Winter-2022

SYSC 5001: Discrete Simulation or Modelling

Course Project: Study of A Manufacturing Facility

Deliverable 4: New Operating Policy and Project Report

Submitted by: Aziz Al-Najjar

Student ID: 101244840

Contents

Tables and Figures:

Introduction:

In this last deliverable, we are required to introduce an alternative operating policy that aims to improve the system's overall performance. We are asked to explain the new policy's approach and compare it with the initial operating policy described in the project.

As this deliverable is considered the final project report, we will start by formalizing the problem and describing the project plan and objectives. Afterward, in the model conceptualization and translation section, we will describe the model that will be used and include flow charts of the code. Next, we will describe the approach used to generate inputs from given data to the model and include tests and histograms needed to determine each dataset distribution. Moreover, we will use the generated random variates as an input to the model and verify and validate the model with the method described in the book. After validating the model, we will show the results of the production runs and discuss the possible confidence interval calculation, which might contain real-world values. Finally, we will introduce the alternative policy used, including the reasons behind selecting this policy and discuss and compare the results from the new policy with the base policy described in the project manual.

Background information and Problem Formulation:

This project will study and simulate a manufacturing facility's behavior that consists of three workstations W1, W2, and W3. Each workstation is responsible for processing and assembling a specific product, P1, P2, and P3. Each product needs particular components to be assembled. P1 needs C1, P2 needs C1 & C2, and P3 needs C1 & C3. The components come from two inspectors, IN1 and IN2. IN1 is responsible for generating C1, and IN2 is responsible for randomly generating C1 and C2. Before each workstation, there is a buffer for each required component with a maximum capacity of two. When a buffer is reached maximum capacity, the related inspector will be idle until there is an opening. As inspector one supplies C1 for W1, W2, and W3, it will send C1 to the smallest queue (buffer); if all queues have the same component's amount, W1 has the highest priority, then W2 and W3 with the lowest. The assembly will not begin until all components needed are available.

After investigating the manufacturing system, the question that needs to be asked is: **How are the performance metrics affected by such a system? Such as "Product Throughput," "a workstation is busy probability," "average buffer occupancy of each buffer," and "the probability that each inspector is idle," Is changing the priority order for inspector 1 going to improve the performance metrics?**

Objectives and Overall Project Plan:

This project aims to use the provided data and a high programming language such as Python to develop a simulator for the manufacturing system that can keep track and report the following:

- Average time needed for assembling each product (response time)
- Number of products produced per unit time
- Probability each workstation is busy
- Probability each buffer is fully occupied
- Probability each inspector is idle
- Finding the best IN1 priority order
- Is there any better Operating policy?

Firstly, we will be able to reach this objective by developing a model that defines how the system components interact and the main system structure. Afterward, translating the model into a software code as smaller functions. Later, using the data provided to generate an input model and generate arrivals. Next, we need to debug and verify if the model built is the required model working correctly. Then we validate if the model is outputting expected data. Lastly, we compare the results of the system with an alternative approach.

Model Conceptualization and Translation:

We will first start by defining the system components.

- System Components:

System state: - Buffers: BC11(t), BC12(t), BC13(t), BC2(t), BC3(t).

- Workstations' is processing (1 or 0): Pr1(t), Pr2(t), Pr3(t).

- Inspecting: IN1(t), IN2(t)

Entities: The components: C1, C2, C3, (we can consider the products as served components)

Events: - Arrival from IN1 (A1), Arrival from IN2 (A2).

- Departure from Workstations (WD1), (WD2), (WD3)

Event notices: - (A1, C1, t), arrival of C1 event at future time t;

- (A2, C2, t), arrival of C2 event at future time t;

- (A2, C3, t), arrival of C3 event at future time t;

- (WD_i, P_i, t), departure of P_i

Activities: - Inspecting time for IN1, and IN2. (can be considered as interarrival time)

- Processing time for W1, W2, and W3

Delay: time waiting in the buffer.

Next, we will identify general concepts and actions necessary for the model and for generating results

- After Inspecting time is over, the component will go to the corresponding buffer if the buffer is occupied, the Occupied Buffer Counter (OBC_i) will increase.
- Response time is from the arrival of the component from the inspector till the end of production
- After a product is produced, a product counter (PC_i) will increase
- Throughput = $\frac{PC_1+PC_2+PC_3}{Total\ Time}$
- Probability a Workstation is Busy = $\frac{Total\ Busy\ time\ for\ W_i\ (total\ service\ time)}{Total\ Time}$
- Average $Buffer_i$ Occupancy = $\frac{OBC_i}{PC_i}$
- Probability $Inspector_i$ is blocked = $\frac{Total\ Block\ time_{\ i}}{Total\ Time}$

It is always a better idea to translate the concept into flowcharts before starting writing codes. Following are the functions that need to be developed to simulate the system. The functions are for the main system events, such as the arrival from each inspector and the departure from the workstations.

End of Inspecting 2 Event flowchart

Figure 1 shows the process that will be followed when inspector 2 is ready to send components

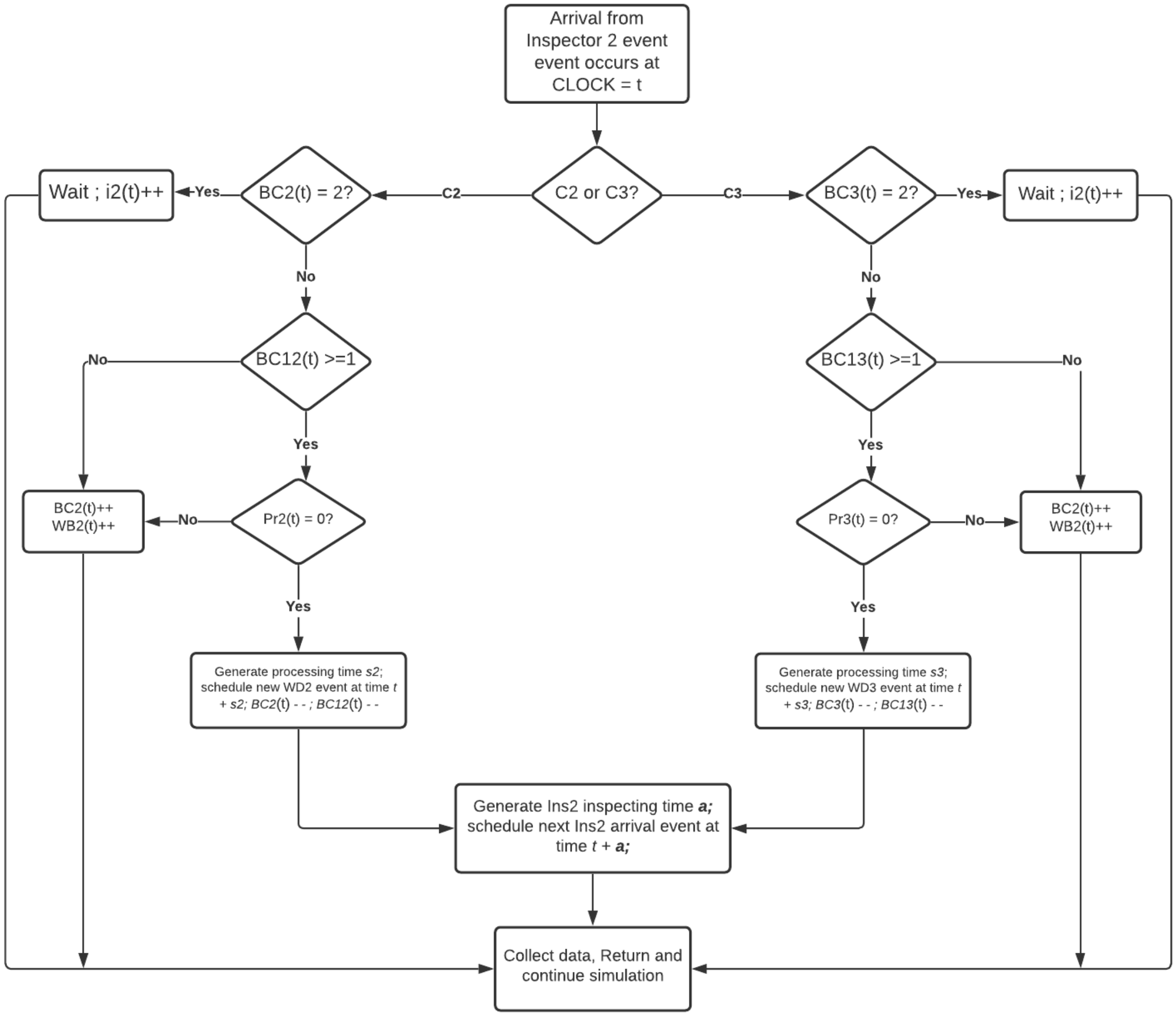


Figure 1 End of Inspecting 2 Event flowchart

Figure 2 shows the function responsible for processing components coming from inspector 1. Note that it has to follow the priority order, W1, W2, then W3.

End of Inspecting 1 event flowchart

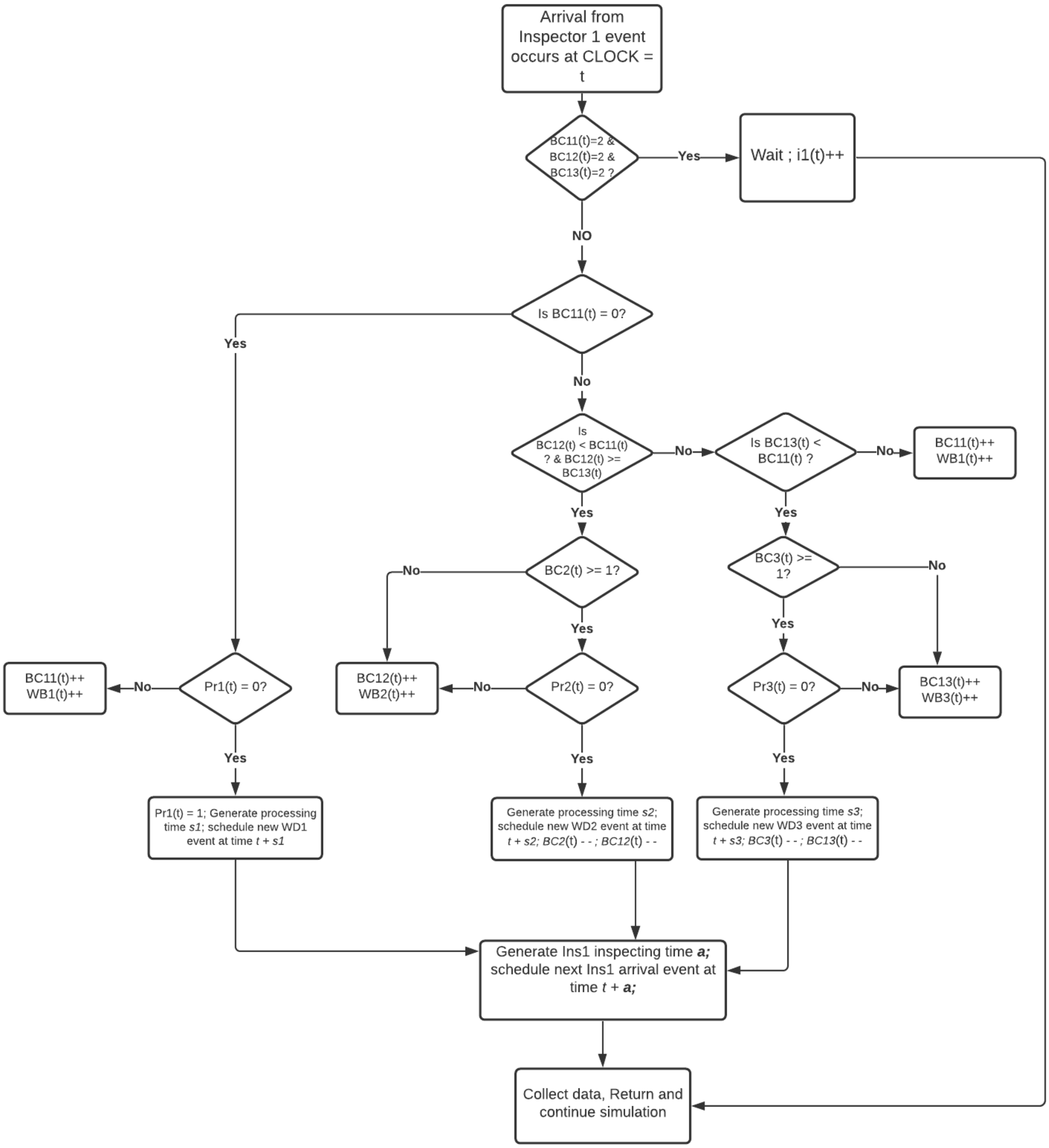


Figure 2 End of Inspecting 1 event flowchart

Departure event flowchart

Figure 3 shows the departure function behavior for workstation 1

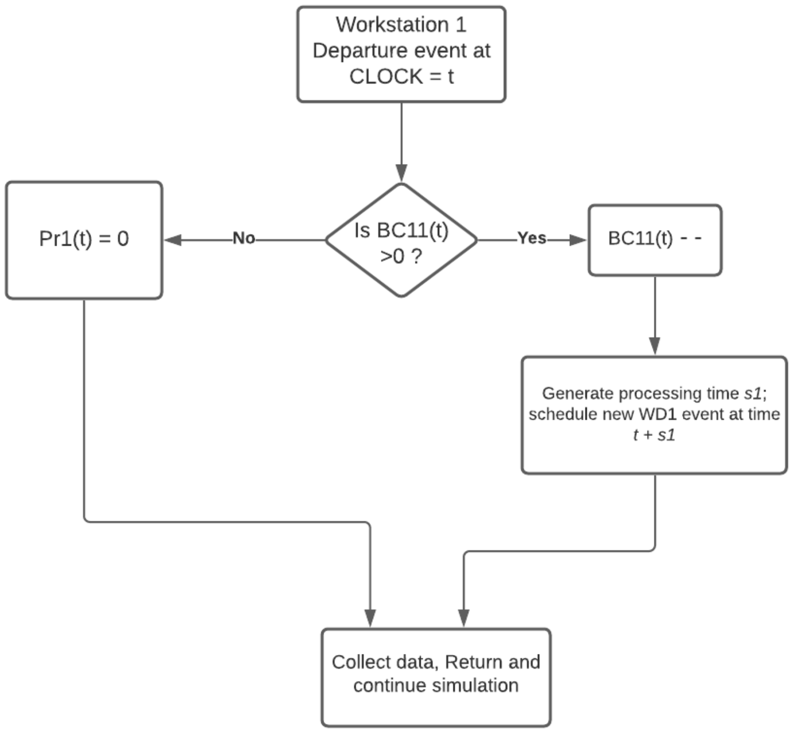


Figure 3 Workstation 1 Departure event flowchart

Figure 4 shows the departure function behavior for workstations 2&3

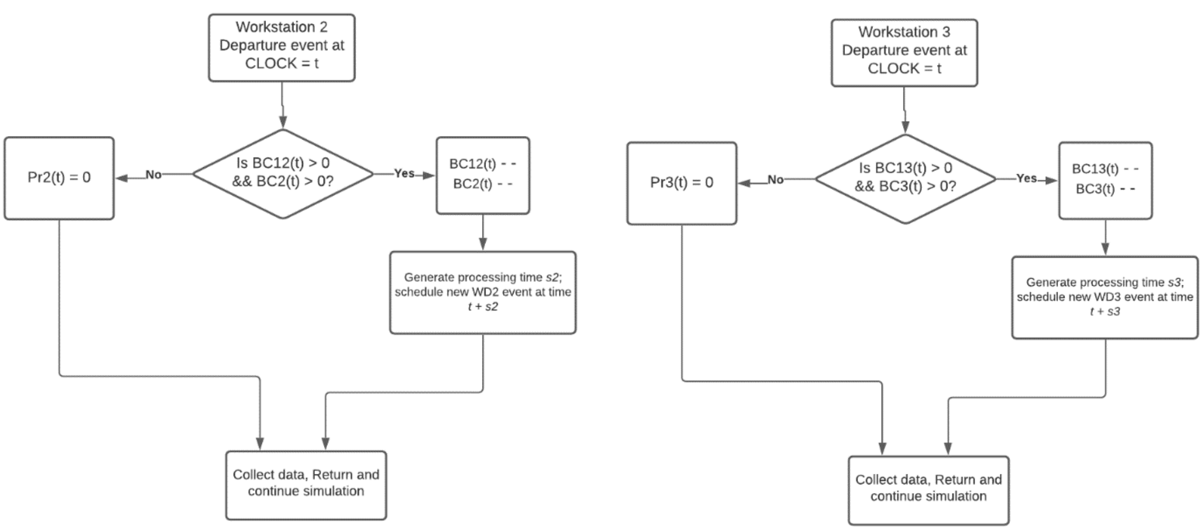


Figure 4 Workstation 2&3 Departure event flowcharts

Input Modelling and Generation:

This part of the report focuses on investigating the given datasets for the inspectors and workstations. We will start by using the histogram technique for identifying the distribution for each dataset. Afterward, the Q-Q plots and chi-square goodness of fit test were used to evaluate the determined distribution. After Identifying the distribution and performing the test, we will describe the steps used to generate inputs based on the chosen distribution

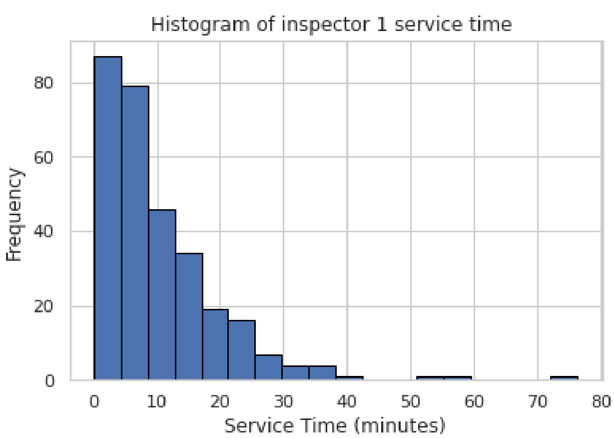
Step 1: Histograms (frequency distributions)

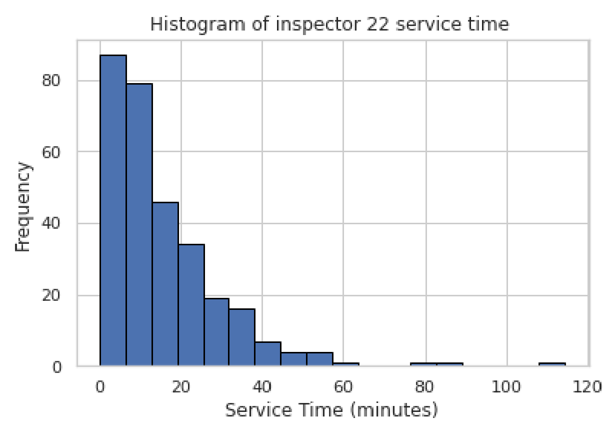
To identify a distribution for each dataset, we will perform the histogram technique for each given data separately, and based on the histogram's shape, we can guess the distribution.

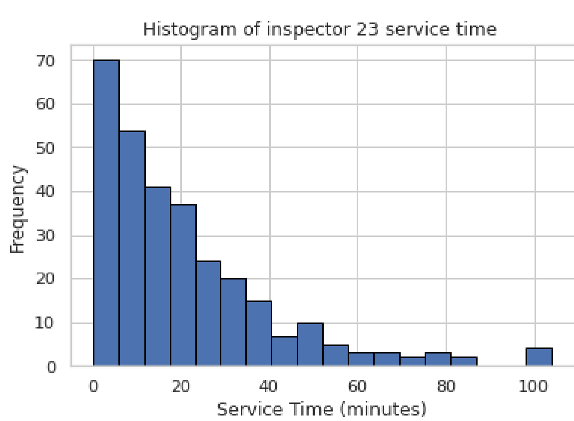
With the help of NumPy and matplotlib libraries, we can perform a loop that will go through all given dataset files to plot the histogram find the mean and max for each dataset.

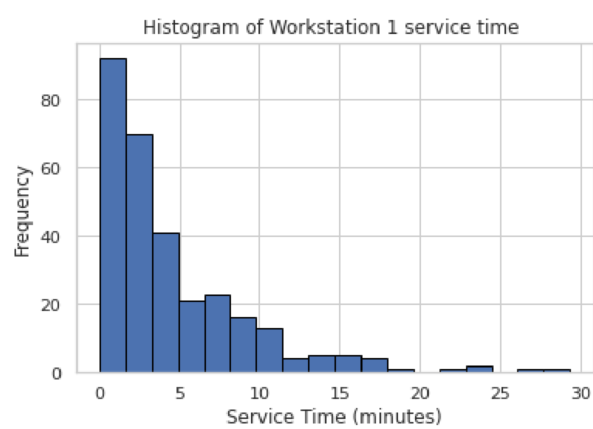
Note: number of bins is chosen to be the square root of the number of samples + 1

The Output and identified distributions were as follows:









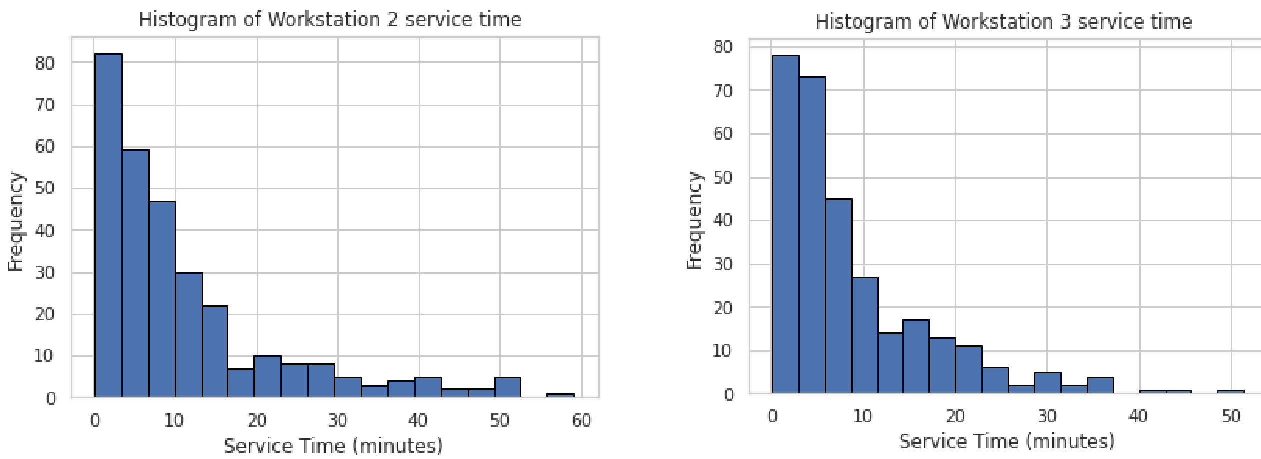


Figure 5 Service and Inspecting times histogram

Table 1 Distribution identification table

	INSP1	INSP22	INSP23	WS1	WS2	WS3
Max Service time	76.28	114.43	104.02	29.38	59.08	51.42
Mean Service time	10.36	15.54	20.63	4.60	11.10	8.80
Identified Distribution	Exponential	Exponential	Exponential	Exponential	Exponential	Exponential

Step 2: Evaluate identified distribution with Q-Q plots:

As all histograms above follow an exponential distribution, let's evaluate the identified distributions using Q-Q plots. We will use the help of `statsmodels.api` and `sm.qqplot` function to plot the QQ plot.

The output is as follows:

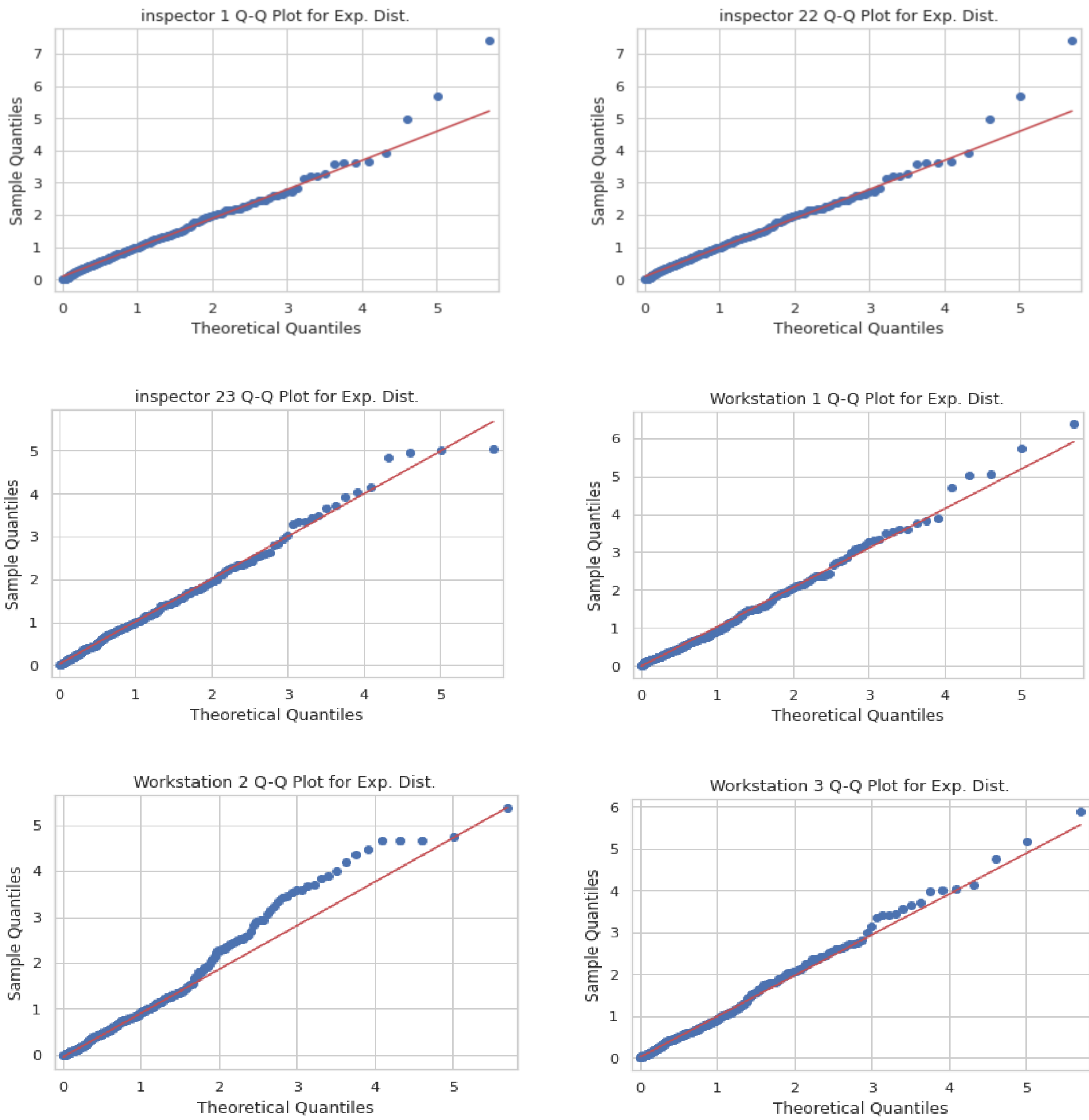


Figure 6 Dataset and exponential distribution Q-Q plots

Comment: The Q-Q plot for all data roughly follows a straight line; hence we accept the exponential distribution.

Step 3: Chi-Square goodness of fit test:

To ensure that the identified exponential distribution is correct, we will perform the Chi-Square goodness of fit test with the help of `scipy.stats.stats` library. We perform a loop that will go through all samples and report the chi-square along with the critical value. If the chi-square is less than the critical value with alpha = 0.05 and the degree of freedom of bins - 1 -1, we will accept the hypotheses and exponentially distribute the data.

The output is as follows:

```
H0: The data is exponentially distributed.
H1: The data is not exponentially distributed.
```

Table 2 Chi-Square test table

	INSP1	INSP22	INSP23	WS1	WS2	WS3
Chi-Square test result	17.027	18.03	18.41	25.55	20.35	22.37
critical value for 0.95 and df = bins - 2	26.296	26.296	26.296	26.296	26.296	26.296
Hypothesis	H0, Accept	H0, Accept	H0, Accept	H0, Accept	H0, Accept	H0, Accept

Step 4: Random Number Generator (RNG)

To generate random numbers we will use the Linear Congruential Method (LCM) with a=1591, c = 459, m = (2^{12}) and seed = 123456789. These parameter values were selected after performing the tests in the next step.

```
#We will use the Linear Congruential Method to generate random numbers and store it in a txt file

seed = 123456789
a = 1591
c = 459
m = (2**12)
n = 1000
counter = 0
outFile = open("lcm.txt", "w")
x_i = seed
while counter < n:
  x_i = (a*x_i + c) % m
  r_i = str(x_i/m)
  outFile.write(r_i + "\n")
  counter+=1
outFile.close()
```

Figure 7 Code for RNG

The code above generates 1000 random numbers using LCM.

Step 2: Tests for Random Numbers:

Test. 1: testing for uniformity:

We will use the Chi-square test of the RNG with uniform distribution for testing for uniformity.

The used code is as follows:

```python
#now after generating the random numbers, lets run some tests

print('chi-square test to test for uniformity:')
print('H0: The data is uniformily distributed.')
print('H1: The data is not uniformly distributed.\n')

lcm = np.loadtxt('lcm.txt', unpack = True)
chi_sq_value = 0.0
expected = [len(lcm)/10.0]
o_i = plt.hist(lcm, bins = 10, edgecolor='black')
o_i = o_i[0].tolist()

chisquare = stats.chisquare(f_obs=o_i, f_exp = expected)
crit = stats.chi2.ppf(q=0.95, df= len(o_i)-1)

print(f' chi-square is {round(chisquare.statistic,3)} | critical value is {round(crit,3)}')
if (chisquare.statistic <= crit):
    print('  H0, accept the hypotheses \n')
else: print('H1, reject the hypotheses')
```

Figure 8 RNG Chi-square test code

The output were as follows:

```
chi-square test to test for uniformity:
H0: The data is uniformly distributed.
H1: The data is not uniformly distributed.

 chi-square is 0.3 | critical value is 16.919
  H0, accept the hypotheses
```

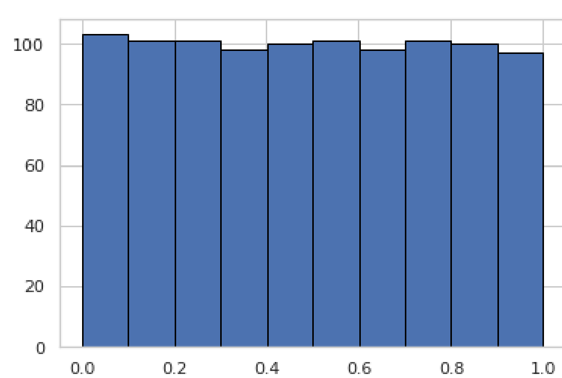


Figure 9 RNG Histogram

Comment: The Chi-Square shows that RNG is following Uniform distribution as expected.

Test. 2: testing for independence:

Using the Autocorrelation test, we can determine if the generated random numbers are dependent. We will perform the autocorrelation test using 100 different lag values and calculate the average. The closer the total autocorrelation average to 0, the more independent the RNG is.

The used code was as follows:

```python
#now test of independence
print('Autocorrelation Test: ')
print('H0 : pil = 0 \nH1 : pil!= 0')
x_av = np.mean(lcm)
x = lcm
c = lambda L: np.mean([(x[i]-x_av)*(x[i+L]-x_av) for i in range(n-L)])
r = lambda L: c(L)/c(0)
corr = [r(L) for L in range(0,100)]
print(f'Average correlation for L 0 to 100 :{round(np.mean(corr),5)}')
correlation = plt.plot(corr)
correlation = plt.title('Autocorrelation for different lag values')
correlation = plt.xlabel('L values')
correlation = plt.ylabel('correlation')
if(np.mean(corr) < 0.01): print('H0, accept the hypothesess, independent  \n')
else: print('H1, reject the hypothesess')
```

Figure 10 Autocorrelation test code

The output were as follows:

```
Autocorrelation Test:
H0 : ρil = 0
H1 : ρil!= 0
Average correlation for L 1 to 100 : 0.00013
H0, accept the hypothesess, independent
```

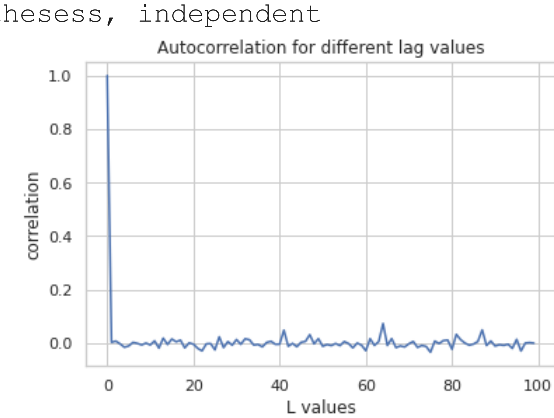


Figure 11 Autocorrelation for different lag values

Comment: As the result above shows, the average autocorrelation result is minimal, concluding that the RNG is independent and ready to generate inputs.

Step 3: Random variate generator

From part 1, we know that all datasets follow an exponential distribution. Hence, we will use the generated random numbers and the inverse-transform algorithm for generating the inputs.

For exponential distribution:

$$X_i = F^{-1}(R_i) = -\frac{1}{\lambda} * \ln{(1 - R_i)}$$

Figure 12 Random Variate Generation code

The histograms of the generated inputs were as follows:

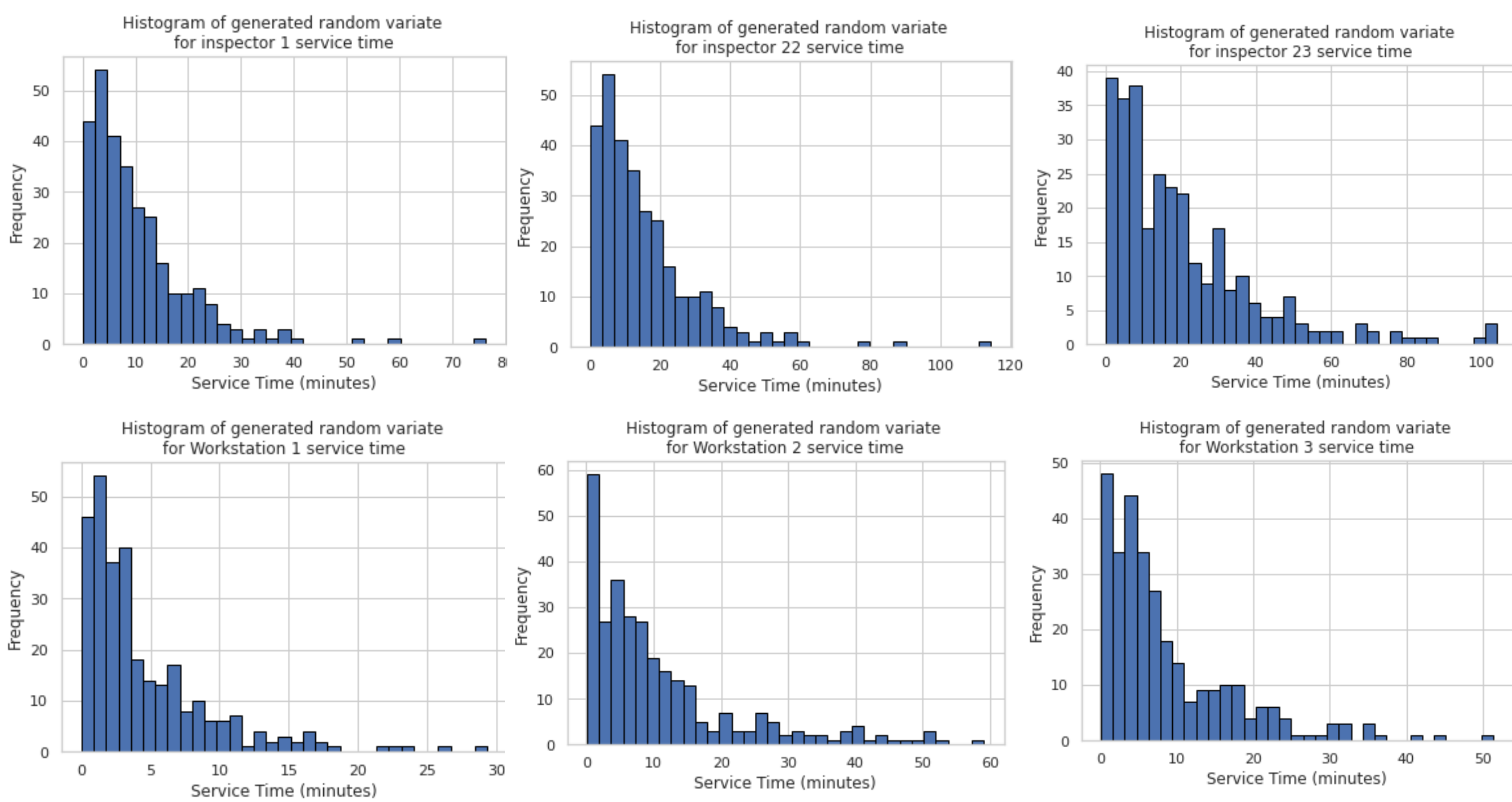


Figure 13 Histograms of generated random variate

As expected, all generated inputs follow an exponential distribution, and the shape is similar to the histograms of part 1 since we used the correct formula and correct average value.

Model Verification and Validation:

After generating random variates of the given data in deliverable 2, we can now use these variates as an input for the system.

Verification:

The conceptual model, flowcharts for each action event, and source code were delivered in deliverable 1. Hence the verification process will start by checking outputs for reasonableness, then using the conservation equation (a.k.a Little's law) to ensure that the model is reliable and ready to be validated. Reported quantities after running the simulation for 1000 products are as follows:

Table 3 Simulation report

CLOCK	6570
AVERAGE COMPONEN 1 ARRIVAL RATE	0.39 C1/ minute
AVERAGE C1 IN THE SYSTEM	10.87
TOTAL COMPONENT 1 IN THE SYSTEM	1010
AVG TIME C1 SPENT IN THE SYSTEM	24.41
Total products ASSEMBLED	1000
SYSTEM THROUGHPUT	0.475
TotalBusy of W1	3361.79
TotalBusy of W2	1071.298
TotalBusy of W3	451.350
PROBABILITY W1 IS BUSY	0.52
PROBABILITY W2 IS BUSY	0.34
PROBABILITY W3 IS BUSY	0.13
W1 AVERAGE RESPONSE TIME	4.46 MINUTES
W2 AVERAGE RESPONSE TIME	10.59 MINUTES
W3 AVERAGE RESPONSE TIME	7.15 MINUTES
PROBABILITY INSPECTOR 1 IS BLOCKED	25.181 %
PROBABILITY INSPECTOR 2 IS BLOCKED	45.642 %
AVERAGE BUFFER 1 OCCUPANCY	0.920
AVERAGE BUFFER 1 2 OCCUPANCY	1.22
AVERAGE BUFFER 1 3 OCCUPANCY	1.12
AVERAGE BUFFER 2 OCCUPANCY	1.46
AVERAGE BUFFER 3 OCCUPANCY	1.58

The conservation equation:

$$\hat{L} = \frac{1}{T}\int_{0}^{T} L(t)dt = \frac{1}{T}\sum_{i=1}^{N} W_i = \frac{N}{T}\frac{1}{N}\sum_{i=1}^{N} W_i = \hat{\lambda}\,\hat{w}$$

$\hat{L}$ represents the average component 1 in the system (Buffers and counter), $\hat{\lambda} = N/T$ is the arrival rate of Component 1, and $\hat{w}$ is the average time C1 spends in the system.

Applying the formula to values of C1 as it is the main component for all products,

$w = \frac{L}{\lambda} = \frac{10.83}{0.39} = 27.87$ which is close to the reported average time C1 spends in the system = 24.41.

Also, total number of products generated was 1000, while the total number of components 1 in the system was 1010. This shows that some component 1 was still in the buffers or the workstations. However, 1000 =~ 1010, and this satisfies Little's law. All entered components will eventually exit the system. Moreover, the utilization of workstation 1 = 0.51 is larger the utilization of workstation 2 = 0.34 and workstation 3 = 0.13. This is reasonable as workstation 1 can start assembling the product as soon as component 1 is available, whereas workstations 2 and 3 must wait for the availability of component 1.

Validation:

In order to validate the system, we need to compare the output variables to the real system. In our case, the only data available from the actual system is the inspector's inspection time and workstations processing times. Hence, we will use these data and compare them with the data received from the model. We will perform multiple replications to obtain more accurate averages. The mean and standard deviation were calculated using the following formula:

$$\bar{Y}_2 = \frac{1}{n}\sum_{i=1}^{n} Y_{2i} \quad S = \left[\frac{\sum_{i=1}^{n}(Y_{2i} - \bar{Y}_2)^2}{n-1}\right]^{1/2}$$

To test the statistic, we used the following *t* distribution formula: $|t_0| = \left|\frac{\bar{Y}_2 - \mu_0}{\frac{S}{\sqrt{n}}}\right|$

Where μ_0 is the actual average of the given data

The table below shows the results from the model for each replication with sample mean and standard deviation. Then it shows the calculated test static value and compares it to the critical value found in the table.

Table 4 Simulation's response time and comparison with given data

Replication	ws1	ws2	ws3	ins1	ins22	ins23
1	4.341	9.038	6.833	9.916	14.972	21.385
2	4.645	10.72	11.608	10.034	15.684	19.82
3	4.303	11.171	11.527	10.311	15.496	20.705
4	4.631	18.355	15.031	10.05	15.385	19.745
5	4.49	10.446	8.772	11.079	14.88	20.274
6	4.581	12.512	8.845	10.446	15.763	20.256
7	4.412	7.939	8.336	10.342	15.542	20.516
8	4.852	13.909	9.73	10.178	15.67	22.049
9	4.366	11.965	8.117	10.632	16.488	19.334
10	4.563	11.676	8.908	10.907	15.814	20.105
11	4.485	8.771	9.269	10.297	16.096	20.445
12	4.506	11.259	8.132	9.754	14.955	20.459
13	4.619	15.844	8.409	10.268	16.022	19.993
14	4.682	11.01	9.595	9.631	15.392	20.121
15	4.733	12.285	7.826	10.786	15.248	20.075
16	4.947	13.425	9.216	9.901	15.62	20.345
17	4.478	12.085	8.848	10.388	15.209	20.136
18	4.832	10.991	10.399	10.417	15.451	20.964
19	4.553	11.727	9.08	10.786	14.981	20.754
20	4.414	11.796	8.728	10.743	15.548	20.476
21	4.667	11.534	7.195	10.42	15.517	20.462
22	4.547	13.282	8.637	10.9	15.175	20.725
23	4.705	11.135	6.581	10.413	15.198	19.742
24	4.462	10.586	7.151	9.922	15.556	20.578
Sample mean	4.576	11.811	9.032	10.355	15.486	20.394
Standard deviation	0.161	2.123	1.753	0.38	0.379	0.55
Given data mean	4.604	11.093	8.796	10.358	15.537	20.633
Test statistic	0.865	1.657	0.66	0.038	0.66	2.124
t static critical for 98% Confidence Level and df = n -1	2.5	2.5	2.5	2.5	2.5	2.5
Confidence interval	[4.543,4.610]	[11.09, 12.529]	[8.67, 9.39]	[10.28, 10.45]	[15.434, 15.539]	[20.15,20.63]
	H0, Accept	H0, Accept	H0, Accept	H0, Accept	H0, Accept	H0, Accept

Twenty-four replications were sufficient for our model to receive accurate results with test statistics less than the critical t value. Also, performing the confidence interval testing, all C.I. contains the given data mean with small best- and worst-case errors.

Production Runs and Analysis:

In this section, we are interested in finding the quantities of interest, which are the "facility **throughput** (total products/ time)", "probability each **workstation is busy** (total_ws_time/ time)", the "average **buffer occupancy** (total_C_in_buffer/ time)", and "probability each **inspector is blocked**" (total_block_time/ time). To find these quantities, we need to perform independent (different random input stream) replications on our simulation. We will follow an iterative approach to find the appropriate number of replications. Also, we will make sure that we consider an initialization phase before starting recording the quantities. Finally, describe the approach used to find the confidence interval for each quantity.

Step 1: The initialization phase:

The initialization phase aims to give the model time until it reaches a steady-state. After this phase, we start collecting the data. This will ensure that the collected quantities will be more representative of the long-run conditions.

We will run a rough simulation that ends after 30,000 minutes to determine how long the initialization phase must be. Afterward, we will divide the total run time into batches of roughly 1000 minutes each to smooth the overall average. The averages of these batches will be used in an ensemble to identify the warm-up phase for each quantity.

The warm-up period for each quantity was as follows:

Table 5 Initialization phase for each quantity

Quantity	**T_0 (minute)**	**Measuring period**
Total products ASSEMBELED	3010	26990
SYSTEM THROUGHPUT	1200	28800
TotalBusy of W1	1000	29000
TotalBusy of W2	1050	28950
TotalBusy of W3	1100	28900
PROBABILITY W1 IS BUSY	3500	26500
PROBABILITY W2 IS BUSY	3150	26850
PROBABILITY W3 IS BUSY	3200	26800
W1 AVERAGE RESPONSE TIME	2500	27500
W2 AVERAGE RESPONSE TIME	2550	27450
W3 AVERAGE RESPONSE TIME	2600	27400
PROBABILITY INSPECTOR 1 IS BLOCKED	4000	26000
PROBABILITY INSPECTOR 2 IS BLOCKED	3500	26500
AVERAGE BUFFER 1 OCCUPANCY	4000	26000
AVERAGE BUFFER 1 2 OCCUPANCY	5000	25000
AVERAGE BUFFER 1 3 OCCUPANCY	5500	24500
AVERAGE BUFFER 2 OCCUPANCY	2000	28000
AVERAGE BUFFER 3 OCCUPANCY	2000	28000

Step 2: The suitable number of replications and confidence intervals:

For the confidence interval, we need to first find the point estimator for each quantity.

For replication, r:

$$\bar{Y}_r = \frac{1}{n-d} * \sum_{j=d+1}^{n} \overline{Y_{rj}}$$

Where d can be interpreted from table 3 for each quantity, and n = 30,000

The overall point estimator:

$$\bar{Y} = \frac{1}{R} * \sum_{r=1}^{R} \bar{Y}_r$$

To estimate the standard error of $\bar{Y}$ and sample variance

$$S^2 = \frac{1}{R-1}\sum_{r=1}^{R}(\bar{Y}_{r.} - \bar{Y}_{..})^2 = \frac{1}{R-1}\left(\sum_{r=1}^{R}\bar{Y}_{r.}^{\,2} - R\bar{Y}_{..}^{\,2}\right) \quad \text{and} \quad s.e.(\bar{Y}_{..}) = \frac{S}{\sqrt{R}}$$

Now With $\varepsilon = 0.04, \alpha = 0.05$, initial R estimate $R_0 = 10$

Table 6 Simulation reports after 10 replications

Replication (r)	System Throughput (product/ minute)	WSi busy % (wsi_service_time/ time)			Average Buffer occupancy (total_C_in_buffer/ time)					Inspector Blockage probability (total_block_time/ time)	
		1	**2**	**3**	**1**	**2**	**3**	**1, 2**	**1, 3**	**Ins 1**	**Ins 2**
1	0.597	0.594	0.098	0.095	0.432	1.069	1.975	1.466	1.362	0.058	0.863
2	0.606	0.566	0.247	0.077	0.351	1.698	1.818	1.212	1.075	0.051	0.638
3	0.334	0.703	0.115	0.142	0.403	1.254	1.802	1.62	1.335	0.037	0.758
4	0.539	0.355	0.165	0.099	0.29	1.689	1.733	1.092	1.014	0.016	0.698
5	0.383	0.673	0.139	0.084	0.439	1.279	1.661	1.629	1.526	0.047	0.598
6	0.426	0.447	0.235	0.089	0.337	1.614	1.681	1.172	1.119	0.039	0.785
7	0.505	0.395	0.246	0.086	0.277	1.502	1.663	0.907	0.891	0.031	0.879
8	0.445	0.537	0.096	0.055	0.415	1.243	1.565	1.545	1.447	0.027	0.619
9	0.586	0.272	0.157	0.134	0.174	1.52	1.894	0.545	0.412	0.013	0.766
10th	0.458	0.622	0.128	0.089	0.342	1.689	1.821	1.193	1.337	0.035	0.732
Until 10th mean	0.488	0.516	0.163	0.095	0.346	1.456	1.761	1.238	1.152	0.035	0.734
Until 10th S.D.	0.089	0.136	0.057	0.024	0.079	0.216	0.117	0.326	0.313	0.014	0.092

Use the following formula to check if the number of replications is enough:

$$R \geq \left(\frac{t_{\alpha/2,R-1} S_0}{\varepsilon}\right)^2$$

Using $S_0 = 1.761$:

$$\left[\frac{2.262 * 1.761}{0.4}\right]^2 = 99.17$$

As 99.17 > R = 10, we need more replications

Now using R = 100:

Table 7 Simulation Results after 100 replications

Replication (r)	System Throughput (product/ minute)	WSi busy % (wsi_service_time/ time)			Average Buffer occupancy (total_C_in_buffer/ time)					Inspector Blockage probability (total_block_time/ time)	
		1	**2**	**3**	**1**	**2**	**3**	**1, 2**	**1, 3**	**Ins 1**	**Ins 2**
100th	0.615	0.425	0.227	0.145	0.21	1.589	1.576	0.742	0.634	0.026	0.718
Until 100th mean	0.458	0.581	0.174	0.11	0.321	1.381	1.739	1.163	1.058	0.034	0.728
Until 100th S.D.	0.106	0.181	0.082	0.036	0.121	0.338	0.146	0.493	0.464	0.018	0.087

$$\left[\frac{1.962 * 1.731}{0.4}\right]^2 = 72.08$$

72.08 is less than R = 100. We will accept 100 replications as enough replications.

Finally, the confidence interval can be calculated using the following formula,

$$\bar{Y}_{..} \pm t_{\alpha/2,R-1} \frac{S}{\sqrt{R}}$$

Now updating the result table with the point estimator and confidence intervals:

Table 8 Simulation Results with Confidence Interval

Replication (r)	System Throughput (product/ minute)	WSi busy % (wsi_service_time/ time)			Average Buffer occupancy (total_C_in_buffer/ time)					Inspector Blockage probability (total_block_time/ time)	
		1	**2**	**3**	**1**	**2**	**3**	**1, 2**	**1, 3**	**Ins 1**	**Ins 2**
1	0.605	0.903	0.11	0.094	0.463	1.682	1.765	1.795	1.725	0.062	0.844
10	0.402	0.906	0.073	0.092	0.401	0.838	1.889	1.677	1.62	0.046	0.702
50	0.439	0.616	0.119	0.08	0.442	1.24	1.536	1.759	1.785	0.061	0.821
100	0.353	0.582	0.083	0.073	0.401	0.926	1.611	1.576	1.518	0.053	0.659
Until 100th mean	0.453	0.636	0.126	0.101	0.411	1.265	1.724	1.653	1.582	0.052	0.737
Until 100th S.D.	0.09	0.172	0.04	0.024	0.05	0.334	0.131	0.195	0.184	0.009	0.091
Until 100th Variance	0.01	0.027	0.002	0.001	0.002	0.112	0.02	0.038	0.034	0.004	0.007
C.I.	[0.435, 0.471]	[0.59, 0.67]	[0.116 ,0.14]	[0.095 ,0.12]	[0.3, 0.42]	[1.18 ,1.34]	[1.69, 1.75]	[1.61, 1.71]	[1.53, 1.63]	[0.045, 0.054]	[0.715, 0.76]

Alternative Operating Policy:

This section will introduce the chosen alternative operating policy, how it is implemented, and compare it with the base manufacture operating policy.

Alternative operating policy description and flow chart:

Multiple operating policies could be applied to our system, such as increasing buffer capacities or adding more component 1 inspectors for workstations 2 and 3. However, these changes are considered costly to demonstrate as we will apply main changes and require adding new equipment to the system. Hence, I have decided that the alternative policy is changing the priority order that inspector 1 follows for distributing component 1.

The new mode of operation for inspector 1 is as follows, C1 will go to the buffer with the smallest number of components in waiting. In case of a tie, workstation 3 has the highest priority, then workstation 2 and workstation 1.

The flow chart of the new inspector one policy is shown in Figure 14.

We will use the same initialization phase for the new operating mode, as shown in Table 5. Also, the total number of replications will be 100 replications as determined in the previous section.

Common random numbers:

we use the same random numbers to simulate both systems. In other words, for replication r, the base system uses the same random numbers as the alternative policy uses, keeping in mind that each replication uses different random number streams. This way, we can induce positive correlation between both systems samples, reducing the variance in the point estimator Y_1-Y_2.

Production runs:

Now, the production runs results after 100 replications are as follows:

Table 9 Simulation Results of the alternative operating policy

Replication (r)	System Throughput (product/ minute)	WSi busy % (wsi_service_time/ time)			Average Buffer occupancy (total_C_in_buffer/ time)					Inspector Blockage probability (total_block_time/ time)	
		1	**2**	**3**	**1**	**2**	**3**	**1, 2**	**1, 3**	**Ins 1**	**Ins 2**
1	0.563	0.519	0.158	0.248	0.733	0.945	0.796	1.872	1.478	0.117	0.792
10	0.687	0.423	0.153	0.239	0.737	1.137	0.77	1.895	1.599	0.088	0.652
50	0.639	0.409	0.135	0.239	0.595	1.657	0.727	1.879	1.637	0.102	0.763
100	0.788	0.33	0.136	0.193	0.791	1.095	0.683	1.934	1.323	0.091	0.373
Until 100th mean	0.634	0.434	0.122	0.29	0.66	1.198	0.768	1.889	1.531	0.092	0.466
Until 100th S.D.	0.106	0.096	0.038	0.071	0.117	0.334	0.079	0.039	0.158	0.021	0.158
Until 100th Variance	0.011	0.009	0.001	0.005	0.014	0.111	0.006	0.001	0.025	0	0.025

The first look at the results shows how the system throughput boosted from 0.453 to 0.634 and the inspector 2 blockage probability reduced to 0.5 instead of 0.73. However, this came at the cost of increasing inspector 1 blockage time.

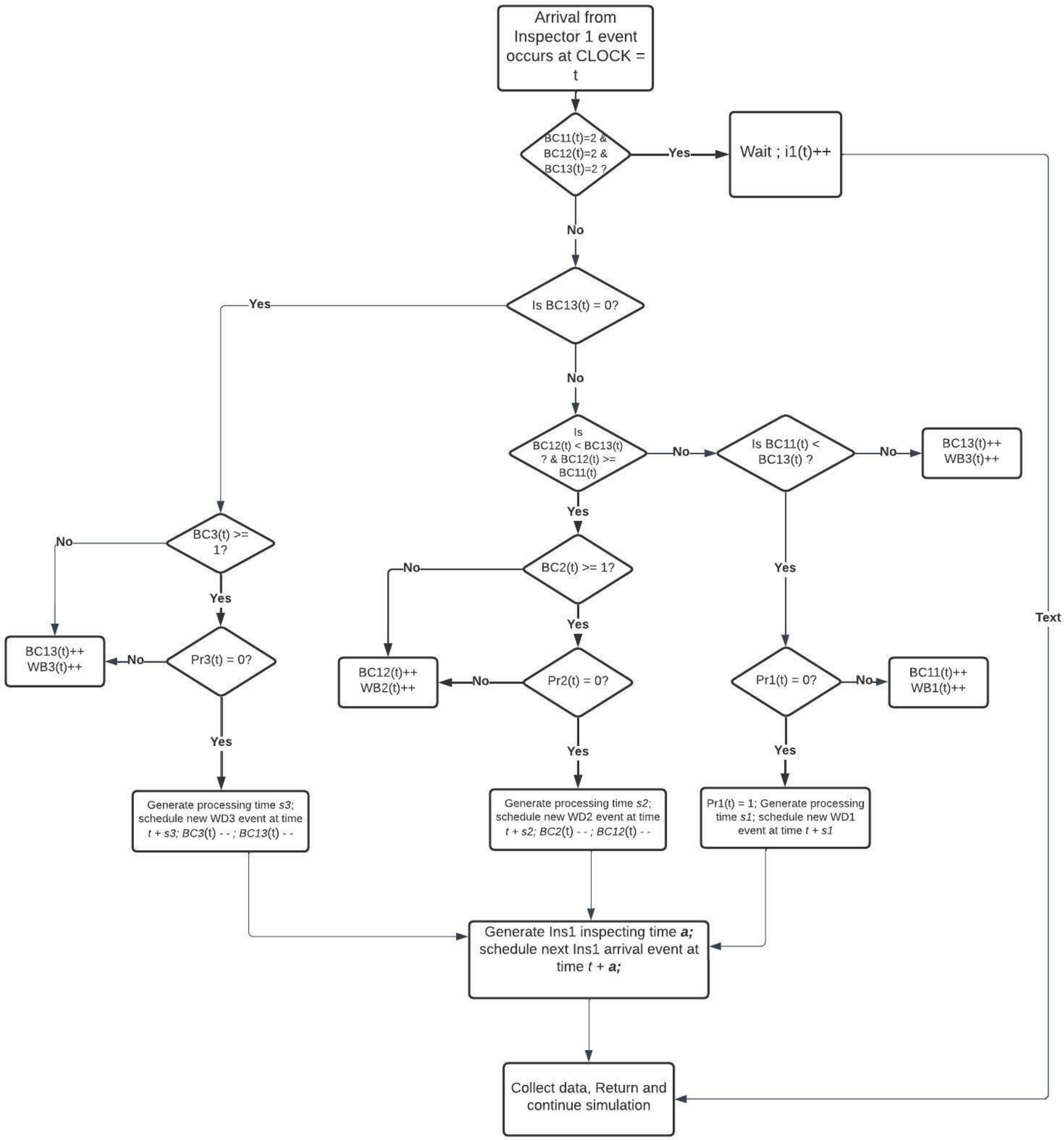


Figure 14 Alternative policy: inspector 1 new operating mode

Comparison of Two System Designs:

To further analyze the new operating policy, next is computing the confidence interval for all metrics.

To calculate the confidence interval, we will use the following formula:

$$\bar{Y}_{.1} - \bar{Y}_{.2} \pm t_{\alpha/2,\upsilon} s.e.(\bar{Y}_{.1} - \bar{Y}_{.2})$$

It is reasonable to assume that sample variances for both designs are almost equal; hence we will use the following formula to estimate the standard error,

$$s.e.\left(\bar{Y}_{.1} - \bar{Y}_{.2}\right) = S_p \sqrt{\frac{1}{R_1} + \frac{1}{R_2}}$$

S_p is the pooled estimate of variance and can be calculated as follows:

$$S_p^2 = \frac{(R_1 - 1)S_1^2 + (R_2 - 1)S_2^2}{R_1 + R_2 - 2}$$

S1 and S2 are the sample variances for designs 1 and 2, respectively. R1 = R2 = 100

And $v = R1 + R2 - 2 = 198$ degrees of freedom, t = 2.330.

The following table shows the confidence interval for each metric:

Table 10 Comparison of Two System Designs and confidence interval

	System Throughput (product/ minute)	WSi busy % (wsi_service_time/ time)			Average Buffer occupancy (total_C_in_buffer/ time)					Inspector Blockage probability (total_block_time/ time)	
		1	**2**	**3**	**1**	**2**	**3**	**1, 2**	**1, 3**	**Ins 1**	**Ins 2**
Design 1 $\bar{Y}_1$	0.453	0.636	0.126	0.101	0.411	1.265	1.724	1.653	1.582	0.052	0.737
S1	0.01	0.027	0.002	0.001	0.002	0.112	0.02	0.038	0.034	0.004	0.007
Design 2 $\bar{Y}_2$	0.634	0.434	0.122	0.29	0.66	1.198	0.768	1.889	1.531	0.092	0.466
S2	0.011	0.009	0.001	0.005	0.014	0.111	0.006	0.001	0.025	0.005	0.025
$\bar{Y}_1 - \bar{Y}_2$	-0.19	0.21	0.01	-0.19	-0.25	0.07	0.96	-0.24	0.06	-0.04	0.28
Standard error	1.49E-03	2.85E-03	2.24E-04	5.10E-04	1.41E-03	1.58E-02	2.09E-03	3.80E-03	4.22E-03	6.40E-04	2.60E-03
C.I. +	-1.78E-01	2.09E-01	4.52E-03	-1.88E-01	-2.46E-01	1.04E-01	9.61E-01	-2.27E-01	6.08E-02	-3.85E-02	2.77E-01
C.I. -	-1.84E-01	1.95E-01	3.48E-03	-1.90E-01	-2.52E-01	3.03E-02	9.51E-01	-2.45E-01	4.12E-02	-4.15E-02	2.65E-01

The table above shows how the second operation policy has better overall performance. This is expected as the new approach ensures that workstations 2 and 3, which have longer service times receive necessary components earlier than workstation 1, whose service time is shorter. This way, workstations 2 and 3 are better utilized with a negligible effect on workstation 1. The new policy ensures that more products arrive from workstations 2 and 3 in addition to those from workstation 1.

Conclusion:

This report discussed the procedure of simulating a manufacturing facility. We started by formulating the problem and discussed how simulation is the appropriate method to solve this problem. Later on, we introduced the concepts of the simulation model and introduced flowcharts of the python functions used in the evaluating the model. Then we started the procedure of input modelling using the provided data. Using histograms, Q-Q plots, and Chi-square tests we proved that all data follows an exponential distribution. Then we used Linear Congruential Method to generate random numbers and performed the Chi-square test to check its uniformity and the Autocorrelation test to check its independence. With the help of the inverse-transform method and the generated random numbers, we generated exponentially distributed random variates which has been used as in input to the model. Next, we started the verifying the model by investigating the initial output of the results and discussing its reasonability, as well as using Little's law to ensure that our model follows the conservation rule. Afterward, we tried validating the model by using the given input data and comparing them with the service times calculated during the simulation, keeping in mind that validating the model could improve if output quantities data from the real world were available. Subsequently, we started the production runs and analysis process. We first assumed the initialization phase for each quantity by using the moving average approach. Secondly, we performed the replication test to determine the appropriate number of replications. Lastly, we used the final output quantities to estimate a confidence interval where the real-life quantities could fall. Finally, we introduced the new operating policy where we changed the priority of inspector 1. We used the approach described in chapter 11 of the book to compare both models. We came to a conclusion that the new changes helped improve the throughput by 28% and reduced inspector 2 blockage by 36.7%.

Appendix:

Random variates Generation source sode:

```
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
import statsmodels.api as sm
import scipy.stats as stats
from scipy.stats.stats import chisquare

color = '#fc4f30'
sns.set()
sns.set_style("whitegrid")

insp1 = np.loadtxt('servinsp1.dat', unpack = True)
insp22 = np.loadtxt('servinsp22.dat', unpack = True)
insp23 = np.loadtxt('servinsp23.dat', unpack = True)
ws1 = np.loadtxt('ws1.dat', unpack = True)
ws2 = np.loadtxt('ws2.dat', unpack = True)
ws3 = np.loadtxt('ws3.dat', unpack = True)
data = [("inspector 1",insp1), ("inspector 22",insp22), ("inspector 23",insp23),
        ("Workstation 1",ws1), ("Workstation 2",ws2), ("Workstation 3",ws3)]

```

```
for sample in data:
  bins = round(np.sqrt(len(sample[1])))+1
  """First step is plotting the histogram for the provided data"""
  sample_mean = round(sum(sample[1])/ len(sample[1]),3)
  _sample = plt.hist(sample[1], bins = bins, edgecolor='black')
  _sample = plt.title(f'Histogram of {sample[0]} service time ')
  _sample = plt.xlabel('Service Time (minutes)')
  _sample = plt.ylabel('Frequency')
  plt.show()
  print(f'Max {sample[0]} Service time: {max(sample[1])}')
  print(f"Mean {sample[0]} Service time: {sample_mean}\n\n")

```

```
#QQ plot:
#since all histograms above are having a exponential distributions shape
#lets plot the Q_Q plot for exponential dist.

for sample in data:
  print("\n \n ")
  sm.qqplot(sample[1],fit=True, line='q', dist=stats.expon)
  _insp1 = plt.title(f'{sample[0]} Q-Q Plot for Exp. Dist.')
  plt.show()
```

```python
# Now lets perform the chi-square goodness of fit test
print('H0: The data is exponentially distributed.')
print('H1: The data is not exponentially distributed.\n')
for sample in data:
  print(f'Chi-Square test for {sample[0]} with the exponential disribution: ')
  bins = round(np.sqrt(len(sample[1])))+1
  expected = [len(sample[1])/bins] * 19
  a_i = []
  lambda_ = 1/round(sum(sample[1])/ len(sample[1]),3)
  for i in range(20):
    x = (-1/lambda_)*np.log(1-i*0.05)
    a_i.append(x)
  #print(a_i)
  o_i = plt.hist(sample[1], bins = a_i, edgecolor='black')
  o_i = o_i[0].tolist()
  #print(o_i)
  chisquare = stats.chisquare(f_obs=o_i, f_exp = expected)
  crit = stats.chi2.ppf(q=0.95, df= bins-2)
  plt.close()
  print(f' chi-square is {round(chisquare.statistic,3)} | critical value is {round(crit,3)}')
  if (chisquare.statistic <= crit):
    print('  H0, accept the hypothesess \n')
  else: print('H1, reject the hypothesess')
```

Simulation Source code:

```
"""

Study of A Manufacturing Facility.

Created on Tue April. 11, 2022, 22:08:48

@author: Aziz
"""
import numpy as np
import random
from scipy.stats import expon
from scipy.stats import norm
import matplotlib.pyplot as plt
import queue
from dataclasses import dataclass, field
from typing import Any
import seaborn as sns

color = '#fc4f30'
sns.set()
sns.set_style("whitegrid")

insp1 = np.loadtxt('inspector 1 random variate.dat', unpack = True)
insp22 = np.loadtxt('inspector 22 random variate.dat', unpack = True)
insp23 = np.loadtxt('inspector 23 random variate.dat', unpack = True)
ws1 = np.loadtxt('Workstation 1 random variate.dat', unpack = True)
ws2 = np.loadtxt('Workstation 2 random variate.dat', unpack = True)
ws3 = np.loadtxt('Workstation 3 random variate.dat', unpack = True)
uinsp1 = np.loadtxt('servinsp1.dat', unpack = True)
uinsp22 = np.loadtxt('servinsp22.dat', unpack = True)
uinsp23 = np.loadtxt('servinsp23.dat', unpack = True)
uws1 = np.loadtxt('ws1.dat', unpack = True)
uws2 = np.loadtxt('ws2.dat', unpack = True)
uws3 = np.loadtxt('ws3.dat', unpack = True)
```

```
  def put1(self, C1, clock):
    if self.alternat == 0:
        """ add a component C1 into buffer1"""
        """ update clock"""
        self._Clock = clock
        minBufferLength = min(self._BufferLength1, self._BufferLength12, self._BufferLength13)
        """ start service if W1 is empty """
        if (self._NumberInServiceW1 == 0):
            self.idlew1Total += (clock - self.idlew1start)
            self._NumberInServiceW1 = 1
            self._InServiceW1.append(C1)
            depart = self.scheduleDepartureW1(C1)
            self._BlockIns1 = 0

            """if W1 is busy, check W2 then W3 to determine which workstation C1 goes to"""
            """Case C2 is already aviailble and waiting for C12 and WS2 is empty"""
        elif (self._NumberInServiceW2 == 0 and self._C2available ):
            # self.idlew2Total += (clock - self.idlew2start)
            self._NumberInServiceW2 = 1
            self._InServiceW2.append(C1)
            depart = self.scheduleDepartureW2(C1)
            self._BlockIns1 = 0

            """Case C3 is already aviailble and waiting for C13 and WS3 is empty"""
        elif (self._NumberInServiceW3 == 0 and self._C3available ):
            # self.idlew3Total += (clock - self.idlew3start)
            self._NumberInServiceW3 = 1
            self._InServiceW3.append(C1)
            depart = self.scheduleDepartureW3(C1)
            self._BlockIns1 = 0

            """if all W are busy or no enough components, C1 will go to the shortest buffer"""
            """Note that the buffer can't exceed 2 components"""
    #check if there is a buffer less than 2, else Inspector 1 will go idle


        elif (minBufferLength < 2 and self._BlockIns1 == 0):
            self._BlockIns1 = 0
            if(self._BufferLength1 == minBufferLength):
                self._Buffer1.append(C1)
                self._BufferLength1 += 1
                depart = None
                """ update W1 statistics"""
                # self._BufferLengthTime1.append([self._BufferLength1, self._Clock])
                self._TotalBusyW1 += (self._Clock - self._LastEventTime)

            elif(self._BufferLength12 == minBufferLength):
                self._Buffer12.append(C1)
                self._BufferLength12 += 1
                self._C12available = 1
                depart = None
                """ update W2 statistics"""
                # self._BufferLengthTime12.append([self._BufferLength12, self._Clock])
                if(self._C2available):
                    self._TotalBusyW2 += (self._Clock - self._LastEventTime)

```

```
        elif(self._BufferLength13 == minBufferLength):
            self._Buffer13.append(C1)
            self._BufferLength13 += 1
            self._C13available = 1
            depart = None

            """ update W3 statistics"""
            # self._BufferLengthTime13.append([self._BufferLength13, self._Clock])
            if(self._C3available):
                self._TotalBusyW3 += (self._Clock - self._LastEventTime)


    #if min >= 2 then inspector 1 should be blocked until there is a vacancy
    else:
        """ update W1 statistics"""
        # self._BufferLengthTime1.append([self._BufferLength1, self._Clock])
        self._TotalBusyW1 += (self._Clock - self._LastEventTime)

        """ update W3 statistics"""
        # self._BufferLengthTime13.append([self._BufferLength13, self._Clock])
        if(self._C3available):
          self._TotalBusyW3 += (self._Clock - self._LastEventTime)

        """ update W2 statistics"""
        # self._BufferLengthTime12.append([self._BufferLength12, self._Clock])
        if(self._C2available):
          self._TotalBusyW2 += (self._Clock - self._LastEventTime)

        self._BlockIns1 = 1
        self._Ins1Blockage +=1
        self._OccupiedBuffer+=3
        # print(f'INS1 is blocked for the {self._Ins1Blockage} time')
        depart = None


      #*******************Aleternative solution
if self.alternat == 1:
    """ add a component C1 into buffer1"""
    """ update clock"""
    self._Clock = clock
    minBufferLength = min(self._BufferLength1, self._BufferLength12, self._BufferLength13)
    """ start service if W1 is empty """
    if (self._NumberInServiceW3 == 0 and self._C3available ):
        self._NumberInServiceW3 = 1
        self._InServiceW3.append(C1)
        depart = self.scheduleDepartureW3(C1)
        self._BlockIns1 = 0

        """if W1 is busy, check W2 then W3 to determine which workstation C1 goes to"""
        """Case C2 is already aviailble and waiting for C12 and WS2 is empty"""
    elif (self._NumberInServiceW2 == 0 and self._C2available ):

        self._NumberInServiceW2 = 1
        self._InServiceW2.append(C1)
        depart = self.scheduleDepartureW2(C1)
```

```python
            self._BlockIns1 = 0

            """Case C3 is already aviailble and waiting for C13 and WS3 is empty"""
        elif (self._NumberInServiceW1 == 0):
            self.idleW1Total += (clock - self.idleW1start)
            self._NumberInServiceW1 = 1
            self._InServiceW1.append(C1)
            depart = self.scheduleDepartureW1(C1)
            self._BlockIns1 = 0

            """if all W are busy or no enough components, C1 will go to the shortest buffer"""
            """Note that the buffer can't exceed 2 components"""
    #check if there is a buffer less than 2, else Inspector 1 will go idle


        elif (minBufferLength < 2 and self._BlockIns1 == 0):
            # self._BlockIns1 = 0
            if(self._BufferLength13 == minBufferLength):
                self._Buffer13.append(C1)
                self._BufferLength13 += 1
                self._C13available = 1
                depart = None
                """ update W3 statistics"""
                # self._BufferLengthTime13.append([self._BufferLength13, self._Clock])
                if(self._C3available):
                    self._TotalBusyW3 += (self._Clock - self._LastEventTime)

            elif(self._BufferLength12 == minBufferLength):
                self._Buffer12.append(C1)
                self._BufferLength12 += 1
                self._C12available = 1
                depart = None
                """ update W2 statistics"""
                # self._BufferLengthTime12.append([self._BufferLength12, self._Clock])
                if(self._C2available):
                    self._TotalBusyW2 += (self._Clock - self._LastEventTime)

            elif(self._BufferLength1 == minBufferLength):
                self._Buffer1.append(C1)
                self._BufferLength1 += 1
                depart = None
                """ update W1 statistics"""
                # self._BufferLengthTime1.append([self._BufferLength1, self._Clock])
                self._TotalBusyW1 += (self._Clock - self._LastEventTime)


        #if min >= 2 then inspector 1 should be blocked until there is a vacancy
        else:
            """ update W1 statistics"""
            # self._BufferLengthTime1.append([self._BufferLength1, self._Clock])
            self._TotalBusyW1 += (self._Clock - self._LastEventTime)

            """ update W3 statistics"""
            # self._BufferLengthTime13.append([self._BufferLength13, self._Clock])
            if(self._C3available):
              self._TotalBusyW3 += (self._Clock - self._LastEventTime)
```

```python

              """ update W2 statistics"""
              # self._BufferLengthTime12.append([self._BufferLength12, self._Clock])
              if(self._C2available):
                self._TotalBusyW2 += (self._Clock - self._LastEventTime)

              self._BlockIns1 = 1
              self._Ins1Blockage +=1
              self._OccupiedBuffer+=3
              # print(f'INS1 is blocked for the {self._Ins1Blockage} time')
              depart = None

        # print('self._BlockIns1', self._BlockIns1)
      self._BufferLengthTime1.append([self._BufferLength1, self._Clock])
      self._BufferLengthTime12.append([self._BufferLength12, self._Clock])
      self._BufferLengthTime13.append([self._BufferLength13, self._Clock])

      self._LastEventTime = self._Clock
      self._INS1Blokage_Time.append([self._BlockIns1, self._Clock])
      return depart


    def put2(self, C2, clock):
      """ add a component C2 into buffer2 """
      """ update clock"""
      self._Clock = clock

      """ start service if W2 is empty and C1 availble"""
      if (self._NumberInServiceW2 == 0 and self._C12available ):
            self._BlockIns2 = 0
            self._NumberInServiceW2 = 1
            self._InServiceW2.append(C2)
            depart = self.scheduleDepartureW2(C2)

     # """if W2 is busy or there is no C1 availble, add C2 to C2 buffer"""
      #"""Note that the buffer can't exceed 2 components"""
      elif(self._BufferLength2 < 2 and self._BlockIns2 == 0):
          self._Buffer2.append(C2)
          self._BufferLength2 += 1
          self._C2available = 1
          if self._BufferLength2 == 2:
            self._BlockIns2 = 1
          depart = None
          """ update W2 statistics"""

          if(self._C12available):
            self._TotalBusyW2 += (self._Clock - self._LastEventTimeW2)


        #if _BufferLength2 >= 2 then inspector 2 should be blocked until there is a vacancy
      else:
            self._BlockIns2 = 1
            self._Ins2Blockage +=1
            self._OccupiedBuffer+=1
            depart = None
```

```
    self._BufferLengthTime2.append([self._BufferLength2, self._Clock])
    self._LastEventTimeW2 = self._Clock
    self._INS2Blokage_Time.append([self._BlockIns2, self._Clock])
    return depart


  def put3(self, C3, clock):
    """ add a component C3 into buffer3 """
    """ update clock """
    self._Clock = clock

    """ start service if W3 is empty and C1 availble"""
    if (self._NumberInServiceW3 == 0 and self._C13available ):
      self._BlockIns3 = 0
      self._NumberInServiceW3 = 1
      self._InServiceW3.append(C3)
      depart = self.scheduleDepartureW3(C3)

      """if W3 is busy or there is no C1 availble, add C3 to C3 buffer"""
      """Note that the buffer can't exceed 2 components"""
    elif(self._BufferLength3 < 2 and self._BlockIns3 == 0):
      self._BlockIns3 = 0
      self._Buffer3.append(C3)
      self._BufferLength3 += 1
      self._C3available = 1
      if self._BufferLength3 == 2:
          self._BlockIns3 = 1
      depart = None

      """ update W3 statistics"""
      if(self._C13available):
        self._TotalBusyW3 += (self._Clock - self._LastEventTimeW3)


    #if _BufferLength3 >= 2 then inspector 3 should be blocked until there is a vacancy
    else:
      self._BlockIns3 = 1
      self._Ins3Blockage +=1
      self._OccupiedBuffer+=1
      depart = None

    #print('self._BlockIns3', self._BlockIns3)
    self._BufferLengthTime3.append([self._BufferLength3, self._Clock])
    self._INS3Blokage_Time.append([self._BlockIns3, self._Clock])
    self._LastEventTimeW3 = self._Clock
    return depart



  def get3(self, clock):
      """ get product from the workstaion 3"""

      P3 = self._InServiceW3.pop(0)
```

```
        """ update clock"""
        self._Clock = clock

        """ if buffer13 and buffer3 is not empty, schedule next departure"""
        if ( self._C3available and self._C13available):
            """ move component from buffers to workstation 3"""

            C33 = self._Buffer3.pop(0)
            C13 = self._Buffer13.pop(0)

            if(C33[0] > C13[0]):
              self._InServiceW3.append(C33)
              depart = self.scheduleDepartureW3(C33)
            else:
              self._InServiceW3.append(C13)
              depart = self.scheduleDepartureW3(C13)

            if ( self._BufferLength3 > 0):
              self._BufferLength3  -= 1
              self._BlockIns3 = 0
              if self._BufferLength3 == 0: self._C3available = 0

            if self._BufferLength13 > 0 :
              self._BufferLength13 -= 1
              # self._BlockIns1 = 0
              if self._BufferLength13 == 0: self._C13available = 0


            self._BufferLengthTime3.append([self._BufferLength3, self._Clock])
            self._BufferLengthTime13.append([self._BufferLength13, self._Clock])

        else:
            self._BlockIns3 = 0
            self._INS3Blokage_Time.append([self._BlockIns3, self._Clock])
            # self._BlockIns1 = 0
            self._NumberInServiceW3 = 0
            depart = None

        """ update statistics"""

        responsew3 = clock - P3[0]
        self._SumResponseTimeW3 += responsew3
        self._TotalBusyW3 += (self._Clock - self._LastEventTime)
        self._NumberOfDeparturesW3 += 1
        self.AVGWS3_time.append([self.totalservicetimeW3/self._NumberOfDeparturesW3, self._Clock])
        self._LastEventTime = self._Clock
        self._TotalP3s += 1
        return P3, depart

    def get2(self, clock):

        """ get product from the workstaion 2"""
        P2 = self._InServiceW2.pop(0)

```

```
        """ update clock"""
        self._Clock = clock

        """ if buffer12 and buffer2 is not empty, schedule next departure"""
        if ( self._C2available and self._C12available):
            """ move component from buffers to workstation 2"""

            C22 = self._Buffer2.pop(0)
            C12 = self._Buffer12.pop(0)
            if(C22[0] > C12[0]):
              self._InServiceW2.append(C22)
              depart = self.scheduleDepartureW2(C22)
            else:
              self._InServiceW2.append((C12[0],C22[1]))
              depart = self.scheduleDepartureW2((C12[0],C22[1]))

            if ( self._BufferLength2 > 0):
              self._BufferLength2  -= 1
              # self._BlockIns2 = 0

              if self._BufferLength2 == 0: self._C2available = 0

            if self._BufferLength12 > 0 :
              self._BufferLength12 -= 1
              # self._BlockIns1 = 0
              if self._BufferLength12 == 0: self._C12available = 0

            self._BufferLengthTime2.append([self._BufferLength2, self._Clock])
            self._BufferLengthTime12.append([self._BufferLength12, self._Clock])

        else:
            self._NumberInServiceW2 = 0   #ws2 is empty
            self._BlockIns2 = 0
            self._INS2Blokage_Time.append([self._BlockIns2, self._Clock])
            # self._BlockIns1 = 0
            depart = None

        """ update statistics"""

        responsew2 = clock - P2[0]
        self._SumResponseTimeW2 += responsew2
        self._TotalBusyW2 += (self._Clock - self._LastEventTime)
        self._NumberOfDeparturesW2 += 1
        self.AVGWS2_time.append([self.totalservicetimeW2/self._NumberOfDeparturesW2, self._Clock])
        self._LastEventTime = self._Clock
        self._TotalP2s += 1
        return P2, depart

    def get1(self, clock):

        """ get product from the workstaion 1"""
        P1 = self._InServiceW1.pop(0)

        """ update clock"""
        self._Clock = clock
```

```python
        """ if buffer is not empty, schedule next departure"""
        if ( self._BufferLength1 > 0):
         #   """ move component from buffer to workstation"""

            C11 = self._Buffer1.pop(0)
            self._InServiceW1.append(C11)

            self._BufferLength1 -= 1
            # self._BlockIns1 = 0

        #    """ schedule departure for head-of-line product"""
            depart = self.scheduleDepartureW1(C11)
            self._BufferLengthTime1.append([self._BufferLength1, self._Clock])

        else:
            self._NumberInServiceW1 = 0
            self.idlew1start = clock
            depart = None

        """ update statistics"""

        responsew1 = clock - P1[0]
        self._SumResponseTimeW1 += responsew1
        self._TotalBusyW1 += (self._Clock - self._LastEventTime)
        self._NumberOfDeparturesW1 += 1
        self.AVGWS1_time.append([self.totalservicetimeW1/self._NumberOfDeparturesW1, self._Clock])
        self._LastEventTime = self._Clock
        self._TotalP1s += 1
        return P1, depart

    def scheduleDepartureW1(self, P1):
        ServiceTimeW1 = ws1[np.random.randint(1*self.replication, 1000*self.replication)]
        self.totalservicetimeW1 += ServiceTimeW1
        depart = (self._Clock + ServiceTimeW1, self._departurep1, P1)
        return depart

    def scheduleDepartureW2(self, P2):
        ServiceTimeW2 = ws2[np.random.randint(1*self.replication, 1000*self.replication)]
        self.totalservicetimeW2 += ServiceTimeW2
        depart = (self._Clock + ServiceTimeW2, self._departurep2, P2)
        return depart

    def scheduleDepartureW3(self, P3):
        ServiceTimeW3 = ws3[np.random.randint(1*self.replication, 1000*self.replication)]
        self.totalservicetimeW3 += ServiceTimeW3
        depart = (self._Clock + ServiceTimeW3, self._departurep3, P3)
        return depart



    def qReportGeneration(self, clock):
        self.TotalProducts = (self._TotalP1s + self._TotalP2s + self._TotalP3s)
        Throughput = self.TotalProducts/ clock
        W1RHO = self._TotalBusyW1/clock
```

```
        W2RHO = self._TotalBusyW2/clock
        W3RHO = self._TotalBusyW3/clock
        if self._NumberOfDeparturesW1 != 0:
            AVGRW1 = self.totalservicetimeW1/self._NumberOfDeparturesW1
        else:
            AVGRW1 = 0
        if self._NumberOfDeparturesW2 != 0:
            AVGRW2 = self.totalservicetimeW2/self._NumberOfDeparturesW2
        else:
            AVGRW2 = 0
        if self._NumberOfDeparturesW3 != 0:
            AVGRW3 = self.totalservicetimeW3/self._NumberOfDeparturesW3
        else:
            AVGRW3 = 0

        AVGBufferOcc =self._OccupiedBuffer / clock
        Ins1Blockage = self._Ins1Blockage #/ clock

        print("\n CLOCK: ",clock)
        print("\n Total products ASSEMBELED:  ", self.TotalProducts)
        print("\n SYSTEM THROUGHPUT", self.TotalProducts*5/clock)
        print("\n TotalBusy of W1: ", self._TotalBusyW1)
        print("\n TotalBusy of W2: ", self._TotalBusyW2)
        print("\n TotalBusy of W3: ", self._TotalBusyW3)
        print("\n PROBABILITY W1 IS BUSY: {0:.2f}".format(W1RHO))
        print("\n PROBABILITY W2 IS BUSY: {0:.2f}".format(W2RHO))
        print("\n PROBABILITY W3 IS BUSY: {0:.2f}".format(W3RHO))
        print("\n W1 AVERAGE RESPONSE TIME: {0:.2f} MINUTES".format(AVGRW1))
        print("\n W2 AVERAGE RESPONSE TIME: {0:.2f} MINUTES".format(AVGRW2))
        print("\n W3 AVERAGE RESPONSE TIME: {0:.2f} MINUTES".format(AVGRW3))
        print("\n AVERAGE BUFFER OCCUPANCY per unit time: {0:.2f} ".format(AVGBufferOcc))
        print("\n PROBABILITY INSPECTOR IS BLOCKED: {0:.2f} ".format(Ins1Blockage))
        print("\n Total idle workstation 1 time:", self.idlew1Total)
```

```python
class Sim(object):

    def __init__(self):
        self._arrivalc1 = 1
        self._departurep1 = 2
        self._arrivalc2 = 3
        self._departurep2 = 4
        self._arrivalc3 = 5
        self._departurep3 = 6
        self._MeanInterArrivalTime = 4.5
        self._TotalProducts = 1000
        self._Clock = 0.0
        self._NumberOfSystemDepartures = 0
        self._C1ID = 0
        self._C2ID = 0
        self._C3ID = 0
        self._NumberOfQueues = 1
        self.ins2out = 0
        self.totalIns1time = 0
        self.totalIns22time = 0
        self.totalIns23time = 0
        self.AVGINS1_time = [[0, 0.0],[0, 0.0]]
        self.AVGINS2_time = [[0, 0.0]]
        self.AVGINS3_time = [[0, 0.0]]
        """ create a future event list"""
        self._FutureEventList1 = queue.PriorityQueue()
        self._FutureEventList2 = queue.PriorityQueue()
        self._FutureEventList3 = queue.PriorityQueue()
        self.replication = 1

        self._QList = CustomerQueue()
        self._QList.replication = self.replication

    def scheduleArrivalfromIns1(self):
          """ create C1 arrival event """
          arrivalTimeC1 = self._Clock + insp1[np.random.randint(1*self.replication, 1000*self.replication)]
          self.totalIns1time += arrivalTimeC1 - self._Clock
          C1 = (arrivalTimeC1, self._C1ID)
          self._C1ID += 1
          self.AVGINS1_time.append([self.totalIns1time/self._C1ID, self._Clock])
          evt = (arrivalTimeC1, self._arrivalc1, C1)
          self._FutureEventList1.put(evt)


    def processArrivalC1(self, C1):
        #print("processArrivalC1", self._Clock)
        """ add C1 to customer queue"""
        depart = self._QList.put1(C1, self._Clock)
        """ check if the customer needs to start service immediately"""
        if depart is not None:
            self._FutureEventList1.put(depart)

    def processDeparture1(self, evt):
        #print("processDepartureC1", self._Clock)
        """ get the product """
        P1, depart = self._QList.get1(self._Clock)
```

```
        P1, depart = self._QList.get1(self._Clock)

        """ if there are still C1 in queue, schedule next departure"""
        if depart is not None:
            self._FutureEventList1.put(depart)
        return P1

    def scheduleArrivalfromIns2(self):
        """ create C2 and C3 arrival event """
        random = np.random.random_sample()
        """select c2 or c3 randomlly"""
        if(random < 0.5):
            self.ins2out = 2
            arrivalTimeC2 = self._Clock + insp22[np.random.randint(1*self.replication, 1000*self.replication)]
            self.totalIns22time += arrivalTimeC2 - self._Clock
            C2 = (arrivalTimeC2, self._C2ID)
            self._C2ID +=1
            self.AVGINS2_time.append([self.totalIns22time/self._C2ID, self._Clock])
            evt = (arrivalTimeC2, self._arrivalc2, C2)
            self._FutureEventList2.put(evt)
        else:
            self.ins2out = 3
            arrivalTimeC3 = self._Clock + insp23[np.random.randint(1*self.replication, 1000*self.replication)]
            self.totalIns23time += arrivalTimeC3 - self._Clock
            C3 = (arrivalTimeC3, self._C3ID)
            self._C3ID += 1
            self.AVGINS3_time.append([self.totalIns23time/self._C3ID, self._Clock])
            evt = (arrivalTimeC3, self._arrivalc3, C3)
            self._FutureEventList3.put(evt)

    def processArrivalC2(self, C2):
        #print("processArrivalC2", self._Clock)
        """ add C2 to customer queue"""
        depart = self._QList.put2(C2, self._Clock)
        """ check if the customer needs to start service immediately"""
        if depart is not None:
            self.ins2out = 0
            self._FutureEventList2.put(depart)

    def processDeparture2(self, evt):
        #print("processDC2", self._Clock)
        """ get the product """
        #print('insw2',len(self._QList[queueID]._InServiceW2))
        if(len(self._QList._InServiceW2) > 0):
          P2, depart = self._QList.get2(self._Clock)

          """ if there are still C2 in queue, schedule next departure"""
          if depart is not None:
              self._FutureEventList2.put(depart)
        else:
          # print('HBIBI')
          P2 = None
        return P2


    def processArrivalC3(self, C3):
```

```
        #print("processArrivalC3", self._Clock)
        """ add C3 to customer queue"""
        self._BlockIns2 = 0
        depart = self._QList.put3(C3, self._Clock)
        """ check if the customer needs to start service immediately"""
        if depart is not None:
            self.ins2out = 0
            self._FutureEventList3.put(depart)

    def processDeparture3(self, evt):
        #print("processDC3", self._Clock)
        """ get the product """
        # print('insw3',len(self._QList[queueID]._InServiceW3))
        if(len(self._QList._InServiceW3) > 0):
          P3, depart = self._QList.get3(self._Clock)
          """ if there are still C3 in queue, schedule next departure"""
          if depart is not None:
            self._FutureEventList3.put(depart)
        else:
          # print('HBIBI')
          P3 = None
        return P3
```

```
seed = 1234567
totalproducts = []
productionruns = [[]]
Responsetimes = [[]]
ws1total = []
ws2total = []
ws3total = []
ins1total = []
ins2total = []
ins3total = []
Throughput = []
WS1_Busy = []
WS2_Busy = []
WS3_Busy = []
B1_OCC = []
B2_OCC = []
B3_OCC = []
B12_OCC = []
B13_OCC = []
INS1_Block = []
INS2_Block = []
means = ['Sample mean']
stds = ['Standard deviation']
pmeans = ['Sample mean']
pvar = ['Sample varience']
pstds = ['Standard deviation']
u_0 = ['Given data mean']
t_0s = ['Test statistic']
```

```
productionruns.append(['Replication', 'Throughput',
                       'WS1 Busy', 'WS2 Busy', 'WS3 Busy',
                       'B1 OCC','B2 OCC', 'B3 OCC', 'B12 OCC', 'B13 OCC',
                       'INS1 Blockage', 'INS2 Blockage'])
Responsetimes.append(['Repliction','ws1','ws2','ws3','ins1','ins22','ins23'])
for i in range(1,101):
  np.random.seed((int(seed*i)))

  """ create simulation instance"""
  ms = Sim()
  ms.replication = i
  """ schedule first arrival. event is identified by a tuple (time, type, queue ID, customer)"""
  ms.scheduleArrivalfromIns1()
  ms.scheduleArrivalfromIns2()
  evt2 = [1,1,1,1]
  evt3 = [1,1,1,1]
  while (ms._Clock < 30000):
    """ get imminent event"""
    evt1 = ms._FutureEventList1.get()

    if(ms.ins2out == 2): evt2 = ms._FutureEventList2.get()
    elif(ms.ins2out == 3): evt3 = ms._FutureEventList3.get()

    """ update clock"""
    ms._Clock = max(evt1[0], evt2[0], evt3[0])
    # print('_NumberInServiceW2', ms._QList._NumberInServiceW2)
    """ check event type for c2"""
    if (evt2[1] == ms._arrivalc2):

      """ get C2 info. C2 is identified by a tuple (creation time, C2ID)"""
      C2 = evt2[2]
      ms.processArrivalC2(C2)

      """ schedule next arrival"""
      ms.scheduleArrivalfromIns2()

    elif(evt2[1] == ms._departurep2):
      """ process departure at the queue to be left"""
      P2 = ms.processDeparture2(evt2)
      """ tracking total number of customers leaving the system"""
      ms._NumberOfSystemDepartures += 1
      # print("out p2")

    """ check event type for c3"""
    if (evt3[1] == ms._arrivalc3):

      """ get C3 info. C3 is identified by a tuple (creation time, C3ID)"""
      C3 = evt3[2]
      ms.processArrivalC3(C3)

      """ schedule next arrival"""
      ms.scheduleArrivalfromIns2()

    elif(evt3[1] == ms._departurep3):
```

```python

    """ process departure at the queue to be left"""
    P3 = ms.processDeparture3(evt3)
    """ tracking total number of customers leaving the system"""
    ms._NumberOfSystemDepartures += 1
    # print("out p3")
  """ check event type for c1"""
  if ( evt1[1] == ms._arrivalc1):

    """ get C1 info. C1 is identified by a tuple (creation time, C1ID)"""
    C1 = evt1[2]
    ms.processArrivalC1(C1)

    """ schedule next arrival"""
    ms.scheduleArrivalfromIns1()


  elif(evt1[1] == ms._departurep1):

    """ process departure at the queue to be left"""
    P1 = ms.processDeparture1(evt1)
    """ tracking total number of customers leaving the system"""
    ms._NumberOfSystemDepartures += 1
    # print("out p1")

    """Check p2 coming from put1"""
  elif(evt1[1] == ms._departurep2):
    """ process departure at the queue to be left"""
    P2 = ms.processDeparture2(evt1)
    """ tracking total number of customers leaving the system"""
    ms._NumberOfSystemDepartures += 1
    # print("out p2 ")

    """Check p3 coming from put1"""
  elif(evt1[1] == ms._departurep3):
    """ process departure at the queue to be left"""
    P3 = ms.processDeparture3(evt1)
    """ tracking total number of customers leaving the system"""
    ms._NumberOfSystemDepartures += 1

y1, x1 = zip(*ms._QList._BufferLengthTime1)
y2, x2 = zip(*ms._QList._BufferLengthTime2)
y3, x3 = zip(*ms._QList._BufferLengthTime3)
y12, x12 = zip(*ms._QList._BufferLengthTime12)
y13, x13 = zip(*ms._QList._BufferLengthTime13)
y11, x11 = zip(*ms._QList._INS1Blokage_Time)
y22, x22 = zip(*ms._QList._INS2Blokage_Time)
y33, x33 = zip(*ms._QList._INS3Blokage_Time)
y_2 = (y22 and y33)
```

```python
  Throughput.append(productionruns[i+1][1])
  WS1_Busy.append(productionruns[i+1][2])
  WS2_Busy.append(productionruns[i+1][3])
  WS3_Busy.append(productionruns[i+1][4])
  B1_OCC.append(productionruns[i+1][5])
  B2_OCC.append(productionruns[i+1][6])
  B3_OCC.append(productionruns[i+1][7])
  B12_OCC.append(productionruns[i+1][8])
  B13_OCC.append(productionruns[i+1][9])
  INS1_Block.append(productionruns[i+1][10])
  INS2_Block.append(productionruns[i+1][11])

  ws1total.append(Responsetimes[i+1][1])
  ws2total.append(Responsetimes[i+1][2])
  ws3total.append(Responsetimes[i+1][3])
  ins1total.append(Responsetimes[i+1][4])
  ins2total.append(Responsetimes[i+1][5])
  ins3total.append(Responsetimes[i+1][6])

Y_S = [("Workstation 1",ws1total), ("Workstation 2",ws2total), ("Workstation 3",ws3total),
       ("inspector 1",ins1total), ("inspector 22",ins2total), ("inspector 23",ins3total)]

PRR = [Throughput, WS1_Busy, WS2_Busy, WS3_Busy, B1_OCC, B2_OCC, B3_OCC,
       B12_OCC, B13_OCC, INS1_Block, INS2_Block]

data = [("Workstation 1",uws1), ("Workstation 2",uws2), ("Workstation 3",uws3),
        ("inspector 1",uinsp1), ("inspector 22",uinsp22), ("inspector 23",uinsp23)]

for sample in data:
  mean = np.mean(sample[1])
  u_0.append(round(mean,3))


for sample in PRR:
  mean = np.mean(sample)
  std = np.std(sample)
  var = np.var(sample)
  pmeans.append(round(mean,3))
  pstds.append(round(std,3))
  pvar.append(round(var,3))

n=0
Hypothisis = [' ']
crit = round(stats.t.ppf(q=1-0.01, df= i-1),3)
critical = ['t static critical \nfor 98% Confidence Level']
for sample in Y_S:
  n +=1
  mean = np.mean(sample[1])
  std = np.std(sample[1])
  t_0 = abs((mean-u_0[n])/(std/np.sqrt(i)))
  means.append(round(mean,3))
  stds.append(round(std,3))
  t_0s.append(round(t_0,3))
  critical.append(crit)
  if t_0 < crit:
    Hypothisis.append('H0, Accept')
```

```
  else: Hypothisis.append('H1, Reject')


productionruns.append(pmeans)
productionruns.append(pstds)
productionruns.append(pvar)
Responsetimes.append(means)
Responsetimes.append(stds)
Responsetimes.append(u_0)
Responsetimes.append(t_0s)
Responsetimes.append(critical)
Responsetimes.append(Hypothisis)
print("productionruns*******************", len(Responsetimes), len(Responsetimes[0]))

# print('ws1total',ws1total)
ms.reportSGeneration()
with open('Responsetimes.txt', 'w') as f:
    for item in Responsetimes:
        f.write("%s\n" % item)

with open('productionruns.csv', 'w') as f:
    for item in productionruns:
        f.write("%s\n" % item)
```