Product P1 has one component C1, product P2 has components C1 and C2, and product P3 has components C1 and C3.. Inspector 1 and Inspector 2 are in charge of cleaning and fixing the components. Inspector 1 operates on component C1, whereas Inspector 2 works in a random sequence on components C2 and C3. Inspectors never have to wait for components since there is a limitless supply of them on hand.

W1, W2, and W3 are three workstations in the manufacturing facility which assemble items P1, P2, and P3, respectively. Each workstation has a buffer capacity of two components, with one buffer for each type of component required. A product can only be assembled if all of the necessary components are accessible. If all buffers for a certain component type are full, the associated inspector will be deemed "blocked" until an opening becomes available. Inspector 1 sends component C1 to the buffer with the fewest pending components. In the event of a tie, W1 has the greatest priority and W3 has the lowest.

The simulation aims to assess the performance of the facility based on the historical data of the inspectors' and workstations' service times and average buffer occupancy. The study will focus on the facility throughput, the probability of each workstation being busy, the average buffer occupancy of each buffer, and the probability of each inspector being blocked. An additional objective is to evaluate the current routing policy followed by Inspector 1 and possibly recommend an alternative policy to increase throughput and decrease inspector blockage time. The simulation study will include statistical justification and validation of the random aspects of the model, steady-state estimates of the quantities of interest with 95% confidence intervals, and at least one recommendation for an alternative operating policy.

2. Setting of Objectives and Overall Project Plan

The study is based on historical data of the inspectors' and workstations' service times, including Inspector 1’s inspection time for component C1, Inspector 2’s inspection time for components C2 and C3 , Workstation 1 processing time, Workstation 2 processing time, and Workstation 3 processing time. In addition, the average buffer occupancy for each buffer based on historical data is provided.

The quantities of interest in the study are: the facility "throughput" or product output per unit time, the probability (or proportion of time) that each workstation is busy, the average buffer occupancy of each buffer, and the probability (or proportion of time) that each inspector remains "blocked." The additional objective is to improve the policy that Inspector 1 follows when delivering C1 components to the different workstations, in order to increase throughput and decrease inspector "blocked" time.

Simulation is a suitable approach for this project as it allows for the modelling and analysis of complex systems in a controlled environment. It enables the study of the interdependence of different elements of the system, such as the inspectors and workstations, and the impact of different operating policies on the system's performance. Simulation affords the ability to validate the model, estimate steady-state values, and confidently determine the intervals of interest for the relevant quantities. The simulation results will provide valuable insights into the performance of the manufacturing facility and support decision-making for alternative operating policies.

3. Model Conceptualization

This simulation model takes four key entities into account: workstations, inspectors, components, and buffers. The focus is on two main activities: the time taken for inspection and the processing time at each workstation. It is assumed that the average buffer occupancy remains constant throughout the simulation.

The state of each component in the system is categorised as follows with its corresponding variable:

- Inspector Blocked: 2
- Inspector Busy: 1
- Inspector Idle: 0 (component is ready to be sent but waiting for an available slot in the destination buffer)
- Buffer Half-Full: 1
- Buffer Full: 2
- Buffer Empty: 0
- Number of Components in All Buffers: 0 to 10 (with a maximum of 10)
- Workstation Busy: 1
- Workstation Idle: 0

LI(t) = the inspector is blocked, busy or idle (0, 1 or 2)
LB(t) = the buffer is empty, half-full or full (0, 1 or 2)
LW(t) = the workstation is busy or idle (1 or 0)
LC(t) = the number of components in all buffers [0,10]

There are several lists or queues in the model, including the buffer queue, which sends a component to a full buffer and requires the inspector to wait until it is free. The capacity of each buffer is noted and the components are ordered by FIFO. The workstation queue includes the delay to finish assembling a product, with the components in the buffer waiting to be assembled until the workstation is finished assembling their current product. The inspector 1 queue includes the delay to finish inspecting component 1, with other components waiting and ordered by FIFO. The inspector 2 queue includes the delay to finish inspecting component 2 or 3, with other components waiting and ordered by FIFO. The collection of workstations W1, W2, and W3 are ordered by priority because when Inspector 1 is ready to send out a component to one of its assigned buffers, all three buffers have similar occupancy of components. The buffer that the component is sent to is based on priority of the collection of workstations.

The model also considers the delay in several forms. The buffer may spend time waiting for a component to be entered into a slot, or waiting for the workstation to finish assembling a product to assemble another. If the buffer contains the required components, those components will wait to be assembled together, causing a delay at the workstation. There is also a delay at inspection.

These lists and queues are important to track throughout the simulation to measure the outputs of the system and then to measure the system performance. These system measures include the facility “throughput" or product output per unit time, the probability (or proportion of time) that each workstation is busy, the average buffer occupancy of each buffer, the probability (or proportion of time) that each inspector remains“blocked" (and therefore idle).

Finally, the model encompasses various events, such as the component entering the buffer, and transitioning the state from "Buffer Empty" to "Buffer Half-Full". The second instance of this event changes the state from "Buffer Half-Full" to "Buffer Full". Additionally, there are events for the beginning and end of inspection and assembly, which result in the final product.

Components shall always be available to be inspected by their respected Inspector and then to be sent to their respected buffer and assembled by their assigned workstation. Inspector 1 is delegated 3 buffers to control, 1 for each workstation 1, 2 or 3. Inspector 2 is delegated two buffers one for each type of component it is tasked to inspect (component 2 and 3) and then is transferred to their respective buffers when finished inspecting to be assembled by the workstation 2 or 3. This process shows that the workstation and the inspectors interact with each other through the buffers. As it is the inspectors job to send components to these buffers in the correct manner when they are done inspecting a component and for the workstation to begin assembling.

4. Model Translation

4.1 Choice of Simulation Language

The simulation language chosen for this project is python as it is considered the most ideal programming language for simulation due to its ease of use, large community, and powerful libraries. Python provides a high level of abstraction that makes it easier for developers to model simulations and use its library resources to build complex mathematical algorithms comfortably. Moreover, real-world system modelling requires information storage, big datasets, computation, and data analysis that could be easily achieved using python's data management libraries such as NumPy, Pandas, and Scipy. All in all, these factors make python the most suitable language to develop simulations in many fields.

4.2 Model Implementation

The business logic of our system will be divided into four modules, the Inspector (a class for each inspector), Workstation (classes for each workstation), Buffer and the main module "Simulation" that starts the simulation process. The modules will be divided by separation of concerns and manage/delegate tasks with other classes such as inspector to buffer to workstation. The "Simulation" class is the starting point of the simulation process which will be used to initiate and keep track of the tasks between the rest of the modules.

Each class is responsible for handling and coordinating events with each other. An event will be identified as a tuple in the following format:
event = (time of event, type of event, type of component, workstation ID, inspector ID)

The inspector class will have functions to determine the target buffer to send the component, add components to the buffer, calculate the random inspection time, create an arrival event for the component at the buffer, and check if the inspector is blocked from performing any more inspections.

The workstation class will have functions to check its availability to start processing a new product, schedule departures of a product , process departures to update the status of the workstation from busy to idle after a successful departure, and calculate the service time for processing a component to output measures.

The buffer class will have functions to check if it is full, empty, or half-full as well as functions to increment/decrement components from the buffer.

The simulation file will contain the Simulation class and a main method. The Simulation class will manage the arrival and departure of components and consist of inspectors and workstations. The main method will use a loop to continuously call the Simulation classes that schedule the arrival and the departure as long as certain conditions are met. All objects will be initialised, and the future event list (FEL) will be continuously read with actions being assigned based on the event type.

At present, there will be no input modelling, and the component arrival times will be taken from data files provided. The simulation will run until all components have been processed, both inspectors are blocked, or one inspector is blocked and the other has processed all components.

The simulation model submitted with the first project deliverable (project proposal) does not fully incorporate all of the methodologies and requirements outlined in this section. This section serves as a high-level overview of the long-term objectives. For a more detailed examination of the submitted simulation code, please see section 4.4.

4.3 First Deliverable Code

In subsection 4.2, it was noted that the simulation code submitted with the initial project proposal did not fully encompass all the outlined methodologies and requirements in the model implementation. The model implementation outlines the desired outcome for the simulation model of the manufacturing facility, which will be achieved in the final deliverable.

The current simulation code is designed to capture the functionality and coordination of each class as directed by the main simulation class. However, it does not account for the time required for inspection, buffering, and work station assembly. Instead, it focuses on demonstrating the coordination of actions between the Buffer, Component, Inspector, Product, and Workstation classes based on the simulation class.

In subsequent deliverables, the simulation class will manage the arrival and departure of components and calculate wait times for each class by scheduling events based on past data. The current deliverable manually instructs the classes to perform actions and displays the status of each simulation element after each action, as demonstrated in the output screenshot (Figure 1, additional outputs can be see).

Please note that this design is subject to change in future deliverables as new automation requirements are implemented in the simulation class

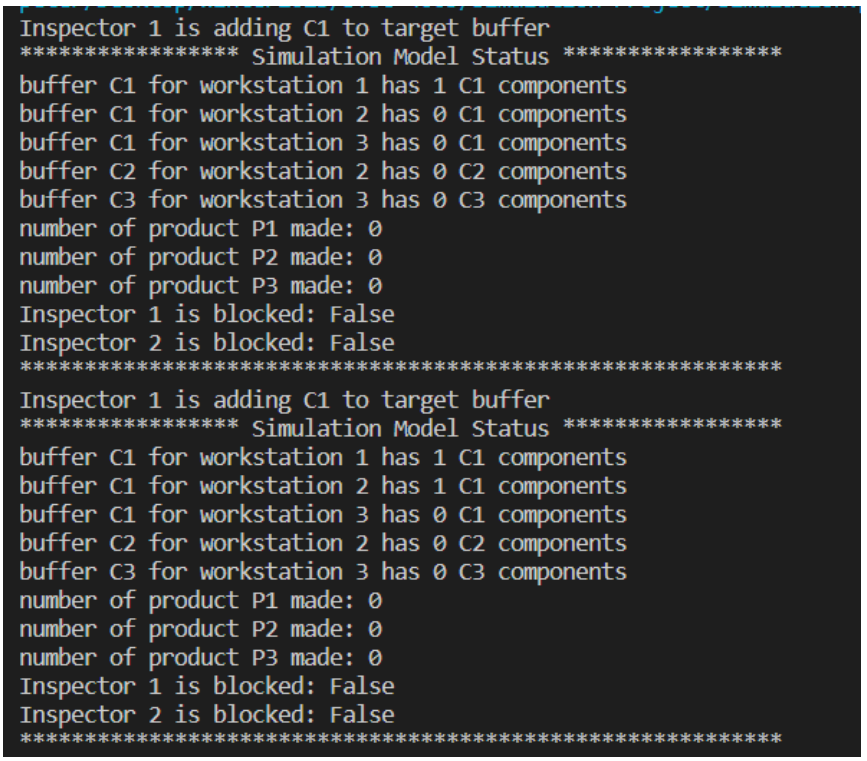


Figure 1: Snippet of simulation output

4.4 Diagrams

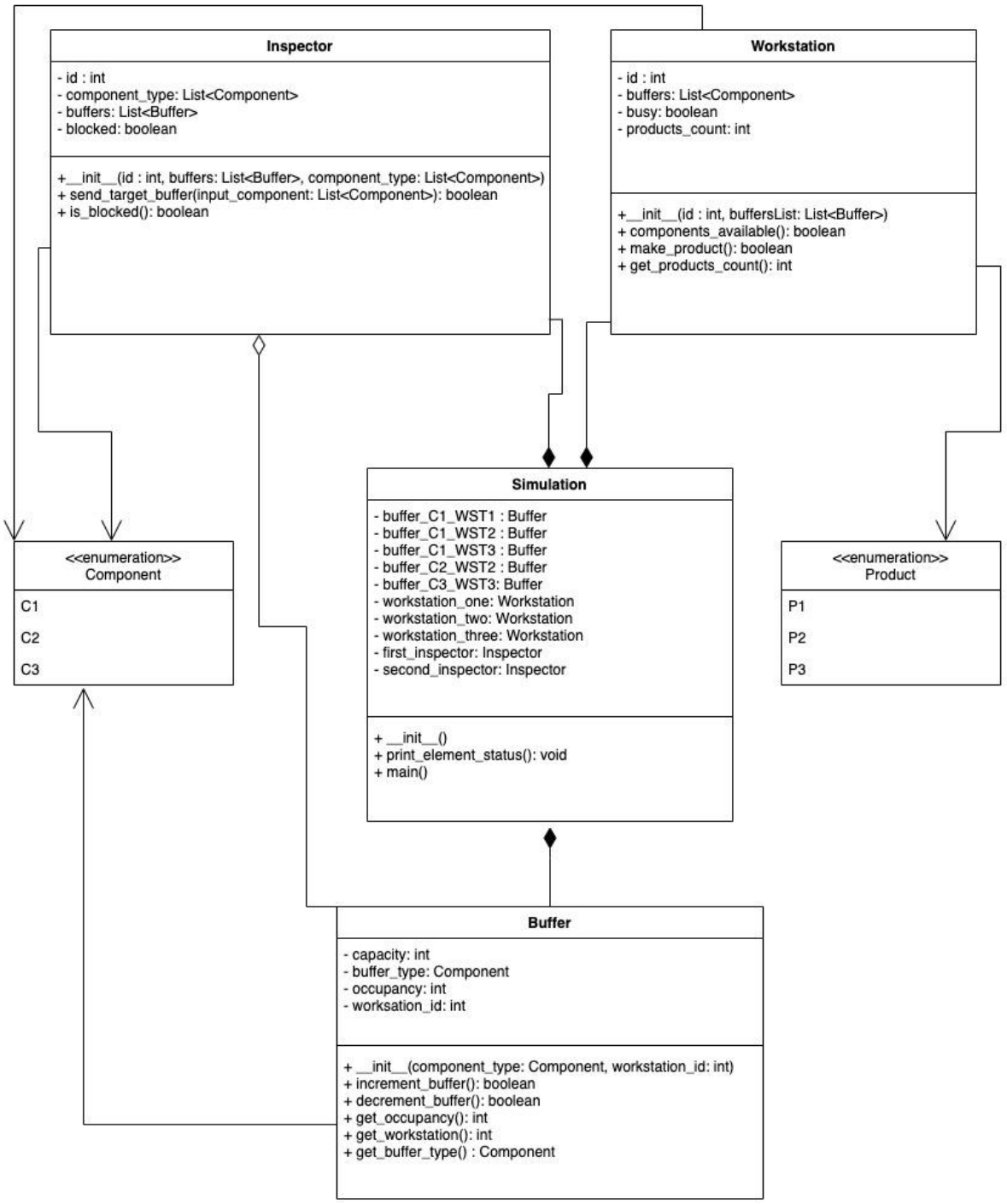


Figure 2: UML Class Diagram

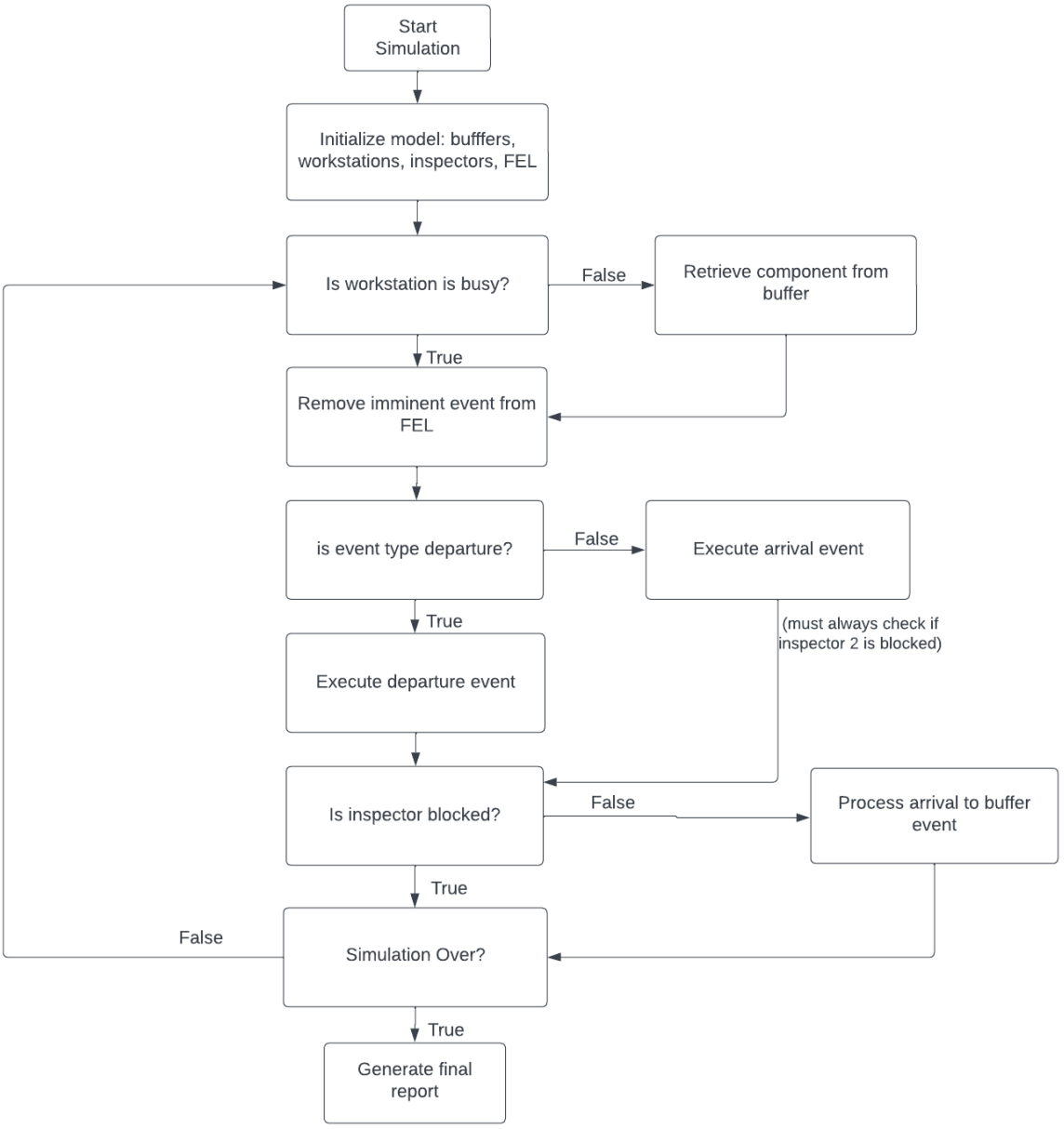


Figure 3: Flow Chart diagram of High Level Code

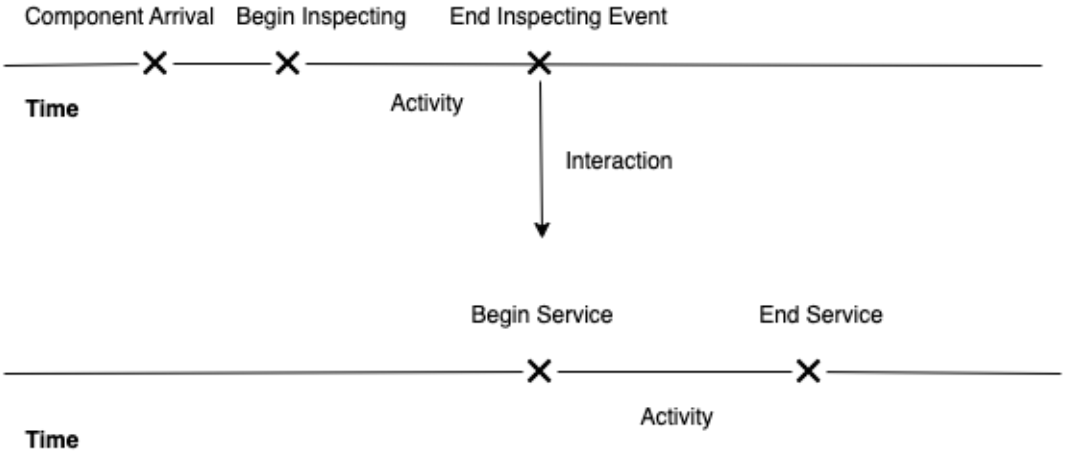


Figure 4: Inspector 1 and Workstation 1 Interaction Diagram

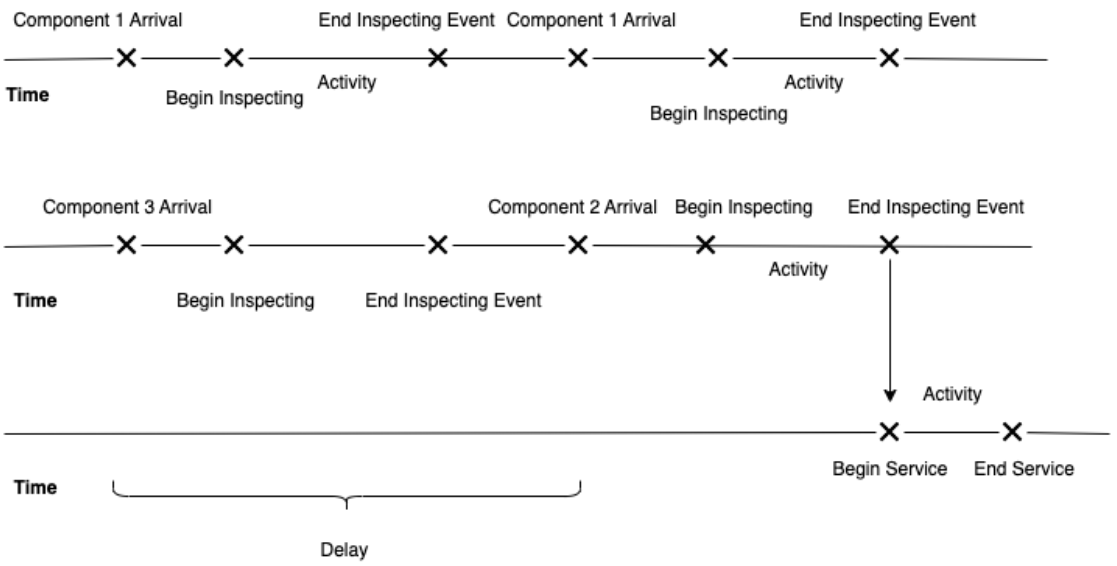


Figure 5: Inspector 1, Inspector 2, and Workstation 2 Interaction Diagram

Milestone 2

5. Data Collection and Input Modelling

We shall create a histogram for each of the historical data provided as input to the system.

The purpose of creating Historgrams for our historical data is to infer a known pmf or pdf based on the shape and lines that may aid in the identifying the resulting shape.

5.5 Histograms

Histogram For inspector 1 (servinsp1.dat):

The histogram is prepared by placing the data in class intervals. The range of the data is rather large, from 0.087 minutes to 76.284 minutes.The number of class intervals needed would be $n = \sqrt{300} = 17.32$. We shall round down to 18 to avoid intervals being too wide. Bin Width = (76.284 - 0.087) / 18 = 4.48.

Therefore we shall arrange the class intervals as follows (horizontal axis of Histogram):

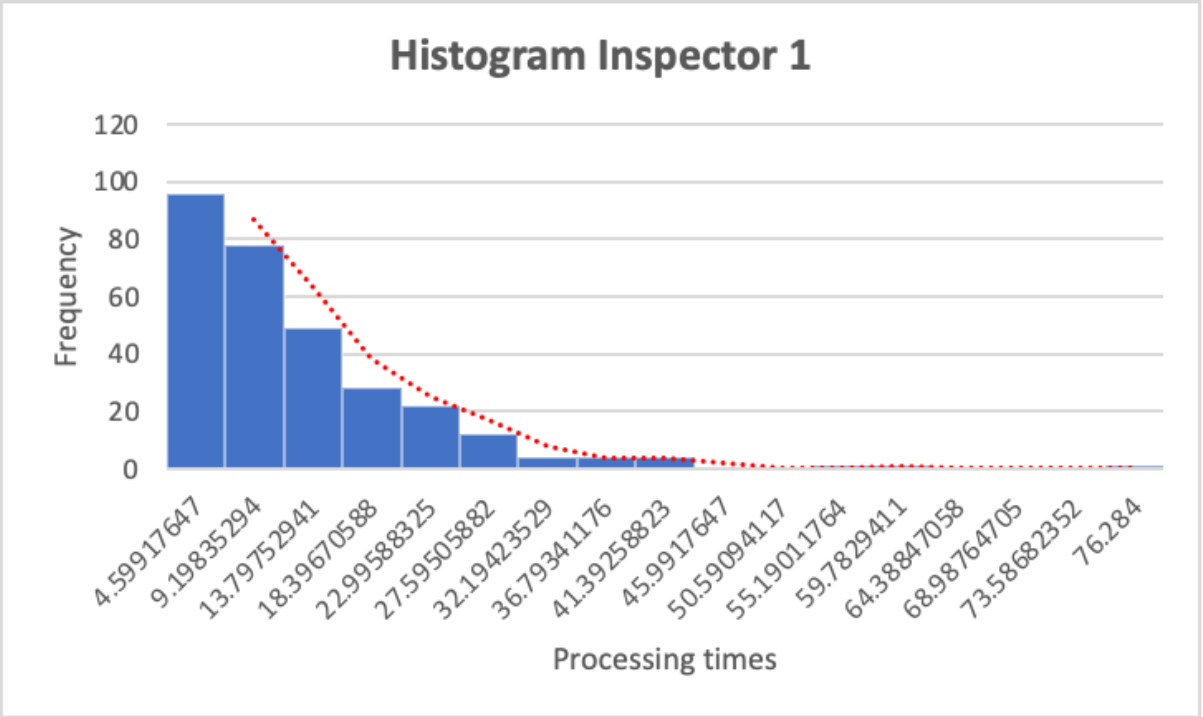
[**0.087**, 4.59917647] ,]4.59917647, 9.19835294],]9.19835294, 13.79752941],]13.79752941, 18.39670588],]18.39670588, 22.99588325] ,]22.99588325, 27.59505882],]27.59505882, 32.19423529],]32.19423529, 36.79341176],]36.79341176, 41.39258823],]41.39258823, 45.9917647],]45.9917647, 50.59094117],]50.59094117, 55.19011764],]55.19011764, 59.7829411],]59.7829411, 64.38847058] ,]64.38847058, 68.98764705],]68.98764705, 73.58682352],]73.58682352, **76.284**]

The times in bold are the minimum and maximum values from the data set, to create our class intervals we continuously add the bin width until we reach the maximum value.

Using excel, the frequency distribution amongst these intervals respectively are as follows (vertical axis of Histogram):

96, 78, 49, 28, 22, 12, 4, 4, 4, 0, 0, 1, 1, 0, 0, 0, 1

Service Times	Bin Limit	Frequency
10.16	4.59917647	96
13.508	9.19835294	78
1.586	13.79752941	49
16.705	18.39670588	28
4.552	22.99588325	22
5.818	27.59505882	12
0.362	32.19423529	4
7.095	36.79341176	4
8.989	41.39258823	4
0.358	45.9917647	0
2.256	50.59094117	0
33.691	55.19011764	1
12.34	59.7829411	1
28.015	64.38847058	0
3.853	68.98764705	0
14.182	73.58682352	0
5.869	76.284	1

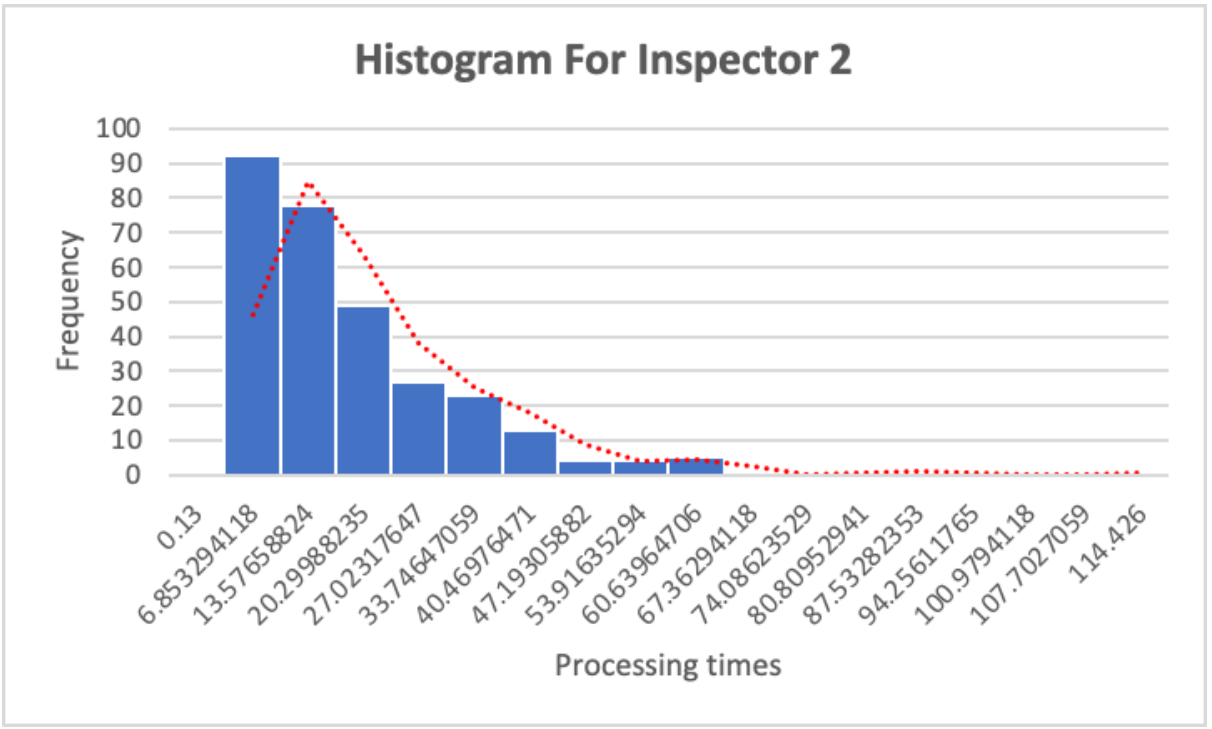


As you can see in the above histogram, it is possible to infer an exponential distibtuion from the family of distributions to be a suitable statistical model. A line is drawn through the centre point of each class interval frequency which results in a similar shape to the pdf of the exponential distribution. The same process is preformed in the Histograms of the other service times. A bell shape occurs in the first interval.

Histogram For inspector 2 (servinsp2.dat):

The histogram is prepared by placing the data in class intervals. The range of the data is from a minimum of 0.13 minutes to 114.426 minutes.The number of class intervals needed would be n = $\sqrt{300}$ = 17. 32. We shall round up to 18 in order to reach max value. Bin Width = (114.426 - 0.13) / 18 = 6.72.

Service Times	Bin Limit	Frequency
15.24	0.13	1
20.262	6.853294118	92
2.38	13.57658824	78
25.057	20.29988235	49
6.828	27.02317647	27
8.727	33.74647059	23
0.543	40.46976471	13
10.642	47.19305882	4
13.484	53.91635294	4
0.537	60.63964706	5
3.384	67.36294118	0
50.537	74.08623529	0
18.511	80.80952941	1
42.023	87.53282353	1
5.78	94.25611765	0
21.273	100.9794118	0
8.804	107.7027059	0
24.779	114.426	1
1.832		0

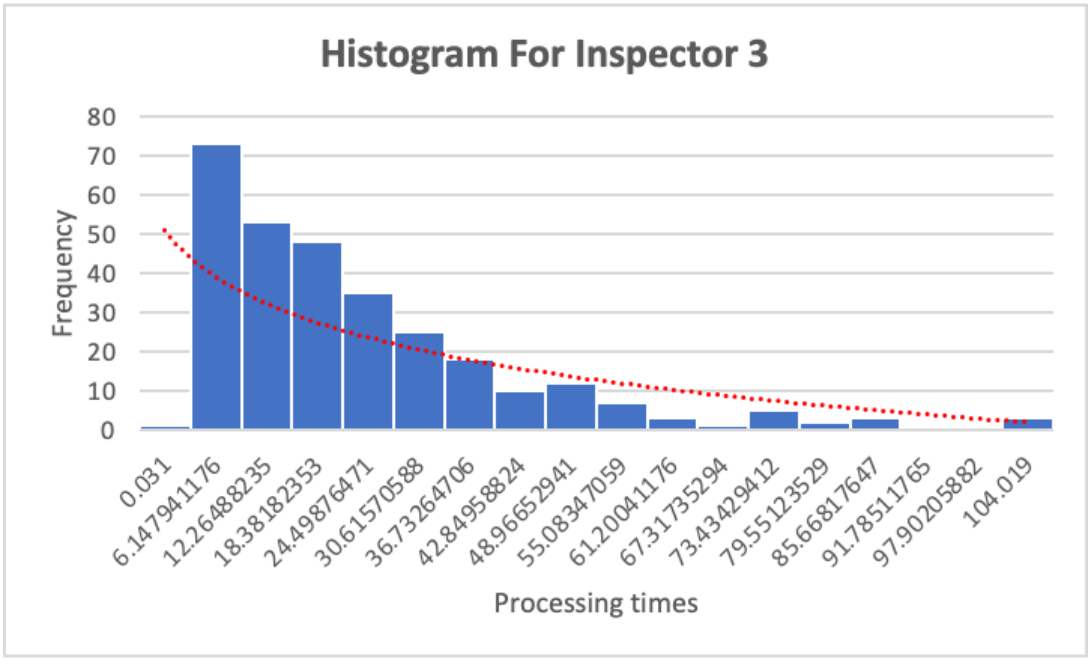


As you can see in the above histogram, it is possible to infer an exponential distribution from the family of distributions to be a suitable statistical model.

Histogram For inspector 3 (servinsp3.dat):

The histogram is prepared by placing the data in class intervals. The range of the data is from a minimum of 0.031 minutes to 104.019 minutes.The number of class intervals needed would be $n = \sqrt{300} = 17.32$. We shall round up to 18 in order to reach max value. Bin Width = (104.019 - 0.031) / 18 = 6.11.

Service Times	Bin Limit	Frequency
102.108	0.031	1
3.819	6.147941176	73
6.617	12.26488235	53
48.184	18.38182353	48
18.066	24.49876471	35
4.097	30.61570588	25
41.307	36.73264706	18
22.029	42.84958824	10
0.302	48.96652941	12
38.12	55.08347059	7
8.254	61.20041176	3
35.57	67.31735294	1
1.608	73.43429412	5
15.311	79.55123529	2
30.442	85.66817647	3
19.292	91.78511765	0
39.723	97.90205882	0
8.781	104.019	3

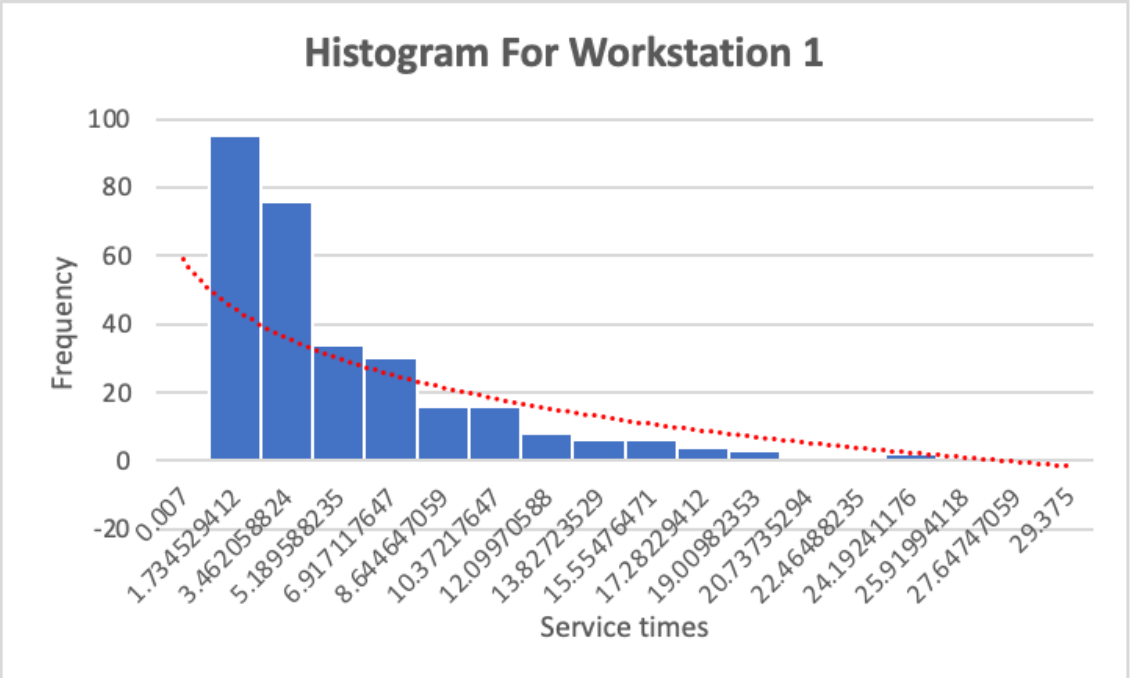


As you can see in the above histogram, it is possible to infer an exponential distibtuion from the family of distributions to be a suitable statistical model.

Histogram For Workstation 1 (ws1.dat):

The histogram is prepared by placing the data in class intervals. The range of the data is from a minimum of 0.007 minutes to 29.375 minutes.The number of class intervals needed would be n = $\sqrt{300}$ = 17.32. We shall round up to 18 in order to reach max value. Bin Width = (29.375 - 0.007) / 18 = 1.7275.

Service Times	Bin Limit	Frequency
0.85	0.007	1
0.476	1.734529412	95
3.016	3.462058824	76
7.33	5.189588235	34
0.956	6.917117647	30
4.197	8.644647059	16
0.951	10.37217647	16
0.242	12.09970588	8
3.259	13.82723529	6
3.342	15.55476471	6
10.956	17.28229412	4
7.002	19.00982353	3
1.746	20.73735294	0
0.289	22.46488235	1
3.514	24.19241176	2
2.495	25.91994118	0
1.823	27.64747059	1
6.069	29.375	0

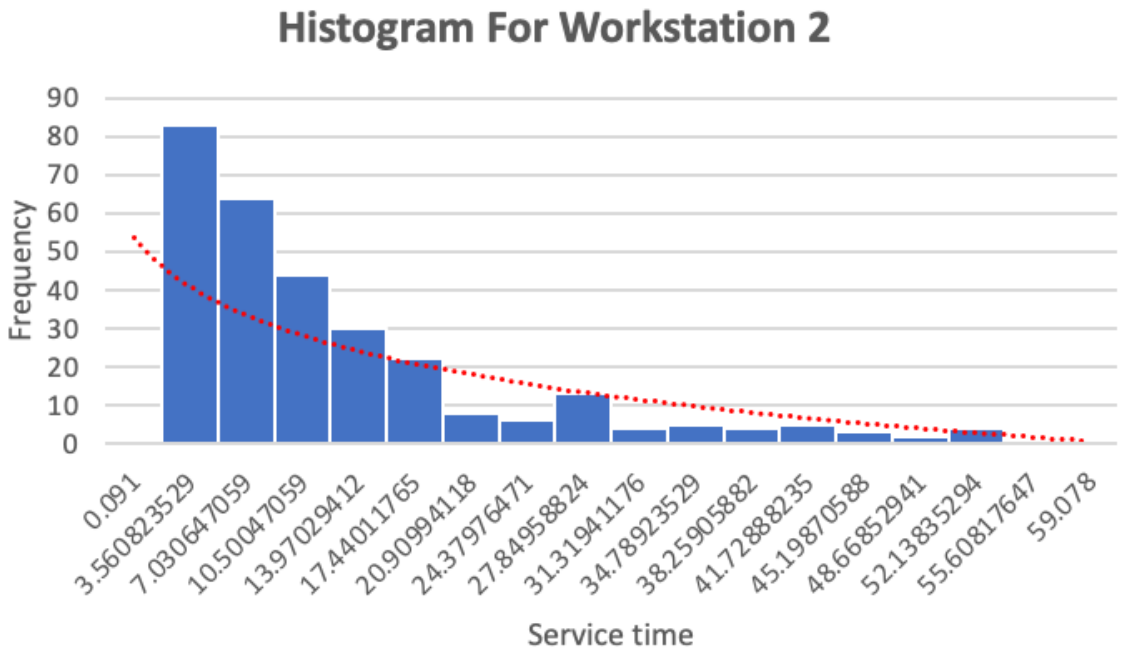


As you can see in the above histogram, it is possible to infer an exponential distribution from the family of distributions to be a suitable statistical model.

Histogram For Workstation 2 (ws2.dat):

The histogram is prepared by placing the data in class intervals. The range of the data is from a minimum of 0.091 minutes to 59.078 minutes. The number of class intervals needed would be $n = \sqrt{300} = 17.32$. We shall round up to 18 in order to reach max value. Bin Width = (59.078 - 0.091) / 18 = 3.469.

Service Times	Bin Limit	Frequency
13.43	0.091	1
6.808	3.560823529	83
0.32	7.030647059	64
49.324	10.50047059	44
1.872	13.97029412	30
5.728	17.44011765	22
13.989	20.90994118	8
8.321	24.37976471	6
14.654	27.84958824	13
0.863	31.31941176	4
2.131	34.78923529	5
7.884	38.25905882	4
0.282	41.72888235	5
6.586	45.19870588	3
10.234	48.66852941	2
2.114	52.13835294	4
1.202	55.60817647	1
4.513	59.078	0

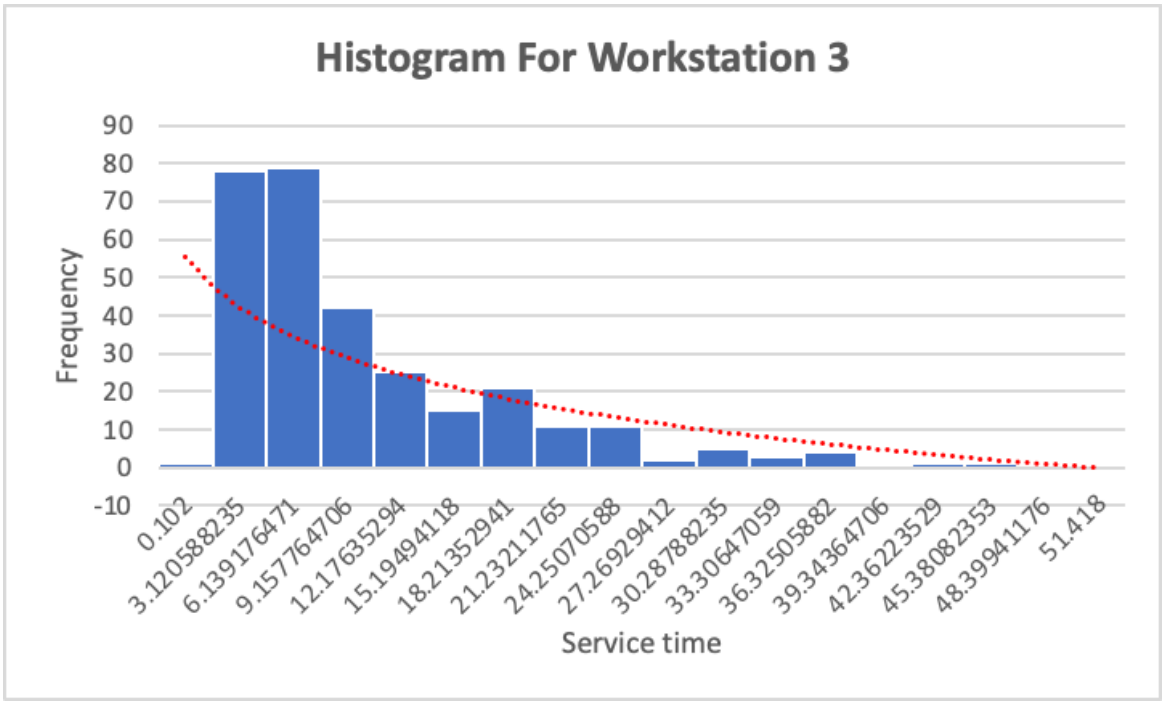


As you can see in the above histogram, it is possible to infer an exponential distribution from the family of distributions to be a suitable statistical model.

Histogram For Workstation 3 (ws3.dat):

The histogram is prepared by placing the data in class intervals. The range of the data is from a minimum of 0.102 minutes to 51.418 minutes. The number of class intervals needed would be n = $\sqrt{300}$ = 17.32. We shall round up to 18 in order to reach max value. Bin Width = (51.418 - 0.102) / 18 = 3.01.

Service Times	Bin Limit	Frequency
23.858	0.102	1
9.536	3.120588235	78
2.397	6.139176471	79
7.597	9.157764706	42
13.63	12.17635294	25
4.384	15.19494118	15
3.936	18.21352941	21
3.637	21.23211765	11
9.539	24.25070588	11
34.879	27.26929412	2
1.801	30.28788235	5
0.844	33.30647059	3
6.604	36.32505882	4
10.283	39.34364706	0
15.847	42.36223529	1
6.015	45.38082353	1
0.102	48.39941176	0
45.125	51.418	0



As you can see in the above histogram, it is possible to infer an exponential distribution from the family of distributions to be a suitable statistical model.

When selecting the exponential distribution as our statistical assumption we are not only looking at the shape of the histogram, but the physical characteristics of the process as well. And the context of the data we are estimating the distribution of. An input model is an approximation of reality, so the goal is to obtain a close approximation that produces reliable results. The exponential distribution is for continuous random variables, and models the time between independent random events. This is the case with the service times for the inspector and the workstation. As we have seen when creating a histogram for each of the data sets, the service times were high variable (i.e. scaling from low service times to very high service times). The exponential is a highly variable distribution. We also must take into account that the service times for the inspector and workstation are assumed to be memoryless. Knowing how much time a service was passed gives no new information on how much additional time will pass before the service is complete.

5.2 Q-Q Plot

As we have seen with histograms, they are useful in identifying the shape of the distribution by inferring a pdf or pmf. However, a histogram is not useful for evaluating the distribution fit of the chosen distribution. Even if we efficiently group our data in well chosen class intervals, grouping them into intervals makes it hard to compare a histogram to a continuous probability density function.

The theory behind Q-Q plots is that if you take two sample quantiles of a random variable. Say X and Z, the plotted ordered values of x1,x2,x3..xn and z1,z2,z3..zn will reveal approximately a straight line on a scatter plot if both samples come from the same distribution. We use the cdf of the chosen distribution to generate a q-q plot, if the cdf is an appropriate fit, then the line will have a slope = 1. If not, then the points plotted in the plot will deviate from a straight line, hence suggesting that the chosen distribution is inappropriate.

We shall generate Q-Q plots for each data set, with the chosen distribution being the exponential distribution as we have assumed it to be a good statistical model based on the generated Histograms.

X is the random variable with CDF of the exponential distribution, the q-quantile of X is the value γ such that F(γ) = P (X<= γ) = q. $\gamma = F^{-1}(q)$.

We shall order the service/processing times x1,x2,x3 .. x300 from smallest to largest (i.e. ascending order) and denote these as γ1, γ2, … γ300 where j = 1,2, … 300.

The Q-Q plot is based on the knowledge that γj is an estimate of the (j- ½)/n quantile of X. γj is approximately F^{-1}((j- ½)/n).

We must now calculate the inverse CDF of the exponential model to generate the appropriate values:

F(x) = 1 - $e^{-\lambda x}$, x >= 0 - CDF of Exponential Distribution

Performing the inverse of F(x):
y = 1 - $e^{-\lambda x}$

$e^{-\lambda x}$ = 1 - y

ln($e^{-\lambda x}$) = ln(1-y)

$-\ \lambda x =$ ln(1-y)

x = $\frac{-ln(1-y)}{\lambda}$

F^{-1}(x) = $\frac{-ln(1-x)}{\lambda}$

F^{-1}((j- 0.5)/n) = $\frac{-ln(1-(j-0.5)/n))}{\lambda}$

Estimating Parameters:

The next step is to estimate the parameters for the hypothesised exponential distribution. A good parameter for the exponential distribution is λ. To calculate the estimator for the parameter, $\hat{\lambda} = 1/\overline{X}$, where $\overline{X}$ is the sample mean. Since we have raw data available, $\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n}$

We can now plot γj vs F^{-1}((j- 0.5)/n) for each inspector and workstation using the estimator.

Q-Q Plot For inspector 1 (servinsp1.dat):

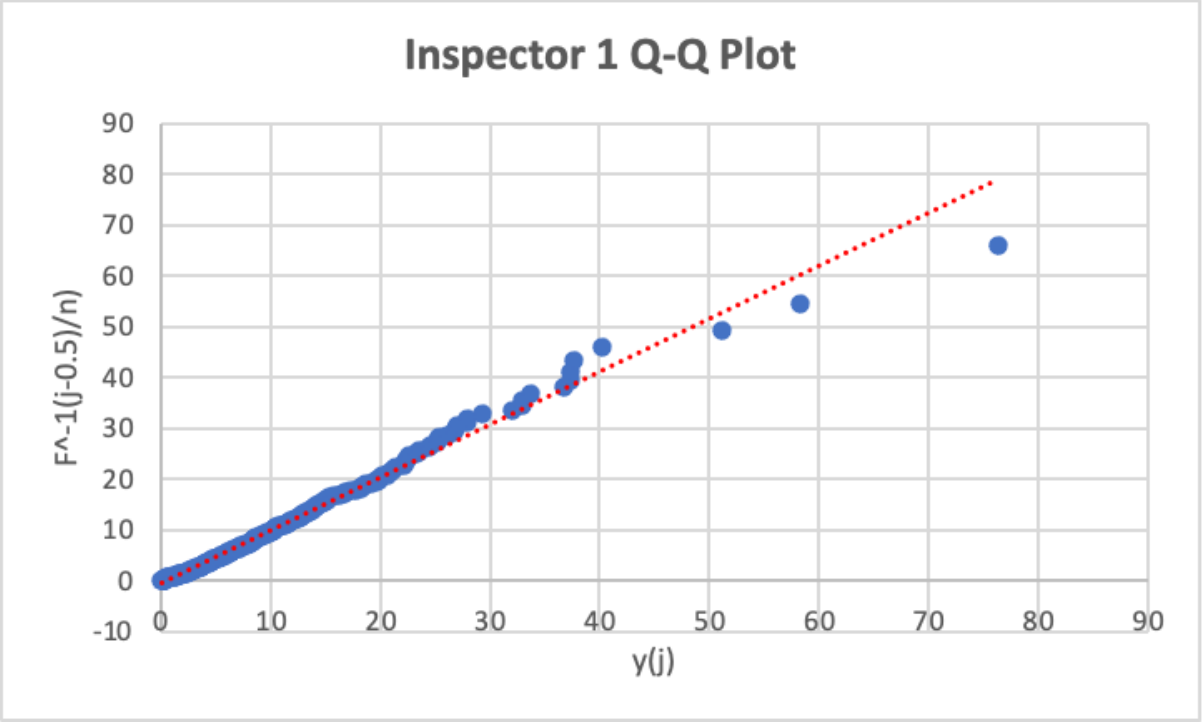
$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{300} = 10.35791$$

$$\widehat{\lambda} = 1/\overline{X} = 1/\ 10.35791 = 0.09654457318$$

Service Times	$F^{-1}(x)$
0.087	0.017196465
0.1	0.051675689
0.114	0.086270615
0.121	0.120982023
0.2	0.155810699
0.266	0.190757439
0.334	0.225823045
0.351	0.26100833
0.358	0.296314112
0.362	0.33174122
0.387	0.367290491
0.418	0.40296277
0.475	0.438758912
0.486	0.474679778
0.496	0.510726243
0.502	0.546899186
0.556	0.5831995
0.6	0.619628083
0.651	0.656185846

A sample calculation for γ1 = 0.087, j = 1, $\widehat{\lambda}$ = 0. 09654457318:

$$F^{-1}((1- 0.5)/300) = \frac{-ln(1-(1-0.5)/300))}{0.09654457318} = 0.01727758532$$



By using the excel slope function, the slope of the above plot is 1.040114649. The plot is approximately a straight line with a slope close to 1. The points do not deviate from the straight lines except for the last three points located to the right of the plot. In the evaluation of linearity of a q-q plot, the linearity in the middle of the plot is more important than the linearity at the extremes. In this case, the three large points that deviate further from the straight line than the other points are to be expected as they have much higher variances. It is also the case that if one point is below the straight line then it is likely that the next point will also be below the line (i.e. the ordered values are dependent upon another). We can see this in the points γj , j >= 50. Another important note regarding linearity of a q-q plot is that the observed points will never fall exactly on the straight line. But it still gives a linear line that is suitable enough to decide whether or not to reject the hypothesised model or not.

As you can see, comparing the Q-Q plot for Inspector 1 to the Histogram of Inspector 1, it is easier to tell whether the data are well represented by an exponential distribution. The straight line is clear in the Q-Q plot and it supports the hypothesis of an exponential distribution.

Q-Q Plot For inspector 2 (servinsp2.dat):

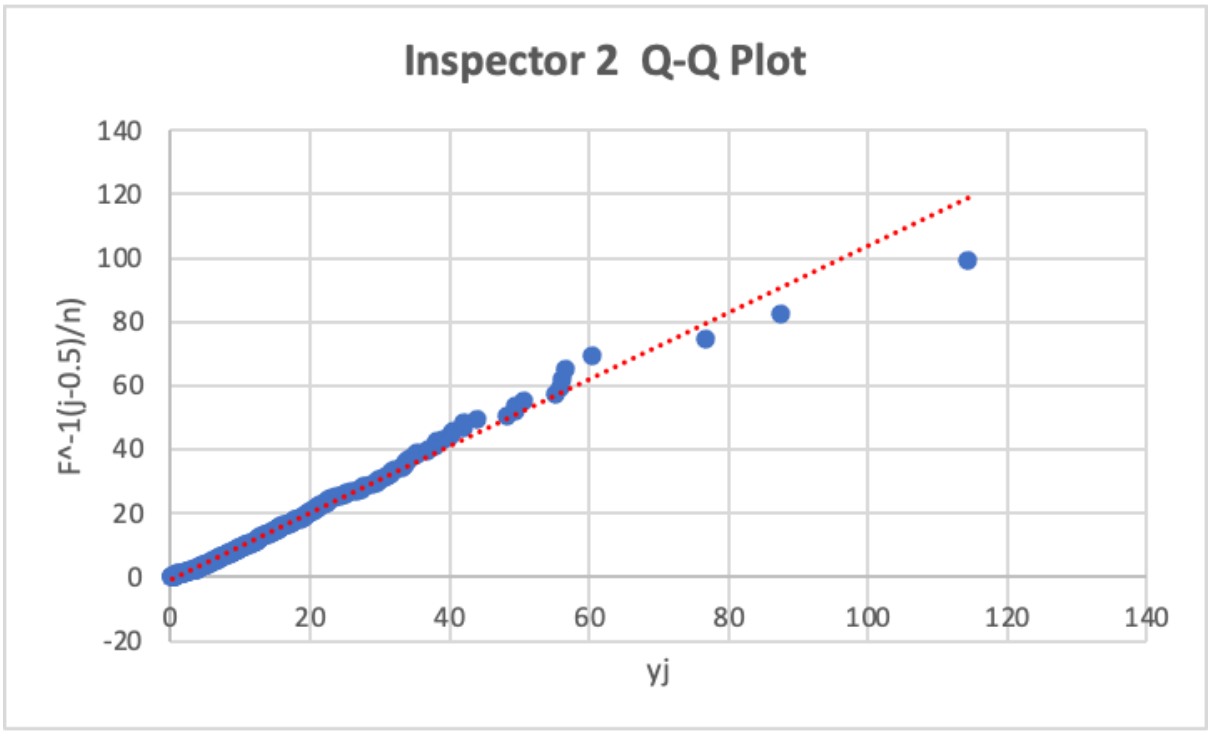
$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{300} = 15.53690333$$

$$\widehat{\lambda} = 1/\overline{X} = 1/\ 15.53690333 = 0.06436289$$

yj	$F^{-1}(x)$
0.13	0.02591644
0.151	0.07787938
0.171	0.13001669
0.181	0.18232954
0.299	0.23481913
0.4	0.28748665
0.501	0.34033331
0.526	0.39336034
0.537	0.44656896
0.543	0.49996043
0.58	0.55353601
0.628	0.60729698
0.712	0.66124462
0.729	0.71538022
0.745	0.76970511
0.752	0.82422062
0.834	0.87892808

A sample calculation for γ2 = 0.151, j = 2, $\widehat{\lambda}$ = 0. 06436289 :

$$F^{-1}((1- 0.5)/300) = \frac{-ln(1 - (2- 0.5)/300))}{0.06436289} = 0.0778793$$



The slope of the data points is 1.0450245. As with inspector 1, the line is approximately linear with a slope close to 1. As with inspector 1, we can see the plot is not a perfect line. The three larger points deviate from the straight line due to their high variances. But because the middle observed values are close to the straight line and the slope is close to 1. We shall not reject the hypothesis of the exponential distribution.

Q-Q Plot For inspector 3 (servinsp3.dat):

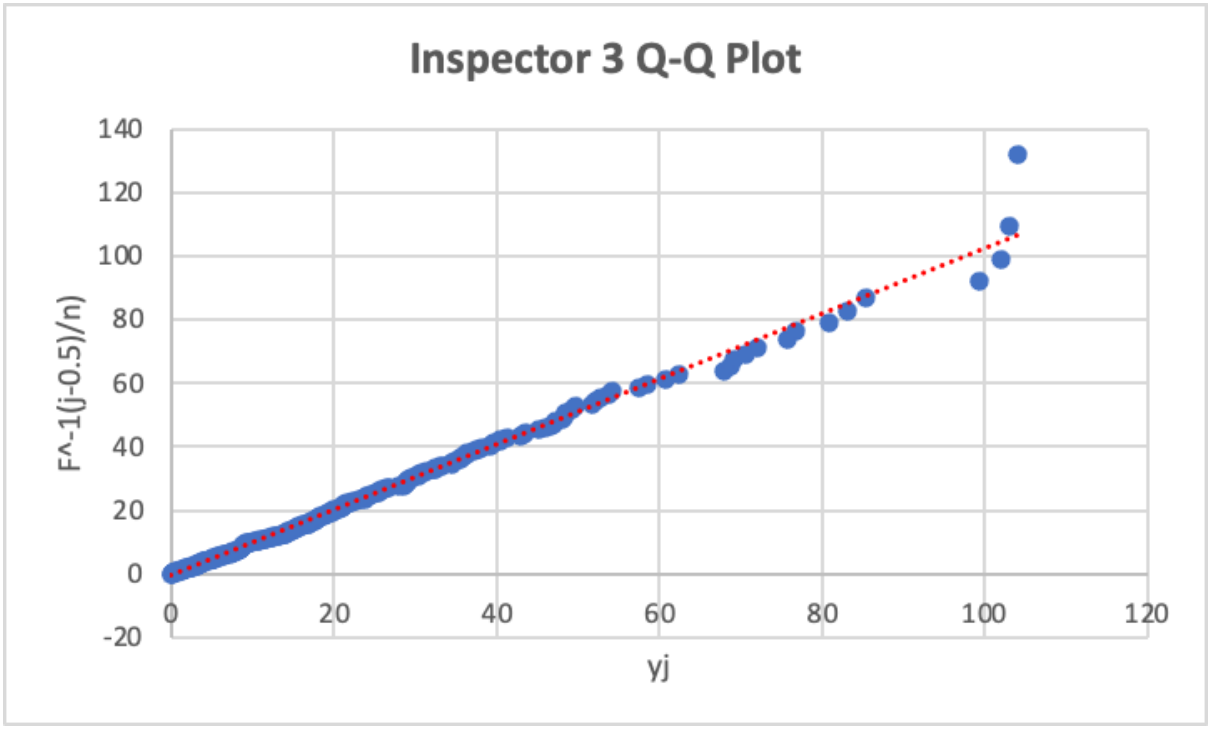
$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{300} = 20.63275667$$

$$\hat{\lambda} = 1/\overline{X} = 1/\ 20.63275667 = 0.04846662111$$

yj	$F^{-1}(x)$
0.031	0.03441662
0.137	0.10342256
0.214	0.17266006
0.256	0.24213069
0.302	0.31183601
0.327	0.38177763
0.34	0.45195714
0.384	0.52237617
0.577	0.59303637
0.68	0.66393938
0.686	0.73508689
0.844	0.80648058
0.903	0.87812217
0.941	0.95001338
1.053	1.02215596
1.135	1.09455167
1.216	1.1672023
1.299	1.24010964

A sample calculation for γ3 = 0.214, j = 2, $\hat{\lambda}$ = 0. 04846662111 :

$F^{-1}((1- 0.5)/300) = \frac{-ln(1-(3-0.5)/300))}{0.04846662111} = 0.1726600592$



The slope of the data points is 1.02821234. The line is approximately linear with a slope close to 1. The linearity in this plot is closer a straight line then inspector 1 and 2. We also accept the greater discrepancies at the extremes. We shall not reject the hypothesis of the exponential distribution.

Q-Q Plot For Workstation 1 (ws1.dat):

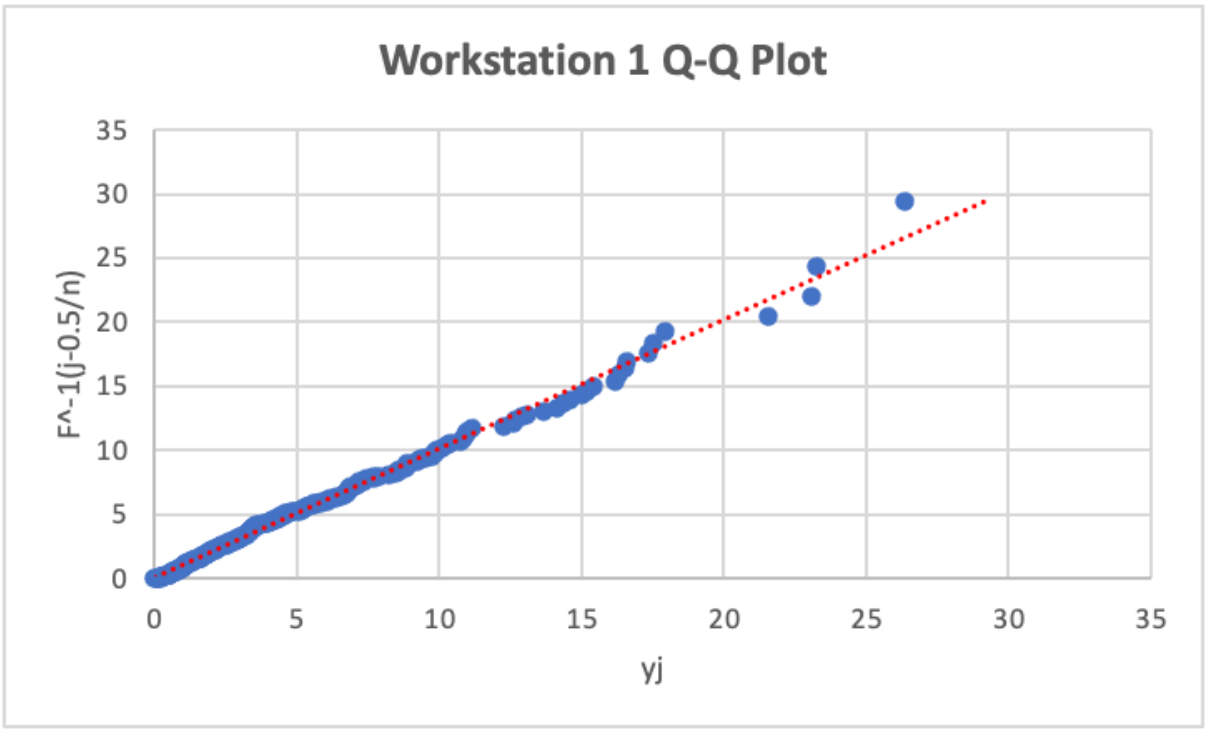
$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{300} = 4.604416667$$

$$\hat{\lambda} = 1/\overline{X} = 1/\ 4.604416667 = 0.21718277739$$

yj	$F^{-1}(x)$
0.007	0.00768043
0.069	0.02307983
0.071	0.03853091
0.11	0.05403401
0.197	0.06958949
0.216	0.08519769
0.242	0.10085899
0.25	0.11657374
0.255	0.1323423
0.282	0.14816506
0.288	0.16404237
0.289	0.17997462
0.33	0.1959622
0.394	0.21200548
0.444	0.22810485
0.453	0.24426072
0.476	0.26047347
0.479	0.27674351

A sample calculation for γ4 = 0.11, j = 4, $\hat{\lambda} = 0.21718277739$:

$$F^{-1}((1- 0.5)/300) = \frac{-ln(1 - (4- 0.5)/300))}{0.21718277739} = 0.05403400932$$



The slope of the data points is 0.95970623. The line is approximately linear with a slope close to 1. The slope is now less then 1 then greater then 1 as we have seen with the inspector times. We can accept the greater discrepancies at the extremes. We shall not reject the hypothesis of the exponential distribution.

Q-Q Plot For Workstation 2 (ws2.dat):

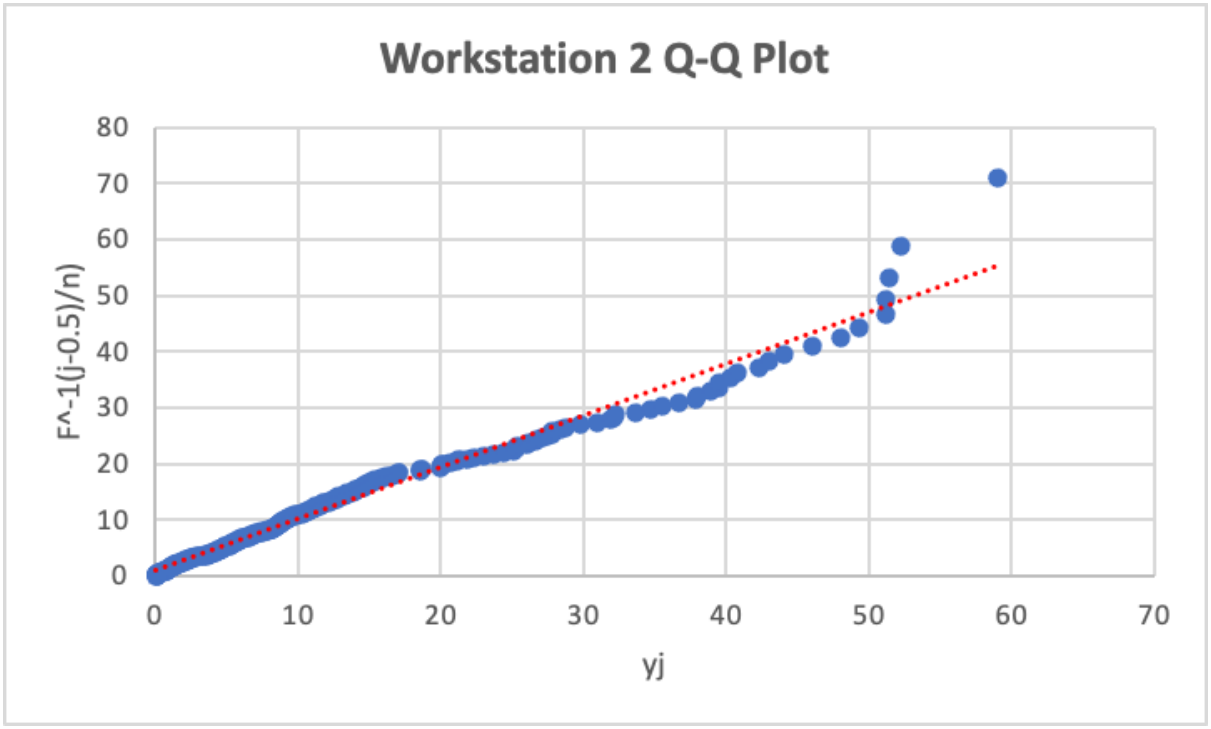
$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{300} = 11.09260667$$

$$\hat{\lambda} = 1/\overline{X} = 1/\ 11.09260667 = 0.090150136$$

yj	$F^{-1}(x)$
0.091	0.0185031
0.092	0.05560215
0.106	0.0928257
0.106	0.13017458
0.116	0.16764964
0.143	0.20525173
0.149	0.24298172
0.15	0.28084049
0.203	0.3188289
0.257	0.35694786
0.281	0.39519827
0.282	0.43358103
0.314	0.47209706
0.32	0.5107473
0.368	0.54953268

A sample calculation for γ5 = 0.116, j = 5, $\hat{\lambda} = 0.090150136$:

$F^{-1}((1- 0.5)/300) = \frac{-ln(1-(5-0.5)/300))}{0.090150136} = 0.16764964$



The slope of the data points is 0.92118067. The line is approximately linear with a slope close to 1. We have more data points below the straight line then above. We can accept the greater discrepancies at the extremes. We shall not reject the hypothesis of the exponential distribution.

Q-Q Plot For Workstation 3 (ws3.dat):

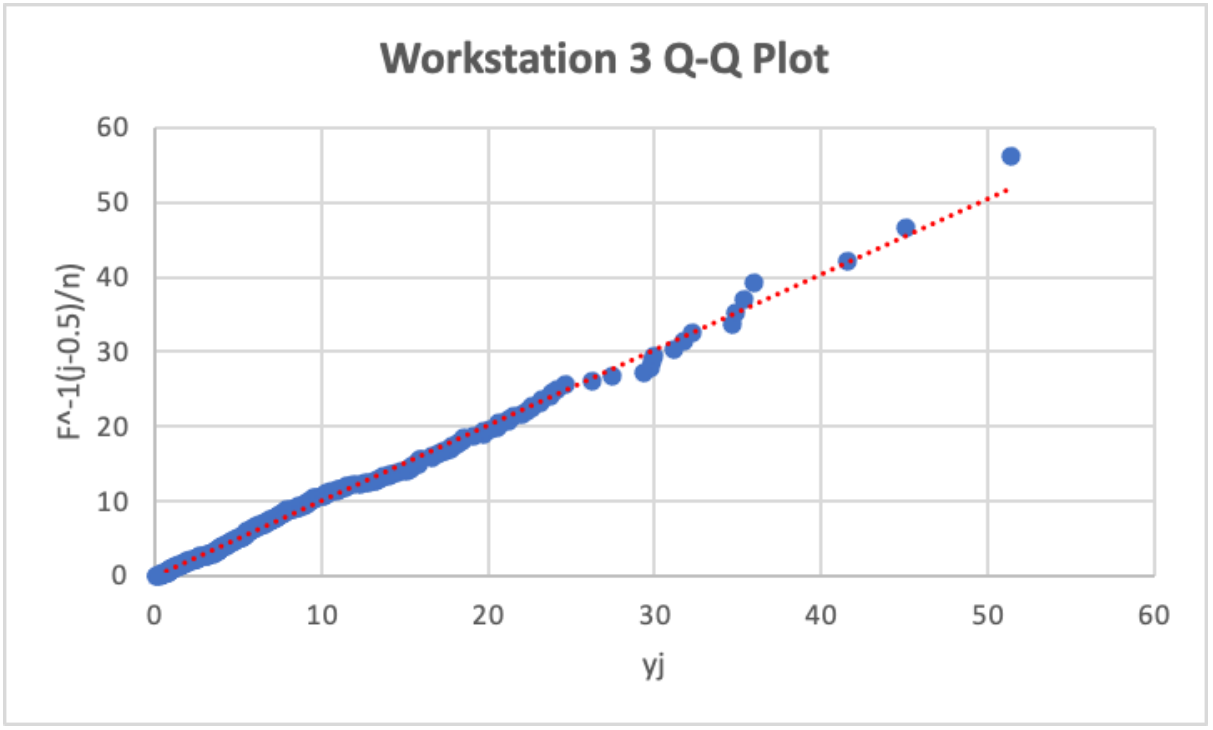
$$\overline{X} = \frac{\sum_{i=1}^{n} X_i}{300} = 8.79558$$

$$\hat{\lambda} = 1/\overline{X} = 1/\ 8.79558 = 0.11369346876$$

yj	$F^{-1}(x)$
0.102	0.01467153
0.138	0.04408821
0.271	0.07360361
0.338	0.10321838
0.414	0.13293321
0.436	0.16274876
0.45	0.19266573
0.494	0.22268481
0.505	0.25280668
0.507	0.28303207
0.52	0.31336169
0.638	0.34379625
0.69	0.37433649
0.708	0.40498314
0.73	0.43573695
0.736	0.46659867
0.775	0.49756905
0.794	0.52864887
0.808	0.55983891

A sample calculation for γ6 = 0.436, j = 6, $\hat{\lambda} = 0.11369346876$:

$$F^{-1}((1- 0.5)/300) = \frac{-ln(1-(6-0.5)/300))}{0.11369346876} = 0.16274876443$$



The slope of the data points is 1.00810407. The line is approximately linear with a slope close to 1. There are not great discrepancies at the extremes and the linearity at the extremes fall closer to the straight line then the other Workstation plots. Therefore, we shall not reject the hypothesis of the exponential distribution.

5.3 Chi-square goodness-of-fit test

Finally, we are now able to perform the CHI-square test for goodness of fit.

inspector 1 (servinsp1.dat):

$\widehat{\lambda} = 1/\overline{X} = 1/\ 10.35791 = 0.0965$
H_o: The random variable is exponentially distributed
H_1: The random variable is not exponentially distributed
Since n=300, using the recommended number of class intervals (k) table;
$k = \sqrt{n}\ to\ \frac{n}{5} = \sqrt{300}\ to\ \frac{300}{5} =\ 17.32\ to\ 60$
Rounding down to 17 as a smaller interval results in a closer look at the variance in data, let $k\ =\ 17$ (for all tests).
Then, each interval has a probability of $p\ = \frac{1}{k} = \frac{1}{17} =\ 0.0588$ (for all tests). The end points of the intervals are derived from the CDF of the exponential distribution as follows;

$F(ai) = 1 - e{-}\lambda ai$

$ip = 1{-}e^{-\lambda ai}$

$-e^{-\lambda ai} = 1- ip$

$\ln(-e^{-\lambda ai}) = \ln(1-ip)$

$-\lambda a_i = \ln(1-ip)$

$a_i = \frac{-1}{\lambda} ln(1\ -\ ip)$

Now $a_i = \frac{-1}{\lambda} ln(1\ -\ ip)$ where i is the ith interval. We will use this for all the following tests as seen below:
$a_1 = \frac{-1}{0.0965} ln(1\ -\ (1\ \times\ 0.0588))\ =\ 0.6277$ for the first interval and so on.
(Note: the observed frequency is found by =COUNTIFS(B2:B301, ">=0.6277",B2:B301, "<1.2959") on excel where B2:B301 is the selected data.)

Class Interval	Observed Frequency O_i	Expected Frequency (np) E_i	$\frac{(O_i - E_i)^2}{E_i}$
[0, 0.6277)	18	17.64	0.007347
[0.6277, 1.2959)	10	17.64	3.3089
[1.2959, 2.0102)	11	17.64	2.4994
[2.0102, 2.7774)	14	17.64	0.7511

[2.7774, 3.6060)	17	17.64	0.0232
[3.6060, 4.5068)	22	17.64	1.0776
[4.5068, 5.4933)	22	17.64	1.0776
[5.4933, 6.5839)	20	17.64	0.3157
[6.5839, 7.8029)	17	17.64	0.0232
[7.8029, 9.1847)	23	17.64	1.6287
[9.1847, 10.7798	20	17.64	0.3157
[10.7798, 12.6659	16	17.64	0.1525
[12.6659, 14.9737	25	17.64	3.0708
[14.9737, 17.9476)	13	17.64	1.2205
[17.9476, 22.1357)	19	17.64	0.1049
[22.1357, 29.2803)	20	17.64	0.3157
[29.2803, ∞)	13	17.64	1.2205
SUM	= 300	= 300	=17.1133

$\chi_o^2 = 17.11$

Degree of freedom is given by $k - s - 1 = 17 - 1 - 1 = 15$ (for all tests). Note s=1 because the number of parameters is constant.

At $\alpha = 0.05$ so $\chi^2_{0.05,15} = 25$ (using the table from the textbook, seen below pg.550)

Thus, since $\chi_o^2 < \chi^2_{o.05,15}$ then we fail to reject the null hypothesis which means that this distribution follows an exponential distribution at a significance level of $\alpha = 0.05$.

inspector 2 (servinsp2.dat):

$\hat{\lambda} = 1/\overline{X} = 1/\ 15.53690333 = 0.0644$

H_o: The random variable is exponentially distributed

H_1: The random variable is not exponentially distributed

$a_1 = \frac{-1}{0.0644} ln(1 - (1 \times 0.0588)) = 0.9410$ for the first interval and so on.

Class Interval	Observed Frequency O_i	Expected Frequency (np) E_i	$\frac{(O_i - E_i)^2}{E_i}$
[0, 0.9410)	18	17.64	0.0073

[0.9410, 1.9427)	10	17.64	3.3090
[1.9427,3.0135)	11	17.64	2.4994
[3.0135,4.1637)	14	17.64	0.7511
[4.1637, 5.4059)	17	17.64	0.0232
[5.4059, 6.7562)	22	17.64	1.0776
[6.7562,8.2352)	22	17.64	1.0776
[8.2352,9.8701)	20	17.64	0.3157
[9.8701,11.6975)	16	17.64	0.1525
[11.6975,13.7691)	24	17.64	2.2931
[13.7691, 16.1603)	20	17.64	0.3157
[16.1603, 18.9878)	16	17.64	0.1525
[18.9878, 22.4475)	25	17.64	3.0708
[22.4475, 26.9058)	13	17.64	1.2205
[26.9058, 33.1843)	18	17.64	0.0073
[33.1843,43.8949)	21	17.64	0.64
[43.8949, ∞)	13	17.64	1.2205
SUM	= 300	= 300	χ_o^2 =18.1338

χ_o^2 =18.13

At $\alpha = 0.05$ so $\chi^2_{0.05,15} = 25$ (using the table from the textbook, seen below pg.550)

Thus, since $\chi_o^2 < \chi^2_{o.05,15}$ then we fail to reject the null hypothesis which means that this distribution follows an exponential distribution at a significance level of $\alpha = 0.05$.

inspector 3 (servinsp3.dat):

$\hat{\lambda} = 1/\overline{X} = 1/ 20.63275667 = 0.0485$

H_o: The random variable is exponentially distributed
H_1: The random variable is not exponentially distributed

Since n=300, using the recommended number of class intervals (k) table;

$$a_1 = \frac{-1}{0.0485} ln(1 - (1 \times 0.0588)) = 1.2495$$

Class Interval	Observed Frequency O_i	Expected Frequency (np) E_i	$\frac{(O_i - E_i)^2}{E_i}$
[0, 1.2495)	17	17.64	0.0232
[1.2495, 2.5796)	15 =COUNTIFS(B2:B301, ">=0.6277",B2:B301, "<1.2959")	17.64	0.3951
[2.5796, 4.0014)	19	17.64	0.1049
[4.0014, 5.5287)	15	17.64	0.3951
[5.5287, 7.1781)	15	17.64	0.3951
[7.1781, 8.9711)	26	17.64	3.9620
[8.9711,10.9350)	15	17.64	0.3951
[10.9350, 13.1058)	10	17.64	3.3089
[13.1058, 15.5324)	21	17.64	0.64
[15.5324,18.2831)	22	17.64	1.0776
[18.2831, 21.4582)	22	17.64	1.0776
[21.4582, 25.2127)	15	17.64	0.3951
[25.2127, 29.8066)	20	17.64	0.3157
[29.8066, 35.7265)	18	17.64	0.0073
[35.7265, 44.0633)	16	17.64	0.1525
[44.0633, 58.2852)	17	17.64	0.0232
[58.2852, ∞)	17 =COUNTIFS(B2:B301, ">=29.2803)	17.64	0.0232
SUM	= 300	= 300	$\chi_o^2 = 12.69$

$\chi_o^2 = 12.69$

At $\alpha = 0.05$ so $\chi^2_{0.05,15} = 25$ (using the table from the textbook, seen below pg.550)

Thus, since $\chi_o^2 < \chi_{o.05,15}^2$ then we fail to reject the null hypothesis which means that this distribution follows an exponential distribution at a significance level of $\alpha = 0.05$.

Workstation 1 (ws1.dat):

$\hat{\lambda} = 1/\overline{X} = 1/\ 4.604416667 = 0.2172$

H_o: The random variable is exponentially distributed
H_1: The random variable is not exponentially distributed

$a_1 = \frac{-1}{0.2172} ln(1 - (1 \times 0.0588)) = 0.2790$

Class Interval	Observed Frequency O_i	Expected Frequency (np) E_i	$\frac{(O_i - E_i)^2}{E_i}$
[0, 0.2790)	9	17.64	4.2318
[0.2790, 0.5760)	14 =COUNTIFS(B2:B301, ">=0.6277",B2:B301, "<1.2959")	17.64	0.7511
[0.5760, 0.8935)	23	17.64	1.6287
[0.8935, 1.2345)	26	17.64	3.9620
[1.2345, 1.6029)	15	17.64	0.3951
[1.6029, 2.0032)	25	17.64	3.0708
[2.0032, 2.4418)	17	17.64	0.0232
[2.4418, 2.9265)	18	17.64	0.0073
[2.9265, 3.4683)	25	17.64	3.0708
[3.4683, 4.0826)	12	17.64	1.8033
[4.0826, 4.7915)	17	17.64	0.0232
[4.7915, 5.6299)	14	17.64	0.7511
[5.6299, 6.6557)	11	17.64	2.4994
[6.6557, 7.9776)	21	17.64	0.64
[7.9776, 9.8392)	16	17.64	0.1525

[9.8392, 13.0149)	17	17.64	0.0232
[13.0149, ∞)	20 =COUNTIFS(B2:B301, ">=29.2803")	17.64	0.3157
SUM	= 300	= 300	χ_o^2 =23.35

$\chi_o^2 = 23.35$

At $\alpha = 0.05$ so $\chi^2_{0.05,15} = 25$ (using the table from the textbook, seen below pg.550)

Thus, since $\chi_o^2 < \chi^2_{o.05,15}$ then we fail to reject the null hypothesis which means that this distribution follows an exponential distribution at a significance level of $\alpha = 0.05$.

Workstation 2 (ws2.dat):

$\hat{\lambda} = 1/\overline{X} = 1/\ 11.09260667 = 0.0902$

H_o: The random variable is exponentially distributed
H_1: The random variable is not exponentially distributed

$$a_1 = \frac{-1}{0.0902} ln(1 - (1 \times 0.0588)) = 0.6718$$

Class Interval	Observed Frequency O_i	Expected Frequency (np) E_i	$\frac{(O_i - E_i)^2}{E_i}$
[0, 0.6718)	20	17.64	0.3157
[0.6718, 1.3870)	28 =COUNTIFS(B2:B301, ">=0.6277",B2:B301, "<1.2959")	17.64	6.0844
[1.3870, 2.1516)	18	17.64	0.0073
[2.1516, 2.9727)	14	17.64	0.7511
[2.9727, 3.8596)	10	17.64	3.3089
[3.8596, 4.8237)	21	17.64	0.64
[4.8237, 5.8797)	20	17.64	0.3157

[5.8797, 7.0469)	17	17.64	0.0232
[7.0469, 8.3517)	14	17.64	0.7511
[8.3517, 9.8307)	25	17.64	3.0708
[9.8307, 11.5379)	16	17.64	0.1525
[11.5379,13.5567)	18	17.64	0.0073
[13.5567, 16.0268)	18	17.64	0.0073
[16.0268, 19.2099)	8	17.64	5.2681
[19.2099, 23.6926)	11	17.64	2.4994
[23.6926, 31.3396)	17	17.64	0.0232
[31.3396, ∞)	25 =COUNTIFS(B2:B301, ">=29.2803)	17.64	3.0708
SUM	= 300	= 300	$\chi_o^2 = 26.30$

$\chi_o^2 = 26.30$

At $\alpha = 0.05$ so $\chi^2_{0.05,15} = 25$ (using the table from the textbook, seen below pg.550)

Thus, since $\chi_o^2 < \chi^2_{o.05,15}$ then w reject the null hypothesis which means that this distribution does not follow an exponential distribution at a significance level of $\alpha = 0.05$. However, at a $\alpha = 0.025$, $\chi^2_{0.025,15} = 27.5$ then $\chi_o^2 < \chi^2_{o.05,15}$ so we fail to reject the null hypothesis which means that this distribution follows an exponential distribution as a significant level of $\alpha = 0.025$.

Workstation 3 (ws3.dat):

$\hat{\lambda} = 1/\overline{X} = 1/\ 8.79558 = 0.1137$

H_o: The random variable is exponentially distributed
H_1: The random variable is not exponentially distributed

$a_1 = \frac{-1}{0.1137} ln(1 - (1 \times 0.0588)) = 0.5330$

Class Interval	Observed Frequency O_i	Expected Frequency (np) E_i	$\frac{(O_i-E_i)^2}{E_i}$
[0, 0.5330)	11	17.64	2.4994
[0.5330, 1.1004)	20	17.64	0.3157
[1.1004,1.7069)	18	17.64	0.0073
[1.7069, 2.3583)	15	17.64	0.3951
[2.3583, 3.0619)	14	17.64	0.7511
[3.0619, 3.8267)	20	17.64	0.3157
[3.8267, 4.6645)	24	17.64	2.2931
[4.6645, 5.5904)	25	17.64	3.0708
[5.5904, 6.6255)	18	17.64	0.0073
[6.6255, 7.7989)	21	17.64	0.64
[7.7989, 9.1532)	14	17.64	0.7511
[9.1532, 10.7548)	17	17.64	0.0232
[10.7548,12.7143)	10	17.64	3.3089
[12.7143,15.2395)	13	17.64	1.2205
[15.2395, 18.7957)	24	17.64	2.2931
[18.7957,24.8622)	20	17.64	0.3157
[24.8622, ∞)	16	17.64	0.1525
SUM	= 300	= 300	$\chi_o^2 = 18.36$

$\chi_o^2 = 18.36$

At $\alpha = 0.05$ so $\chi^2_{0.05,15} = 25$ (using the table from the textbook, seen below pg.550)

Thus, since $\chi_o^2 < \chi^2_{o.05,15}$ then we fail to reject the null hypothesis which means that this distribution follows an exponential distribution at a significance level of $\alpha = 0.05$.

6. Input Generation

6.1 Linear Congruential Generator

Previously, random digits assignments were created using the cumulative probability of the workstation and inspection times and used random numbers from a table to generate these service times. We must now generate these random numbers instead of using a table.

We first needed to model our inputs to accurately reflect the manufacturing facility system. Determining an appropriate statistical distribution allows us to better characterise our input variables that are used to affect the behaviour and performance of our system. Once we have come up with an appropriate statistical distribution we can begin exploring the behaviour of our system under various conditions.

A sequence of random number must have two important properties. They must be uniform and independent. The sequence of random numbers must be drawn uniformly distributed between the interval [0,1] and each sample must be independent. That is there is no dependance between a random number and a subsequent random number. Uniformity is important because it ensures that the system being modelled is not biassed towards a particular value. That is that every generated random value has an equal probability of occurring. Independence is important because it means that the occurrence of one event has no effect on the other. This ensures that the system being model is not skewed or biassed towards any particular pattern of dependencies in the data. Overall, the two properties ensures the reliability and accuracy of the results of the simulation.

The method we shall use for implementing our random number generator is the Linear Congruential Method. LCM produces a sequence of random integers X1,X2 , . . . between zero and $m - 1$ by following a recursive relationship defined as:

$$X_{i+1} = (aX_i + c) mod\ m \quad , \qquad i = 1,2,3\ldots$$

a is the multiplier = 1 + 4k, where k is a integer value
X_0 is the seed value
m is the modulus = 2^b, b >0, longest possible period P = m
c is the increment

After calculating the random number using the equation above, the random numbers are then generated between [0,1]:

$$Ri = Xi/m$$

Properties of a Generator:

Generators must avoid cycling and have the largest possible period. Cycling occurs when generated numbers reoccur in the same sequence again. LCM generates random numbers with cycles because it generates random integers within the restrictive range of [0,m-1] and it generates random integers with a recursive formula (maximum period property). The random numbers generated are also discrete and not continuous on [0,1] because each Xi is an integer in the interval [0,m-1] and Ri = Xi/m (maximum density property). Recall the characteristic of uniformity on random numbers is critical to the accuracy of overall simulation results.

We strive to achieve maximum density and to avoid cycling by having a maximum/large period. To achieve the values for a,m,c and X_0 are critical.

We have 300 data points in each of the 6 data files. Therefore, since m represents the longest possible period of generated random numbers (i.e. m defines the largest range of the generated random numbers). m must be at least 300 to avoid cycling.

$$m = 2^b$$
$$300 = 2^b \quad \text{- let m = 300}$$
$$log_2(300) = log_2(2^b) = b$$
$$b = 8.2288$$

Therefore, b = 9 and **m** = 512

X_0 should be a number from [0,511]. $X_0 = 50$

k = 3 as it is efficient for k to be a prime number to avoid cycling. Therefore, **a** = 13. 13 is a prime number and a and m are relatively prime.

c must be relatively prime to m so **c** can be 9.

We shall generate 300 random numbers produced by our generator so we can perform test for uniformity and independence.

6.2 Kolmogorov - Smirnov Goodness of fit test - using Excel

The Kolmogorov-Smirnov test is used to test if the random numbers are uniform. This test measures the degree of agreement between the distribution of a sample of generated random numbers and the theoretical uniform distribution. It compares the continuous CDF F(x) of the uniform distribution to the empirical CDF and is based on the largest absolute difference between the two.

$H0$: $Ri \sim$ Uniform[0, 1]

$H1$: Ri !~ Uniform[0, 1]

In the excel spreadsheet below the first column ranks the random generated numbers in ascending order. D^+ is computed as ($\frac{i}{N}$ - Ri) and D^- is computed as (Ri - $\frac{i-1}{N}$). Where N = 300.

Random #s	D^+	D^-	D = max(D^-, D^+)
0.001953125	0.00138020833	0.001953125	0.03083333333
0.00390625	0.002760416667	0.000572916666	
0.005859375	0.004140625	-0.00080729166	
0.01171875	0.001614583333	0.00171875	
0.01953125	-0.00286458333	0.006197916667	
0.021484375	-0.001484375	0.004817708333	
0.02734375	-0.00401041666	0.00734375	
0.037109375	-0.01044270833	0.01377604167	
0.0390625	-0.0090625	0.01239583333	
0.041015625	-0.00768229166	0.011015625	
0.04296875	-0.00630208333	0.009635416667	
0.044921875	-0.004921875	0.008255208333	
0.052734375	-0.00940104166	0.012734375	
0.0546875	-0.00802083333	0.01135416667	
0.056640625	-0.006640625	0.009973958333	
0.05859375	-0.00526041666	0.00859375	
0.060546875	-0.00388020833	0.007213541667	
0.0625	-0.0025	0.005833333333	
0.06640625	-0.00307291666	0.00640625	
0.068359375	-0.00169270833	0.005026041667	
0.072265625	-0.002265625	0.005598958333	
0.07421875	-0.00088541666	0.00421875	
0.080078125	-0.00341145833	0.006744791667	
0.08203125	-0.00203125	0.005364583333	
0.083984375	-0.00065104166	0.003984375	
0.0859375	0.000729166666	0.002604166667	

As we can see the sample statistic D = max(D^+, D^-), which is 0.030833333

The critical value is $D_{0.05}$= 1.36/$\sqrt{300}$ = 0.0785

D < $D_{0.05}$Therefore, generated random numbers are uniform and we can conclude that there is no difference between the distribution of our generated numbers and the uniform distribution. We fail to reject the null hypothesis.

The AutoCorrelation test to test for Independence is done in code. The Kolomogorov - Smirnov Test is also done again in code and explained further in the next section.

7. Linear Congruential Generator and Testing - Implementation in Code

Kindly note that the code and outputs for all the functions discussed in this section are available in the "simulation.py" script located in the project folder. Running the script will display the results in the same order as presented here.

7.1 Random Numbers Generation

The Linear Congruential Method (LCM) mentioned above to generate random variables was implemented in python (can be seen in the figure below) and used to generate 300 samples using the a, c, and m constants calculated in the above subsection to determine the properties of the LCM generator.

```
"""
lcg_generate function takes four arguments: seed, a, c, and m
seed is the chosen starting or current value in the sequence
a, c, and m are the constants that determine the properties of the lcg
generator. This function/generator calculates the next random number in
the sequence and returns it
"""
def lcg_generate(self,seed, a, c, m):
    return (a * seed + c)% m

"""
generate_LCG_random_numbers takes five arguments: seed, a, c, m, and n
seed is the chosen starting value or current value in the sequence
a, c, and m are the constants that determine the properties of the lcg generator
n is the numnber of random numbers to generate
The function generates the n random numbers and returns them in an array
"""
def generate_LCG_random_numbers(self,seed, a, c, m, n):
    rand_variates = []
    for i in range(n):
        seed = self.lcg_generate(seed, a, c, m)
        rand_variates.append(seed)
    rand_variates = [i / m for i in rand_variates]
    return rand_variates
```

Figure below shows generating the LCG random numbers for the following LCG parameters.
seed : 50, a = 13, c = 9, m = 512
N (sample number) = 300

```
print(f"generating LCG random numbers")
print(f"seed value = 50, a = 13, c = 9, m = 512")
print(f"number of values = 300")
z = simulate.generate_LCG_random_numbers(50, 13,  9, 512, 300)
print(z)
print(f"-----------------Separator-----------------")
```

Figure below shows the generated random numbers after running the above code snippet

```
generating LCG random numbers
seed value = 50, a = 13, c = 9, m = 512
number of values = 300
[0.287109375, 0.75, 0.767578125, 0.99609375, 0.966796875, 0.5859375, 0.634765625, 0.26953125, 0.521484375, 0.796875, 0.376953
125, 0.91796875, 0.951171875, 0.3828125, 0.994140625, 0.94140625, 0.255859375, 0.34375, 0.486328125, 0.33984375, 0.435546875,
 0.6796875, 0.853515625, 0.11328125, 0.490234375, 0.390625, 0.095703125, 0.26171875, 0.419921875, 0.4765625, 0.21289062 5, 0.7
8515625, 0.224609375, 0.9375, 0.205078125, 0.68359375, 0.904296875, 0.7734375, 0.072265625, 0.95703125, 0.458984375, 0.984375
, 0.814453125, 0.60546875, 0.888671875, 0.5703125, 0.431640625, 0.62890625, 0.193359375, 0.53125, 0.923828125, 0.02734375, 0.
373046875, 0.8671875, 0.291015625, 0.80078125, 0.427734375, 0.578125, 0.533203125, 0.94921875, 0.357421875, 0.6640625, 0.6503
90625, 0.47265625, 0.162109375, 0.125, 0.642578125, 0.37109375, 0.841796875, 0.9609375, 0.509765625, 0.64453125, 0.396484375,
 0.171875, 0.251953125, 0.29296875, 0.826171875, 0.7578125, 0.869140625, 0.31640625, 0.130859375, 0.71875, 0.361328125, 0.714
84375, 0.310546875, 0.0546875, 0.728515625, 0.48828125, 0.365234375, 0.765625, 0.970703125, 0.63671875, 0.294921875, 0.851562
5, 0.087890625, 0.16015625, 0.099609375, 0.3125, 0.080078125, 0.05859375, 0.779296875, 0.1484375, 0.947265625, 0.33203125, 0.
333984375, 0.359375, 0.689453125, 0.98046875, 0.763671875, 0.9453125, 0.306640625, 0.00390625, 0.068359375, 0.90625, 0.798828
125, 0.40234375, 0.248046875, 0.2421875, 0.166015625, 0.17578125, 0.302734375, 0.953125, 0.408203125, 0.32421875, 0.232421875
, 0.0390625, 0.525390625, 0.84765625, 0.037109375, 0.5, 0.517578125, 0.74609375, 0.716796875, 0.3359375, 0.384765625, 0.01953
125, 0.271484375, 0.546875, 0.126953125, 0.66796875, 0.701171875, 0.1328125, 0.744140625, 0.69140625, 0.005859375, 0.09375, 0
.236328125, 0.08984375, 0.185546875, 0.4296875, 0.603515625, 0.86328125, 0.240234375, 0.140625, 0.845703125, 0.01171875, 0.16
9921875, 0.2265625, 0.962890625, 0.53515625, 0.974609375, 0.6875, 0.955078125, 0.43359375, 0.654296875, 0.5234375, 0.82226562
5, 0.70703125, 0.208984375, 0.734375, 0.564453125, 0.35546875, 0.638671875, 0.3203125, 0.181640625, 0.37890625, 0.943359375,
0.28125, 0.673828125, 0.77734375, 0.123046875, 0.6171875, 0.041015625, 0.55078125, 0.177734375, 0.328125, 0.283203125, 0.6992
1875, 0.107421875, 0.4140625, 0.400390625, 0.22265625, 0.912109375, 0.875, 0.392578125, 0.12109375, 0.591796875, 0.7109375, 0
.259765625, 0.39453125, 0.146484375, 0.921875, 0.001953125, 0.04296875, 0.576171875, 0.5078125, 0.619140625, 0.06640625, 0.88
0859375, 0.46875, 0.111328125, 0.46484375, 0.060546875, 0.8046875, 0.478515625, 0.23828125, 0.115234375, 0.515625, 0.72070312
5, 0.38671875, 0.044921875, 0.6015625, 0.837890625, 0.91015625, 0.849609375, 0.0625, 0.830078125, 0.80859375, 0.529296875, 0.
8984375, 0.697265625, 0.08203125, 0.083984375, 0.109375, 0.439453125, 0.73046875, 0.513671875, 0.6953125, 0.056640625, 0.7539
0625, 0.818359375, 0.65625, 0.548828125, 0.15234375, 0.998046875, 0.9921875, 0.916015625, 0.92578125, 0.052734375, 0.703125,
0.158203125, 0.07421875, 0.982421875, 0.7890625, 0.275390625, 0.59765625, 0.787109375, 0.25, 0.267578125, 0.49609375, 0.46679
6875, 0.0859375, 0.134765625, 0.76953125, 0.021484375, 0.296875, 0.876953125, 0.41796875, 0.451171875, 0.8828125, 0.494140625
, 0.44140625, 0.755859375, 0.84375, 0.986328125, 0.83984375, 0.935546875, 0.1796875, 0.353515625, 0.61328125, 0.990234375, 0.
890625, 0.595703125, 0.76171875, 0.919921875, 0.9765625, 0.712890625, 0.28515625, 0.724609375, 0.4375, 0.705078125, 0.1835937
5, 0.404296875, 0.2734375, 0.572265625, 0.45703125, 0.958984375, 0.484375, 0.314453125, 0.10546875]
----------Separator----------
```

To examine the randomness and independence of the generated variables, the Kolmogorov-Smirnov Test and the Autocorrelation Test will be performed. These tests will assess the uniformity and correlation structure of the variables, respectively.

7.2 Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov test is used for assessing the uniformity of a set of random numbers. Specifically, it measures the degree of conformity between the probability distribution of the numbers and the expected uniform distribution. The procedure follows the following steps.

Step 1: Ranking the data in ascending order. (Ri denotes the smallest observation)

$$R(1) \leq R(2) \leq \cdots \leq R(N)$$

Step 2: Compute D+ and D- such that

$$D^+ = \max_{1 \leq i \leq N} \left\{ \frac{i}{N} - R_{(i)} \right\}$$

$$D^- = \max_{1 \leq i \leq N} \left\{ R_{(i)} - \frac{i-1}{N} \right\}$$

Step 3: Compute D = max(D+, D−).

Step 4: find critical value Dα where α is the significance level for sample size N

Step 5: Compare D with Dα to confirm or reject the hypothesis that the distribution of the generated numbers is the uniform distribution is not rejected.

Steps 1 to 3 (computing D) were implemented using python function "kolmogorov_smirnov_uniform_test" shown in the figure below.

```
"""
    performs kolmogorov_smirnov uniformity test on sequence of random numbers
    parameter: data is the input array of random variates
    returns the value of d by finding the maximum value between d_plus and d_minus
"""
def kolmogorov_smirnov_uniform_test(self, data):
    sorted_data = sorted(data)
    i_divided_N = [(i + 1) / len(data) for i in range(len(data))]
    i_divided_N_minus_Ri = []
    for i in range(len(sorted_data)):
        i_divided_N_minus_Ri.append(i_divided_N[i] - sorted_data[i])
    i_minus_one_divided_by_n = [i / 300 for i in range(300)]
    Ri_minus_i_minus_one_divided_by_n = []
    for i in range(len(sorted_data)):
        Ri_minus_i_minus_one_divided_by_n.append(sorted_data[i] - i_minus_one_divided_by_n[i])
    d_plus = max(i_divided_N_minus_Ri)
    d_minus = max(Ri_minus_i_minus_one_divided_by_n)
    d = max(d_plus, d_minus)
    return d
```

Figure below shows running "kolmogorov_smirnov_uniform_test" with the random numbers generated.

```
print(f"performing kolmogorov smirnov uniformity test on generated data")
D = simulate.kolmogorov_smirnov_uniform_test(z)
print(f"computed D value = " + str(D))
print(f"----------------Separator----------------")
```

Figure below shows the computed D output after running the code snippet above.

```
----------------Separator----------------
performing kolmogorov smirnov uniformity test on generated data
computed D value = 0.03083333333333338
----------------Separator----------------
```

D value = 0.03083

Step 4 - 5 were done manually to confirm that the hypothesis that the distribution of the generated numbers is the uniform distribution is not rejected.

Step 4: finding Dα, α is the significance level 0.05 and sample size is 300

D0.05 = 1.36/sqrt(300) = 0.0785

Step 5: Comparing Dα with computed D

Since Dα = 0.0785 and D = 0.03083

D < Dα

Therefore we can confirm that the hypothesis that the distribution of the generated numbers is the uniform distribution is not rejected.

7.3 Autocorrelation Independence test

The autocorrelation test is used to test if the generated random numbers are independent. This test is used to confirm or reject the pre-assumed null hypothesis, which is that the generated random numbers are independent.

Follow these steps to conduct the test

Step 1: one must calculate the autocorrelation for every lag number (l) starting from the ith position. Such that

$$R_i, R_{i+\ell}, R_{i+2\ell}, \ldots, R_{i+(M+1)\ell}$$

Step 2: Compute the value of M using the equation below

$$i + (M+1)\ell \leq N$$

Where N is the number of samples.

Step 3: compute the autocorrelation Pil

For large numbers of M the autocorrelation Pil is defined as

$$\widehat{\rho}_{i\ell} = \frac{1}{M+1}\left[\sum_{k=0}^{M} R_{i+k\ell}R_{i+(k+1)\ell}\right] - 0.25$$

Step 4: Compute the standard deviation of the estimator

The standard deviation of the estimator is defined as

$$\sigma_{\hat{\rho}_{i\ell}} = \frac{\sqrt{13M+7}}{12(M+1)}$$

Step 5: compute the test statistics as follows

$$Z_0 = \frac{\hat{\rho}_{i\ell}}{\sigma_{\hat{\rho}_{i\ell}}}$$

Step 6: Compare Z0 with Zα (Where α is the level of significance)

Reject the null hypothesis of independence if

$$-z_{\alpha/2} \leq Z_0 \leq z_{\alpha/2}$$

Step 1 to 5 were implemented using the python function “autocorrelation_test” shown below

```
def autocorrelation_test(self,values, lag, M, i):

    sum_products = 0
    for k in range(M):
        index1 = i + (k * lag)
        index2 = i + ((k + 1) * lag)
        sum_products += values[index1] * values[index2]

    p = 1 / (M + 1) * sum_products - 0.25
    standard_dev = (13 * M + 7) ** 0.5 / (12 * (M + 1))
    z = p / standard_dev

    print(f"p = "+ str(p))
    print(f"standard deviation of the estimator = " +str(standard_dev))
    print(f"z = "+ str(z))
```

The chosen lag value is 4 and index i value is 1

M was calculated as follows:

$$i+(M+1)\ell \leq N$$

$$1 + (M + 1)\ 4 \ <= \ 300$$

$$M = 73$$

Figure below shows running “autocorrelation_test” with the random numbers generated and the constants defined above.

```
print(f"performing autocorrelation test")
print(f"lag value = 4, M = 73, i = 1 ")
simulate.autocorrelation_test(z, 4, 73, 1)
print(f"----------------Separator----------------")
```

Figure below shows the computed Z0 output after running the code snippet above.

```
performing autocorrelation test
lag value = 4, M = 73, i = 1
p = 0.038917335304054 06
standard deviation of the estimator = 0.034818974850766 46
z = 1.117704799490884
----------------Separator----------------
```

Step 6 is done manually as shown below.

The critical value for z0.025 (since α = 0.05) looked up from Z statistics table = 1.96
And Z0 was found to be = 1.12

Therefore because Z0 < Z0.025, we fail to reject the null hypothesis.

7.4 Random Variate Generation - Inverse Transform Technique

Random numbers are used to generate random variates. The random variates are used to generate service and processing times for both the workstation and inspector elements. This generation of random variates is done using the inverse transform technique. Using the inverse transform technique and using the workstations and inspectors respected $\hat{\lambda}$ values we were able to generate service times (Xi) that follow the exponential distribution we found from input modelling. Using the uniform generated random numbers from our LCG (Ri).

Recall the CDF of an exponential distribution and its inverse:

Ri = F(Xi) = 1 - $e^{-\lambda Xi}$, x >= 0

Xi = F^{-1}(Ri) = $\frac{-ln(1-Ri)}{\lambda}$, Xi must follow the distribution of F(x), therefore Xi is the reverse function of Ri.

The goal is to produce values {X1, X2, …, X300} that have an exponential distribution, that shall represent the service times. The python function shown below "get_exponential_value" and "compute_exponential_array" were used to generate the exponentially distributed service times using the uniform generated random numbers from our LCG and the respected $\hat{\lambda}$ values for all inspectors and workstations.

```
"""
    generates the exponential of the input value using the input lambda
    paramter:
"""
def get_exponential_value(self,value, lam_bda):
    # check if lam_bda is positive
    if lam_bda <= 0:
        print("Error: lambda must be positive.")
        return 0
    # compute the exponential value
    return (-math.log(1 - value)) / lam_bda

def compute_exponential_array(self, data, lam_bda):
    result =[]
    for x in data:
        result.append(self.get_exponential_value(x,lam_bda))
    return result
```

Figure below shows running "get_exponential_value" and "compute_exponential_array" with the random numbers generated from LCG and inspector 1 $\hat{\lambda}$ value to get inspector 1 service times.

```
print(f"generating exponentially distributed service times for inspector 1, lambda = 0.09654457318")
inspector_one_service_times = simulate.compute_exponential_array(z, 0.09654457318)
print(f"exponentially distributed service times for inspector 1 have been generated ")
print(f"Note that only inspector 1 service times will be printed as a sample")
print(f"The exponentially distributed service times for inspector 2, 3 and workstation 1, 2, and 3 wi
print(f" ")
print(f"expoentially distributed service times for inspector 1")
print(f" ")
print(inspector_one_service_times)
print(f" ")
```

Figure below shows inspector 1 service time output after running the code snippet above

```
generating exponentially distributed service times for inspector 1, lambda = 0.09654457318
exponentially distributed service times for inspector 1 have been generated
Note that only inspector 1 service times will be printed as a sample
The exponentially distributed service times for inspector 2, 3 and workstation 1, 2, and 3 will be computed but not printed

expoentially distributed service times for inspector 1

[3.5053992193434262, 14.359112226176094, 15.114273996606821, 57.43644890470438, 35.26983618887331, 9.132966476775048, 10.4326
52763475826, 3.2530966503877794, 7.6344675854599995, 16.509822079034596, 4.9006744415612244, 25.90155950136999, 31.2751789221
93536, 4.998545185474668, 53.23667780672453, 29.386708645837036, 3.061023941364602, 4.362891162105851, 6.900135045599289, 4.3
014197057170005, 5.9236673599542, 11.792047545673446, 19.895851711125637, 1.2453048683948804, 6.97920344482292, 5.13049486796
6699, 1.0419805861663214, 3.142905079234202, 5.640839959191253, 6.705065061727907, 2.479560012622398, 15.928852431507448, 2.6
34931575884951, 28.71822445235219, 2.3772588353465114, 11.919140060432795, 24.304880633259447, 15.378745640782617, 0.77694497
50947314, 32.599265479306894, 6.362937849513721, 43.07733667852828, 17.44735802289418, 9.633445951480699, 22.738443859625725,
 8.7492963184194, 5.852233214751629, 10.26780190980613, 2.2256769593652166, 7.848040306572895, 26.66916320723084, 0.287168373
1323205, 4.83593729332692, 20.91072396269722, 3.5623109559295587, 16.71095286471737, 5.781288348354998, 8.939355045324598, 7.
891288427859795, 30.868934305210686, 4.580960479773037, 11.29872050076067, 10.885529704910057, 6.628054223701246, 1.831979781
8333554, 1.3831061469976518, 10.65661629970538, 4.803719817896237, 19.098696173520842, 33.58647974381689, 7.383860764277542,
10.713372112944201, 5.230572071619803, 1.9534103636870006, 3.0067939221983915, 3.5908845188779024, 18.123113476768992, 14.687
962385939992, 21.064177287904, 3.9400606168389967, 1.4527004289695444, 13.139126143304447, 4.644118663265651, 12.996256154054
825, 3.851656865403639, 0.5825259408213594, 13.505168120406786, 6.939593798082095, 4.707664322699845, 15.027596419660943, 36.
56626475892509, 10.488191287960143, 3.619537123594821, 19.758658844517527, 0.9528797356542467, 1.807863566049272, 1.086820093
42184, 3.8810410269547035, 0.86453881636074, 0.6254158985853365, 15.65014745582997, 1.6643336481574518, 30.478023384588738, 4
.179560535469785, 4.2098911867186, 4.612491432585399, 12.112751491883246, 40.766035856904935, 14.941638166997455, 30.10133059
934984, 3.7931374442075296, 0.04053981691792414, 0.7334242808153515, 24.518453354372344, 16.6098992826877, 5.331625653649475,
 2.952846352329004, 2.8724481996434297, 1.8803813185067855, 2.0023840246625446, 3.7349467855389276, 31.698009467460 39, 5.4336
74853500851, 4.059118364438178, 2.739822700102247, 0.4127203346055507, 7.719368962452949, 19.489607094142794, 0.3916890380395
9704, 7.179556113088047, 7.550256471199934, 14.198521257411242, 13.067444819159265, 4.240310914897773, 5.031375355590699, 0.2
0430465118949517, 3.2808286883732256, 8.199189527694568, 1.4062523242276244, 11.41986702798582, 12.511181766737522, 1.4760028
235000069, 14.119149911170142, 12.178101298562716, 0.06086938183494326, 1.019633414606521, 2.7926692943316613, 0.975083195389
4181, 2.1258409093388915, 5.8167000409667775, 9.582295674692219, 20.61047366463844, 2.845786897921438, 1.569740205331352, 19.
35765741165076, 0.12209858476524098, 1.929010093274421, 2.661054944083054, 34.11777107069362, 7.9347178835187755, 38.04849041
8298734, 12.047811404552743, 32.13883812325077, 5.887888706071217, 11.001911940563811, 7.6768312845971804, 17.892928226032247
```

The service times for inspectors 2 and 3, and workstations 1,2, and 3 were computed as seen in the figure below but not printed.

```
print(f"----------------Separator----------------")
print(f"generating exponentially distributed service times for inspector 2, lambda = 0.06436289")
inspector_two_service_times = simulate.compute_exponential_array(z, 0.06436289)
print(f"exponentially distributed service times for inspector 2 have been generated ")
print(f"----------------Separator----------------")
print(f"generating exponentially distributed service times for inspector 3, lambda = 0.04846662111")
inspector_three_service_times = simulate.compute_exponential_array(z, 0.04846662111)
print(f"exponentially distributed service times for inspector 3 have been generated ")
print(f"----------------Separator----------------")
print(f"generating exponentially distributed service times for workstation 1, lambda = 0.21718277739")
workstation_one_service_times = simulate.compute_exponential_array(z, 0.21718277739)
print(f"exponentially distributed service times for workstation 1 have been generated ")
print(f"----------------Separator----------------")
print(f"generating exponentially distributed service times for workstation 2, lambda = 0.090150136")
workstation_two_service_times = simulate.compute_exponential_array(z, 0.090150136)
print(f"exponentially distributed service times for workstation 2 have been generated ")
print(f"----------------Separator----------------")
print(f"generating exponentially distributed service times for workstation 3, lambda = 0.11369346876")
workstation_three_service_times = simulate.compute_exponential_array(z, 0.11369346876)
print(f"exponentially distributed service times for workstation 3 have been generated ")
print(f"----------------Separator----------------")
```

Milestone 3

8. Model Validation and Verification

8.1 Verification

Multiple techniques were used to verify our model. These techniques gave us developers the reassurance that the model is built correctly. This was done by comparing the conceptual model to the computer representation that implements that model. This section outlines the techniques used to verify that the assumptions about system components and system structure, parameter values, abstractions, and simplifications are accurately represented by the operational model.

8.1.1 Flow Diagrams

We created flow diagrams that include each logically possible action a system can take when an event occurs for each event type. This allowed us to verify that the different events within the system are handled as expected. The types of events that exist within our model are products leaving workstations (end of activity), arriving components to a buffer (end of activity), components arriving to a workstation (start of activity) and components arriving to the inspector (start of activity). We have created a flow diagram for each type of event. Only end of activity events are placed on the FEL. This can be seen in the figures below.

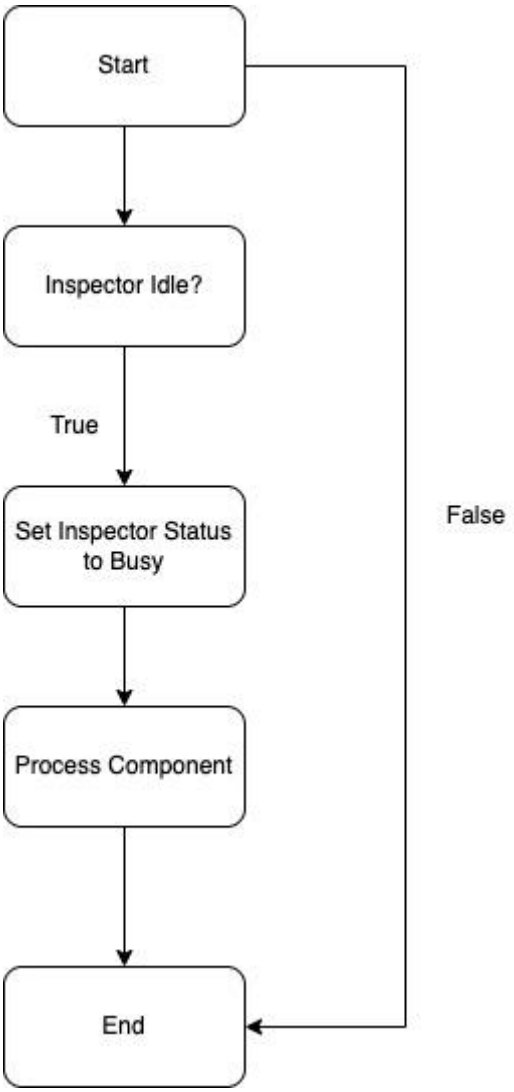


Figure 6: Flow Chart diagram of Component Arrive to Inspector Event

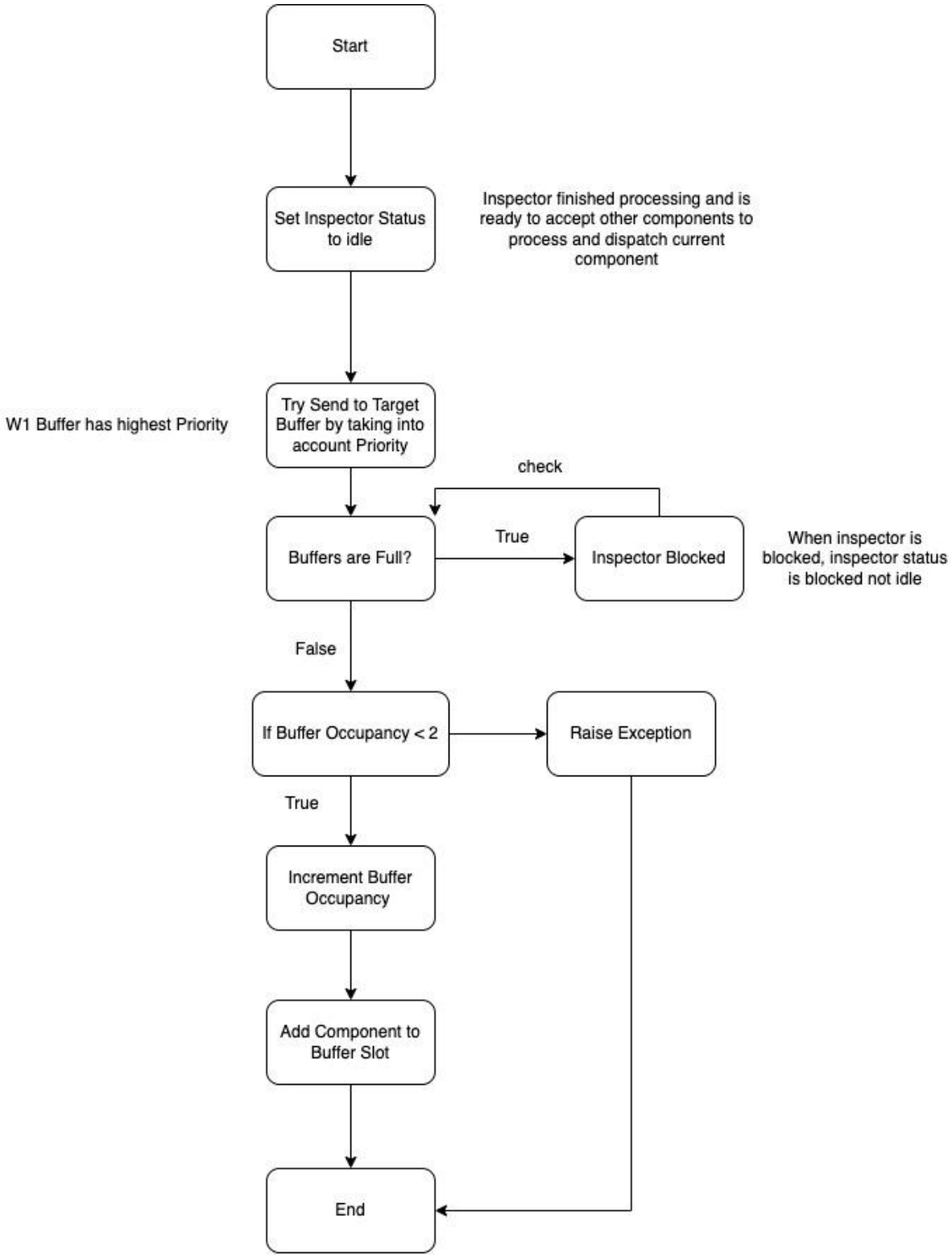


Figure 7: Flow Chart diagram of Component Arrive to Buffer

Figure 7 flow chart happens right after the component arrives at the inspector to be processed. Which is illustrated in Figure 6.

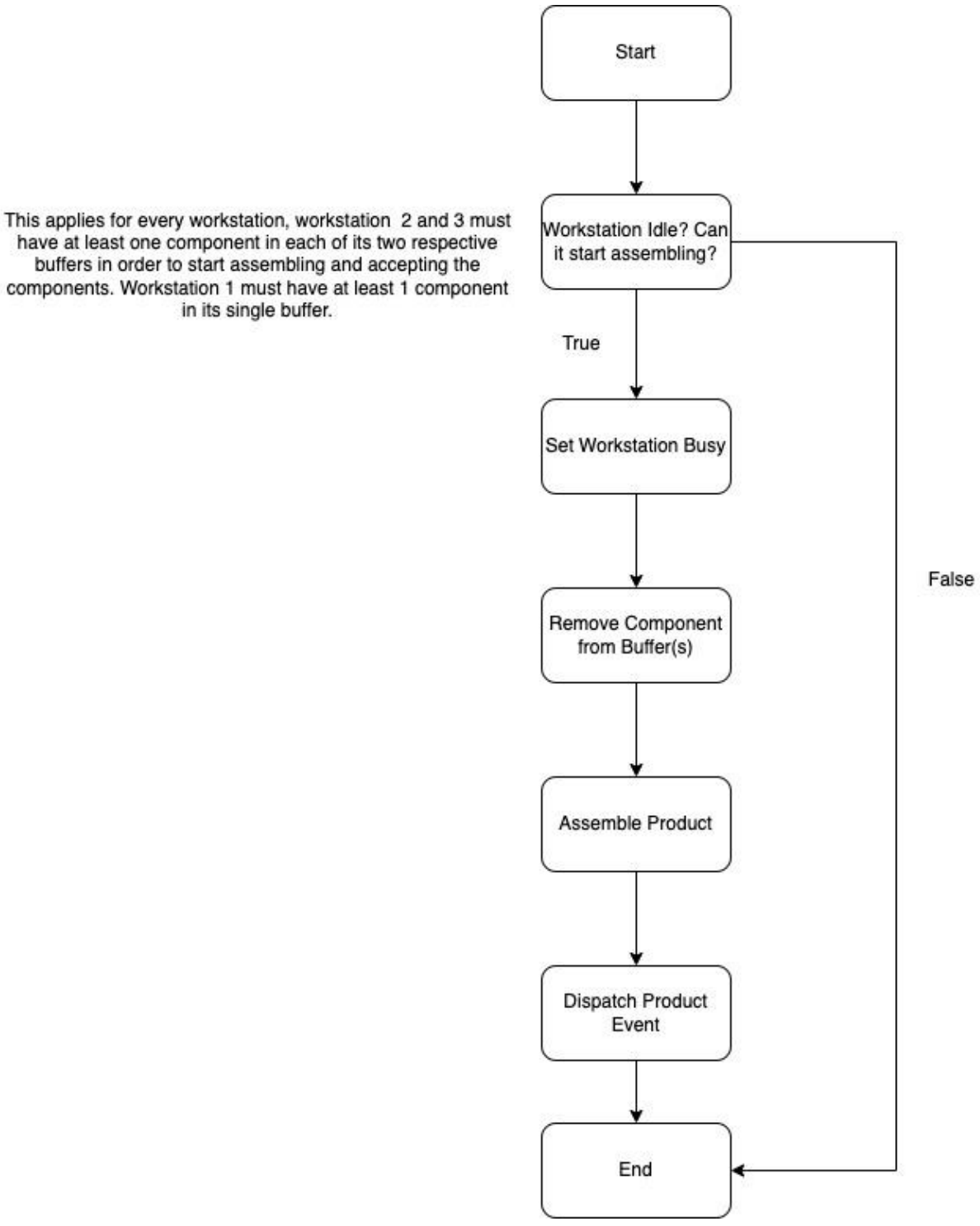


Figure 8: Flow Chart diagram of Component Arrive to Workstation

Figure 8 flow chart happens right after the component arrives at the buffer event. Which is illustrated in Figure 7.

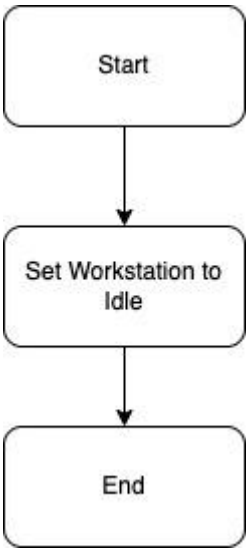


Figure 9: Flow Chart diagram of Product Assembled Event

Figure 9 flow chart happens right after the component arrives at the workstation and the workstation has valid component(s) to assemble a Product. Which is illustrated in Figure 8.

8.1.2 Documentation

We have built the operational model with documentation that includes precise definitions of every variable used and a general description of the purpose of each submodel, function and component. This is important to establish pre and post conditions on our model for clarity and reassurance of our outputs.

8.1.3 Little Law

Little's Law is used to verify the average number of components in each buffer in a given time. Little's law is used to ensure that our queuing system, which is the 5 buffers in this project, are accepting and departing components. Recall that the minimum and maximum capacity values of the buffers is between 0 and 2.

Little Law's equation is defined as $L = \lambda w$, where L = the average number of components in the buffer, λ = the arrival rate, and w = the average time the component spends in the buffer.

Multiplying the average time the component spends in the buffer by the arrival rate of the components will give us an estimate on the number of components at a given buffer at a given time. We track how long the component spends in the buffer and measure the arrival rate of components to the buffer to check if the conditions hold. It is also important to note that the average number of components in buffer 1 shall be greater than the average number of components in the other buffers as buffer 1 is the highest priority buffer for component 1.

8.1.4 Conceptual Model Requirements

There are a few model requirements that must be verified to measure the overall correctness of our model and how closely it is reflected in the operational model.

Some model requirements that have been checked and verified in the operational model are the size of the buffer and ensuring that it is not more than 2 or less than 0. In the Buffer class, we check to see if the occupancy (number of components) is less than 2, if it is then we add a component. Otherwise, if it is greater than two the simulation is ternimated.

The throughput of each product assembled per unit time cannot be negative. This was checked during simulation in the output measures of each product type (1, 2 and 3). If it were to be negative this means we are not assembling the product type enough and there is a fault in our simulation code.

The total count of products assembled can't be more than the number of components inspected. This is checked by keeping a count of the number of components expected, the number of components in the buffer and the number of products assembled. The number of components inspected must equal the number of components in the respected buffer + the number of components assembled from the respected workstation. This equation must hold true at the end of the simulation. Also, the number of components inspected equals the number of components that left the inspector.

8.2 Validation

To ensure that we were building the correct model, we used multiple approaches for validation. Typically, calibration is used for validation, which involves comparing the model to the actual system behavior and using any discrepancies to improve the conceptual model. However, since we did not have a real system to compare our model to in this project, we used a specification of the system to validate our model.
The process of validating the simulation model to ensure it accurately represents the system being simulated is known as validation. This process involves comparing the simulation findings to actual data from the system or to projected results based on system knowledge. The validation process includes three steps: first, creating a model with high face validity, then validating the model assumptions, and finally, comparing model input-output transformations to the real-world system's input-output transformations.
Validation is a critical phase in the simulation modelling process as it tests the model's ability to predict the future behavior of the real system.

8.2.1 Face Validity

The subjective evaluation of whether a simulation model accurately represents the system being simulated, based on expert judgment, is known as face validity, a form of qualitative validation that focuses on the model's perceived credibility rather than factual information. Including users in the calibration process is beneficial because it allows the model to be updated iteratively based on early model flaws, increasing the model's perceived validity, and credibility, which is essential for managers to rely on simulation findings as a basis for decision-making. Inspectors, for example, are supposed to be unable to examine components faster than one second per component on average, and they typically inspect components in less than an hour. With these presumptions, the inspector's face validity appears to be adequate. The workstation's validity is comparable to that of examiners, but most components are anticipated to require more than one second to create. Overall, the system appears to be legitimate, with an acceptable number of products created per second in comparison to the number of things needed to process and manufacture components into products.

8.2.1.1 Sensitivity Analysis

Sensitivity analysis is a method that can be employed to evaluate the face validity of a model. It involves modifying one or more input variables and assessing whether the model performs as expected. In this particular case, the goal is to validate whether the model behaves as anticipated based on the input service times for inspectors and workstations. If the service times for either of these components are increased, it can be observed in the operational

model that the number of products produced per unit time will decrease, and the waiting time of components in the buffers will increase, which is the desired outcome to be achieved from the model.

8.2.2 Model assumptions

The validation of a model involves two types of assumptions: structural and data assumptions. Structural assumptions are based on observations and include questions about how the system works, as well as requirements and abstractions of reality. These assumptions are verified based on actual observations during appropriate time periods. Data assumptions, on the other hand, are based on the gathering of accurate data and the right statistical analysis of data.

Validating structural assumptions involves validating how the operational model operates following the real system operations. The simulation model, for example, showcases the behavior of the buffer queues by allocating a maximum capacity of two, restricting components to enter buffers based on their type, and applying a priority algorithm for putting components in buffers based on buffer number and size.

Validating data assumptions requires the collection of reliable data and completing statistical analysis on the data. In some cases, such as in our project, reliable data from the real system could not be collected. In this case, validating data assumptions was completed by using randomly generated numbers and observing the output of the operational model and validating that the output is reasonable and within the assumptions expected from the model.

It was observed that the average buffer size in our operational model is always between 0 and 2, which is within the requirements of the model. In addition, the product per unit time is always between greater than 0 and less than 3 per unit time, which is a reasonable number since we have three workstations working so the expected value would be between zero and three. Goodness-of-fit tests can be applied to validate the data assumptions.

8.2.3 Input and Output Transformation

To test the model's ability to predict the future behaviour of the real system when the input data match the real inputs, we use input-output transformation. If some input variable increases or decreases, the model should predict what happens to the system in that circumstance. This can be tested by providing our system with different input variables and comparing it to the historical data provided in the project specification. In this case we used the historical data provided in the project document, and compared it to the mean values generated later on from table 9.4.

Historical data provided:

C1 Buffer of Workstation 1	C1 Buffer of Workstation 2	C2 Buffer of Workstation 2	C1 Buffer of Workstation 3	C3 Buffer of Workstation 3
0.28	0.41	0.60	0.32	1.75

Our values that were calculated in table 9.4:

C1 Buffer of Workstation 1	C1 Buffer of Workstation 2	C2 Buffer of Workstation 2	C1 Buffer of Workstation 3	C3 Buffer of Workstation 3
0.20	0.075	1.62	0.02	1.86

As seen above, the values differ for C1 buffer and C2 buffer in Workstation 2, however, the C1 buffer in Workstation 1 and C3 buffer in Workstation 3 values are close to the historical values.

9. Production Run and Analysis

Production runs allow us developers to get meaningful results. To get meaningful results we need to estimate the systems performance using simulation and analysis and observe the simulation output to check if it is close to the real systems output and is variable on the input parameters of our model.

9.1 Heuristics

The heuristics/quantities of interest to be measured is the following:

- 3x facility throughput as products per unit time
- 2x inspector blocked time (or idle time)
- 3x workstation idle time
- 5x average buffer occupancy/capacity for each buffer
- 5x the average time component is in buffer

9.2 Steady State Analysis

Our simulation is expected to run as a non-terminating steady state simulation. The steady state of the system is when we have no non biased results. As seen later in the report through table 9.4 and through the screenshot outputs of each run. The throughput on the number of P1 products is significantly greater than the throughput of all other products. Which means that workstations 1 buffer accepts more components than the other buffers and hence produces more products. Causing workstation 2 and 3 to be underutilized and skewing our results. The system runs for approximately 3266 seconds to reach steady state. This is how long it takes to

read the files until there are no service times for inspecting component C1. We examine component C1 with its 300 service times as C1 is a component that is used for all products.

9.3 Determining Number of Replications

A number of replications is needed to see more independent sample values of the quantities of interest to achieve unbiased estimates of the mean and variance. A single replication shall be defined as a single run until the end of the data file has been reached (i.e. 300 service times is read).

To determine the number of replications needed, we first must choose an arbitrary number R_0 as our initial basis for replications. R_0 was chosen to be 5. We then found the average values of the desired quantities for each replication R.

A sample calculation on the heuristic on the average buffer occupancy of buffer C2 of workstation 2 within each replication is shown in the outputs in Table 9.1 below. The Table 9.2 shows the average across the 5 replications Y, the sample standard deviation, the sample variance, the standard error, the CI half width and the minimum number of replications required for the average capacity of buffer C2 workstation 2's buffer.

Replication (R)	**Buffer C2 Output (Y_i)**
1	1.51643
2	1.63536
3	1.72613
4	1.59154
5	1.65101

Table 9.1 Heuristic Average Buffer Occupancy Within 5 Replications (Buffer C2 Workstation 2)

The formulas used for the calculated statistics in Table 9.2 are as follows with the number of replications R is 5.

The average across replications is calculated as $Y = \frac{1}{R}\sum_{i=1}^{R} Y_i$

The sample variance is calculated as $S_o^2 = \frac{1}{R-1}\sum_{i=1}^{R} (Y_i - Y)^2$

The sample standard deviation is $\sqrt{S_o^2}$

The CI half-width = H = $t_{\alpha/2}, R \; - \; 1 \frac{S}{\sqrt{R}}$, where α = 0.05

The Standard Error = $\frac{S}{\sqrt{R}}$

Average Across Replication (Y)	1.624094
Sample Variance	0.0152203862
Sample Standard Deviation	0.1233709293
CI Half Width	0.0244
Standard Error	0.05517
Number Of Required Replications	15.78

Table 9.2 Calculated Statistics for Average Buffer Occupancy Within 5 Replications (Buffer C2 Workstation 2)

The number of required replications R must be >= $\left(\frac{z_{\alpha/2} S_0}{\epsilon}\right)^2$, where ϵ = 0.01. This epsilon value was chosen because it was found to be an efficient error factor.

The estimates of the quantities of interest are accompanied by 95% confidence intervals with a width that does not exceed 20% of the estimated values. The CI half width was calculated to be 0.0244. The value multiplied by 2 would be 0.0488 and this is indeed less then the 20% required width as stated. The required number of replications was chosen to be 17. Based on the calculated number of replications. 17 was the greatest number of replications.

Table 9.3 shows the calculated statistics of 17 replications for the average buffer occupancy of buffer C2 of workstation 2.

Average Across Replication (*Y*)	1.6149
Sample Variance	1.542E^-2
Sample Standard Deviation	0.124177
CI Half Width	0.0152
Standard Error	0.0301

Number Of Required Replications	15.80

Table 9.3 Calculated Statistics for Average Buffer Occupancy Within 17 Replications (Buffer C2 Workstation 2)

As we can see in Table 9.3. As we increased the number of replications from 5 to 17, the precision and accuracy increases proportionally with the number of replications. The standard error and CI half-width values in Table 9.3 is less than its previous values in Table 9.2 for 5 replications. The CI half-width was calculated to be 0.0152. The value multiplied by 2 would be 0.0304 and this is indeed less then the 20% required width as stated. Then we know that 17 replications is sufficient to show statistical confidence/significance that our simulation is correct.

After running simulation.py for 17 replications, the following outputs are shown for each run:

Run 1:

```
ktop/winter2023/SYSC 4005/Simulation Project/Simulation.py"
generating LCG random numbers
seed value = 50, a = 13, c = 9, m = 512
number of values = 300
----------Separator----------
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 157.43747758968436
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.20187793427230047
Average buffer occupancy for buffer C1 of workstation 2 is 0.0892018779342723
Average buffer occupancy for buffer C1 of workstation 3 is 0.010954616588419406
Average buffer occupancy for buffer C2 of workstation 2 is 1.516431924882629
Average buffer occupancy for buffer C3 of workstation 3 is 1.8810641627543037
Average time component C1 spends in buffer C1 of workstation 1 is 1.2076111929325883
Average time component C1 spends in buffer C1 of workstation 2 is 6.821140325081894
Average time component C1 spends in buffer C1 of workstation 3 is 2.3718874194274577
Average time component C2 spends in buffer C2 of workstation 2 is 151.7299228326054
Average time component C3 spends in buffer C3 of Workstation 3 is 624.0897572014601
```

Run 2:

```
**********2267**********
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 145.05620335074013
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.20500782472613457
Average buffer occupancy for buffer C1 of workstation 2 is 0.06259780907668232
Average buffer occupancy for buffer C1 of workstation 3 is 0.01564945226917058
Average buffer occupancy for buffer C2 of workstation 2 is 1.6353677621283256
Average buffer occupancy for buffer C3 of workstation 3 is 1.8716744913928012
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 4.6599651614544
Average time component C1 spends in buffer C1 of workstation 3 is 5.989586078737057
Average time component C2 spends in buffer C2 of workstation 2 is 165.18563702250273
Average time component C3 spends in buffer C3 of Workstation 3 is 615.8699120344712
```

Run 3:

```
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 136.22300987641347
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.20500782472613457
Average buffer occupancy for buffer C1 of workstation 2 is 0.06416275430359937
Average buffer occupancy for buffer C1 of workstation 3 is 0.018779342723004695
Average buffer occupancy for buffer C2 of workstation 2 is 1.726134585289515
Average buffer occupancy for buffer C3 of workstation 3 is 1.8732394366197183
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 4.6599651614544
Average time component C1 spends in buffer C1 of workstation 3 is 5.953082822416673
Average time component C2 spends in buffer C2 of workstation 2 is 167.87557525823578
Average time component C3 spends in buffer C3 of Workstation 3 is 613.5224908436455
```

Run 4:

```
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 211.54282485072645
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.20500782472613457
Average buffer occupancy for buffer C1 of workstation 2 is 0.06729264475743349
Average buffer occupancy for buffer C1 of workstation 3 is 0.03129890453834116
Average buffer occupancy for buffer C2 of workstation 2 is 1.591549295774648
Average buffer occupancy for buffer C3 of workstation 3 is 1.8482003129890454
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 5.065309520992852
Average time component C1 spends in buffer C1 of workstation 3 is 11.12083614617245
Average time component C2 spends in buffer C2 of workstation 2 is 156.1249156051003
Average time component C3 spends in buffer C3 of Workstation 3 is 606.0990262621748
```

Run 5:

```
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 134.22352492478632
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.20500782472613457
Average buffer occupancy for buffer C1 of workstation 2 is 0.06259780907668232
Average buffer occupancy for buffer C1 of workstation 3 is 0.03129890453834116
Average buffer occupancy for buffer C2 of workstation 2 is 1.6510172143974962
Average buffer occupancy for buffer C3 of workstation 3 is 1.8794992175273866
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 4.6599651614544
Average time component C1 spends in buffer C1 of workstation 3 is 7.41422551710325
Average time component C2 spends in buffer C2 of workstation 2 is 160.16019448151846
Average time component C3 spends in buffer C3 of Workstation 3 is 622.2160761830874
```

Run 6:

```
*********2270*********
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 147.79399483410486
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.21003134796238246
Average buffer occupancy for buffer C1 of workstation 2 is 0.07836990595611286
Average buffer occupancy for buffer C1 of workstation 3 is 0.017241379310344827
Average buffer occupancy for buffer C2 of workstation 2 is 1.6300940438871474
Average buffer occupancy for buffer C3 of workstation 3 is 1.8432601880877744
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 5.598527993870896
Average time component C1 spends in buffer C1 of workstation 3 is 5.953082822416673
Average time component C2 spends in buffer C2 of workstation 2 is 162.59322403047958
Average time component C3 spends in buffer C3 of Workstation 3 is 596.312960873645
```

Run 7:

```
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 133.33269543408866
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.20344287949921752
Average buffer occupancy for buffer C1 of workstation 2 is 0.1189358372456964
Average buffer occupancy for buffer C1 of workstation 3 is 0.01564945226917058
Average buffer occupancy for buffer C2 of workstation 2 is 1.513302034428795
Average buffer occupancy for buffer C3 of workstation 3 is 1.9045383411580594
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 8.740795395623037
Average time component C1 spends in buffer C1 of workstation 3 is 5.953082822416673
Average time component C2 spends in buffer C2 of workstation 2 is 145.80390230738655
Average time component C3 spends in buffer C3 of Workstation 3 is 623.3633469394124
```

Run 8:

```
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 30
Number of products P3 made is 6
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 33.059405841080526
The throughput per hour for Product 3 is 6.611881168216105
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 143.77478656429696
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2829.4029263080706
Total idle time for workstation 3 in seconds is 3152.363542917974
Average buffer occupancy for buffer C1 of workstation 1 is 0.20031298904538342
Average buffer occupancy for buffer C1 of workstation 2 is 0.06572769953051644
Average buffer occupancy for buffer C1 of workstation 3 is 0.01564945226917058
Average buffer occupancy for buffer C2 of workstation 2 is 1.6150234741784038
Average buffer occupancy for buffer C3 of workstation 3 is 1.8669796557120502
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 5.006313054774288
Average time component C1 spends in buffer C1 of workstation 3 is 4.9609023520138935
Average time component C2 spends in buffer C2 of workstation 2 is 168.54766541529645
Average time component C3 spends in buffer C3 of Workstation 3 is 569.3016902341743
```

Run 9:

```
**********2259**********
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 164.96001164045103
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.2081377151799687
Average buffer occupancy for buffer C1 of workstation 2 is 0.06572769953051644
Average buffer occupancy for buffer C1 of workstation 3 is 0.017214397496087636
Average buffer occupancy for buffer C2 of workstation 2 is 1.7449139280125197
Average buffer occupancy for buffer C3 of workstation 3 is 1.8998435054773084
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 4.6599651614544
Average time component C1 spends in buffer C1 of workstation 3 is 5.953082822416673
Average time component C2 spends in buffer C2 of workstation 2 is 175.4892787060331
Average time component C3 spends in buffer C3 of Workstation 3 is 588.5817404516796
```

Run 10:

```
**********2276**********
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 30
Number of products P3 made is 6
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 33.059405841080526
The throughput per hour for Product 3 is 6.611881168216105
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 213.79398759738493
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2829.4029263080706
Total idle time for workstation 3 in seconds is 3152.363542917974
Average buffer occupancy for buffer C1 of workstation 1 is 0.20187793427230047
Average buffer occupancy for buffer C1 of workstation 2 is 0.09702660406885759
Average buffer occupancy for buffer C1 of workstation 3 is 0.01564945226917058
Average buffer occupancy for buffer C2 of workstation 2 is 1.408450704225352
Average buffer occupancy for buffer C3 of workstation 3 is 1.8435054773082942
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 6.955515051657593
Average time component C1 spends in buffer C1 of workstation 3 is 4.9609023520138935
Average time component C2 spends in buffer C2 of workstation 2 is 151.75335162057738
Average time component C3 spends in buffer C3 of Workstation 3 is 533.2716310465293
```

Run 11:

```
**********2273**********
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 30
Number of products P3 made is 6
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 33.059405841080526
The throughput per hour for Product 3 is 6.611881168216105
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 176.93300004701086
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2829.4029263080706
Total idle time for workstation 3 in seconds is 3152.363542917974
Average buffer occupancy for buffer C1 of workstation 1 is 0.20500782472613457
Average buffer occupancy for buffer C1 of workstation 2 is 0.08450704225352113
Average buffer occupancy for buffer C1 of workstation 3 is 0.01564945226917058
Average buffer occupancy for buffer C2 of workstation 2 is 1.6948356807511737
Average buffer occupancy for buffer C3 of workstation 3 is 1.8575899843505477
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 5.910446888706845
Average time component C1 spends in buffer C1 of workstation 3 is 4.9609023520138935
Average time component C2 spends in buffer C2 of workstation 2 is 172.19338874682745
Average time component C3 spends in buffer C3 of Workstation 3 is 562.8759471758359
```

Run 12:

```
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 162.07583596503463
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.20500782472613457
Average buffer occupancy for buffer C1 of workstation 2 is 0.06729264475743349
Average buffer occupancy for buffer C1 of workstation 3 is 0.017214397496087636
Average buffer occupancy for buffer C2 of workstation 2 is 1.6697965571205007
Average buffer occupancy for buffer C3 of workstation 3 is 1.8716744913928012
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 4.792300342096071
Average time component C1 spends in buffer C1 of workstation 3 is 5.953082822416673
Average time component C2 spends in buffer C2 of workstation 2 is 165.86000229748282
Average time component C3 spends in buffer C3 of Workstation 3 is 618.8816085585607
```

Run 13:

```
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 212.60630032256884
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.20187793427230047
Average buffer occupancy for buffer C1 of workstation 2 is 0.06572769953051644
Average buffer occupancy for buffer C1 of workstation 3 is 0.028169014084507043
Average buffer occupancy for buffer C2 of workstation 2 is 1.5477308294209702
Average buffer occupancy for buffer C3 of workstation 3 is 1.8497652582159625
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 4.817652912909829
Average time component C1 spends in buffer C1 of workstation 3 is 8.796794806382014
Average time component C2 spends in buffer C2 of workstation 2 is 158.35227108549617
Average time component C3 spends in buffer C3 of Workstation 3 is 578.4770492514821
```

Run 14:

```
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 30
Number of products P3 made is 6
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 33.059405841080526
The throughput per hour for Product 3 is 6.611881168216105
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 164.68720400524595
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2829.4029263080706
Total idle time for workstation 3 in seconds is 3152.363542917974
Average buffer occupancy for buffer C1 of workstation 1 is 0.20500782472613457
Average buffer occupancy for buffer C1 of workstation 2 is 0.06416275430359937
Average buffer occupancy for buffer C1 of workstation 3 is 0.0297339593114241
Average buffer occupancy for buffer C2 of workstation 2 is 1.6322378716744914
Average buffer occupancy for buffer C3 of workstation 3 is 1.862284820031299
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 5.006313054774288
Average time component C1 spends in buffer C1 of workstation 3 is 7.169430533888128
Average time component C2 spends in buffer C2 of workstation 2 is 166.19224751863032
Average time component C3 spends in buffer C3 of Workstation 3 is 567.1736245255762
```

Run 15:

```
**********2285**********
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 30
Number of products P3 made is 6
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 33.059405841080526
The throughput per hour for Product 3 is 6.611881168216105
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 205.18349739014218
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2829.4029263080706
Total idle time for workstation 3 in seconds is 3152.363542917974
Average buffer occupancy for buffer C1 of workstation 1 is 0.20657276995305165
Average buffer occupancy for buffer C1 of workstation 2 is 0.09233176838810642
Average buffer occupancy for buffer C1 of workstation 3 is 0.028169014084507043
Average buffer occupancy for buffer C2 of workstation 2 is 1.5054773082942097
Average buffer occupancy for buffer C3 of workstation 3 is 1.84037558685446
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 7.137020411653324
Average time component C1 spends in buffer C1 of workstation 3 is 7.1941334034648134
Average time component C2 spends in buffer C2 of workstation 2 is 156.46438429723963
Average time component C3 spends in buffer C3 of Workstation 3 is 551.2294376699733
```

Run 16:

```
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 188.0669496911579
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.20187793427230047
Average buffer occupancy for buffer C1 of workstation 2 is 0.06416275430359937
Average buffer occupancy for buffer C1 of workstation 3 is 0.028169014084507043
Average buffer occupancy for buffer C2 of workstation 2 is 1.6103286384976525
Average buffer occupancy for buffer C3 of workstation 3 is 1.8544600938967135
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 4.75474881068916
Average time component C1 spends in buffer C1 of workstation 3 is 8.005197862026899
Average time component C2 spends in buffer C2 of workstation 2 is 164.26819239960528
Average time component C3 spends in buffer C3 of Workstation 3 is 613.7884479426555
```

Run 17:

```
**********2262**********
clock end time is 3266.846371019658
Number of products P1 made is 263
Number of products P2 made is 31
Number of products P3 made is 5
The throughput per hour for Product 1 is 289.82079120680595
The throughput per hour for Product 2 is 34.16138603578321
The throughput per hour for Product 3 is 5.509900973513422
Total blocked time for Inspector 1 in seconds is 0
Total blocked time for Inspector 2 in seconds is 146.82882828634746
Total idle time for workstation 1 in seconds is 2040.5494394146804
Total idle time for workstation 2 in seconds is 2826.74748868241
Total idle time for workstation 3 in seconds is 3160.11894311804
Average buffer occupancy for buffer C1 of workstation 1 is 0.20500782472613457
Average buffer occupancy for buffer C1 of workstation 2 is 0.06729264475743349
Average buffer occupancy for buffer C1 of workstation 3 is 0.017214397496087636
Average buffer occupancy for buffer C2 of workstation 2 is 1.705790297339593
Average buffer occupancy for buffer C3 of workstation 3 is 1.8826291079812207
Average time component C1 spends in buffer C1 of workstation 1 is 1.2089717570832017
Average time component C1 spends in buffer C1 of workstation 2 is 4.6599651614544
Average time component C1 spends in buffer C1 of workstation 3 is 5.953082822416673
Average time component C2 spends in buffer C2 of workstation 2 is 167.29858960432983
Average time component C3 spends in buffer C3 of Workstation 3 is 613.6048881844736
```

The values were then used to calculate the confidence intervals of the quantities of interest from the 17 replications. The quantities of interest along with their respective average across replications, estimated variance, and 95% confidence interval are shown in Table 9.4 below. The 95% confidence interval means that with 95% confidence we can say that the true population mean lies between those values of the interval based on the sample data.

Table 9.4: Statistical Analysis

Heuristic	Estimated Mean	Estimated Variance	95% confidence Interval
Total Products Produced	299	0	0
Product 1 throughput	289.82	0	0
Product 2 throughput	35.81	4.405	(32.66, 38.96)
Product 3 throughput	5.83	0.267	(5.28, 6.38)
Total blocked time for Inspector 1	0	0	0
Total blocked time for Inspector 2	167.32	833.089	(144.62, 190.02)
Total idle time for workstation 1	2040.55	0	0
Total idle time for workstation 2	2827.53	1.549	(2823.29, 2831.77)
Total idle time for workstation 3	3157.84	13.283	(3136.96, 3178.72)
Average buffer occupancy for buffer C1, workstation 1	0.2045	5.969E^-6	(0.2015, 0.2075)
Average buffer occupancy for buffer C1, workstation 2	0.0751	2.5675E^-4	(0.0607, 0.0895)
Average buffer occupancy for buffer C1, workstation 3	0.0208	4.700E^-5	(0.0194, 0.0222)
Average buffer occupancy for buffer C2, workstation 2	1.6149	1.542E^-3	(1.5662, 1.6636)
Average buffer occupancy for buffer C3, workstation 3	1.8612	4.692E^-4	(1.8128, 1.9096)
Average time C1 spends in buffer C1 of workstation 1	1.2089	1.156E^-7	(1.2081, 1.2097)
Average time C1 spends in buffer C1 of workstation 2	5.5212	1.4387	(4.2488, 6.7936)
Average time C1 spends in buffer C1 of workstation 3	6.3920	3.5414	(4.5741, 8.2099)
Average time C2 spends in buffer C2 of workstation 2	162.11	62.14	(147.62, 176.60)

Average time C3 spends in buffer C3 of workstation 3	594.04	854.29	(568.577, 619.503)

The analysis suggests that Inspector 1 prioritizes Component 1 in Buffer 1 at workstation 3, causing Inspector 1 to never be blocked while Inspector 2 is blocked around 17% of the time. Also, the total idle time for workstation 1 in comparison to workstation 2 and 3 is smaller which means that workstation 2 is busier. This could be the reason as to why the production of product 2 and 3 is significantly lower than the production of product 1. Thus, the manufacturing facility should reconsider the prioritization of the components to make all the products produce a fair amount.

10. Code Review

The main class responsible for running the simulation is "simulation.py". It orchestrates the collaboration between various classes representing the manufacturing facility parts, such as Workstation, Component, Inspector, and Buffer. These classes are in the Simulation Project folder submitted with this report. In addition to coordinating these classes, "simulation.py" also implements the Future Event List (FEL) and its related functions. The FEL will be explained in the following subsection. The simulation class also manages the simulation clock, which measures time in seconds. All service times computed in the previous milestone are considered to be in seconds. Once the simulation starts, it displays the time elapsed in seconds. The constructor of the Simulation class initializes the mentioned components can be seen in the figure below.

```
"""
Constructor
"""
def __init__(self):
    #the simulation has three C1 buffers for workstations 1,2, and 3
    self.buffer_C1_WST1 = Buffer(Component.C1, 1) #C1 buffer for workstation 1
    self.buffer_C1_WST2 = Buffer(Component.C1, 2) #C1 buffer for workstation 2
    self.buffer_C1_WST3 = Buffer(Component.C1, 3) #C1 buffer for workstation 3
    #the simulation has one C2 buffer for workstation 2
    self.buffer_C2_WST2 = Buffer(Component.C2, 2) #C2 buffer for workstation 2
    #the simulation has one C3 buffer for workstation 3
    self.buffer_C3_WST3 = Buffer(Component.C3, 3) #C3 buffer for workstation 3
    #the simulation has 3 workstations
    self.workstation_one = Workstation(1, [self.buffer_C1_WST1])
    self.workstation_two = Workstation(2, [self.buffer_C1_WST2, self.buffer_C2_WST2])
    self.workstation_three = Workstation(3, [self.buffer_C1_WST3, self.buffer_C3_WST3])

    #the simulation has two inspectors
    self.first_inspector = Inspector(1,[self.buffer_C1_WST1, self.buffer_C1_WST2, self.buffer_C1_WST3], [Component.C1] )
    self.second_inspector = Inspector(2, [self.buffer_C2_WST2, self.buffer_C3_WST3], [Component.C2 , Component.C3] )

    self.FEL = []
    self.clock = 0
```

10.1 Future Event List Implementation

The Future Event List is the data structure in simulation that keeps track of events that are scheduled to occur in the future. It contains information such as the time the event is scheduled to occur, the type of event, and its parameters or data. The FEL is an initialized empty list as can be seen in the previous constructor figure. However, events are added to it and sorted when the simulation runs. An event is a tuple that consists of three elements. First element is the type which can either be “Arrival”, or “Completion”. This allows the simulation to differentiate between arrival events and completion events. The second element is the object that is associated with the event which can either be an Inspector or a workstation object. The third element is the event time to which an event has arrived or has been completed. The Figure below shows the implementation of function “create_event” that is used to create new events.

```
def create_event(self, e_type, object_type, event_time):
    new_tuple = (e_type, object_type, event_time)
    return new_tuple
```

Adding new events to the FEL requires events to be appended to the list and then sorted in ascending order based on their scheduled times. This ensures that the events occurring earlier in time are positioned at the top of the list and are executed first during the simulation. The Figure below shows the implementation of function “add_event_notice_to_FEL” that is used to add events to FEL and sort them as per their event times.

```
"""
This function is used to add event notices to FEL
"""
def add_event_notice_to_FEL(self, event_notice):
    self.FEL.append(event_notice)
    self.FEL.sort(key = lambda x: x[2])
```

In the following subsection the run method will utilize the FEL list by popping events from the top of the list, executing them, and subsequently creating and adding new events as necessary.

10.2 Run Function

The “run” function in the simulation class is of utmost importance, as it is responsible for calculating initialization parameters (i.e., service times), creating initial events and adding them to FEL before starting the simulation, and printing outputs and statuses for each event. Additionally, it coordinates the while loop that runs the simulation until it is stopped, and calculates and prints the heuristics at the end of the simulation to help reflect on the simulation results. In this sub-section, the function will be segmented and explained in detail.

Note that in the next milestone this function will be divided into smaller functions to make it easier for readability and code management.

The Figure below shows the first segment of the run function where the services time are calculated, tested, and saved in arrays as explained in the previous milestone (Refer to section 7 Linear Congruential Generator and Testing – Implementation Code)

```
def run(self):
    """
    This part is only for initialization
    """
    # m = 512, seed = 50, a = 13, c = 9
    #generate_LCG_random_numbers(seed, a, c, m, n)
    print(f"generating LCG random numbers")
    print(f"seed value = 50, a = 13, c = 9, m = 512")
    print(f"number of values = 300")
    z = self.generate_LCG_random_numbers(50, 13,  9, 512, 300)
    print(f"----------------Separator----------------")
    print(f"performing kolmogorov smirnov uniformity test on generated data")
    D = self.kolmogorov_smirnov_uniform_test(z)
    print(f"computed D value = " + str(D))
    print(f"----------------Separator----------------")
    print(f"performing autocorrelation test")
    print(f"lag value = 4, M = 73, i = 1 ")
    self.autocorrelation_test(z, 4, 73, 1)
    print(f"----------------Separator----------------")
    #inspector 1 lambda = 0.09654457318, inspector 2 component 2 lambda = 0.06436289, inspector 2 componen
    #workstation 1 lambda = 0.21718277739, workstation 2 lambda =  0.090150136, workstation 3 lambda =  0.
    inspector_one_service_times = self.compute_exponential_array(z, 0.09654457318)
    inspector_two_service_times_component_2 = self.compute_exponential_array(z, 0.06436289)
    inspector_two_service_times_component_3 = self.compute_exponential_array(z, 0.04846662111)
    workstation_one_service_times = self.compute_exponential_array(z, 0.21718277739)
    workstation_two_service_times = self.compute_exponential_array(z, 0.090150136)
    workstation_three_service_times = self.compute_exponential_array(z, 0.11369346876)
    print(f"All service times for inspectors 1, 2, 3, and workstations 1, 2, 3 have been generated")
    print(f"Initializing clock to 0, the clock times here is only counting seconds")
    print(f"                                        ")
    print(f"                                        ")
    print(f" STARTING SIMULATION ")
```

The figure below shows the initialization of the simulation state before running the while loop. This includes creating an empty list of blocked inspectors. This empty list of blocked inspectors is used to add the blocked inspectors to it, so they could have new arrival events created once the buffers have room for them to add new components. It also has an empty list of blocked workstations. This list is used to add blocked workstations in the simulation, so they could have new arrival events created once new components are added to the buffers. Additionally, there is also a variable called simulation_stopper that is initialized to 3000 so it could be used to stop the while loop if it exceeds 3000 iterations. Further, the run function also creates 5 arrival events for inspectors and workstations at time 0 to kick off the simulation once the while loop starts executing.

```
blocked_inspectors = [] #The array that holds the blocked inspectors to create arrival events
#for when buffers are emptied
blocked_workstations = [] #The array that holds the blocked workstations to create arrival events
#for them when new components are added to buffer.
simulation_stopper = 3000  #A simulation stopper variable used in while loop to track iterations

self.second_inspector.set_next_component_to_make(Component.C3)

#Initializing first Arrival events to be used on start of the while loop (simulation)
self.add_event_notice_to_FEL(self.create_event("arrival",  self.first_inspector, 0))
self.add_event_notice_to_FEL(self.create_event("arrival", self.second_inspector,  0))
self.add_event_notice_to_FEL(self.create_event("arrival", self.workstation_one, 0))
self.add_event_notice_to_FEL(self.create_event("arrival", self.workstation_two, 0))
self.add_event_notice_to_FEL(self.create_event("arrival", self.workstation_three, 0))
counter = 0 #to count iterations numbers, was used in testing only.
```

The figure below shows the last segment in the “run” function before executing the while loop. In this segment, various variables are defined to calculate the heuristics required to analyze the simulation. The first four variables, namely "inspector_one_blocked_time_start," "inspector_one_blocked_time_total," "inspector_two_blocked_time_start," and "inspector_two_blocked_time_total," are used to keep track of the total block time of inspectors 1 and 2 during the simulation. The subsequent three variables, "total_time_workstation_1_working," "total_time_workstation_2_working," and "total_time_workstation_3_working," are employed to calculate the total working time for each of the three workstations, which is then used to compute the idle time for each workstation. Moreover, the code declares variables to track the occupancy of each buffer at every clock change. These values are appended to arrays and are subsequently used to compute the average buffer occupancies. Overall, the figure below shows the code that is setting up the necessary variables to calculate different performance metrics for the manufacturing facility simulation. Refer to the comments found in the code for better understanding.

```
"""
The variables declared below until the while loop are used to calculate the heuristics
required to reflect on the simulation
"""
inspector_one_blocked_time_start = None #Holds the start time of which the first inpector is blocked
#^ This value is used and re-initialized in while loop as inspector 1 gets blocked or unblocked
#and is used to calulate the total block time of inspector 1 through the simulation

inspector_one_blocked_time_total = 0 #Tracks the sum of total block time for inspector 1 in simulation

#Same is done for inspector 2.
inspector_two_blocked_time_start = None
inspector_two_blocked_time_total = 0

#These variables are used to calculate the sum of the working stations working times in the simulation
#done for all 3 workstations to compute later idle time for each workstation (required in heuristics)
total_time_workstation_1_working = 0
total_time_workstation_2_working = 0
total_time_workstation_3_working = 0

#These variables are used to find each buffer's occupancy in each CLOCK Change and adds them to
#the arrays of occupancies below. This is used to calculate average buffer occupancies heuristics
#Note C1WS1 means Component 1 of Workstation 1
last_clock = None
last_clock_buffer_occupancy_C1WS1 = 0
last_clock_buffer_occupancy_C1WS2 = 0
last_clock_buffer_occupancy_C1WS3 = 0
last_clock_buffer_occupancy_C2WS2 = 0
last_clock_buffer_occupancy_C3WS3 = 0


occupancies_C1WS1 = []
occupancies_C1WS2 = []
occupancies_C1WS3 = []
occupancies_C2WS2 = []
occupancies_C3WS3 = []
```

The figure below shows that at the start of while loop, a counter is incremented to keep counts of the number of iterations run (used for testing). After that, the first event of the FEL is removed to run through the while loop and change the clock to the clock of the event in the FEL.

```
while simulation_stopper > 0:
    counter = counter + 1
    first_element = self.FEL.pop(0)
    self.clock = first_element[2]

    if first_element[0] == "arrival":
```

The figure below shows the part of the while loop that is executed if the event (first_element) is an ARRIVAL event (first_element[0]). The first if statement checks if the first element in the event queue is an "arrival" event for an inspector. If so, the code checks which inspector has arrived and if it is available to work. If Inspector 1 has arrived, the code calculates the time it will take to complete the inspection using the service time value stored in a list (The service time used from the list is removed/popped to not be used again). If Inspector 2 has arrived, the code checks which component the inspector is working on and uses the corresponding service time value. Once the completion time for the inspector is calculated, a new "completion" event is created and added to the future event list (FEL). The inspector is then set to busy, and their status is updated accordingly.

If the first element in the event queue is not an inspector arrival event, the code assumes it is a workstation arrival event. The code first checks if the workstation has the necessary components to make a product. If so, the code calculates the service time value for the workstation and updates the total work time for that workstation. A new "completion" event is created for the workstation and added to the FEL. The workstation is set to busy, and its status is updated. If any inspectors were previously blocked due to buffers being full, they are now freed, and their events are added to the FEL with the current clock time. Finally, if the workstation is unable to make the product due to a lack of components, its status is updated, and it is added to a list of blocked workstations.

```
if first_element[0] == "arrival":
    if isinstance(first_element[1], Inspector): #checks if the arrival event is for an inspector
        if first_element[1].get_ID() == 1: #checks if it is inspector 1
            completion_time = self.clock + inspector_one_service_times.pop(0) #get inspector 1 service time
        elif first_element[1].get_ID() == 2:#checks else if it is inpector 2
            if first_element[1].get_next_component_to_make() == Component.C2: #if it is making component C2
                completion_time = self.clock + inspector_two_service_times_component_2.pop(0) #Get Inspector2 C2 service time
            elif first_element[1].get_next_component_to_make() == Component.C3:#if it is making component C3
                completion_time = self.clock + inspector_two_service_times_component_3.pop(0)# Get Inspector2 C3 service time
        new_event = self.create_event("completion", first_element[1], completion_time ) #creates the completion event accordingly
        self.add_event_notice_to_FEL(new_event) #adds the scheduled completion event to FEL
        first_element[1].set_busy(True) #makes inspector busy until the completion event is reached
        first_element[1].update_inspector_status() #updates the inspector status
    else: #else it is a workstation arrival event
        if first_element[1].make_product(self.clock):# checks if workstation is able to make product
            if first_element[1].get_ID() == 1: #if it is the first workstation
                w_1_st = workstation_one_service_times.pop(0) #get workstation 1 service time to compute completion time
                completion_time = self.clock + w_1_st #compute future completion time
                total_time_workstation_1_working = total_time_workstation_1_working + w_1_st #update total work time for workstation
            elif first_element[1].get_ID() == 2: #if it is the second workstation
                w_2_st = workstation_two_service_times.pop(0) #get workstation 2 service time to compute completion time
                completion_time = self.clock + w_2_st  #compute future completion time
                total_time_workstation_2_working = total_time_workstation_2_working + w_2_st#update total work time for workstation
            elif first_element[1].get_ID() == 3:#if it is the third workstation
                w_3_st = workstation_three_service_times.pop(0)#get workstation 3 service time to compute completion time
                completion_time = self.clock + w_3_st #compute future completion time
                total_time_workstation_3_working = total_time_workstation_3_working + w_3_st#update total work time for workstation
            new_event = self.create_event("completion", first_element[1], completion_time )#create the completion event accordingly
            first_element[1].set_busy(True) #make workstation busy
            first_element[1].update_workstation_status() #update workstation status
            self.add_event_notice_to_FEL(new_event) #add the completion event to the FEL
            for f in blocked_inspectors: #now that workstation used the components in buffers
                new_f = (f[0], f[1], self.clock) #free the blocked inspectors to add their components to the buffer
                self.add_event_notice_to_FEL(new_f) #add the blocked inspectors events to FEL
            blocked_inspectors = [] #re- initialize blocked_inpectors to be empty
        else: #if the workstation cannot make the product (components are not available)
            first_element[1].update_workstation_status() #update the workstation status
            blocked_workstations.append(first_element) #add the workstation to blocked_workstation list
```

The figure below shows the part of the while loop that is executed if the event (first_element) is a COMPLETION event (first_element[0]). The code first checks if the event is a completion event, meaning that an inspector has completed inspecting a component. It then checks if the inspector that completed the inspection is the first or the second inspector. If it is the first inspector, it sets the “addition_component” to component C1, and if it is the second inspector, it generates a random number 2 or 3 to decide on the component the inspector should inspect next, either C2 or C3. The inspector is then set to know the component to inspect.

The code then checks if the inspector can send the inspected component to the target buffer. If it can, it checks which inspector has sent the component and calculates the blocked time total for that inspector if it was blocked at some point. The code then adds a new event notice to the future event list to have the inspector start inspecting a new component after sending to the buffer. It also updates the status of the inspector to "Sent Component to Buffer." Also, if there are blocked workstations, the code adds arrival events to the future event list with the current time clock for the blocked workstations to start processing the newly added components to the buffers.

If the inspector is blocked and cannot send the inspected component to the target buffer because it is full, the completion event of the inspector is added to the “blocked_inspectors” list. If it is the first inspector, the blocked time start of it is re-initialized with the current clock time, and if it is the second inspector, the blocked time start of it is re-initialized with the current clock time.

If the event is not a completion event from an inspector, it means that a workstation has completed processing a component, and it increments the product count and updates its status. It then creates a new arrival event for the workstation with the current clock time and adds it to the future event list. This will make the workstation start working on creating the next product if the component are available.

Refer to the code snippet in the figure below and the comments for further understanding.

```
elif first_element[0] == "completion": #else if the event is a completion event
    if isinstance(first_element[1], Inspector): #it checks if the event is from an inspector
        if first_element[1].get_ID() == 1: # If it is the first inspector
            addition_component = Component.C1 #The component to be inspected by the inspector is component 1
            first_element[1].set_next_component_to_make(addition_component) #inspector is set to know the component to inspect
        else: #else if it is the second inspector
            x = self.generate_random_2_or_3() #The code generate a random number 2 or 3 (to decide on the component Inspector shall inspect)
            if x == 2:  #if the random number is 2
                addition_component = Component.C2 # the additional component to inspect is C2
                first_element[1].set_next_component_to_make(addition_component) #the inspector is set know the component to inspect
            elif x == 3: #if the random number is 3
                addition_component = Component.C3 #the additional component to inspect is C3
                first_element[1].set_next_component_to_make(addition_component) #the inspector is set know the component to inspect
        if first_element[1].send_target_buffer(addition_component, self.clock): #then the inspector checks if it can send the component to the target b
            if first_element[1].get_ID() == 1: # if inspector 1 sent it to the buffer
                if inspector_one_blocked_time_start == None: #checks if it wasnt been blocked at a certain time
                    pass
                else: #if it was blocked at a time
                    inspector_one_blocked_time_total = inspector_one_blocked_time_total +  (self.clock - inspector_one_blocked_time_start) # the blocke
                    inspector_one_blocked_time_start = None #and now there is no bloked time as it is not blocked
            else: #second inspector
                if inspector_two_blocked_time_start == None: #checks if it wasnt blocked at a certain time
                    pass
                else:# if it was blocked
                    inspector_two_blocked_time_total = inspector_two_blocked_time_total + (self.clock - inspector_two_blocked_time_start) # the blocked
                    inspector_two_blocked_time_start = None#and now there is no blocked time as it is not blocked

            self.add_event_notice_to_FEL(self.create_event("arrival", first_element[1] , self.clock)) #adds a new event notice to have the inspector st
            first_element[1].set_status("Sent Component to Buffer") #updates the status of the inspector
            for f in blocked_workstations: #if there are blocked workstation
                new_f = (f[0], f[1], self.clock) # these workstations are now free since new components have been added to buffers
                self.add_event_notice_to_FEL(new_f) #arrival events with current time clock are sent to FEL
            blocked_workstations = [] #no more blocked workstations
        else: #if inspector is blocked and cannot send to the target buffer (because it is full)
            blocked_inspectors.append(first_element) #the inspector completion event is blocked and is added to the blocked_inspectors list
            if first_element[1].get_ID() == 1: #if it is the first inspector
                inspector_one_blocked_time_start = self.clock #the blocked time start of it is re-initialized with the current clock time
            elif first_element[1].get_ID() == 2: #else if it is the second inspector
                inspector_two_blocked_time_start = self.clock # the blocked time start of it is re-initialized with the current clock time
    else: #a workstation instance
        first_element[1].increment_product_count() #the workstation increments the product count
        first_element[1].set_status("Product Created") #updates its status
        new_event = self.create_event("arrival", first_element[1], self.clock ) #creates a new arrival event for the workstation with the current clock
        self.add_event_notice_to_FEL(new_event) #adds the arrival event to FEL
```

The Figure below shows the remaining part of the while loop that is run in each iteration regardless of whether the event was arrival or completion. The "simulation_stopper" variable is decremented by 1 to ensure that the simulation stops at some point. Then, the code checks if the clock has changed since the last iteration. If the clock has changed, the buffer occupancies for each workstation are appended to their respective lists. These lists will be used later to calculate the average buffer occupancy on clock change. The "last_clock" variable is then re-initialized with the current clock value. If the clock did not change, the buffer occupancies for each workstation are updated for the current iteration. After the buffer occupancies are updated, the status of each element in the manufacturing plant is printed using the "self.print_elements_status()" method.

Finally, the while loop checks if the total number of products produced (Product 1, Product 2, and Product 3) is equal to 299. If it is, then the simulation is ended, and a message is printed indicating that there are no more service times available for Inspector 1 to inspect Component 1. The "simulation_stopper" variable is set to 0 to end the simulation.

```
simulation_stopper = simulation_stopper - 1 #simulation stopper is decremented to makesure that the simulation stops at some point
if self.clock != last_clock: #the clock has changed since the last iteration (not events arriving at the same time)
    occupancies_C1WS1.append(last_clock_buffer_occupancy_C1WS1) #update Component 1 Buffer of workstation 1 occupancies list by adding the buffer
    occupancies_C1WS2.append(last_clock_buffer_occupancy_C1WS2) #update Component 1 Buffer of workstation 2 occupancies list by adding the buffer
    occupancies_C1WS3.append(last_clock_buffer_occupancy_C1WS3)#update Component 1 Buffer of workstation 3 occupancies list by adding the buffer
    occupancies_C2WS2.append(last_clock_buffer_occupancy_C2WS2)#update Component 2 Buffer of workstation 2 occupancies list by adding the buffer
    occupancies_C3WS3.append(last_clock_buffer_occupancy_C3WS3)#update Component 3 Buffer of workstation 3 occupancies list by adding the buffer
    last_clock = self.clock #re-initialize last_clock
else: #if the clock didnot change
    last_clock_buffer_occupancy_C1WS1 = self.buffer_C1_WST1.get_occupancy()#update Component 1 Buffer of workstation 1 occupancy in this clock it
    last_clock_buffer_occupancy_C1WS2 = self.buffer_C1_WST2.get_occupancy()#update Component 1 Buffer of workstation 2 occupancy in this clock it
    last_clock_buffer_occupancy_C1WS3 = self.buffer_C1_WST3.get_occupancy()#update Component 1 Buffer of workstation 3 occupancy in this clock it
    last_clock_buffer_occupancy_C2WS2 = self.buffer_C2_WST2.get_occupancy()#update Component 2 Buffer of workstation 2 occupancy in this clock it
    last_clock_buffer_occupancy_C3WS3 = self.buffer_C3_WST3.get_occupancy()#update Component 3 Buffer of workstation 3 occupancy in this clock it

#print out the element status to view the status of each element in the manufacturing plant on each iteration
self.print_elements_status()

#if the products produced add up to 299, This means that 300 Component 1 have been used and this means that we ran out of service times for inpec
#because we only had 300 service times calculated for all inspectors and workstations in the previous milestone
if (self.workstation_one.get_products_count() + self.workstation_two.get_products_count() + self.workstation_three.get_products_count()) == 299:
    #ends the simulation
    print(f"END OF SIMULATION")
    print(f"No more service times are available for Inspector 1 to inspect C1")
    print(f"                                    ")
    print(f"PRINTING REQUIRED HEURISTICS")
    simulation_stopper = 0
```

The code presented in the Figure below displays the last part of the run function that prints and calculates the 12 heuristics required for milestone 3 of the simulation after the while loop has ended and the simulation is completed. These heuristics are computed based on the simulation results.

Firstly, the code calculates the throughput of each product using the calculate_throughput function, which takes the number of products produced by a workstation and the simulation clock time as inputs and returns the throughput of that product. The throughput is then converted from throughput per second to throughput per hour by multiplying it by 3600.

Next, the code prints the clock end time, the number of products produced by each workstation, the throughput per hour of each product, and the total blocked time for each inspector. After that, the code calculates and prints the total idle time for each workstation by subtracting the total time each workstation was working from the current clock time. Additionally, the code prints the average buffer occupancy for all buffers of each workstation using the Average function, which takes a list of the buffer occupancies that was updated at each iteration of the simulation as input.

Finally, the code prints the average time that components spent in each buffer. This is done by calling the get_times_spent_in_buffer function on each buffer, which keeps track of the times spent by each component and adds them to a list (refer to Buffer class). The resulting list is passed to the Average function to compute the average time spent in each buffer.

```
#This is run after the simulation has reached to an end to show the 12 heuristics required for milestone 3
throughput_P1 = self.calculate_throughput(self.clock, self.workstation_one.get_products_count()) #calculate the throughput given the number of p
throughput_P2 = self.calculate_throughput(self.clock, self.workstation_two.get_products_count())#calculate the throughput given the number of pr
throughput_P3 = self.calculate_throughput(self.clock, self.workstation_three.get_products_count())#calculate the throughput given the number of
throughput_P1_per_hour = throughput_P1 * 3600 #converts throughput per second to throughput per hour for product 1
throughput_P2_per_hour = throughput_P2 * 3600 #converts throughput per second to throughput per hour for product 2
throughput_P3_per_hour = throughput_P3 * 3600  #converts throughput per second to throughput per hour for product 3
print(f"clock end time is {self.clock}") # prints the clock end time
print(f"Number of products P1 made is {self.workstation_one.get_products_count()}") #prints the number of products P1 made
print(f"Number of products P2 made is {self.workstation_two.get_products_count()}") #prints the number of products P2 made
print(f"Number of products P3 made is {self.workstation_three.get_products_count()}")#prints the number of products P2 made
print(f"The throughput per hour for Product 1 is {throughput_P1_per_hour}") #prints the calculated throughput per hour for P1
print(f"The throughput per hour for Product 2 is {throughput_P2_per_hour}")#prints the calculated throughput per hour for P2
print(f"The throughput per hour for Product 3 is {throughput_P3_per_hour}")#prints the calculated throughput per hour for P3
print(f"Total blocked time for Inspector 1 in seconds is {inspector_one_blocked_time_total}") #prints the total blocked time for inspector 1 tha
print(f"Total blocked time for Inspector 2 in seconds is {inspector_two_blocked_time_total}")#prints the total blocked time for inspector 2 that
print(f"Total idle time for workstation 1 in seconds is {self.clock - total_time_workstation_1_working}") #prints the total idle time for workst
print(f"Total idle time for workstation 2 in seconds is {self.clock - total_time_workstation_2_working}")#prints the total idle time for worksta
print(f"Total idle time for workstation 3 in seconds is {self.clock - total_time_workstation_3_working}")#prints the total idle time for worksta
print(f"Average buffer occupancy for buffer C1 of workstation 1 is {self.Average(occupancies_C1WS1)}") # prints and finds the average buffer C1
print(f"Average buffer occupancy for buffer C1 of workstation 2 is {self.Average(occupancies_C1WS2)}")# prints and finds the average buffer C1 o
print(f"Average buffer occupancy for buffer C1 of workstation 3 is {self.Average(occupancies_C1WS3)}")# prints and finds the average buffer C1 o
print(f"Average buffer occupancy for buffer C2 of workstation 2 is {self.Average(occupancies_C2WS2)}")# prints and finds the average buffer C2 o
print(f"Average buffer occupancy for buffer C3 of workstation 3 is {self.Average(occupancies_C3WS3)}")# prints and finds the average buffer C3 o
print(f"Average time component C1 spends in buffer C1 of workstation 1 is {self.Average(self.buffer_C1_WST1.get_times_spent_in_buffer())}") #eac
print(f"Average time component C1 spends in buffer C1 of workstation 2 is {self.Average(self.buffer_C1_WST2.get_times_spent_in_buffer())}") #pri
print(f"Average time component C1 spends in buffer C1 of workstation 3 is {self.Average(self.buffer_C1_WST3.get_times_spent_in_buffer())}")
print(f"Average time component C2 spends in buffer C2 of workstation 2 is {self.Average(self.buffer_C2_WST2.get_times_spent_in_buffer())}")
print(f"Average time component C3 spends in buffer C3 of Workstation 3 is {self.Average(self.buffer_C3_WST3.get_times_spent_in_buffer())} ")
```

10.3 Running the simulation

This sub-section will go through the first few outputs of the simulation model, last outputs, and the 12 heuristics printed after running the simulation. Run “Simulation.py” in the delivered project folder to start the simulation. The figure below shows that the run function is triggered in the main method. Please note that the simulation outputs many lines to the terminal, ensure that the terminal scroll back is set to a large number of lines to avoid the terminal from not showing you the entire output.

```
def main():
    simulate = Simulation()
    simulate.run()


if __name__ == "__main__":
    main()
```

The figure below shows the first part of the output before the while loop starts. It shows the results of generating random numbers using the linear congruential generator (LCG) method. The Kolmogorov-Smirnov uniformity test and the autocorrelation test are performed to check the quality of the generated numbers for service times as explained in the above sub-sections (Milestone 2).

```
generating LCG random numbers
seed value = 50, a = 13, c = 9, m = 512
number of values = 300
----------Separator----------
performing kolmogorov smirnov uniformity test on generated data
computed D value = 0.03083333333333338
----------Separator----------
performing autocorrelation test
lag value = 4, M = 73, i = 1
p = 0.03891733530405406
standard deviation of the estimator = 0.03481897485076646
z = 1.117704799490884
----------Separator----------
All service times for inspectors 1, 2, 3, and workstations 1, 2, 3 have been generated
Initializing clock to 0, the clock times here is only counting seconds


 STARTING SIMULATION
```

Before the simulation starts, as mentioned previously, The FEL is populated 5 arrival events at time 0 for all inspectors and workstations in the manufacturing facility as seen in the figure below.

```
#Initializing first Arrival events to be used on start of the while loop (simulation)
self.add_event_notice_to_FEL(self.create_event("arrival",  self.first_inspector, 0))
self.add_event_notice_to_FEL(self.create_event("arrival", self.second_inspector,  0))
self.add_event_notice_to_FEL(self.create_event("arrival", self.workstation_one, 0))
self.add_event_notice_to_FEL(self.create_event("arrival", self.workstation_two, 0))
self.add_event_notice_to_FEL(self.create_event("arrival", self.workstation_three, 0))
```

The first arrival event of inspector one in the FEL is processed in the while loop, thus the first inspector starts inspecting at time 0 as can be seen in the Figure below showing the first printed simulation model status (Note that other elements are still waiting for arrival events).

```
**************** Simulation Model Status ****************

Clock time in seconds is 0
Inspector 1 status: Inspecting
Inspector 2 status: Waiting for Arrival Event

buffer C1 for workstation 1 has 0 C1 components
buffer C1 for workstation 2 has 0 C1 components
buffer C1 for workstation 3 has 0 C1 components
buffer C2 for workstation 2 has 0 C2 components
buffer C3 for workstation 3 has 0 C3 components

Workstation 1 status: Waiting for Arrival Event
Workstation 2 is status: Waiting for Arrival Event
Workstation 3 is status: Waiting for Arrival Event

number of product P1 made: 0
number of product P2 made: 0
number of product P3 made: 0
*********************************************************
```

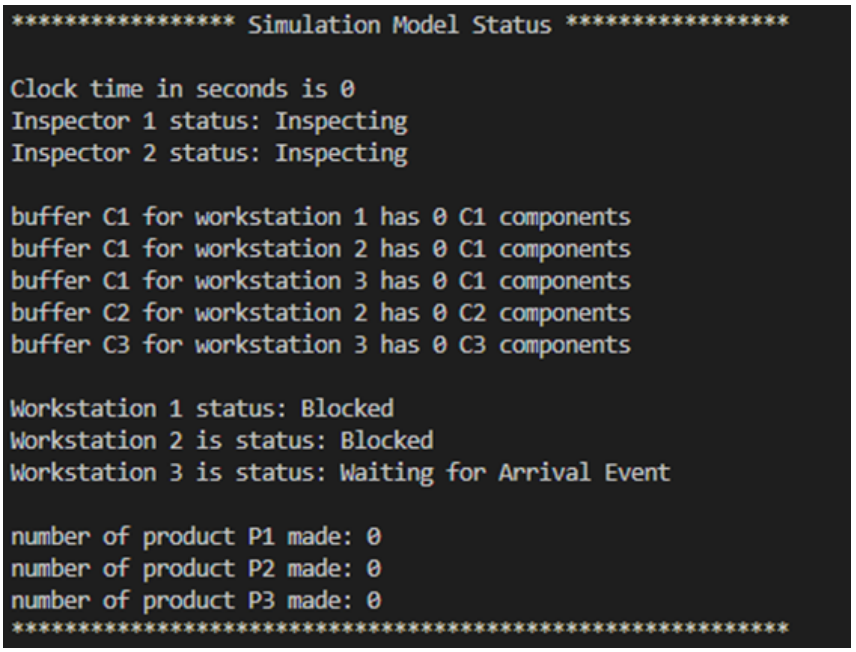
On the arrival event of the first inspector seen in the figure above, the first inspector starts inspecting, thus a completion event shall be added to the FEL as seen in the while loop, with the completion time based on Inspector 1 service times. The next event in FEL is still at time 0 for inspector 2 to start inspecting. The figure below shows the simulation status once that event is processed (Note that Inspector 2 is now also inspecting while other elements are waiting for arrival event).

```
***************** Simulation Model Status *****************

Clock time in seconds is 0
Inspector 1 status: Inspecting
Inspector 2 status: Inspecting

buffer C1 for workstation 1 has 0 C1 components
buffer C1 for workstation 2 has 0 C1 components
buffer C1 for workstation 3 has 0 C1 components
buffer C2 for workstation 2 has 0 C2 components
buffer C3 for workstation 3 has 0 C3 components

Workstation 1 status: Waiting for Arrival Event
Workstation 2 is status: Waiting for Arrival Event
Workstation 3 is status: Waiting for Arrival Event

number of product P1 made: 0
number of product P2 made: 0
number of product P3 made: 0
***********************************************************
```

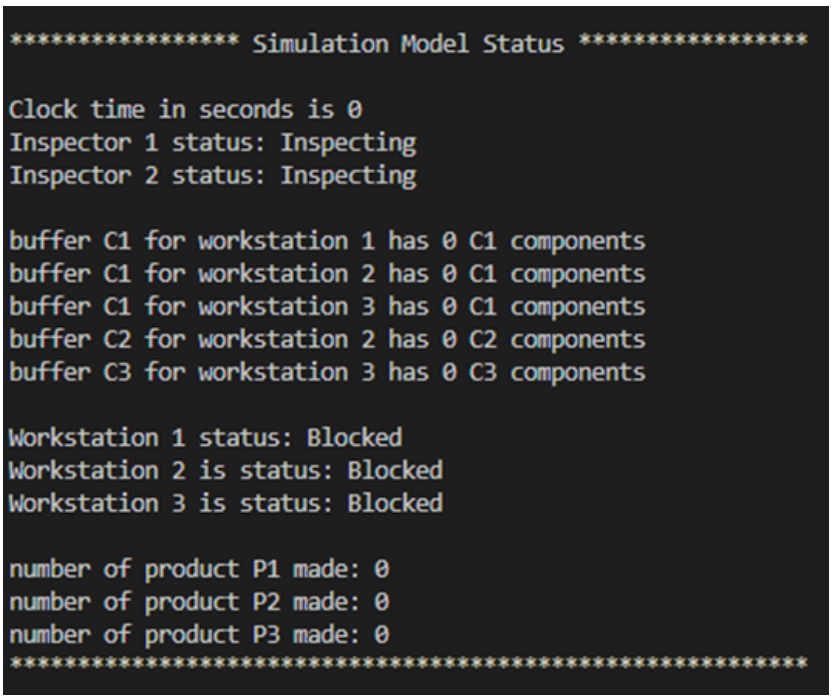
The Figure below shows the simulation model status after receiving the first arrival event of workstation 1 still at time 0 (Note that its status changes to blocked as there are still no components available in buffer C1 of workstation 1).

```
***************** Simulation Model Status *****************

Clock time in seconds is 0
Inspector 1 status: Inspecting
Inspector 2 status: Inspecting

buffer C1 for workstation 1 has 0 C1 components
buffer C1 for workstation 2 has 0 C1 components
buffer C1 for workstation 3 has 0 C1 components
buffer C2 for workstation 2 has 0 C2 components
buffer C3 for workstation 3 has 0 C3 components

Workstation 1 status: Blocked
Workstation 2 is status: Waiting for Arrival Event
Workstation 3 is status: Waiting for Arrival Event

number of product P1 made: 0
number of product P2 made: 0
number of product P3 made: 0
***********************************************************
```

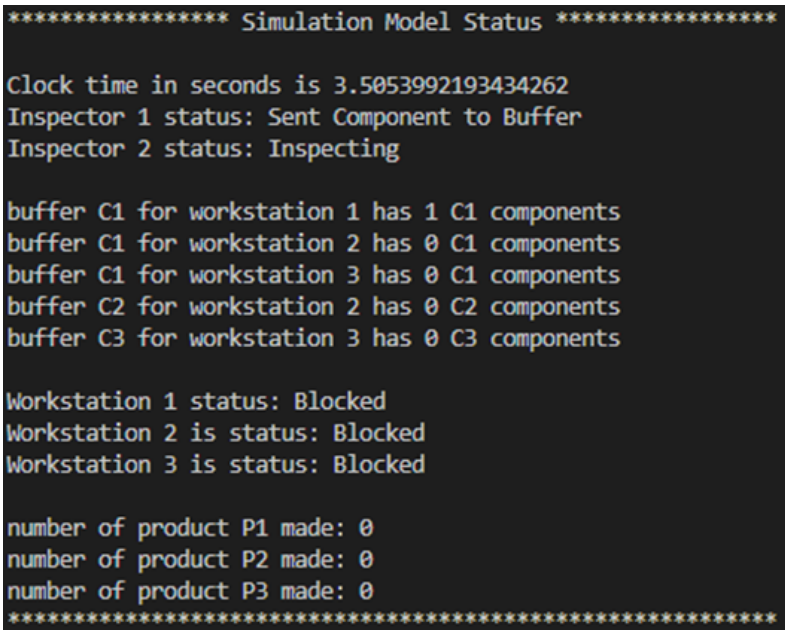
The Figure below shows the simulation model status after receiving the first arrival event of workstation 2 still at time 0 (Note that its status changes to blocked as there are still no components available in buffers C1 and C2 of workstation 2).



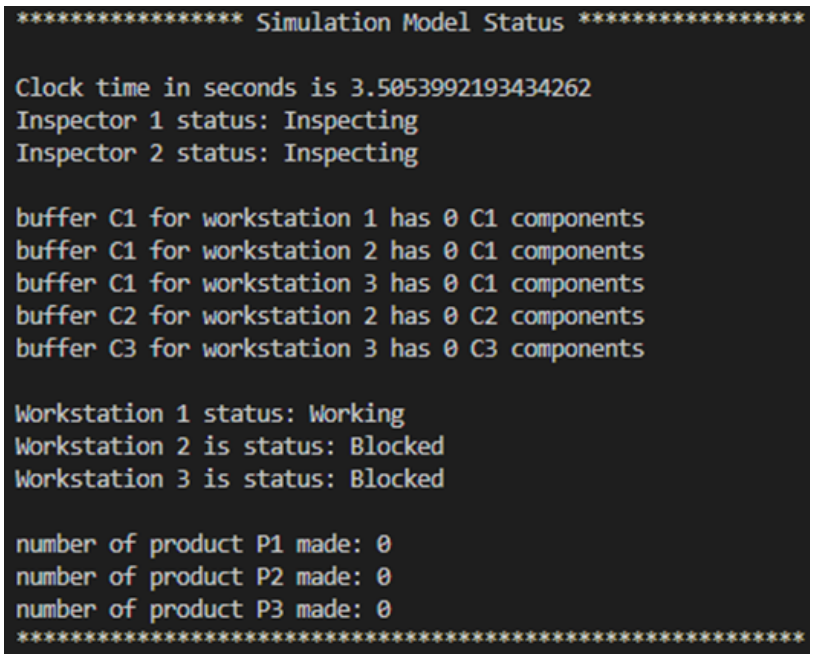
The Figure below shows the simulation model status after receiving the first arrival event of workstation 3 still at time 0 (Note that its status changes to blocked as there are still no components available in buffers C1 and C3 of workstation 3).



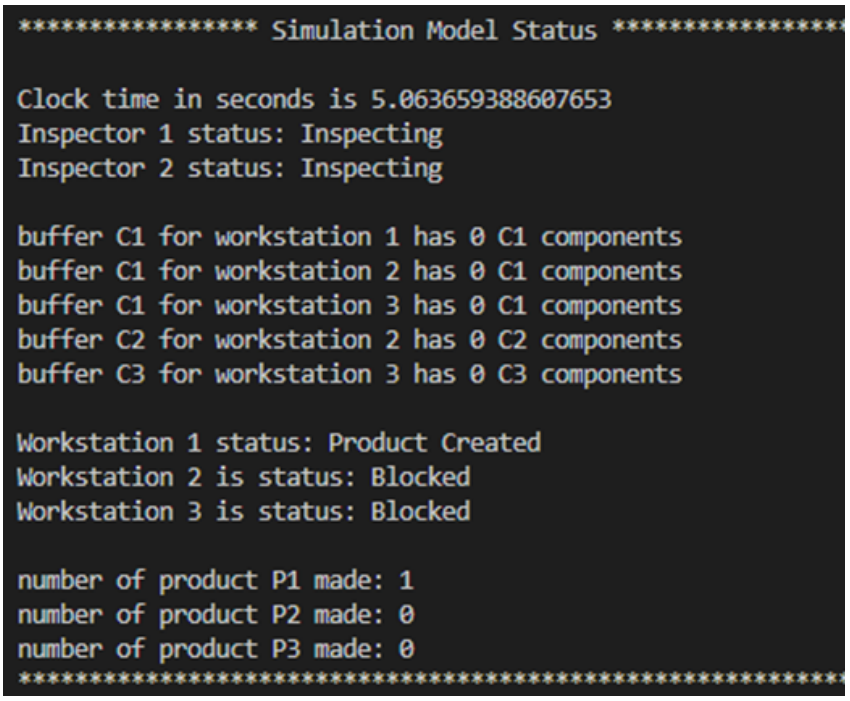
The figure below shows the arrival of the first completion inspection event of Inspector 1 at time 3.51 (Note that buffer C1 of workstation 1 now has 1 component, and that inspector 1 status is now “Sent component to buffer”).



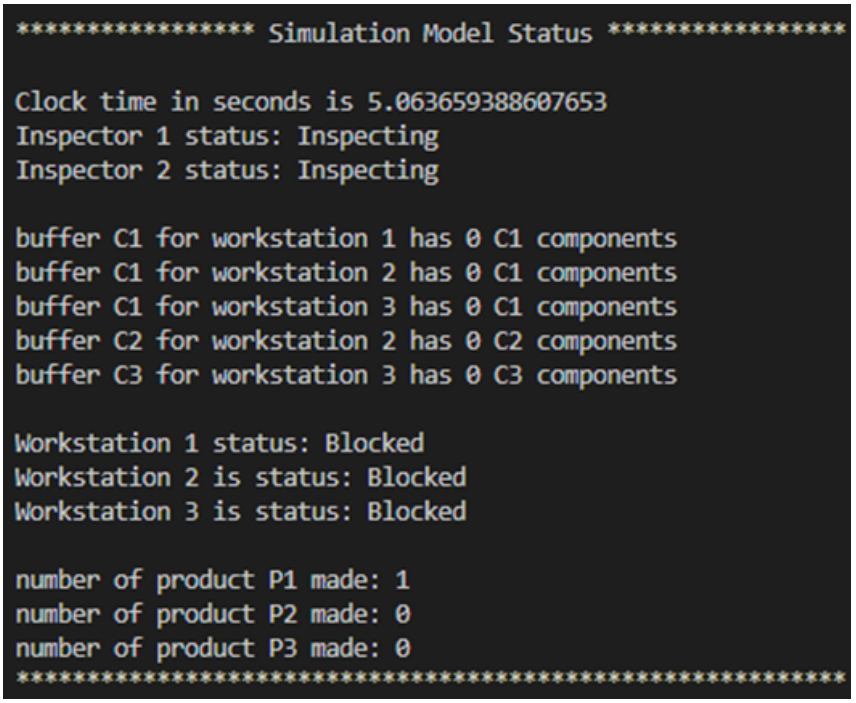
As mentioned previously in the while loop explanation, on completion events of inspectors, blocked workstation arrival events are re-added to FEL at the current clock (3.51). Thus, after they are processed in the while we see the status in the figure below (Note that Inspector has already started inspecting again, workstation 1 has started working, and the components of C1 of workstation 1 have been used by the workstation).



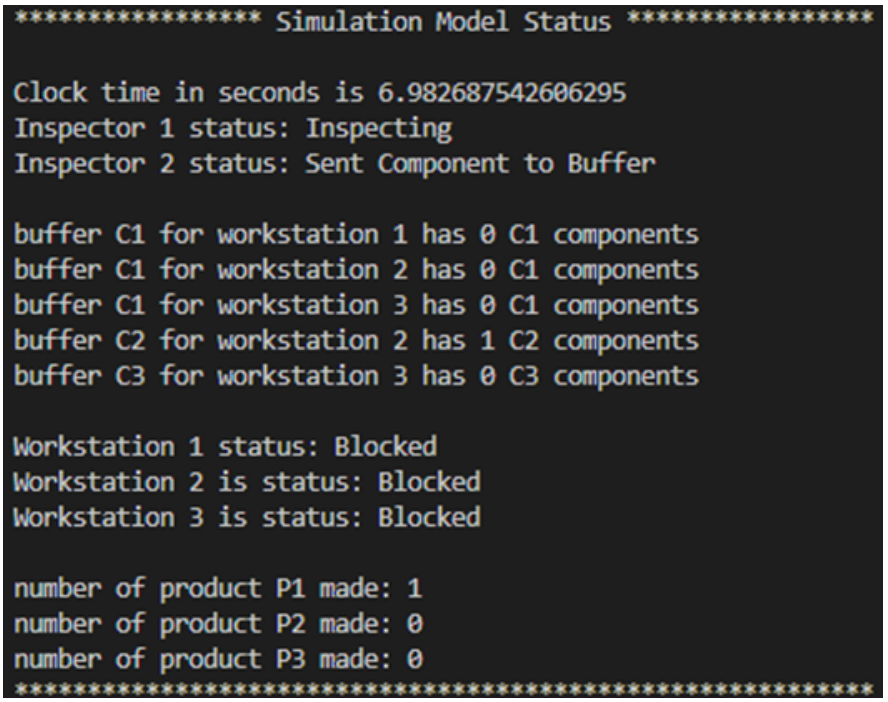
The figure below shows the simulation model status at clock time 5.06 after workstation 1 has finished working on making product P1(Note that workstation 1 status shows “Product Created” and number of product P1 made is now 1).



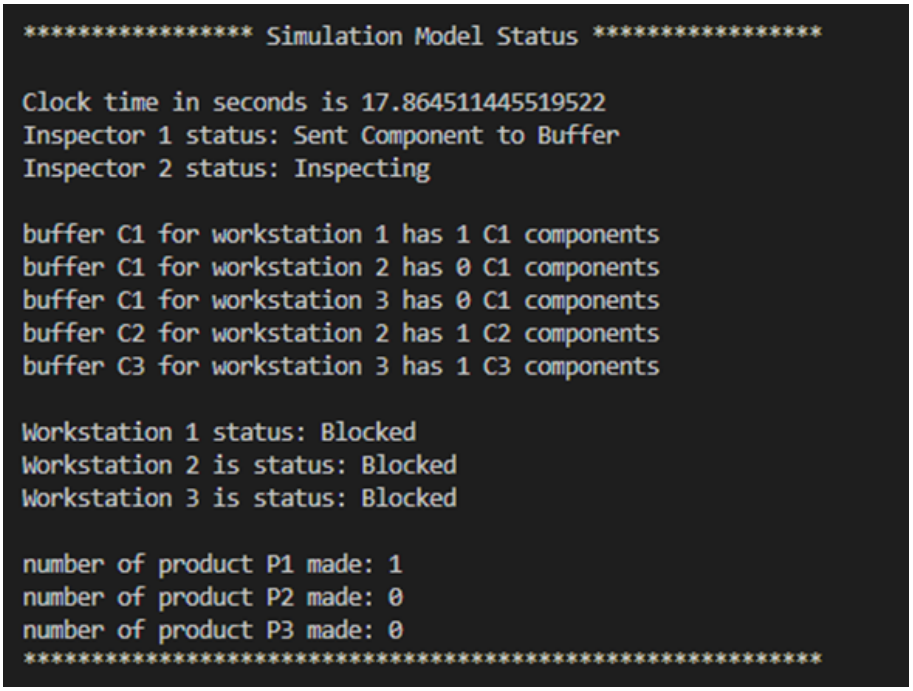
The figure below shows that at the same clock time workstation 1 status is change to blocked as there no more components in buffer C1 of workstation 1.



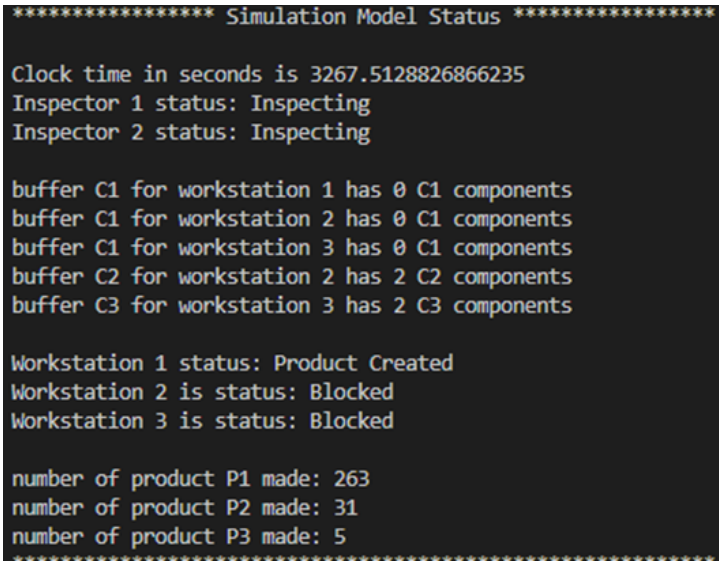
The figure below shows the completion event of inspector 2 at time 6.98. (Note inspector 2 status and buffer C2 of workstation 2).



The figure below shows the completion event of inspector 1 at time 17.7. (Note inspector 1 status and buffer C1 of workstation 1).



Finally, the figure shows the last printed simulation model status after the simulation is run. The values were expected as inspector 1 has lower average inspection service time than the other inspectors, and inspector 1 sending components to buffer algorithm favors Component 1 buffer of workstation 1 than the other workstations' Component 1 buffers. Furthermore, the first workstation has lower average service times than all other workstation. Therefore, in the figure below we see that there much more products 1 created than product 2 and product 3.



Additionally, at the end the 12 heuristics required are printed as seen in the Figure below.