SYSC 4005 Project 2022

Study of Manufacturing Facility

By Harjap Gill and David Haighton

1.0 Problem Formulation

A manufacturing facility assemples 3 different types of products, P_1, P_2 and P_3. Products are made from 3 work stations, W_1, W_2, W_3. P_1 is produced by W_1. P_2 is produced by W_2. P_3 is produced by W_3. There are 3 components that are named C_1, C_2 and C_3. Products are made through workstations when components are supplied to them. W_1 requires component C_1. W_2 requires components C_1 and C_2. W_3 requires components C_1 and C_3. Each workstation has a 2 item buffer for each type of component that it requires. Buffers may only contain a specific type of component. There are 2 inspectors, I_1 and I_2, who are responsible for cleaning and repairing components before they are put into the buffer. I_1 inspects C_1 components. I_2 inspects both C_2 and C_3 components. I_2 will pick which type of component to inspect random. Inspectors contain an infinite supply of all components.

After being examined by the inspector, these components enter a workstation buffer which maps to their component type. The inspector will only send components to a buffer which allows the appropriate component type. The inspector's component will be sent to the buffer which has the fewest components in waiting. If there are multiple buffers that contain the fewest number of components in waiting, the workstations are ranked from highest to lowest priority as: W_1, W_2, W_3. If all buffers of that component type are full, the inspector who inspects that component type is considered blocked until a spot opens up. A blocked inspector must hold its component and not begin inspecting a new one until a buffer of that component type is not full.

2.0 Setting of Objectives and Overall Project Plan

Objective: How should the factory's policy for how Inspector 1 distributes Component 1 to the buffers and workstations?

Criteria:

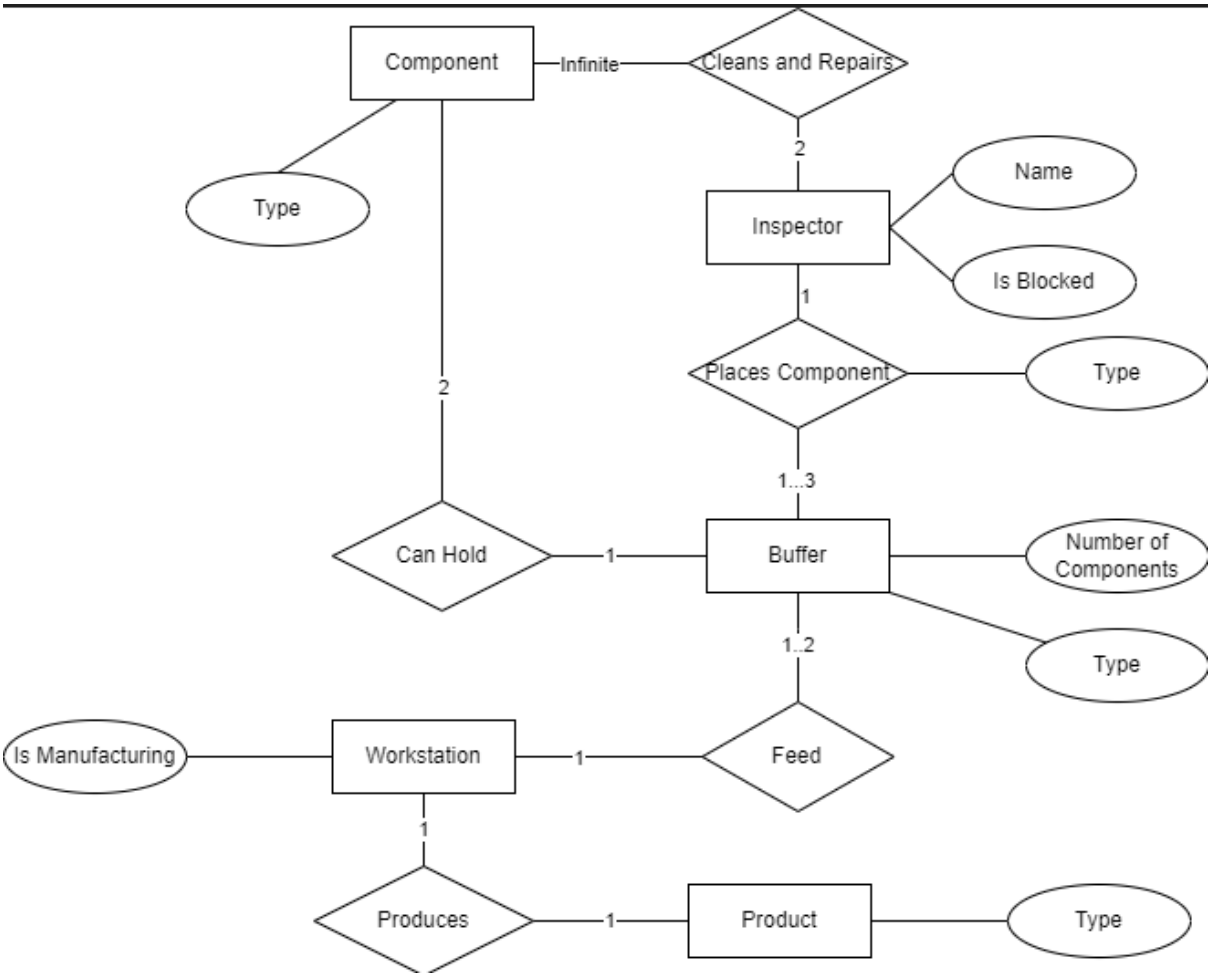
- Throughput of system (defined as products being manufactured over a period of time)
- Proportion of time where the workstation is busy
- Average buffer occupancy of each buffer
- Proportion of time where a inspector is being blocked
- Number of each component being made

Resources: The resources specified below are assuming if we were working only on this project and we have already learned all the content of this course.

The data collection will require no resources as the data from the factory is provided.
Model building will require 2 personnel and will take the time of 1 week.
Implementation of the model will require 2 personnel and take the time of 2 days.
Running the model will require 2 personnel and will take the time of 1 day.
Data analysis will require 2 personnel and will take the time of 2 weeks.

Running a simulation is a valid method for analysing this scenario. It would be very costly to make changes in the real world and then view their effect on the system. Modelling the changes in a simulation allows us to make an informed decision before real world changes are made.

3.0 Model Conceptualization



3.1 Entities and Relationships

Component: There are 3 different types of components. Buffers can hold components and inspectors clean and repair components.
Inspector: Inspectors clean and repair an infinite supply of components and place them into the buffer. Inspectors can be blocked. Inspectors have a name
Buffer: A buffer holds 2 components and has a type. It feeds the components into the workstation.
Workstation: A workstation produces a corresponding product when it is given the component that it requires. A workstation can be manufacturing a product or not.
Product: There are 3 different types of products. Products are produced by the workstations.

3.2 Assumptions

- When an inspector is blocked as there is no buffer to place its component, the inspector will hold onto its component. Once a buffer spot for its corresponding component opens up, the item will immediately be moved to that buffer and the inspector will be taken out of its blocked state

3.3 Events

There will be 2 event types which will be responsible for moving the system forward.

Component Done Event

- Event that occurs when an inspector is done inspecting a component and it can now be passed to a buffer

Product Done Event

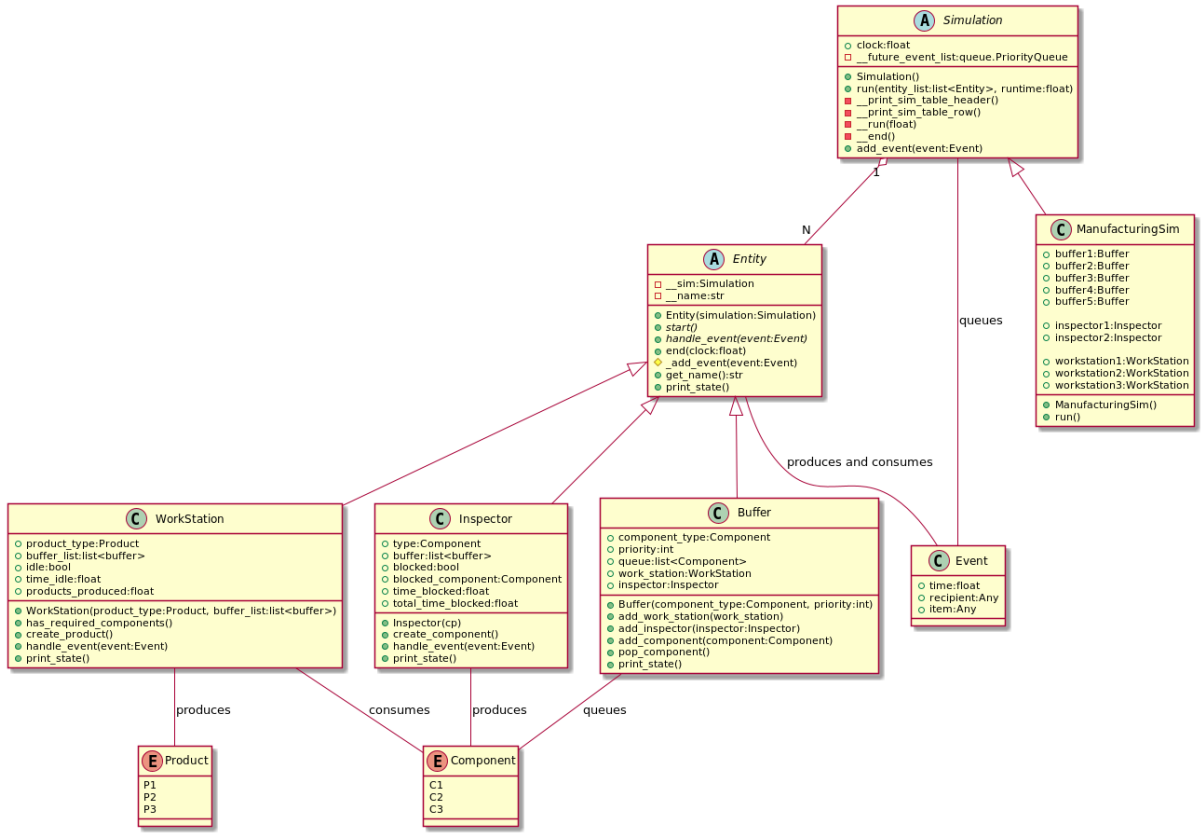
- Event that occurs when a workstation is done the production of a product

4.0 Model Translation

4.1 Choice of Simulation Language

When deciding on a language to use for this project, we decided to use the Python language. Both of us have extensively used Python before so we are familiar with the language. This will allow us to have an easier time implementing our functionality rather than attempting to learn the intricacies of the language. Also, as the sample code provided by the professor is in Python, we found it easier to create our simulation by basing it off that existing python file.

4.1 UML Class Diagram



4.1 Overview

In order to work on the simulation concurrently, we split the project into two parts: Framework and Business Logic/Model. The framework is designed to be a lightweight simple simulation library that can be easily interfaced with the business logic portion. The business logic encapsulates the model and interfaces itself with the framework.

As per the framework, there are three classes. These are Simulation, Event, Entity. The simulation is responsible for dealing with the future event list, notifying entities, and logging for each clock change. The Event is a data class used to encapsulate the event data caused by entities. Entities are responsible for adding to the Future Event List at the beginning and during the simulation, consuming events, and have limited logging abilities.

The model portion of the program contains the enums Product and Component, the WorkStation, Inspector, Buffer and ManufacturingSim classes. The enums are used to represent the different types of components and products being created or consumed. Workstation is responsible for encapsulating the responsibilities for the workstations in the model. The Inspector and buffer do the same for the Inspector and Buffer models respectively.

4.1 Code

View the files submitted to view the program.

5.0 Input Modelling

5.1 Identifying Distributions

The following matlab code is an example of what was used to analyse the distributions of the input data. It creates a histogram, creates a Q Q plot for the said distribution, does the chi^2 test and finally outputs the distributions parameters. 1 is returned from the chi^2 if we should keep the null hypothesis.

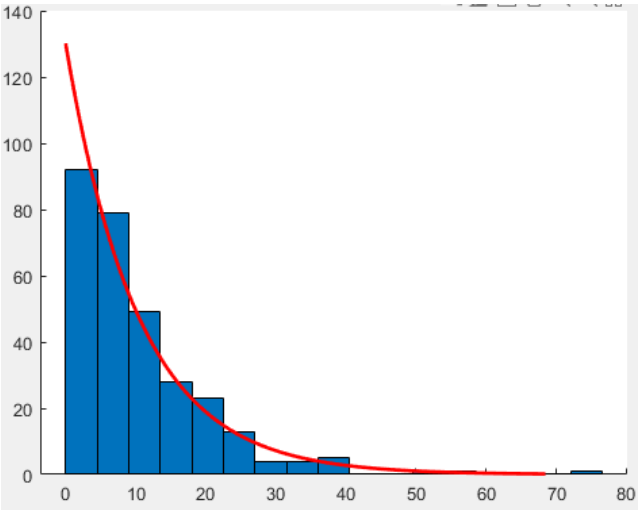
```
%read data
data = readtable('servinsp1.dat');
data = data.Var1;
figure, hold on
%histogram
histfit(data,round(length(data)^0.5),'exponential')
%qq plot
figure, hold on
exponential = fitdist(data,'Exponential');
qqplot(data, exponential);
v = var(data);
m = mean(data);
fprintf("mean %f",m)
%chi^2 test
fprintf("Exponential: %d",chi2gof(data,"CDF",exponential));

%print out parameters
disp(exponential)
```

Servinsp1.dat

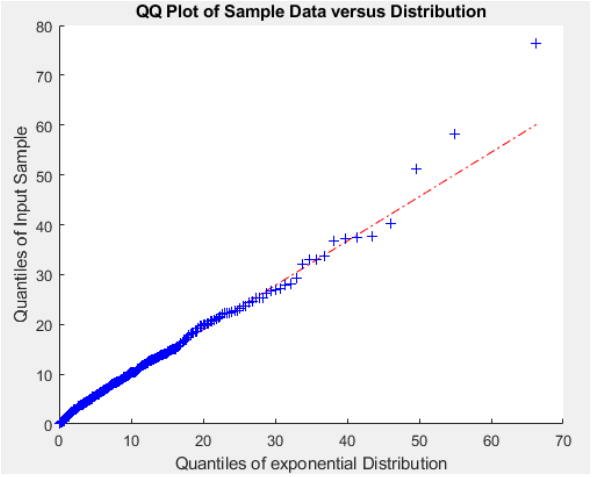
Exponential Distribution

- Estimated Parameters
  - μ = 10.3579 [9.27894, 11.6377]
  - λ = 0.09654

Histogram



QQ Plot



Chi Squared

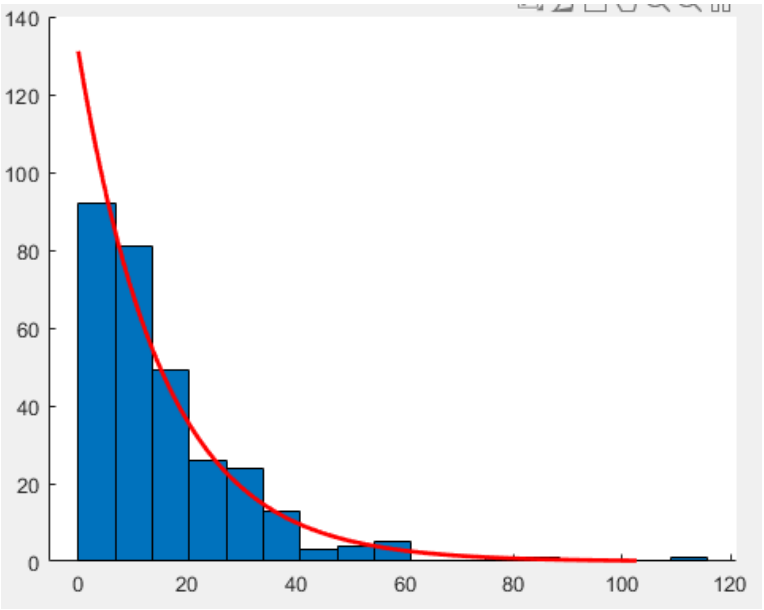
- Using matlab and the chi2gof() method, we passed in our data and compared it to an exponential distribution using the estimated parameters and it returned 0, meaning that we do not reject the null hypothesis at 5% significance that our data comes from the exponential distribution.

Servinsp2.dat

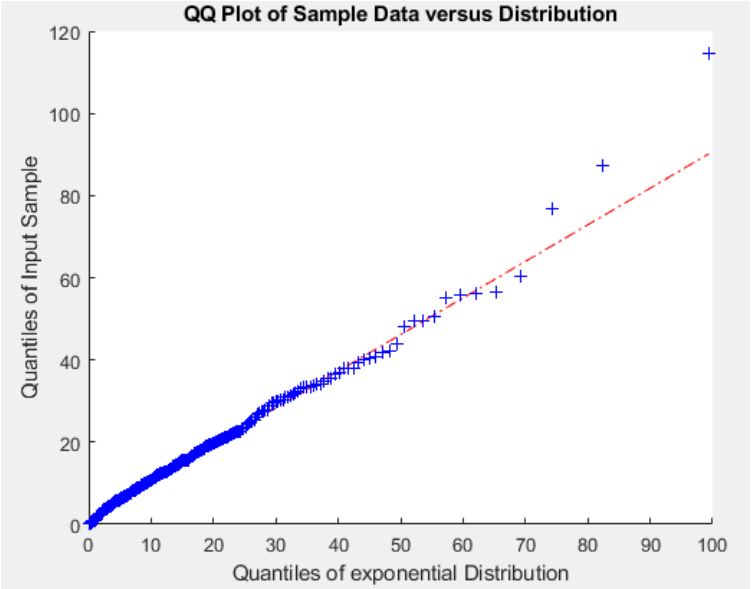
Exponential Distribution

- Estimated Parameters
  - μ = 15.5369 [13.9184, 17.4566]
  - λ = 0.06436

Histogram



QQ Plot



Chi Squared

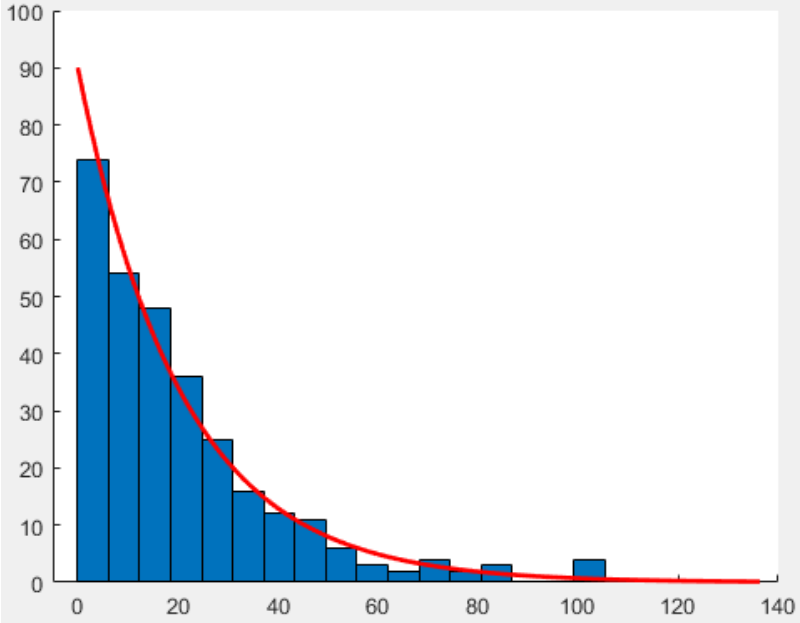
- Using matlab and the chi2gof() method, we passed in our data and compared it to an exponential distribution using the estimated parameters and it returned 0, meaning that we do not reject the null hypothesis at 5% significance that our data comes from the exponential distribution.

Servinsp3.dat

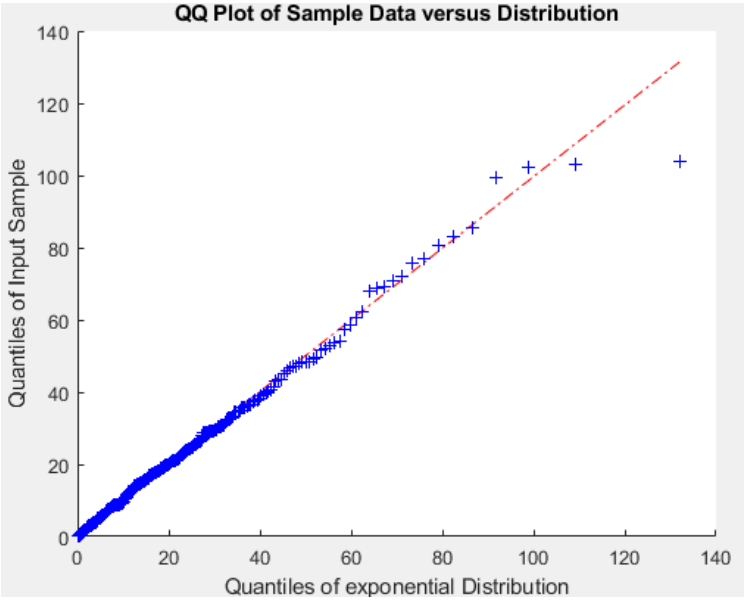
Exponential Distribution

- Estimated Parameters
  - μ =20.6328 [18.4835, 23.1821]
  - λ = 0.048467

Histogram



QQ Plot



Chi Squared

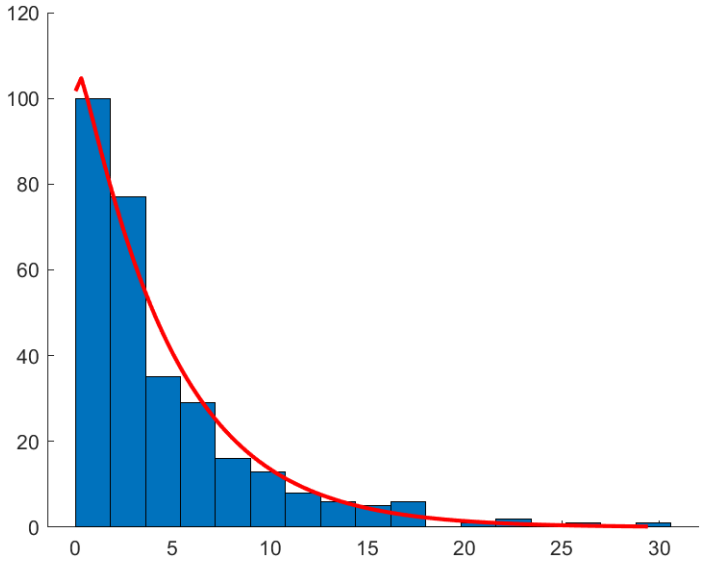
- Using matlab and the chi2gof() method, we passed in our data and compared it to an exponential distribution using the estimated parameters and it returned 0, meaning that we do not reject the null hypothesis at 5% significance that our data comes from the exponential distribution.

WK1_Analysis.dat

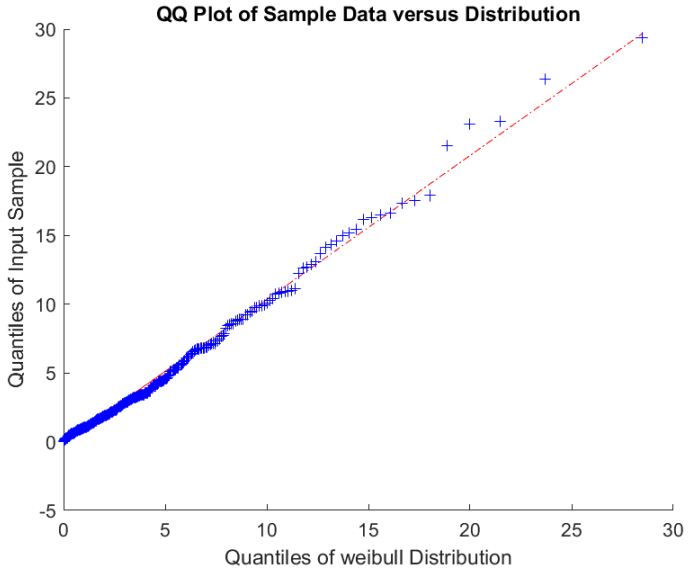
Weibull Distribution

- Estimated Parameters
  - A = 4.6521 [4.13938, 5.22832]
  - B = 1.02396 [0.939526, 1.11598]

Histogram



QQ Plot



Chi Squared

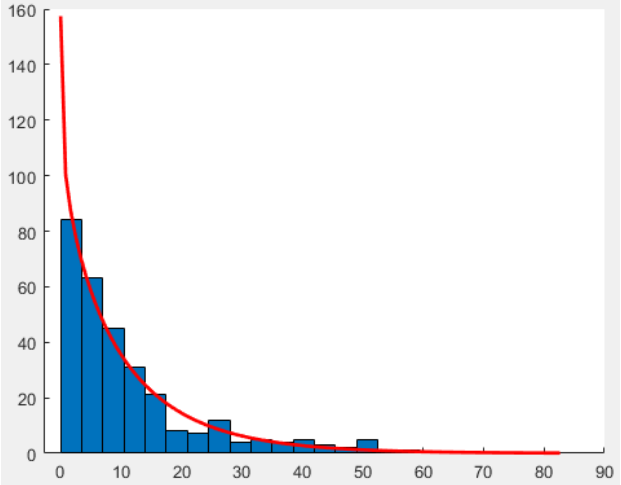
- Using matlab and the chi2gof() method, we passed in our data and compared it to an exponential distribution using the estimated parameters and it returned 0, meaning that we do not reject the null hypothesis at 5% significance that our data comes from the weibull distribution.

WK2_Analysis.dat

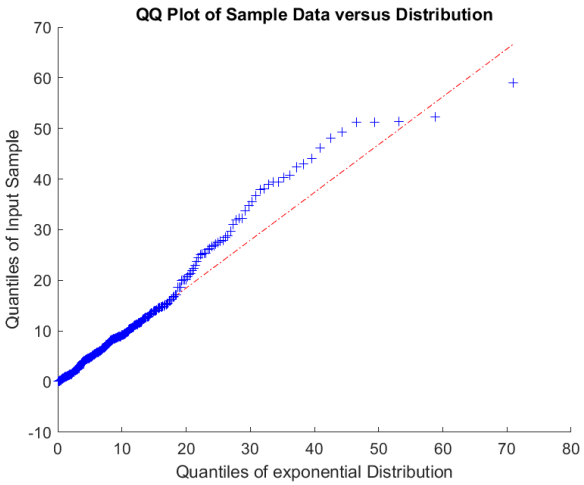
Weibull Distribution

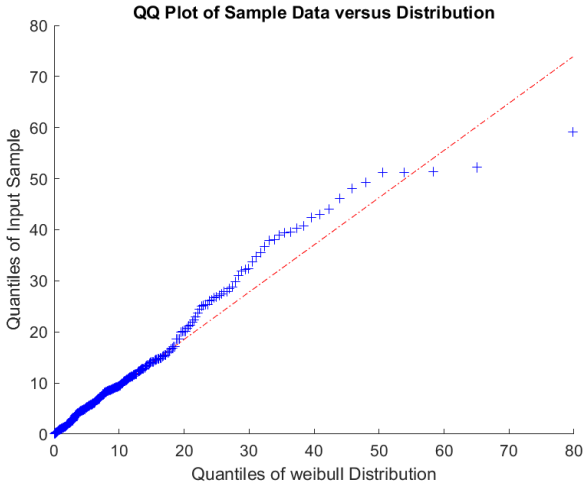
- Estimated Parameters
  - A = 10.6815 [9.38672, 12.1549]
  - B = 0.922808 [0.844718, 1.00812]

Histogram



QQ Plot





Chi Squared

- Using matlab and the chi2gof() method, we passed in our data and compared it to an exponential and weibull distribution as they both looked reasonable in the QQ plot. From the chi squared test we found that weibull passed while exponential failed. For this reason, we chose weibull distribution.

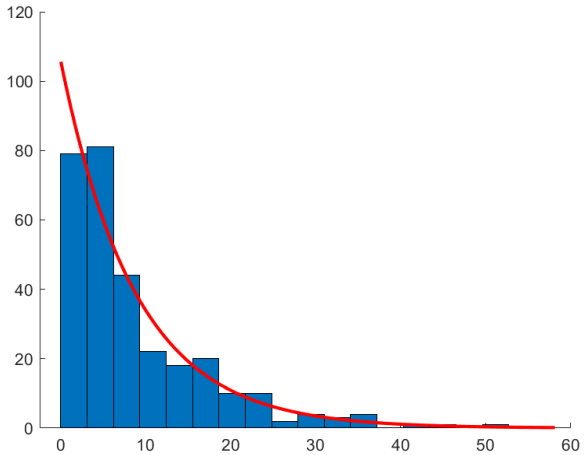
WK3_Analysis.dat

Exponential Distribution

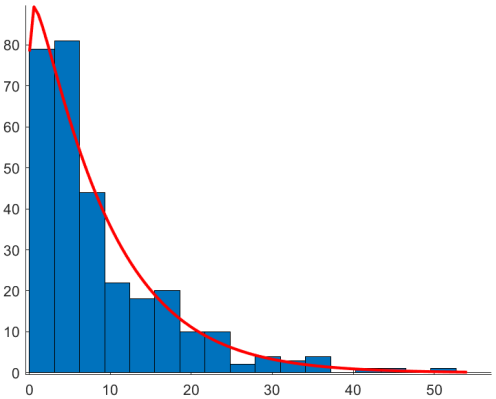
- Estimated Parameters
  - μ = 8.79558 [7.87935, 9.88233]
  - λ = 0.113693

Histogram

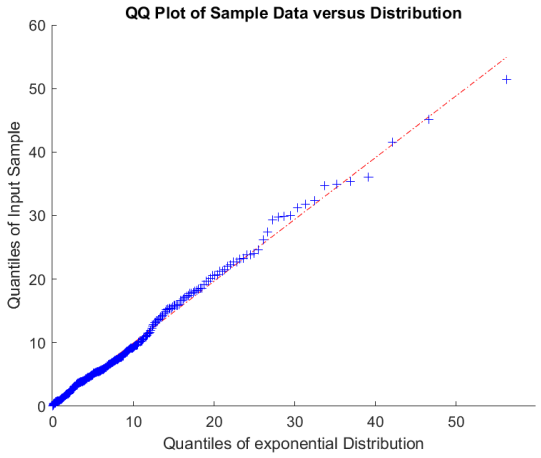
Exponential

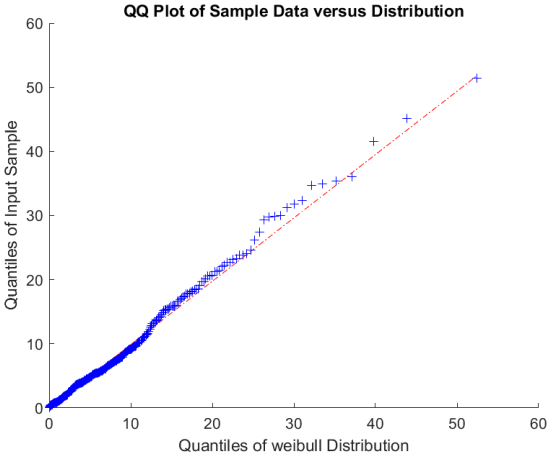


Weibull



QQ Plot





Chi Squared

- Using matlab and the chi2gof() method, we passed in our data and compared it to an exponential and weibull distribution as they both looked reasonable in the QQ plot. From the chi squared test we found that both weibull and exponential pass the chi squared test. We took a reasonable guess and chose exponential as the distribution.

Dataset	Chosen Distribution	Parameters
Servinsp1.dat	Exponential	λ = 0.09654
Servinsp2.dat	Exponential	λ = 0.06436
Servinsp3.dat	Exponential	λ = 0.048467
WK1_Analysis.dat	Weibull	λ = 4.6521 k = 1.02396
WK2_Analysis.dat	Weibull	λ = 10.6815 k = 0.922808
WK3_Analysis.dat	Exponential	λ = 0.113693

5.2 Generating Random Values

5.2.1 Generation of random value in a uniform distribution

To generate random variates, we first need to make random numbers in a uniform distribution. To perform this, we have a function which stores global variables for parameters m, a and c. To ensure good quality random numbers, we ensured m and a were coprime. Finally, we generated a list of pseudo random numbers. This process was done using a linear congruential generator as shown below.

```
def linear_congruential_generator(x0):
    return (a * x0 + c) % m


def generate_random_values(n, seed):
    values = []
    x = linear_congruential_generator(seed)
    values.append(x / 1024)
    for i in range(n-1):
        x = linear_congruential_generator(x)
        values.append(x / 1024)

    return values
```

5.2.2 Generation of random variates

After generating a uniform random distribution, random distributions for weibull and exponential must be made. To achieve this, the is a need to get the inverse cdf of these functions. Then the values are subbed into the equations and the output will yield a random variable of said distribution. View the code below for implementation details.

```python
import math


def generate_weibull_value(lam, k, x):
    return lam * (-1 * math.log(1-x)) ** (1/k)


def generate_exponential_value(lam, x):
    return (-math.log(1-x))/lam
```

5.3 Testing Random Variables

5.3.1 Frequency Test

We used matlab for testing the code random variate generation code we produced.. We first generated a series of 1024 numbers for each input which needs to be modelled (6 total for 2 different distributions). We used the built in function “fitdist” to fit the random data to the model it is supposed to represent, then used the K-S test to evaluate if the model being created is the same as the random variate distribution. K-S and Chi^2 naturally assume 5% p values. Below are examples of tests for exponential and weibull distributions. Although we could have tested them with virtually any parameters, we decided to test them with the parameters generated from the actual data (because that is what we will be using). That explains the names of the files we are using in said examples.

```
random = readtable('server3.dat');
random = random.Var1;

histfit(random,round(length(random)^0.5), 'exponential')

model = fitdist(random,'Exponential');


fprintf("Chi^2 (zero is pass):%d\n",chi2gof(random,"CDF",model));


fprintf("ks (zero is pass):%d\n",kstest(random,"CDF",model));
```

```
random = readtable('work1.dat');
random = random.Var1;
random = nonzeros(random);
histfit(random,round(length(random)^0.5), 'weibull')

model = fitdist(random,'Weibull');


fprintf("Chi^2 (zero is pass):%d\n",chi2gof(random,"CDF",model));


fprintf("ks (zero is pass):%d\n",kstest(random,"CDF",model));
```

Results are shown below for exponential and weibull:
Exponential:

```
>> tester
Chi^2 (zero is pass):0
ks (zero is pass):0
>> tester
```

Weibull:

```
>> tester
Chi^2 (zero is pass):0
ks (zero is pass):0
```

5.3.2 Autocorrelation Test

We will complete an autocorrelation test to ensure that our random number generation is generating random numbers which are independent of each other. This autocorrelation test was done using these equations with lag 1.

Test statistics is:

$$Z_0 = \frac{\hat{\rho}_{im}}{\hat{\sigma}_{\hat{\rho}_{im}}}$$

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

$$\hat{\sigma}_{\rho_{im}} = \frac{\sqrt{13M+7}}{12(M+1)}$$

This process was completed in python as shown below.

```python
def auto_correlation_test(values):
    m=1
    i=0
    M = 299
    sum = 0

    for k in range(M):
        sum += values[i+k*m] * values[i+(k+1)*m]

    p = (1/(M+1)) * sum - 0.25
    print(f'row {p}')
    o = (math.sqrt(13 * M + 7)) / (12*(M+1))
    print(f'sigma {o}')
    z = p / o
    print(f'Z {z}')

# Auto Correlation Test below
val = generate_random_values(300, 123)
auto_correlation_test(val)
```

```
row 0.007493680318196638
sigma 0.017333867513136328
Z 0.4323143875720531
```

As we are using 5% significance, we want to prove that our distribution falls within the inner 95% of the Z distribution. As this is the case, we want to ensure that our Z value is within the 2.5% threshold on their side of the normal distribution. $Z_{0.025} = 1.96$

Our Z value is equal to 0.4323. As -1.96 < 0.4323 < 1.96, we can ensure our hypothesis is not rejected with 5% significance and that our random number generation produces independent values.

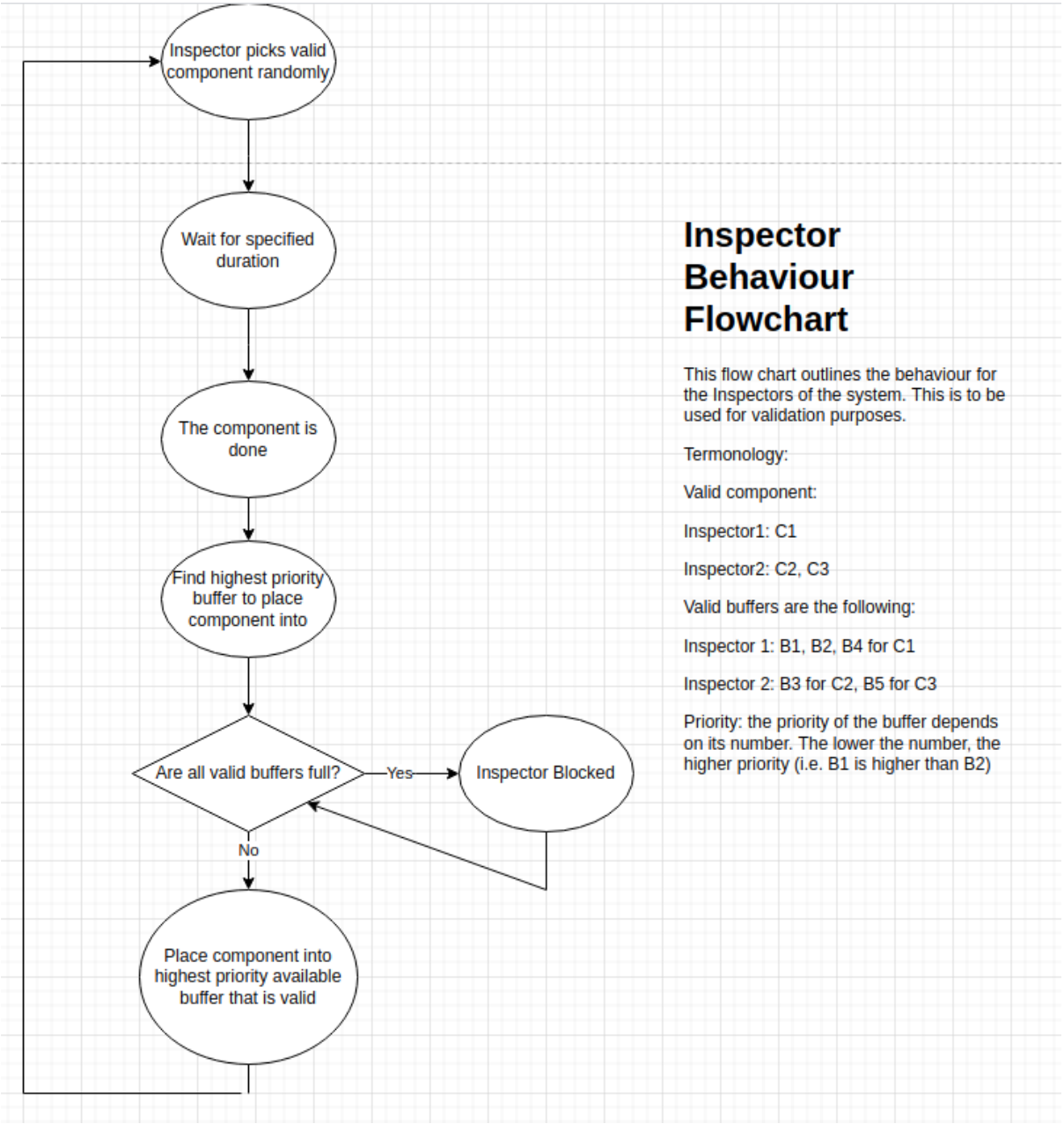
6.0 Verification

Verification is the method of ensuring that the output of our system matches the conceptual implementation of our code. This does not actually validate that our data is accurate for the real world implementation, it only ensures that our code implementation completes what we expect it to. This section will go over some of the methods we used to verify our model.

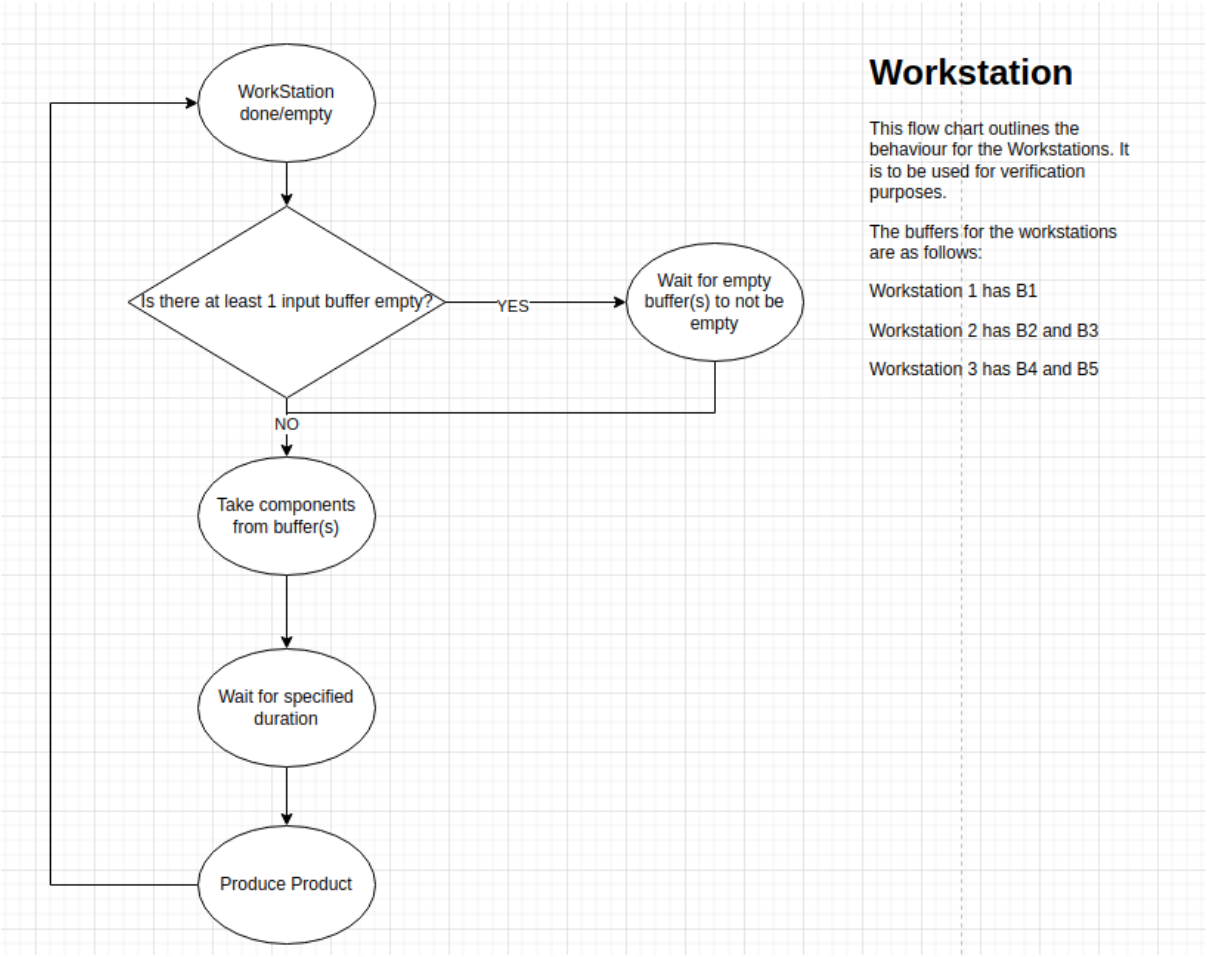
6.1 Flow Diagram

We created our flow diagrams which show every single logical decision that can be made. By creating this flow diagram and ensuring that our code handles all of these potential decisions. These flow diagram show the steps which take place when different events arrive to the system for each subsystem. By handling all of these events, we show that we are covering events which we may have otherwise missed.

6.1.1 Inspector



6.1.2 Workstation



6.1.3 Little's Law

Little's law is a way we could confirm if our queuing system is accurately queuing and holding components until they are actually used in our system. The only queuing that occurs in our system is within our 5 buffers so, we could use Little's Law to determine if our buffers are working correctly. Little's Law is defined as the equation below

$L = \lambda w$
$L = Average\ \#\ of\ items\ in\ buffer$
$\lambda = Arrival\ rate$
$w = Average\ system\ time\ item\ spends\ in\ buffer$

First we would measure the arrival rate of components coming into the buffers. This would tell us how often these components are arriving into our system. For each component we would then track how long it was held in the buffer. If we then multiplied the arrival rate of the components by their average time spent in the system, we would get an estimation of how many components are in the buffers at a given time.

We could also store the actual average amount of components in the buffers and then compare these 2 values to see if they are similar. If they are not similar, we know there is an issue with our project as we have most likely broken one of the Little's Law conditions.

1. The average arrival rate is equal to the average departure rate
2. All entered will eventually exit the system once completed

If these values are similar, then we can have another factor which proves our verification of the system.

6.1.4 Debug Event by Event

Another method of verifying our model was to print out every single event executed and the state of the model at that time into a CSV data file. Both team members then iterated through these events while keeping track of the state of the model. As we continued through all of the events, we were able to ensure that our handwritten model state was equivalent to the state within our CSV file. This gave use confidence that we were handling

6.1.5 Self Documenting

We attempted to ensure that our model makes sense and there are comments everywhere to ensure that it is documented on what is being attempted to be done. Having this documentation allows us to be more confidence on the verification of our model.

6.1.6 Ensuring that Products == Components

Another form of verification is ensuring that we have an accurate number of products produced based on the number of components that are used in our model. As we know that the creation of each product takes a fixed amount of components. We can then look at the number and type of components used and see if it matches the number of expected products.

$$\#\,P1\;Produced = \#\,C1\;Inspected - \#\,C1\;still\;in\;buffers - \#P1\;still\;in\;production - \#P2\;Produced - \#P3\;Produced$$
$$\#\,P2\;Produced = \#\,C2\;Inspected - \#\,C2\;still\;in\;buffers - \#P2\;still\;in\;production$$
$$\#\,P3\;Produced = \#\,C3\;Inspected - \#\,C3\;still\;in\;buffers - \#P3\;still\;in\;production$$

By ensuring these 3 equations are true, we can ensure that extra or too few products are not created based on the passed in components.

7.0 Validation

Validation is the act of comparing the model to actual system behaviour. This is made difficult for this project because there is no actual system behaviour to compare to. Therefore in order to validate, we will check the face validity and perform sensitivity analysis. A discussion about other possible methods under different circumstances will also be included.

7.1 Face Validity

In order to deem the model valid, we must first check its face validity. This is ensuring that the model appears resonable.To start, we must first make assumptions about the subsystems of the model and the model as a whole. This is made difficult as there is very little information about each subsystem. Firstly for the inspector, it would be a reasonable assumption that they are human and cannot inspect components any faster than 1 second per component on average and tend to inspect components faster than an hour. Given these assumptions, the inspector's face validity is okay. There is too little information about the buffers to make claims, however a buffer which holds 2 components would not be a cause for alarm. The workstation's validity is identical to the one for inspectors; however, I would expect the majority of components to be made in more than 1 seconds. As a whole, the system appears valid. This is because the average number of products produced per second compared to the number of things to process and produce components into products seems reasonable. Another thing to note is the number of products being produced. I would expect there to be a significant amount of P1's and few P2's and far less P3's. This appears to be true.

7.2 Sensitivity Analysis

Our model only has 2 inputs that are required to begin running: seed and stop time. To perform sensitivity analysis, we alter the inputs to our system and observe how those inputs being changed, results in differences in our output. Our model has a lot of randomness as the time to inspect components and the time to create products is generated by random. Due to this randomness, when our stop time is very low, we will see very large fluctuations in our outputs to our system. Changes in our seed will also result in changes to the random numbers generated. Luckily, the longer that the model runs for, the less the randomness matters as we will reach a steady state. This means that as stop time increases, the seed matters less and less.

I will set the stop time to 1000 and assume we will reach steady state by then.

Seed	123	984	695	326	764	817	556
Products Created	107	111	90	100	94	106	96

Here we can observe that as we change the seed, this does not drastically affect the number of products that are produced by the system, further helping us validate the mode.

7.3 Validation Alternatives

As stated before, actual system behaviour was not provided for the project and therefore all of the following statements are hypothetical. Firstly, historical data would be useful in validating the model. By having real world data collected from the system, we could use the real world input data as the input of the model and then after the model has been ran for a sufficient amount of time, we use hypothesis testing on the outputs of the simulation to ensure that the model predicts the same distribution and parameters for said distribution as the actual historical one. This will be very similar to input modelling, however it is done with outputs.

8.0 Production Runs and Analysis

8.1 Ensuring Steady State Analysis

An issue arises with our collected data as our data could be skewed to be lower than expected. This is due to our buffers being empty to begin. This will skew all of the buffer average times toward 0 as it will take time before these buffers begin to fill up. This will also skew workstation busy time along with products produced downward. If our simulation is not done for an adequate length of time, we have the potential issue of not reaching the steady state of the system. The steady state is the point at which we no longer have the bias of the initialization of the system. To determine if we have reached the steady state, we will look at the occupancy of the buffers to see at what point do the buffers fill up.

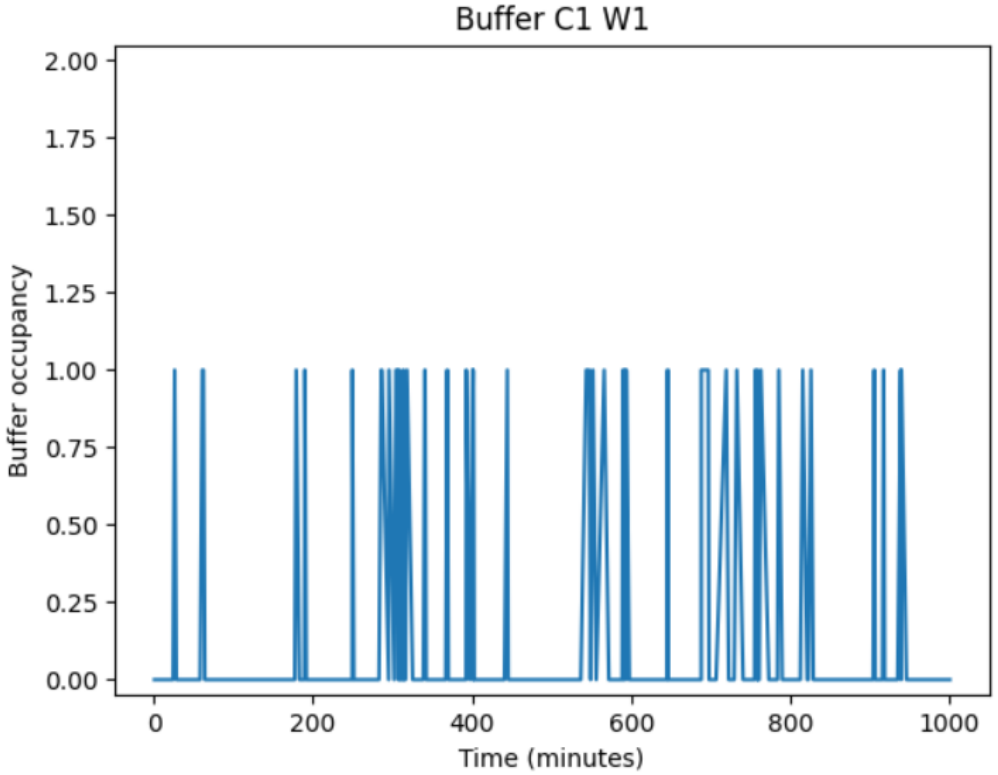


Figure 1: Buffer C1 W1 occupancy overtime during a run with seed 123

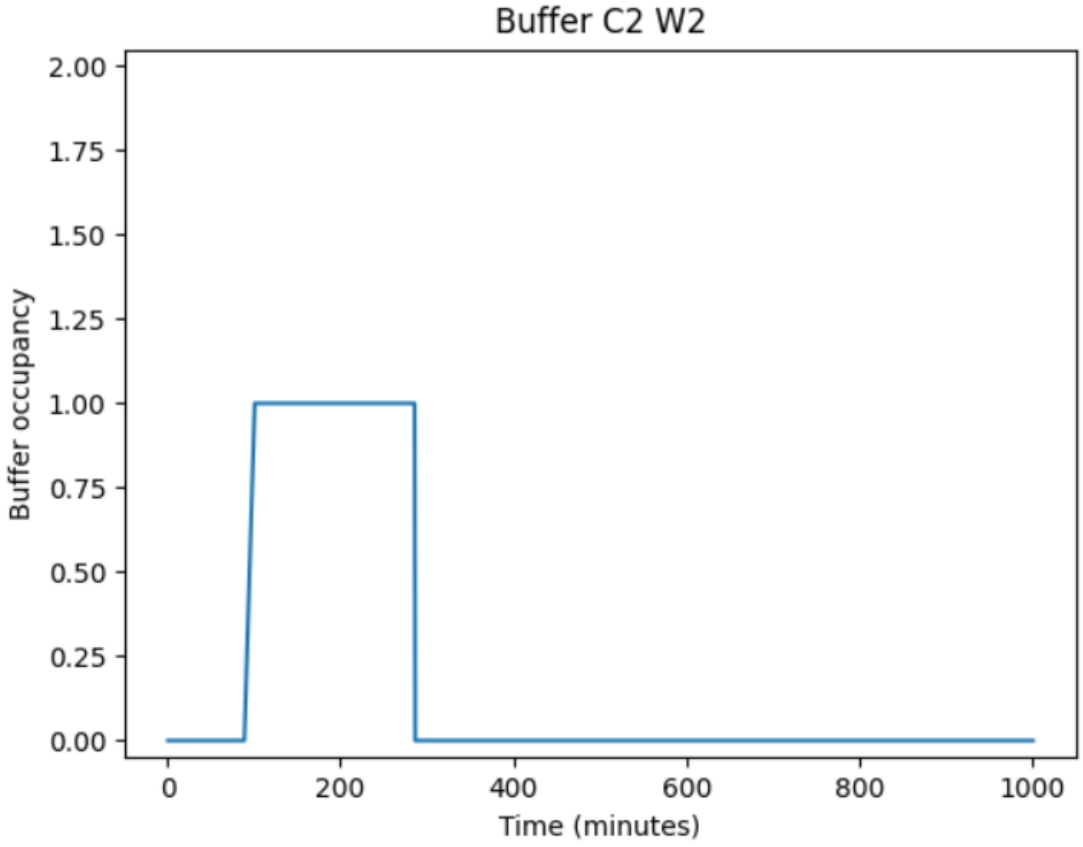


Figure 2: Buffer C2 W2 occupancy overtime during a run with seed 123

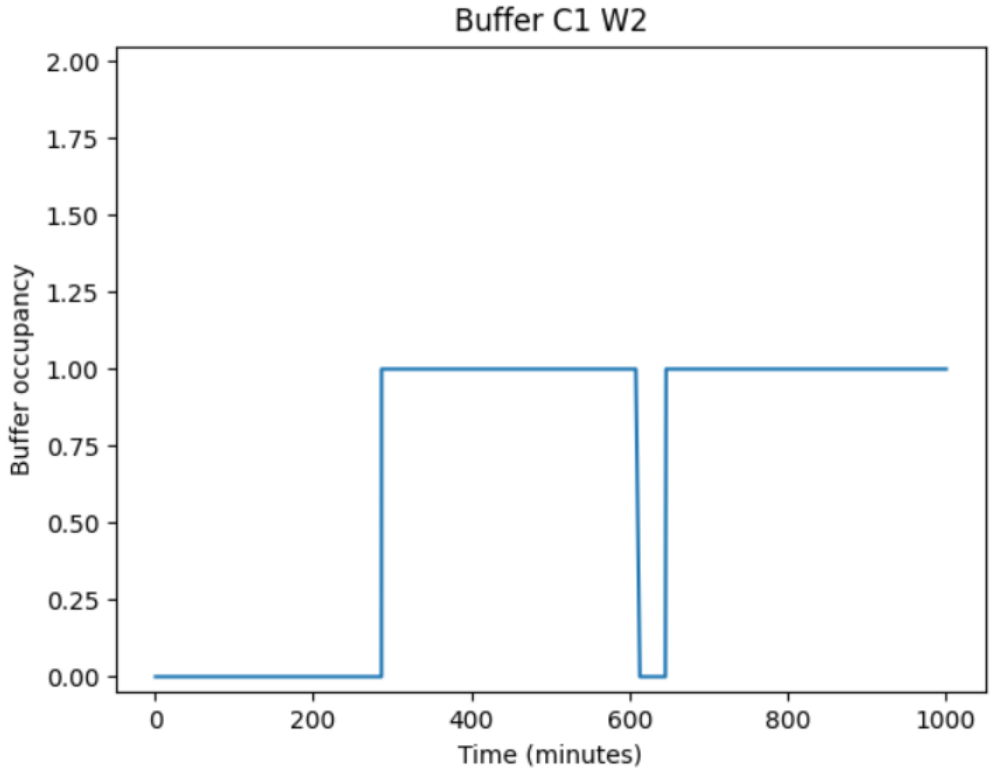


Figure 3: Buffer C1 W2 occupancy overtime during a run with seed 123

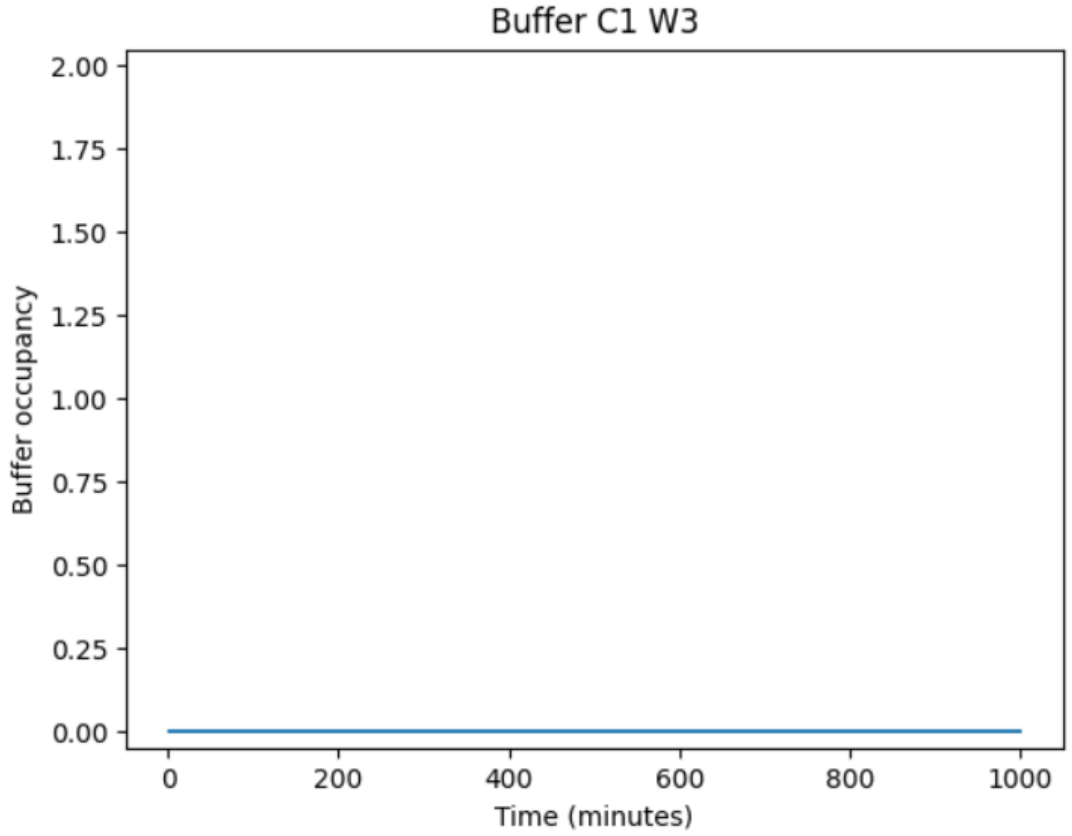


Figure 4: Buffer C1 W3 occupancy overtime during a run with seed 123

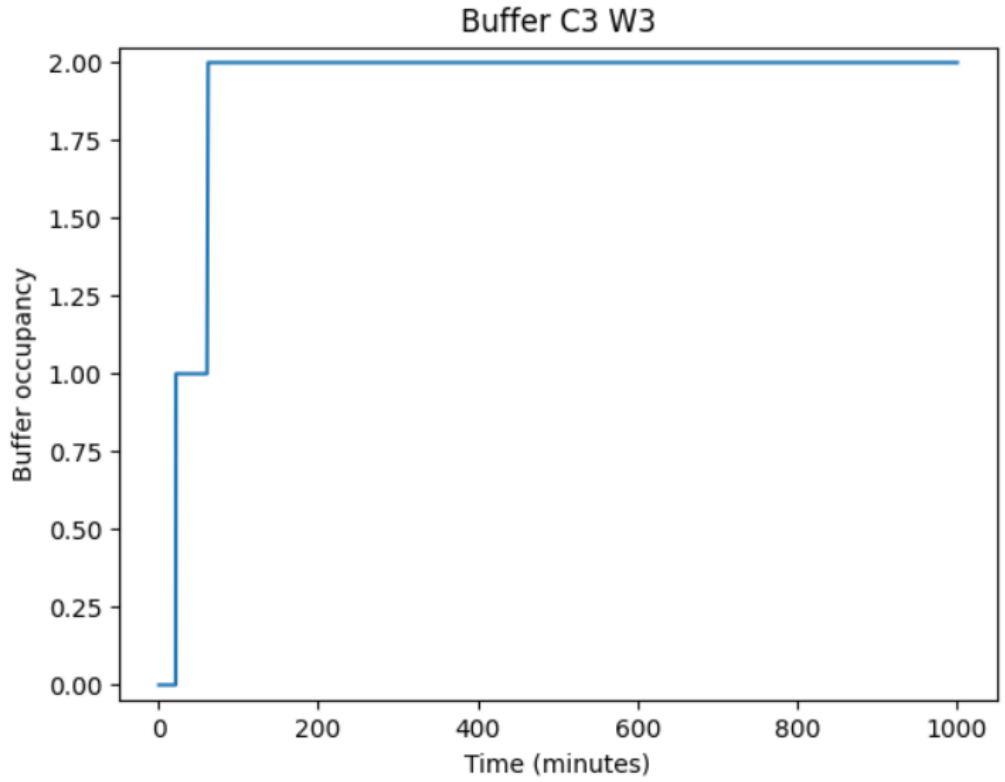


Figure 5: Buffer C3 W3 occupancy overtime during a run with seed 123

From the different buffer occupancy graphs, you can observe that most buffers do not drastically change their behaviour from the beginning. The only exception to this was found to be the two buffers found in Figure 3 and Figure 5. These 2 buffers seem to reach their steady state at time 250 and 50 respectively. As each of our replications is done until a time of 1000, we can be sure that the majority of each of our replications cover the steady state behaviour of our system.

8.2 Determining Number of Replications

To determine how many replications were needed we changed our code to allow replications to be run at once and their data was accumulated. When you run our main.py, you can enter the number of replications to perform. Our code also ensures that each replication is done with a new prime number to ensure that each replication we perform is seeded differently, ensuring we have different behaviour. To ensure we had enough replications we found that value of R that satisfied the condition.

$$R \geq \left(\frac{z_{\alpha/2} S_0}{\varepsilon} \right)^2$$

Which was found to equal 7.5 for all quantities. This was the minimum number of replications required. We then incremented R until its value of the equation below was found to be less than the value of R.

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

Value of R	**Value for Products Produced used for sample calculation**
8	10.93143279
9	10.39281998
10	9.998610533

This shows that 10 replications is the correct amount to use for our data. The rest of the analysis shown is based on 10 replications being used. All data that was collected can be found in the Appendix below.

8.3 Analysis

Statistic	Estimated Mean	Estimated Variance
Total Products Produced	92.4 ± 4.50	39.6
Throughput	0.0924	0.0000398
Product 1 Produced	81.2 ± 3.5336	24.4
Product 2 Produced	7.2 ± 1.9313	7.288
Product 3 Produced	4 ± 2.078	8.44
Inspector 1 Blocked time	0 ± 0	0
Inspector 2 Blocked time	744.3457± 55.21	5956.53
Inspector 1 blocked proportion	0	0
Inspector 2 blocked proportion	0.7443457±0.05521	0.00595653
Buffer 1 avg size	0.1257 ± 0.0239	0.001112
Buffer 2 avg size	0.28115 ± 0.1815	0.064338
Buffer 3 avg size	0.99965 ± 0.3905	0.29805
Buffer 4 avg size	0.0198 ± 0.02234	0.000975
Buffer 5 avg size	1.5614 ± 0.24495	0.117255
Workstation 1 busy time	373.7032 ± 39.8667	3105.85
Workstation 2 busy time	67.1347 ± 27.5976	1488.32
Workstation 3 busy time	35.7536 ± 16.3836	524.534
Workstation 1 busy proportion	0.0015614 ± 0.0024495	0.00310585
Workstation 2 busy proportion	0.3737032 ± 0.0398667	0.00148832
Workstation 3 busy proportion	0.035756± 0.0163836	0.00148832

The table above was done with 10 replications where each replication would last for 1000 time units. The number of replications was chosen by running replications and then using the equation provided in the slides in order to find if it is acceptable to stop the replications. All replication runs had their values line up within the confidence interval. All seeds used were prime numbers in order from the following table:

Replication	**seed**
1	987
2	911
3	919
4	929
5	937
6	941
7	947
8	553
9	967
10	971

As per analysis, the first table of the section which shows statistics against their predicted values is what will be used to draw conclusions. Said results indicate there is an issue when it comes to priority of which buffers Inspector 1 should place Component 1's on. This is the case because of the following insights. Inspector 1 is never blocked. Inspector 2 is blocked ~75% of the time. Workstation 2 and 3 are rarely busy compared to Workstation 1. Finally Buffer 3 and 5 (which inspector 2 puts components into) have a significantly higher average size. These factors point to an issue where there are very few product 2 and 3's produced. The reason why this happens is because Inspector 1 prioritises buffer 1 over the other ones.

9.0 Appendix 1

Products Produced

Trial	Value		
1	87	Mean	92.4
2	99	Variance	39.6
3	92	Confidence Interval	4.501635915
4	87		
5	97		
6	100		
7	100		
8	92		
9	83		
10	87		

Product 1 Produced

Trial	Value		
1	78	Mean	81.2
2	89	Variance	24.4
3	81	Confidence Interval	3.53360245
4	79		
5	80		
6	80		
7	89		
8	85		
9	75		
10	76		

Product 2 Produced

Trial	Value		
1	7	Mean	7.2
2	3	Variance	7.288888889
3	8	Confidence Interval	2.699794231
4	7		
5	11		
6	12		
7	5		
8	7		
9	7		
10	5		

Product 3 Produced

Trial	Value		
1	2	Mean	4
2	7	Variance	8.444444444
3	3	Confidence Interval	2.078778974
4	1		
5	6		
6	8		
7	6		
8	0		
9	1		
10	6		

Inspector 1 Blocked

Trial	Value		
1	0	Mean	0
2	0	Variance	0
3	0	Confidence Interval	0
4	0		
5	0		
6	0		
7	0		
8	0		
9	0		
10	0		

Inspector 2 Blocked

Trial	Value		
1	720.664	Mean	744.3457
2	768.1	Variance	5956.534
3	674.7452	Confidence Interval	55.21023
4	849.4855		
5	646.6796		
6	644.2643		
7	770.171		
8	865.1717		
9	723.4135		
10	780.7618		

Buffer 1 Average Size

Trial	Value		
1	0.102392	Mean	0.125704
2	0.162023	Variance	0.001116
3	0.139947	Confidence Interval	0.023897
4	0.061295		
5	0.166443		
6	0.166272		
7	0.112572		
8	0.113611		
9	0.119889		
10	0.112595		

Buffer 2 Average Size

Trial	Value		
1	0.037052	Mean	0.281156
2	0.835007	Variance	0.064338
3	0.04477	Confidence Interval	0.181451
4	0.172879		
5	0.11191		
6	0.398624		
7	0.402274		
8	0.03354		
9	0.43222		
10	0.343288		

Buffer 3 Average Size

Trial	Value		
1	1.74996401373706	Mean	0.99965218966187
2	0	Variance	0.298053742266079
3	1.07047056578735	Confidence Interval	0.39054408418986
4	0.964722617646612		
5	1.03414531940895		
6	0.571310480887129		
7	0.576388018173699		
8	1.81676134170426		
9	0.933596442885876		
10	1.27916309638776		

Buffer 4 Average Size

Trial	Value		
1	0.103114219526786	Mean	0.019826553207772
2	0.006326648131991	Variance	0.00097540938261
3	0.031368839361719	Confidence Interval	0.022341701165371
4	0		
5	0.012658661961494		
6	0.023397003510835		
7	0.003071553548161		
8	0		
9	0		
10	0.018328606036733		

Buffer 5 Average Size

Trial	Value		
1	0.906864009144412	Mean	1.56141073496319
2	1.95163854905939	Variance	0.117254758397836
3	1.63556074713369	Confidence Interval	0.244955960855152
4	1.90422309642072		
5	1.4538053268209		
6	1.74083386403721		
7	1.90114024945198		
8	1.13436823410759		
9	1.4754112905958		
10	1.5102619828602		

Workstation 1 Busy

Trial	Value		
1	378.963128216631	Mean	373.703198979935
2	454.256834527524	Variance	3105.85243844924
3	421.729101273039	Confidence Interval	39.8669657684767
4	265.183982711644		
5	441.043056949683		
6	360.867173113183		
7	348.448557181065		
8	344.290863956078		
9	381.550938932332		
10	340.698352938173		

Workstation 2 Busy

Trial	Value		
1	41.6533836573994	Mean	67.1347028198241
2	6.69175018345383	Variance	1488.32197036021
3	118.928562940348	Confidence Interval	27.5975939532607
4	47.8336567596546		
5	76.2121971232655		
6	131.625323748064		
7	62.4789471047758		
8	94.3228681534061		
9	37.9110270969169		
10	53.6893114309564		

Workstation 3 Busy

Trial	Value		
1	12.0117347762728	Mean	35.7536358696962
2	39.4274094758014	Variance	524.534419112592
3	61.6179371989	Confidence Interval	16.3836163518129
4	18.7122405149498		
5	64.8982891005169		
6	40.1442013757267		
7	36.7156566611481		
8	0		
9	20.1162860198788		
10	63.8926035737674		

10.0 Alternative Operating Policy

10.1 Policy

The alternative operating policy chosen was to reverse the priorities of the Inspector 1. This means Inspector 1 places its components in the following priority order: Workstation 3, then Workstation 2 and then finally Workstation 1.The following code within the Inspector class is the code changes

```
# Sort list by both buffer size and priority
# Regular mode
if self.alternative_mode == 0:
    valid_buffers.sort(key=lambda x: (len(x.queue), x.priority))
# Alternative Mode
else:
    valid_buffers.sort(key=lambda x: (len(x.queue), -x.priority))
```

10.2 Results

We applied the same method as per the determining number of replication sections and found 10 replications is sufficient for the analysis of the alternative policy. The calculations are provided in the excel sheet. The following table's mean differences are calculated by the system as is minus the system proposed. More detail can be seen in the excel file iteration_4_calculations.xlsx

Statistic	Estimated Mean Difference (neg means alternative higher)	Result
Total Products Produced	1.3 ± 5.881331031	Inconclusive
Throughput	0.00132±0.00588133	Inconclusive
Product 1 Produced	42.7±6.99060092	Alternative Value Lower
Product 2 Produced	-18.9±3.970913575	Alternative Value High
Product 3 Produced	-22.5±4.801119	Alternative Value High
Inspector 1 Blocked time	-0.364407687±0.565004819	Inconclusive
Inspector 2 Blocked time	696.0862±58.07612039	Alternative Value Lower
Inspector 1 blocked proportion	-0.0000364±0.0000565	Inconclusive
Inspector 2 blocked proportion	0.69608±0.058076	Alternative Value Lower
Buffer 1 avg size	0.073016836±0.032648944	Alternative Value Lower
Buffer 2 avg size	-0.364786494±0.203688658	Alternative Value Higher
Buffer 3 avg size	0.554802±0.399842	Alternative Value Less
Buffer 4 avg size	-0.80327±0.120037	Alternative Value High
Buffer 5 avg size	1.36933±0.250908	Alternative Value Less
Workstation 1 busy time	196.655±48.84908	Alternative Value Less
Workstation 2 busy time	-214.649±51.80009	Alternative Value More
Workstation 3 busy time	-205.491±60.90232	Alternative Value More
Workstation 1 busy proportion	0.196655±0.048849	Alternative Value Less
Workstation 2 busy proportion	-0.214649±0.05180009	Alternative Value More
Workstation 3 busy proportion	-0.205491±0.06090232	Alternative Value More

11.0 Conclusion

Our objective with this report was to answer the question, “How should the factory’s policy for how Inspector 1 distributes Component 1 to the buffers and workstations?”. To answer this, we simulated the existing system and compared it to the system after making policy changes.

The original design of the system was implemented and simulated across various replications. By simulating the system we were able to calculate various metrics which showed how effective the existing system is at creating products. Using the old system we found that we are generating on average 92.4 products per 1000 minutes. Of all products that were produced, 87.7% of the products were product 1s. This unbalance is further shown by inspector 2 being blocked for 74.4% of the time. With the current system, product 1s are used by the workstation 1 before they can reach other queues which require them. This indicates that workstation 2 and 3 along with inspector 2 are stuck waiting. If the factory needs a lot of product 1s, this system is effective; however, if they require a more equal amount of products between product 1,2 and 3, their system is not effective.

With our updated alternative strategy, we aim to eliminate the issues found above. With our new policy we changed the priority for where to place component 1s in the system. Now the system will place components 1s with the priority order:Buffer 3, Buffer 2, Buffer 1. This will allow us to first ensure our workstations that require multiple components are able to receive the components they need to free up the inspector when blocked on components 2 and 3s. With this alternative strategy, we found our distribution of product 1,2 and 3 became 42.26%, 28.65% and 29.09% respectively. This shows that with the alternative we have a much more balanced combination of products being produced. We also found that the inspector 2 block time went down to 4.8259%. This reduction in blocked time shows that our system now has less time being wasted and more time being productive. The workstation idle times are also reduced by utilising the alternative policy.

Our final conclusion with this report is that the factory should use this new policy if they are interested in generating more product 2 and 3s. The overall throughput of the system does not change with the policy change. The increase in product 2 and 3s also by extension decreases the number of product 1s. This decision must be taken by the factory to determine where their priorities lie.