

## BACHELOR

### Optimization of ticketing system for TIOBE Software B.V.

Swami, Dhvani

*Award date:*  
2023

[Link to publication](#)

#### **Disclaimer**

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

#### **General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

**TU/e**

**EINDHOVEN  
UNIVERSITY OF  
TECHNOLOGY**



**TIOBE**

⟨ the software quality company ⟩

Department of Mathematics and Computer Science  
Stochastics Operations Research Group

# Optimization of ticketing system for TIOBE Software B.V.

*Bachelor Final Project*

Dhwani Swami

Supervisors:

TU/e - Marko Boon, Jacques Resing  
TIOBE Software B.V. - Paul Jansen, Laurens Jansen

Final Report

Monday 3<sup>rd</sup> April, 2023



# Abstract

This study inspects the implementation of queueing theory to improve the ticketing system for TIOBE B.V. This study aims to reduce the waiting times, and queue lengths and improve the system efficiency. This is done by proposing a newer, modified version of their existing ticketing system. The research first analyses the existing ticketing system, and later expands to analysing the response of the ticketing system in various situations. Using these results, a recommendation is made which takes into account the number of servers, queue length and waiting times.

This thesis proves how the practical applications of queueing theory can be used to solve optimization problems in businesses such as TIOBE and how this theory can be used to improve day-to-day business operations.



# Contents

<b>Contents</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	2
1.2 Research Questions . . . . .	2
1.3 Significance . . . . .	3
<b>2 Model Description</b>	<b>5</b>
2.1 The Process . . . . .	5
2.1.1 Tickets . . . . .	5
2.1.2 Servers . . . . .	8
2.1.3 Process Flowchart . . . . .	8
2.2 Theory & Methodology . . . . .	10
2.2.1 Service Times & Arrival Times . . . . .	12
2.3 Assumptions . . . . .	12
2.4 Limitations . . . . .	14
<b>3 Simulation</b>	<b>15</b>
3.1 Overview of the simulated system . . . . .	15
3.1.1 Entities . . . . .	15
3.1.2 System Queues . . . . .	20
3.1.3 Workflows . . . . .	21
3.2 Events . . . . .	27
3.2.1 Arrival & Departure events for Queues . . . . .	27
3.3 Simulation description . . . . .	31
<b>4 Results</b>	<b>35</b>
4.1 Queue analysis . . . . .	35
4.1.1 Queue Lengths . . . . .	35
4.1.2 Queue Lengths at End-Of-Day . . . . .	39
4.1.3 Waiting Times in Queues . . . . .	43
4.2 Server analysis . . . . .	47
4.2.1 Work times of servers . . . . .	47
4.2.2 Server tickets analysis . . . . .	47
4.3 Tickets analysis . . . . .	48
4.3.1 Sojourn Time . . . . .	48
4.3.2 On the basis of Priorities . . . . .	48
<b>5 Improving the system performance</b>	<b>51</b>
5.1 Impact of insufficient information and tickets testing negative . . . . .	51
5.1.1 Build up between Open state and Analysed state . . . . .	51
5.1.2 Build up between Scheduled state and Verified state . . . . .	51

5.1.3	Effect on Mean Sojourn time . . . . .	52
5.1.4	Effect on Priority tickets . . . . .	54
5.2	Impact of adding servers . . . . .	54
5.2.1	Adding 28 engineers - Total 35 engineers . . . . .	55
5.2.2	Suggested Model . . . . .	59
<b>6</b>	<b>Conclusions</b>	<b>61</b>
	<b>References</b>	<b>63</b>
	<b>Appendix</b>	<b>65</b>
<b>A</b>	<b>Simulation Code</b>	<b>65</b>
A.1	Classes . . . . .	65
A.2	Distributions . . . . .	68
A.3	Future Event Set . . . . .	69
A.4	Priority Queue . . . . .	69
A.5	Queue . . . . .	70
A.6	Simulation Results . . . . .	71
A.7	Main Simulation . . . . .	74
A.8	Printing Aggregate results for multiple runs . . . . .	110

# List of Algorithms

1	Arrival Event on Open ( $S_1$ ) . . . . .	27
2	Departure event from Open . . . . .	27
3	Arrival Event on Accepted ( $E[i]$ ) . . . . .	28
4	Departure event from Accepted ( $E[i]$ ) . . . . .	28
5	Arrival event on Analysed . . . . .	28
6	Departure event from Analysed . . . . .	28
7	Arrival Event on Scheduled ( $E[i]$ ) . . . . .	29
8	Departure event from Scheduled ( $E[i]$ ) . . . . .	29
9	Arrival event on Implemented . . . . .	29
10	Departure event from Implemented . . . . .	30
11	Arrival event on Verified . . . . .	30
12	Departure event from Verified . . . . .	30
13	End-Of-Day Event . . . . .	31





# Chapter 1

## Introduction

Ticket lines can be found in hospitals, banks, retail stores, amusement parks, government agencies, and a variety of other service systems. Ticket lines are the queues where customers arrive with their requests or issues and wait to be served by a free server. Each customer is given a numbered ticket upon arrival. When a server becomes available, the ticket numbers are called out in sequence, and the holders of the tickets are served accordingly. Certain online services have also implemented ticket queues. (Xiao, Xu, Yao & Zhang, 2022) Most queues are first-come-first-served (First In First Out or FIFO) and are simulated by inputting the server's average service time (for a particular distribution). In this paper, Priority Queues (Singh, Albert, Mieghem, Gurvich & Mieghem, 2022) are modelled alongside dequeues (*docs.python*, n.d.). These are used to create diversity (of queues) that is appropriate for the problem and to support the complexity of the queueing system.

TIOBE B.V. is a software quality company and is specialized in assessing and tracking the quality of software. They do so by applying several software metrics to it. TIOBE checks several lines of software code for its customers each day. TiCS is their software quality framework that allows their customers to effectively measure and monitor the software quality of their software projects. (*TIOBE*, n.d.) Like any other company, TIOBE uses a ticketing system to track and prioritise customer issues and requests. A ticketing system is an IT service management platform. The ticketing system provides TIOBE with a platform that helps it in connecting to all its customers, organise their requests/issues and assign tasks internally. The platform of the ticketing system is hosted on a separate server which thus can be accessed by anyone within the organization. (Gohil & Vikash Kumar, n.d.) However, their ticketing system suffers from long queues, extremely high waiting times, and tickets being in the system for several years. To solve this issue, TIOBE proposed this project which can further aid them in making decisions on how to change and improve their ticketing system.

For most queueing models with multiple servers, it is assumed that servers have equal capabilities, and hence the choice of the server (to which the customer is assigned) doesn't make a difference (Garrido, 2009). However, in real life, and in the case of TIOBE, most servers<sup>1</sup> have quite different capabilities, skills and paces of working. The challenging task of analysing a combination of priority queues and dequeues in a multi-server system stems from the fact that jobs with different priorities might be in service at the same time (at different servers). So the Markov chain representation of the multi-class, multi-server queueing system appears to be necessary for tracking the number of jobs of each class. (Harchol-Balter & Wierman, 2005)

Queueing theory is a mathematical framework that provides an in-depth understanding of customer waiting times, server working time & idle time, and system performance. By applying queueing theory to the ticketing system, this paper aims to determine the most efficient way to

---

<sup>1</sup>Servers are employees and are an integral part of the software team. They actively work on tickets to resolve the issue or request the customer has.

manage customer demand, reduce queue lengths and reduce waiting times for requests/issues. (Adan & Resing, 2015) In doing so, it is hoped to provide valuable insights into how businesses such as TIOBE can improve their ticketing system and provide better service to their customers alongside improving their performance.

The significance of this study lies in its potential to provide practical solutions to such a common business problem. By optimizing the ticketing system using queueing theory, businesses such as TIOBE can improve customer satisfaction, reduce waiting times, reduce ticket build-up in their system and increase their operational efficiency.

## 1.1 Problem Statement

Several ticketing systems have long waiting times, high abandonment rates and low server productivity which could lead to customer dissatisfaction. This is something TIOBE wants to avoid.

A solution to this problem is to apply queueing theory to analyse and optimise their ticketing system. This thesis aims to investigate how queueing theory can be used to optimize the ticketing system and to develop results and suggestions for implementing these optimizations. Through analysis of the system's queueing characteristics such as arrival rates, service times, queue capacity and end-of-day queue lengths, the goal of this research is to identify the bottlenecks in the system and further propose solutions to improve the overall system performance. The subsequent objective of this thesis is to help TIOBE make improvements in their ticketing systems which will lead to better customer satisfaction and increased server productivity.

## 1.2 Research Questions

What are the best ways for reducing waiting times (for issues and requests to be serviced), increasing server productivity and improve system performance? How can queueing theory be used to improve the operation of the ticketing system? Which state are the biggest bottlenecks in the system?

In this paper, the analysis of several key performance indicators (KPIs) is performed. The KPIs discussed are as follows -

1. Waiting times - The amount of time an issue or request (from a customer) waits in a queue before it is served by a server.
2. Queue lengths - The number of issues or requests that wait (or pile up) before the server can work on them.
3. Number of tickets resolved - The number of tickets that are solved and closed, and are therefore incorporated into TIOBE's software.
4. Sojourn times and Mean Sojourn times - Sojourn time is the amount of time an issue/request spends in the system before being resolved and closed. The Mean Sojourn time is the average of all Sojourn times (of the resolved issues/requests).
5. Work and idle time of servers - The Work time of a server is the amount of time the server works on the request/issue before moving onto the next step in the system. The Idle time of a server is the amount of time the server does not work or is "idle" before a task is assigned to them.

Specifically, this research will investigate the following sub-questions:

1. How can queueing theory be applied to the ticket system in a busy service environment?

2. What are the primary bottlenecks and inefficiencies in the system, and how do they impact performance?
3. What are the most effective strategies for reducing waiting times, increasing server productivity, and improving system performance, and how can these strategies be implemented in the ticketing system?

Through answering these research questions, this thesis aims to give insights into how queueing theory can be used to improve the ticketing system and to develop suggestions for implementing these improvements in practice.

### **1.3 Significance**

With the results from the simulation, potential bottlenecks, the system's scalability and optimizing its performance can be identified. The simulation of the ticketing system also provides insights into the system's behaviour under various scenarios. For example, it is possible to simulate how the system would respond to an increase in the number of servers, which can help in identifying the potential performance increase/decrease of certain servers.

Hence, performing the simulation of the ticketing system is a valuable contribution as it provides a thorough analysis of the system's performance, recognising bottlenecks and helping in improving the system's performance. Moreover, the results of the simulation can be used to improve TIOBE's ticketing system and help them provide a better service to its customers.



# Chapter 2

## Model Description

In this chapter, the model used to simulate TIOBE's ticketing system is discussed. To imitate the real world, a certain number of assumptions are made which are further discussed in this chapter. Moreover, the limitations of these assumptions, the queueing model used and the important elements of TIOBE's ticketing system are discussed in detail.

### 2.1 The Process

#### 2.1.1 Tickets

Tickets in the context of ticketing systems are a method for tracking and resolving issues and requests related to the software being developed by TIOBE (TiCS)<sup>1</sup>. A ticket is a request or issue (sent to TIOBE) from customers, that should be worked on in order to resolve the customer complaint and improve their software. The tickets are created, managed and tracked using a ticket-tracking system called Redmine. TIOBE uses Redmine to organise, prioritise and manage tickets.

When a customer faces a problem/issue with their software or has a request for a new feature (in their next release), the customer informs TIOBE about it. Then, TIOBE creates a ticket for said request/issue and is then assigned to the appropriate server. After going through all the steps in the system, from analysing to testing & verification, the ticket is resolved & closed and then the customer is notified about the same. On average, TIOBE receives 1600 tickets a year and 400 per quarter. Generally, a ticket goes through 8-11 states before it exits the system or is rejected/abandoned.

#### States

The State of a ticket is the position in which the ticket waits for it to be picked up by the server for the next task/state. The name of the state signifies the task that has already been performed (on the ticket).

The number of given states of a ticket is 8, they are as follows,

1. *Open*  
The ticket has entered the system, has been opened and is waiting to be assigned to a server (for the next task).
2. *Accepted*  
The ticket has been accepted and is now waiting to be analysed. At this stage, the ticket is

---

<sup>1</sup><https://www.tio.be.com/products/tics/>

also assigned to the server that will be performing the analysis and implementation. Once analysed, the ticket leaves this state and moves on to the next one.

3. *Analysed*

Here, the ticket has been analysed and is waiting to be scheduled back to the server that analysed it. Once the ticket is scheduled, it moves on to the next state.

4. *Scheduled*

In this state, the ticket is scheduled for a later time for the server to implement. Once the ticket is implemented into TIOBE's software, the ticket leaves the Scheduled state and moves on to the next state.

5. *Implemented*

In this state, the ticket has been implemented and is waiting to be tested and verified. Once the ticket is tested and verified, it leaves this state to move on to the next one.

6. *Verified*

In this state, the ticket has been verified and is waiting to be closed (i.e., leaving the system) by the respective server.

7. *Rejected*

The ticket has been rejected due to certain issues<sup>2</sup> and has exited the system. This can occur at any stage of the system.

8. *Wait*

In this state, TIOBE is waiting for extra info or extra action taken from entities outside TIOBE. This could be a customer or it could be a supplier. Let's give an example of both to get this clear:

- (a) Waiting for a supplier: Some tool is not working and can't be fixed, only the supplier can. Then TIOBE is waiting for an update from the supplier that the problem has been fixed in a certain release.
- (b) Waiting for the customer: If a ticket has been submitted by the customer but there is vital information missing. Then TIOBE will ask the customer for this extra info and wait for it.

These states can be used to provide a clear picture of the status of each ticket, and to help manage the priorities and workload of the software team. The state of a ticket can be easily changed, and the changes are automatically recorded for future reference.

### Priorities of tickets

Ticket priorities are the levels of importance that are assigned to the ticket when it enters the ticketing system. The priorities are assigned on the basis of how critical the issue is, the urgency for the customer, how the issue impacts the customer's business and how complex the issue is. Assigning a level of priority is an important aspect as it aids the software team in targeting the more important and critical issues first.

TIOBE mainly works with 5 different types of priorities, Blocking, Urgent, High, Normal and Low. Their level of importance is as follows,

$$Low < Normal < High < Urgent < Blocking.$$

1. *Low, Normal, High*

These tickets are of almost the same priority (in the case compared to higher priorities), *High* is given more priority than *Normal* and *Normal* is given more priority than *Low*.

---

<sup>2</sup>An issue is already solved; the issue is not worth solving; the issue appears to be no issue at all.

2. *Urgent*

The ticket with a priority *Urgent* jumps the queue (or goes to the top of the queue, there aren't any blocking tickets). Once there is an *Urgent* ticket in the system, it needs to be solved within 2 weeks.

3. *Blocking*

*Blocking* is the most important priority. If a ticket with a priority of *Blocking* enters the system, every server at each state will drop everything to work on that ticket until it is closed and leaves the system.

### Categories and Components

Category and component (as seen in Figure 2.1) are characteristics or elements used to identify the tool in which the issue/request has occurred. The Category is used to classify the tickets on the basis of the type of task being performed on it. "Components" is used to categorize tickets based on the specific component of the TiCS software being worked on. This field aids in providing a more descriptive view of the work being performed and in tracking the progress of the component of the software.

The "Component" (of a ticket) is also used to categorize and organise information about servers (such as their skills). This information is used to track the progress of the server and to identify the trends and patterns in their work. There are 86 different components in the system and each ticket falls under a particular component and each engineer is specialised in a certain number of components (as seen in Figure 2.2).

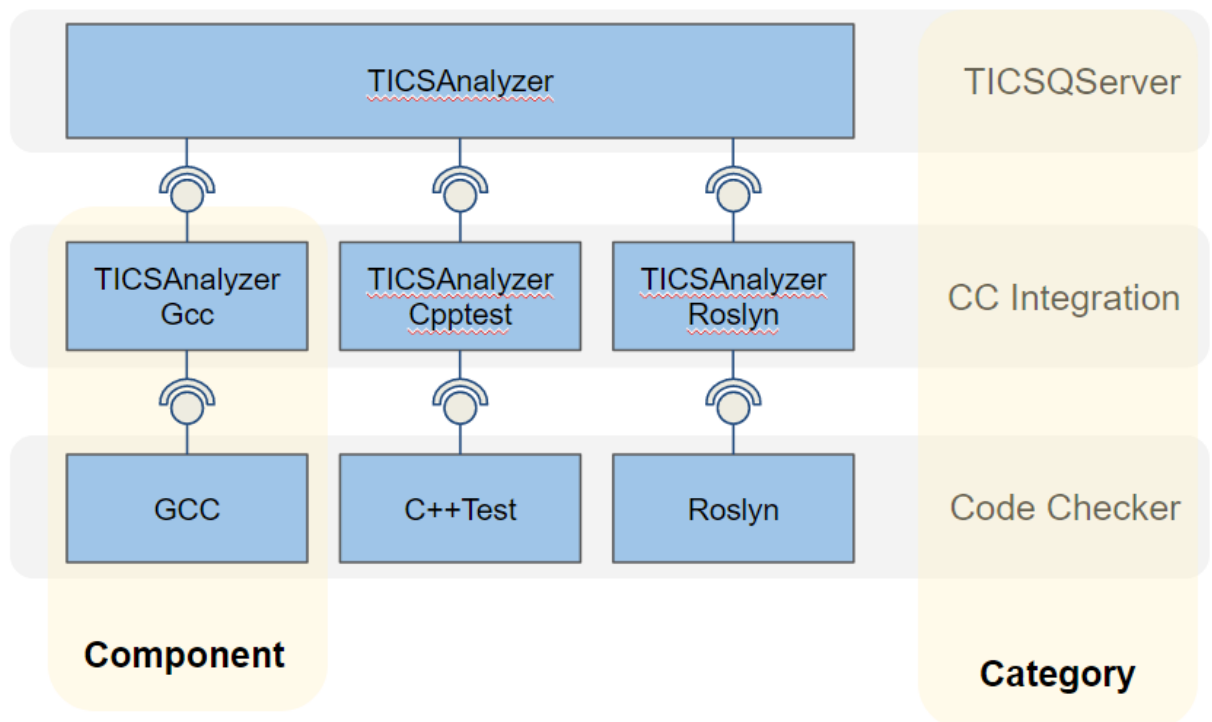


Figure 2.1: Components and Categories



## 2.1.2 Servers

”Servers” are the individuals responsible for working on the tickets and projects managed by TIOBE. They include developers, project managers, product managers, software engineers and other individuals who might be responsible for delivering the project. Servers are assigned certain tickets and projects on the basis of their roles in the organisation, skills and their work area of the software (the specific component(s) they work on).

There are three different types of servers, **Engineers**, **Testers** and **the Process Manager**. There are 7 engineers (as seen in Figure 2.2 - E.1 to E.7) and each engineer is skilled in a certain number of components. The engineers mainly have the job of analysing, implementing and testing the implemented tickets. Including the engineers, there are 8 testers, who work on the implemented tickets and test them to see if they are well implemented or not. If tested positive, the ticket goes on to the next state, otherwise, it is returned to the scheduled queue of the engineer. The Process Manager (denoted as  $S_1$  in this thesis) is the manager for the engineers and their job is to open, accept, schedule and close the tickets. The Process Manager also tests & verifies tickets in case there is a build-up of the Implemented state tickets. The queues for these servers are discussed in depth in Section 3.1.2.

Engineer	E_1	E_2	E_3	E_4	E_5	E_6	E_7
Tools	C++test	CPD	TFS	PMD	GCC	TICSc	Coverity
	VS C++ compiler (CL)	Roslyn	Visual Studio Code	TICSCil	Clang	TICScyclox	Compile.py
	dotTEST	Resharper	TFS (TFVC)	C#	CppCheck	TICSp	flake8
	IAR	IntelliJ		Java	Make	C	pylint
	Jtest	PMD		Jenkins	C++test	C++	ESLint
	VS C# compiler (CSC)	TICSCil		Eclipse	dotTEST		PC-Lint
	ARM	C#		TICSSQL	ARM		tslint
	Code Analyzer			CPD			Tsc
	mlint			Roslyn			Angular
	VS/MSBuild C++			VS C# compiler (CSC)			SCons
	EWP			Gradle			Tsc
	Cobertura			Maven			Python
	VS/MSBuild C#			Eclipse			JavaScript
	dotCover			Visual Studio			TypeScript
	.NetCore						Visual Studio
	OpenCover						PC-Lint
	ReportGenerator						Bamboo
	Keil						TeamCity
	Matlab						VS C++ compiler (CL)
	coverage.py						VS/MSBuild C++
	Bullseye						Cobertura
	Icov						VS/MSBuild C#
	Jacoco						.NetCore
	Istanbul						OpenCover
	Pycover						coverage.py
	LLVM						Bullseye
	MSBuild						Icov
	Coverity						Jacoco
	GCC						Istanbul
	Clang						Pycover
CppCheck						LLVM	
Resharper						MSBuild	
Make							

Figure 2.2: Engineers and their components

## 2.1.3 Process Flowchart

In this section, it is discussed how a ticket travels through the system, starting from its arrival until it leaves the system.

## Flowchart Description

Once the ticket arrives (Figure 2.3), it is opened by  $S_1$ . The ticket is assigned a certain priority based on customer request, urgency for the customer or how important it could be for the release. If the ticket is of high priority, it is put in the start of the Accepted queue, or else after the last ticket. Now, the ticket is assigned to one of the engineers by  $S_1$  to analyse and implement. Once the ticket is analysed and implemented, an engineer will pick it up to test it. This engineer is not the same as the engineer who analysed and implemented the ticket. If it tests positive, it moves on to be verified by the same engineer, else it will be sent back to scheduled (if tested negative). Then, it is closed by  $S_1$ . This process is discussed in depth in chapter 3.

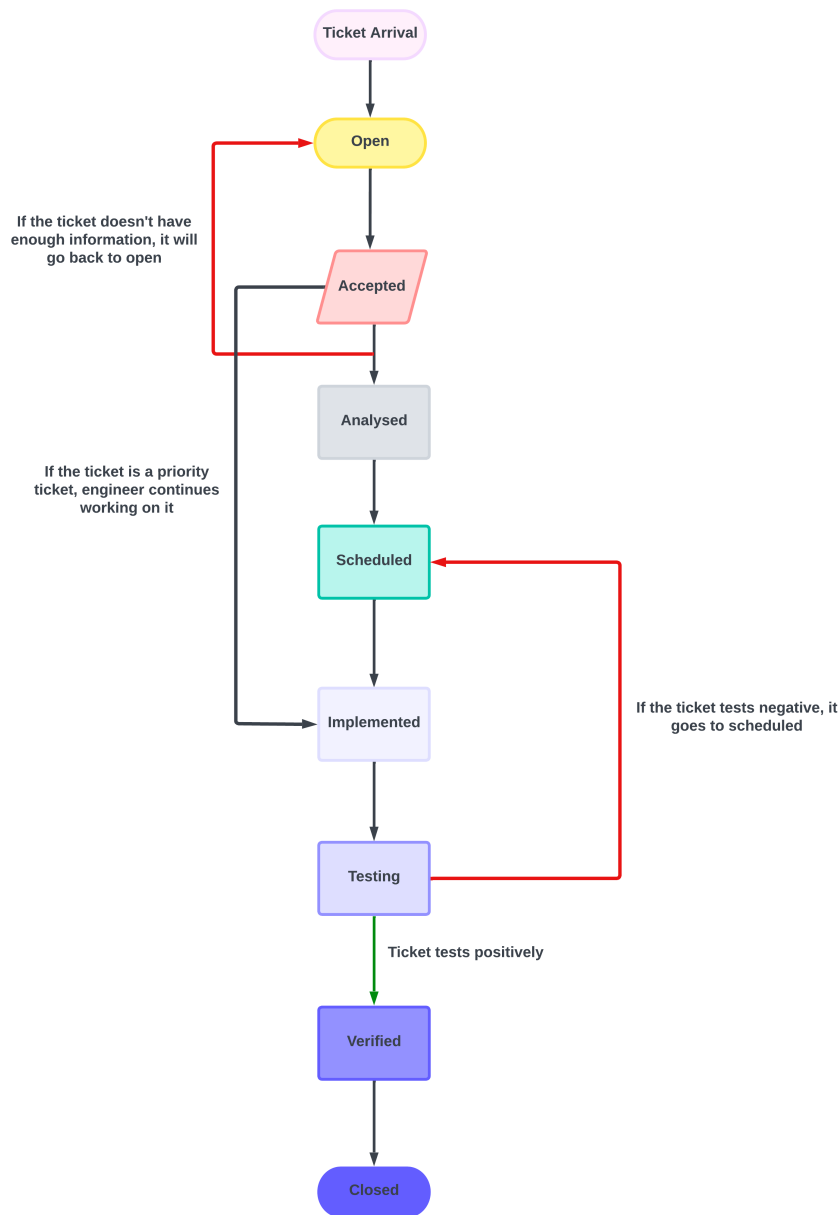


Figure 2.3: Process Flowchart

## 2.2 Theory & Methodology

A queueing network model (QN) is a collection of service centres representing the system resources that provide service to a collection of customers that represent the users. (Goos et al., n.d.) Queueing networks are used to model and analyse several real-life systems. In this paper, the dynamic assignment of servers to tickets is discussed. The number of tickets (or tasks) may exceed the capacity for service and the aim is to maximize the system throughput.

In this thesis, a multi-class network similar to that of Kelly and Laws (Kelly & Laws, 1993) is applied. Tickets of different types (i.e., different components) arrive at the network and go through the system by one of the several possible routes<sup>3</sup> and the route of the ticket depends on the type (in this case, component) of the ticket. The different routes that can be taken by the ticket can be seen in Figure 2.4. The aim is to reduce the number of tickets stuck in the system.

As much as the Jackson network is the simplest to apply, the ticketing system discussed in this thesis doesn't follow the necessary conditions for it to follow the Jackson network. A Jackson network has a certain number of stations, where each station represents a queue in which the service rate can be both station-dependent (different stations have different service rates) and state-dependent (service rates change depending on queue lengths). The tickets travel through the network on fixed routes. All tickets on a station belong to a single type and they have the same service-time distribution. As a result, there is no priority in tickets and all tickets are served on a first-come-first-served basis. (Goodman & Massey, 1984) However, in TIOBE's ticketing system, each ticket has a different type and on a station/system queue, multiple types of tickets can arrive. Furthermore, most of the queues in the system are priority queues. Therefore, it is unlikely that TIOBE's ticketing system follows the Jackson network.

A number of important metrics, such as the average number of tickets, average waiting time in a queue and average waiting time in the system characterise this model. These metrics are used to analyse the performance of the ticketing system in various conditions.

---

<sup>3</sup>A route in the network is the ordered set of service stations (or system queues) that the ticket goes through before it leaves the system.

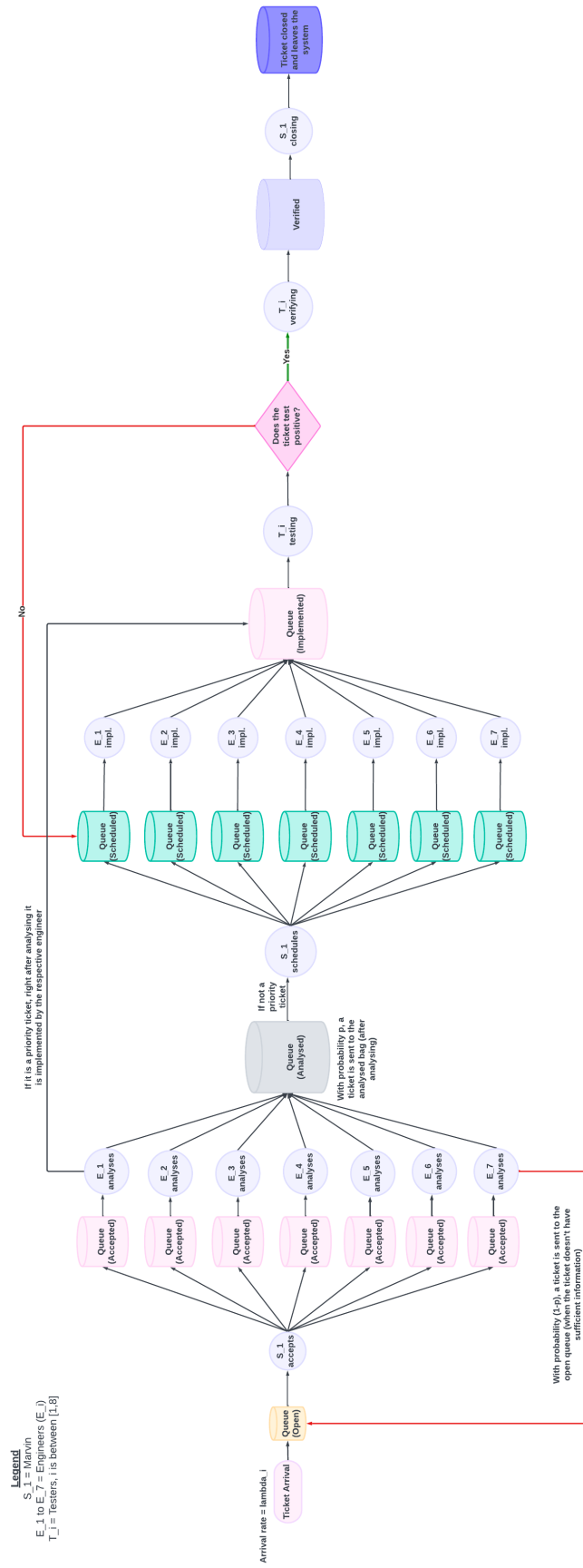


Figure 2.4: Ticket through the system

### 2.2.1 Service Times & Arrival Times

The distribution used for the arrival times is exponential, making it memoryless. This distribution is one of the widely used continuous distributions and is used to model the time elapsed between several events. The scale parameter is the inverse of the rate at which the tickets arrive over a period of time.

Now, the distribution used for the service times is an array of gamma distributions, the tickets arrive randomly and are served by the servers (in accordance with the gamma distribution). The gamma distribution is a flexible continuous distribution which is used to model the service time distributions at different states. This distribution is useful for simulating service times as the services require a non-negligible variability. The gamma distribution is used to model service times for multiple server queueing systems since the tickets are served by several parallel servers. To use the gamma distribution to model service times, the parameters of the distribution are estimated from historical data provided by TIOBE's Redmine UI.

The shape and scale parameters of the gamma distribution are estimated using means and variances. (*Statistical Compendium*, n.d.) These estimated parameters are then used to simulate service times in the queuing model. Given the means ( $E[X]$ ) and variances ( $\sigma^2$ ) provided for each state (from historical data), these can be expressed in terms of the scaling parameter  $\alpha = k$  and inverse scale parameter or rate parameter  $\beta = \frac{1}{\theta}$  where ( $\theta$  is the scale parameter) as,

$$E[X] = k\theta = \frac{\alpha}{\beta} \quad \& \quad \sigma^2 = k\theta^2 = \frac{\alpha}{\beta^2} \quad (2.1)$$

and now substituting the sample estimates to obtain the method of moments estimates, the estimated parameters are,

$$\hat{\alpha} = \frac{(E[X])^2}{\sigma^2} \quad \& \quad \hat{\beta} = \frac{E[X]}{\sigma^2}. \quad (2.2)$$

While these are estimated parameters and gamma distributions are used to best estimate the service times, future research of the company's historical data is necessary to more accurately identify the probability distributions of service times at different states. Future data analysis of the service times in different states would lead to a model that is closer to reality.

## 2.3 Assumptions

Assumptions are important as they affect the research approach and the results. The following assumptions were made to simplify the system while keeping the system close to reality -

### **Each server has unique components.**

It is assumed that no components have multiple servers (engineers), i.e., the engineers don't have any overlapping components. In the situation where this assumption is not made, the tickets would pile up for certain engineers and would unnecessarily overload certain engineers. This assumption was made to make sure that the tickets are distributed evenly. Having several engineers working on the same components of tickets increases the complexity of the simulation, hence making it difficult to analyse and interpret results.

### **Each component and priority is equally likely to be assigned to a ticket.**

To make sure that the tickets are distributed evenly amongst the servers, it is assumed that each component is equally likely to be assigned to a ticket. By making this assumption, the system's complexity is reduced and maintains the scope of this thesis. In addition, each priority is equally likely to be assigned to a ticket. This helps maintain variability in the kind of tickets entering

the system and further helps in the evaluation of the queues and how well the priority tickets are handled.

### **Rejected state not taken into account.**

Rejected tickets are assumed to have no impact on the ticketing system. The purpose of the ticketing system simulation is to optimize the service of successful tickets, and not understand the behaviour of rejected tickets. Here, the rejected tickets are assumed to be outside the scope of the simulation.

### **Wait state not taken into account.**

The wait state occurs mainly due to external factors such as waiting for suppliers and customers. Since such factors are difficult to model perfectly, it is simpler to exclude the Wait state (as this can add complexity to the simulation).

### **Independent server not taken into account.**

Not only does the Independent server have his queue (like the engineers), but he also has an additional workflow similar to  $S_1$  (apart from the workflow similar to the engineers). This server works with all states and works throughout the system. Since the Independent server works in all the state queues, each queue has different handling. It is seen that some queues require special handling (such as priority queues), and tickets from these queues need to be routed to a separate queue to ensure that they are processed appropriately. However, this can increase delays and complications in the handling of these tickets. Tickets need to be prioritized differently within each queue. To implement this server such that he has his queue and multiple other queues, it is quite complicated to implement his workflow into the simulation (while achieving close-to-reality results).

### **Holidays, meetings and weekends not taken into account.**

By including hours of holidays, weekends, meetings and other activities (where servers are not working on the tickets), the complexity of the simulation drastically increases. This makes the simulation more difficult to interpret and analysed. This assumption also reduces the complexity as the individual servers have different sick leaves and personal days.

### **Service times**

The service time distributions are in form of the gamma distributions, as discussed in section 2.2.1. However, the exact values that are used as parameters are approximations discussed with TIOBE. The mean service times and variances assumed for each state are,

1. Open state -  $\mu = 2$  minutes per ticket and  $\sigma = 39.37$  minutes,
2. Accepted state -  $\mu = 30$  hours per ticket and  $\sigma = 2.635$  hours,
3. Analysed state -  $\mu = 20$  minutes per ticket and  $\sigma = 18.811$  minutes,
4. Scheduled state -  $\mu = 20$  hours per ticket and  $\sigma = 5.099$  hours,
5. Implemented state -  $\mu = 10$  hours per ticket and  $\sigma = 14.164$  hours,
6. Verified state -  $\mu = 10$  minutes per ticket and  $\sigma = 48.591$  minutes.

## 2.4 Limitations

It's important to note that the system simulation can have certain limitations that can affect the accuracy of the predictions -

### Assumptions and simplifications

Although the assumptions and simplification (in section 2.3) are necessary to make the simulation feasible and so they might not exactly reflect the real-world system. For instance, the simulation assumes that all servers work on unique components, but that is not the case in real life.

### Complexity and scale

TIOBE deals with 1600 tickets every year (on average) and over 33,000 tickets in the past 21 years. As a consequence, the simulation might not be able to represent all of the relevant factors and interactions the company has with the customers, suppliers and internally due to the magnitude of tickets entering the system every year.

### Human behaviour

Simulations usually don't accurately reflect human behaviour. For example, the simulation of the ticketing system assumes that the servers follow a certain work pattern (i.e., workflow) when in reality the server behaviour is unpredictable and may vary widely from person to person. (for example, checking emails, and getting coffee and lunch breaks). Due to the complexity of implementing such behaviours, the servers adhere to their workflow and complex human behaviour is not taken into account.

### Unforeseen events

Sometimes, real-world events can deviate from the assumptions in unexpected ways. The simulation doesn't take these into account which might result in an inaccurate representation of the ticketing system. For example, the simulation doesn't take into account how the COVID-19 pandemic might have affected the work of the servers or influenced the inflow of tickets.

# Chapter 3

## Simulation

In this section, the simulation used to evaluate the performance of the ticketing system used by TIOBE is discussed. Simulating the ticketing system enables the analysis and modelling of the queueing model in different situations, aiding in predicting how the system would respond to different conditions. Furthermore, it is a cost-effective way to analyse and assess the system's performance before the changes could be implemented in the ticketing system of the company. Simulating the model can help identify areas of improvement in the ticketing system and how the improvements can be implemented. Lastly, due to the complexity of the system, mathematical analysis is difficult to implement. Hence, simulating the model helps one study such complex models with multiple servers and multiple queues.

The simulation model was developed using the discrete event simulation technique, which models the system's behaviour over time and analyses its performance under different situations. The model was built using multiple libraries in Python, and it includes different modules and scripts to represent the system's components (as mentioned in section 3.1.1). The simulation considers various parameters such as ticket processing time, ticket arrival rate and the number of servers available.

The results from the simulation provide valuable insights into the ticketing system's performance and helped in identifying potential bottlenecks. The outcomes of the simulation are used to improve and optimize the ticketing system which can then help the company provide better service to its customers. The simulation results provided valuable insights into the system's performance and helped in identifying potential bottlenecks and areas for improvement. The findings of the simulation were used to optimize the ticketing system and provide better service to the company's customers

### 3.1 Overview of the simulated system

#### 3.1.1 Entities

Entities refer to objects, variables, functions, and classes. All of these entities form the fundamental building blocks in the code, and they are used to model and simulate the ticketing system. The classes used in the code are described below.

##### **Ticket Class**

A ticket that enters the system has the following attributes:

1. **Attributes**



- (a) PRIORITY  
Each ticket is assigned a priority upon arrival into the system (as seen in section 2.1.1).
- (b) COMPONENT  
Every ticket that enters the system has a component that needs to be worked on (as mentioned in 2.1.1).
- (c) STATE  
There are 8 states under which a ticket could go through (as mentioned in 2.1.1). These states progressively change as the ticket goes through the system.
- (d) SERVER  
There are typically 3 servers who work on the ticket (as mentioned in 2.1.2).
- (e) POSITION (OR QUEUE NUMBER)  
This is the attribute which indicates the queue number to which the ticket is added. These queues are further discussed in section 3.1.2.
- (f) ARRIVAL TIME  
Arrival time refers to the time at which the ticket arrives at the particular state/queue.
- (g) SYSTEM ARRIVAL TIME  
System arrival time refers to when a ticket arrives in the system and is ready to be processed.
- (h) ENGINEER  
"Engineer" refers to the engineer that has been assigned to the ticket to perform analysis and to implement the ticket.
- (i) TESTER  
The tester attribute stores the tester that is allotted to the ticket (to perform testing).
- (j) TEST PROBABILITY  
This is the probability with which the ticket tests positive and has the approval to be verified (i.e., moving on to the next stage).
- (k) PROBABILITY  
This is the probability with which the ticket goes back to open if it has insufficient information, that is, it re-enters that system after a certain time once TIOBE has sufficient information to process the ticket.
- (l) TICKET NUMBER  
This is the "ID" or unique number assigned to each ticket for identification.

## 2. Functions

- (a) Shifting to new position in the system  
This function changes or shifts the ticket to its new position. It changes the "State" and "Arrival Time" attributes to do so.
- (b) Leaving the system  
"leaveSystem" function makes the ticket leave the system i.e., once it has been resolved and implemented.
- (c) Sorting  
The function "\_lt\_" compares the priorities of the tickets and helps sort the tickets in a queue.
- (d) Printing  
The function "\_str\_" prints the ticket number.

## Engineer

The engineer class has the following properties:

### 1. Attributes

- (a) COMPONENT  
There are several components under which a server could work (as mentioned in 2.1.1).
- (b) WORK TIME  
This attribute stores the total time the engineer has been working.
- (c) IDLE TIME  
This attribute stores the total time the engineer has been idle or not working.
- (d) TICKET  
This attribute is to identify the ticket the engineer is working on currently.
- (e) QUEUE  
"Queue" stores the tickets (objects of the Ticket class) which are in the engineer's queue. The engineer's queue comprises tickets in the Accepted state and Scheduled state.
- (f) IDLE  
In "Idle", a Boolean value of True or False is stored. It is set to True when the engineer is idle and False when the engineer is working.
- (g) COUNTS  
The following counts are used to track the frequency of the events in the simulation, and these are used to implement the workflow for engineers (Figure 3.2),
  - i. ACCEPTED  
This counts the number of tickets that have been analysed and are ready to enter the Analysed queue, i.e., the number of tickets that have been worked on from the Accepted Queue.
  - ii. IMPLEMENTED  
This attribute counts the number of tickets that have been worked on by the engineer from the Implemented Queue and are ready to be tested.
  - iii. SCHEDULED  
This count stores the number of tickets that have been implemented and are ready to enter the implemented queue, i.e., the number of tickets that have been worked on from the Scheduled Queue.
- (h) ENGINEER NUMBER  
This is the "ID" or unique number assigned to each engineer for identification.

### 2. Functions

- (a) Dequeuing a ticket  
The function "dequeue" removes a particular ticket from the engineer's queue ("Queue").
- (b) Setting the engineer as working  
This function adds the amount of time the engineer has been idle to "Idle Time" after storing the time the engineer begins working.
- (c) Setting the engineer as idle  
This function adds the amount of time the engineer has been working to "Work Time" after storing the time the engineer begins being idle.
- (d) Printing  
The function "\_str\_" prints the engineer number.

## Tester

### 1. Attributes

- (a) TICKET  
This attribute is to identify the ticket that is currently working on.
- (b) IDLE  
In "Idle", a Boolean value of True or False is stored. It is set to True when the tester is idle and False when the engineer is working.

### 2. Functions

- (a) Setting the tester as working  
This function adds the amount of time the tester has been idle to "Idle Time" after storing the time the tester begins working.
- (b) Setting the tester as idle  
This function adds the amount of time the tester has been working to "Work Time" after storing the time the tester begins being idle.
- (c) Printing  
The function "\_str\_" prints the string "tester".

## $S_1$ Server

### 1. Attributes

- (a) TICKET  
This attribute is to identify the ticket that is currently working on.
- (b) IDLE  
Here, similar to Tester and Engineer class, "Idle" is set to True when the tester is idle and False when the engineer is working.
- (c) COUNTS
  - i. OPEN  
This counts the number of tickets that have been opened by the server.
  - ii. IMPLEMENTED  
Similar to the Engineer's count, the "Implemented" count keeps track of the number of tickets that have been implemented or have been worked on by the server.
  - iii. VERIFIED  
This keeps the count of the number of tickets that have been closed by the server and have left the system.
  - iv. ANALYSED  
This attribute keeps track of the number of tickets that have been scheduled by the server to engineers.

### 2. Functions

- (a) Setting the  $S_1$  as working  
This function adds the amount of time the  $S_1$  has been idle to "Idle Time" after storing the time the  $S_1$  begins working.
- (b) Setting the  $S_1$  as idle  
This function adds the amount of time the  $S_1$  has been working to "Work Time" after storing the time the  $S_1$  begins being idle.
- (c) Printing  
The function "\_str\_" prints the string " $S_1$ ".

## Queue & Priority Queue

### 1. Attributes

#### (a) TICKETS

In this attribute, we store the list of tickets entering the particular queue.

### 2. Functions <sup>1</sup>

#### (a) Adding ticket to queue

The function "enqueue" adds the ticket (that is passed as a parameter) to the queue. For the priority queue, after the ticket is queued, it is automatically sorted on the basis of priority.

#### (b) Adding ticket to the start of the queue

The function "enqueue\_front" adds the ticket (that is passed as a parameter) to the front of the queue. For the priority queue, after the ticket is queued, it is automatically sorted on the basis of priority.

#### (c) Removing the first ticket

The function "dequeue[0]" removes the first ticket from the queue.

#### (d) Removing a particular ticket

The function "dequeue.ticket" removes the ticket (that is passed as a parameter) from the queue.

## Future Event Set

### 1. Attributes

#### (a) EVENTS

Events are added to and removed from this list, which has a changeable size. The order of these events should be determined by when they occurred. This structure will often be a "binary heap" kind. A priority queue, sometimes known as a heap queue, is what it is known as in Java and Python.

### 2. Functions

#### (a) Adding an event to the queue

The function "add" enqueues the event (that is passed as a parameter) to the "Events" queue. This queue is automatically sorted on the basis of the time it occurs.

#### (b) Returning the next event

The function "next" returns the next event from the "Events" queue. This queue is automatically sorted on the basis of the time it occurs.

## Events

### 1. Attributes

#### (a) TYPE

"Type" stores whether the Event is of type Arrival, Departure or End-Of-Day.

#### (b) SERVER

Here, the server who is responsible for handling the event and ticket is stored.

#### (c) TICKET

"Ticket" stores the ticket that has been involved in the event. For example, it stores the ticket whose departure event is being handled.

---

<sup>1</sup>For priority queues, each time a function is performed, the queue is sorted on the basis of priority.

- (d) TIME  
This stores the time at which the event is scheduled for.
- (e) CANCELLING OF TICKET  
The attribute "iscancelled" stores a boolean value which tells us whether the departure event is cancelled or not.

## 2. Functions

- (a) Sorting  
The function "\_lt\_" compares the time of the events and helps sort the events on the basis of the time in ascending order (the event that occurs first, goes first).
- (b) Cancelling event  
The "cancel" function sets the attribute "iscancelled" to true, which in turn helps to identify whether the event is cancelled or not.

### 3.1.2 System Queues

In a ticketing system, system queues are the collection and queue of tickets waiting at a particular state/stage that are waiting to be processed by the servers. When a ticket enters the system, it is added to a queue that further organizes the tickets based on their priority and/or arrival time. The queues ensure that tickets are handled in a timely and efficient manner, and higher priority tickets receive attention before lower priority ones.

#### Open State & Queue

The Open queue of type "Queue" (as mentioned in Section 3.1.1 under Queue & Priority Queue) is at the start of the ticketing system and is the queue that has all the newly submitted issues and tickets that have been opened and are waiting to be assigned to an engineer. When a customer submits a new issue/query, it is added to the open queue (after being opened) and is waiting to be assigned to an engineer.

The open queue's main purpose is to provide a preliminary classification of incoming tickets. The tickets are prioritised based on external and internal factors which in turn helps ensure that the most important tickets are processed first. The open queue ensures the prioritization of the tickets which avoids the servers from getting overloaded.

#### Accepted and Scheduled States - Engineer's Queue

The engineer's queues are all of type "Priority Queue" (as mentioned in Section 3.1.1 under Queue & Priority Queue). This queue has all the tickets that are assigned to the engineer for analysis (i.e., in the Accepted state) and implementation (i.e., in the Scheduled state). The tickets are sorted within the queue on the basis of priority and urgency, in turn allowing the engineers to focus on the most important tickets first. In addition to that, each engineer's queue is customized in such a way that it reflects their skills and tools (i.e., the components they work with). This aids in tickets being added to the queue of the right server (engineer), ensuring that the tickets are handled by the most appropriate personnel. Engineers monitor their respective queues and track the status of tickets and update the ticket details. Personalised queues allow the engineers to focus on the work being assigned to them without being bothered by the new tasks coming in.

#### Analysed State & Queue

The analysed queue of tickets (of type "Queue" - as mentioned in Section 3.1.1 under Queue & Priority Queue) is the queue of tickets that are waiting to be scheduled so that they can be implemented by the engineers in their software. The queue represents the backlog of tickets that have not yet been scheduled by  $S_1$  to the respective engineers and have been analysed.

### Implemented State & Queue

The Implemented queue of tickets (of type "Priority Queue" - as mentioned in Section 3.1.1 under Queue & Priority Queue) is the queue of tickets that have been implemented into the software by the engineers and are waiting to be tested and verified by the tester or a different engineer (than the one who implemented the ticket). The queue represents the "test backlog" that is, a list of tickets that need to be tested to ensure the quality of the software made by TIOBE. The test backlog is organized by priority.

The test backlog plays an important role in ensuring that the quality standards are met and provides a structured way to test tickets such that they can be tracked and monitored within the ticketing system. Due to regular updating and reviewing of the test backlog, TIOBE ensures that testing efforts are focused on the most critical areas and that quality issues are addressed in a timely manner.

### Verified State & Queue

The Verified queue of tickets (of type "Priority Queue" - as mentioned in Section 3.1.1 under Queue & Priority Queue) is the queue of tickets that have been tested and verified by the testers and are waiting to be closed by  $S_1$  so that they can leave the system. After the testing and verification process, the ticket is added to the Verified queue, and it is ready for closure and leaving the system. Once the ticket is closed, the respective customer is notified.

### 3.1.3 Workflows

A Workflow Flowchart is a pictorial representation that shows the sequence of events and tasks in the process. It depicts how the server navigates through tasks throughout the day.

#### $S_1$ workflow

##### *Workflow description*

$S_1$  mainly performs closure of tickets, scheduling of tickets (to respective engineers) and testing & verification of tickets if the test backlog is more than 20 tickets. As seen from figure 3.1,  $S_1$  first opens 5-6 tickets at the start of the day. Once this task has been completed, he checks whether there are any blocking and urgent priority tickets in the Verified queue that need immediate attention, if there are any blocking or urgent priority tickets, he will first work on them and close them. Next,  $S_1$  checks if there are any tickets in the verified queue, and if there are any tickets,  $S_1$  will close them and then move on to check for tickets in the test backlog. If there are any tickets in the test backlog, the server  $S_1$  will test 1-2 tickets. He then moves on to check if there are any tickets in the Analysed queue that are waiting to be scheduled. If there are any tickets to be scheduled,  $S_1$  will schedule 3-4 tickets. And lastly,  $S_1$  will check if the test backlog still has tickets, and if there are, he will test 1-2 tickets. Even after searching through the workflow, if there are no tickets for  $S_1$  to work on,  $S_1$  is set to idle and he checks for new tickets to work on from the beginning of his workflow (when new tickets enter the system). Every time the server is done working at a particular stage of his workflow, he continues working from where he left off.

##### *Dependencies*

The completion of certain activities in the workflow may depend on the completion of other activities. For instance, testing of tickets from the text backlog can only be done if the engineers have implemented tickets. The same applies when  $S_1$  has to schedule tickets to the engineers after they have been analysed. Moreover,  $S_1$  can only close tickets if there are tickets that have been tested and verified by the testers. This implies that the workload of the server  $S_1$  heavily depends on the work of the engineers and testers. The only task for which  $S_1$  does not depend on anyone is the opening of tickets (as the arrival of tickets is influenced by external factors and not internal system factors).

## Engineers workflow

### *Workflow description*

The engineers mainly perform three tasks - analysis of tickets, implementation of tickets (into the TiCS software) and testing & verification of tickets. Following the workflow in figure 3.2, the engineer first checks for high-priority tickets - blocking and urgent tickets. If there are any priority tickets in their queue, the engineer first works on them and then moves on to the next stage of the workflow. Now, once all the priority tickets are worked on, the engineer checks for tickets that are in the accepted state, i.e. the tickets waiting to be analysed (as seen in section 3.1.2 under "Accepted & Scheduled state - Engineer's Queue"). The engineer works on a maximum of 2 tickets in the accepted state (if there are any) and then checks for tickets in the test backlog. If there are any tickets in the test backlog, the engineer picks up the one they have not worked on, i.e. the engineer tests a ticket that another engineer implemented.

If the test backlog has a non-zero length, the engineer test 1-2 tickets and then moves on to the next stage of their workflow - checking if there are any tickets that were scheduled for them. The engineer checks in their queue if there are any tickets that are in the scheduled state i.e., there are tickets waiting to be implemented into the TiCS software (as seen in section 3.1.2 under "Accepted & Scheduled state - Engineer's Queue"). If there are tickets in the scheduled state in the engineer's queue, the engineer will implement a maximum of 2 tickets. Even after searching through the workflow, if there are no tickets for the engineer to work on, the engineer is set to idle and they check for new tickets to work on from the beginning of their workflow (when new tickets enter the system). Every time the server is done working at a particular stage of its workflow, they continue working from where it left off.

### *Dependencies*

Like the server  $S_1$ , the engineers have certain dependencies as well. The engineer, for example, needs to wait for  $S_1$  to open, accept and assign the tickets to them so they can start working on it and analysing it. Similarly, the engineer has to wait for  $S_1$  to assign (non-priority) tickets to them (from the analysed queue) so they can continue working on (i.e., implement) the tickets they analysed. Also, the engineer must wait for other engineers to implement tickets before testing them.

## Tester workflow

### *Workflow description*

The tester performs mainly one task - testing & verification of tickets. Following the workflow in figure 3.3, the tester first checks if there are any priority tickets in the implemented queue (that is the queue of tickets waiting to be tested and verified - as seen in section 3.1.2 under "Implemented state & Queue") and if there are, the tester will first test and verify these priority tickets and then move on to his next task. Now, the tester will check if there are any tickets in the implemented queue and if there are, the tester will work on them and test & verify them. If there are no tickets for the tester to work on (after all these searches), the tester will be set to idle. The tester checks for new tickets to work on from the beginning of his workflow (when new tickets enter the system). Every time the server is done working at a particular stage of its workflow, they continue working from where it left off.

### *Dependencies*

The tester heavily depends on  $S_1$  and the engineers to complete their tasks so he can start working.  $S_1$  should be scheduling and assigning tickets on time and the engineers should finish analysing and implementing their tickets on time. If these tasks are not performed timely, the tester has no ticket to work on.

### **Difference in workflows**

The main differences between the engineers, tester and the server  $S_1$  working patterns are -

1. The engineers are the only servers who have specific skills (i.e., components). This server can only work on a certain type of components, whereas,  $S_1$  and the tester have no such restriction.
2. The engineers have their own queue (where tickets of state Accepted and Scheduled enter) and work on the Implemented queue, while the server  $S_1$  works on multiple queues and the tester works on just one queue. Moreover, the tester only works on the Implemented queue (as his main tasks are testing and verifying).
3. While the server  $S_1$  has a start-of-day task (opening of tickets), the engineers and tester continue working on what they were working on the previous day.
4. While the server  $S_1$  and the engineers have counts to keep track of how many tickets they work on (in a particular state), the tester keeps working on the implemented queue without keeping a queue (as testing and verification are his primary tasks).



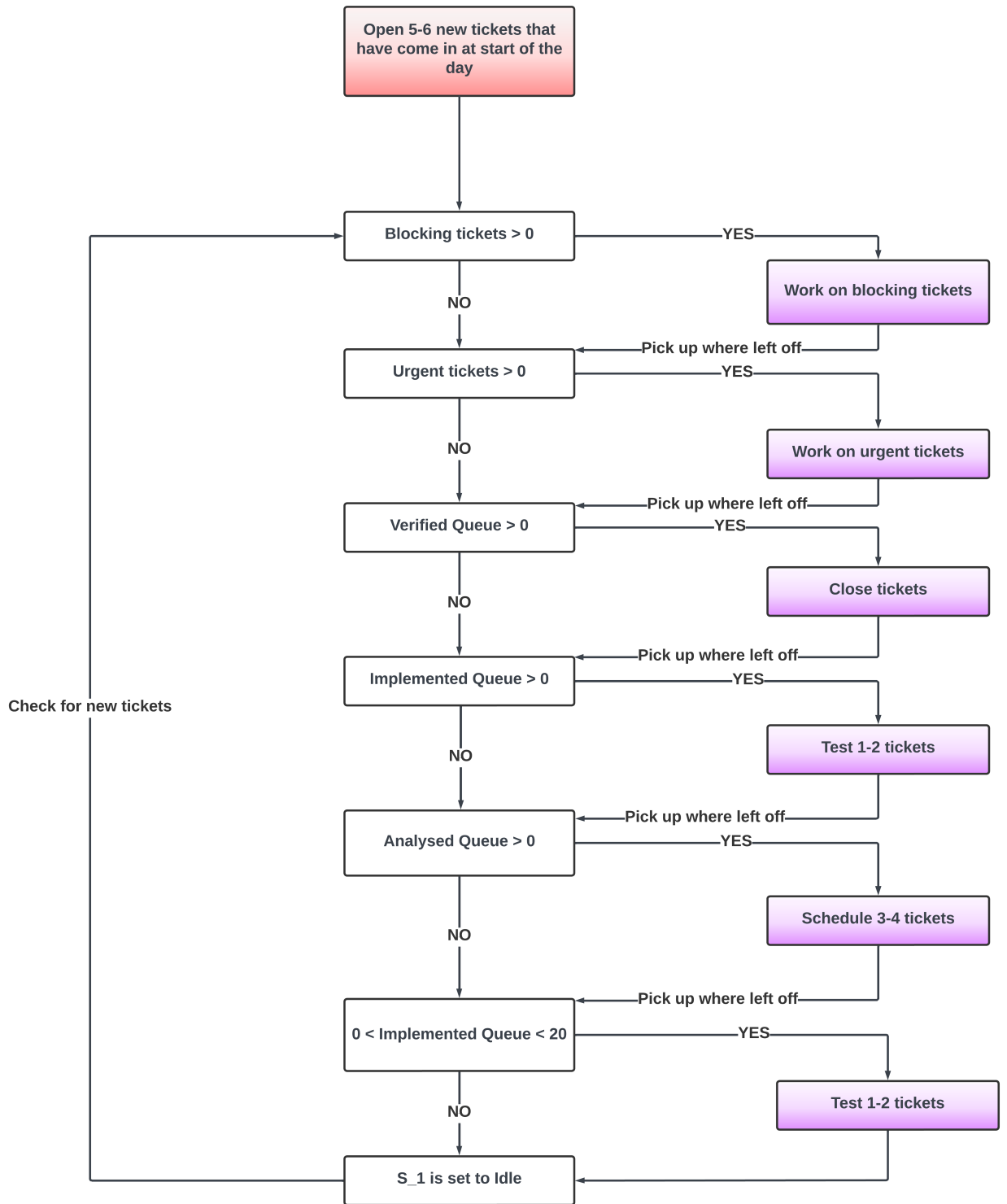


Figure 3.1: S<sub>1</sub> Workflow

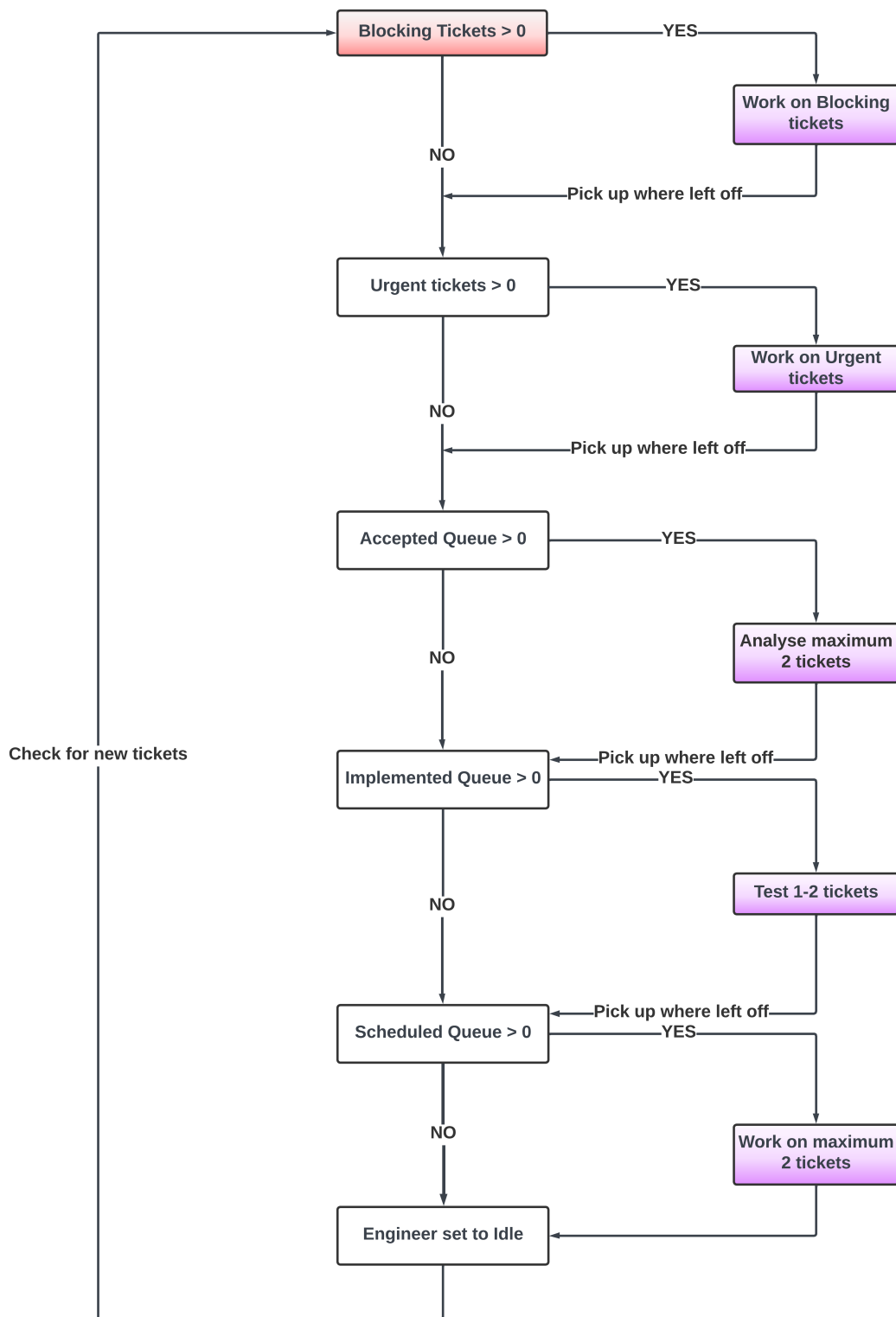


Figure 3.2: Engineer Workflow

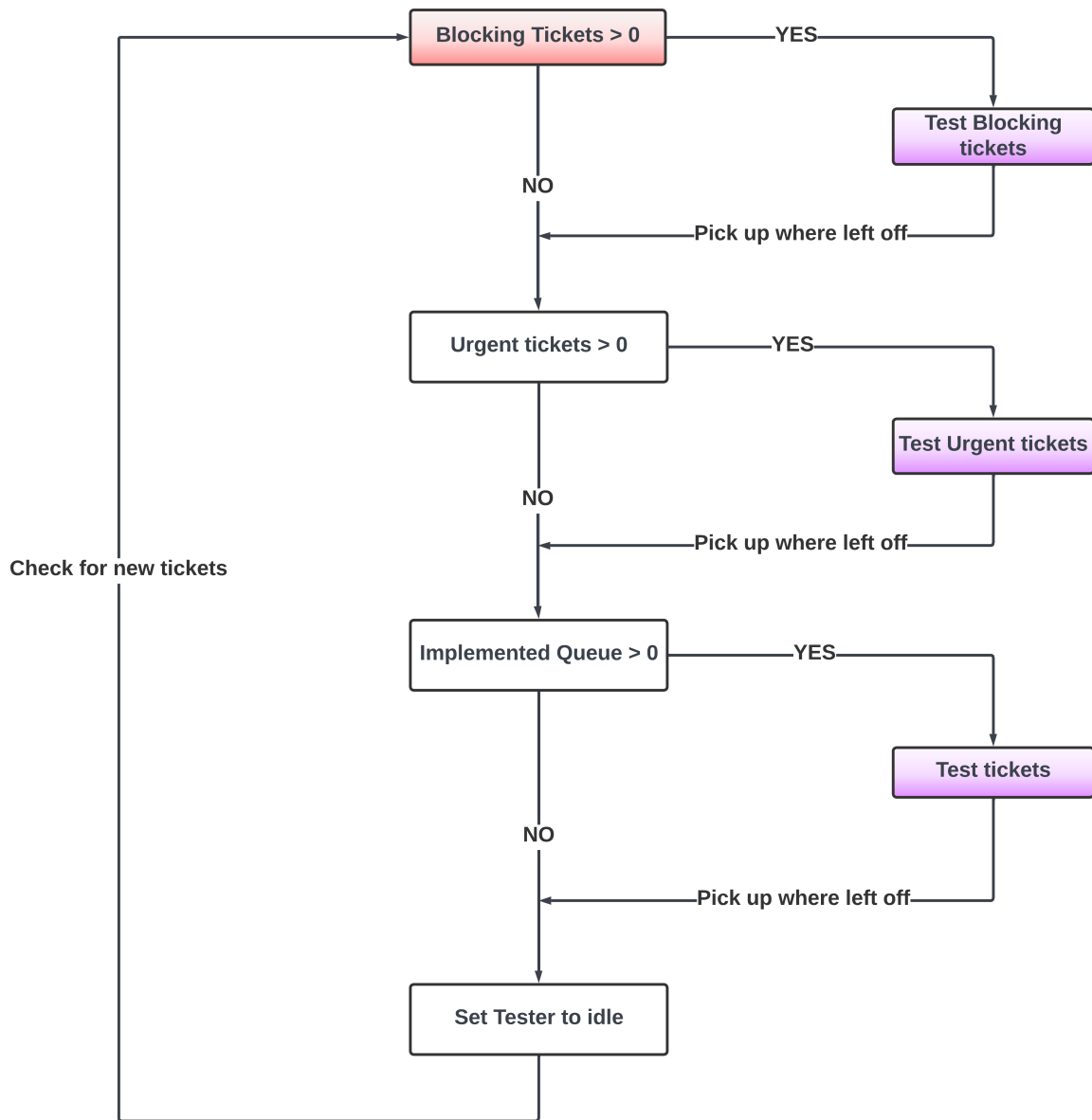


Figure 3.3: Tester Workflow

## 3.2 Events

Events are the occurrences that take place in a queueing system. In queueing theory, analysing these events and their effects on the model is an essential part. By modelling these events, several predictions are made, especially the prediction of performance measures such as waiting times, queue lengths, and service times. These predictions are significant in improving the operation of the simulated ticketing system. The events that are dealt with in the ticketing system are - Arrival events, Departure events and End-Of-Day events.

### 3.2.1 Arrival & Departure events for Queues

In this section, the handling of arrival and departure events from queues is discussed as pseudo-codes and algorithms. These pseudo-codes are discussed in depth in the section 3.3.

---

**Algorithm 1** Arrival Event on Open ( $S_1$ )

---

```
1: procedure ARRIVAL
2:   Create ticket with random attributes (component & priority)
3:   Add ticket to Queue ▷ Queue sorts the tickets on the basis of priorities
4:   Remove ticket from Open queue
5:   if  $S_1$  is idle then
6:     Schedule departure at  $t +$  service time ( $b$ )
7:     Update ticket attributes ▷ Attributes (such as waiting time and arrival time) , assign
       new state, assign time at which server stopped being idle etc.
8:     Assign  $S_1$  as working on ticket
9:     Schedule next arrival at  $t +$  interarrival time ( $a$ )
```

---

---

**Algorithm 2** Departure event from Open

---

```
1: procedure DEPARTURE FROM OPEN
2:    $i \leftarrow 0$ 
3:   while  $i < N$  do
4:     if  $E[i]$  component is same as the ticket component then
5:       Schedule Arrival at  $E[i]$  Queue at  $t$ 
6:       goto here
        $i = i + 1$ 
7:   here:
8:   Check  $S_1$ 's decision tree
9:   if there is no ticket to work on then
10:    Set  $S_1$  to idle
11:  else
12:    Change ticket attributes (which server was working on)
13:    Change server attributes
14:    Schedule departure at time  $t + b$ 
15:    Remove ticket from the respective queue
```

---

---

**Algorithm 3** Arrival Event on Accepted ( $E[i]$ )

---

```
1: procedure ARRIVAL
2:   Add ticket to Queue ▷ Queue sorts the tickets on the basis of priorities
3:   if ticket priority is blocking then
4:     Cancel departure event of ticket they were working on
5:     Change ticket attributes
6:     Change server attributes
7:     Schedule departure at time  $t + b$ 
8:     Remove ticket from queue
9:   else if  $E[i]$  is idle then
10:    Schedule departure at  $t + b$ 
11:    Remove ticket from queue
12:    Update ticket attributes
13:    Assign  $E[i]$  as working on ticket
```

---

---

**Algorithm 4** Departure event from Accepted ( $E[i]$ )

---

```
1: procedure DEPARTURE FROM ACCEPTED
2:   if ticket is a priority ticket (Blocking) then
3:     Schedule arrival at the implemented queue at time  $t$ 
4:   else
5:     With probability  $p$  schedule arrival at time  $t$  in analysed bag
6:     With probability  $1 - p$  schedule arrival at time  $t$  in open queue
7:   Check  $E[i]$  decision tree to determine next queue/ticket
8:   if there is no ticket to work on then
9:     Set server attribute to idle ▷ Server here is  $E[i]$ 
10:  else
11:    Change ticket attributes
12:    Change  $E[i]$  attributes
13:    Schedule departure at time  $t + b$ 
14:    Remove ticket from queue
```

---

---

**Algorithm 5** Arrival event on Analysed

---

```
1: procedure ARRIVAL ON ANALYSED
2:   Add ticket to Bag
3:   if  $S_1$  is idle then
4:     Change ticket attributes
5:     Change  $S_1$  attributes
6:     Schedule departure at  $t + b$ 
7:     Remove ticket from queue
```

---

---

**Algorithm 6** Departure event from Analysed

---

```
1: procedure DEPARTURE FROM ANALYSED
2:   Schedule arrival at  $E[i]$  queue at time  $t$ 
3:   Check  $S_1$ 's decision tree to see which queue they'll work on
4:   if no ticket to work on then
5:     Set server  $S_1$  attribute to idle
6:   else
7:     Change ticket attributes
8:     Change server attributes
9:     Schedule departure at time  $t + b$ 
10:    Remove ticket from queue
```

---

---

**Algorithm 7** Arrival Event on Scheduled ( $E[i]$ )

---

```
1: procedure ARRIVAL ON SCHEDULED
2:   Add ticket to Queue ▷ Queue sorts the tickets on the basis of priorities
3:   if  $E[i]$  is idle then
4:     Schedule departure at  $t + b$ 
5:     Remove ticket from queue
6:     Update ticket attributes ▷ Attributes (such as arrival time, work time), assign new state, assign time at which server stopped being idle etc.
7:     Assign  $E[i]$  as working on ticket
```

---

---

**Algorithm 8** Departure event from Scheduled ( $E[i]$ )

---

```
1: procedure DEPARTURE FROM SCHEDULED
2:   Schedule arrival in the implemented queue at time  $t$ 
3:   Check  $E[i]$ 's decision tree
4:   if no ticket to work on then
5:     Assign  $E[i]$  as idle
6:   else
7:     Change ticket attributes
8:     Change server attributes
9:     Schedule departure at time  $t + b$ 
10:    Remove ticket from queue
```

---

---

**Algorithm 9** Arrival event on Implemented

---

```
1: procedure ARRIVAL ON IMPLEMENTED
2:   Add ticket to Queue
3:    $i \leftarrow 0$ 
4:   while  $i < N$  do ▷  $N$  is the number of testers
5:     if  $T[i]$  is idle AND  $T[i] \neq E[i]$  then
6:       Change ticket attributes
7:       Change server attributes
8:       Schedule departure at time  $t + b$ 
9:       Remove ticket from queue
10:    break
11:     $i = i + 1$ 
11:   if ticket priority is Blocking then
12:      $k \leftarrow 0$ 
13:     while  $k < N$  do ▷  $N$  is the number of testers
14:       if server  $T[i]$  not working on a Blocking ticket &  $T[i] \neq E[i]$  then
15:         Cancel departure event of ticket they were working on
16:         Change ticket attributes
17:         Change server attributes
18:         Schedule departure at time  $t + b$ 
19:         Remove ticket from queue
20:       break
21:        $k = k + 1$ 
```

---

---

**Algorithm 10** Departure event from Implemented

---

```
1: procedure DEPARTURE FROM IMPLEMENTED
2:   if (random) test with probability  $q$  is negative then
3:     Schedule arrival at time  $t$  at  $E[i]$  scheduled queue
4:   else
5:     Schedule arrival at time  $t$  in verified queue
6:   Check tester's decision tree
7:   if no tickets to work on then
8:     Assign  $T[i]$  as idle
9:   else
10:    Schedule departure at time  $t + b$ 
11:    Remove ticket from queue
12:    Change ticket attributes
13:    Change server attributes
```

---

---

**Algorithm 11** Arrival event on Verified

---

```
1: procedure ARRIVAL ON VERIFIED
2:   Add ticket to Queue
3:   if  $S_1$  is idle then
4:     Change ticket attributes
5:     Change server attributes
6:     Schedule departure from the system at time  $t + b$ 
7:     Remove ticket from queue
8:   else
9:     if ticket has priority (blocking) then
10:      if  $S_1$  is not working on priority (blocking) then
11:        Cancel departure of ticket (they are working on)
12:        Change ticket attributes
13:        Change server attributes
14:        Schedule departure from the system at time  $t + b$ 
15:        Remove ticket from queue
```

---

---

**Algorithm 12** Departure event from Verified

---

```
1: procedure DEPARTURE FROM IMPLEMENTED
2:   Update performance measures
3:   Check  $S_1$ 's decision tree
4:   if no ticket to work on then
5:     Assign  $S_1$  as idle
6:   else
7:     Change ticket attributes
8:     Change server attributes
9:     Schedule departure at time  $t + b$ 
10:    Remove ticket from queue
```

---

---

**Algorithm 13** End-Of-Day Event

---

```
1: procedure END-OF-DAY
2:   Performance measures for the day are updated
3:   All counters (for every server - Engineers and  $S_1$ ) are set to 0
4:   if  $S_1$  is idle and is not working on any ticket then
5:     Remove a ticket from the Open queue
6:     Change ticket attributes
7:     Change server attributes
8:     Schedule departure at time  $t + b$ 
9:   else
10:    if the ticket  $S_1$  is working on is not priority then
11:      Cancel departure event of the current ticket  $S_1$  is working on
12:      Remove a ticket from the Open queue
13:      Change ticket attributes
14:      Change server attributes
15:      Schedule departure at time  $t + b$ 
16:      Remove ticket from queue
17:   Next End-Of-Day event is scheduled at  $t + 8 * 3600$ 
```

---

### 3.3 Simulation description

As seen in figure 2.4, the ticket arrives at the system at the rate of  $\lambda = \frac{8492}{365 \cdot 5 \cdot 3600 \cdot 8} \simeq 5$  tickets per day. Moreover, as mentioned in the section 2.2.1 this rate is used for simulating the exponential distribution of the arrival of tickets. The tickets arriving are then opened by  $S_1$  with a mean service time of 2 minutes per ticket (as mentioned in section 2.3) and this is handled in the Arrival event of the open state (in algorithm 1). Once the ticket is opened, it departs to the appropriate engineer (in algorithm 2). The engineer is chosen by matching their component with the ticket's component, i.e., the engineer should have skills in the component of the ticket that is being assigned. Once the ticket departs from the Open queue, it is added to the queue of the appropriate engineer. The tickets are sorted based on priority to ensure that the critical tickets (i.e., the ones with the highest priority) are serviced first. Upon arrival (in algorithm 3) into the engineer's (priority) queue (in the Accepted state), the ticket waits to be analysed by the engineer. If it is a priority ticket (blocking or urgent), the current task is cancelled if the current ticket being served is not a priority ticket, else the engineer will first finish working on the current priority ticket before picking up the next one. Once the engineer is done analysing the ticket, a departure event is scheduled for this ticket from the engineer's queue at time  $t + b$ , where  $t$  is the current time and  $b$  is the service time. When the departure event is being handled, the ticket is dequeued from the engineer's queue, the ticket attributes are updated and the engineer is set as working.

Now, when the ticket is departing (in algorithm 4) from the engineer's queue (Accepted state), the arrival of the ticket is scheduled at the Analysed queue. This only happens if the ticket has sufficient information. If the ticket has insufficient information (from the customer), the ticket is sent back to open with probability  $1 - p$ , else the ticket moves onto the next state (Analysed) with probability  $p$ . Once this arrival is scheduled, the engineer checks their workflow to determine their next task (as discussed in section 3.1.3). If there are no tickets to be worked on by the engineer, they are set to idle. When handling the arrival event in the Analysed queue (in algorithm 5), the ticket is first added to the queue (which is a dequeue) where the ticket waits to be scheduled by  $S_1$ . It is checked whether the server  $S_1$  is idle or not, if he is,  $S_1$  schedules the ticket back to the engineer who analysed the ticket and a departure event from the Analysed queue is set up and the ticket is dequeued from the queue. The server  $S_1$  is set as working on the ticket and the ticket attributes are updated accordingly. When the ticket is leaving the Analysed queue, its arrival is first scheduled in the engineer's queue, as seen in algorithm 6. Once the arrival event is scheduled,



the server  $S_1$  checks his workflow (discussed in section 3.1.3) to foretell his next task. If he has no ticket to work on, he will be set to idle.

After leaving the Analysed queue, the ticket enters the engineer's queue again. Upon the arrival of the engineer's queue (in the Scheduled state, as seen in algorithm 7), the ticket is added back into the engineer's queue where it waits to be implemented by the engineer. It is now checked if the engineer is idle and if they are indeed idle, the engineer starts working on (implementing) the ticket, setting the engineer as working. Furthermore, a departure event is scheduled from the engineer's queue and the ticket attributes are updated accordingly. Upon handling the departure event from the engineer's queue (in the Scheduled state, as seen in algorithm 8), an arrival event is scheduled at the Implemented queue (indicating that the ticket is moving onto the next state). The engineer then checks their workflow to foretell their next task (as discussed in section 3.1.3). If the engineer has no tickets to work on, the engineer is set to idle. Once the ticket leaves the engineers' queue (Scheduled state), it then arrives at the Implemented queue where it waits to be tested and verified by a server. Upon arrival at the Implemented (priority) queue (as seen in algorithm 9), the ticket is added to the queue. Now, the system searches for a free server amongst the engineers (excluding the engineer who analysed and implemented the ticket) and the main tester. Once the system has found the free server, the server starts the testing and verification process of the ticket and is set to working. Furthermore, a departure event from the Implemented queue is scheduled and the ticket is removed from the queue. If no servers are free and the ticket is a priority ticket, a server is searched for who is not working on a priority ticket and is not the engineer who analysed and implemented the ticket. Once such a server is found, the departure of the ticket they are currently working on is cancelled, and the server starts working on the priority ticket and is hence set as working. Further, a departure event is scheduled for this ticket from the Implemented queue and the ticket attributes are updated accordingly.

While leaving the Implemented queue (as seen in algorithm 10), it is first checked whether the ticket tested positive (i.e., it can move onto the next state) or negative (i.e., it will have to be sent back to the engineer who implemented it). If the ticket tests negative with probability  $q$ , then an arrival is scheduled at the engineer's queue (i.e., the engineer who analysed and implemented the ticket). Else, when the ticket tests positive, an arrival is scheduled at the verified queue (indicating that it has been successfully tested and verified). Once these arrivals are scheduled, the server (who tested the ticket) checks their workflow; if the server is an engineer, the engineer checks their workflow (as discussed in section 3.1.3) and if the server is the tester, he will check his workflow (as discussed in section 3.1.3). If there are no tickets for the server to work on, the server is set to idle. Upon departure from the Implemented queue, the ticket arrives at the Verified queue. When handling the arrival event at the Verified (priority) queue (as demonstrated in algorithm 11), the ticket is first added to the queue, where the ticket waits to be closed by  $S_1$ . If the server  $S_1$  is idle, he works on the ticket right away and therefore is set as working. Furthermore, a departure event from the Verified queue is scheduled for the ticket, the ticket is removed from the queue and the ticket attributes are updated. Else, if the ticket is a priority ticket and  $S_1$  is not working on a priority ticket, the departure event of the current event is cancelled for the current ticket and  $S_1$  starts working on the priority ticket, hence setting him as working. Now, a departure event is scheduled for the ticket from the Verified queue, the ticket is removed from the queue and the ticket attributes are updated accordingly. While handling the departure event from the Verified queue (in algorithm 12), the performance measures (such as sojourn time and waiting times) are updated and the ticket leaves the system. Once the ticket has left the system,  $S_1$  checks his workflow (as discussed in section 3.1.3) to determine his next task. If he has no tickets to work on,  $S_1$  is set to idle.

After every 8 hours, an End-Of-Day event is scheduled (as seen in algorithm 13). At the end of the day (i.e., after 8 working hours), a new day starts and all the counters for the servers are reset. The day starts with  $S_1$  opening the tickets that entered the system at the start of the day. If the ticket  $S_1$  was working on from the previous day is a priority ticket, he will continue working on it, else the departure event of that ticket will be cancelled and  $S_1$  will first work on the Open

queue. When  $S_1$  is working on the tickets, a departure event is scheduled (from the Open queue) and the ticket is removed from the Open queue. Furthermore, the ticket attributes and server attributes are updated. And lastly, the next End-Of-Day event is scheduled after 8 hours.



# Chapter 4

## Results

In this chapter, the results of the simulation study on the ticketing system for TIOBE are presented. The simulation was modelled to evaluate the system's performance under different scenarios and potential bottlenecks.

The simulation helped in identifying potential bottlenecks in the system. For instance, it was found that the ticket processing time was a critical factor that affected the system's performance. By reducing the ticket processing time, one can significantly improve the system's response time and reduce the average wait time for customers.

In this chapter, the analysis of results from simulating the current system is done assuming that the tickets always have sufficient information and always test positive. This means that the tickets go through the system **without** interruptions due to external factors and tests performed by the testers (mathematically, the probability of tickets being sent back to previous states due to external factors or negative testing is zero). The simulation is logged for 5 years, over 10 times to accumulate dependable results.

### 4.1 Queue analysis

#### 4.1.1 Queue Lengths

The analysis of the ticketing system revealed significant differences in the queue lengths for tickets in different states. Specifically, it is established that tickets in the Accepted state had consistently longer queue lengths compared to tickets in the other states. To collect data on the queue lengths, the simulation logs over a period of 5 years (and 10 times). A Python script is used to extract the queue lengths for each state and plotted the data using a histogram chart. The analysis revealed that the average queue length for different states can be seen in Table 4.1.

State	Mean Queue length	Standard deviation of queue length	Rate of increase of Queue (per day)	95% Confidence Interval
Open	0.343	2.066	0.0	[0.216, 0.471]
Accepted	1950.601	1108.863	2.119	[1931.393, 1969.81]
Analysed	0.0148	0.174	0.0	[0.0032, 0.0264]
Scheduled	216.466	125.543	0.239	[203.044, 229.887]
Implemented	0.0152	0.130	0.0	[0.0128, 0.0175]
Verified	0.0521	0.45	0.0	[0.0323, 0.0719]

Table 4.1: Queue lengths in states

It is believed that the long queue lengths for "Accepted" tickets may be due to a backlog of unresolved issues, which could be serviced by improving the prioritization and assignment of tickets to the engineers. On the other hand, it is necessary to increase the number of engineers in order to handle the higher volume of "Accepted" tickets. Moreover, it is observed that the verified and analysed queues tend to have the highest probability to have a queue length of 0 (with a probability close to 1, as seen in figures 4.6 and 4.3), followed by the Implemented queue ( $\mathbb{P}(Q = 0) \simeq 0.99$ , as seen in figure 4.5), Open queue ( $\mathbb{P}(Q = 0) = 0.95$ , as seen in figure 4.1), Scheduled queue ( $\mathbb{P}(Q = 0) \simeq 0.0015$ , as seen in figure 4.4) and Accepted queue ( $\mathbb{P}(Q = 0) \simeq 0.000245$ , as seen in figure 4.2). Additionally, the probability of queue lengths for all states decreases to 0 as the queue lengths increase, however, for Accepted and Scheduled states that are not true. It is also indicated that the rate of increased tickets in the Accepted state and Scheduled state is positive, suggesting that as the number of days (the system is being run) increases, the queue length increases. Furthermore, the rate of increase for the Accepted state is much higher compared to the rate of increase for the Scheduled state, signifying that tickets pile up much faster at the Accepted state as compared to the Scheduled state. In addition to that, it is observed that the Accepted and Scheduled states have the widest confidence intervals. This indicates that the build-up in these two states is higher and more unstable.

In conclusion, this analysis suggests that the ticketing system could benefit from improvements to reduce the queue lengths for "Accepted" tickets. By dealing with this issue, the system could improve its overall efficiency and provide a better user experience for customers.

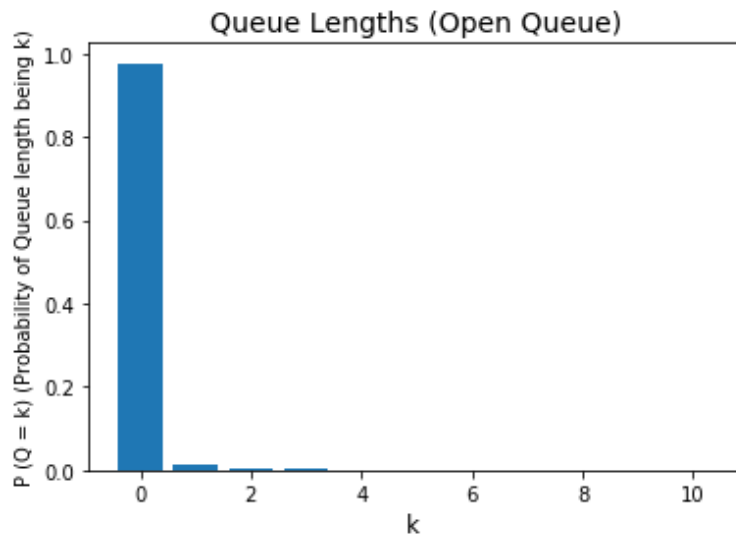


Figure 4.1: Queue Length at Open

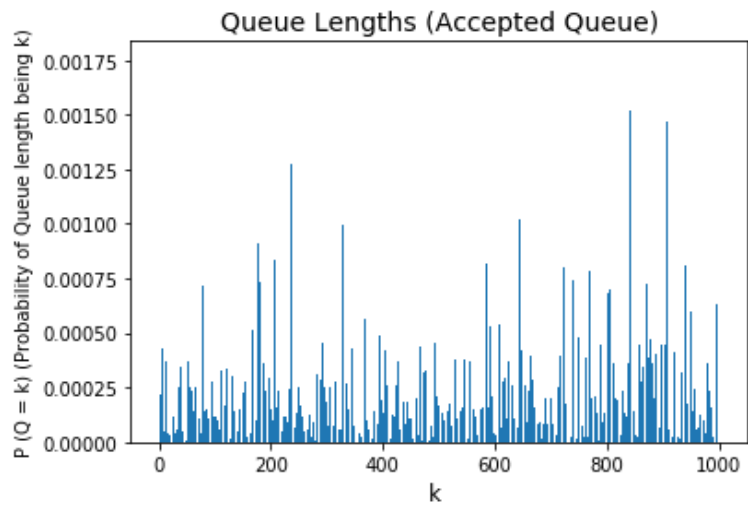


Figure 4.2: Queue Length at Accepted

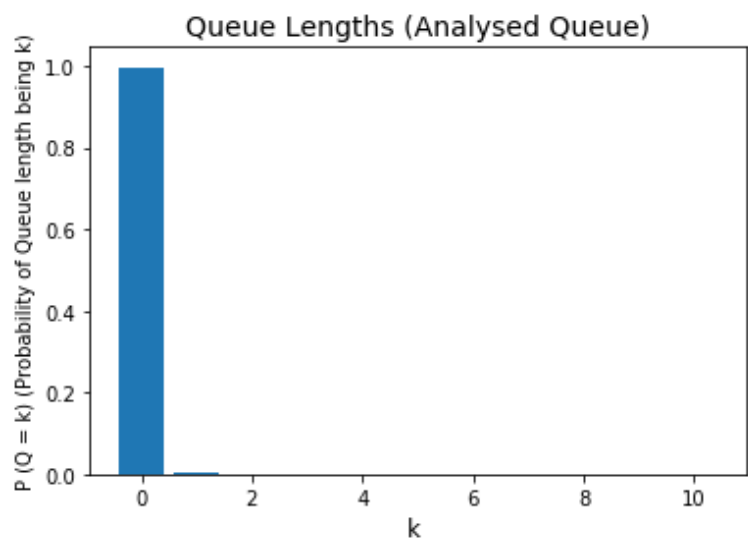


Figure 4.3: Queue Length at Analysed

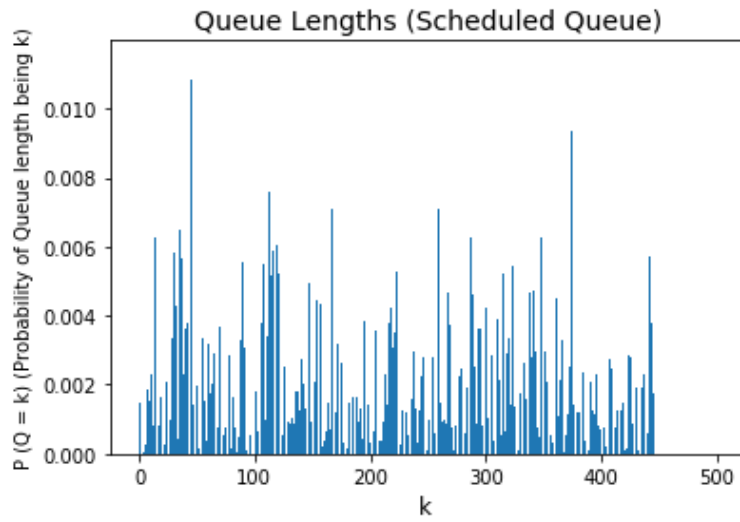


Figure 4.4: Queue Length at Scheduled

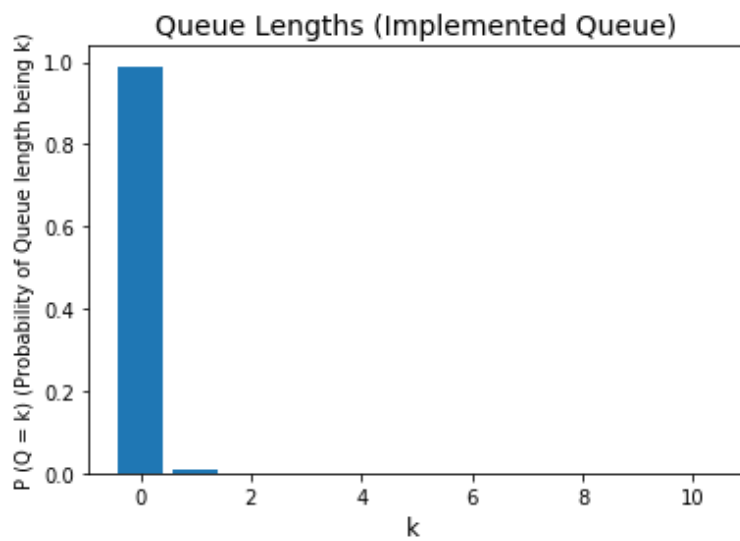


Figure 4.5: Queue Length at Implemented

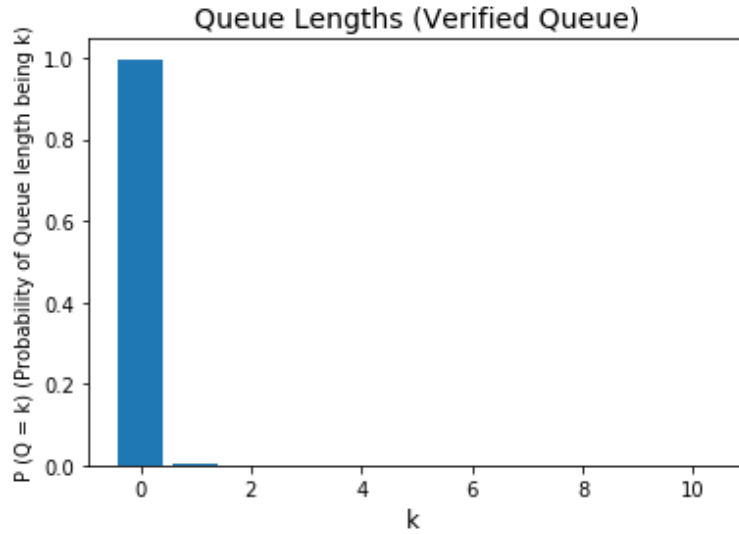


Figure 4.6: Queue Length at Verified

#### 4.1.2 Queue Lengths at End-Of-Day

The analysis of the number of tickets in the queues at the end of the day revealed that at the end of the day, the queue lengths in different states varied significantly. It is observed that the Accepted state has the highest queue length, followed by the Scheduled state.

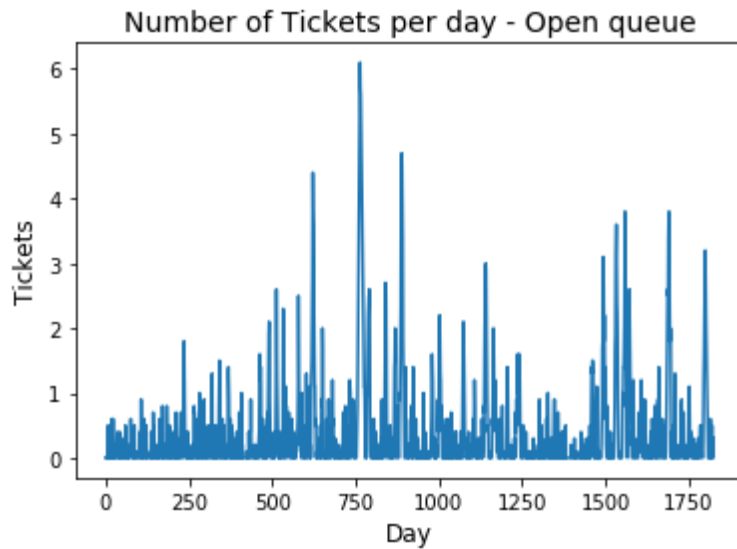


Figure 4.7: Queue Length at Open (End-Of-Day)

It is observed that the queue lengths at the end of the day, the queue length in the Accepted state (Figure 4.8) and the Scheduled state (Figure 4.10) keep on increasing and never stabilise. This leads one to believe that the system is unstable due to unresolved tickets piling up in these two states. And so, these states are the potential bottlenecks of the ticketing system. On the other hand, the queue lengths in the Open (Figure 4.7), Analysed (Figure 4.9), Verified (Figure 4.12), and Implemented (Figure 4.11) states are stable and do not exceed a certain value. It is believed that this is due to the fact that the service times in these states are much lower than



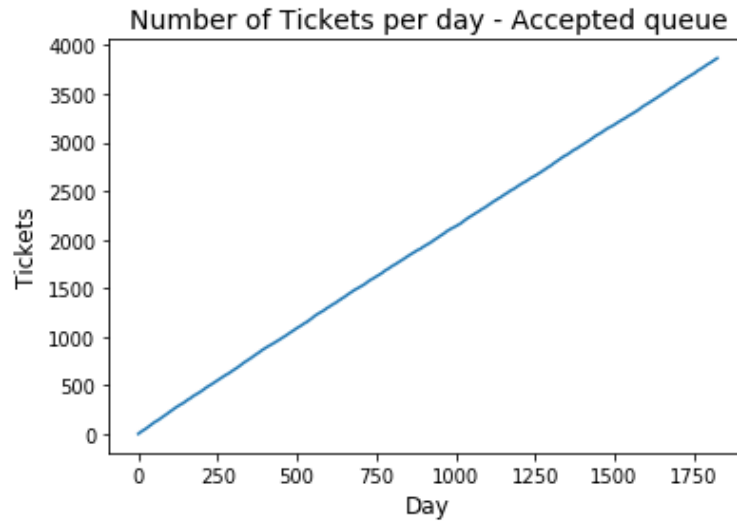


Figure 4.8: Queue Length at Accepted (End-Of-Day)

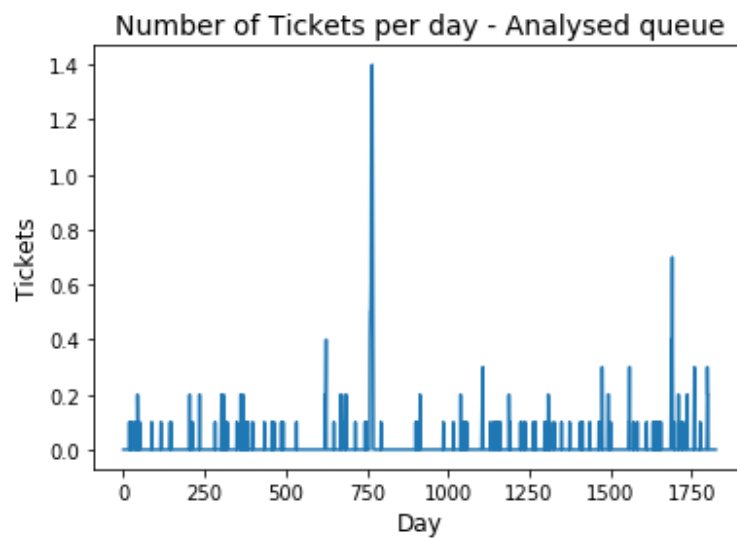


Figure 4.9: Queue Length at Analysed (End-Of-Day)

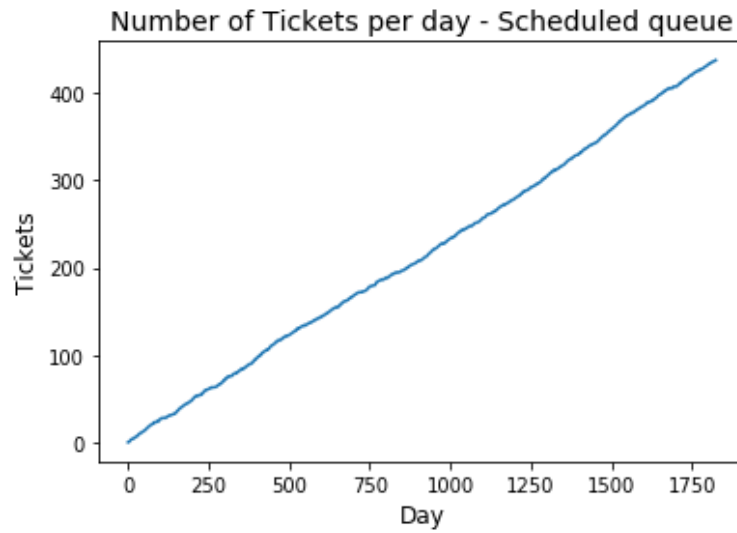


Figure 4.10: Queue Length at Scheduled (End-Of-Day)

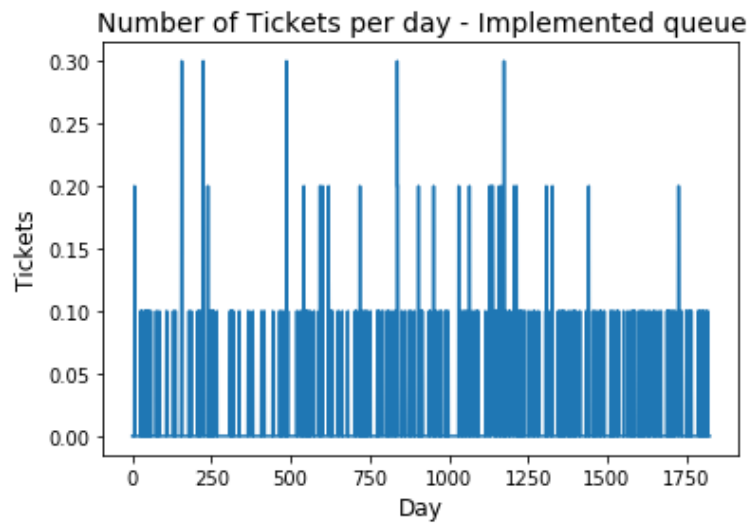


Figure 4.11: Queue Length at Implemented (End-Of-Day)

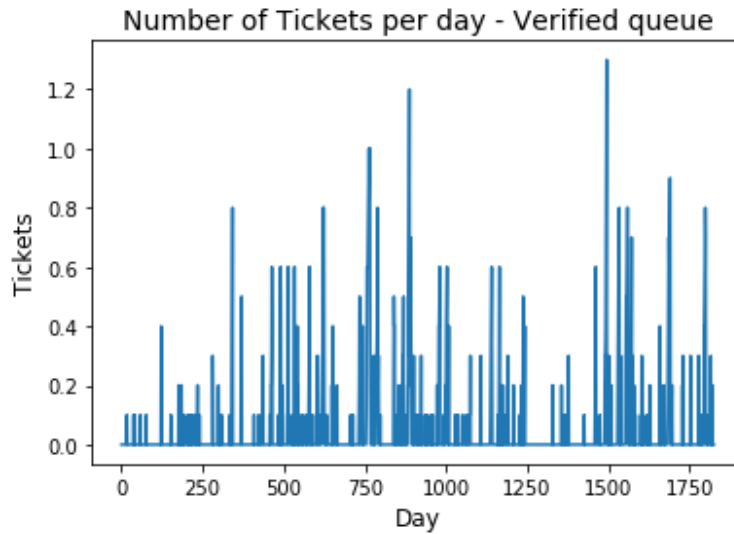


Figure 4.12: Queue Length at Verified (End-Of-Day)

the service times in the Accepted and Scheduled states. It can also be deduced that due to the bottlenecks at the Accepted and Scheduled states, not all tickets reach the Analysed, Implemented and Verified states implying that the arrival rate in these states is quite low. Even though the mean service time in the Implemented queue is 10 hours, the queue is stable due to the number of servers actively working on the queue (7 Engineers and 1 tester).

Since the Accepted and Scheduled states are a part of the engineer's queue, the End-Of-Day queue lengths for the engineers' queues are depicted below.

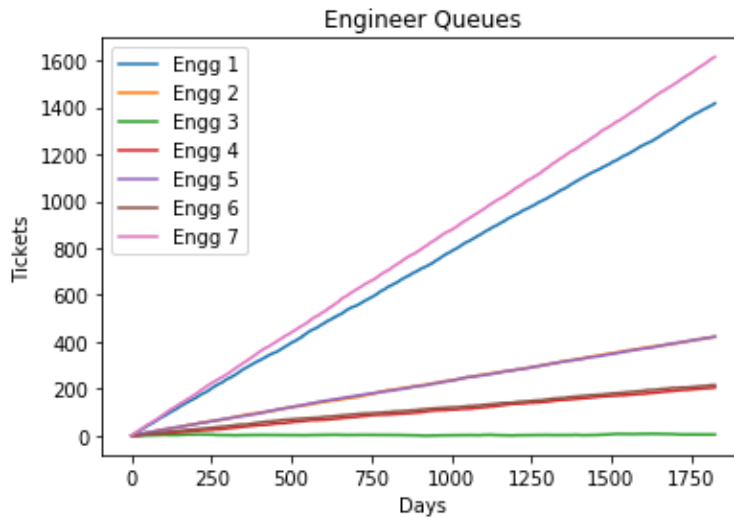


Figure 4.13: Queue Length for Engineers (End-Of-Day)

It is noticed that queues for engineers 1 and 7 are the longest and for engineer 3 it is the shortest. Moreover, the rate of increase of queues for engineers 1 and 7 is much higher as compared to other engineers (as seen in Table 4.2). This imbalance is a consequence of having engineers who are not skilled in all components. Since engineers 1 and 7 are skilled in most components and engineer 3 is skilled in the least (as seen in figure 2.2), this creates an extra workload for engineers 1 and 7

while engineer 3 has the least amount of work. To avoid such an imbalance, TIOBE should train their engineers (such as engineers 2,3,5) in more components or hire engineers who are skilled in the components engineer 1 and 7 work with.

Engineer	Rate of Increase of their Queues (tickets per day)
Engineer 1	0.777
Engineer 2	0.231
Engineer 3	0.0029
Engineer 4	0.113
Engineer 5	0.232
Engineer 6	0.119
Engineer 7	0.885

Table 4.2: Rate of increase of engineers' queues

### 4.1.3 Waiting Times in Queues

The investigation of the ticketing system revealed that the waiting times for tickets in different states varied significantly. It is found that tickets in the Accepted state had the highest waiting times than tickets in the other states.

The analysis revealed the average waiting time for different states, which are logged in Table 4.3. This suggests that the system is taking quite a lot of time for the accepted tickets to be serviced due to its servers being extremely busy, corresponding to results for the queue lengths in the Accepted state (as discussed in section 4.1.1). Furthermore, the confidence interval of the Accepted and Scheduled states is much higher, as compared to other states. This indicates the previously established instability of queues in these states.

State	Mean Waiting time (in seconds)	Mean Waiting time (in hours)	95% Confidence Interval (in hours)
Open	8269.401	2.29	[2.083, 2.511]
Accepted	1104002.33	306.67 (approx. 38 working days)	[270.536, 342.799]
Analysed	2062.768	0.573 (or 34.4 minutes)	[0.368, 0.777]
Scheduled	559155.36	155.32 (approx 19.4 working days)	[122.915, 187.726]
Implemented	34795.165	9.67 (approx. 1.2 working days)	[9.465, 9.866]
Verified	2185.005	0.607 (approx. 36.4 minutes)	[0.439, 0.775]

Table 4.3: Waiting times in states

It is observed that the waiting time in the Accepted state (figure 4.15) is very likely to be between 0 to 1100 hours (approximately), followed by the waiting time in the Scheduled state (figure 4.17) which is more likely to be between 0 to 200 hours, waiting time in the Implemented state (figure 4.18) which is most likely to be between 0 to 50 hours, waiting time in Open state (figure 4.14) that is more probable to be between 0 to 10 hours, waiting time in Analysed state (figure 4.16) that is probable to be between 0 to 1.5 hours and lastly, waiting time in Verified state (figure 4.19) which is most likely to be between 0 to 1.3 hours.

Once again, such a vast difference in waiting times occurs due to insufficient servers and high

service times. To address this issue, it is recommended to explore options to balance the workload better, such as cross-training or hiring more engineers.

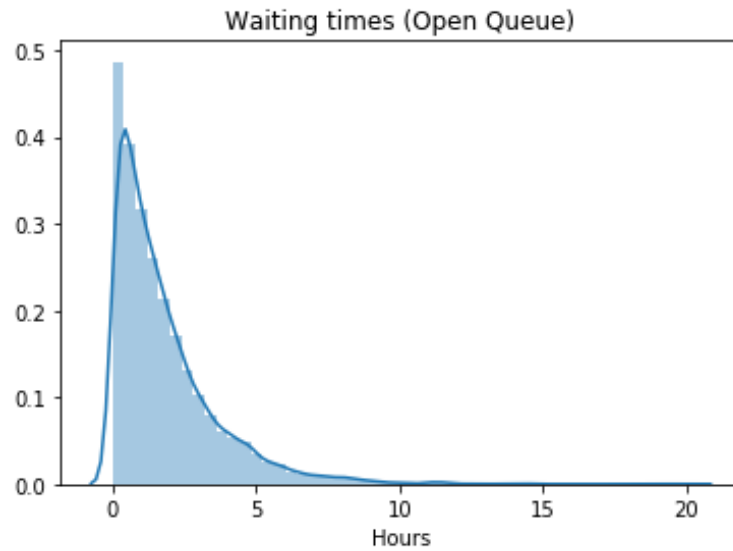


Figure 4.14: Waiting times over every point in time

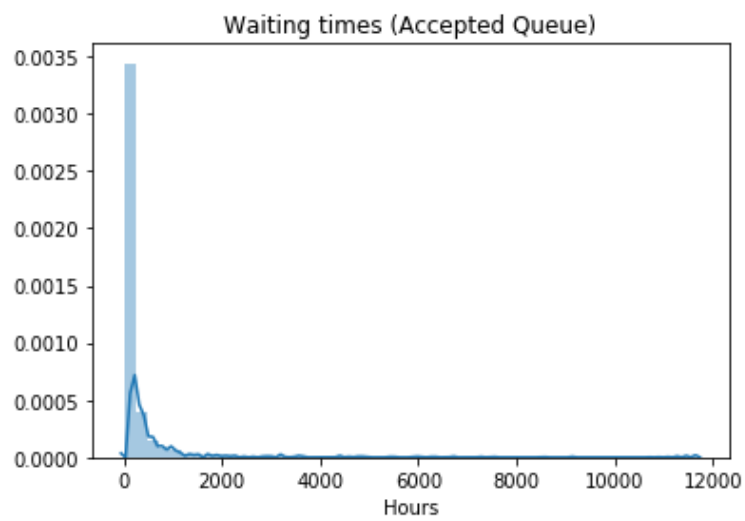


Figure 4.15: Waiting times over every point in time

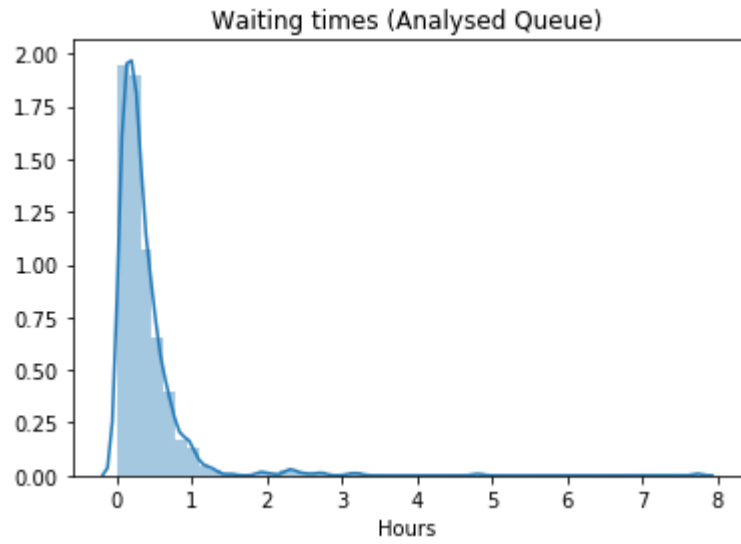


Figure 4.16: Waiting times over every point in time

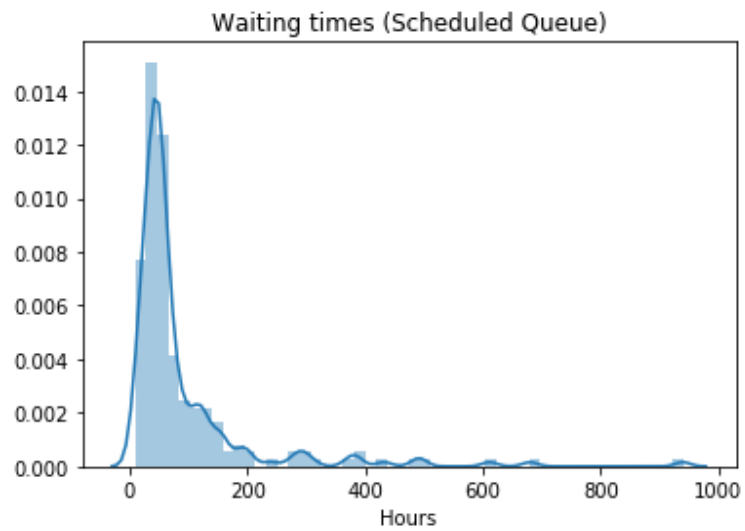


Figure 4.17: Waiting times over every point in time

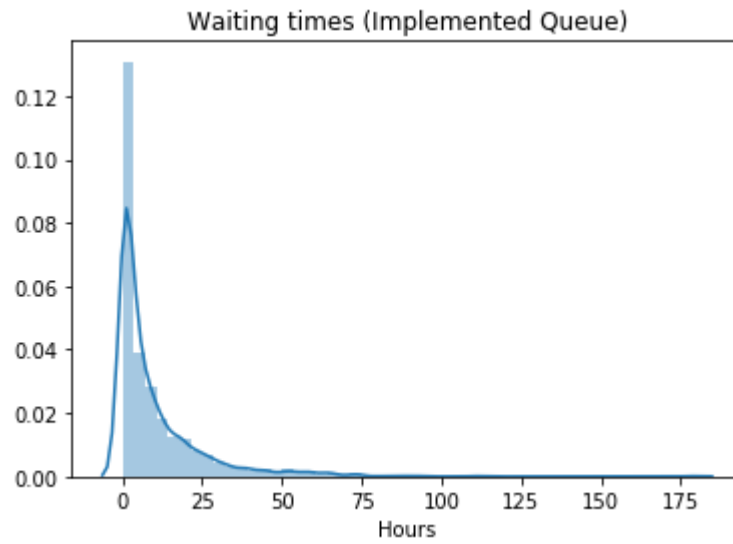


Figure 4.18: Waiting times over every point in time

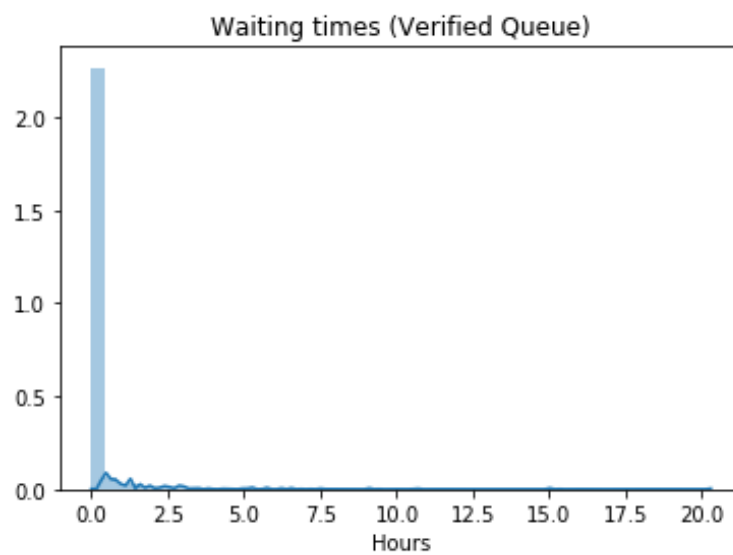


Figure 4.19: Waiting times over every point in time

## 4.2 Server analysis

### 4.2.1 Work times of servers

The investigation of the ticketing system revealed that the working times (the total time servers spend on working on tickets) and idle times (the total time servers spend on being idle/having no tickets to work on) of different servers varied considerably. It is found that some servers had consistently longer working times than others, which affects the performance and efficiency of the system.

The analysis revealed that the mean working time and idle time for servers are -

Server	Work time in hours (& %age of time they work)	Idle times in hours (& %age of time they work)
$S_1$	1262.005 (8.57%)	13457.99 (91.43%)
Engineer 1	14577.791 (99.848%)	3.468 (0.044%)
Engineer 2	14554.461 (99.69%)	15.391 (0.17%)
Engineer 3	13335.787 (91.341%)	1171.152 (8.201%)
Engineer 4	14544.331 (99.62%)	17.004 (0.188%)
Engineer 5	14573.756 (99.82%)	16.811 (0.115%)
Engineer 6	14523.209 (99.47%)	30.519 (0.237%)
Engineer 7	14577.523 (99.846%)	4.942 (0.034%)
Tester	8586.976 (58.815%)	5992.021 (41.041%)

Table 4.4: Working and idle times of servers

From table 4.4, it is observed that engineers 1 and 7 are the servers that work the most, the servers that work the least are  $S_1$  and the tester, and the engineer that works the least is engineers 3.

To solve the issue of uneven workload, it is recommended to conduct a more detailed analysis of engineers 1 and 7 to identify the root cause of the longer working times. According to the scope of the information available, this is mainly a consequence of these servers having more components as compared to other servers. The detailed analysis could involve monitoring their performance metrics, reviewing server configurations, or running stress tests to simulate high levels of traffic. Consequently, it is necessary to redistribute tickets to other servers in order to balance the workload more effectively, i.e., more servers (engineers) need to be skilled in additional components.

### 4.2.2 Server tickets analysis

The analysis revealed the number of ticket servers worked on in 5 years, as represented in Table 4.5.

The difference in the number of tickets worked on by the servers is due to differences in workload distribution or ticket routing. It is the case that server  $S_1$  is handling a higher volume of tickets due to his role - opening all tickets, scheduling all tickets, implementing some tickets (when the queue length is high) and closing all tickets. This means that  $S_1$  has worked on every ticket (opening and closing). Similarly, for the tester, since his task is mainly testing and verification, the number of tickets he tests and verifies is quite high. It is also observed that the number of tickets worked on by engineers 1 and 7 is higher than the rest of the engineers. This could be a result of a pile-up of tickets in their specific queues, resulting in engineers 1 and 7 working on more tickets as compared to other engineers. Consequently, there may be issues with the ticket routing algorithm that are causing an imbalance in the workload distribution.



Server	Number of tickets
$S_1$	8482.2
Engineer 1	480.0
Engineer 2	458.9
Engineer 3	469.2
Engineer 4	473.4
Engineer 5	465.4
Engineer 6	470.4
Engineer 7	481.8
Tester	868.3

Table 4.5: Average number of tickets worked on (in 5 years)

### 4.3 Tickets analysis

#### 4.3.1 Sojourn Time

The mean sojourn time for a ticket in the system is the amount of time the ticket is expected to spend in the system before it leaves the system (after being closed). (Melamed, 1982)

The mean sojourn time in the simulated system is **244.086 hours** with the 95% confidence interval of [**197.409, 290.764**]. In addition to that, the average number of tickets closed in a day (number of tickets leaving the system in a day) is  $0.987 \simeq 1$  ticket. From figure 4.20 it is observed that a ticket typically spends up to 500 hours in the system. A high sojourn time indicates high waiting times in the queues. If the ticket passes through the queues uninterrupted and without any waiting, it should take about 60.5 hours. However, the mean sojourn time is approximately 4.55 times the uninterrupted time (of 60.5 hours). This indicates that the tickets spend most of their time waiting to be served which leads to inefficiency of the system, increased workload of servers and build-ups in queues.

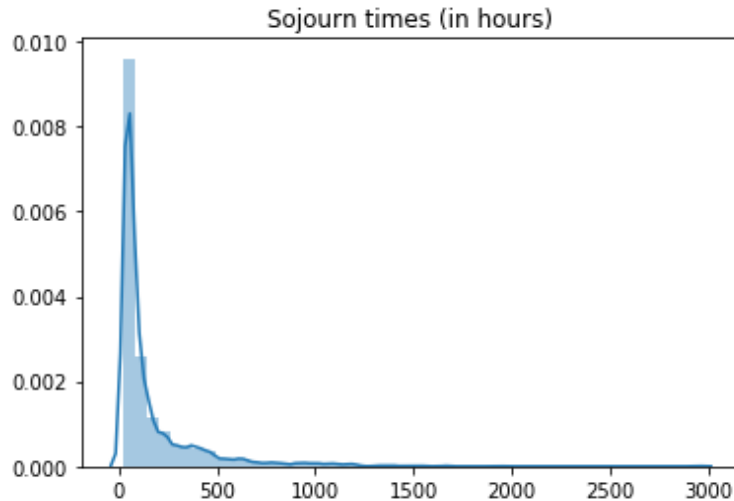


Figure 4.20: Sojourn times

#### 4.3.2 On the basis of Priorities

Through the simulation, it is also seen how much time is spent by each priority ticket through the system. It is observed that the priority tickets have the following sojourn times -

Priority	Mean Sojourn time (in hours)	Number of tickets resolved
Blocking	245.14	1630.9 $\simeq$ 1631
Urgent	125.46	65.9 $\simeq$ 66
High	375.85	12.2
Normal	398.56	10.6
Low	501.88	11.4

Table 4.6: Mean Sojourn times of tickets (grouped by priority)

It is observed that the mean sojourn time for the highest priority ticket, Blocking, is quite high. This is a consequence of the build-up of priority tickets for each server, as in the case where there is more than one blocking ticket in a queue, the blocking tickets are served on a first come first serve basis. This leads to increased waiting time for the blocking tickets, and hence a higher sojourn time as well.

However, it is observed that the sojourn times for urgent tickets are lower, as compared to blocking tickets. But, it is also discovered that the number of tickets resolved in these priorities is quite low (as compared to blocking tickets). This indicates that most of the urgent tickets build up in the system and are seldom closed. Similarly, for tickets with priority high, normal and low, not only is the mean sojourn times higher but also the number of tickets resolved is quite low, suggesting that there is an enormous build of high, normal and low priority tickets as well. It can be concluded that a ticket leaving a system has a very high probability of having a priority of blocking. This imbalance is potentially a result of the inefficient distribution of workload amongst the engineers.



## Chapter 5

# Improving the system performance

In this chapter, it is further discussed the implications of the simulation study on the ticketing system for TIOBE and its potential impact on the company's business operations.

### 5.1 Impact of insufficient information and tickets testing negative

As discussed previously, it is possible that the ticket is sent back to Open (from analysed) due to insufficient information from the customer and can be sent back to the engineer (as a result of testing negative). Now, the probability of the ticket being sent back in both cases is set to 0.1 and the simulation is logged for 5 years (the code is run 10 times and the results below are averaged over 10 runs).

#### 5.1.1 Build up between Open state and Analysed state

As observed in table 5.1, the waiting time for tickets increases by at least 2 times for all states. However, in the Accepted state, it is seen that the queue length decreases by a factor of 0.9 even though the waiting time increases by a factor of 1.5 due to engineers getting busier at every state they work on and due to tickets being sent back to the Open state, not all tickets make it past the Open state into the Accepted state. The waiting time in the states Open and Analysed increase as the server  $S_1$  gets busier implementing and closing tickets at the Implemented state and Verified state (respectively). This is also reflected in the increase in the work time of the server  $S_1$  by a factor of 8.6 (as seen in table 5.2). In addition to that, there is a significant pile up at the Open and Analysed states, where the number of tickets in the Open state increased by a factor of 1644.4 and in the Analysed state increased by a factor of 750.7. Furthermore, the waiting time in Open, Accepted and Analysed states increased by a factor of 372.3, 1.5 and 17.1 times respectively. This leads to an increase in the work time of the engineers and  $S_1$  as well, as seen in table 5.2 and the average work time of the engineers and  $S_1$  has increased even with a small probability of the ticket having insufficient information. Consequently, with a small probability of having insufficient information, the build-up increases for all the engineers and the server  $S_1$ , leading to an increased build-up in the Open and Analysed states, in addition to an increase in waiting times in Open, Accepted and Analysed states.

#### 5.1.2 Build up between Scheduled state and Verified state

Since not many tickets reach the Analysed state, not all tickets are scheduled for the engineers to implement. This results in a decrease in the number of tickets in the Scheduled state by a factor

of 0.53 even though the waiting time increases by a factor of 1.6 (as seen in table 5.1). Moreover, the number of tickets in the Implemented and Verified states increase by a factor of 1300 and 13.05 times (respectively) in addition to an increase in waiting time by a factor of 9.6 and 14.7 times (respectively). This heavy pile-up is the result of all the servers getting busier at almost every stage of their workflow. For instance, as the engineers get busier analysing the tickets (due to pile up at the accepted queue), they'll get to test and verify the tickets later which in turn increases their waiting time in the Implemented state finally, when they get to their Scheduled (state) tickets, the waiting time for the tickets in the Scheduled state has increased drastically. Similarly, for the server  $S_1$ , as he gets busier opening the tickets, the waiting time for the tickets in Analysed state and Verified state increases. In addition to that, due to the increase in the number of tickets in the Implemented state,  $S_1$  now has to actively work on this queue as well, resulting in an increased queue length in the Verified state (alongside the increase in wait time for tickets in that state). Furthermore, the work time for the tester increases by 1.6 times, which indicates that due to tickets testing negative (and insufficient information), the tester is overworked.

Consequently, due to insufficient information and tickets testing negative, the work times increase for all servers (as seen in Figure 5.2) and waiting times increase in all states as well, leading the system to become more unstable and imbalanced.

State	Mean Queue length (in nr. of tickets)	Mean Waiting time (in hours)
Open	564.035 Increase by 1644.4 times	852.54 Increase by 372.3 times
Accepted	1782.05 Decrease by 0.9 times	458.713 (approx. 57.3 working days) Increase by 1.5 times
Analysed	11.11 Increase by 750.7 times	9.78 Increase by 17.1 times
Scheduled	113.904 Decrease by 0.53 times	244.87 (approx. 30.6 working days) Increase by 1.6 times
Implemented	19.76 Increase by 1300 times	92.99 (approx. 11.6 working days) Increase by 9.6 times
Verified	0.68 Increase by 13.05 times	8.9 Increase by 14.7 times

Table 5.1: Queue lengths in states when tickets have insufficient information and test negative with a probability of 0.1

### 5.1.3 Effect on Mean Sojourn time

It is observed that the mean sojourn time for the tickets in this situation increases to **1301.25 hours**, which is an increase by a factor of 5.33 (as compared to the sojourn time in 4.3.1). This indicates that the tickets are spending much more time waiting in the queues as compared to the situation where the probability of having insufficient information and testing negative was zero. As mentioned previously (in 4.3.1), a ticket could go through the system uninterrupted and without waiting within 60.5 hours. This means that the ticket spends 21.5 times more time in the system than it ideally should, which results in extreme build-up in the system.

Furthermore, from figure 5.1 it is seen that in this situation a ticket could spend up to 4000 hours in the system, as compared to tickets in the previous situation, where the ticket could spend only up to 500 hours (as seen in figure 4.20). This is an 8-fold increase and indicates that insufficient information and tickets testing negative can take a heavy toll on the efficiency of the ticketing system.

Server	Work time in hours (& %age of time they work)
$S_1$	11936.035 (81.09%) Increase by 8.6
Engineer 1	14579.919 (99.862%) Increase by 1.0001 times or 2.6 hours
Engineer 2	14589.063 (99.925%) Increase by 1.0012 times or 35 hours
Engineer 3	13645.996 (93.466%) Increase by 1.023 times
Engineer 4	14566.135 (99.77%) Increase by 1.001 times or 21.8 hours
Engineer 5	14573.809 (99.82%) Increase by 4 minutes
Engineer 6	14589.255 (99.93%) Increase by 11.7 hours
Engineer 7	14597.22 (99.95%) Increase by 1.001 times or 19.69 hours
Tester	13633.04 (93.37%) Increase by 1.6 times

Table 5.2: Working and idle times of servers when tickets have insufficient information and test negative with a probability of 0.1

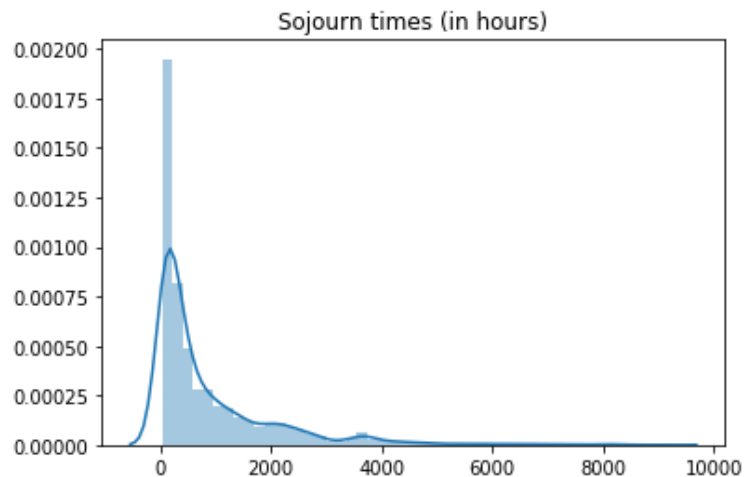


Figure 5.1: Sojourn when tickets have insufficient information and test negative with a probability of 0.1

### 5.1.4 Effect on Priority tickets

Priority	Mean Sojourn time (in hours)	Number of tickets resolved
Blocking	1316.48 Increase by 5.4 times	1104.4 Decrease by 0.67 times
Urgent	110.21 Decrease by 0.88 times	9.3 Decrease by 0.14 times
High	472.59 Increase by 1.3 times	3.7 Decrease by 0.3 times
Normal	521.69 Increase by 1.3 times	2.0 Decrease by 0.19 times
Low	778.7 Increase by 1.6 times	0.89 $\simeq$ 1 Decrease by 0.078 times

Table 5.3: Mean Sojourn times of tickets (grouped by priority) when tickets have insufficient information and test negative with a probability of 0.1

As observed in table 5.3, the number of tickets resolved or closed (for each priority) decreases and the mean sojourn time spent in the system increases. However, for Urgent priority tickets, the mean sojourn time decreases. This is a result of lesser tickets being resolved - the number of tickets with priority Urgent being resolved decreased by a factor of 0.14 (from 66 tickets to 9 tickets). This occurs due to an increase in the number of Blocking tickets (which is a higher priority as compared to urgent) leading to the reduction of Urgent priority tickets leaving the system (and hence building up in the system).

Tickets with priority Blocking have higher sojourn time in this situation as these tickets are more likely to be resolved first, resulting in their sojourn time being registered (and in the build-up of Urgent, High, Normal and Low priority tickets as they are not resolved). Furthermore, the number of Blocking tickets being resolved decreases by a factor of 0.67 due to these tickets being sent back (due to insufficient information - back to Open state; or due to testing negative - back to Scheduled state/Engineer's queue). Therefore, this circumstance not only causes a build of tickets in states but also a build of crucial Blocking tickets in different states. This could lead to customer dissatisfaction and great delays in resolving issues and requests. Hence, it can be concluded that in this situation, the system is **highly unstable**.

It can be concluded that it is necessary to make sure that the tickets have sufficient information and are implemented properly to ensure that there is no increase in the build-up of tickets in the system and to avoid overworking the servers.

## 5.2 Impact of adding servers

By adding more engineers, the customers can be served promptly, leading to increase customer satisfaction. By reduction in ticket processing time, TIOBE can significantly optimise the system's response time, which could further improve customer satisfaction and increase revenue.

After testing and adding a number of servers, it is observed that the number of engineers needed to make the system stable is 35, i.e., to make the system stable, 28 engineers needed to be added to it. The components assigned to these engineers are distributed randomly, and each engineer is skilled in only 2 components. In reality, it would be better to have engineers who are skilled in essentially all the components (to ensure fair distribution of workload). The findings are discussed in the following section (the code is run 10 times and the results below are averaged over 10 runs).

## 5.2.1 Adding 28 engineers - Total 35 engineers

### Effect on Queues

It is observed that the build-up at the Accepted and Scheduled states decreases drastically (as seen in table 5.4). However, the Analysed state (on which  $S_1$  works) has a slight increase in waiting times. This is a consequence of more tickets needing to be assigned in the Open state (as seen in figure 5.2) due to an increase in engineers, more tickets needing to be closed at the Verified state (as more tickets reach the Verified state due to increase in engineers), as seen in figure 5.4. Lastly, there is a decrease in waiting times and queue lengths in all states. This indicates that the workload of each server (engineers and tester) except  $S_1$  has decreased significantly, leading to reduced waiting times and queue lengths.

State	Mean Queue length (in nr. of tickets)	Mean Waiting time (in hours)
Open	0.058 Decrease by 0.17 times	1.82 Decrease by 0.8 times
Accepted	14.91 Decrease by 0.0076 times	55.46 (approx. 7 working days) Decrease by 0.18 times
Analysed	0.067 Increase by 4.5 times	0.48 (approx. 28.8 minutes) Decrease by 0.84 times or 5.6 minutes
Scheduled	91.25 Decrease by 0.42 times	151.2 (approx. 19 working days) Decrease by 0.97 times or approx. 4.12 hours
Implemented	0.0015 Decrease by 0.098 times	9.47 Decrease by 12 minutes
Verified	0.046 Decrease by 0.88 times	0.22 (approx. 13.2 minutes) Decrease by 0.36 times

Table 5.4: Queue lengths in states when there are 35 engineers in the system

Furthermore, it is observed that the increase in the number of engineers leads to the stabilisation of the Accepted queue (figure 5.3) and Scheduled queue (figure 5.5), that is, stabilisation of the Engineer's queue (as seen in figure 5.8). It can be seen that the queues don't exceed a certain number of queue lengths, for instance, the number of tickets in the Accepted state doesn't exceed 30 tickets, the number of tickets in the Scheduled state doesn't exceed 70 tickets, and so, the number of tickets in the Engineers' queues don't exceed a certain number tickets.



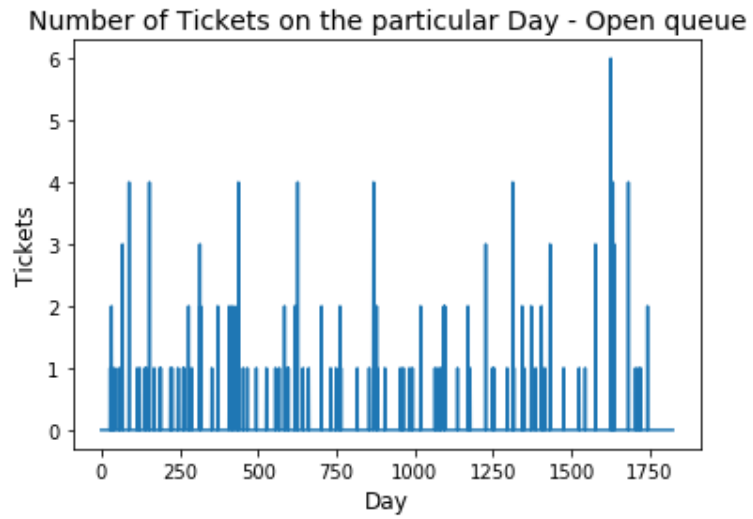


Figure 5.2: Open queue when there are 35 engineers in the system

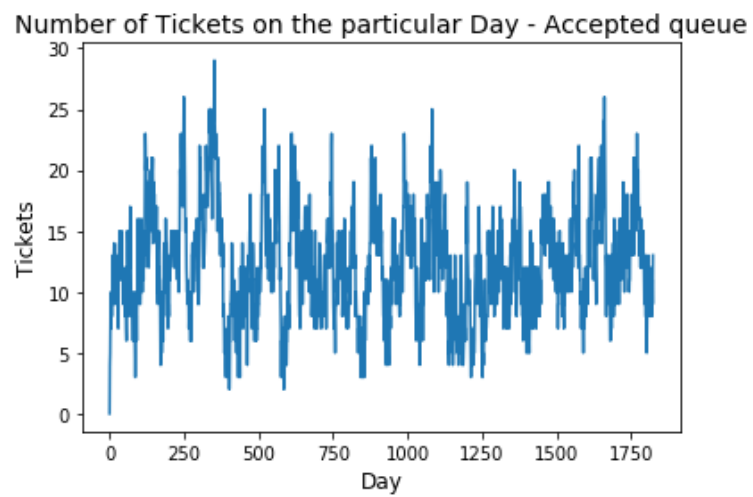


Figure 5.3: Accepted queue when there are 35 engineers in the system

Number of Tickets on the particular Day - Analysed queue

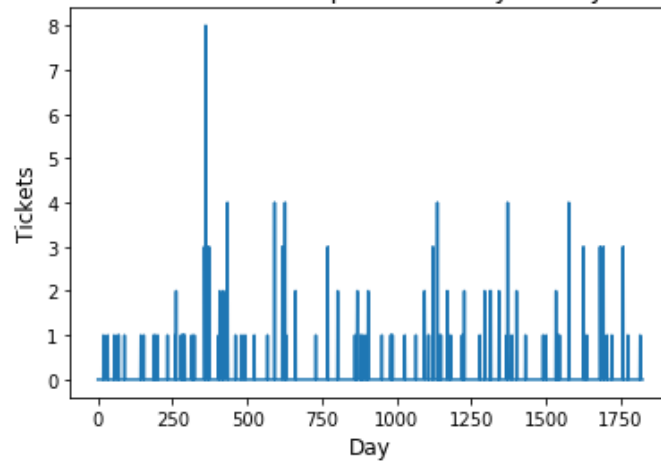


Figure 5.4: Analysed queue when there are 35 engineers in the system

Number of Tickets on the particular Day - Scheduled queue

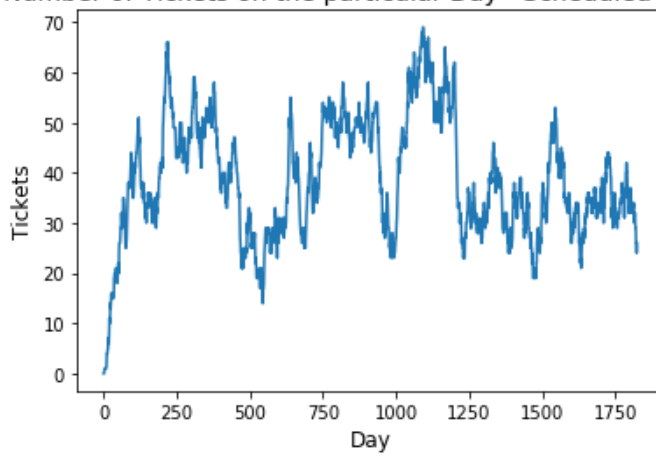


Figure 5.5: Scheduled queue when there are 35 engineers in the system

Number of Tickets on the particular Day - Implemented queue

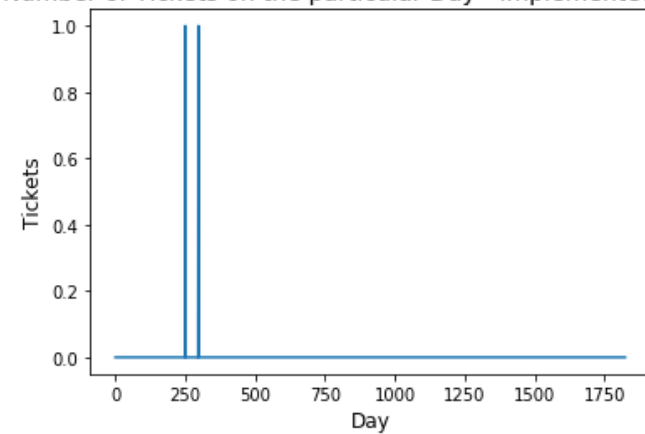


Figure 5.6: Implemented queue when there are 35 engineers in the system

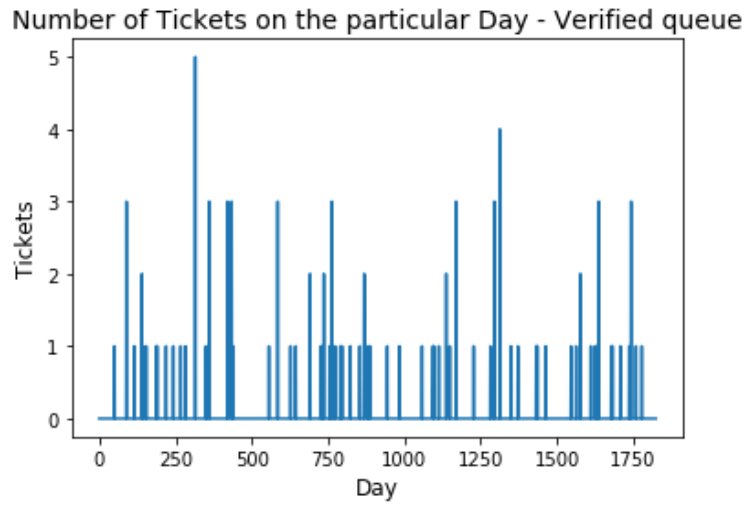


Figure 5.7: Verified queue when there are 35 engineers in the system

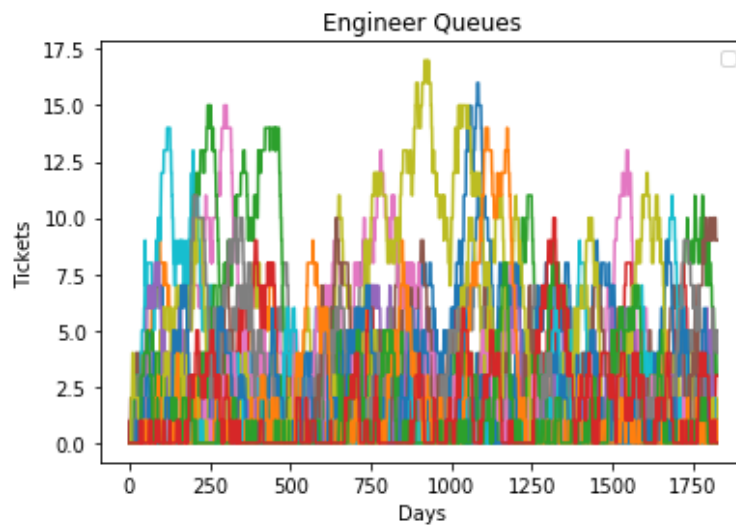


Figure 5.8: Engineers' queues when there are 35 engineers in the system

### Effect on Priority tickets

Priority	Mean Sojourn time (in hours)	Number of tickets resolved
Blocking	44.04 Decrease by 0.18 times	1693.2 Increase by 1.04 times
Urgent	101.17 Decrease by 0.81 times	1256 Increase by 19 times
High	207.60 Decrease by 0.55 times	1244 Increase by 102 times
Normal	215.3 Decrease by 0.54 times	1175.6 Increase by 111 times
Low	291.9 Decrease by 0.6 times	1204.3 Increase by 105.6 times

Table 5.5: Mean Sojourn times of tickets (grouped by priority) when there are 35 engineers in the system

As seen in table 5.5, the number of tickets of all priority that are resolved are more or less balanced (as compared to the original system, in table 4.6). Here, the number of tickets resolved in all priorities increases. Furthermore, the mean sojourn time for tickets (in every priority) decreases while the resolved tickets increase.

It is indicated that the increase in the number of engineers, therefore, leads to resolving more tickets in all priorities, which could increase customer satisfaction, improve TIOBE's service quality and distribute the workload amongst servers evenly.

### Effect on Sojourn time

The mean sojourn time for tickets when the number of engineers is increased is **167.77 hours** and the mean number of tickets resolved in a day **3.81 tickets**, as compared to the mean sojourn time and the mean number of tickets resolved in a day discussed in section 4.3.1.

It is observed that the mean sojourn time through the system decreases by 76.32 hours and the number of tickets being resolved in a day increases by a factor of  $3.87 \simeq 4$ . This indicates that the increase in the number of engineers improves the system performance by (approximately) 4 fold.

### 5.2.2 Suggested Model

Through the analysis done by adding more servers, it is observed that increasing the number of engineers (by adding 28 engineers to the team) leads to the stabilisation of the overall system and improves the system's performance. Even though there is a slight increase in the workload of the server  $S_1$ , the trade-off is worth investing in. Furthermore, the engineers should be trained in other components and skills, which could help distribute the workload fairly, even on the days when the engineers take leaves or are on a holiday.



## Chapter 6

# Conclusions

After a thorough analysis of the existing ticketing system, it can be concluded that the main bottlenecks in TIOBE's ticketing system are the engineer's queues. These queues have the highest waiting times and queue lengths, leading the system to instability. In addition to that, it is concluded that insufficient information from customers and tickets testing negative can lead to an increase in the size of these bottlenecks, which further increases the workload on the engineers.

Through the analysis of the existing system and the system conditioned to different situations, the thesis proposes a new ticketing system by increasing the number of engineers and cross-training engineers in each other's components. This model takes into account the factors such as service times, arrival rates and the number of servers to calculate the mean waiting times and queue lengths of tickets. The simulation is logged over 5 years to compare the performance of the existing system and the proposed model (as discussed in section 5.2). The results concluded that the system with 35 engineers (as discussed in section 5.2) significantly reduced the mean waiting times, queue lengths, and mean sojourn times and increased the number of tickets resolved daily. This would lead to higher customer satisfaction and system efficiency. Furthermore, by reminding the customers to provide sufficient information and implementing the tickets well, the ticketing system can avoid increased queue lengths in several queues and states.

However, in the real world, hiring 28 more engineers could heavily affect the company's revenue. To balance this, it is suggested to cross-train (to ensure that all engineers are adept in most components) and hire a few engineers that are proficient in most components. This could not only help TIOBE improve their customer satisfaction but also aid them in taking on more projects from bigger companies. It can be concluded that the practical applications of queueing theory can help improve and optimize the ticketing system for businesses like TIOBE. Moreover, as a future study, such a system can be implemented as a pilot in the company and the impact of the suggested system on the company's day-to-day operation and their comprehensive performance.



# References

- Adan, I. & Resing, J. (2015). *Queueing Systems* (Tech. Rep.). 2
- Boon, M., Van Der Boor, M., Van Leeuwaarden, J., Mathijsen, B., Van Der Pol, J. & Resing, J. (n.d.). *Stochastic Simulation using Python* (Tech. Rep.). 65
- docs.python*. (n.d.). Retrieved from <https://docs.python.org/3/library/collections.html?highlight=collections#collections.deque> 1
- Garrido, J. M. (2009). Models of Multi-Server Systems. In *Object oriented simulation* (pp. 281–295). Springer US. doi: 10.1007/978-1-4419-0516-1{\\_}22 1
- Gohil, F. & Vikash Kumar, M. (n.d.). *Ticketing System the Creative Commons Attribution License (CC BY 4.0)* (Tech. Rep.). Retrieved from <http://creativecommons.org/licenses/by/4.0> 1
- Goodman, J. B. & Massey, W. A. (1984, 12). The non-ergodic Jackson network. *Journal of Applied Probability*, 21(4), 860–869. doi: 10.2307/3213702 10
- Goos, G., Hartmanis, J., Van, J., Board, L. E., Hutchison, D., Kanade, T., ... Weikum, G. (n.d.). *LNCS 4486 - Formal Methods for Performance Evaluation* (Tech. Rep.). 10
- Harchol-Balter, M. & Wierman, A. (2005). Multi-Server Queueing Systems with Multiple Priority Classes. *Queueing Systems*, 51, 331–360. 1
- Kelly, F. P. & Laws, C. N. (1993). *Dynamic routing in open queueing networks: Brownian models, cut constraints and resource pooling* (Vol. 13; Tech. Rep.). 10
- Melamed, B. (1982). *Sojourn Times in Queueing Networks* (Vol. 7; Tech. Rep. No. 2). 48
- Singh, S., Albert, J., Mieghem, V., Gurvich, I. & Mieghem, J. A. V. (2022). *Feature-Based Priority Queuing Learning by Doing versus Learning by Viewing: An Empirical Study of Data Analyst Productivity on a Collaborative Platform at eBay View project Digital operations View project Feature-Based Priority Queuing* (Tech. Rep.). Retrieved from <https://www.researchgate.net/publication/345959314> 1
- Statistical Compendium* (Tech. Rep.). (n.d.). 12
- TIOBE*. (n.d.). Retrieved from <https://www.tiobe.com/> 1
- Xiao, L., Xu, S. H., Yao, D. D. & Zhang, H. (2022, 8). Optimal staffing for ticket queues. *Queueing Systems*. Retrieved from <https://link.springer.com/10.1007/s11134-022-09854-8> doi: 10.1007/s11134-022-09854-8 1





# Appendix A

## Simulation Code

This chapter covers the text searchable code for this thesis. The main reference used for the Source code is the Stochastic Simulation lecture notes (Boon et al., n.d.).

### A.1 Classes

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue Oct 15 15:04:11 2022
4
5 @author: 20181301
6 """
7
8 import heapq
9 from priorityQueue import PriorityQueue
10 import numpy as np
11
12
13 class Ticket:
14
15     number = 0
16
17     def __init__(self, priority, server, arrivalTime, component):
18         self.priority = priority
19         self.queue = None
20         self.pos = 0
21         self.server = server
22         self.arrivalTime = arrivalTime
23         self.systemArrivalTime = arrivalTime
24         self.component = component
25         self.depEvent = None
26         self.engineer = None
27         self.probability = np.random.uniform(0,1) #random number between 0 & 1
28         self.state = 0
29         self.testProb = np.random.uniform(0,1) #random number between 0 & 1
30         self.testster = None
31         self.number = Ticket.number
32         Ticket.number = Ticket.number + 1
33
34
35     def newPos(self, location, time):
36         self.state = location
37         self.arrivalTime = time
38
39     def leaveSystem(self, time):
40         self.state = -1
41         self.arrivalTime = -1
```

```

42     self.pos = -1
43
44     def __lt__(self, other):
45         return self.priority < other.priority
46
47     def __str__(self):
48         return "Ticket number: " + str(self.number) + " in state: " + str(self.
state) + " and priority: " + str(self.priority) + " and tester: " + str(self.
tester) + " " + str(self.depEvent)
49
50
51 class Engineer(object):
52
53     number = 0
54
55     def __init__(self, component):
56         self.pos = 0
57         self.component = component
58         self.idleTime = 0
59         self.workTime = 0
60         self.idle = True
61         self.queue = PriorityQueue() #new class PriorityQueue - enqueue, dequeue,
location check etc
62         self.startWorkingTime = 0 #time when engg starts working
63         self.startIdleTime = 0 #time when engg stops working
64         self.countAccept = 0
65         self.countImplt = 0
66         self.countSch = 0
67         self.ticket = None
68         self.acceptTickets = []
69         self.schTickets = []
70         self.impltTickets = []
71         self.nrTickets = []
72         self.nrOfTickets = 0
73         self.workTimeFraction = self.workTime/(365*5*8*3600)
74         self.idleTimeFraction = self.idleTime/(365*5*8*3600)
75         self.number = Engineer.number
76         Engineer.number = Engineer.number + 1
77
78     def dequeue(self):
79         return heapq.heappop(self.queue)
80
81     def setWorking(self, time):
82         self.idleTime = self.idleTime + time - self.startIdleTime
83         self.startWorkingTime = time
84         self.nrTickets.append(self.ticket.number)
85         self.nrOfTickets = self.nrOfTickets + 1
86
87         if self.ticket.state == 1:
88             self.acceptTickets.append(self.ticket.number)
89         elif self.ticket.state == 3:
90             self.schTickets.append(self.ticket.number)
91         elif self.ticket.state == 4:
92             self.impltTickets.append(self.ticket.number)
93
94     def setIdle(self, time):
95         self.workTime = self.workTime + time - self.startWorkingTime
96         self.startIdleTime = time
97
98     def __str__(self):
99         return "Engineer: " + str(self.number + 1)
100
101 class Tester(object): #
102
103     def __init__(self):
104         self.idleTime = 0
105

```

```

106     self.workTime = 0
107     self.idle = True
108     self.startWorkingTime = 0
109     self.startIdleTime = 0
110     self.ticket = None
111     self.nrTickets = []
112     self.nrOfTickets = 0
113     self.workTimeFraction = self.workTime/(365*5*8*3600)
114     self.idleTimeFraction = self.idleTime/(365*5*8*3600)
115
116     def setWorking(self, time):
117         self.idleTime = self.idleTime + time - self.startIdleTime
118         self.startWorkingTime = time
119         self.nrTickets.append(self.ticket.number)
120         self.nrOfTickets = self.nrOfTickets + 1
121
122     def setIdle(self, time):
123         self.workTime = self.workTime + time - self.startWorkingTime
124         self.startIdleTime = time
125
126     def __str__(self):
127         return str("tester")
128
129
130 class S(object):                                #for S1
131
132     def __init__(self):
133         self.pos = 0
134         self.idleTime = 0
135         self.workTime = 0
136         self.idle = True
137         self.startWorkingTime = 0                #time when starts working
138         self.startIdleTime = 0                  #time when stops working
139         self.workTimeFraction = self.workTime/(365*5*8*3600)
140         self.idleTimeFraction = self.idleTime/(365*5*8*3600)
141         self.openCounter = 0
142         self.impltCounter = 0
143         self.analyseCounter = 0
144         self.verfCounter = 0
145         self.nrTickets = []
146         self.nrOfTickets = 0
147         self.ticket = None
148
149     def setWorking(self, time):
150         self.idleTime = self.idleTime + time - self.startIdleTime
151         self.startWorkingTime = time
152         self.nrTickets.append(self.ticket.number)
153         self.nrOfTickets = self.nrOfTickets + 1
154
155
156     def setIdle(self, time):
157         self.workTime = self.workTime + time - self.startWorkingTime
158         self.startIdleTime = time
159
160     def __str__(self):
161         return str("s1")
162
163
164 class Event(object):
165
166     ARRIVAL = 0
167     DEPARTURE = 1
168     ABANDONMENTS = -1 #ticket rejected/wait
169     EOD = 2          #end of day
170
171
172     def __init__(self, typ, server, ticket, time):

```

```

173     self.type = typ                                #type of event
174     self.server = server
175     self.ticket = ticket
176     self.time = time
177     self.state = None
178     self.iscancelled = False                       #for cancelling event; when handelling event
, first check if it is cancelled or not
179
180     def __lt__(self, other):
181         return self.time < other.time
182
183     def __str__( self ):
184         s = ('Arrival', 'Departure', 'Abandonments', 'EOD')
185         return s[ self . type ] + " of ticket " + str( self . ticket.number )+ ' at
t = ' + str ( self . time / (8*3600) ) + ' with server = ' + str(self.server)
+ ' and tester = ' + str(self.ticket.tester)
186
187     def cancel(self):
188         self.iscancelled = True

```

## A.2 Distributions

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Wed Nov 23 21:19:28 2022
4
5  @author: 20181301
6  """
7
8
9  from scipy import stats
10 import pandas as pd
11 import numpy as np
12 import matplotlib.pyplot as plt
13
14
15 class Distribution :
16
17     n = 10000 # standard random numbers to generate
18
19     def __init__(self, dist):
20         self.dist = dist
21         self.resample()
22
23     def __str__(self):
24         return str(self.dist)
25
26     def resample(self):
27         self.randomNumbers = self.dist.rvs(self.n)
28         self.idx = 0
29
30     def rvs(self, n=1):
31
32         if self.idx >= self.n - n :
33             while n > self.n :
34                 self.n *= 10
35                 self.resample()
36         if n == 1 :
37             rs = self.randomNumbers[self.idx]
38         else :
39             rs = self.randomNumbers[self.idx:(self.idx+n)]
40         self.idx += n
41         return rs
42
43     def mean(self):

```

```

44         return self.dist.mean()
45
46     def std(self):
47         return self.dist.std()
48
49     def var(self):
50         return self.dist.var()
51
52     def cdf(self, x):
53         return self.dist.cdf(x)
54
55     def pdf(self, x):
56         return self.dist.pdf(x)
57
58     def sf(self, x):
59         return self.dist.sf(x)
60
61     def ppf(self, x):
62         return self.dist.ppf(x)
63
64     def moment(self, n):
65         return self.dist.moment(n)
66
67     def median(self):
68         return self.dist.median()
69
70     def interval(self, alpha):
71         return self.dist.interval(alpha)

```

### A.3 Future Event Set

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Fri Nov 14 00:26:08 2022
4
5  @author: 20181301
6  """
7
8  import heapq
9
10 class FES:
11
12     def __init__(self):
13         self.events = []
14
15     def add(self, event):
16         heapq.heappush(self.events, event)
17
18     def next(self):
19         return heapq.heappop(self.events)
20
21     def __str__(self):
22         r = []
23         for i in self.events:
24             r.append(str(i))
25         return str(r)

```

### A.4 Priority Queue

```

1  # -*- coding: utf-8 -*-
2  """

```

```

3 Created on Wed Nov 16 11:21:19 2022
4
5 @author: 20181301
6 """
7
8
9 from collections import deque
10 from heapq import heappop, heappush, heapify
11 import heapq
12
13 class PriorityQueue:
14     def __init__(self):
15         self.tickets = []
16         self.state = None
17
18     def enqueue(self, ticket):
19         heappush(self.tickets, ticket)
20         heapq.heapify(self.tickets)
21
22     def dequeue_0(self):
23         return heappop(self.tickets)
24
25     def dequeue_ticket(self, ticket):
26         self.tickets.remove(ticket)
27         heapq.heapify(self.tickets)
28
29     def firstTicket(self):
30         if len(self.tickets) == 0:
31             return None
32         else:
33             return self.tickets[0]
34
35     def enqueue_front(self, ticket):
36         return self.appendleft(ticket)
37
38     def __str__(self):
39         s = str(" ")
40         for i in self.tickets:
41             s = s + str(i.number) + str(", ")
42         return str("Tickets in Queue: ") + s

```

## A.5 Queue

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Nov 14 00:31:07 2022
4
5 @author: 20181301
6 """
7
8 from collections import deque
9 import collections
10
11
12 class Queue:
13     def __init__(self):
14         self.tickets = deque()
15         self.state = None
16
17     def enqueue(self, ticket):
18         self.tickets.append(ticket)
19
20     def dequeue_0(self):
21         return self.tickets.popleft()
22

```

```

23     def dequeue_ticket(self, ticket):
24         self.tickets.remove(ticket)
25
26     def firstTicket(self):
27         if len(self.tickets) == 0:
28             return None
29         else:
30             return self.tickets[0]
31
32     def enqueue_front(self, ticket):
33         return self.appendleft(ticket)
34
35     def __str__(self):
36         s = str(" ")
37         for i in self.tickets:
38             s = s + str(i.number) + str(", ")
39         return str("Tickets in Queue: ") + s

```

## A.6 Simulation Results

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Fri Nov 18 00:47:47 2022
4
5  @author: 20181301
6  """
7
8  from collections import deque
9  from numpy . ma . core import zeros
10 import matplotlib . pyplot as plt
11 import pandas as pd
12 import seaborn as sns
13 import numpy as np
14
15
16
17 class SimResults :
18
19     MAX_QL = 10000 # maximum queue length that will be recorded
20
21     def __init__ ( self ):
22         self . sumQL = 0
23         self . sumQL2 = 0
24         self . oldTime = 0
25         self . queueLengthHistogram = zeros ( self . MAX_QL + 1 )
26         self . sumW = 0
27         self . sumW2 = 0
28         self . nW = 0
29         self . waitingTimes = deque ( )
30         self.waitingTimes_hrs = deque ( )
31
32
33     def registerQueueLength ( self , time , ql ):
34         self . sumQL = self . sumQL + ql * ( time - self . oldTime )
35         self . sumQL2 = self . sumQL2 + ql * ql * ( time - self . oldTime )
36         self . queueLengthHistogram [ min ( ql , self . MAX_QL ) ] = self .
queueLengthHistogram [ min ( ql , self . MAX_QL ) ] + ( time - self . oldTime )
37         self . oldTime = time
38
39     def registerWaitingTime ( self , W ):
40         w = W
41         self.waitingTimes.append(w)
42         self.waitingTimes_hrs.append(w/3600)
43         self.nW = self.nW + 1
44         self.sumW = self.sumW + w

```



```

45     self.sumW2 = self.sumW2 + w*w
46
47     def getMeanQueueLength ( self ):
48         return self.sumQL/self.oldTime
49
50     def getVarianceQueueLength ( self ):
51         return self . sumQL2 / self . oldTime - self . getMeanQueueLength ()**2
52
53     def getMeanWaitingTime ( self ):
54         return self . sumW / self . nW
55
56     def getVarianceWaitingTime ( self ):
57         return self . sumW2 / self . nW - self . getMeanWaitingTime ()**2
58
59     def getQueueLengthHistogram ( self ) :
60         return [x/ self . oldTime for x in self . queueLengthHistogram ]
61
62     def getWaitingTimes ( self ):
63         return self . waitingTimes_hrs
64
65     def __str__ ( self ):
66         s = ' Mean queue length : ' + str ( (self . getMeanQueueLength ())) + '\n '
67         s += ' Variance queue length : ' + str ( (self . getVarianceQueueLength ()))
68         s += '\n '
69         s += ' Mean waiting time (in seconds) : ' + str (float(( self .
70         getMeanWaitingTime ())) + '\n '
71         s += ' Mean waiting time (in hours) : ' + str (float(( self .
72         getMeanWaitingTime ())/3600) + '\n '
73         s += ' Variance waiting time (in hrs^2) : ' + str (float(( self .
74         getVarianceWaitingTime ())/3600**2) + '\n '
75         return s
76
77     def histQueueLength ( self ,queueNr, maxq =50):
78
79         if queueNr == 0:
80             ql = self . getQueueLengthHistogram ( )
81             maxx = maxq + 1
82             plt.figure(figsize=(12, 6))
83             plt . figure ( )
84             plt . bar ( range ( 0 , maxx ), ql [0: maxx ])
85             plt.title("Queue Lengths (Open Queue)", fontsize = 14)
86             plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)',
87             fontsize = 10)
88             plt . xlabel ( 'k ', fontsize = 12)
89             plt . show ( )
90
91             print("\nSum of probability open queue lengths - ", sum(self.
92             getQueueLengthHistogram()))
93
94         elif queueNr == 1:
95             ql = self . getQueueLengthHistogram ( )
96             maxx = maxq + 1
97             plt.figure(figsize=(12, 6))
98             plt . figure ( )
99             plt . bar ( range ( 0 , maxx ), ql [0: maxx ])
100             plt.title("Queue Lengths (Accepted Queue)", fontsize = 14)
101             plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)',
102             fontsize = 10)
103             plt . xlabel ( 'k ', fontsize = 12)
104             plt . show ( )
105
106             print("\nSum of probability accepted queue lengths - ", sum(self.
107             getQueueLengthHistogram()))
108
109         elif queueNr == 2:
110             ql = self . getQueueLengthHistogram ( )
111             maxx = maxq + 1

```

```

104         plt.figure(figsize=(12, 6))
105         plt . figure ()
106         plt . bar ( range ( 0 , maxx ), ql [0: maxx ])
107         plt.title("Queue Lengths (Analysed Queue)", fontsize = 14)
108         plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)',
109         fontsize = 10)
110         plt . xlabel ( 'k ', fontsize = 12)
111         plt . show ()
112
113         print("Sum of probability analysed queue lengths - ", sum(self.
114         getQueueLengthHistogram()))
115
116         elif queueNr == 3:
117             ql = self . getQueueLengthHistogram ()
118             maxx = maxq + 1
119             plt.figure(figsize=(12, 6))
120             plt . figure ()
121             plt . bar ( range ( 0 , maxx ), ql [0: maxx ])
122             plt.title("Queue Lengths (Scheduled Queue)", fontsize = 14)
123             plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)',
124             fontsize = 10)
125             plt . xlabel ( 'k ', fontsize = 12)
126             plt . show ()
127
128             print("\nSum of probability scheduled queue lengths - ", sum(self.
129             getQueueLengthHistogram()))
130
131             elif queueNr == 4:
132                 ql = self . getQueueLengthHistogram ()
133                 maxx = maxq + 1
134                 plt.figure(figsize=(12, 6))
135                 plt . figure ()
136                 plt . bar ( range ( 0 , maxx ), ql [0: maxx ])
137                 plt.title("Queue Lengths (Implemented Queue)", fontsize = 14)
138                 plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)',
139                 fontsize = 10)
140                 plt . xlabel ( 'k ', fontsize = 12)
141                 plt . show ()
142
143                 print("\nSum of probability implemented queue lengths - ", sum(self.
144                 getQueueLengthHistogram()))
145
146                 elif queueNr == 5:
147                     ql = self . getQueueLengthHistogram ()
148                     maxx = maxq + 1
149                     plt.figure(figsize=(12, 6))
150                     plt . figure ()
151                     plt . bar ( range ( 0 , maxx ), ql [0: maxx ])
152                     plt.title("Queue Lengths (Verified Queue)", fontsize = 14)
153                     plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)',
154                     fontsize = 10)
155                     plt . xlabel ( 'k ', fontsize = 12)
156                     plt . show ()
157
158                     print("\nSum of probability verified queue lengths - ", sum(self.
159                     getQueueLengthHistogram()))
160
161         def histWaitingTimes ( self ,queueNr, nrBins =100):
162
163             if queueNr == 0:
164                 sns.distplot(self.getWaitingTimes(), kde=True)
165                 plt.title("Waiting times (Open Queue)")
166                 plt.xlabel("Hours")
167                 plt . show ()
168
169             elif queueNr == 1:
170                 sns.distplot(self.getWaitingTimes(), kde=True)

```

```

163     plt.title("Waiting times (Accepted Queue)")
164     plt.xlabel("Hours")
165     plt . show ()
166
167     elif queueNr == 2:
168         sns.distplot(self.getWaitingTimes(), kde=True)
169         plt.title("Waiting times (Analysed Queue)")
170         plt.xlabel("Hours")
171         plt . show ()
172
173     elif queueNr == 3:
174         sns.distplot(self.getWaitingTimes(), kde=True)
175         plt.title("Waiting times (Scheduled Queue)")
176         plt.xlabel("Hours")
177         plt . show ()
178
179     elif queueNr == 4:
180         sns.distplot(self.getWaitingTimes(), kde=True)
181         plt.title("Waiting times (Implemented Queue)")
182         plt.xlabel("Hours")
183         plt . show ()
184
185     elif queueNr == 5:
186         sns.distplot(self.getWaitingTimes(), kde=True)
187         plt.title("Waiting times (Verified Queue)")
188         plt.xlabel("Hours")
189         plt . show ()
190
191 class NetworkSimResults:
192
193     def __init__ (self , nrOfQueues , nrOfTickets ):
194         self .nS = 0
195         self . sojournTimes = []
196         self . sojournTimes_hrs = []
197         self . nrOfAbandonments = 0
198         self . queueResults = [ None ] * nrOfQueues
199         for i in range ( nrOfQueues ):
200             self . queueResults [i] = SimResults ()
201
202     def registerWaitingTime (self , queueNr , w):
203         self . queueResults [ queueNr ]. registerWaitingTime (w)
204
205     def registerSojournTime (self , s):
206         self.sojournTimes.append(s)
207         self.sojournTimes_hrs.append(s/3600)
208
209     def registerAbandonment (self , queueNr ):
210         self . nrOfAbandonments += 1
211         self . queueResults [ queueNr ]. registerAbandonment ()
212
213     def histSojournTimes ( self , nrBins =100):
214         sns.distplot(self.sojournTimes_hrs, kde=True)
215         plt.title("Sojourn times (in hours)")
216         plt . show ()

```

## A.7 Main Simulation

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Wed Nov 16 09:37:36 2022
4
5 @author: 20181301
6 """
7
8 """

```

```

9
10 States -> 0 (Open), 1 (Accepted), 2 (Analysed), 3 (Scheduled), 4 (Implemented), 5 (
    Verifying)
11
12 """
13
14 # Libraries
15
16 from classes import Event, Ticket, S, Engineer, Tester
17 import pandas as pd
18 import numpy as np
19 from numpy.random import default_rng
20 import time
21 import math
22 import random
23 from scipy import stats, integrate
24 from distribution import Distribution
25 from FES import FES
26 from simResults import SimResults, NetworkSimResults
27 from Queue import Queue
28 from collections import deque
29 from priorityQueue import PriorityQueue
30 from statistics import mean
31 import matplotlib . pyplot as plt
32 import seaborn as sns
33 import random
34 import csv
35
36 def sample_from_bernoulli(p):
37     return 1 if random.random() < p else 0
38
39 # Definitions and Declarations
40
41 s1 = S() #server 1 (Marvin)
42
43 tester = Tester() #tester (Joep)
44
45 p = sample_from_bernoulli(1) # probability of ticket being sent to analysed
    instead of being sent back to open
46 test_p = 1 # probability of ticket testing positive
47
48 df1 = pd.read_excel(r'C:\Users\20181301\Desktop\APPLIED MATH\YEAR-3\BFP\Simulation\
    Components.xlsx')
49 components = df1["Component"]
50
51 df2 = pd.read_excel(r'C:\Users\20181301\Desktop\APPLIED MATH\YEAR-3\BFP\Simulation\
    Tools & Servers.xlsx')
52 # Engineers
53 engg = []
54 engg.append(Engineer(df2["E1"]))
55 engg.append(Engineer(df2["E2"]))
56 engg.append(Engineer(df2["E3"]))
57 engg.append(Engineer(df2["E4"]))
58 engg.append(Engineer(df2["E5"]))
59 engg.append(Engineer(df2["E6"]))
60 engg.append(Engineer(df2["E7"]))
61
62 priorities = [0, 1, 2, 3, 4]
63
64
65 nrQueues = 5 #0 = Open, 1 = Analysed, 2 = Implemented, 3 = Verified
66 nrServers = 2 + len(engg) # 2 (tester and s1) and number of engineers
67
68 # Variables for storing results
69
70 priority_0 = []
71 priority_1 = []

```

```

72 priority_2 = []
73 priority_3 = []
74 priority_4 = []
75
76 count_priority_0 = []
77 count_priority_1 = []
78 count_priority_2 = []
79 count_priority_3 = []
80 count_priority_4 = []
81
82 headers = list(components.values)
83 headers.remove("Component")
84 sojournTimes_Components = pd.DataFrame(columns=headers)
85
86 headers = list(components.values)
87 headers.remove("Component")
88 totalTimes_Components = pd.DataFrame(columns=headers)
89
90 headers = [s1, tester, engg[0], engg[1], engg[2], engg[3], engg[4], engg[5], engg
    [6]]
91 sojournTimes_Servers = pd.DataFrame(columns=headers)
92 sojournTimes_Servers_idle = pd.DataFrame(columns=headers)
93
94 sojournTimes_Priorities = pd.DataFrame(columns=priorities)
95
96 totalTimes_priorities = pd.DataFrame(columns=priorities)
97
98 TotalTicketsNr = []
99
100 OpenTickets = []
101 AcceptedTickets = []
102 AnalysedTickets = []
103 ScheduledTickets = []
104 ImplementedTickets = []
105 VerifiedTickets = []
106
107 engg_1 = []
108 engg_2 = []
109 engg_3 = []
110 engg_4 = []
111 engg_5 = []
112 engg_6 = []
113 engg_7 = []
114 s1_tickets = []
115 tester_tickets = []
116
117 header = ["Tickets"]
118 nrDays_vs_Tickets = pd.DataFrame(columns = header)
119
120 def storeSojournTimes_priorities(ticket, t):      # Stores how much time each
    priority ticket spends in the system & ticket nr.
121     priority = ticket.priority
122     if priority == 0:
123         count_priority_0.append(ticket.number)
124         priority_0.append(t)
125     elif priority == 1:
126         count_priority_1.append(ticket.number)
127         priority_1.append(t)
128     elif priority == 2:
129         count_priority_2.append(ticket.number)
130         priority_2.append(t)
131     elif priority == 3:
132         count_priority_3.append(ticket.number)
133         priority_3.append(t)
134     elif priority == 4:
135         count_priority_4.append(ticket.number)
136         priority_4.append(t)

```

```

137
138 def assignFreeTester(ticket):                                # Assign idle server
139     free_testers = [server for server in engg if server.idle == True and server !=
140     ticket.engineer]
141     if free_testers:
142         return random.choice(free_testers)
143     else:
144         return None
145
146 def assignTester(ticket):                                    # Assigning tester to a priority ticket
147     servers = [server for server in engg if server != ticket.engineer and server.
148     ticket.priority != 0]
149     if servers:
150         return random.choice(servers)
151     else:
152         return None
153
154 def remove_state_ticket(engineer):
155     if engineer.ticket.state == 1:
156         engineer.acceptTickets.remove(engineer.ticket.number)
157     elif engineer.ticket.state == 3:
158         engineer.schTickets.remove(engineer.ticket.number)
159     elif engineer.ticket.state == 4:
160         engineer.impltTickets.remove(engineer.ticket.number)
161
162 def add_state_ticket(engineer, ticket):
163     if ticket.state == 1:
164         engineer.acceptTickets.append(ticket.number)
165     elif ticket.state == 3:
166         engineer.schTickets.append(ticket.number)
167     elif ticket.state == 4:
168         engineer.impltTickets.append(ticket.number)
169
170 def assignPos(t):                                           # Assigning position (used in Engineer'
171     s workflow)
172     # states - 0 (Open), 1 (Accepted), 2 (Analysed), 3 (Scheduled), 4 (Implemented)
173     , 5 (Verifying)
174     # positions - 0 for open, 1 for analysed, 2 for implemented, 3 for verified, 4
175     for engineer
176     if t.state == 0:
177         return 0
178     elif t.state == 1 or t.state == 3:
179         return 4
180     elif t.state == 2:
181         return 1
182     elif t.state == 4:
183         return 2
184     elif t.state == 5:
185         return 3
186
187 def countAccepted(server):                                  # Counting number of tickets in
188     accepted state
189     count = 0
190     for i in server.queue.tickets:
191         if i.state == 1:
192             count = count + 1
193     return count
194
195 def queueAccepted(server):                                  # Returns list of tickets in accepted
196     state
197     q_ = []
198     for i in server.queue.tickets:
199         if i.state == 1:
200             q_.append(i)
201     return q_

```

```

197 def queueImplt(queue,server):          # Returns list of tickets in
    implemented state that arent worked on by the server
198     q_ = []
199     for i in queue.tickets:
200         if i.testster == server:
201             q_.append(i)
202     return q_
203
204 def countSch(server):                  # Counts number of tickets in scheduled
    state
205     count = 0
206     for i in server.queue.tickets:
207         if i.state == 3:
208             count = count + 1
209     return count
210
211 def queueSch(server):                  # Returns list of tickets in scheduled
    state
212     q_ = []
213     for i in server.queue.tickets:
214         if i.state == 3:
215             q_.append(i)
216     return q_
217
218 def searchPriorityBlocking(q):          # Searches for blocking tickets in the
    queue
219     for ticket in q.tickets:
220         if ticket.priority == 0:
221             q.dequeue_ticket(ticket)
222             return ticket
223     return None
224
225 def searchPriorityUrgent(q):           # Searches for urgent tickets in the queue
226     for ticket in q.tickets:
227         if ticket.priority == 1:
228             q.dequeue_ticket(ticket)
229             return ticket
230     return None
231
232 def CreateTicket(t):                   # Creates tickets
233     return Ticket(random.choice(priorities), s1, t, random.choice(components))
234
235
236 class Simulation :
237
238     def __init__ ( self , arrDist , servDist , nrServers):
239         self.arrDist = arrDist
240         self.servDist = servDist
241         self.nrServers = nrServers
242
243     def simulate ( self , T ):
244         nrOfTickets = 8492
245         nrStates = 6
246         fes = FES()
247         res = NetworkSimResults (nrStates , nrOfTickets)
248
249         qs = [ None ] * nrQueues        #make them a property for s1 and tester1
250         qs[0] = Queue()                 #deque for open
251         qs[1] = Queue()                 #deque for analysed
252         qs[2] = PriorityQueue()        #heapq with priority for implemented
253         qs[3] = PriorityQueue()        #heapq with priority for verified
254         #4 <- engineer
255
256         qs[0].state = 0
257         qs[1].state = 2
258         qs[2].state = 4
259         qs[3].state = 5

```

```

260
261     #engineer service times
262
263
264     t = 0 # current time
265     a0 = self.arrDist.rvs()           # Create arrival of first ticket
266     t0 = CreateTicket(t)             #ticket creation
267     firstEvent = Event(Event.ARRIVAL, s1, t0, a0 )   #remove S_1 and put queue
number 0 = open queue
268     fes.add(firstEvent)
269
270     fes.add(Event(Event.EOD, s1, t0, 8*3600))       # first end-of-day event
271     day = 0
272     tickets_in_system_day = []
273
274     while t < T :
275         e = fes.next()
276         t = e.time
277         c = e.ticket
278         queueNr = c.pos
279
280         tot_sch_queue = 0
281         tot_accepted_queue = 0
282
283         for i in engg:
284             tot_sch_queue = tot_sch_queue + countSch(i)
285             tot_accepted_queue = tot_accepted_queue + countAccepted(i)
286
287         res.queueResults[0].registerQueueLength(t, len (qs[0].tickets))
288         res.queueResults[1].registerQueueLength(t, tot_accepted_queue)
289         res.queueResults[2].registerQueueLength(t, len (qs[1].tickets))
290         res.queueResults[3].registerQueueLength(t, tot_sch_queue)
291         res.queueResults[4].registerQueueLength(t, len (qs[2].tickets))
292         res.queueResults[5].registerQueueLength(t, len (qs[3].tickets))
293
294
295         if e.type == Event.ARRIVAL :
296
297             #ALGORITHM 1 (ARRIVAL AT OPEN)
298             if queueNr == 0:
299
300                 qs[0].enqueue(c)
301
302                 if s1.idle == True:
303                     b0 = self.servDist[0].rvs()
304                     dep = Event(Event.DEPARTURE, s1, c, t+b0)
305                     fes.add(dep)
306                     c.depEvent = dep   #ticket knows its departure and service
time
307
308                     qs[0].dequeue_ticket(c)
309                     # print(e)
310                     s1.idle = False
311                     s1.ticket = c
312                     c.state = 0
313                     c.pos = 0
314                     s1.openCounter = s1.openCounter + 1
315                     s1.setWorking(t)
316                     a1 = self.arrDist.rvs()
317                     c1 = CreateTicket(t)
318                     fes.add(Event(Event.ARRIVAL , s1 , c1, t+a1))
319
320             #ALGORITHM 3 (ARRIVAL AT ACCEPTED)
321             elif queueNr == 4 and c.state == 1:           #arrival at engineer's
queue
322                 i = c.engineer.number
323                 engg[i].queue.enqueue(c)

```



```

324         serv = engg[i]
325
326
327         if serv.idle == True:
328             b1 = self.servDist[1].rvs()
329             dep = Event(Event.DEPARTURE, serv, c, t+b1)
330             fes.add(dep)
331             serv.ticket = c
332             c.depEvent = dep      #ticket knows its departure and service
333
334         time
335
336             serv.idle = False
337             c.state = 1
338             c.pos = 4
339             c.engineer = serv
340             serv.queue.dequeue_ticket(c)
341             serv.setWorking(t)
342             serv.countAccept = serv.countAccept + 1
343
344         elif c.priority == 0 and c.depEvent == None:      #if ticket
345         hasn't been assigned (since serv is busy) & has priority
346         #cancelling departure event of the ticket the engineer is
347         working on if that ticket lower priority
348         c2 = serv.ticket      # current ticket he's working
349         on
350         if c2 != None:
351             if c2.priority != 0 and c2.depEvent != None:
352             # the current ticket has a priority of 0
353             c2.depEvent.iscancelled = True
354             c2.depEvent = None
355             serv.nrTickets.remove(serv.ticket.number)
356             remove_state_ticket(serv)
357             serv.nrTickets.append(c.number)
358             add_state_ticket(serv, c)
359             serv.queue.dequeue_ticket(c)
360             b1 = self.servDist[1].rvs()
361             dep = Event(Event.DEPARTURE, serv, c, t+b1)
362             fes.add(dep)
363             c.depEvent = dep      #ticket knows its departure and
364         service time
365
366             serv.idle = False
367             serv.ticket = c
368             c.state = 1
369             c.pos = 4
370             serv.countAccept = serv.countAccept + 1
371
372         #ALGORITHM 5 (ARRIVAL AT ANALYSED)
373         elif queueNr == 1:
374             qs[1].enqueue(c)
375
376         if s1.idle == True:
377             b2 = self.servDist[2].rvs()
378             dep = Event(Event.DEPARTURE, s1, c, t+b2)
379             qs[1].dequeue_ticket(c)
380             fes.add(dep)
381             c.depEvent = dep      #ticket knows its departure and service
382         time
383
384             s1.idle = False
385             s1.ticket = c
386             c.state = 2
387             c.pos = 1
388             s1.setWorking(t)
389             s1.analyseCounter = s1.analyseCounter + 1
390
391         #ALGORITHM 7 (ARRIVAL AT SCHEDULED)
392         elif queueNr == 4 and c.state == 3:
393             serv = c.engineer

```

```

384         serv.queue.enqueue(c)
385
386
387         if c.engineer.idle == True:
388             b2 = self.servDist[3].rvs()
389             serv.queue.dequeue_ticket(c)
390             dep = Event(Event.DEPARTURE, serv, c, t+b2)
391             fes.add(dep)
392             c.depEvent = dep      #ticket knows its departure and service
time
393
394             serv.idle = False
395             c.state = 3
396             c.pos = 4
397             serv.ticket = c
398             c.engineer.setWorking(t)
399             serv.countSch = serv.countSch + 1
400
401         #ALGORITHM 9 (ARRIVAL AT IMPLEMENTED)
402         elif queueNr == 2:
403
404             qs[2].enqueue(c)
405
406
407             if tester.idle == True:
408                 b2 = self.servDist[4].rvs()
409                 dep = Event(Event.DEPARTURE, tester, c, t+b2)
410                 fes.add(dep)
411                 c.depEvent = dep      #ticket knows its departure and service
time
412
413                 qs[2].dequeue_ticket(c)
414                 tester.idle = False
415                 c.state = 4
416                 tester.ticket = c
417                 c.testers = tester
418                 c.pos = 2
419                 tester.setWorking(t)
420
421             else:
422
423                 serv = assignFreeTester(c)
424                 if serv != None:
425                     b2 = self.servDist[4].rvs()
426                     dep = Event(Event.DEPARTURE, serv, c, t+b2)
427                     qs[2].dequeue_ticket(c)
428                     fes.add(dep)
429                     c.depEvent = dep      #ticket knows its departure and
service time
430
431                     serv.idle = False
432                     c.state = 4
433                     serv.ticket = c
434                     c.testers = serv
435                     c.pos = 2
436                     serv.setWorking(t)
437                     serv.countImplt = serv.countImplt + 1
438
439                 if c.priority == 0 and c.testers == None:      # if priority
is 0 and no testers is assigned (since none are idle)
440
441                     if tester.ticket != None and tester.ticket.priority != 0 :
442                         c2 = tester.ticket      #current ticket he's working on
443                         if c2.depEvent != None:
444                             c2.depEvent.iscancelled = True
445                             c2.depEvent = None
446                             tester.nrTickets.remove(testers.ticket.number)
447                             tester.nrTickets.append(c.number)
448                             b2 = self.servDist[4].rvs()

```

```

447         dep = Event(Event.DEPARTURE, tester, c, t+b2)
448         fes.add(dep)
449         c.depEvent = dep      #ticket knows its departure and
service time
450         qs[2].dequeue_ticket(c)
451         tester.idle = False
452         tester.ticket = c
453         c.state = 4
454         c.testter = tester
455         c.pos = 2
456
457     else:
458         i = assignTester(c)
459         if i != None:
460             c2 = i.ticket      #current ticket he's working on
461             if c2 != None and c2.priority != 0 and c2.depEvent
!= None:
462                 c2.depEvent.iscancelled = True
463                 c2.depEvent = None
464                 i.nrTickets.remove(i.ticket.number)
465                 i.nrTickets.append(c.number)
466                 remove_state_ticket(i)
467                 add_state_ticket(i, c)
468                 b2 = self.servDist[4].rvs()
469                 dep = Event(Event.DEPARTURE, i, c, t+b2)
470                 qs[2].dequeue_ticket(c)
471                 fes.add(dep)
472                 c.depEvent = dep      #ticket knows its departure
and service time
473                 i.idle = False
474                 i.ticket = c
475                 c.state = 4
476                 c.testter = i
477                 c.pos = 2
478                 i.countImplt = i.countImplt + 1
479
480
481     #ALGORITHM 11 (ARRIVAL AT VERIFIED)
482     elif queueNr == 3:
483         qs[3].enqueue(c)
484
485
486     if s1.idle == True:
487         b3 = self.servDist[5].rvs()
488         dep = Event(Event.DEPARTURE, s1, c, t+b3)
489         qs[3].dequeue_ticket(c)
490         fes.add(dep)
491         c.depEvent = dep      #ticket knows its departure and service
time
492         s1.idle = False
493         s1.ticket = c
494         c.state = 5
495         c.pos = 3
496         s1.setWorking(t)
497
498     else:
499         # print(e)
500         if len(qs[3].tickets) > 0:
501             if c.priority == 0:
502                 c2 = s1.ticket
503                 if c2 != None:
504                     if c2.priority != 0 and c2.depEvent != None:
505                         c2.depEvent.iscancelled = True
506                         c2.depEvent = None
507                         s1.nrTickets.remove(s1.ticket.number)
508                         s1.nrTickets.append(c.number)
509                         b2 = self.servDist[5].rvs()

```

```

510         dep = Event(Event.DEPARTURE, s1, c, t+b2)
511         qs[3].dequeue_ticket(c)
512         fes.add(dep)
513         c.depEvent = dep      #ticket knows its
departure and service time
514         s1.idle = False
515         s1.ticket = c
516         c.state = 5
517         c.pos = 3
518
519
520     elif e.type == Event.DEPARTURE :
521         if e.iscancelled == False:
522             c.depEvent = None
523
524             #ALGORITHM 2 (DEPARTURE FROM OPEN)
525             if queueNr == 0:
526
527                 for i in engg:
528                     for comp in i.component:
529                         if comp == c.component:
530                             c.engineer = i
531                             fes.add(Event(Event.ARRIVAL, i, c, t)) #
schedule arrival time
532                             res.registerWaitingTime (0, t - c.arrivalTime )
533                             sojournTimes_Components.loc[len(
sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
534                             c.newPos(1, t)
535                             c.pos = 4
536                             break
537
538                             # Once server stops working, set to idle
539                             s1.setIdle(t)
540                             s1.idle = True
541
542                             # S1 WORKFLOW
543                             if s1.openCounter <= 5 and len(qs[0].tickets) > 0:
544                                 c2 = qs[0].dequeue_0()
545                                 b0 = self.servDist[0].rvs()
546                                 dep = Event(Event.DEPARTURE, s1, c2, t+b0)
547                                 c2.depEvent = dep      #ticket knows its departure and
service time
548                                 fes.add(dep)
549                                 s1.idle = False
550                                 s1.ticket = c2
551                                 c2.server = s1
552                                 c2.state = 0
553                                 c2.pos = 0
554                                 s1.openCounter += 1
555                                 s1.setWorking(t)
556
557                             else:
558                                 c1 = searchPriorityBlocking(qs[3])
559                                 c3 = searchPriorityUrgent(qs[3])
560
561                                 if c1 != None:
562
563                                     if s1.idle == True:
564                                         b3 = self.servDist[5].rvs()
565                                         dep = Event(Event.DEPARTURE, s1, c1, t+b3)
566                                         fes.add(dep)
567                                         c1.depEvent = dep      #ticket knows its
departure and service time
568                                         s1.idle = False
569                                         s1.ticket = c1
570                                         c1.state = 5      #goes into verified state
571                                         c1.pos = 3

```

```

572         s1.setWorking(t)
573
574         else:
575             if s1.ticket.priority != 0 and s1.ticket.
depEvent != None:
576                 s1.ticket.depEvent.iscancelled = True
577                 s1.ticket.depEvent = None
578                 s1.nrTickets.remove(s1.ticket.number)
579                 s1.nrTickets.append(c1.number)
580                 b3 = self.servDist[5].rvs()
581                 dep = Event(Event.DEPARTURE, s1, c1, t+b3)
582                 fes.add(dep)
583                 c1.depEvent = dep      #ticket knows its
departure and service time
584                 s1.idle = False
585                 s1.ticket = c1
586                 c1.state = 5      #goes into verified state
587                 c1.pos = 3
588
589             elif c3 != None:
590                 if s1.idle == True:
591                     b3 = self.servDist[5].rvs()
592                     dep = Event(Event.DEPARTURE, s1, c3, t+b3)
593                     fes.add(dep)
594                     c3.depEvent = dep      #ticket knows its
departure and service time
595                     s1.idle = False
596                     c3.state = 5      #goes into verified state
597                     c3.pos = 3
598                     s1.ticket = c3
599                     s1.setWorking(t)
600
601                 else:
602                     if s1.ticket.priority != 0 and s1.ticket.
priority != 1 and serv.ticket.depEvent != None:
603                         s1.ticket.depEvent.iscancelled = True
604                         s1.ticket.depEvent = None
605                         s1.nrTickets.remove(s1.ticket.number)
606                         s1.nrTickets.append(c3.number)
607                         b3 = self.servDist[5].rvs()
608                         dep = Event(Event.DEPARTURE, s1, c3, t+b3)
609                         fes.add(dep)
610                         c3.depEvent = dep      #ticket knows its
departure and service time
611                         s1.idle = False
612                         c3.state = 5      #goes into verified state
613                         c3.pos = 3
614                         s1.ticket = c3
615
616                 else:
617                     if s1.idle == True:
618
619                         if len(qs[3].tickets) > 0:
620                             c3 = qs[3].dequeue_0()
621                             b3 = self.servDist[5].rvs()
622                             dep = Event(Event.DEPARTURE, s1, c3, t+b3)
623                             fes.add(dep)
624                             c3.depEvent = dep      #ticket knows its
departure and service time
625                             s1.ticket = c3
626                             c3.pos = 3
627                             c3.state = 5
628                             s1.idle = False
629                             s1.setWorking(t)
630
631                     elif len(qs[2].tickets) > 0:
632

```

```

633         if s1.impltCounter < 2:
634             c3 = qs[2].dequeue_0() #ticket is not
being served
635             b3 = self.servDist[4].rvs()
636             dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
637             fes.add(dep)
638             c3.depEvent = dep #ticket knows its
departure and service time
639             s1.ticket = c3
640             c3.testster = s1
641             s1.idle = False
642             c3.state = 4 #goes into implemented
state
643             c3.pos = 2
644             s1.impltCounter += 1
645             s1.setWorking(t)
646
647         elif len(qs[1].tickets) > 0:
648
649             if s1.analyseCounter < 4:
650                 c3 = qs[1].dequeue_0()
651                 b3 = self.servDist[2].rvs()
652                 dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
653                 fes.add(dep)
654                 c3.depEvent = dep #ticket knows its
departure and service time
655                 s1.ticket = c3
656                 c3.state = 2 #goes into analysed
state
657                 c3.pos = 1
658                 s1.idle = False
659                 s1.analyseCounter += 1
660                 s1.setWorking(t)
661
662             elif len(qs[2].tickets) > 0 and len(qs[2].
tickets) < 20:
663
664                 #arrival at implemented
665                 if s1.impltCounter < 2:
666                     c3 = qs[2].dequeue_0()
667                     b3 = self.servDist[4].rvs()
668                     dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
669                     fes.add(dep)
670                     c3.depEvent = dep #ticket knows its
departure and service time
671                     s1.idle = False
672                     s1.ticket = c3
673                     c3.testster = s1
674                     c3.state = 4 #goes into implemented
state
675                     c3.pos = 2
676                     s1.impltCounter += 1
677                     s1.setWorking(t)
678
679
680             #ALGORITHM 4 (DEPARTURE FROM ACCEPTED - ENGINEER QUEUE)
681             elif queueNr == 4 and c.state == 1: #engineer's queue
682
683                 if c.priority == 0:
684                     fes.add(Event(Event.ARRIVAL, c.engineer, c, t)) #
schedule arrival at implemented. c.engineer SHOULD NOT work on testing
685                     res.registerWaitingTime(1, t - c.arrivalTime)
686                     sojournTimes_Components.loc[len(sojournTimes_Components
.index), c.component] = (t-c.arrivalTime)/3600

```

```

687         c.newPos(4, t)
688         c.pos = 2
689
690     else:
691
692         if c.probability < p: #sample p from a bernoulli
dist. - c.probability is probability of NOT being sent back due to insufficient
information
693             fes.add(Event(Event.ARRIVAL, s1, c, t)) #schedule
arrival at analysed
694             res.registerWaitingTime (1, t - c.arrivalTime)
695             sojournTimes_Components.loc[len(
sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
696             c.newPos(2, t)
697             c.pos = 1
698
699     else:
700         fes.add(Event(Event.ARRIVAL, s1, c, t)) #schedule
arrival at open
701         res.registerWaitingTime (1, t - c.arrivalTime)
702         sojournTimes_Components.loc[len(
sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
703         c.newPos(0, t)
704         c.pos = 0
705
706         # Once server stops working, set to idle
707         c.engineer.idle = True
708         c.engineer.setIdle(t)
709
710         # Engineer's Workflow
711         serv = c.engineer
712         c3 = searchPriorityBlocking(serv.queue)
713         c4 = searchPriorityUrgent(serv.queue)
714
715         if c3 != None:
716             if serv.idle == True:
717                 s = c3.state
718                 b3 = self.servDist[s].rvs()
719                 dep = Event(Event.DEPARTURE, serv, c3, t+b3)
720                 fes.add(dep)
721                 c3.depEvent = dep #ticket knows its departure
and service time
722                 serv.idle = False
723                 serv.ticket = c3
724                 c3.state = s #goes into next state
725                 c3.pos = assignPos(c3)
726                 serv.setWorking(t)
727
728             else:
729                 if serv.ticket.priority != 0 and serv.ticket.
depEvent != None:
730                     serv.ticket.depEvent.iscancelled = True
731                     serv.ticket.depEvent = None
732                     serv.nrTickets.remove(serv.ticket.number)
733                     remove_state_ticket(serv)
734                     add_state_ticket(serv, c3)
735                     serv.nrTickets.append(c3.number)
736                     s = c3.state
737                     b3 = self.servDist[s].rvs()
738                     dep = Event(Event.DEPARTURE, serv, c3, t+b3)
739                     fes.add(dep)
740                     c3.depEvent = dep #ticket knows its
departure and service time
741                     serv.idle = False
742                     serv.ticket = c3
743                     c3.state = s #goes into next state
744                     c3.pos = assignPos(c3)

```

```

745
746         elif c4 != None:
747             if serv.idle == True:
748                 s = c4.state
749                 b3 = self.servDist[s].rvs()
750                 dep = Event(Event.DEPARTURE, serv, c4, t+b3)
751                 fes.add(dep)
752                 c4.depEvent = dep      #ticket knows its departure
and service time
753
754                 serv.idle = False
755                 serv.ticket = c4
756                 c4.state = s      #goes into next state
757                 c4.pos = assignPos(c4)
758                 serv.setWorking(t)
759
760             else:
761                 if serv.ticket.priority !=0 and serv.ticket.
priority != 1 and serv.ticket.depEvent != None:
762                     serv.ticket.depEvent.iscancelled = True
763                     serv.ticket.depEvent = None
764                     serv.nrTickets.remove(serv.ticket.number)
765                     serv.nrTickets.append(c4.number)
766                     remove_state_ticket(serv)
767                     add_state_ticket(serv, c4)
768                     s = c4.state
769                     b3 = self.servDist[s].rvs()
770                     dep = Event(Event.DEPARTURE, serv, c4, t+b3)
771                     fes.add(dep)
772                     c4.depEvent = dep      #ticket knows its
departure and service time
773
774                     serv.idle = False
775                     serv.ticket = c4
776                     c4.state = s      #goes into next state
777                     c4.pos = assignPos(c4)
778
779                 else:
780                     if serv.idle == True:
781                         if countAccepted(serv) > 0:
782                             if serv.countAccept < 2:
783                                 q_ = queueAccepted(serv)
784                                 c3 = q_[0]
785                                 serv.queue.dequeue_ticket(c3)
786                                 b3 = self.servDist[1].rvs()
787                                 dep = Event(Event.DEPARTURE, serv, c3, t+b3
)
788
789                                 fes.add(dep)
790                                 c3.depEvent = dep      #ticket knows its
departure and service time
791
792                                 serv.idle = False
793                                 serv.ticket = c3
794                                 c3.state = 1      #goes into accepted state
795                                 c3.pos = 4
796                                 serv.countAccept += 1
797                                 serv.setWorking(t)
798
799                                 elif len(qs[2].tickets) > 0:
800                                     if serv.countImplt < 2:
801                                         q_ = queueImplt(qs[2], serv)
802                                         if len(q_) != 0:
803                                             c3 = q_[0]
804                                             qs[2].dequeue_ticket(c3)
805                                             b3 = self.servDist[4].rvs()
806                                             dep = Event(Event.DEPARTURE, serv, c3,
t+b3)
807
808                                             fes.add(dep)
809                                             c3.depEvent = dep      #ticket knows its
departure and service time

```



```

805         serv.idle = False
806         serv.ticket = c3
807         c3.testster = serv
808         c3.state = 4    #goes into implemented
state
809
810         c3.pos = 2
811         serv.countImplt += 1
812         serv.setWorking(t)
813
814     elif countSch(serv) > 0:
815         if serv.countSch < 2:
816             q_ = queueSch(serv)
817             c3 = q_[0]
818             serv.queue.dequeue_ticket(c3)
819             b3 = self.servDist[3].rvs()
820             dep = Event(Event.DEPARTURE, serv, c3, t+b3
)
821
822         fes.add(dep)
823         c3.depEvent = dep    #ticket knows its
departure and service time
824
825         serv.idle = False
826         serv.ticket = c3
827         c3.state = 3    #goes into scheduled state
828         c3.pos = 4
829         serv.countSch += 1
830         serv.setWorking(t)
831
832     #ALGORITHM 6 (DEPARTURE FROM ANALYSED)
833     elif queueNr == 1:
834
835         serv = c.engineer
836         fes.add(Event(Event.ARRIVAL, serv, c, t)) #schedule
arrival at engineer's queue.
837         res.registerWaitingTime (2, t - c.arrivalTime)
838         sojournTimes_Components.loc[len(sojournTimes_Components.
index), c.component] = (t-c.arrivalTime)/3600
839         c.pos = 4
840         c.newPos(3, t)
841
842         # Once server stops working, set to idle
843         s1.idle = True
844         s1.setIdle(t)
845
846         # S1 WORKFLOW
847         if s1.openCounter <= 5 and len(qs[0].tickets) > 0:
848
849             c2 = qs[0].dequeue_0()
850             b0 = self.servDist[0].rvs()
851             dep = Event(Event.DEPARTURE, s1, c2, t+b0)
852             c2.depEvent = dep    #ticket knows its departure and
service time
853
854         fes.add(dep)
855         s1.idle = False
856         s1.ticket = c2
857         c2.server = s1
858         c2.state = 0
859         c2.pos = 0
860         s1.openCounter += 1
861         s1.setWorking(t)
862
863     else:
864         c1 = searchPriorityBlocking(qs[3])
865         c3 = searchPriorityUrgent(qs[3])
866         if c1 != None:

```

```

866         if s1.idle == True:
867             b3 = self.servDist[5].rvs()
868             dep = Event(Event.DEPARTURE, s1, c1, t+b3)
869             fes.add(dep)
870             c1.depEvent = dep      #ticket knows its
departure and service time
871             s1.idle = False
872             s1.ticket = c1
873             c1.state = 5      #goes into verified state
874             c1.pos = 3
875             s1.setWorking(t)
876
877         else:
878             if s1.ticket.priority != 0:
879                 s1.ticket.depEvent.iscancelled = True
880                 s1.ticket.depEvent = None
881                 s1.nrTickets.remove(s1.ticket.number)
882                 s1.nrTickets.append(c1.number)
883                 b3 = self.servDist[5].rvs()
884                 dep = Event(Event.DEPARTURE, s1, c1, t+b3)
885                 fes.add(dep)
886                 c1.depEvent = dep      #ticket knows its
departure and service time
887                 s1.idle = False
888                 s1.ticket = c1
889                 c1.state = 5      #goes into verified state
890                 c1.pos = 3
891
892             elif c3 != None:
893                 if s1.idle == True:
894                     b3 = self.servDist[5].rvs()
895                     dep = Event(Event.DEPARTURE, s1, c3, t+b3)
896                     fes.add(dep)
897                     c3.depEvent = dep      #ticket knows its
departure and service time
899                     s1.idle = False
900                     c3.state = 5      #goes into verified state
901                     c3.pos = 3
902                     s1.ticket = c3
903                     s1.setWorking(t)
904
905                 else:
906                     if s1.ticket.priority != 0 and s1.ticket.
priority != 1:
907                         s1.ticket.depEvent.iscancelled = True
908                         s1.ticket.depEvent = None
909                         s1.nrTickets.remove(s1.ticket.number)
910                         s1.nrTickets.append(c3.number)
911                         b3 = self.servDist[5].rvs()
912                         dep = Event(Event.DEPARTURE, s1, c3, t+b3)
913                         fes.add(dep)
914                         c3.depEvent = dep      #ticket knows its
departure and service time
915                         s1.idle = False
916                         c3.state = 5
917                         c3.pos = 3
918                         s1.ticket = c3
919                         # s1.setWorking(t)
920
921                 else:
922                     if s1.idle == True:
923
924                         if len(qs[3].tickets) > 0:
925                             c3 = qs[3].dequeue_0()
926                             b3 = self.servDist[5].rvs()
927                             dep = Event(Event.DEPARTURE, s1, c3, t+b3)

```

```

928         fes.add(dep)
929         c3.depEvent = dep      #ticket knows its
departure and service time
930         s1.ticket = c3
931         c3.pos = 3
932         c3.state = 5
933         s1.idle = False
934         s1.setWorking(t)
935
936
937     elif len(qs[2].tickets) > 0:
938
939         if s1.impltCounter < 2:
940             c3 = qs[2].dequeue_0() #ticket is not
being served
941
942             b3 = self.servDist[4].rvs()
943             dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
944
945             fes.add(dep)
946             c3.depEvent = dep      #ticket knows its
departure and service time
947
948             s1.ticket = c3
949             c3.testster = s1
950             s1.idle = False
951             c3.state = 4      #goes into implemented
state
952
953             c3.pos = 2
954             s1.impltCounter += 1
955             s1.setWorking(t)
956
957
958     elif len(qs[1].tickets) > 0:
959
960         if s1.analyseCounter < 4:
961             c3 = qs[1].dequeue_0()
962             b3 = self.servDist[2].rvs()
963             dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
964
965             fes.add(dep)
966             c3.depEvent = dep      #ticket knows its
departure and service time
967
968             s1.ticket = c3
969             c3.state = 2      #goes into analysed
state
970
971             c3.pos = 1
972             s1.idle = False
973             s1.analyseCounter += 1
974             s1.setWorking(t)
975
976
977     elif len(qs[2].tickets) > 0 and len(qs[2].
tickets) < 20:
978
979         if s1.impltCounter < 2:
980             c3 = qs[2].dequeue_0()
981             b3 = self.servDist[4].rvs()
982             dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
983
984             fes.add(dep)
985             c3.depEvent = dep      #ticket knows its
departure and service time
986
987             s1.idle = False
988             s1.ticket = c3
989             c3.testster = s1
990             c3.state = 4      #goes into implemented
state
991
992             c3.pos = 2
993             s1.impltCounter += 1
994             s1.setWorking(t)

```

```

983
984
985     #ALGORITHM 8 (DEPARTURE FROM SCHEDULED)
986     elif queueNr == 4 and c.state == 3:
987         fes.add(Event(Event.ARRIVAL, tester, c, t)) #schedule
arrival at implemented
988         res.registerWaitingTime (3, t - c.arrivalTime)
989         sojournTimes_Components.loc[len(sojournTimes_Components.
index), c.component] = (t-c.arrivalTime)/3600
990         c.newPos(4, t)
991         c.pos = 2
992
993         # Once server stops working, set to idle
994         c.engineer.idle = True
995         c.engineer.setIdle(t)
996
997         # Engineer's decision tree
998         serv = c.engineer
999         c3 = searchPriorityBlocking(serv.queue)
1000         c4 = searchPriorityUrgent(serv.queue)
1001         if c3 != None:
1002             if serv.idle == True:
1003                 s = c3.state
1004                 b3 = self.servDist[s].rvs()
1005                 dep = Event(Event.DEPARTURE, serv, c3, t+b3)
1006                 fes.add(dep)
1007                 c3.depEvent = dep #ticket knows its departure
and service time
1008                 serv.idle = False
1009                 serv.ticket = c3
1010                 c3.state = s #goes into next state
1011                 c3.pos = assignPos(c3)
1012                 serv.setWorking(t)
1013
1014             else:
1015                 if serv.ticket.priority != 0 and serv.ticket.
depEvent != None:
1016                     serv.ticket.depEvent.iscancelled = True
1017                     serv.ticket.depEvent = None
1018                     remove_state_ticket(serv)
1019                     serv.nrTickets.remove(serv.ticket.number)
1020                     serv.nrTickets.append(c3.number)
1021                     add_state_ticket(serv, c3)
1022                     s = c3.state
1023                     b3 = self.servDist[s].rvs()
1024                     dep = Event(Event.DEPARTURE, serv, c3, t+b3)
1025                     fes.add(dep)
1026                     c3.depEvent = dep #ticket knows its
departure and service time
1027                     serv.idle = False
1028                     serv.ticket = c3
1029                     c3.state = s #goes into next state
1030                     c3.pos = assignPos(c3)
1031
1032                 elif c4 != None:
1033                     if serv.idle == True:
1034                         s = c4.state
1035                         b3 = self.servDist[s].rvs()
1036                         dep = Event(Event.DEPARTURE, serv, c4, t+b3)
1037                         fes.add(dep)
1038                         c4.depEvent = dep #ticket knows its departure
and service time
1039                         serv.idle = False
1040                         serv.ticket = c4
1041                         c4.state = s #goes into next state
1042                         c4.pos = assignPos(c4)
1043                         serv.setWorking(t)

```

```

1044
1045         else:
1046             if serv.ticket.priority != 0 and serv.ticket.
priority != 1 and serv.ticket.depEvent != None:
1047                 serv.ticket.depEvent.iscancelled = True
1048                 serv.ticket.depEvent = None
1049                 remove_state_ticket(serv)
1050                 serv.nrTickets.remove(serv.ticket.number)
1051                 serv.nrTickets.append(c4.number)
1052                 add_state_ticket(serv, c4)
1053                 s = c4.state
1054                 b3 = self.servDist[s].rvs()
1055                 dep = Event(Event.DEPARTURE, serv, c4, t+b3)
1056                 fes.add(dep)
1057                 c4.depEvent = dep      #ticket knows its
departure and service time
1058
1059                 serv.idle = False
1060                 serv.ticket = c4
1061                 c4.state = s      #goes into next state
1062                 c4.pos = assignPos(c4)
1063
1064         else:
1065             if serv.idle == True:
1066                 if countAccepted(serv) > 0:
1067                     if serv.countAccept < 2:
1068                         q_ = queueAccepted(serv)
1069                         c3 = q_[0]
1070                         serv.queue.dequeue_ticket(c3)
1071                         b3 = self.servDist[1].rvs()
1072                         dep = Event(Event.DEPARTURE, serv, c3, t+b3
)
1073
1074                         fes.add(dep)
1075                         c3.depEvent = dep      #ticket knows its
departure and service time
1076
1077                         serv.idle = False
1078                         serv.ticket = c3
1079                         c3.state = 1      #goes into accepted state
1080                         c3.pos = 4
1081                         serv.countAccept += 1
1082                         serv.setWorking(t)
1083
1084                 elif len(qs[2].tickets) > 0:
1085                     if serv.countImplt < 2:
1086                         q_ = queueImplt(qs[2], serv)
1087                         if len(q_) != 0:
1088                             c3 = q_[0]
1089                             qs[2].dequeue_ticket(c3)
1090                             b3 = self.servDist[4].rvs()
1091                             dep = Event(Event.DEPARTURE, serv, c3,
t+b3)
1092
1093                             fes.add(dep)
1094                             c3.depEvent = dep      #ticket knows its
departure and service time
1095
1096                             serv.idle = False
1097                             serv.ticket = c3
1098                             c3.testster = serv
1099                             c3.state = 4      #goes into implemented
state
1100
1101                             c3.pos = 2
1102                             serv.countImplt += 1
1103                             serv.setWorking(t)
1104
1105                 elif countSch(serv) > 0:
1106                     if serv.countSch < 2:
1107                         q_ = queueSch(serv)
1108                         c3 = q_[0]
1109                         serv.queue.dequeue_ticket(c3)

```

```

1104         b3 = self.servDist[3].rvs()
1105         dep = Event(Event.DEPARTURE, serv, c3, t+b3
)
1106         fes.add(dep)
1107         c3.depEvent = dep      #ticket knows its
departure and service time
1108         serv.idle = False
1109         serv.ticket = c3
1110         c3.state = 3      #goes into scheduled state
1111         c3.pos = 4
1112         serv.countSch += 1
1113         serv.setWorking(t)
1114
1115
1116         #ALGORITHM 10 (DEPARTURE FROM IMPLEMENTED)
1117         elif queueNr == 2:
1118
1119             if c.testProb > test_p:      #
testing negative
1120                 fes.add(Event(Event.ARRIVAL, c.engineer, c, t)) #
schedule arrival at engineer's queue - scheduled for the engineer again.
1121                 res.registerWaitingTime (4, t - c.arrivalTime)
1122                 sojournTimes_Components.loc[len(sojournTimes_Components
.index), c.component] = (t-c.arrivalTime)/3600
1123                 c.newPos(3, t)
1124                 c.pos = 4
1125                 #break
1126
1127             else:      #testing
positive
1128                 fes.add(Event(Event.ARRIVAL, s1, c, t)) #schedule
arrival at for S1 queue - goes to verified.
1129                 res.registerWaitingTime (4, t - c.arrivalTime)
1130                 sojournTimes_Components.loc[len(sojournTimes_Components
.index), c.component] = (t-c.arrivalTime)/3600
1131                 c.newPos(5, t)
1132                 c.pos = 3
1133
1134                 # Once server stops working, set to idle
1135                 c.testster.idle = True
1136                 c.testster.setIdle(t)
1137
1138                 # Workflows
1139                 if c.testster == tester:
1140                     c3 = searchPriorityBlocking(qs[2])
1141                     c4 = searchPriorityUrgent(qs[2])
1142
1143                 # TESTER WORKFLOW
1144                 if c3 != None:
1145                     if tester.idle == True:
1146                         b3 = self.servDist[4].rvs()
1147                         dep = Event(Event.DEPARTURE, tester, c3, t+b3)
1148                         fes.add(dep)
1149                         c3.depEvent = dep      #ticket knows its
departure and service time
1150                         tester.idle = False
1151                         c3.testster = tester
1152                         tester.ticket = c3
1153                         c3.state = 4
1154                         c3.pos = 2
1155                         tester.setWorking(t)
1156
1157                     else:
1158                         if tester.ticket.priority != 0:
1159                             tester.ticket.depEvent.iscancelled = True
1160                             tester.ticket.depEvent = None

```

```

1161         tester.nrTickets.remove(tester.ticket.
number)
1162         tester.nrTickets.append(c3.number)
1163         b3 = self.servDist[4].rvs()
1164         dep = Event(Event.DEPARTURE, tester, c3, t+
b3)
1165         fes.add(dep)
1166         c3.depEvent = dep      #ticket knows its
departure and service time
1167         tester.idle = False
1168         c3.testers = tester
1169         tester.ticket = c3
1170         c3.state = 4
1171         c3.pos = 2
1172
1173
1174         elif c4 != None:
1175             if tester.idle == True:
1176                 b3 = self.servDist[4].rvs()
1177                 dep = Event(Event.DEPARTURE, tester, c4, t+b3)
1178                 fes.add(dep)
1179                 c4.depEvent = dep      #ticket knows its
departure and service time
1180                 tester.idle = False
1181                 tester.ticket = c4
1182                 c4.testers = tester
1183                 c4.state = 4
1184                 c4.pos = 2
1185                 tester.setWorking(t)
1186
1187             else:
1188                 if tester.ticket.priority != 0 and tester.
ticket.priority != 1:
1189                     tester.ticket.depEvent.iscancelled = True
1190                     tester.ticket.depEvent = None
1191                     tester.nrTickets.remove(tester.ticket.
number)
1192                     tester.nrTickets.append(c4.number)
1193                     b3 = self.servDist[4].rvs()
1194                     dep = Event(Event.DEPARTURE, tester, c4, t+
b3)
1195                     fes.add(dep)
1196                     c4.depEvent = dep      #ticket knows its
departure and service time
1197                     tester.idle = False
1198                     c4.testers = tester
1199                     tester.ticket = c4
1200                     c4.state = 4
1201                     c4.pos = 2
1202
1203                 elif len(qs[2].tickets) > 0 and tester.idle == True:
1204                     c3 = qs[2].dequeue_0()
1205                     b3 = self.servDist[4].rvs()
1206                     dep = Event(Event.DEPARTURE, tester, c3, t+b3)
1207                     fes.add(dep)
1208                     c3.depEvent = dep      #ticket knows its departure
and service time
1209                     tester.ticket = c3
1210                     tester.idle = False
1211                     c3.testers = tester
1212                     c3.state = 4
1213                     c3.pos = 2
1214                     tester.setWorking(t)
1215
1216
1217                 elif c.testers in engg:
1218                     serv = c.testers

```

```

1219
1220     # Engineer's Workflow
1221     c3 = searchPriorityBlocking(serv.queue)
1222     c4 = searchPriorityUrgent(serv.queue)
1223     if c3 != None:
1224         if serv.idle == True:
1225             s = c3.state
1226             b3 = self.servDist[s].rvs()
1227             dep = Event(Event.DEPARTURE, serv, c3, t+b3)
1228             fes.add(dep)
1229             c3.depEvent = dep     #ticket knows its
departure and service time
1230             serv.idle = False
1231             serv.ticket = c3
1232             c3.state = s
1233             c3.pos = assignPos(c3)
1234             serv.setWorking(t)
1235
1236         else:
1237             if serv.ticket.priority != 0 and serv.ticket.
depEvent != None:
1238                 serv.ticket.depEvent.iscancelled = True
1239                 serv.ticket.depEvent = None
1240                 remove_state_ticket(serv)
1241                 serv.nrTickets.remove(serv.ticket.number)
1242                 serv.nrTickets.append(c3.number)
1243                 add_state_ticket(serv, c3)
1244                 s = c3.state
1245                 b3 = self.servDist[s].rvs()
1246                 dep = Event(Event.DEPARTURE, serv, c3, t+b3
)
1247                 fes.add(dep)
1248                 c3.depEvent = dep     #ticket knows its
departure and service time
1249                 serv.idle = False
1250                 serv.ticket = c3
1251                 c3.state = s
1252                 c3.pos = assignPos(c3)
1253
1254             elif c4 != None:
1255                 if serv.idle == True:
1256                     s = c4.state
1257                     b3 = self.servDist[s].rvs()
1258                     dep = Event(Event.DEPARTURE, serv, c4, t+b3)
1259                     fes.add(dep)
1260                     c4.depEvent = dep     #ticket knows its
departure and service time
1261                     serv.idle = False
1262                     serv.ticket = c4
1263                     c4.state = s
1264                     c4.pos = assignPos(c4)
1265                     serv.setWorking(t)
1266
1267                 else:
1268                     if serv.ticket.priority != 0 and serv.ticket.
priority != 1 and serv.ticket.depEvent != None:
1269                         serv.ticket.depEvent.iscancelled = True
1270                         serv.ticket.depEvent = None
1271                         remove_state_ticket(serv)
1272                         serv.nrTickets.remove(serv.ticket.number)
1273                         serv.nrTickets.append(c4.number)
1274                         add_state_ticket(serv, c4)
1275                         s = c4.state
1276                         b3 = self.servDist[s].rvs()
1277                         dep = Event(Event.DEPARTURE, serv, c4, t+b3
)
1278                         fes.add(dep)

```



```

1279         c4.depEvent = dep      #ticket knows its
departure and service time
1280         serv.idle = False
1281         serv.ticket = c4
1282         c4.state = s
1283         c4.pos = assignPos(c4)
1284
1285
1286     else:
1287         if serv.idle == True:
1288             if countAccepted(serv) > 0:
1289                 if serv.countAccept < 2:
1290                     q_ = queueAccepted(serv)
1291                     c3 = q_[0]
1292                     serv.queue.dequeue_ticket(c3)
1293                     b3 = self.servDist[1].rvs()
1294                     dep = Event(Event.DEPARTURE, serv, c3,
t+b3)
1295
1296                     fes.add(dep)
1297                     c3.depEvent = dep      #ticket knows its
departure and service time
1298
1299                     serv.idle = False
1300                     serv.ticket = c3
1301                     c3.state = 1      #goes into accepted
state
1302
1303                     c3.pos = 4
1304                     serv.countAccept += 1
1305                     serv.setWorking(t)
1306
1307             elif len(qs[2].tickets) > 0:
1308                 if serv.countImplt < 2:
1309                     q_ = queueImplt(qs[2], serv)
1310                     if len(q_) != 0:
1311                         c3 = q_[0]
1312                         qs[2].dequeue_ticket(c3)
1313                         b3 = self.servDist[4].rvs()
1314                         dep = Event(Event.DEPARTURE, serv,
c3, t+b3)
1315
1316                         fes.add(dep)
1317                         c3.depEvent = dep      #ticket knows
its departure and service time
1318
1319                         serv.idle = False
1320                         serv.ticket = c3
1321                         c3.testster = serv
1322                         c3.state = 4      #goes into
implemented state
1323
1324                         c3.pos = 2
1325                         serv.countImplt += 1
1326                         serv.setWorking(t)
1327
1328             elif countSch(serv) > 0:
1329                 if serv.countSch < 2:
1330                     q_ = queueSch(serv)
1331                     c3 = q_[0]
1332                     serv.queue.dequeue_ticket(c3)
1333                     b3 = self.servDist[3].rvs()
1334                     dep = Event(Event.DEPARTURE, serv, c3,
t+b3)
1335
1336                     fes.add(dep)
1337                     c3.depEvent = dep      #ticket knows its
departure and service time
1338
1339                     serv.idle = False
1340                     serv.ticket = c3
1341                     c3.state = 3      #goes into scheduled
state
1342
1343                     c3.pos = 4
1344                     serv.countSch += 1

```

```

1336         serv.setWorking(t)
1337
1338
1339     elif c.testter == s1:
1340
1341         # S1 Workflow
1342
1343         if s1.openCounter <= 5 and len(qs[0].tickets) > 0:
1344             c2 = qs[0].dequeue_0()
1345             b0 = self.servDist[0].rvs()
1346             dep = Event(Event.DEPARTURE, s1, c2, t+b0)
1347             c2.depEvent = dep      #ticket knows its departure
1348
1349         and service time
1350
1351         fes.add(dep)
1352         s1.idle = False
1353         s1.ticket = c2
1354         c2.server = s1
1355         c2.state = 0
1356         c2.pos = 0
1357         s1.setWorking(t)
1358         s1.openCounter += 1
1359
1360     else:
1361
1362         c1 = searchPriorityBlocking(qs[3])
1363         c3 = searchPriorityUrgent(qs[3])
1364
1365         if c1 != None:
1366
1367             if s1.idle == True:
1368                 b3 = self.servDist[5].rvs()
1369                 dep = Event(Event.DEPARTURE, s1, c1, t+b3)
1370                 fes.add(dep)
1371                 c1.depEvent = dep      #ticket knows its
1372
1373             departure and service time
1374
1375             s1.idle = False
1376             s1.ticket = c1
1377             c1.state = 5
1378             c1.pos = 3
1379             s1.setWorking(t)
1380
1381         else:
1382             if s1.ticket.priority != 0:
1383                 s1.ticket.depEvent.iscancelled = True
1384                 s1.ticket.depEvent = None
1385                 s1.nrTickets.remove(s1.ticket.number)
1386                 s1.nrTickets.append(c1.number)
1387                 b3 = self.servDist[5].rvs()
1388                 dep = Event(Event.DEPARTURE, s1, c1, t+
1389
1390             b3)
1391
1392             fes.add(dep)
1393             c1.depEvent = dep      #ticket knows its
1394
1395             departure and service time
1396
1397             s1.idle = False
1398             s1.ticket = c1
1399             c1.state = 5
1400             c1.pos = 3
1401
1402         elif c3 != None:
1403             if s1.idle == True:
1404                 b3 = self.servDist[5].rvs()
1405                 dep = Event(Event.DEPARTURE, s1, c3, t+b3)
1406                 fes.add(dep)
1407                 c3.depEvent = dep      #ticket knows its
1408
1409             departure and service time
1410
1411             s1.idle = False

```

```

1398         c3.state = 5
1399         c3.pos = 3
1400         s1.ticket = c3
1401         s1.setWorking(t)
1402
1403         else:
1404             if s1.ticket.priority != 0 and s1.ticket.
priority != 1 and serv.ticket.depEvent != None:
1405                 s1.ticket.depEvent.iscancelled = True
1406                 s1.ticket.depEvent = None
1407                 s1.nrTickets.remove(s1.ticket.number)
1408                 s1.nrTickets.append(c3.number)
1409                 b3 = self.servDist[5].rvs()
1410                 dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
1411
1412                 fes.add(dep)
1413                 c3.depEvent = dep      #ticket knows its
departure and service time
1414
1415                 s1.idle = False
1416                 c3.state = 5
1417                 c3.pos = 3
1418                 s1.ticket = c3
1419
1420             else:
1421                 if s1.idle == True:
1422                     if len(qs[3].tickets) > 0:
1423                         c3 = qs[3].dequeue_0()
1424                         b3 = self.servDist[5].rvs()
1425                         dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
1426
1427                         fes.add(dep)
1428                         c3.depEvent = dep      #ticket knows its
departure and service time
1429
1430                         s1.ticket = c3
1431                         c3.pos = 3
1432                         c3.state = 5
1433                         s1.idle = False
1434                         s1.setWorking(t)
1435
1436                     elif len(qs[2].tickets) > 0:
1437                         if s1.impltCounter < 2:
1438                             c3 = qs[2].dequeue_0()      #ticket is
not being served
1439
1440                             b3 = self.servDist[4].rvs()
1441                             dep = Event(Event.DEPARTURE, s1, c3
, t+b3)
1442
1443                             fes.add(dep)
1444                             c3.depEvent = dep      #ticket knows
its departure and service time
1445
1446                             s1.ticket = c3
1447                             c3.testor = s1
1448                             s1.idle = False
1449                             c3.state = 4      #goes into
implemented state
1450
1451                             c3.pos = 2
1452                             s1.impltCounter += 1
1453                             s1.setWorking(t)
1454
1455                     elif len(qs[1].tickets) > 0:
1456
1457                         if s1.analyseCounter < 4:
1458                             c3 = qs[1].dequeue_0()
1459                             b3 = self.servDist[2].rvs()
1460                             dep = Event(Event.DEPARTURE, s1, c3
, t+b3)

```

```

1455         fes.add(dep)
1456         c3.depEvent = dep      #ticket knows
its departure and service time
1457         s1.ticket = c3
1458         c3.state = 2      #goes into analysed
state
1459         c3.pos = 1
1460         s1.idle = False
1461         s1.analyseCounter += 1
1462         s1.setWorking(t)
1463
1464         elif len(qs[2].tickets) > 0 and len(qs[2].
tickets) < 20:
1465             if s1.impltCounter < 2:
1466                 c3 = qs[2].dequeue_0()
1467                 b3 = self.servDist[4].rvs()
1468                 dep = Event(Event.DEPARTURE, s1, c3
, t+b3)
1469                 fes.add(dep)
1470                 c3.depEvent = dep      #ticket knows
its departure and service time
1471                 s1.idle = False
1472                 s1.ticket = c3
1473                 c3.testner = s1
1474                 c3.state = 4      #goes into
implemented state
1475                 c3.pos = 2
1476                 s1.impltCounter += 1
1477                 s1.setWorking(t)
1478
1479
1480         #ALGORITHM 12 (DEPARTURE FROM VERIFIED)
1481         elif queueNr == 3:
1482             # print("ALGORITHM 12")
1483             res.registerWaitingTime (5, t - c.arrivalTime)
1484             c.leaveSystem(t)
1485             res.registerSojournTime(t - c.systemArrivalTime)
1486             storeSojournTimes_priorities(c, (t-c.systemArrivalTime))
1487
1488             # Different results
1489             sojournTimes_Components.loc[len(sojournTimes_Components.
index), c.component] = (t-c.arrivalTime)
1490             totalTimes_Components.loc[len(totalTimes_Components.index),
c.component] = (t-c.systemArrivalTime)
1491
1492             TotalTicketsNr.append(c.number)
1493             tickets_in_system_day.append(c.number)
1494
1495             # Once server stops working, set to idle
1496             s1.idle = True
1497             s1.setIdle(t)
1498
1499
1500             # S1 WORKFLOW
1501
1502             if s1.openCounter <= 5 and len(qs[0].tickets) > 0:
1503
1504                 c2 = qs[0].dequeue_0()
1505                 b0 = self.servDist[0].rvs()
1506                 dep = Event(Event.DEPARTURE, s1, c2, t+b0)
1507                 c2.depEvent = dep      #ticket knows its departure and
service time
1508                 fes.add(dep)
1509                 s1.idle = False
1510                 s1.ticket = c2
1511                 c2.server = s1
1512                 c2.state = 0

```

```

1513         c2.pos = 0
1514         s1.openCounter += 1
1515         s1.setWorking(t)
1516
1517     else:
1518
1519         c1 = searchPriorityBlocking(qs[3])
1520         c3 = searchPriorityUrgent(qs[3])
1521
1522         if c1 != None:
1523
1524             if s1.idle == True:
1525                 b3 = self.servDist[5].rvs()
1526                 dep = Event(Event.DEPARTURE, s1, c1, t+b3)
1527                 fes.add(dep)
1528                 c1.depEvent = dep      #ticket knows its
departure and service time
1529
1530                 s1.idle = False
1531                 s1.ticket = c1
1532                 c1.state = 5
1533                 c1.pos = 3
1534                 s1.setWorking(t)
1535
1536             else:
1537                 if s1.ticket.priority != 0:
1538                     s1.ticket.depEvent.iscancelled = True
1539                     s1.ticket.depEvent = None
1540                     s1.nrTickets.remove(s1.ticket.number)
1541                     s1.nrTickets.append(c1.number)
1542                     b3 = self.servDist[5].rvs()
1543                     dep = Event(Event.DEPARTURE, s1, c1, t+b3)
1544                     fes.add(dep)
1545                     c1.depEvent = dep      #ticket knows its
departure and service time
1546
1547                     s1.idle = False
1548                     s1.ticket = c1
1549                     c1.state = 5
1550                     c1.pos = 3
1551
1552         elif c3 != None:
1553             if s1.idle == True:
1554                 b3 = self.servDist[5].rvs()
1555                 dep = Event(Event.DEPARTURE, s1, c3, t+b3)
1556                 fes.add(dep)
1557                 c3.depEvent = dep      #ticket knows its
departure and service time
1558
1559                 s1.idle = False
1560                 c3.state = 5
1561                 c3.pos = 3
1562                 s1.ticket = c3
1563                 s1.setWorking(t)
1564
1565             else:
1566                 if s1.ticket.priority != 0 and s1.ticket.
priority != 1:
1567
1568                     s1.ticket.depEvent.iscancelled = True
1569                     s1.ticket.depEvent = None
1570                     s1.nrTickets.remove(s1.ticket.number)
1571                     s1.nrTickets.append(c3.number)
1572                     b3 = self.servDist[5].rvs()
1573                     dep = Event(Event.DEPARTURE, s1, c3, t+b3)
1574                     fes.add(dep)
1575                     c3.depEvent = dep      #ticket knows its
departure and service time
1576
1577                     s1.idle = False
1578                     c3.state = 5

```

```

1575         c3.pos = 3
1576         s1.ticket = c3
1577
1578
1579     else:
1580         if s1.idle == True:
1581
1582             if len(qs[3].tickets) > 0:
1583                 c3 = qs[3].dequeue_0()
1584                 b3 = self.servDist[5].rvs()
1585                 dep = Event(Event.DEPARTURE, s1, c3, t+b3)
1586                 fes.add(dep)
1587                 c3.depEvent = dep      #ticket knows its
departure and service time
1588
1589                 s1.ticket = c3
1590                 c3.pos = 3
1591                 c3.state = 5
1592                 s1.idle = False
1593                 s1.setWorking(t)
1594
1595             elif len(qs[2].tickets) > 0:
1596                 if s1.impltCounter < 2:
1597                     c3 = qs[2].dequeue_0()      #ticket is not
being served
1598
1599                     b3 = self.servDist[4].rvs()
1600                     dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
1601
1602                     fes.add(dep)
1603                     c3.depEvent = dep      #ticket knows its
departure and service time
1604
1605                     s1.ticket = c3
1606                     c3.testster = s1
1607                     s1.idle = False
1608                     c3.state = 4      #goes into implemented
state
1609
1610                     c3.pos = 2
1611                     s1.impltCounter += 1
1612                     s1.setWorking(t)
1613
1614             elif len(qs[1].tickets) > 0:
1615                 if s1.analyseCounter < 4:
1616                     c3 = qs[1].dequeue_0()
1617                     b3 = self.servDist[2].rvs()
1618                     dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
1619
1620                     fes.add(dep)
1621                     c3.depEvent = dep      #ticket knows its
departure and service time
1622
1623                     s1.ticket = c3
1624                     c3.state = 2      #goes into analysed
state
1625
1626                     c3.pos = 1
1627                     s1.idle = False
1628                     s1.analyseCounter += 1
1629                     s1.setWorking(t)
1630
1631             elif len(qs[2].tickets) > 0 and len(qs[2].
tickets) < 20:
1632                 if s1.impltCounter < 2:
1633                     c3 = qs[2].dequeue_0()
1634                     b3 = self.servDist[4].rvs()
1635                     dep = Event(Event.DEPARTURE, s1, c3, t+
b3)
1636
1637                     fes.add(dep)

```

```

1631         c3.depEvent = dep      #ticket knows its
departure and service time
1632         s1.idle = False
1633         s1.ticket = c3
1634         c3.testor = s1
1635         c3.state = 4      #goes into implemented
state
1636         c3.pos = 2
1637         s1.impltCounter += 1
1638         s1.setWorking(t)
1639
1640
1641     elif e.type == Event.EOD:
1642
1643         # Resetting all counters for the day
1644         s1.openCounter = 0
1645         s1.impltCounter = 0
1646         s1.analyseCounter = 0
1647
1648         for i in engg:
1649             serv = i
1650             serv.countAccept = 0
1651             serv.countImplt = 0
1652             serv.countSch = 0
1653
1654         # Storing results for the day
1655
1656         OpenTickets.append(len(qs[0].tickets))
1657         AcceptedTickets.append(countAccepted(engg[0]) + countAccepted(engg
[1]) + countAccepted(engg[2]) + countAccepted(engg[3]) + countAccepted(engg[4])
+ countAccepted(engg[5]) + countAccepted(engg[6]))
1658         AnalysedTickets.append(len(qs[1].tickets))
1659         ScheduledTickets.append(countSch(engg[0]) + countSch(engg[1]) +
countSch(engg[2]) + countSch(engg[3]) + countSch(engg[4]) + countSch(engg[5]) +
countSch(engg[6]))
1660         ImplementedTickets.append(len(qs[2].tickets))
1661         VerifiedTickets.append(len(qs[3].tickets))
1662
1663         engg_1.append(len(engg[0].queue.tickets))
1664         engg_2.append(len(engg[1].queue.tickets))
1665         engg_3.append(len(engg[2].queue.tickets))
1666         engg_4.append(len(engg[3].queue.tickets))
1667         engg_5.append(len(engg[4].queue.tickets))
1668         engg_6.append(len(engg[5].queue.tickets))
1669         engg_7.append(len(engg[6].queue.tickets))
1670         s1_tickets.append(len(qs[0].tickets) + len(qs[1].tickets) + len(qs
[3].tickets))
1671         tester_tickets.append(len(qs[2].tickets))
1672
1673         # Opening of tickets at the start of the day
1674
1675         if s1.ticket == None:
1676             if len(qs[0].tickets) > 0:
1677                 c2 = qs[0].dequeue_0()
1678                 b0 = self.servDist[0].rvs()
1679                 dep = Event(Event.DEPARTURE, s1, c2, t+b0)
1680                 c2.depEvent = dep      #ticket knows its departure and
service time
1681                 fes.add(dep)
1682                 s1.idle = False
1683                 s1.ticket = c2
1684                 c2.server = s1
1685                 c2.state = 0
1686                 c2.pos = 0
1687                 s1.openCounter += 1
1688                 s1.setWorking(t)
1689

```

```

1690         else:
1691             c2 = s1.ticket
1692             if c2 != None and c2.state != 0 and c2.depEvent != None and c2.
priority != 0:
1693                 if len(qs[0].tickets) > 0:
1694                     c2.depEvent.iscancelled = True
1695                     c2.depEvent = None
1696                     s1.nrTickets.remove(s1.ticket.number)
1697                     s1.nrTickets.append(c.number)
1698                     c = qs[0].dequeue_0()
1699                     b2 = self.servDist[0].rvs()
1700                     dep = Event(Event.DEPARTURE, s1, c, t+b2)
1701                     c.depEvent = dep #ticket knows its departure and
service time
1702                     fes.add(dep)
1703                     s1.idle = False
1704                     s1.ticket = c
1705                     c.state = 0
1706                     c.pos = 0
1707                     s1.openCounter += 1
1708
1709                     fes.add(Event(Event.EOD, s1, c, t+8*3600))
1710                     nrDays_vs_Tickets.loc[day, "Tickets"] = len(set(
tickets_in_system_day))
1711                     day = day + 1
1712                     tickets_in_system_day = []
1713
1714             return res
1715
1716
1717 #arrival distribution seconds in per ticket
1718 servDist = [] # manually create them # service times
1719
1720 arrDist = Distribution(stats.expon(scale = ((365*5*3600*8)/8492))) #arrival time
1721
1722 servDist.append(Distribution(stats.gamma(scale = 1/((2*60)/5580000), a = ((2*60)
**2/(5580000)))) # open
1723 servDist.append(Distribution(stats.gamma(scale = 1/((30*3600/90000000)), a =
((30*3600)**2/90000000)))) # accepted
1724 servDist.append(Distribution(stats.gamma(scale = 1/((20*60)/(900000)), a = (20*60)
**2/(900000)))) # analysed
1725 servDist.append(Distribution(stats.gamma(scale = 1/((20*3600)/337000000), a =
(20*3600)**2/(337000000)))) # scheduled
1726 servDist.append(Distribution(stats.gamma(scale = 1/((10*3600)/2600000000), a =
(10*3600)**2/(2600000000)))) # implemented
1727 servDist.append(Distribution(stats.gamma(scale = 1/((10*60)/8500000), a = (10*60)
**2/(8500000)))) # verified
1728
1729 years = 5 # number of years for which the simulation needs to run
1730 run = 10 # index of the current run
1731
1732 r = 365*years # in days
1733
1734 sim = Simulation(arrDist , servDist , 8)
1735 res = sim.simulate(r*8*3600) # input in hours
1736 bins = 100
1737
1738 ##### FOR RESULTS #####
1739
1740 # Printing results
1741
1742 for i in range(6):
1743
1744     if i == 0:
1745         print("#### RESULTS FOR OPEN QUEUE ####")
1746         print(res.queueResults[i])
1747         res.queueResults[i].histQueueLength(i,10)

```



```

1748         res.queueResults[i].histWaitingTimes(i,100)
1749
1750     elif i == 1:
1751         print("##### RESULTS FOR ACCEPTED QUEUE #####")
1752         print(res.queueResults[i])
1753         res.queueResults[i].histQueueLength(i,1000)
1754         res.queueResults[i].histWaitingTimes(i,50)
1755
1756     elif i == 2:
1757         print("##### RESULTS FOR ANALYSED QUEUE #####")
1758         print(res.queueResults[i])
1759         res.queueResults[i].histQueueLength(i,10)
1760         res.queueResults[i].histWaitingTimes(i,50)
1761
1762     elif i == 3:
1763         print("##### RESULTS FOR SCHEDULED QUEUE #####")
1764         print(res.queueResults[i])
1765         res.queueResults[i].histQueueLength(i,500)
1766         res.queueResults[i].histWaitingTimes(i,100)
1767
1768     elif i == 4:
1769         print("##### RESULTS FOR IMPLEMENTED QUEUE #####")
1770         print(res.queueResults[i])
1771         res.queueResults[i].histQueueLength(i,10)
1772         res.queueResults[i].histWaitingTimes(i,100)
1773
1774
1775     elif i == 5:
1776         print("##### RESULTS FOR VERIFIED QUEUE #####")
1777         print(res.queueResults[i])
1778         res.queueResults[i].histQueueLength(i,10)
1779         res.queueResults[i].histWaitingTimes(i,0.5)
1780
1781     print("Mean Sojourn time: (in hours) ", mean(res.sojournTimes)/3600)
1782     res.histSojournTimes(50)
1783
1784     print("\nS1 Working time: (in hours) ", s1.workTime/3600, " (", s1.workTime
1785         *100/(3600*8*years*365), "%)", " and Idle time: (in hours) ", s1.idleTime/3600,
1786         " (", s1.idleTime*100/(3600*8*years*365), "%)", " and number of tickets worked
1787         on - ", len(set(s1.nrTickets)))
1788     sojournTimes_Servers.loc[len(sojournTimes_Servers.index), s1] = s1.workTime/3600
1789     sojournTimes_Servers_idle.loc[len(sojournTimes_Servers_idle.index), s1] = s1.
1790         idleTime/3600
1791
1792     for i in engg:
1793         print("\n", i, " Working time: (in hours) ", i.workTime/3600, " (", i.workTime
1794             *100/(3600*8*years*365), "%)", " and Idle time: (in hours) ", i.idleTime/3600,
1795             " (", i.idleTime*100/(3600*8*years*365), "%)", " and number of tickets worked
1796             on - ", len(set(i.nrTickets)))
1797         print("Number of tickets worked on distribution - \nAccepted state tickets: ",
1798             len(set(i.acceptTickets)), " Schedule state tickets: ", len(set(i.schTickets)),
1799             " Implemented state tickets: ", len(set(i.impltTickets)))
1800         sojournTimes_Servers.loc[len(sojournTimes_Servers.index), i] = i.workTime/3600
1801         sojournTimes_Servers_idle.loc[len(sojournTimes_Servers_idle.index), i] = i.
1802             idleTime/3600
1803
1804     print("\nTester Working time: (in hours) ", tester.workTime/3600, " (", tester.
1805         workTime*100/(3600*8*years*365), "%)", " and Idle time: (in hours) ", tester.
1806         idleTime/3600, " (", tester.idleTime*100/(3600*8*years*365), "%)", " and number
1807         of tickets worked on - ", len(set(tester.nrTickets)))
1808     sojournTimes_Servers.loc[len(sojournTimes_Servers.index), tester] = tester.workTime
1809         /3600
1810     sojournTimes_Servers_idle.loc[len(sojournTimes_Servers_idle.index), tester] =
1811         tester.idleTime/3600
1812
1813     def max_work(servers):

```

```

1800     workTimes = []
1801     for i in servers:
1802         workTimes.append(i.workTime)
1803
1804     for i in servers:
1805         if max(workTimes) == i.workTime:
1806             return i
1807
1808 server = max_work([s1, tester]+engg)
1809 print("\nServer who works the most - ", server)
1810
1811 def max_idle(servers):
1812     idleTimes = []
1813     for i in servers:
1814         idleTimes.append(i.idleTime)
1815
1816     for i in servers:
1817         if max(idleTimes) == i.idleTime:
1818             return i
1819
1820 server = max_idle([s1, tester] + engg)
1821 print("\nServer who is the most idle - ", server)
1822
1823 Comp_Sojourn = sojournTimes_Components.mean(axis=0)
1824 plt.figure(figsize=(12, 6))
1825 Comp_Sojourn.plot(kind='bar')
1826 plt.xlabel('Components', fontsize = 12)
1827 plt.ylabel('Time Spent (in hours)', fontsize = 12)
1828 plt.title('Time Spent (Component-wise)', fontsize = 14)
1829 # show the plot
1830 plt.show()
1831
1832 serverss = sojournTimes_Servers.mean(axis=0)
1833 plt.figure(figsize=(12, 6))
1834 serverss.plot(kind='bar')
1835 plt.xlabel('Servers', fontsize = 12)
1836 plt.ylabel('Work time (in hours)', fontsize = 12)
1837 plt.title('Work time for each Server', fontsize = 14)
1838 # show the plot
1839 plt.show()
1840
1841 Comp_Total = totalTimes_Components.mean(axis=0)
1842 plt.figure(figsize=(12, 6))
1843 Comp_Total.plot(kind='bar')
1844 plt.xlabel('Components', fontsize = 12)
1845 plt.ylabel('Time Spent (in hours)', fontsize = 12)
1846 plt.title('Total Time in the System (Component-wise)', fontsize = 14)
1847
1848
1849 nrDays_vs_Tickets.plot(y = "Tickets", kind='line', figsize = (30,10), fontsize =
1850 16)
1851 plt.title("Number of Tickets per day", fontsize = 16)
1852 plt.xlabel("Day", fontsize = 16)
1853 plt.ylabel("Tickets", fontsize = 16)
1854 plt.show()
1855
1856 plt.plot(OpenTickets)
1857 plt.title("Number of Tickets per day - Open queue", fontsize = 14)
1858 plt.xlabel("Day", fontsize = 12)
1859 plt.ylabel("Tickets", fontsize = 12)
1860 plt.show()
1861 dy1 = np.gradient(OpenTickets)
1862 print("Rate of increase of tickets in Open state (per day):", mean(dy1))
1863
1864 plt.plot(AcceptedTickets)
1865 plt.title("Number of Tickets per day - Accepted queue", fontsize = 14)
1866 plt.xlabel("Day", fontsize = 12)

```

```

1866 plt.ylabel("Tickets", fontsize = 12)
1867 plt.show()
1868 dy2 = np.gradient(AcceptedTickets)
1869 print("Rate of increase in Accepted state (per day):", mean(dy2))
1870
1871 plt.plot(AnalysedTickets)
1872 plt.title("Number of Tickets per day - Analysed queue", fontsize = 14)
1873 plt.xlabel("Day", fontsize = 12)
1874 plt.ylabel("Tickets", fontsize = 12)
1875 plt.show()
1876 dy3 = np.gradient(AnalysedTickets)
1877 print("Rate of increase in Analysed state (per day):", mean(dy3))
1878
1879 plt.plot(ScheduledTickets)
1880 plt.title("Number of Tickets per day - Scheduled queue", fontsize = 14)
1881 plt.xlabel("Day", fontsize = 12)
1882 plt.ylabel("Tickets", fontsize = 12)
1883 plt.show()
1884 dy4 = np.gradient(ScheduledTickets)
1885 print("Rate of increase in Scheduled state (per day):", mean(dy4))
1886
1887 plt.plot(ImplementedTickets)
1888 plt.title("Number of Tickets per day - Implemented queue", fontsize = 14)
1889 plt.xlabel("Day", fontsize = 12)
1890 plt.ylabel("Tickets", fontsize = 12)
1891 plt.show()
1892 dy5 = np.gradient(ImplementedTickets)
1893 print("Rate of increase in Implemented state (per day):", mean(dy5))
1894
1895 plt.plot(VerifiedTickets)
1896 plt.title("Number of Tickets per day - Verified queue", fontsize = 14)
1897 plt.xlabel("Day", fontsize = 12)
1898 plt.ylabel("Tickets", fontsize = 12)
1899 plt.show()
1900 dy6 = np.gradient(VerifiedTickets)
1901 print("Rate of increase in Verified state (per day):", mean(dy6))
1902
1903 fig, ax = plt.subplots()
1904 ax.plot(engg_1, label='Engg 1')
1905 ax.plot(engg_2, label='Engg 2')
1906 ax.plot(engg_3, label='Engg 3')
1907 ax.plot(engg_4, label='Engg 4')
1908 ax.plot(engg_5, label='Engg 5')
1909 ax.plot(engg_6, label='Engg 6')
1910 ax.plot(engg_7, label='Engg 7')
1911 ax.set_xlabel('Days')
1912 ax.set_title('Engineer Queues')
1913 ax.set_ylabel('Tickets')
1914 ax.legend()
1915 plt.show()
1916
1917 dy_engg1 = np.gradient(engg_1)
1918 print("\n\nRate of increase Engg 1:", mean(dy_engg1))
1919 dy_engg2 = np.gradient(engg_2)
1920 print("Rate of increase Engg 2:", mean(dy_engg2))
1921 dy_engg3 = np.gradient(engg_3)
1922 print("Rate of increase Engg 3:", mean(dy_engg3))
1923 dy_engg4 = np.gradient(engg_4)
1924 print("Rate of increase Engg 4:", mean(dy_engg4))
1925 dy_engg5 = np.gradient(engg_5)
1926 print("Rate of increase Engg 5:", mean(dy_engg5))
1927 dy_engg6 = np.gradient(engg_6)
1928 print("Rate of increase Engg 6:", mean(dy_engg6))
1929 dy_engg7 = np.gradient(engg_7)
1930 print("Rate of increase Engg 7:", mean(dy_engg7))
1931

```

```

1932 print("\nMean number of tickets that go through the system in a day: ", mean(
      nrDays_vs_Tickets["Tickets"]))
1933
1934
1935 data = {
1936     'engg 1': mean(dy_engg1),
1937     'engg 2': mean(dy_engg2),
1938     'engg 3': mean(dy_engg3),
1939     'engg 4': mean(dy_engg4),
1940     'engg 5': mean(dy_engg5),
1941     'engg 6': mean(dy_engg6),
1942     'engg 7': mean(dy_engg7),
1943     'open': mean(dy1),
1944     'accepted': mean(dy2),
1945     'analysed': mean(dy3),
1946     'scheduled': mean(dy4),
1947     'implemented': mean(dy5),
1948     'verified': mean(dy6),
1949     'mean nr. of tickets through the system': mean(nrDays_vs_Tickets["Tickets"
    ])
1950 }
1951
1952 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
      prob 1/rate of increase.xlsx')
1953 df = df.append(data, ignore_index=True)
1954 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
      rate of increase.xlsx', index=False)
1955
1956 print("\nMean time spent by each priority in the system - ")
1957
1958 if len(priority_0) > 0:
1959     print(" Blocking '0' : ", mean(priority_0)/3600, " for ", len(set(
      count_priority_0)), " tickets")
1960
1961 if len(priority_1) > 0:
1962     print(" Urgent '1' : ", mean(priority_1)/3600, " for ", len(set(
      count_priority_1)), " tickets")
1963
1964 if len(priority_2) > 0:
1965     print(" High '2' : ", mean(priority_2)/3600, " for ", len(set(count_priority_2)
      ), " tickets")
1966
1967 if len(priority_3) > 0:
1968     print(" Normal '3' : ", mean(priority_3)/3600, " for ", len(set(
      count_priority_3)), " tickets")
1969
1970 if len(priority_4) > 0:
1971     print(" Low '4' : ", mean(priority_4)/3600, " for ", len(set(count_priority_4))
      , " tickets")
1972
1973 data = {
1974     'blocking - time': mean(priority_0)/3600,
1975     'blocking - nr. Of tickets': len(set(count_priority_0)),
1976     'urgent - time': mean(priority_1)/3600,
1977     'urgent - nr. Of tickets': len(set(count_priority_1)),
1978     'high - time': mean(priority_2)/3600,
1979     'high - nr. Of tickets': len(set(count_priority_2)),
1980     'normal - time': mean(priority_3)/3600,
1981     'normal - nr. Of tickets': len(set(count_priority_3)),
1982     'low - time': mean(priority_4)/3600,
1983     'low - nr. Of tickets': len(set(count_priority_4)),
1984 }
1985
1986 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
      prob 1/priority - time and tickets.xlsx')
1987 df = df.append(data, ignore_index=True)

```

```

1988 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
      priority - time and tickets.xlsx', index=False)
1989
1990 a = 1
1991 print("\nService times:")
1992 for i in servDist:
1993     print(" Queue ", a, ": ", np.mean(i.rvs(365*years)))
1994     a = a+1
1995
1996 # Storing data
1997
1998 data = {
1999     0: res.queueResults[0].getMeanWaitingTime(),
2000     1: res.queueResults[1].getMeanWaitingTime(),
2001     2: res.queueResults[2].getMeanWaitingTime(),
2002     3: res.queueResults[3].getMeanWaitingTime(),
2003     4: res.queueResults[4].getMeanWaitingTime(),
2004     5: res.queueResults[5].getMeanWaitingTime()
2005 }
2006
2007 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
      prob 1/waiting times.xlsx')
2008 df = df.append(data, ignore_index=True)
2009 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
      waiting times.xlsx', index=False)
2010
2011 data = {
2012     0: res.queueResults[0].getMeanQueueLength(),
2013     "0 - std dev": res.queueResults[0].getVarianceQueueLength() ** (1/2),
2014     1: res.queueResults[1].getMeanQueueLength(),
2015     "1 - std dev": res.queueResults[1].getVarianceQueueLength() ** (1/2),
2016     2: res.queueResults[2].getMeanQueueLength(),
2017     "2 - std dev": res.queueResults[2].getVarianceQueueLength() ** (1/2),
2018     3: res.queueResults[3].getMeanQueueLength(),
2019     "3 - std dev": res.queueResults[3].getVarianceQueueLength() ** (1/2),
2020     4: res.queueResults[4].getMeanQueueLength(),
2021     "4 - std dev": res.queueResults[4].getVarianceQueueLength() ** (1/2),
2022     5: res.queueResults[5].getMeanQueueLength(),
2023     "5 - std dev": res.queueResults[5].getVarianceQueueLength() ** (1/2)
2024 }
2025
2026 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
      prob 1/queue lengths.xlsx')
2027 df = df.append(data, ignore_index=True)
2028 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
      queue lengths.xlsx', index=False)
2029
2030 data = {
2031     'mean sojourn time': mean(res.sojournTimes)/3600,
2032     's1': s1.workTime/3600,
2033     's1 - tickets solved': len(set(s1.nrTickets)),
2034     1: engg[0].workTime/3600,
2035     '1 - tickets solved': len(set(engg[0].nrTickets)),
2036     2: engg[1].workTime/3600,
2037     '2 - tickets solved': len(set(engg[1].nrTickets)),
2038     3: engg[2].workTime/3600,
2039     '3 - tickets solved': len(set(engg[2].nrTickets)),
2040     4: engg[3].workTime/3600,
2041     '4 - tickets solved': len(set(engg[3].nrTickets)),
2042     5: engg[4].workTime/3600,
2043     '5 - tickets solved': len(set(engg[4].nrTickets)),
2044     6: engg[5].workTime/3600,
2045     '6 - tickets solved': len(set(engg[5].nrTickets)),
2046     7: engg[6].workTime/3600,
2047     '7 - tickets solved': len(set(engg[6].nrTickets)),
2048     'tester': tester.workTime/3600,
2049     'tester - tickets solved': len(set(tester.nrTickets))

```

```

2050     }
2051
2052 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
prob 1/sojourn times servers and mean.xlsx')
2053 df = df.append(data, ignore_index=True)
2054 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
sojourn times servers and mean.xlsx', index=False)
2055
2056 data = {
2057     's1': s1.idleTime/3600,
2058     1: engg[0].idleTime/3600,
2059     2: engg[1].idleTime/3600,
2060     3: engg[2].idleTime/3600,
2061     4: engg[3].idleTime/3600,
2062     5: engg[4].idleTime/3600,
2063     6: engg[5].idleTime/3600,
2064     7: engg[6].idleTime/3600,
2065     'tester': tester.idleTime/3600,
2066 }
2067
2068 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
prob 1/idle times.xlsx')
2069 df = df.append(data, ignore_index=True)
2070 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
idle times.xlsx', index=False)
2071
2072 data = {
2073     1: OpenTickets
2074 }
2075
2076 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
prob 1/eod queue lengths open.xlsx')
2077 df = df.append(data, ignore_index=True)
2078 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
eod queue lengths open.xlsx', index=False)
2079
2080 data = {
2081     1: AcceptedTickets
2082 }
2083
2084 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
prob 1/eod queue length accepted.xlsx')
2085 df = df.append(data, ignore_index=True)
2086 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
eod queue length accepted.xlsx', index=False)
2087
2088 data = {
2089     1: AnalysedTickets
2090 }
2091
2092 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
prob 1/eod queue lengths analysed.xlsx')
2093 df = df.append(data, ignore_index=True)
2094 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
eod queue lengths analysed.xlsx', index=False)
2095
2096 data = {
2097     1: ScheduledTickets
2098 }
2099
2100 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
prob 1/eod queue lengths scheduled.xlsx')
2101 df = df.append(data, ignore_index=True)
2102 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
eod queue lengths scheduled.xlsx', index=False)
2103
2104 data = {

```

```

2105         1: ImplementedTickets
2106     }
2107
2108 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
prob 1/eod queue lengths implemented.xlsx')
2109 df = df.append(data, ignore_index=True)
2110 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
eod queue lengths implemented.xlsx', index=False)
2111
2112 data = {
2113     1: VerifiedTickets
2114 }
2115
2116 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
prob 1/eod queue lengths verified.xlsx')
2117 df = df.append(data, ignore_index=True)
2118 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
eod queue lengths verified.xlsx', index=False)
2119
2120 data = {
2121     1: engg_1,
2122     2: engg_2,
2123     3: engg_3,
2124     4: engg_4,
2125     5: engg_5,
2126     6: engg_6,
2127     7: engg_7
2128 }
2129
2130 df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
prob 1/eod queue length engineers.xlsx')
2131 df = df.append(data, ignore_index=True)
2132 df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
eod queue length engineers.xlsx', index=False)

```

## A.8 Printing Aggregate results for multiple runs

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Wed Mar 01 19:17:25 2023
4
5  @author: 20181301
6  """
7
8
9  import csv
10 import pandas as pd
11 import numpy as np
12 from scipy.stats import t
13 import math
14 import matplotlib.pyplot as plt
15 import seaborn as sns
16 import ast
17
18 def calc_CI(m):
19     n = len(m)
20     X = np.mean(m)
21     s = np.std(m, ddof=1)
22     confidence_level = 0.95
23     dof = n - 1
24     t_value = t.ppf((1 + confidence_level) / 2, dof)
25     error = t_value * (s / np.sqrt(n))
26     l_bound = X - error
27     u_bound = X + error
28     return (l_bound, u_bound)

```

```

29
30
31 print("##### WAITING TIMES #####")
32 waitingTimes_states = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/
BFP/Simulation/prob 1/waiting times.xlsx')
33 for i in range(0,6):
34     if i == 0:
35         print("##### Open State #####")
36     elif i == 1:
37         print("##### Accepted State #####")
38     elif i == 2:
39         print("##### Analysed State #####")
40     elif i == 3:
41         print("##### Scheduled State #####")
42     elif i == 4:
43         print("##### Implemented State #####")
44     elif i == 5:
45         print("##### Verified State #####")
46     print("\n",waitingTimes_states[i].describe())
47     print("\nConfidence interval: ", calc_CI(waitingTimes_states[i]), "\n")
48
49 print("##### QUEUE LENGTHS #####")
50 queueLengths_states = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/
BFP/Simulation/prob 1/queue lengths.xlsx')
51 for i in range(0,6):
52     if i == 0:
53         print("##### Open State #####")
54         print("\n",queueLengths_states[i].describe())
55         print("\nFor standard dev.:",queueLengths_states["0 - std dev"].describe())
56         print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
57     elif i == 1:
58         print("##### Accepted State #####")
59         print("\n",queueLengths_states[i].describe())
60         print("\nFor standard dev.:",queueLengths_states["1 - std dev"].describe())
61         print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
62     elif i == 2:
63         print("##### Analysed State #####")
64         print("\n",queueLengths_states[i].describe())
65         print("\nFor standard dev.:",queueLengths_states["2 - std dev"].describe())
66         print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
67     elif i == 3:
68         print("##### Scheduled State #####")
69         print("\n",queueLengths_states[i].describe())
70         print("\nFor standard dev.:",queueLengths_states["3 - std dev"].describe())
71         print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
72     elif i == 4:
73         print("##### Implemented State #####")
74         print("\n",queueLengths_states[i].describe())
75         print("\nFor standard dev.:",queueLengths_states["4 - std dev"].describe())
76         print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
77     elif i == 5:
78         print("##### Verified State #####")
79         print("\n",queueLengths_states[i].describe())
80         print("\nFor standard dev.:",queueLengths_states["5 - std dev"].describe())
81         print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
82
83
84 print("##### SOJOURN TIMES, WORK TIMES & IDLE TIMES #####")
85 work_sojourn_times = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/
BFP/Simulation/prob 1/sojourn times servers and mean.xlsx')
86 idle_sojourn_times = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/
BFP/Simulation/prob 1/idle times.xlsx')
87
88 total_time = 5*365*8
89
90 print("\nMean Sojourn Time: ", np.mean(work_sojourn_times["mean sojourn time"]), "
and CI: ", calc_CI(work_sojourn_times["mean sojourn time"]))

```



```

91 print("\nS1 work time: ", np.mean(work_sojourn_times["s1"]), "(", np.mean(
    work_sojourn_times["s1"]*100/total_time, "%) for tickets: ", np.mean(
    work_sojourn_times["s1 - tickets solved"]))
92 print("S1 idle time: ", np.mean(idle_sojourn_times["s1"]), "(", np.mean(
    idle_sojourn_times["s1"]*100/total_time, "%)")
93
94 print("\nEngg 1 work time: ", np.mean(work_sojourn_times[1]), "(", np.mean(
    work_sojourn_times[1]*100/total_time, "%) for tickets: ", np.mean(
    work_sojourn_times["1 - tickets solved"]))
95 print("Engg 1 idle time: ", np.mean(idle_sojourn_times[1]), "(", np.mean(
    idle_sojourn_times[1]*100/total_time, "%)")
96
97 print("\nEngg 2 work time: ", np.mean(work_sojourn_times[2]), "(", np.mean(
    work_sojourn_times[2]*100/total_time, "%) for tickets: ", np.mean(
    work_sojourn_times["2 - tickets solved"]))
98 print("Engg 2 idle time: ", np.mean(idle_sojourn_times[2]), "(", np.mean(
    idle_sojourn_times[2]*100/total_time, "%)")
99
100 print("\nEngg 3 work time: ", np.mean(work_sojourn_times[3]), "(", np.mean(
    work_sojourn_times[3]*100/total_time, "%) for tickets: ", np.mean(
    work_sojourn_times["3 - tickets solved"]))
101 print("Engg 3 idle time: ", np.mean(idle_sojourn_times[3]), "(", np.mean(
    idle_sojourn_times[3]*100/total_time, "%)")
102
103 print("\nEngg 4 work time: ", np.mean(work_sojourn_times[4]), "(", np.mean(
    work_sojourn_times[4]*100/total_time, "%) for tickets: ", np.mean(
    work_sojourn_times["4 - tickets solved"]))
104 print("Engg 4 idle time: ", np.mean(idle_sojourn_times[4]), "(", np.mean(
    idle_sojourn_times[4]*100/total_time, "%)")
105
106 print("\nEngg 5 work time: ", np.mean(work_sojourn_times[5]), "(", np.mean(
    work_sojourn_times[5]*100/total_time, "%) for tickets: ", np.mean(
    work_sojourn_times["5 - tickets solved"]))
107 print("Engg 5 idle time: ", np.mean(idle_sojourn_times[5]), "(", np.mean(
    idle_sojourn_times[5]*100/total_time, "%)")
108
109 print("\nEngg 6 work time: ", np.mean(work_sojourn_times[6]), "(", np.mean(
    work_sojourn_times[6]*100/total_time, "%) for tickets: ", np.mean(
    work_sojourn_times["6 - tickets solved"]))
110 print("Engg 6 idle time: ", np.mean(idle_sojourn_times[6]), "(", np.mean(
    idle_sojourn_times[6]*100/total_time, "%)")
111
112 print("\nEngg 7 work time: ", np.mean(work_sojourn_times[7]), "(", np.mean(
    work_sojourn_times[7]*100/total_time, "%) for tickets: ", np.mean(
    work_sojourn_times["7 - tickets solved"]))
113 print("Engg 7 idletime: ", np.mean(idle_sojourn_times[7]), "(", np.mean(
    idle_sojourn_times[7]*100/total_time, "%)")
114
115 print("\nTester work time: ", np.mean(work_sojourn_times["tester"]), "(", np.mean(
    work_sojourn_times["tester"]*100/total_time, "%) for tickets: ", np.mean(
    work_sojourn_times["tester - tickets solved"]))
116 print("Tester idle time: ", np.mean(idle_sojourn_times["tester"]), "(", np.mean(
    idle_sojourn_times["tester"]*100/total_time, "%)")
117
118 print("\n##### RATE OF INCREASE OF QUEUES #####")
119 rates_of_increase = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/
    BFP/Simulation/prob 1/rate of increase.xlsx')
120 print("\nOpen state: ", np.mean(rates_of_increase["open"]))
121 print("Accepted state: ", np.mean(rates_of_increase["accepted"]))
122 print("Analysed state: ", np.mean(rates_of_increase["analysed"]))
123 print("Scheduled state: ", np.mean(rates_of_increase["scheduled"]))
124 print("Implemented state: ", np.mean(rates_of_increase["implemented"]))
125 print("Verified state: ", np.mean(rates_of_increase["verified"]))
126 print("Engg 1: ", np.mean(rates_of_increase["engg 1"]))
127 print("Engg 2: ", np.mean(rates_of_increase["engg 2"]))
128 print("Engg 3: ", np.mean(rates_of_increase["engg 3"]))
129 print("Engg 4: ", np.mean(rates_of_increase["engg 4"]))

```

```

130 print("Engg 5: ", np.mean(rates_of_increase["engg 5"]))
131 print("Engg 6: ", np.mean(rates_of_increase["engg 6"]))
132 print("Engg 7: ", np.mean(rates_of_increase["engg 7"]))
133 print("Average number of tickets going through the system (per day): ", np.mean(
    rates_of_increase["mean nr. of tickets through the system"]))
134
135 print("\n##### PRIORITY TICKETS #####")
136 priority = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/priority - time and tickets.xlsx')
137 print("\nBlocking: average time spent in system : ", np.mean(priority["blocking -
    time"]), " for ", np.mean(priority["blocking - nr. Of tickets"]))
138 print("Urgent: average time spent in system : ", np.mean(priority["urgent - time"])
    , " for ", np.mean(priority["urgent - nr. Of tickets"]))
139 print("High: average time spent in system : ", np.mean(priority["high - time"]), "
    for ", np.mean(priority["high - nr. Of tickets"]))
140 print("Normal: average time spent in system : ", np.mean(priority["normal - time"])
    , " for ", np.mean(priority["normal - nr. Of tickets"]))
141 print("Low: average time spent in system : ", np.mean(priority["low - time"]), "
    for ", np.mean(priority["low - nr. Of tickets"]))
142
143
144 ### Plotting EOD queues
145
146 def return_avg_eod_queue(df):
147     result = []
148     for i in range(df.shape[0]):
149         for j in range(df.shape[1]):
150             cell_list = df.iloc[i, j]
151             cell = ast.literal_eval(cell_list)
152             if result == []:
153                 result = [0]*len(cell)
154             for i in range(len(cell)):
155                 result[i] = result[i] + cell[i]
156
157     n = len(df)
158     result = [r/n for r in result ]
159     return result
160
161 OpenTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue lengths open.xlsx')
162 AcceptedTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue length accepted.xlsx')
163 AnalysedTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue lengths analysed.xlsx')
164 ScheduledTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue lengths scheduled.xlsx')
165 ImplementedTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/
    BFP/Simulation/prob 1/eod queue lengths implemented.xlsx')
166 VerifiedTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue lengths verified.xlsx')
167
168 engineers = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue length engineers.xlsx')
169 engg_1 = pd.DataFrame(engineers[1])
170 engg_2 = pd.DataFrame(engineers[2])
171 engg_3 = pd.DataFrame(engineers[3])
172 engg_4 = pd.DataFrame(engineers[4])
173 engg_5 = pd.DataFrame(engineers[5])
174 engg_6 = pd.DataFrame(engineers[6])
175 engg_7 = pd.DataFrame(engineers[7])
176
177 plt.plot(return_avg_eod_queue(OpenTickets))
178 plt.title("Number of Tickets per day - Open queue", fontsize = 14)
179 plt.xlabel("Day", fontsize = 12)
180 plt.ylabel("Tickets", fontsize = 12)
181 plt.show()
182 plt.plot(return_avg_eod_queue(AcceptedTickets))

```

```

183 plt.title("Number of Tickets per day - Accepted queue", fontsize = 14)
184 plt.xlabel("Day", fontsize = 12)
185 plt.ylabel("Tickets", fontsize = 12)
186 plt.show()
187
188 plt.plot(return_avg_eod_queue(AnalysedTickets))
189 plt.title("Number of Tickets per day - Analysed queue", fontsize = 14)
190 plt.xlabel("Day", fontsize = 12)
191 plt.ylabel("Tickets", fontsize = 12)
192 plt.show()
193
194 plt.plot(return_avg_eod_queue(ScheduledTickets))
195 plt.title("Number of Tickets per day - Scheduled queue", fontsize = 14)
196 plt.xlabel("Day", fontsize = 12)
197 plt.ylabel("Tickets", fontsize = 12)
198 plt.show()
199
200 plt.plot(return_avg_eod_queue(ImplementedTickets))
201 plt.title("Number of Tickets per day - Implemented queue", fontsize = 14)
202 plt.xlabel("Day", fontsize = 12)
203 plt.ylabel("Tickets", fontsize = 12)
204 plt.show()
205
206 plt.plot(return_avg_eod_queue(VerifiedTickets))
207 plt.title("Number of Tickets per day - Verified queue", fontsize = 14)
208 plt.xlabel("Day", fontsize = 12)
209 plt.ylabel("Tickets", fontsize = 12)
210 plt.show()
211
212 fig, ax = plt.subplots()
213 ax.plot(return_avg_eod_queue(engg_1), label='Engg 1')
214 ax.plot(return_avg_eod_queue(engg_2), label='Engg 2')
215 ax.plot(return_avg_eod_queue(engg_3), label='Engg 3')
216 ax.plot(return_avg_eod_queue(engg_4), label='Engg 4')
217 ax.plot(return_avg_eod_queue(engg_5), label='Engg 5')
218 ax.plot(return_avg_eod_queue(engg_6), label='Engg 6')
219 ax.plot(return_avg_eod_queue(engg_7), label='Engg 7')
220 ax.set_xlabel('Days')
221 ax.set_title('Engineer Queues')
222 ax.set_ylabel('Tickets')
223 ax.legend()
224 plt.show()

```