BACHELOR

Optimization of ticketing system for TIOBE Software B.V.

Swami, Dhwani

Award date:
2023

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 3rd 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

List of Algorithms

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 deques (*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[1] have quite different capabilities, skills and paces of working. The challenging task of analysing a combination of priority queues and deques 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

[1] 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)[1]. 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

[1] https://www.tiobe.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[2] 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.*

[2] 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).

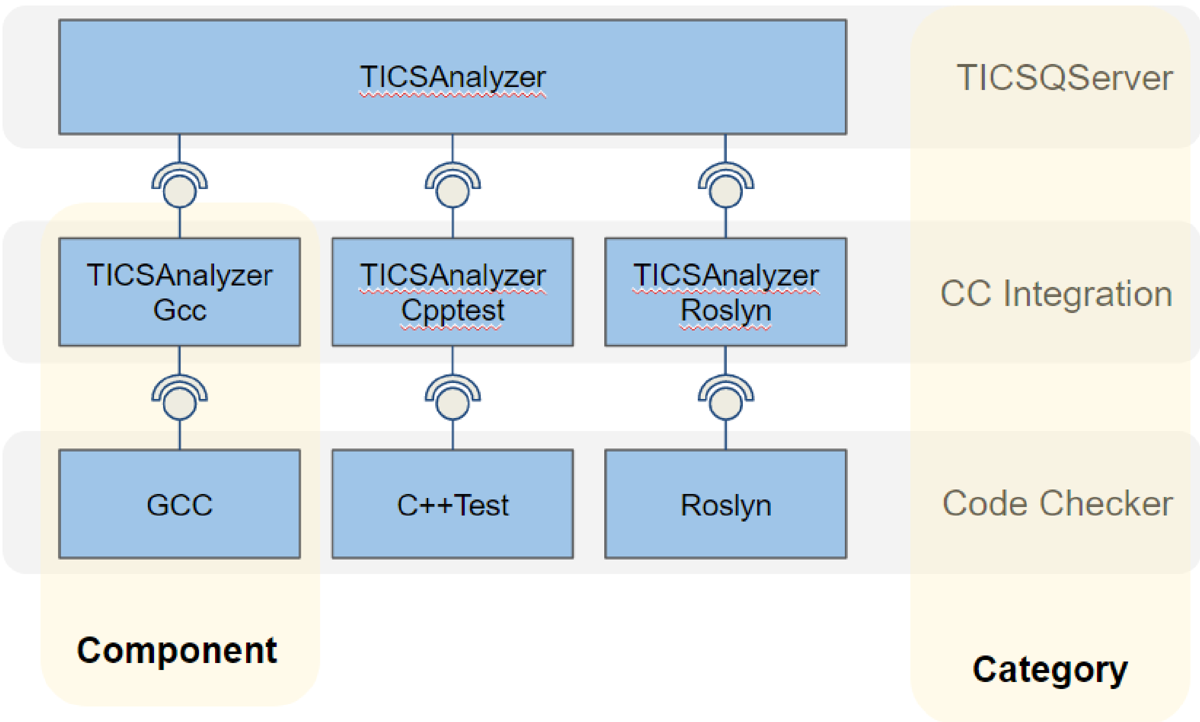

Figure 2.1: Components and Categories

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	TICSpp	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
	lcov						VS/MSBuild C#
	Jacoco						.NetCore
	Istanbul						OpenCover
	Pycover						coverage.py
	LLVM						Bullseye
	MSBuild						lcov
	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.

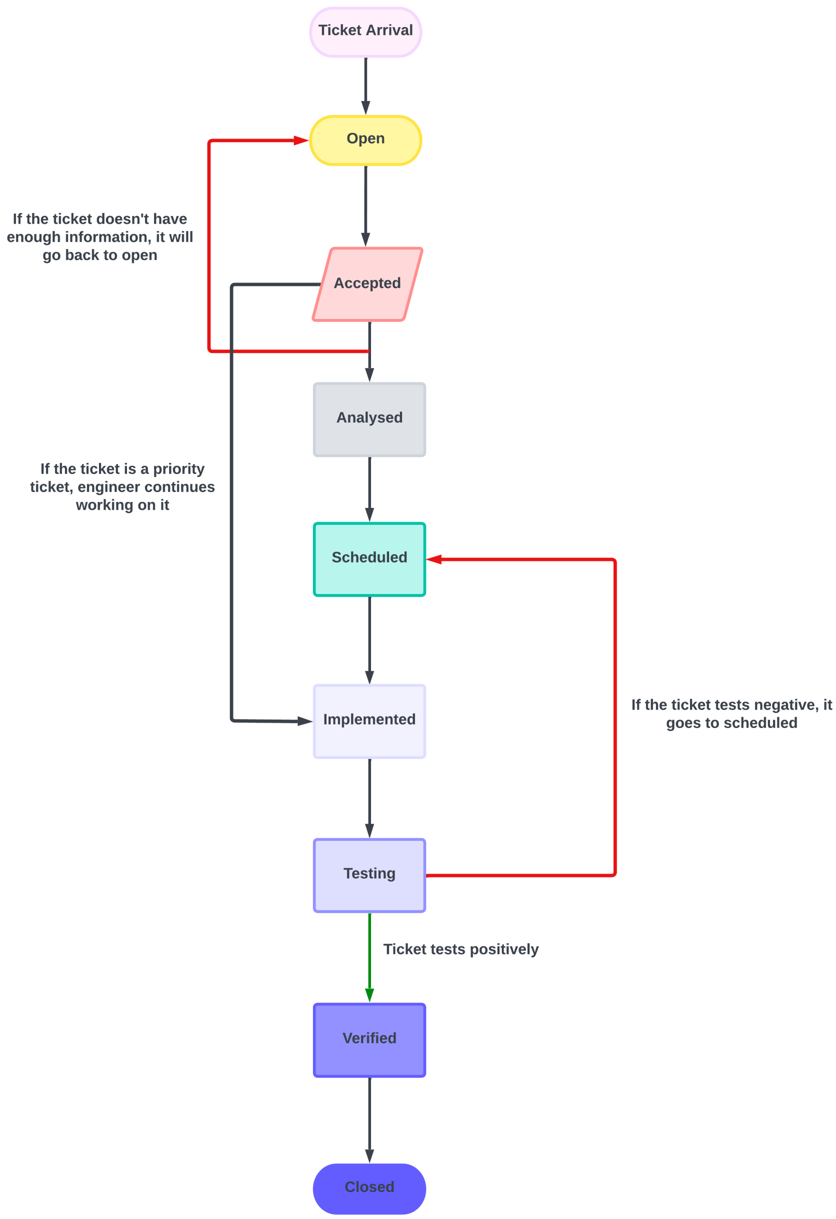

Figure 2.3: Process Flowchart

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[3] 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.

[3] 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.

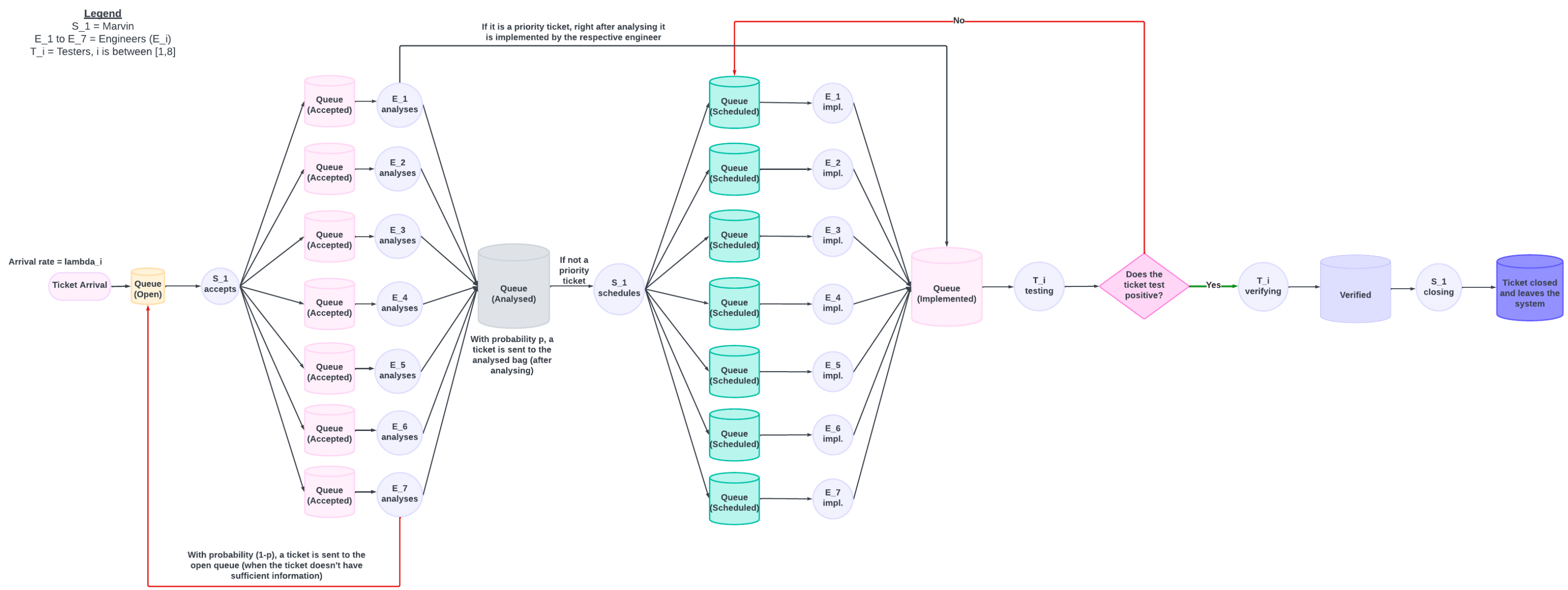

Figure 2.4: Ticket through the system

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 (σ^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 (θ is the scale parameter) as,

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

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

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

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** [1]

   (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.

[1] 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).

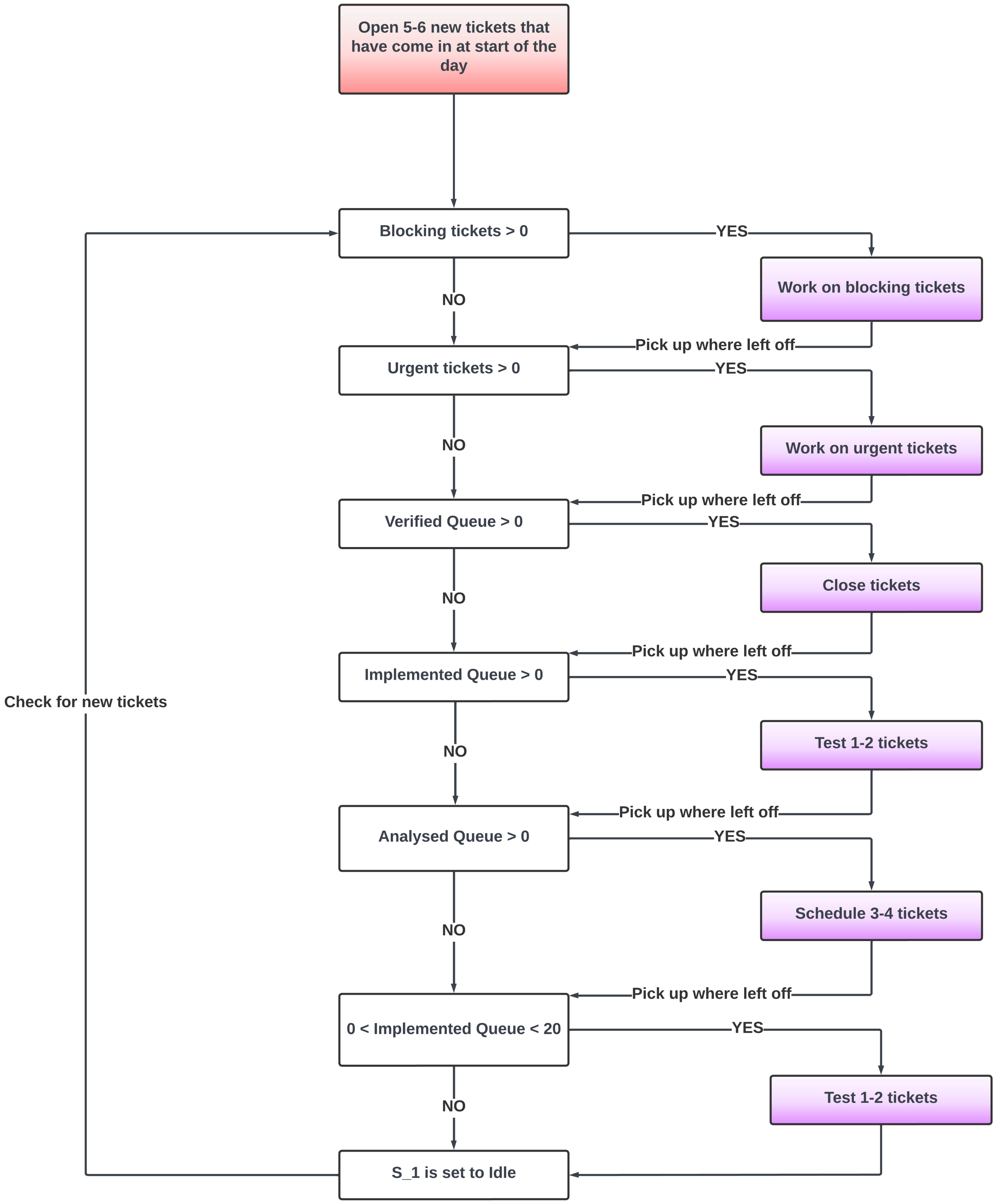

Figure 3.1: S_1 Workflow

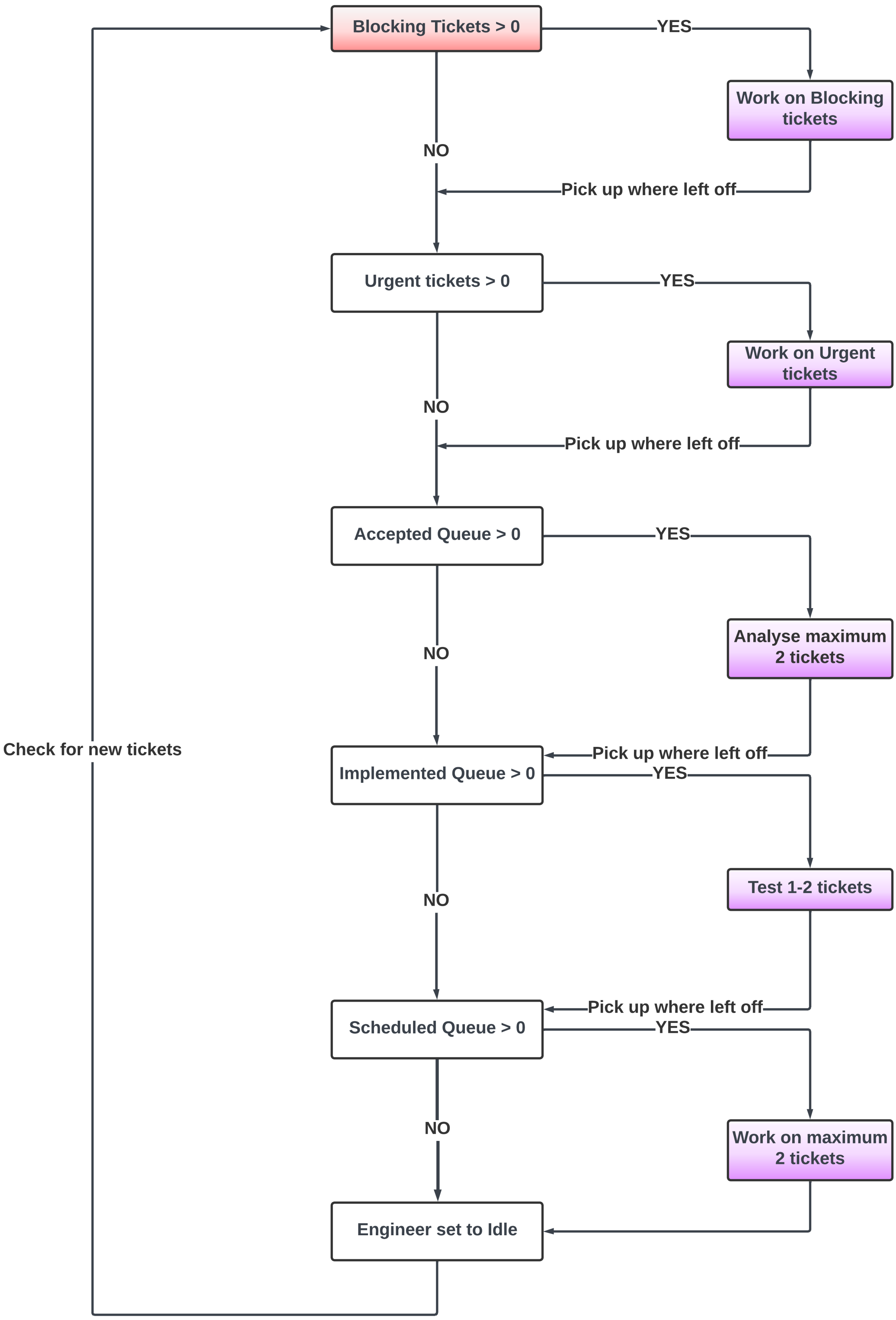

Figure 3.2: Engineer Workflow

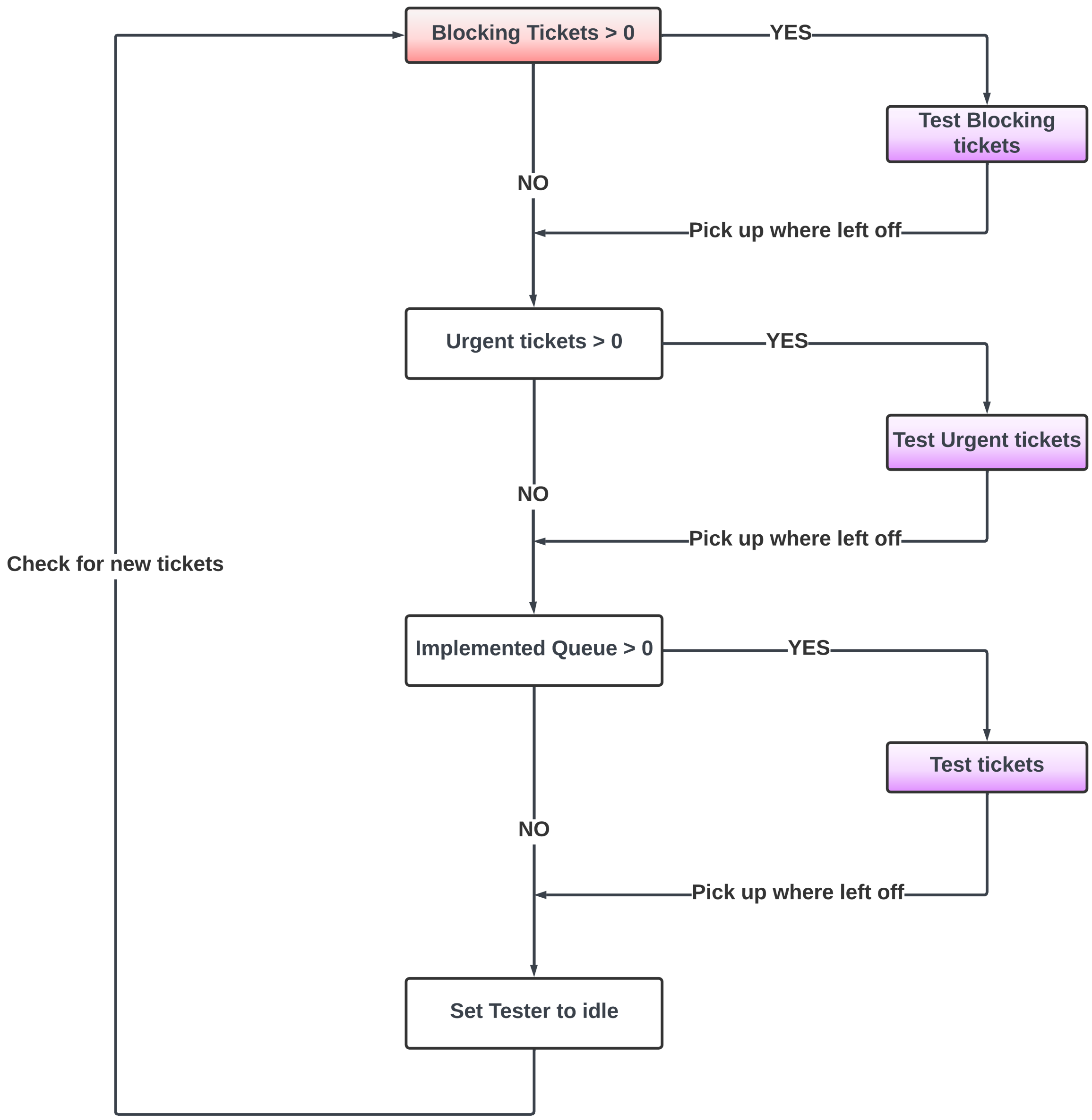

Figure 3.3: Tester Workflow

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)

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

Algorithm 2 Departure event from Open

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

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

procedure ARRIVAL
    Add ticket to **Queue** ▷ **Queue** sorts the tickets on the basis of priorities
    if ticket priority is blocking **then**
        Cancel departure event of ticket they were working on
        Change ticket attributes
        Change server attributes
        Schedule departure at time $t + b$
        Remove ticket from queue
    else if $E[i]$ is idle **then**
        Schedule departure at $t + b$
        Remove ticket from queue
        Update ticket attributes
        Assign $E[i]$ as working on ticket

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

procedure DEPARTURE FROM ACCEPTED
    if ticket is a priority ticket (Blocking) **then**
        Schedule arrival at the implemented queue at time t
    else
        With probability p schedule arrival at time t in analysed bag
        With probability $1 - p$ schedule arrival at time t in open queue
    Check $E[i]$ decision tree to determine next queue/ticket
    if there is no ticket to work on **then**
        Set server attribute to idle ▷ Server here is $E[i]$
    else
        Change ticket attributes
        Change $E[i]$ attributes
        Schedule departure at time $t + b$
        Remove ticket from queue

Algorithm 5 Arrival event on Analysed

procedure ARRIVAL ON ANALYSED
    Add ticket to **Bag**
    if S_1 is idle **then**
        Change ticket attributes
        Change S_1 attributes
        Schedule departure at $t + b$
        Remove ticket from queue

Algorithm 6 Departure event from Analysed

procedure DEPARTURE FROM ANALYSED
    Schedule arrival at $E[i]$ queue at time t
    Check S_1's decision tree to see which queue they'll work on
    if no ticket to work on **then**
        Set server S_1 attribute to idle
    else
        Change ticket attributes
        Change server attributes
        Schedule departure at time $t + b$
        Remove ticket from queue

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

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

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

procedure DEPARTURE FROM SCHEDULED
  Schedule arrival in the implemented queue at time t
  Check $E[i]$'s decision tree
  if no ticket to work on **then**
    Assign $E[i]$ as idle
  else
    Change ticket attributes
    Change server attributes
    Schedule departure at time $t + b$
    Remove ticket from queue

Algorithm 9 Arrival event on Implemented

procedure ARRIVAL ON IMPLEMENTED
  Add ticket to Queue
  $i \leftarrow 0$
  while $i < N$ **do** ▷ N is the number of testers
    if $T[i]$ is idle AND $T[i] \neq E[i]$ **then**
      Change ticket attributes
      Change server attributes
      Schedule departure at time $t + b$
      Remove ticket from queue
      break
    $i = i + 1$
  if ticket priority is Blocking **then**
    $k \leftarrow 0$
    while $k < N$ **do** ▷ N is the number of testers
      if server $T[i]$ not working on a Blocking ticket & $T[i] \neq E[i]$ **then**
        Cancel departure event of ticket they were working on
        Change ticket attributes
        Change server attributes
        Schedule departure at time $t + b$
        Remove ticket from queue
        break
      $k = k + 1$

Algorithm 10 Departure event from Implemented

procedure DEPARTURE FROM IMPLEMENTED
    if (random) test with probability q is negative **then**
        Schedule arrival at time t at $E[i]$ scheduled queue
    else
        Schedule arrival at time t in verified queue
    Check tester's decision tree
    if no tickets to work on **then**
        Assign $T[i]$ as idle
    else
        Schedule departure at time $t + b$
        Remove ticket from queue
        Change ticket attributes
        Change server attributes

Algorithm 11 Arrival event on Verified

procedure ARRIVAL ON VERIFIED
    Add ticket to Queue
    if S_1 is idle **then**
        Change ticket attributes
        Change server attributes
        Schedule departure from the system at time $t + b$
        Remove ticket from queue
    else
        if ticket has priority (blocking) **then**
            if S_1 is not working on priority (blocking) **then**
                Cancel departure of ticket (they are working on)
                Change ticket attributes
                Change server attributes
                Schedule departure from the system at time $t + b$
                Remove ticket from queue

Algorithm 12 Departure event from Verified

procedure DEPARTURE FROM IMPLEMENTED
    Update performance measures
    Check S_1's decision tree
    if no ticket to work on **then**
        Assign S_1 as idle
    else
        Change ticket attributes
        Change server attributes
        Schedule departure at time $t + b$
        Remove ticket from queue

Algorithm 13 End-Of-Day Event

```
procedure End-Of-Day
    Performance measures for the day are updated
    All counters (for every server - Engineers and S_1) are set to 0
    if S_1 is idle and is not working on any ticket then
        Remove a ticket from the Open queue
        Change ticket attributes
        Change server attributes
        Schedule departure at time t + b
    else
        if the ticket S_1 is working on is not priority then
            Cancel departure event of the current ticket S_1 is working on
            Remove a ticket from the Open queue
            Change ticket attributes
            Change server attributes
            Schedule departure at time t + b
            Remove ticket from queue
    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 to 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.

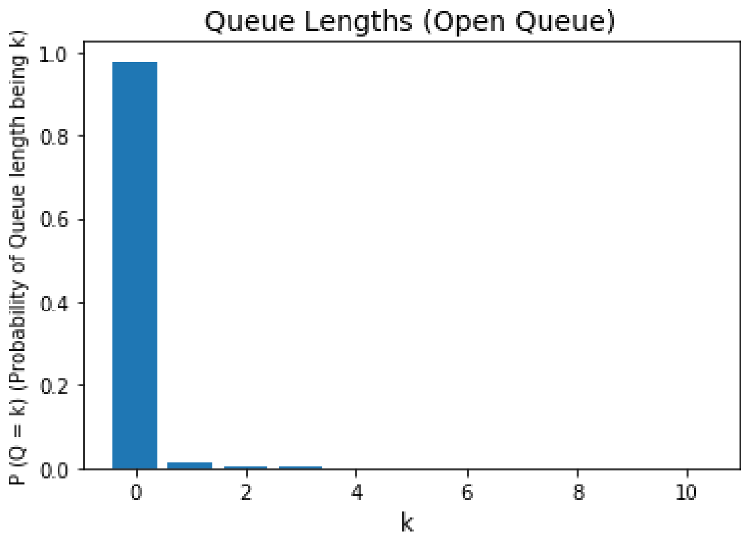

Figure 4.1: Queue Length at Open

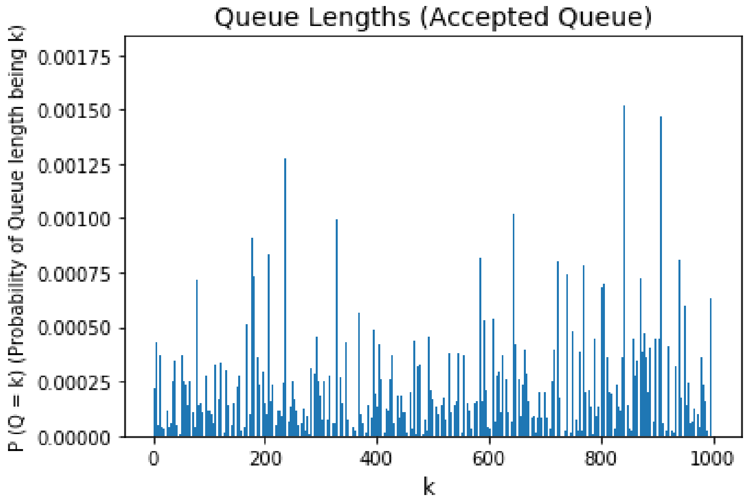

Figure 4.2: Queue Length at Accepted

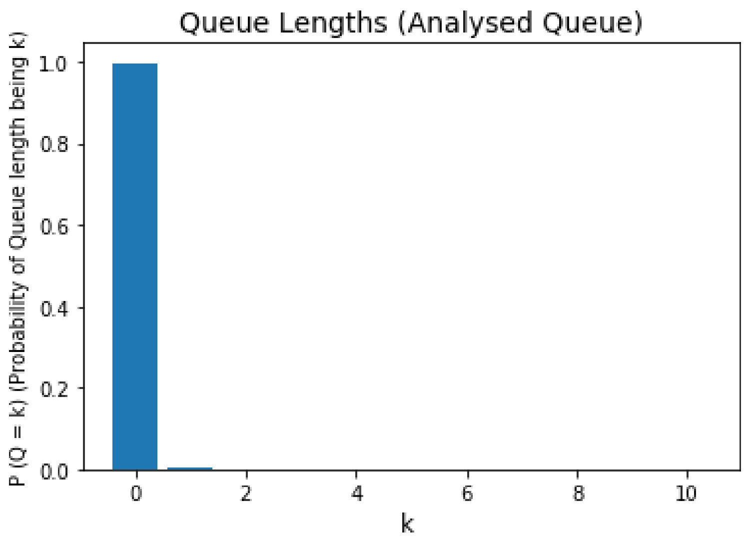

Figure 4.3: Queue Length at Analysed

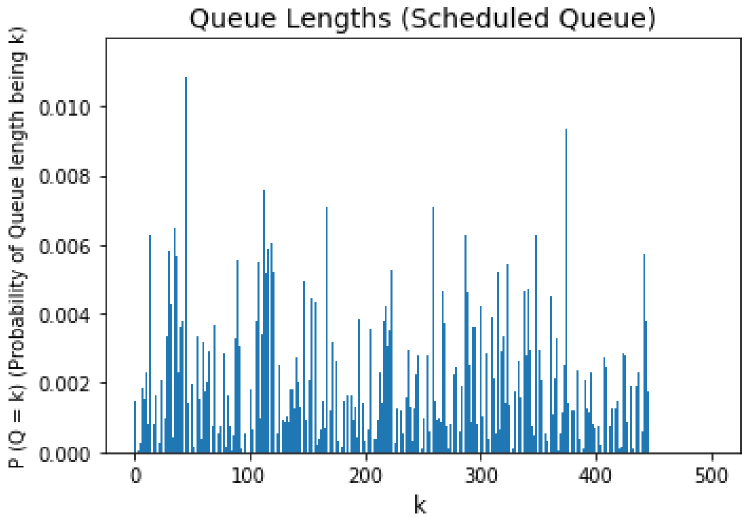

Figure 4.4: Queue Length at Scheduled

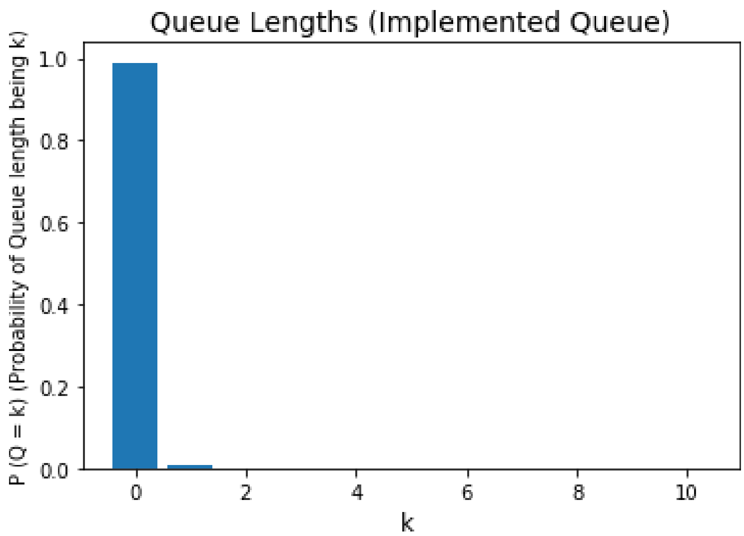

Figure 4.5: Queue Length at Implemented

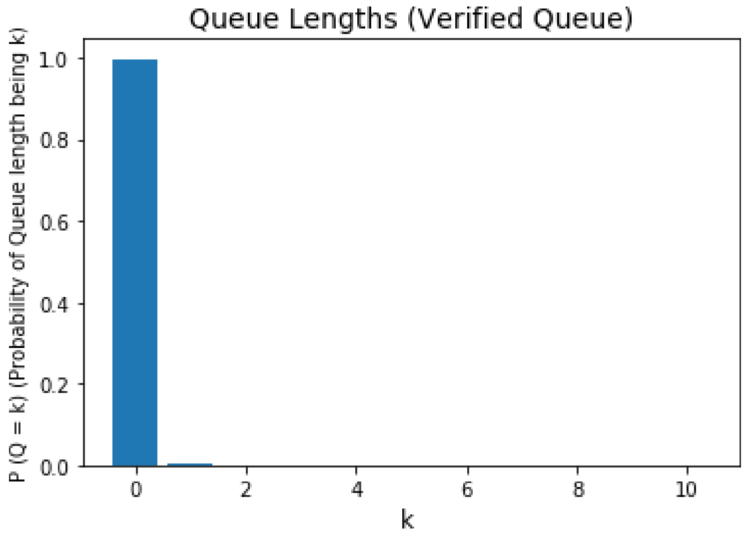

Figure 4.6: Queue Length at Verified

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.

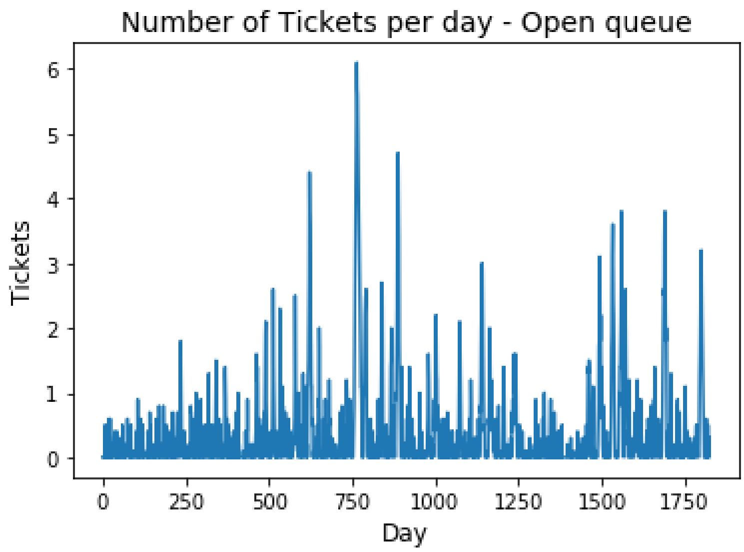

Figure 4.7: Queue Length at Open (End-Of-Day)

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

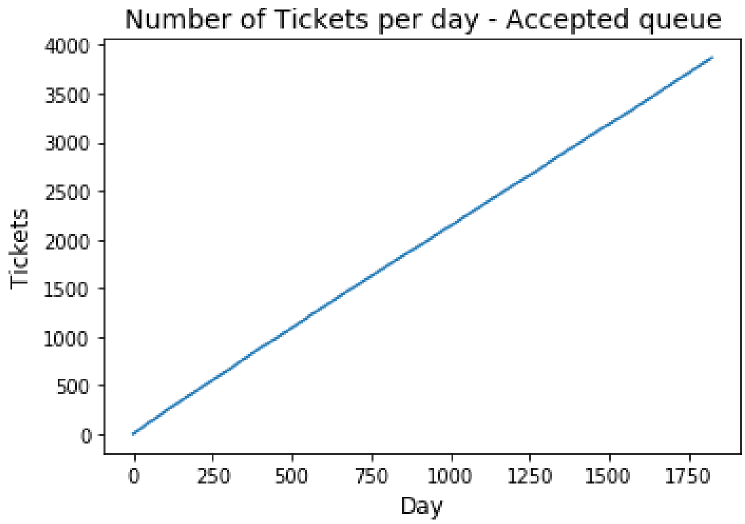

Figure 4.8: Queue Length at Accepted (End-Of-Day)

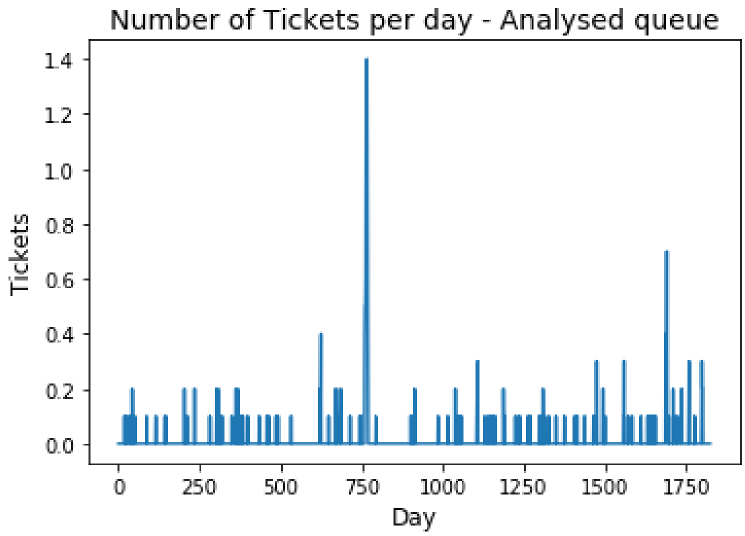

Figure 4.9: Queue Length at Analysed (End-Of-Day)

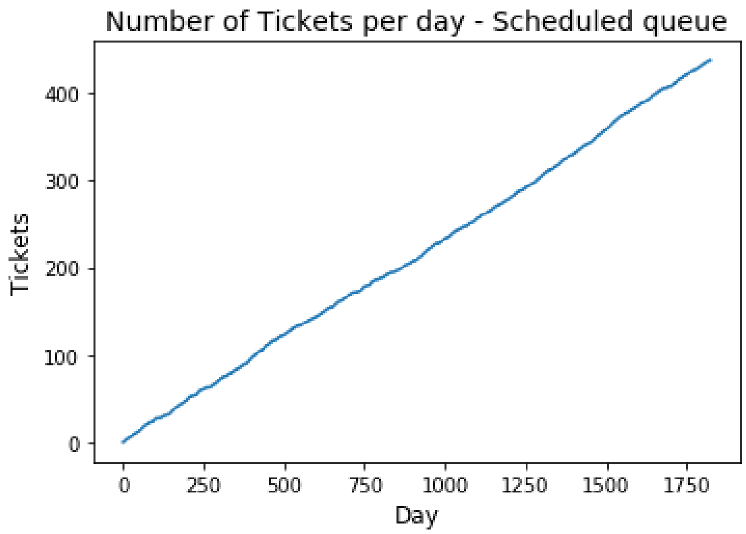

Figure 4.10: Queue Length at Scheduled (End-Of-Day)

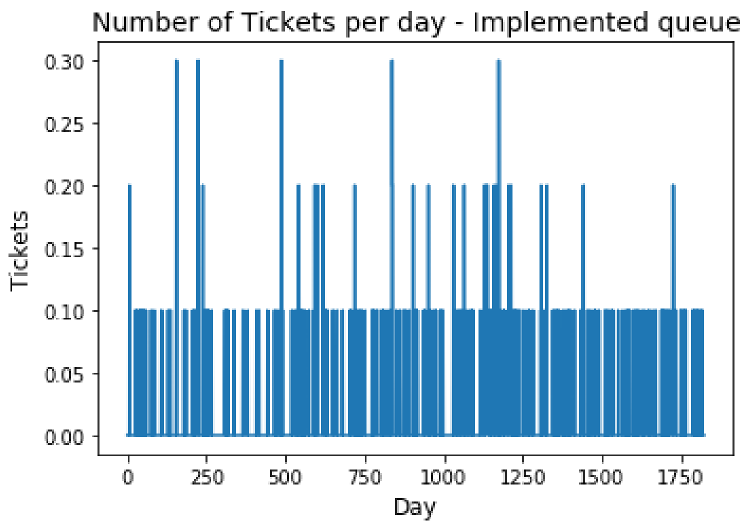

Figure 4.11: Queue Length at Implemented (End-Of-Day)

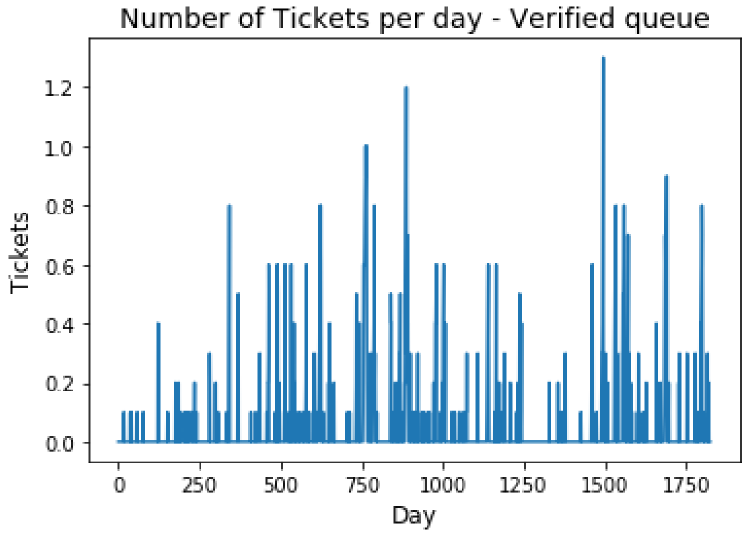

Figure 4.12: Queue Length at Verified (End-Of-Day)

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.

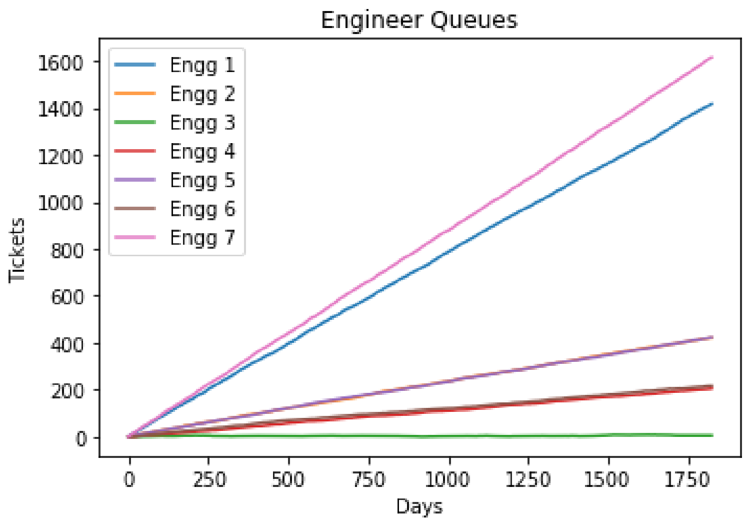

Figure 4.13: Queue Length for Engineers (End-Of-Day)

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.

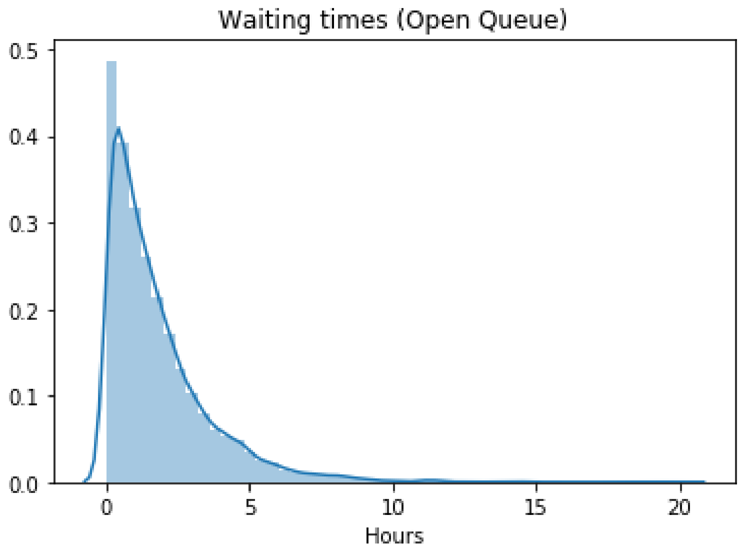

Figure 4.14: Waiting times over every point in time

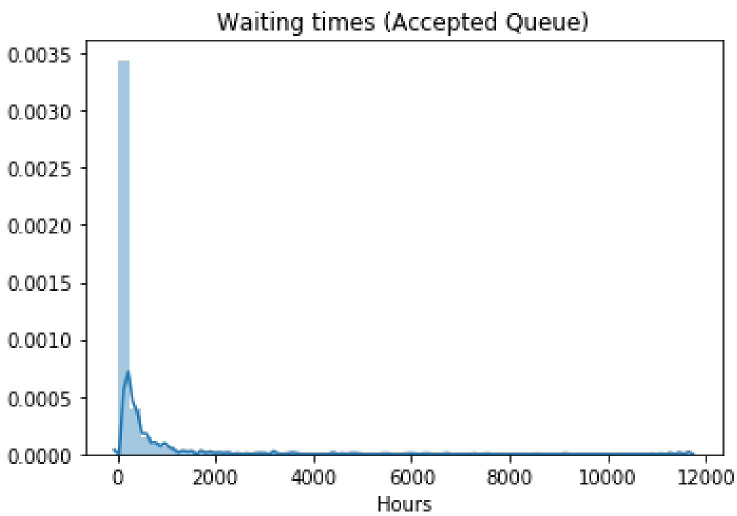

Figure 4.15: Waiting times over every point in time

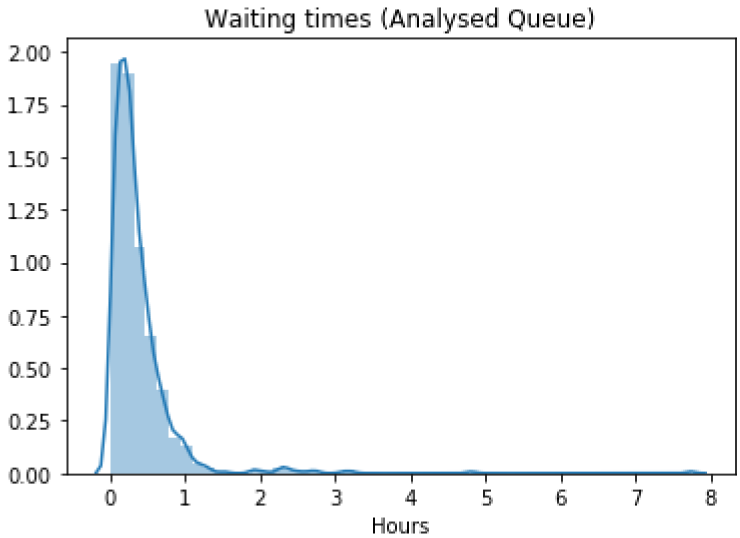

Figure 4.16: Waiting times over every point in time

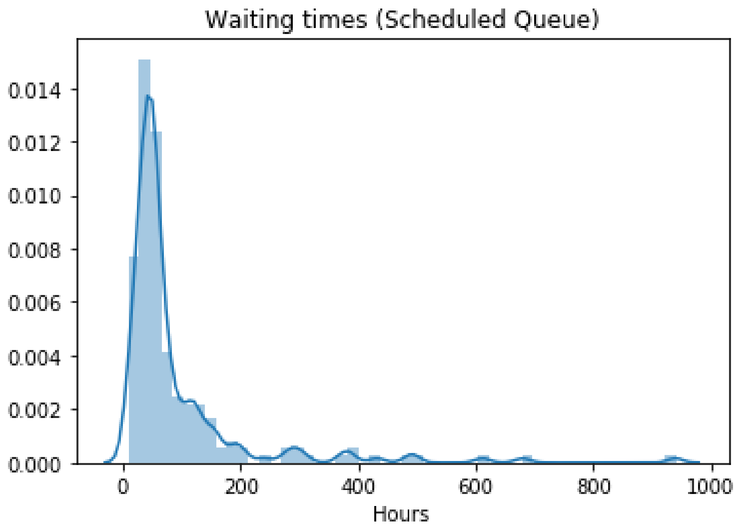

Figure 4.17: Waiting times over every point in time

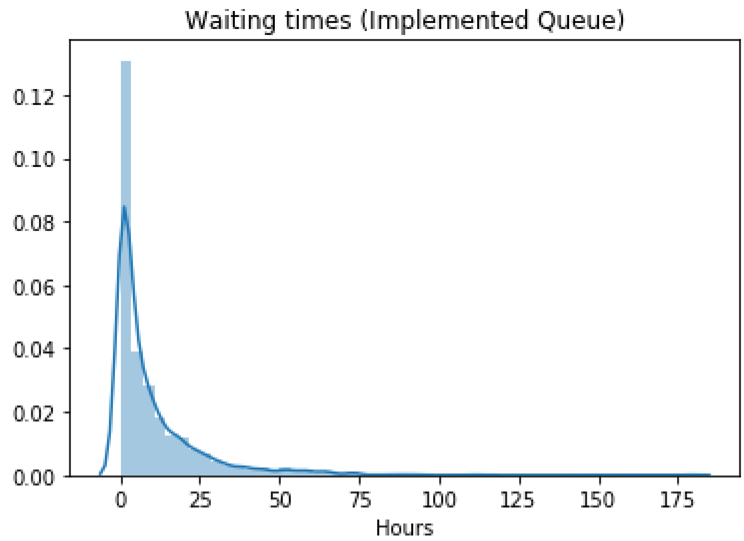

Figure 4.18: Waiting times over every point in time

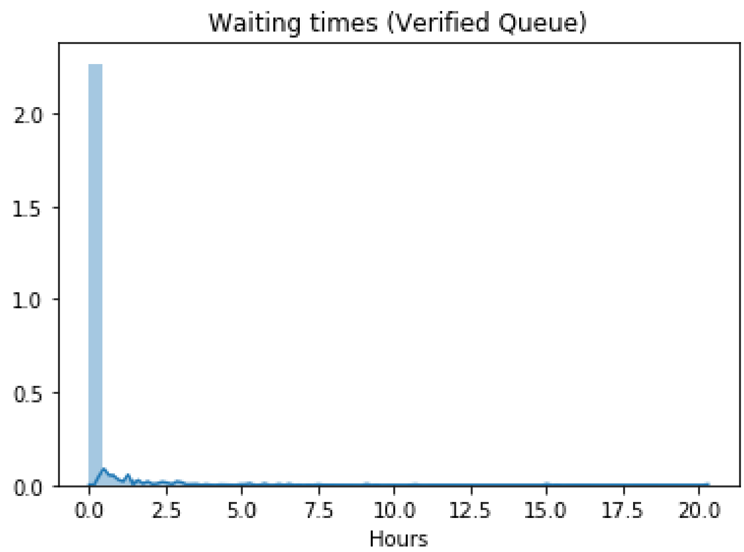

Figure 4.19: Waiting times over every point in time

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.

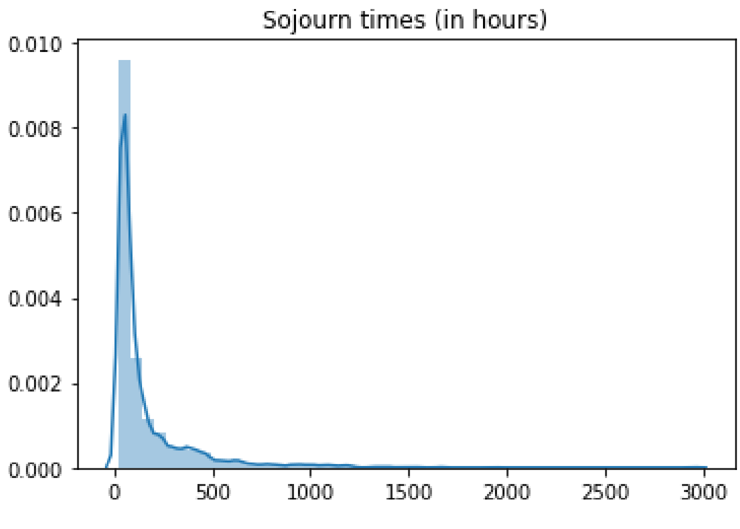

Figure 4.20: Sojourn times

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

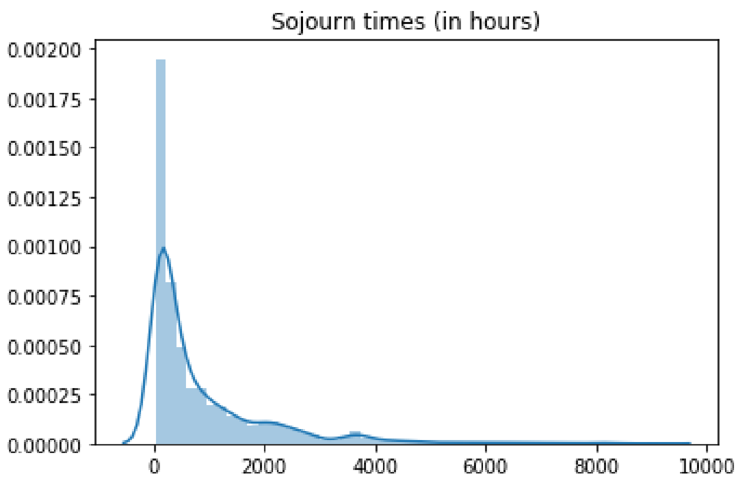

Figure 5.1: Sojourn when tickets have insufficient information and test negative with a probability of 0.1

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.

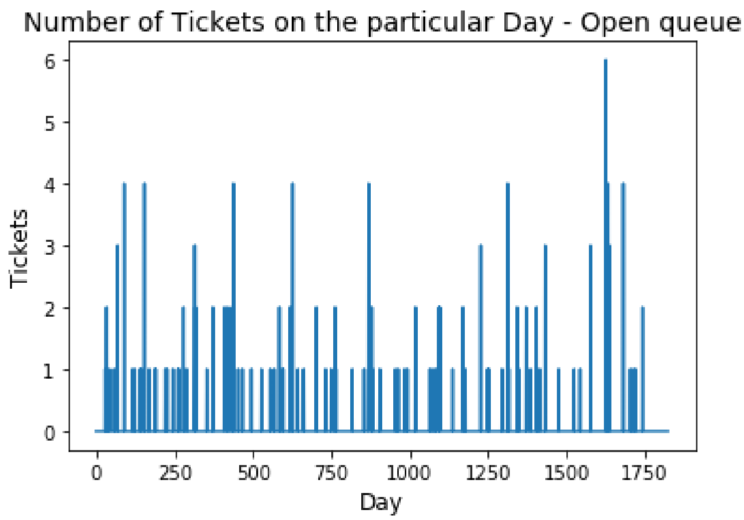

Figure 5.2: Open queue when there are 35 engineers in the system

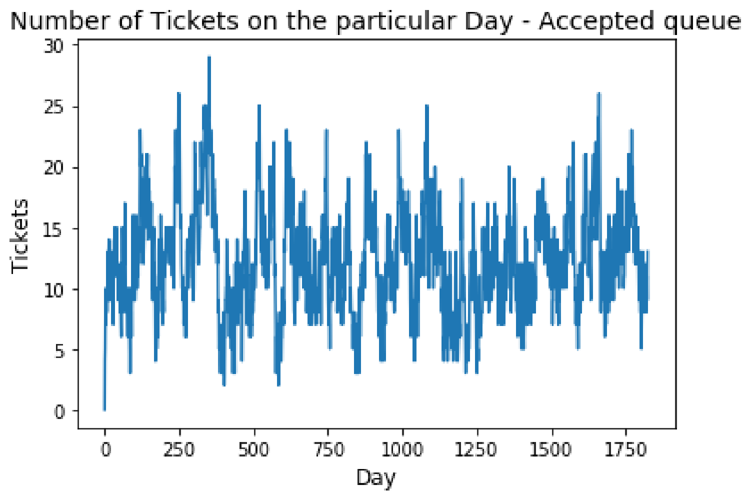

Figure 5.3: Accepted queue when there are 35 engineers in the system

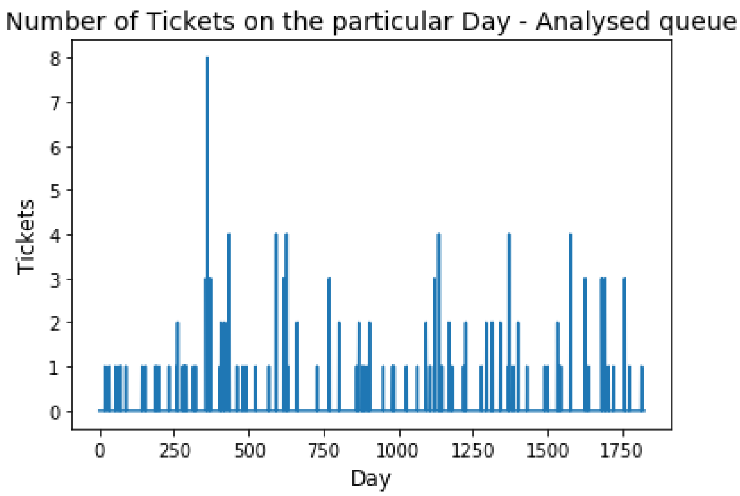

Figure 5.4: Analysed queue when there are 35 engineers in the system

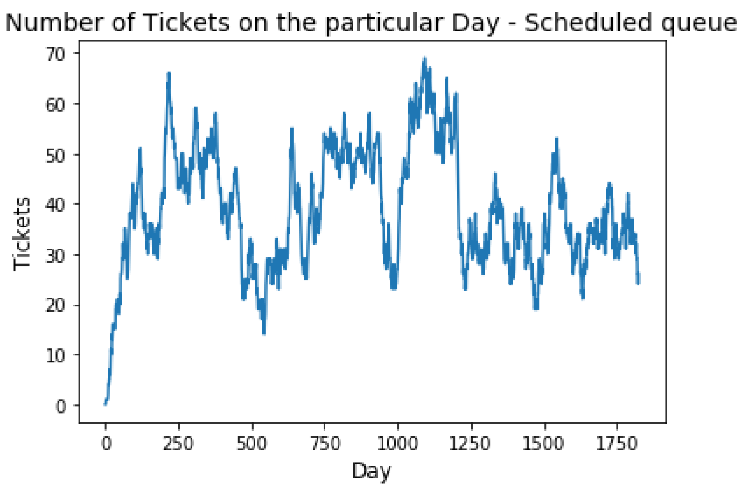

Figure 5.5: Scheduled queue when there are 35 engineers in the system

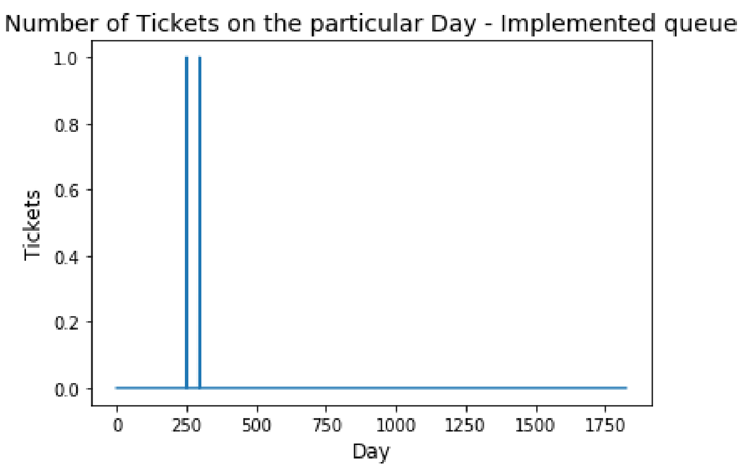

Figure 5.6: Implemented queue when there are 35 engineers in the system

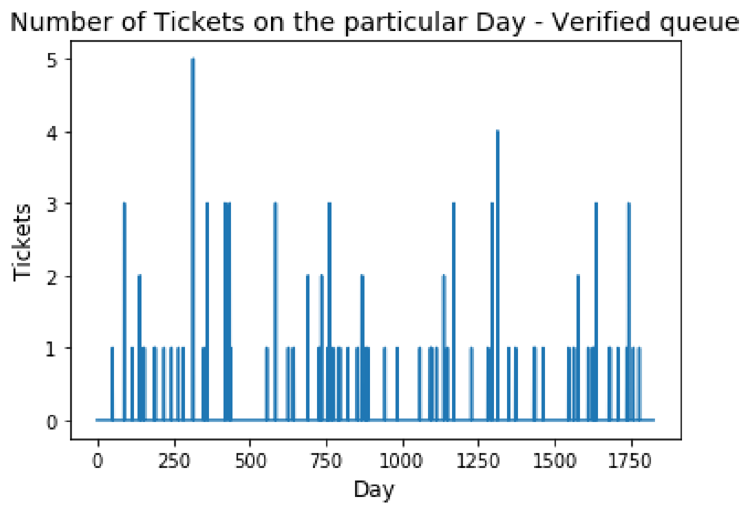

Figure 5.7: Verified queue when there are 35 engineers in the system

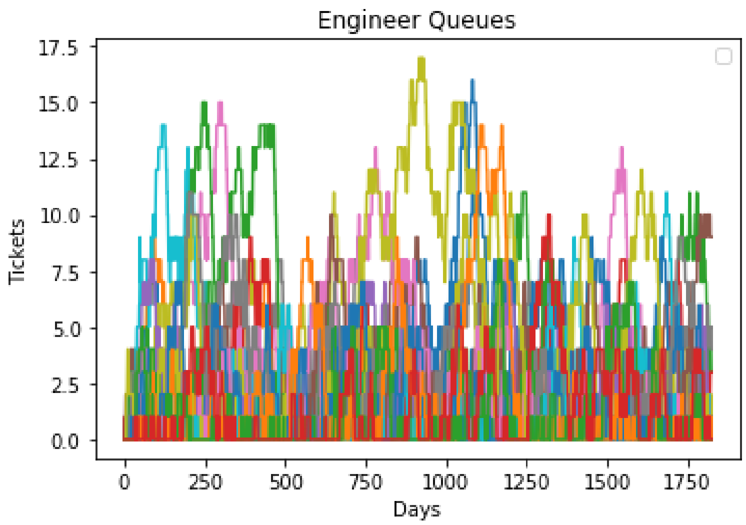

Figure 5.8: Engineers' queues when there are 35 engineers in the system

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

```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct  15 15:04:11 2022

@author: 20181301
"""

import heapq
from priorityQueue import PriorityQueue
import numpy as np


class Ticket:

    number = 0

    def __init__(self, priority, server, arrivalTime, component):
        self.priority = priority
        self.queue = None
        self.pos = 0
        self.server = server
        self.arrivalTime = arrivalTime
        self.systemArrivalTime = arrivalTime
        self.component = component
        self.depEvent = None
        self.engineer = None
        self.probability = np.random.uniform(0,1)   #random number between 0 & 1
        self.state = 0
        self.testProb = np.random.uniform(0,1)      #random number between 0 & 1
        self.tester = None
        self.number = Ticket.number
        Ticket.number = Ticket.number + 1


    def newPos(self, location, time):
        self.state = location
        self.arrivalTime = time

    def leaveSystem(self, time):
        self.state = -1
        self.arrivalTime = -1
```

```
        self.pos = -1

    def __lt__ (self, other):
        return self.priority < other.priority

    def __str__(self):
        return "Ticket number: " + str(self.number) + " in state: " + str(self.state) + " and priority: " + str(self.priority) + " and tester: " + str(self.tester) + " " + str(self.depEvent)


class Engineer(object):

    number = 0

    def __init__(self, component):
        self.pos = 0
        self.component = component
        self.idleTime = 0
        self.workTime = 0
        self.idle = True
        self.queue = PriorityQueue()  #new class PriorityQueue - enqueue, dequeue, location check etc
        self.startWorkingTime = 0       #time when engg starts working
        self.startIdleTime = 0       #time when engg stops working
        self.countAccept = 0
        self.countImplt = 0
        self.countSch = 0
        self.ticket = None
        self.acceptTickets = []
        self.schTickets = []
        self.impltTickets = []
        self.nrTickets = []
        self.nrOfTickets = 0
        self.workTimeFraction = self.workTime/(365*5*8*3600)
        self.idleTimeFraction = self.idleTime/(365*5*8*3600)
        self.number = Engineer.number
        Engineer.number = Engineer.number + 1

    def dequeue(self):
        return heapq.heappop(self.queue)

    def setWorking(self, time):
        self.idleTime = self.idleTime + time - self.startIdleTime
        self.startWorkingTime = time
        self.nrTickets.append(self.ticket.number)
        self.nrOfTickets = self.nrOfTickets + 1

        if self.ticket.state == 1:
            self.acceptTickets.append(self.ticket.number)
        elif self.ticket.state == 3:
            self.schTickets.append(self.ticket.number)
        elif self.ticket.state == 4:
            self.impltTickets.append(self.ticket.number)

    def setIdle(self, time):
        self.workTime = self.workTime + time - self.startWorkingTime
        self.startIdleTime = time

    def __str__(self):
        return "Engineer: " + str(self.number + 1)


class Tester(object):   #

    def __init__(self):
        self.idleTime = 0
```

```
        self.workTime = 0
        self.idle = True
        self.startWorkingTime = 0
        self.startIdleTime = 0
        self.ticket = None
        self.nrTickets = []
        self.nrOfTickets = 0
        self.workTimeFraction = self.workTime/(365*5*8*3600)
        self.idleTimeFraction = self.idleTime/(365*5*8*3600)

    def setWorking(self, time):
        self.idleTime = self.idleTime + time - self.startIdleTime
        self.startWorkingTime = time
        self.nrTickets.append(self.ticket.number)
        self.nrOfTickets = self.nrOfTickets + 1

    def setIdle(self, time):
        self.workTime = self.workTime + time - self.startWorkingTime
        self.startIdleTime = time

    def __str__(self):
        return str("tester")


class S(object):                #for S1

    def __init__(self):
        self.pos = 0
        self.idleTime = 0
        self.workTime = 0
        self.idle = True
        self.startWorkingTime = 0       #time when starts working
        self.startIdleTime = 0       #time when stops working
        self.workTimeFraction = self.workTime/(365*5*8*3600)
        self.idleTimeFraction = self.idleTime/(365*5*8*3600)
        self.openCounter = 0
        self.impltCounter = 0
        self.analyseCounter = 0
        self.verfCounter = 0
        self.nrTickets = []
        self.nrOfTickets = 0
        self.ticket = None

    def setWorking(self, time):
        self.idleTime = self.idleTime + time - self.startIdleTime
        self.startWorkingTime = time
        self.nrTickets.append(self.ticket.number)
        self.nrOfTickets = self.nrOfTickets + 1


    def setIdle(self, time):
        self.workTime = self.workTime + time - self.startWorkingTime
        self.startIdleTime = time

    def __str__(self):
        return str("s1")


class Event(object):

    ARRIVAL = 0
    DEPARTURE = 1
    ABANDONMENTS = -1  #ticket rejected/wait
    EOD = 2    #end of day


    def __init__(self, typ, server, ticket, time):
```

```
        self.type = typ                         #type of event
        self.server = server
        self.ticket = ticket
        self.time = time
        self.state = None
        self.iscancelled = False         #for cancelling event; when handelling event, first check if it is cancelled or not

    def __lt__ (self, other):
        return self.time < other.time

    def __str__( self ):
        s = ('Arrival', 'Departure', 'Abandonments', 'EOD')
        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)

    def cancel(self):
        self.iscancelled = True
```

A.2 Distributions

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 23 21:19:28 2022

@author: 20181301
"""


from scipy import stats
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


class Distribution :

    n = 10000 # standard random numbers to generate

    def __init__(self, dist):
        self.dist = dist
        self.resample()

    def __str__(self):
        return str(self.dist)

    def resample(self):
        self.randomNumbers = self.dist.rvs(self.n)
        self.idx = 0

    def rvs(self, n=1):

        if self.idx >= self.n - n :
            while n > self.n :
                self.n *= 10
            self.resample()
        if n == 1 :
            rs = self.randomNumbers[self.idx]
        else :
            rs = self.randomNumbers[self.idx:(self.idx+n)]
        self.idx += n
        return rs

    def mean(self):
```

```
        return self.dist.mean()

    def std(self):
        return self.dist.std()

    def var(self):
        return self.dist.var()

    def cdf(self, x):
        return self.dist.cdf(x)

    def pdf(self, x):
        return self.dist.pdf(x)

    def sf(self, x):
        return self.dist.sf(x)

    def ppf(self, x):
        return self.dist.ppf(x)

    def moment(self, n):
        return self.dist.moment(n)

    def median(self):
        return self.dist.median()

    def interval(self, alpha):
        return self.dist.interval(alpha)
```

A.3 Future Event Set

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 14 00:26:08 2022

@author: 20181301
"""

import heapq

class FES:

    def __init__ (self):
        self.events = []

    def add (self, event):
        heapq.heappush(self.events, event)

    def next (self):
        return heapq.heappop(self.events)

    def __str__(self):
        r = []
        for i in self.events:
            r.append(str(i))
        return str(r)
```

A.4 Priority Queue

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Wed Nov 16 11:21:19 2022

@author: 20181301
"""


from collections import deque
from heapq import heappop, heappush, heapify
import heapq

class PriorityQueue:
    def __init__(self):
        self.tickets = []
        self.state = None

    def enqueue(self, ticket):
        heappush(self.tickets, ticket)
        heapq.heapify(self.tickets)

    def dequeue_0(self):
        return heappop(self.tickets)

    def dequeue_ticket(self, ticket):
        self.tickets.remove(ticket)
        heapq.heapify(self.tickets)

    def firstTicket(self):
        if len(self.tickets) == 0:
            return None
        else:
            return self.tickets[0]

    def enqueue_front(self, ticket):
        return self.appendleft(ticket)

    def __str__(self):
        s = str(" ")
        for i in self.tickets:
            s = s + str(i.number) + str(", ")
        return str("Tickets in Queue: ") + s
```

A.5 Queue

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 14 00:31:07 2022

@author: 20181301
"""

from collections import deque
import collections


class Queue:
    def __init__(self):
        self.tickets = deque()
        self.state = None

    def enqueue(self, ticket):
        self.tickets.append(ticket)

    def dequeue_0(self):
        return self.tickets.popleft()

```

```
    def dequeue_ticket(self, ticket):
        self.tickets.remove(ticket)

    def firstTicket(self):
        if len(self.tickets) == 0:
            return None
        else:
            return self.tickets[0]

    def enqueue_front(self, ticket):
        return self.appendleft(ticket)

    def __str__(self):
        s = str(" ")
        for i in self.tickets:
            s = s + str(i.number) + str(", ")
        return str("Tickets in Queue: ") + s
```

A.6 Simulation Results

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 18 00:47:47 2022

@author: 20181301
"""

from collections import deque
from numpy . ma . core import zeros
import matplotlib . pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np



class SimResults :

    MAX_QL = 10000 # maximum queue length that will be recorded

    def __init__ ( self ):
        self . sumQL = 0
        self . sumQL2 = 0
        self . oldTime = 0
        self . queueLengthHistogram = zeros ( self . MAX_QL + 1)
        self . sumW = 0
        self . sumW2 = 0
        self . nW = 0
        self . waitingTimes = deque ()
        self.waitingTimes_hrs = deque ()


    def registerQueueLength ( self , time , ql ):
        self . sumQL = self . sumQL + ql * ( time - self . oldTime )
        self . sumQL2 = self . sumQL2 + ql * ql * ( time - self . oldTime )
        self . queueLengthHistogram [ min ( ql , self . MAX_QL )] = self .
    queueLengthHistogram [ min ( ql , self . MAX_QL )] + ( time - self . oldTime )
        self . oldTime = time

    def registerWaitingTime ( self , W ):
        w = W
        self.waitingTimes.append(w)
        self.waitingTimes_hrs.append(w/3600)
        self.nW = self.nW + 1
        self.sumW = self.sumW + w
```

```
        self.sumW2 = self.sumW2 + w*w

    def getMeanQueueLength ( self ):
        return self.sumQL/self.oldTime

    def getVarianceQueueLength ( self ):
        return self . sumQL2 / self . oldTime - self . getMeanQueueLength ()**2

    def getMeanWaitingTime ( self ):
        return self . sumW / self . nW

    def getVarianceWaitingTime ( self ):
        return self . sumW2 / self . nW - self . getMeanWaitingTime ()**2

    def getQueueLengthHistogram ( self ) :
        return [x/ self . oldTime for x in self . queueLengthHistogram ]

    def getWaitingTimes ( self ):
        return self . waitingTimes_hrs

    def __str__ ( self ):
        s = ' Mean queue length : '+ str ( (self . getMeanQueueLength ())) + '\n '
        s += ' Variance queue length : '+ str ( (self . getVarianceQueueLength ())) + '\n '
        s += ' Mean waiting time (in seconds) : '+ str (float(( self . getMeanWaitingTime ()))) + '\n '
        s += ' Mean waiting time (in hours) : '+ str (float(( self . getMeanWaitingTime ()))/3600) + '\n '
        s += ' Variance waiting time (in hrs^2) : '+ str (float(( self . getVarianceWaitingTime ()))/3600**2) + '\n '
        return s

    def histQueueLength ( self ,queueNr , maxq =50):

        if queueNr == 0:
            ql = self . getQueueLengthHistogram ()
            maxx = maxq + 1
            plt.figure(figsize=(12, 6))
            plt . figure ()
            plt . bar ( range (0 , maxx ), ql [0: maxx ])
            plt.title("Queue Lengths (Open Queue)", fontsize = 14)
            plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)', fontsize = 10)
            plt . xlabel ( 'k ', fontsize = 12)
            plt . show ()

            print("\nSum of probability open queue lengths - ", sum(self.getQueueLengthHistogram()))

        elif queueNr == 1:
            ql = self . getQueueLengthHistogram ()
            maxx = maxq + 1
            plt.figure(figsize=(12, 6))
            plt . figure ()
            plt . bar ( range (0 , maxx ), ql [0: maxx ])
            plt.title("Queue Lengths (Accepted Queue)", fontsize = 14)
            plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)', fontsize = 10)
            plt . xlabel ( 'k ', fontsize = 12)
            plt . show ()

            print("\nSum of probability accepted queue lengths - ", sum(self.getQueueLengthHistogram()))

        elif queueNr == 2:
            ql = self . getQueueLengthHistogram ()
            maxx = maxq + 1
```

```
            plt.figure(figsize=(12, 6))
            plt . figure ()
            plt . bar ( range (0 , maxx ), ql [0: maxx ])
            plt.title("Queue Lengths (Analysed Queue)", fontsize = 14)
            plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)', fontsize = 10)
            plt . xlabel ( 'k ', fontsize = 12)
            plt . show ()

            print("Sum of probability analysed queue lengths - ", sum(self.getQueueLengthHistogram()))

        elif queueNr == 3:
            ql = self . getQueueLengthHistogram ()
            maxx = maxq + 1
            plt.figure(figsize=(12, 6))
            plt . figure ()
            plt . bar ( range (0 , maxx ), ql [0: maxx ])
            plt.title("Queue Lengths (Scheduled Queue)", fontsize = 14)
            plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)', fontsize = 10)
            plt . xlabel ( 'k ', fontsize = 12)
            plt . show ()

            print("\nSum of probability scheduled queue lengths - ", sum(self.getQueueLengthHistogram()))

        elif queueNr == 4:
            ql = self . getQueueLengthHistogram ()
            maxx = maxq + 1
            plt.figure(figsize=(12, 6))
            plt . figure ()
            plt . bar ( range (0 , maxx ), ql [0: maxx ])
            plt.title("Queue Lengths (Implemented Queue)", fontsize = 14)
            plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)', fontsize = 10)
            plt . xlabel ( 'k ', fontsize = 12)
            plt . show ()

            print("\nSum of probability implemented queue lengths - ", sum(self.getQueueLengthHistogram()))

        elif queueNr == 5:
            ql = self . getQueueLengthHistogram ()
            maxx = maxq + 1
            plt.figure(figsize=(12, 6))
            plt . figure ()
            plt . bar ( range (0 , maxx ), ql [0: maxx ])
            plt.title("Queue Lengths (Verified Queue)", fontsize = 14)
            plt . ylabel ( 'P (Q = k) (Probability of Queue length being k)', fontsize = 10)
            plt . xlabel ( 'k ', fontsize = 12)
            plt . show ()

            print("\nSum of probability verified queue lengths - ", sum(self.getQueueLengthHistogram()))

    def histWaitingTimes ( self ,queueNr , nrBins =100):

        if queueNr == 0:
            sns.distplot(self.getWaitingTimes(), kde=True)
            plt.title("Waiting times (Open Queue)")
            plt.xlabel("Hours")
            plt . show ()

        elif queueNr == 1:
            sns.distplot(self.getWaitingTimes(), kde=True)
```

```
            plt.title("Waiting times (Accepted Queue)")
            plt.xlabel("Hours")
            plt . show ()

        elif queueNr == 2:
            sns.distplot(self.getWaitingTimes(), kde=True)
            plt.title("Waiting times (Analysed Queue)")
            plt.xlabel("Hours")
            plt . show ()

        elif queueNr == 3:
            sns.distplot(self.getWaitingTimes(), kde=True)
            plt.title("Waiting times (Scheduled Queue)")
            plt.xlabel("Hours")
            plt . show ()

        elif queueNr == 4:
            sns.distplot(self.getWaitingTimes(), kde=True)
            plt.title("Waiting times (Implemented Queue)")
            plt.xlabel("Hours")
            plt . show ()

        elif queueNr == 5:
            sns.distplot(self.getWaitingTimes(), kde=True)
            plt.title("Waiting times (Verified Queue)")
            plt.xlabel("Hours")
            plt . show ()

class NetworkSimResults:

    def __init__ (self , nrOfQueues , nrOfTickets ):
        self .nS = 0
        self . sojournTimes = []
        self . sojournTimes_hrs = []
        self . nrOfAbandonments = 0
        self . queueResults = [ None ] * nrOfQueues
        for i in range ( nrOfQueues ):
            self . queueResults [i] = SimResults ()

    def registerWaitingTime (self , queueNr , w):
        self . queueResults [ queueNr ]. registerWaitingTime (w)

    def registerSojournTime (self , s):
        self.sojournTimes.append(s)
        self.sojournTimes_hrs.append(s/3600)

    def registerAbandonment (self , queueNr ):
        self . nrOfAbandonments += 1
        self . queueResults [ queueNr ]. registerAbandonment ()

    def histSojournTimes ( self , nrBins =100):
        sns.distplot(self.sojournTimes_hrs , kde=True)
        plt.title("Sojourn times (in hours)")
        plt . show ()
```

A.7 Main Simulation

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 16 09:37:36 2022

@author: 20181301
"""

"""
```

```
States -> 0 (Open), 1 (Accepted), 2 (Analysed), 3 (Scheduled), 4 (Implemented), 5 (
    Verifying)

"""

# Libraries

from classes import Event, Ticket, S, Engineer, Tester
import pandas as pd
import numpy as np
from numpy.random import default_rng
import time
import math
import random
from scipy import stats, integrate
from distribution import Distribution
from FES import FES
from simResults import SimResults, NetworkSimResults
from Queue import Queue
from collections import deque
from priorityQueue import PriorityQueue
from statistics import mean
import matplotlib . pyplot as plt
import seaborn as sns
import random
import csv

def sample_from_bernoulli(p):
    return 1 if random.random() < p else 0

# Definitions and Declarations

s1 = S() #server 1 (Marvin)

tester = Tester() #tester (Joep)

p = sample_from_bernoulli(1)        # probability of ticket being sent to analysed
    instead of being sent back to open
test_p = 1     # probability of ticket testing positive

df1 = pd.read_excel(r'C:\Users\20181301\Desktop\APPLIED MATH\YEAR-3\BFP\Simulation\
    Components.xlsx')
components = df1["Component"]

df2 = pd.read_excel(r'C:\Users\20181301\Desktop\APPLIED MATH\YEAR-3\BFP\Simulation\
    Tools & Servers.xlsx')
# Engineers
engg = []
engg.append(Engineer(df2["E1"]))
engg.append(Engineer(df2["E2"]))
engg.append(Engineer(df2["E3"]))
engg.append(Engineer(df2["E4"]))
engg.append(Engineer(df2["E5"]))
engg.append(Engineer(df2["E6"]))
engg.append(Engineer(df2["E7"]))

priorities = [0, 1, 2, 3, 4]


nrQueues = 5         #0 = Open, 1 = Analysed, 2 = Implemented, 3 = Verified
nrServers = 2 + len(engg) # 2 (tester and s1) and number of engineers

# Variables for storing results

priority_0 = []
priority_1 = []
```

```
priority_2 = []
priority_3 = []
priority_4 = []

count_priority_0 = []
count_priority_1 = []
count_priority_2 = []
count_priority_3 = []
count_priority_4 = []

headers = list(components.values)
headers.remove("Component")
sojournTimes_Components = pd.DataFrame(columns=headers)

headers = list(components.values)
headers.remove("Component")
totalTimes_Components = pd.DataFrame(columns=headers)

headers = [s1, tester, engg[0], engg[1], engg[2], engg[3], engg[4], engg[5], engg
    [6]]
sojournTimes_Servers = pd.DataFrame(columns=headers)
sojournTimes_Servers_idle = pd.DataFrame(columns=headers)

sojournTimes_Priorities = pd.DataFrame(columns=priorities)

totalTimes_priorities = pd.DataFrame(columns=priorities)

TotalTicketsNr = []

OpenTickets = []
AcceptedTickets = []
AnalysedTickets = []
ScheduledTickets = []
ImplementedTickets = []
VerifiedTickets = []

engg_1 = []
engg_2 = []
engg_3 = []
engg_4 = []
engg_5 = []
engg_6 = []
engg_7 = []
s1_tickets = []
tester_tickets = []

header = ["Tickets"]
nrDays_vs_Tickets = pd.DataFrame(columns = header)

def storeSojournTimes_priorities(ticket, t):     # Stores how much time each
    priority ticket spends in the system & ticket nr.
    priority = ticket.priority
    if priority == 0:
        count_priority_0.append(ticket.number)
        priority_0.append(t)
    elif priority == 1:
        count_priority_1.append(ticket.number)
        priority_1.append(t)
    elif priority == 2:
        count_priority_2.append(ticket.number)
        priority_2.append(t)
    elif priority == 3:
        count_priority_3.append(ticket.number)
        priority_3.append(t)
    elif priority == 4:
        count_priority_4.append(ticket.number)
        priority_4.append(t)
```

```python

def assignFreeTester(ticket):                                # Assign idle server
    free_testers = [server for server in engg if server.idle == True and server != ticket.engineer]
    if free_testers:
        return random.choice(free_testers)
    else:
        return None

def assignTester(ticket):                       # Assigning tester to a priority ticket
    servers = [server for server in engg if server != ticket.engineer and server.ticket.priority != 0]
    if servers:
        return random.choice(servers)
    else:
        return None

def remove_state_ticket(engineer):
    if engineer.ticket.state == 1:
        engineer.acceptTickets.remove(engineer.ticket.number)
    elif engineer.ticket.state == 3:
        engineer.schTickets.remove(engineer.ticket.number)
    elif engineer.ticket.state == 4:
        engineer.impltTickets.remove(engineer.ticket.number)

def add_state_ticket(engineer, ticket):
    if ticket.state == 1:
        engineer.acceptTickets.append(ticket.number)
    elif ticket.state == 3:
        engineer.schTickets.append(ticket.number)
    elif ticket.state == 4:
        engineer.impltTickets.append(ticket.number)


def assignPos(t):                               # Assigning position (used in Engineer's workflow)
    # states - 0 (Open), 1 (Accepted), 2 (Analysed), 3 (Scheduled), 4 (Implemented), 5 (Verifying)
    # positions - 0 for open, 1 for analysed, 2 for implemented, 3 for verified, 4 for engineer
    if t.state == 0:
        return 0
    elif t.state == 1 or t.state == 3:
        return 4
    elif t.state == 2:
        return 1
    elif t.state == 4:
        return 2
    elif t.state == 5:
        return 3

def countAccepted(server):                      # Counting number of tickets in accepted state
    count = 0
    for i in server.queue.tickets:
        if i.state == 1:
            count = count + 1
    return count

def queueAccepted(server):                     # Returns list of tickets in accepted state
    q_ = []
    for i in server.queue.tickets:
        if i.state == 1:
            q_.append(i)
    return q_

```

```
def queueImplt(queue,server):               # Returns list of tickets in implemented state that arent worked on by the server
    q_ = []
    for i in queue.tickets:
        if i.tester == server:
            q_.append(i)
    return q_

def countSch(server):                       # Counts number of tickets in scheduled state
    count = 0
    for i in server.queue.tickets:
        if i.state == 3:
            count = count + 1
    return count

def queueSch(server):                       # Returns list of tickets in scheduled state
    q_ = []
    for i in server.queue.tickets:
        if i.state == 3:
            q_.append(i)
    return q_

def searchPriorityBlocking(q):          # Searches for blocking tickets in the queue
    for ticket in q.tickets:
        if ticket.priority == 0:
            q.dequeue_ticket(ticket)
            return ticket
    return None

def searchPriorityUrgent(q):             # Searches for urgent tickets in the queue
    for ticket in q.tickets:
        if ticket.priority == 1:
            q.dequeue_ticket(ticket)
            return ticket
    return None

def CreateTicket(t):                    # Creates tickets
    return Ticket(random.choice(priorities), s1, t, random.choice(components))


class Simulation :

    def __init__ ( self , arrDist , servDist , nrServers):
        self.arrDist = arrDist
        self.servDist = servDist
        self.nrServers = nrServers

    def simulate ( self , T ):
        nrOfTickets = 8492
        nrStates = 6
        fes = FES()
        res = NetworkSimResults (nrStates , nrOfTickets)

        qs = [ None ] * nrQueues   #make them a property for s1 and tester1
        qs[0] = Queue()         #deque for open
        qs[1] = Queue()         #deque for analysed
        qs[2] = PriorityQueue()  #heapq with priority for implemented
        qs[3] = PriorityQueue()    #heapq with priority for verified
        #4 <- engineer

        qs[0].state = 0
        qs[1].state = 2
        qs[2].state = 4
        qs[3].state = 5
```

```

        #engineer service times


        t = 0 # current time
        a0 = self.arrDist.rvs()        # Create arrival of first ticket
        t0 = CreateTicket(t)            #ticket creation
        firstEvent = Event(Event.ARRIVAL, s1, t0, a0 )    #remove S_1 and put queue number  0 = open queue
        fes.add(firstEvent)

        fes.add(Event(Event.EOD, s1, t0, 8*3600))       # first end-of-day event
        day = 0
        tickets_in_system_day = []

        while t < T :
            e = fes.next()
            t = e.time
            c = e.ticket
            queueNr = c.pos

            tot_sch_queue = 0
            tot_accepted_queue = 0

            for i in engg:
                tot_sch_queue = tot_sch_queue + countSch(i)
                tot_accepted_queue = tot_accepted_queue + countAccepted(i)

            res.queueResults[0].registerQueueLength(t, len (qs[0].tickets))
            res.queueResults[1].registerQueueLength(t, tot_accepted_queue)
            res.queueResults[2].registerQueueLength(t, len (qs[1].tickets))
            res.queueResults[3].registerQueueLength(t, tot_sch_queue)
            res.queueResults[4].registerQueueLength(t, len (qs[2].tickets))
            res.queueResults[5].registerQueueLength(t, len (qs[3].tickets))


            if e.type == Event.ARRIVAL :


                #ALGORITHM 1 (ARRIVAL AT OPEN)
                if queueNr == 0:

                    qs[0].enqueue(c)

                    if s1.idle == True:
                        b0 = self.servDist[0].rvs()
                        dep = Event(Event.DEPARTURE, s1, c, t+b0)
                        fes.add(dep)
                        c.depEvent = dep    #ticket knows its departure and service time
                        qs[0].dequeue_ticket(c)
                        # print(e)
                        s1.idle = False
                        s1.ticket = c
                        c.state = 0
                        c.pos = 0
                        s1.openCounter = s1.openCounter + 1
                        s1.setWorking(t)
                    a1 = self.arrDist.rvs()
                    c1 = CreateTicket(t)
                    fes.add(Event(Event.ARRIVAL , s1 , c1, t+a1))

                #ALGORITHM 3 (ARRIVAL AT ACCEPTED)
                elif queueNr == 4 and c.state == 1:        #arrival at engineer's queue
                    i = c.engineer.number
                    engg[i].queue.enqueue(c)
```

```
                    serv = engg[i]


                    if serv.idle == True:
                        b1 = self.servDist[1].rvs()
                        dep = Event(Event.DEPARTURE, serv, c, t+b1)
                        fes.add(dep)
                        serv.ticket = c
                        c.depEvent = dep    #ticket knows its departure and service time
                        serv.idle = False
                        c.state = 1
                        c.pos = 4
                        c.engineer = serv
                        serv.queue.dequeue_ticket(c)
                        serv.setWorking(t)
                        serv.countAccept = serv.countAccept + 1

                    elif c.priority == 0 and c.depEvent == None:     #if ticket hasn't been assigned (since serv is busy) & has priority
                        #cancelling departure event of the ticket the engineer is working on if that ticket lower priority
                        c2 = serv.ticket              # current ticket he's working on
                        if c2 != None:
                            if c2.priority != 0 and c2.depEvent != None:  #  the current ticket has a priority of 0
                                c2.depEvent.iscancelled = True
                                c2.depEvent = None
                                serv.nrTickets.remove(serv.ticket.number)
                                remove_state_ticket(serv)
                                serv.nrTickets.append(c.number)
                                add_state_ticket(serv, c)
                                serv.queue.dequeue_ticket(c)
                                b1 = self.servDist[1].rvs()
                                dep = Event(Event.DEPARTURE, serv, c, t+b1)
                                fes.add(dep)
                                c.depEvent = dep    #ticket knows its departure and service time
                                serv.idle = False
                                serv.ticket = c
                                c.state = 1
                                c.pos = 4
                                serv.countAccept = serv.countAccept + 1

                #ALGORITHM 5 (ARRIVAL AT ANALYSED)
                elif queueNr == 1:
                    qs[1].enqueue(c)


                    if s1.idle == True:
                        b2 = self.servDist[2].rvs()
                        dep = Event(Event.DEPARTURE, s1, c, t+b2)
                        qs[1].dequeue_ticket(c)
                        fes.add(dep)
                        c.depEvent = dep    #ticket knows its departure and service time
                        s1.idle = False
                        s1.ticket = c
                        c.state = 2
                        c.pos = 1
                        s1.setWorking(t)
                        s1.analyseCounter = s1.analyseCounter + 1

                #ALGORITHM 7 (ARRIVAL AT SCHEDULED)
                elif queueNr == 4 and c.state == 3:
                    serv = c.engineer
```

```
                serv.queue.enqueue(c)


                if c.engineer.idle == True:
                    b2 = self.servDist[3].rvs()
                    serv.queue.dequeue_ticket(c)
                    dep = Event(Event.DEPARTURE, serv, c, t+b2)
                    fes.add(dep)
                    c.depEvent = dep    #ticket knows its departure and service time
                    serv.idle = False
                    c.state = 3
                    c.pos = 4
                    serv.ticket = c
                    c.engineer.setWorking(t)
                    serv.countSch = serv.countSch + 1


            #ALOGRITHM 9 (ARRIVAL AT IMPLEMENTED)
            elif queueNr == 2:

                qs[2].enqueue(c)


                if tester.idle == True:
                    b2 = self.servDist[4].rvs()
                    dep = Event(Event.DEPARTURE, tester, c, t+b2)
                    fes.add(dep)
                    c.depEvent = dep    #ticket knows its departure and service time
                    qs[2].dequeue_ticket(c)
                    tester.idle = False
                    c.state = 4
                    tester.ticket = c
                    c.tester = tester
                    c.pos = 2
                    tester.setWorking(t)

                else:

                    serv = assignFreeTester(c)
                    if serv != None:
                        b2 = self.servDist[4].rvs()
                        dep = Event(Event.DEPARTURE, serv, c, t+b2)
                        qs[2].dequeue_ticket(c)
                        fes.add(dep)
                        c.depEvent = dep    #ticket knows its departure and service time
                        serv.idle = False
                        c.state = 4
                        serv.ticket = c
                        c.tester = serv
                        c.pos = 2
                        serv.setWorking(t)
                        serv.countImplt = serv.countImplt + 1

                if c.priority == 0 and c.tester == None:        # if priority is 0 and no tester is assigned (since none are idle)

                    if tester.ticket != None and tester.ticket.priority != 0 :
                        c2 = tester.ticket      #current ticket he's working on
                        if c2.depEvent != None:
                            c2.depEvent.iscancelled = True
                            c2.depEvent = None
                            tester.nrTickets.remove(tester.ticket.number)
                            tester.nrTickets.append(c.number)
                            b2 = self.servDist[4].rvs()
```

```
                                dep = Event(Event.DEPARTURE, tester, c, t+b2)
                                fes.add(dep)
                                c.depEvent = dep    #ticket knows its departure and service time
                                qs[2].dequeue_ticket(c)
                                tester.idle = False
                                tester.ticket = c
                                c.state = 4
                                c.tester = tester
                                c.pos = 2

                        else:
                            i = assignTester(c)
                            if i != None:
                                c2 = i.ticket      #current ticket he's working on
                                if c2 != None and c2.priority != 0 and c2.depEvent != None:
                                    c2.depEvent.iscancelled = True
                                    c2.depEvent = None
                                    i.nrTickets.remove(i.ticket.number)
                                    i.nrTickets.append(c.number)
                                    remove_state_ticket(i)
                                    add_state_ticket(i, c)
                                    b2 = self.servDist[4].rvs()
                                    dep = Event(Event.DEPARTURE, i, c, t+b2)
                                    qs[2].dequeue_ticket(c)
                                    fes.add(dep)
                                    c.depEvent = dep    #ticket knows its departure and service time
                                    i.idle = False
                                    i.ticket = c
                                    c.state = 4
                                    c.tester = i
                                    c.pos = 2
                                    i.countImplt = i.countImplt + 1


                #ALGORITHM 11 (ARRIVAL AT VERIFIED)
                elif queueNr == 3:
                    qs[3].enqueue(c)


                    if s1.idle == True:
                        b3 = self.servDist[5].rvs()
                        dep = Event(Event.DEPARTURE, s1, c, t+b3)
                        qs[3].dequeue_ticket(c)
                        fes.add(dep)
                        c.depEvent = dep    #ticket knows its departure and service time
                        s1.idle = False
                        s1.ticket = c
                        c.state = 5
                        c.pos = 3
                        s1.setWorking(t)

                    else:
                        # print(e)
                        if len(qs[3].tickets) > 0:
                            if c.priority == 0:
                                c2 = s1.ticket
                                if c2 != None:
                                    if c2.priority != 0 and c2.depEvent!= None:
                                        c2.depEvent.iscancelled = True
                                        c2.depEvent = None
                                        s1.nrTickets.remove(s1.ticket.number)
                                        s1.nrTickets.append(c.number)
                                        b2 = self.servDist[5].rvs()
```

```
                                    dep = Event(Event.DEPARTURE, s1, c, t+b2)
                                    qs[3].dequeue_ticket(c)
                                    fes.add(dep)
                                    c.depEvent = dep    #ticket knows its departure and service time
                                    s1.idle = False
                                    s1.ticket = c
                                    c.state = 5
                                    c.pos = 3


        elif e.type == Event.DEPARTURE :
            if e.iscancelled == False:
                c.depEvent = None

                #ALGORITHM 2 (DEPARTURE FROM OPEN)
                if queueNr == 0:

                    for i in engg:
                        for comp in i.component:
                            if comp == c.component:
                                c.engineer = i
                                fes.add(Event(Event.ARRIVAL, i, c, t))  # schedule arrival time
                                res.registerWaitingTime (0, t - c.arrivalTime )
                                sojournTimes_Components.loc[len(sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
                                c.newPos(1, t)
                                c.pos = 4
                                break

                    # Once server stops working, set to idle
                    s1.setIdle(t)
                    s1.idle = True

                    # S1 WORKFLOW
                    if s1.openCounter <= 5 and len(qs[0].tickets) > 0:
                        c2 = qs[0].dequeue_0()
                        b0 = self.servDist[0].rvs()
                        dep = Event(Event.DEPARTURE, s1, c2, t+b0)
                        c2.depEvent = dep    #ticket knows its departure and service time
                        fes.add(dep)
                        s1.idle = False
                        s1.ticket = c2
                        c2.server = s1
                        c2.state = 0
                        c2.pos = 0
                        s1.openCounter += 1
                        s1.setWorking(t)

                    else:
                        c1 = searchPriorityBlocking(qs[3])
                        c3 = searchPriorityUrgent(qs[3])

                        if c1 != None:

                            if s1.idle == True:
                                b3 = self.servDist[5].rvs()
                                dep = Event(Event.DEPARTURE, s1, c1, t+b3)
                                fes.add(dep)
                                c1.depEvent = dep    #ticket knows its departure and service time
                                s1.idle = False
                                s1.ticket = c1
                                c1.state = 5    #goes into verified state
                                c1.pos = 3
```

```
                                s1.setWorking(t)

                            else:
                                if s1.ticket.priority != 0 and s1.ticket.depEvent != None:
                                    s1.ticket.depEvent.iscancelled = True
                                    s1.ticket.depEvent = None
                                    s1.nrTickets.remove(s1.ticket.number)
                                    s1.nrTickets.append(c1.number)
                                    b3 = self.servDist[5].rvs()
                                    dep = Event(Event.DEPARTURE, s1, c1, t+b3)
                                    fes.add(dep)
                                    c1.depEvent = dep    #ticket knows its departure and service time
                                    s1.idle = False
                                    s1.ticket = c1
                                    c1.state = 5    #goes into verified state
                                    c1.pos = 3

                        elif c3 != None:
                            if s1.idle == True:
                                b3 = self.servDist[5].rvs()
                                dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                fes.add(dep)
                                c3.depEvent = dep    #ticket knows its departure and service time
                                s1.idle = False
                                c3.state = 5    #goes into verified state
                                c3.pos = 3
                                s1.ticket = c3
                                s1.setWorking(t)

                            else:
                                if s1.ticket.priority != 0 and s1.ticket.priority != 1 and serv.ticket.depEvent != None:
                                    s1.ticket.depEvent.iscancelled = True
                                    s1.ticket.depEvent = None
                                    s1.nrTickets.remove(s1.ticket.number)
                                    s1.nrTickets.append(c3.number)
                                    b3 = self.servDist[5].rvs()
                                    dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                    fes.add(dep)
                                    c3.depEvent = dep    #ticket knows its departure and service time
                                    s1.idle = False
                                    c3.state = 5    #goes into verified state
                                    c3.pos = 3
                                    s1.ticket = c3

                        else:
                            if s1.idle == True:

                                if len(qs[3].tickets) > 0:
                                    c3 = qs[3].dequeue_0()
                                    b3 = self.servDist[5].rvs()
                                    dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                    fes.add(dep)
                                    c3.depEvent = dep    #ticket knows its departure and service time
                                    s1.ticket = c3
                                    c3.pos = 3
                                    c3.state = 5
                                    s1.idle = False
                                    s1.setWorking(t)

                                elif len(qs[2].tickets) > 0:

```

```
                                    if s1.impltCounter < 2:
                                        c3 = qs[2].dequeue_0()  #ticket is not being served
                                        b3 = self.servDist[4].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        s1.ticket = c3
                                        c3.tester = s1
                                        s1.idle = False
                                        c3.state = 4    #goes into implemented state
                                        c3.pos = 2
                                        s1.impltCounter += 1
                                        s1.setWorking(t)

                                elif len(qs[1].tickets) > 0:

                                    if s1.analyseCounter < 4:
                                        c3 = qs[1].dequeue_0()
                                        b3 = self.servDist[2].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        s1.ticket = c3
                                        c3.state = 2    #goes into analysed state
                                        c3.pos = 1
                                        s1.idle = False
                                        s1.analyseCounter += 1
                                        s1.setWorking(t)

                                elif len(qs[2].tickets) > 0  and len(qs[2].tickets) < 20:

                                    #arrival at implemented
                                    if s1.impltCounter < 2:
                                        c3 = qs[2].dequeue_0()
                                        b3 = self.servDist[4].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        s1.idle = False
                                        s1.ticket = c3
                                        c3.tester = s1
                                        c3.state = 4    #goes into implemented state
                                        c3.pos = 2
                                        s1.impltCounter += 1
                                        s1.setWorking(t)


                #ALGORITHM 4 (DEPARTURE FROM ACCEPTED - ENGINEER QUEUE)
                elif queueNr == 4 and c.state == 1:  #engineer's queue

                    if c.priority == 0:
                        fes.add(Event(Event.ARRIVAL, c.engineer, c, t))  # schedule arrival at implemented. c.engineer SHOULD NOT work on testing
                        res.registerWaitingTime (1, t - c.arrivalTime)
                        sojournTimes_Components.loc[len(sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
```

```
                        c.newPos(4, t)
                        c.pos = 2

                    else:

                        if c.probability < p:    #sample p from a bernoulli dist. - c.probability is probability of NOT being sent back due to insufficient information
                            fes.add(Event(Event.ARRIVAL, s1, c, t))  #schedule arrival at analysed
                            res.registerWaitingTime (1, t - c.arrivalTime)
                            sojournTimes_Components.loc[len(sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
                            c.newPos(2, t)
                            c.pos = 1

                        else:
                            fes.add(Event(Event.ARRIVAL, s1, c, t))  #schedule arrival at open
                            res.registerWaitingTime (1, t - c.arrivalTime)
                            sojournTimes_Components.loc[len(sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
                            c.newPos(0, t)
                            c.pos = 0

                    # Once server stops working, set to idle
                    c.engineer.idle = True
                    c.engineer.setIdle(t)

                    # Engineer's Workflow
                    serv = c.engineer
                    c3 = searchPriorityBlocking(serv.queue)
                    c4 = searchPriorityUrgent(serv.queue)

                    if c3 != None:
                        if serv.idle == True:
                            s = c3.state
                            b3 = self.servDist[s].rvs()
                            dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                            fes.add(dep)
                            c3.depEvent = dep    #ticket knows its departure and service time
                            serv.idle = False
                            serv.ticket = c3
                            c3.state = s    #goes into next state
                            c3.pos = assignPos(c3)
                            serv.setWorking(t)

                        else:
                            if serv.ticket.priority != 0 and serv.ticket.depEvent != None:
                                serv.ticket.depEvent.iscancelled = True
                                serv.ticket.depEvent = None
                                serv.nrTickets.remove(serv.ticket.number)
                                remove_state_ticket(serv)
                                add_state_ticket(serv, c3)
                                serv.nrTickets.append(c3.number)
                                s = c3.state
                                b3 = self.servDist[s].rvs()
                                dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                fes.add(dep)
                                c3.depEvent = dep    #ticket knows its departure and service time
                                serv.idle = False
                                serv.ticket = c3
                                c3.state = s    #goes into next state
                                c3.pos = assignPos(c3)
```

```

                    elif c4 != None:
                        if serv.idle == True:
                            s = c4.state
                            b3 = self.servDist[s].rvs()
                            dep = Event(Event.DEPARTURE, serv, c4, t+b3)
                            fes.add(dep)
                            c4.depEvent = dep    #ticket knows its departure and service time
                            serv.idle = False
                            serv.ticket = c4
                            c4.state = s    #goes into next state
                            c4.pos = assignPos(c4)
                            serv.setWorking(t)

                        else:
                            if serv.ticket.priority !=0 and serv.ticket.priority != 1 and serv.ticket.depEvent != None:
                                serv.ticket.depEvent.iscancelled = True
                                serv.ticket.depEvent = None
                                serv.nrTickets.remove(serv.ticket.number)
                                serv.nrTickets.append(c4.number)
                                remove_state_ticket(serv)
                                add_state_ticket(serv, c4)
                                s = c4.state
                                b3 = self.servDist[s].rvs()
                                dep = Event(Event.DEPARTURE, serv, c4, t+b3)
                                fes.add(dep)
                                c4.depEvent = dep    #ticket knows its departure and service time
                                serv.idle = False
                                serv.ticket = c4
                                c4.state = s    #goes into next state
                                c4.pos = assignPos(c4)

                    else:
                        if serv.idle == True:
                            if countAccepted(serv) > 0:
                                if serv.countAccept < 2:
                                    q_ = queueAccepted(serv)
                                    c3 = q_[0]
                                    serv.queue.dequeue_ticket(c3)
                                    b3 = self.servDist[1].rvs()
                                    dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                    fes.add(dep)
                                    c3.depEvent = dep    #ticket knows its departure and service time
                                    serv.idle = False
                                    serv.ticket = c3
                                    c3.state = 1    #goes into accepted state
                                    c3.pos = 4
                                    serv.countAccept += 1
                                    serv.setWorking(t)

                            elif len(qs[2].tickets) > 0:
                                if serv.countImplt < 2:
                                    q_ = queueImplt(qs[2], serv)
                                    if len(q_) != 0:
                                        c3 = q_[0]
                                        qs[2].dequeue_ticket(c3)
                                        b3 = self.servDist[4].rvs()
                                        dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
```

```
                                        serv.idle = False
                                        serv.ticket = c3
                                        c3.tester = serv
                                        c3.state = 4    #goes into implemented state
                                        c3.pos = 2
                                        serv.countImplt += 1
                                        serv.setWorking(t)

                            elif countSch(serv) > 0:
                                if serv.countSch < 2:
                                    q_ = queueSch(serv)
                                    c3 = q_[0]
                                    serv.queue.dequeue_ticket(c3)
                                    b3 = self.servDist[3].rvs()
                                    dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                    fes.add(dep)
                                    c3.depEvent = dep    #ticket knows its departure and service time
                                    serv.idle = False
                                    serv.ticket = c3
                                    c3.state = 3    #goes into scheduled state
                                    c3.pos = 4
                                    serv.countSch += 1
                                    serv.setWorking(t)


                #ALGORITHM 6 (DEPARTURE FROM ANALYSED)
                elif queueNr == 1:

                    serv = c.engineer
                    fes.add(Event(Event.ARRIVAL, serv, c, t))  #schedule arrival at engineer's queue.
                    res.registerWaitingTime (2, t - c.arrivalTime)
                    sojournTimes_Components.loc[len(sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
                    c.pos = 4
                    c.newPos(3, t)

                    # Once server stops working, set to idle
                    s1.idle = True
                    s1.setIdle(t)

                    # S1 WORKFLOW
                    if s1.openCounter <= 5 and len(qs[0].tickets) > 0:

                        c2 = qs[0].dequeue_0()
                        b0 = self.servDist[0].rvs()
                        dep = Event(Event.DEPARTURE, s1, c2, t+b0)
                        c2.depEvent = dep    #ticket knows its departure and service time
                        fes.add(dep)
                        s1.idle = False
                        s1.ticket = c2
                        c2.server = s1
                        c2.state = 0
                        c2.pos = 0
                        s1.openCounter += 1
                        s1.setWorking(t)


                    else:
                        c1 = searchPriorityBlocking(qs[3])
                        c3 = searchPriorityUrgent(qs[3])
                        if c1 != None:

```

```
                                if s1.idle == True:
                                    b3 = self.servDist[5].rvs()
                                    dep = Event(Event.DEPARTURE, s1, c1, t+b3)
                                    fes.add(dep)
                                    c1.depEvent = dep    #ticket knows its departure and service time
                                    s1.idle = False
                                    s1.ticket = c1
                                    c1.state = 5    #goes into verified state
                                    c1.pos = 3
                                    s1.setWorking(t)

                                else:
                                    if s1.ticket.priority != 0:
                                        s1.ticket.depEvent.iscancelled = True
                                        s1.ticket.depEvent = None
                                        s1.nrTickets.remove(s1.ticket.number)
                                        s1.nrTickets.append(c1.number)
                                        b3 = self.servDist[5].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c1, t+b3)
                                        fes.add(dep)
                                        c1.depEvent = dep    #ticket knows its departure and service time
                                        s1.idle = False
                                        s1.ticket = c1
                                        c1.state = 5    #goes into verified state
                                        c1.pos = 3


                            elif c3 != None:
                                if s1.idle == True:
                                    b3 = self.servDist[5].rvs()
                                    dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                    fes.add(dep)
                                    c3.depEvent = dep    #ticket knows its departure and service time
                                    s1.idle = False
                                    c3.state = 5    #goes into verified state
                                    c3.pos = 3
                                    s1.ticket = c3
                                    s1.setWorking(t)

                                else:
                                    if s1.ticket.priority != 0 and s1.ticket.priority != 1:
                                        s1.ticket.depEvent.iscancelled = True
                                        s1.ticket.depEvent = None
                                        s1.nrTickets.remove(s1.ticket.number)
                                        s1.nrTickets.append(c3.number)
                                        b3 = self.servDist[5].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        s1.idle = False
                                        c3.state = 5
                                        c3.pos = 3
                                        s1.ticket = c3
                                        # s1.setWorking(t)

                            else:
                                if s1.idle == True:

                                    if len(qs[3].tickets) > 0:
                                        c3 = qs[3].dequeue_0()
                                        b3 = self.servDist[5].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c3, t+b3)
```

```
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        s1.ticket = c3
                                        c3.pos = 3
                                        c3.state = 5
                                        s1.idle = False
                                        s1.setWorking(t)


                                    elif len(qs[2].tickets) > 0:

                                        if s1.impltCounter < 2:
                                            c3 = qs[2].dequeue_0()  #ticket is not being served
                                            b3 = self.servDist[4].rvs()
                                            dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                            fes.add(dep)
                                            c3.depEvent = dep    #ticket knows its departure and service time
                                            s1.ticket = c3
                                            c3.tester = s1
                                            s1.idle = False
                                            c3.state = 4    #goes into implemented state
                                            c3.pos = 2
                                            s1.impltCounter += 1
                                            s1.setWorking(t)


                                    elif len(qs[1].tickets) > 0:

                                        if s1.analyseCounter < 4:
                                            c3 = qs[1].dequeue_0()
                                            b3 = self.servDist[2].rvs()
                                            dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                            fes.add(dep)
                                            c3.depEvent = dep    #ticket knows its departure and service time
                                            s1.ticket = c3
                                            c3.state = 2    #goes into analysed state
                                            c3.pos = 1
                                            s1.idle = False
                                            s1.analyseCounter += 1
                                            s1.setWorking(t)

                                    elif len(qs[2].tickets) > 0  and len(qs[2].tickets) < 20:
                                        if s1.impltCounter < 2:
                                            c3 = qs[2].dequeue_0()
                                            b3 = self.servDist[4].rvs()
                                            dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                            fes.add(dep)
                                            c3.depEvent = dep    #ticket knows its departure and service time
                                            s1.idle = False
                                            s1.ticket = c3
                                            c3.tester = s1
                                            c3.state = 4    #goes into implemented state
                                            c3.pos = 2
                                            s1.impltCounter += 1
                                            s1.setWorking(t)
```

```python


                #ALGORITHM 8 (DEPARTURE FROM SCHEDULED)
                elif queueNr == 4 and c.state == 3:
                    fes.add(Event(Event.ARRIVAL, tester, c, t))  #schedule arrival at implemented
                    res.registerWaitingTime (3, t - c.arrivalTime)
                    sojournTimes_Components.loc[len(sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
                    c.newPos(4, t)
                    c.pos = 2

                    # Once server stops working, set to idle
                    c.engineer.idle = True
                    c.engineer.setIdle(t)

                    # Engineer's decision tree
                    serv = c.engineer
                    c3 = searchPriorityBlocking(serv.queue)
                    c4 = searchPriorityUrgent(serv.queue)
                    if c3 != None:
                        if serv.idle == True:
                            s = c3.state
                            b3 = self.servDist[s].rvs()
                            dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                            fes.add(dep)
                            c3.depEvent = dep    #ticket knows its departure and service time
                            serv.idle = False
                            serv.ticket = c3
                            c3.state = s    #goes into next state
                            c3.pos = assignPos(c3)
                            serv.setWorking(t)

                        else:
                            if serv.ticket.priority != 0 and serv.ticket.depEvent != None:
                                serv.ticket.depEvent.iscancelled = True
                                serv.ticket.depEvent = None
                                remove_state_ticket(serv)
                                serv.nrTickets.remove(serv.ticket.number)
                                serv.nrTickets.append(c3.number)
                                add_state_ticket(serv, c3)
                                s = c3.state
                                b3 = self.servDist[s].rvs()
                                dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                fes.add(dep)
                                c3.depEvent = dep    #ticket knows its departure and service time
                                serv.idle = False
                                serv.ticket = c3
                                c3.state = s    #goes into next state
                                c3.pos = assignPos(c3)

                    elif c4 != None:
                        if serv.idle == True:
                            s = c4.state
                            b3 = self.servDist[s].rvs()
                            dep = Event(Event.DEPARTURE, serv, c4, t+b3)
                            fes.add(dep)
                            c4.depEvent = dep    #ticket knows its departure and service time
                            serv.idle = False
                            serv.ticket = c4
                            c4.state = s    #goes into next state
                            c4.pos = assignPos(c4)
                            serv.setWorking(t)
```

```
                        else:
                            if serv.ticket.priority != 0 and serv.ticket.priority != 1 and serv.ticket.depEvent != None:
                                serv.ticket.depEvent.iscancelled = True
                                serv.ticket.depEvent = None
                                remove_state_ticket(serv)
                                serv.nrTickets.remove(serv.ticket.number)
                                serv.nrTickets.append(c4.number)
                                add_state_ticket(serv, c4)
                                s = c4.state
                                b3 = self.servDist[s].rvs()
                                dep = Event(Event.DEPARTURE, serv, c4, t+b3)
                                fes.add(dep)
                                c4.depEvent = dep    #ticket knows its departure and service time
                                serv.idle = False
                                serv.ticket = c4
                                c4.state = s    #goes into next state
                                c4.pos = assignPos(c4)

                    else:
                        if serv.idle == True:
                            if countAccepted(serv) > 0:
                                if serv.countAccept < 2:
                                    q_ = queueAccepted(serv)
                                    c3 = q_[0]
                                    serv.queue.dequeue_ticket(c3)
                                    b3 = self.servDist[1].rvs()
                                    dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                    fes.add(dep)
                                    c3.depEvent = dep    #ticket knows its departure and service time
                                    serv.idle = False
                                    serv.ticket = c3
                                    c3.state = 1    #goes into accepted state
                                    c3.pos = 4
                                    serv.countAccept += 1
                                    serv.setWorking(t)

                            elif len(qs[2].tickets) > 0:
                                if serv.countImplt < 2:
                                    q_ = queueImplt(qs[2], serv)
                                    if len(q_) != 0:
                                        c3 = q_[0]
                                        qs[2].dequeue_ticket(c3)
                                        b3 = self.servDist[4].rvs()
                                        dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        serv.idle = False
                                        serv.ticket = c3
                                        c3.tester = serv
                                        c3.state = 4    #goes into implemented state
                                        c3.pos = 2
                                        serv.countImplt += 1
                                        serv.setWorking(t)

                            elif countSch(serv) > 0:
                                if serv.countSch < 2:
                                    q_ = queueSch(serv)
                                    c3 = q_[0]
                                    serv.queue.dequeue_ticket(c3)
```

```
                                    b3 = self.servDist[3].rvs()
                                    dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                    fes.add(dep)
                                    c3.depEvent = dep    #ticket knows its departure and service time
                                    serv.idle = False
                                    serv.ticket = c3
                                    c3.state = 3    #goes into scheduled state
                                    c3.pos = 4
                                    serv.countSch += 1
                                    serv.setWorking(t)


                #ALGORITHM 10 (DEPARTURE FROM IMPLEMENTED)
                elif queueNr == 2:

                    if c.testProb > test_p:                                    # testing negative
                        fes.add(Event(Event.ARRIVAL, c.engineer, c, t))  # schedule arrival at engineer's queue - scheduled for the engineer again.
                        res.registerWaitingTime (4, t - c.arrivalTime)
                        sojournTimes_Components.loc[len(sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
                        c.newPos(3, t)
                        c.pos = 4
                        #break

                    else:                                                   #testing positive
                        fes.add(Event(Event.ARRIVAL, s1, c, t))  #schedule arrival at for S1 queue - goes to verified.
                        res.registerWaitingTime (4, t - c.arrivalTime)
                        sojournTimes_Components.loc[len(sojournTimes_Components.index), c.component] = (t-c.arrivalTime)/3600
                        c.newPos(5, t)
                        c.pos = 3

                    # Once server stops working, set to idle
                    c.tester.idle = True
                    c.tester.setIdle(t)

                    # Workflows
                    if c.tester == tester:
                        c3 = searchPriorityBlocking(qs[2])
                        c4 = searchPriorityUrgent(qs[2])

                        # TESTER WORKFLOW
                        if c3 != None:
                            if tester.idle == True:
                                b3 = self.servDist[4].rvs()
                                dep = Event(Event.DEPARTURE, tester, c3, t+b3)
                                fes.add(dep)
                                c3.depEvent = dep    #ticket knows its departure and service time
                                tester.idle = False
                                c3.tester = tester
                                tester.ticket = c3
                                c3.state = 4
                                c3.pos = 2
                                tester.setWorking(t)

                            else:
                                if tester.ticket.priority != 0:
                                    tester.ticket.depEvent.iscancelled = True
                                    tester.ticket.depEvent = None
```

```
                                        tester.nrTickets.remove(tester.ticket.number)
                                        tester.nrTickets.append(c3.number)
                                        b3 = self.servDist[4].rvs()
                                        dep = Event(Event.DEPARTURE, tester, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        tester.idle = False
                                        c3.tester = tester
                                        tester.ticket = c3
                                        c3.state = 4
                                        c3.pos = 2


                            elif c4 != None:
                                if tester.idle == True:
                                    b3 = self.servDist[4].rvs()
                                    dep = Event(Event.DEPARTURE, tester, c4, t+b3)
                                    fes.add(dep)
                                    c4.depEvent = dep    #ticket knows its departure and service time
                                    tester.idle = False
                                    tester.ticket = c4
                                    c4.tester = tester
                                    c4.state = 4
                                    c4.pos = 2
                                    tester.setWorking(t)

                                else:
                                    if tester.ticket.priority != 0 and tester.ticket.priority != 1:
                                        tester.ticket.depEvent.iscancelled = True
                                        tester.ticket.depEvent = None
                                        tester.nrTickets.remove(tester.ticket.number)
                                        tester.nrTickets.append(c4.number)
                                        b3 = self.servDist[4].rvs()
                                        dep = Event(Event.DEPARTURE, tester, c4, t+b3)
                                        fes.add(dep)
                                        c4.depEvent = dep    #ticket knows its departure and service time
                                        tester.idle = False
                                        c4.tester = tester
                                        tester.ticket = c4
                                        c4.state = 4
                                        c4.pos = 2

                            elif len(qs[2].tickets) > 0 and tester.idle == True:
                                c3 = qs[2].dequeue_0()
                                b3 = self.servDist[4].rvs()
                                dep = Event(Event.DEPARTURE, tester, c3, t+b3)
                                fes.add(dep)
                                c3.depEvent = dep    #ticket knows its departure and service time
                                tester.ticket = c3
                                tester.idle = False
                                c3.tester = tester
                                c3.state = 4
                                c3.pos = 2
                                tester.setWorking(t)


                        elif c.tester in engg:
                            serv = c.tester
```

```

                            # Engineer's Workflow
                            c3 = searchPriorityBlocking(serv.queue)
                            c4 = searchPriorityUrgent(serv.queue)
                            if c3 != None:
                                if serv.idle == True:
                                    s = c3.state
                                    b3 = self.servDist[s].rvs()
                                    dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                    fes.add(dep)
                                    c3.depEvent = dep    #ticket knows its departure and service time
                                    serv.idle = False
                                    serv.ticket = c3
                                    c3.state = s
                                    c3.pos = assignPos(c3)
                                    serv.setWorking(t)

                                else:
                                    if serv.ticket.priority != 0 and serv.ticket.depEvent != None:
                                        serv.ticket.depEvent.iscancelled = True
                                        serv.ticket.depEvent = None
                                        remove_state_ticket(serv)
                                        serv.nrTickets.remove(serv.ticket.number)
                                        serv.nrTickets.append(c3.number)
                                        add_state_ticket(serv, c3)
                                        s = c3.state
                                        b3 = self.servDist[s].rvs()
                                        dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        serv.idle = False
                                        serv.ticket = c3
                                        c3.state = s
                                        c3.pos = assignPos(c3)

                            elif c4 != None:
                                if serv.idle == True:
                                    s = c4.state
                                    b3 = self.servDist[s].rvs()
                                    dep = Event(Event.DEPARTURE, serv, c4, t+b3)
                                    fes.add(dep)
                                    c4.depEvent = dep    #ticket knows its departure and service time
                                    serv.idle = False
                                    serv.ticket = c4
                                    c4.state = s
                                    c4.pos = assignPos(c4)
                                    serv.setWorking(t)

                                else:
                                    if serv.ticket.priority != 0 and serv.ticket.priority != 1 and serv.ticket.depEvent != None:
                                        serv.ticket.depEvent.iscancelled = True
                                        serv.ticket.depEvent = None
                                        remove_state_ticket(serv)
                                        serv.nrTickets.remove(serv.ticket.number)
                                        serv.nrTickets.append(c4.number)
                                        add_state_ticket(serv, c4)
                                        s = c4.state
                                        b3 = self.servDist[s].rvs()
                                        dep = Event(Event.DEPARTURE, serv, c4, t+b3)
                                        fes.add(dep)
```

```
                                        c4.depEvent = dep    #ticket knows its departure and service time
                                        serv.idle = False
                                        serv.ticket = c4
                                        c4.state = s
                                        c4.pos = assignPos(c4)


                            else:
                                if serv.idle == True:
                                    if countAccepted(serv) > 0:
                                        if serv.countAccept < 2:
                                            q_ = queueAccepted(serv)
                                            c3 = q_[0]
                                            serv.queue.dequeue_ticket(c3)
                                            b3 = self.servDist[1].rvs()
                                            dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                            fes.add(dep)
                                            c3.depEvent = dep    #ticket knows its departure and service time
                                            serv.idle = False
                                            serv.ticket = c3
                                            c3.state = 1    #goes into accepted state
                                            c3.pos = 4
                                            serv.countAccept += 1
                                            serv.setWorking(t)

                                    elif len(qs[2].tickets) > 0:
                                        if serv.countImplt < 2:
                                            q_ = queueImplt(qs[2], serv)
                                            if len(q_) != 0:
                                                c3 = q_[0]
                                                qs[2].dequeue_ticket(c3)
                                                b3 = self.servDist[4].rvs()
                                                dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                                fes.add(dep)
                                                c3.depEvent = dep    #ticket knows its departure and service time
                                                serv.idle = False
                                                serv.ticket = c3
                                                c3.tester = serv
                                                c3.state = 4    #goes into implemented state
                                                c3.pos = 2
                                                serv.countImplt += 1
                                                serv.setWorking(t)

                                    elif countSch(serv) > 0:
                                        if serv.countSch < 2:
                                            q_ = queueSch(serv)
                                            c3 = q_[0]
                                            serv.queue.dequeue_ticket(c3)
                                            b3 = self.servDist[3].rvs()
                                            dep = Event(Event.DEPARTURE, serv, c3, t+b3)
                                            fes.add(dep)
                                            c3.depEvent = dep    #ticket knows its departure and service time
                                            serv.idle = False
                                            serv.ticket = c3
                                            c3.state = 3    #goes into scheduled state
                                            c3.pos = 4
                                            serv.countSch += 1
```

```
                                        serv.setWorking(t)


                    elif c.tester == s1:

                        # S1 Workflow

                        if s1.openCounter <= 5 and len(qs[0].tickets) > 0:
                            c2 = qs[0].dequeue_0()
                            b0 = self.servDist[0].rvs()
                            dep = Event(Event.DEPARTURE, s1, c2, t+b0)
                            c2.depEvent = dep    #ticket knows its departure and service time
                            fes.add(dep)
                            s1.idle = False
                            s1.ticket = c2
                            c2.server = s1
                            c2.state = 0
                            c2.pos = 0
                            s1.setWorking(t)
                            s1.openCounter += 1

                        else:

                            c1 = searchPriorityBlocking(qs[3])
                            c3 = searchPriorityUrgent(qs[3])

                            if c1 != None:

                                if s1.idle == True:
                                    b3 = self.servDist[5].rvs()
                                    dep = Event(Event.DEPARTURE, s1, c1, t+b3)
                                    fes.add(dep)
                                    c1.depEvent = dep    #ticket knows its departure and service time
                                    s1.idle = False
                                    s1.ticket = c1
                                    c1.state = 5
                                    c1.pos = 3
                                    s1.setWorking(t)

                                else:
                                    if s1.ticket.priority != 0:
                                        s1.ticket.depEvent.iscancelled = True
                                        s1.ticket.depEvent = None
                                        s1.nrTickets.remove(s1.ticket.number)
                                        s1.nrTickets.append(c1.number)
                                        b3 = self.servDist[5].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c1, t+b3)
                                        fes.add(dep)
                                        c1.depEvent = dep    #ticket knows its departure and service time
                                        s1.idle = False
                                        s1.ticket = c1
                                        c1.state = 5
                                        c1.pos = 3


                            elif c3 != None:
                                if s1.idle == True:
                                    b3 = self.servDist[5].rvs()
                                    dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                    fes.add(dep)
                                    c3.depEvent = dep    #ticket knows its departure and service time
                                    s1.idle = False
```

```
                                    c3.state = 5
                                    c3.pos = 3
                                    s1.ticket = c3
                                    s1.setWorking(t)

                                else:
                                    if s1.ticket.priority != 0 and s1.ticket.priority != 1 and serv.ticket.depEvent != None:
                                        s1.ticket.depEvent.iscancelled = True
                                        s1.ticket.depEvent = None
                                        s1.nrTickets.remove(s1.ticket.number)
                                        s1.nrTickets.append(c3.number)
                                        b3 = self.servDist[5].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        s1.idle = False
                                        c3.state = 5
                                        c3.pos = 3
                                        s1.ticket = c3

                            else:
                                if s1.idle == True:
                                    if len(qs[3].tickets) > 0:
                                        c3 = qs[3].dequeue_0()
                                        b3 = self.servDist[5].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        s1.ticket = c3
                                        c3.pos = 3
                                        c3.state = 5
                                        s1.idle = False
                                        s1.setWorking(t)


                                    elif len(qs[2].tickets) > 0:
                                        if s1.impltCounter < 2:
                                            c3 = qs[2].dequeue_0()  #ticket is not being served
                                            b3 = self.servDist[4].rvs()
                                            dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                            fes.add(dep)
                                            c3.depEvent = dep    #ticket knows its departure and service time
                                            s1.ticket = c3
                                            c3.tester = s1
                                            s1.idle = False
                                            c3.state = 4    #goes into implemented state
                                            c3.pos = 2
                                            s1.impltCounter += 1
                                            s1.setWorking(t)


                                    elif len(qs[1].tickets) > 0:

                                        if s1.analyseCounter < 4:
                                            c3 = qs[1].dequeue_0()
                                            b3 = self.servDist[2].rvs()
                                            dep = Event(Event.DEPARTURE, s1, c3, t+b3)
```

```
                                            fes.add(dep)
                                            c3.depEvent = dep    #ticket knows its departure and service time
                                            s1.ticket = c3
                                            c3.state = 2    #goes into analysed state
                                            c3.pos = 1
                                            s1.idle = False
                                            s1.analyseCounter += 1
                                            s1.setWorking(t)

                                    elif len(qs[2].tickets) > 0  and len(qs[2].tickets) < 20:
                                        if s1.impltCounter < 2:
                                            c3 = qs[2].dequeue_0()
                                            b3 = self.servDist[4].rvs()
                                            dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                            fes.add(dep)
                                            c3.depEvent = dep    #ticket knows its departure and service time
                                            s1.idle = False
                                            s1.ticket = c3
                                            c3.tester = s1
                                            c3.state = 4    #goes into implemented state
                                            c3.pos = 2
                                            s1.impltCounter += 1
                                            s1.setWorking(t)


                #ALGORITHM 12 (DEPARTURE FROM VERIFIED)
                elif queueNr == 3:
                    # print("ALGORITHM 12")
                    res.registerWaitingTime (5, t - c.arrivalTime)
                    c.leaveSystem(t)
                    res.registerSojournTime(t - c.systemArrivalTime)
                    storeSojournTimes_priorities(c, (t-c.systemArrivalTime))

                    # Different results
                    sojournTimes_Components.loc[len(sojournTimes_Components.index), c.component] = (t-c.arrivalTime)
                    totalTimes_Components.loc[len(totalTimes_Components.index), c.component] = (t-c.systemArrivalTime)

                    TotalTicketsNr.append(c.number)
                    tickets_in_system_day.append(c.number)

                    # Once server stops working, set to idle
                    s1.idle = True
                    s1.setIdle(t)


                    # S1 WORKFLOW

                    if s1.openCounter <= 5 and len(qs[0].tickets) > 0:

                        c2 = qs[0].dequeue_0()
                        b0 = self.servDist[0].rvs()
                        dep = Event(Event.DEPARTURE, s1, c2, t+b0)
                        c2.depEvent = dep    #ticket knows its departure and service time
                        fes.add(dep)
                        s1.idle = False
                        s1.ticket = c2
                        c2.server = s1
                        c2.state = 0
```

```
                            c2.pos = 0
                            s1.openCounter += 1
                            s1.setWorking(t)

                        else:

                            c1 = searchPriorityBlocking(qs[3])
                            c3 = searchPriorityUrgent(qs[3])

                            if c1 != None:

                                if s1.idle == True:
                                    b3 = self.servDist[5].rvs()
                                    dep = Event(Event.DEPARTURE, s1, c1, t+b3)
                                    fes.add(dep)
                                    c1.depEvent = dep    #ticket knows its departure and service time
                                    s1.idle = False
                                    s1.ticket = c1
                                    c1.state = 5
                                    c1.pos = 3
                                    s1.setWorking(t)

                                else:
                                    if s1.ticket.priority != 0:
                                        s1.ticket.depEvent.iscancelled = True
                                        s1.ticket.depEvent = None
                                        s1.nrTickets.remove(s1.ticket.number)
                                        s1.nrTickets.append(c1.number)
                                        b3 = self.servDist[5].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c1, t+b3)
                                        fes.add(dep)
                                        c1.depEvent = dep    #ticket knows its departure and service time
                                        s1.idle = False
                                        s1.ticket = c1
                                        c1.state = 5
                                        c1.pos = 3


                            elif c3 != None:
                                if s1.idle == True:
                                    b3 = self.servDist[5].rvs()
                                    dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                    fes.add(dep)
                                    c3.depEvent = dep    #ticket knows its departure and service time
                                    s1.idle = False
                                    c3.state = 5
                                    c3.pos = 3
                                    s1.ticket = c3
                                    s1.setWorking(t)

                                else:
                                    if s1.ticket.priority != 0 and s1.ticket.priority != 1:
                                        s1.ticket.depEvent.iscancelled = True
                                        s1.ticket.depEvent = None
                                        s1.nrTickets.remove(s1.ticket.number)
                                        s1.nrTickets.append(c3.number)
                                        b3 = self.servDist[5].rvs()
                                        dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                                        fes.add(dep)
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        s1.idle = False
                                        c3.state = 5
```

```
                        c3.pos = 3
                        s1.ticket = c3


            else:
                if s1.idle == True:

                    if len(qs[3].tickets) > 0:
                        c3 = qs[3].dequeue_0()
                        b3 = self.servDist[5].rvs()
                        dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                        fes.add(dep)
                        c3.depEvent = dep    #ticket knows its departure and service time
                        s1.ticket = c3
                        c3.pos = 3
                        c3.state = 5
                        s1.idle = False
                        s1.setWorking(t)


                    elif len(qs[2].tickets) > 0:
                        if s1.impltCounter < 2:
                            c3 = qs[2].dequeue_0()  #ticket is not being served
                            b3 = self.servDist[4].rvs()
                            dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                            fes.add(dep)
                            c3.depEvent = dep    #ticket knows its departure and service time
                            s1.ticket = c3
                            c3.tester = s1
                            s1.idle = False
                            c3.state = 4    #goes into implemented state
                            c3.pos = 2
                            s1.impltCounter += 1
                            s1.setWorking(t)

                    elif len(qs[1].tickets) > 0:
                        if s1.analyseCounter < 4:
                            c3 = qs[1].dequeue_0()
                            b3 = self.servDist[2].rvs()
                            dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                            fes.add(dep)
                            c3.depEvent = dep    #ticket knows its departure and service time
                            s1.ticket = c3
                            c3.state = 2    #goes into analysed state
                            c3.pos = 1
                            s1.idle = False
                            s1.analyseCounter += 1
                            s1.setWorking(t)


                    elif len(qs[2].tickets) > 0  and len(qs[2].tickets) < 20:
                        if s1.impltCounter < 2:
                            c3 = qs[2].dequeue_0()
                            b3 = self.servDist[4].rvs()
                            dep = Event(Event.DEPARTURE, s1, c3, t+b3)
                            fes.add(dep)
```

```
                                        c3.depEvent = dep    #ticket knows its departure and service time
                                        s1.idle = False
                                        s1.ticket = c3
                                        c3.tester = s1
                                        c3.state = 4    #goes into implemented state
                                        c3.pos = 2
                                        s1.impltCounter += 1
                                        s1.setWorking(t)


        elif e.type == Event.EOD:

            # Resetting all counters for the day
            s1.openCounter = 0
            s1.impltCounter = 0
            s1.analyseCounter = 0

            for i in engg:
                serv = i
                serv.countAccept = 0
                serv.countImplt = 0
                serv.countSch = 0

            # Storing results for the day

            OpenTickets.append(len(qs[0].tickets))
            AcceptedTickets.append(countAccepted(engg[0]) + countAccepted(engg[1]) + countAccepted(engg[2]) + countAccepted(engg[3]) + countAccepted(engg[4]) + countAccepted(engg[5]) + countAccepted(engg[6]))
            AnalysedTickets.append(len(qs[1].tickets))
            ScheduledTickets.append(countSch(engg[0]) + countSch(engg[1]) + countSch(engg[2]) + countSch(engg[3]) + countSch(engg[4]) + countSch(engg[5]) + countSch(engg[6]))
            ImplementedTickets.append(len(qs[2].tickets))
            VerifiedTickets.append(len(qs[3].tickets))

            engg_1.append(len(engg[0].queue.tickets))
            engg_2.append(len(engg[1].queue.tickets))
            engg_3.append(len(engg[2].queue.tickets))
            engg_4.append(len(engg[3].queue.tickets))
            engg_5.append(len(engg[4].queue.tickets))
            engg_6.append(len(engg[5].queue.tickets))
            engg_7.append(len(engg[6].queue.tickets))
            s1_tickets.append(len(qs[0].tickets) + len(qs[1].tickets) + len(qs[3].tickets))
            tester_tickets.append(len(qs[2].tickets))

            # Opening of tickets at the start of the day

            if s1.ticket == None:
                if len(qs[0].tickets) > 0:
                    c2 = qs[0].dequeue_0()
                    b0 = self.servDist[0].rvs()
                    dep = Event(Event.DEPARTURE, s1, c2, t+b0)
                    c2.depEvent = dep    #ticket knows its departure and service time
                    fes.add(dep)
                    s1.idle = False
                    s1.ticket = c2
                    c2.server = s1
                    c2.state = 0
                    c2.pos = 0
                    s1.openCounter += 1
                    s1.setWorking(t)

```

```
                else:
                    c2 = s1.ticket
                    if c2 != None and c2.state != 0 and c2.depEvent != None and c2.priority != 0:
                        if len(qs[0].tickets) > 0:
                            c2.depEvent.iscancelled = True
                            c2.depEvent = None
                            s1.nrTickets.remove(s1.ticket.number)
                            s1.nrTickets.append(c.number)
                            c = qs[0].dequeue_0()
                            b2 = self.servDist[0].rvs()
                            dep = Event(Event.DEPARTURE, s1, c, t+b2)
                            c.depEvent = dep    #ticket knows its departure and service time
                            fes.add(dep)
                            s1.idle = False
                            s1.ticket = c
                            c.state = 0
                            c.pos = 0
                            s1.openCounter += 1

                fes.add(Event(Event.EOD, s1, c, t+8*3600))
                nrDays_vs_Tickets.loc[day, "Tickets"] = len(set(tickets_in_system_day))
                day = day + 1
                tickets_in_system_day = []

        return res


#arrival distribution seconds in per ticket
servDist = []              # manually create them # service times

arrDist = Distribution(stats.expon(scale = ((365*5*3600*8)/8492)))   #arrival time

servDist.append(Distribution(stats.gamma(scale = 1/((2*60)/5580000), a = ((2*60))**2/(5580000))))     # open
servDist.append(Distribution(stats.gamma(scale = 1/((30*3600/90000000)), a = ((30*3600)**2/90000000))))    # accepted
servDist.append(Distribution(stats.gamma(scale = 1/((20*60)/(900000)), a = (20*60)**2/(900000))))     # analysed
servDist.append(Distribution(stats.gamma(scale = 1/((20*3600)/337000000), a = (20*3600)**2/(337000000))))     # scheduled
servDist.append(Distribution(stats.gamma(scale = 1/((10*3600)/2600000000), a = (10*3600)**2/(2600000000))))   # implemented
servDist.append(Distribution(stats.gamma(scale = 1/((10*60)/8500000), a = (10*60)**2/(8500000))))            # verified

years = 5               # number of years for which the simulation needs to run
run = 10                # index of the current run

r = 365*years          # in days

sim = Simulation(arrDist , servDist , 8)
res = sim.simulate(r*8*3600)       # input in hours
bins = 100

################################ FOR RESULTS ################################

# Printing results

for i in range(6):

    if i == 0:
        print("##### RESULTS FOR OPEN QUEUE #####")
        print(res.queueResults[i])
        res.queueResults[i].histQueueLength(i,10)
```

```python
        res.queueResults[i].histWaitingTimes(i,100)

    elif i == 1:
        print("##### RESULTS FOR ACCEPTED QUEUE #####")
        print(res.queueResults[i])
        res.queueResults[i].histQueueLength(i,1000)
        res.queueResults[i].histWaitingTimes(i,50)

    elif i == 2:
        print("##### RESULTS FOR ANALYSED QUEUE #####")
        print(res.queueResults[i])
        res.queueResults[i].histQueueLength(i,10)
        res.queueResults[i].histWaitingTimes(i,50)

    elif i == 3:
        print("##### RESULTS FOR SCHEDULED QUEUE #####")
        print(res.queueResults[i])
        res.queueResults[i].histQueueLength(i,500)
        res.queueResults[i].histWaitingTimes(i,100)

    elif i == 4:
        print("##### RESULTS FOR IMPLEMENTED QUEUE #####")
        print(res.queueResults[i])
        res.queueResults[i].histQueueLength(i,10)
        res.queueResults[i].histWaitingTimes(i,100)


    elif i == 5:
        print("##### RESULTS FOR VERIFIED QUEUE #####")
        print(res.queueResults[i])
        res.queueResults[i].histQueueLength(i,10)
        res.queueResults[i].histWaitingTimes(i,0.5)

print("Mean Sojourn time: (in hours) ", mean(res.sojournTimes)/3600)
res.histSojournTimes(50)

print("\nS1 Working time: (in hours) ", s1.workTime/3600, " (", s1.workTime*100/(3600*8*years*365), "%)", " and Idle time: (in hours) ", s1.idleTime/3600, " (", s1.idleTime*100/(3600*8*years*365), "%)", " and number of tickets worked on - ", len(set(s1.nrTickets)))
sojournTimes_Servers.loc[len(sojournTimes_Servers.index), s1] = s1.workTime/3600
sojournTimes_Servers_idle.loc[len(sojournTimes_Servers_idle.index), s1] = s1.idleTime/3600

for i in engg:
    print("\n", i, " Working time: (in hours) ", i.workTime/3600, " (", i.workTime*100/(3600*8*years*365), "%)", " and Idle time: (in hours) ", i.idleTime/3600, " (", i.idleTime*100/(3600*8*years*365), "%)", " and number of tickets worked on - ", len(set(i.nrTickets)))
    print("Number of tickets worked on distribution - \nAccepted state tickets: ", len(set(i.acceptTickets)), " Schedule state tickets: ", len(set(i.schTickets)), " Implemented state tickets: ", len(set(i.impltTickets)))
    sojournTimes_Servers.loc[len(sojournTimes_Servers.index), i] = i.workTime/3600
    sojournTimes_Servers_idle.loc[len(sojournTimes_Servers_idle.index), i] = i.idleTime/3600


print("\nTester Working time: (in hours) ", tester.workTime/3600, " (", tester.workTime*100/(3600*8*years*365), "%)", " and Idle time: (in hours) ", tester.idleTime/3600, " (", tester.idleTime*100/(3600*8*years*365), "%)", " and number of tickets worked on - ", len(set(tester.nrTickets)))
sojournTimes_Servers.loc[len(sojournTimes_Servers.index), tester] = tester.workTime/3600
sojournTimes_Servers_idle.loc[len(sojournTimes_Servers_idle.index), tester] = tester.idleTime/3600

def max_work(servers):
```

```
    workTimes = []
    for i in servers:
        workTimes.append(i.workTime)

    for i in servers:
        if max(workTimes) == i.workTime:
            return i

server = max_work([s1,tester]+engg)
print("\nServer who works the most - ", server)

def max_idle(servers):
    idleTimes = []
    for i in servers:
        idleTimes.append(i.idleTime)

    for i in servers:
        if max(idleTimes) == i.idleTime:
            return i

server = max_idle([s1, tester] + engg)
print("\nServer who is the most idle - ", server)

Comp_Sojourn = sojournTimes_Components.mean(axis=0)
plt.figure(figsize=(12, 6))
Comp_Sojourn.plot(kind='bar')
plt.xlabel('Components', fontsize = 12)
plt.ylabel('Time Spent (in hours)', fontsize = 12)
plt.title('Time Spent (Component-wise)', fontsize = 14)
# show the plot
plt.show()

serverss = sojournTimes_Servers.mean(axis=0)
plt.figure(figsize=(12, 6))
serverss.plot(kind='bar')
plt.xlabel('Servers', fontsize = 12)
plt.ylabel('Work time (in hours)', fontsize = 12)
plt.title('Work time for each Server', fontsize = 14)
# show the plot
plt.show()

Comp_Total = totalTimes_Components.mean(axis=0)
plt.figure(figsize=(12, 6))
Comp_Total.plot(kind='bar')
plt.xlabel('Components', fontsize = 12)
plt.ylabel('Time Spent (in hours)', fontsize = 12)
plt.title('Total Time in the System (Component-wise)', fontsize = 14)


nrDays_vs_Tickets.plot(y = "Tickets", kind='line', figsize = (30,10), fontsize =
    16)
plt.title("Number of Tickets per day", fontsize = 16)
plt.xlabel("Day", fontsize = 16)
plt.ylabel("Tickets", fontsize = 16)
plt.show()

plt.plot(OpenTickets)
plt.title("Number of Tickets per day - Open queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()
dy1 = np.gradient(OpenTickets)
print("Rate of increase of tickets in Open state (per day):", mean(dy1))

plt.plot(AcceptedTickets)
plt.title("Number of Tickets per day - Accepted queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
```

```
plt.ylabel("Tickets", fontsize = 12)
plt.show()
dy2 = np.gradient(AcceptedTickets)
print("Rate of increase in Accepted state (per day):", mean(dy2))

plt.plot(AnalysedTickets)
plt.title("Number of Tickets per day - Analysed queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()
dy3 = np.gradient(AnalysedTickets)
print("Rate of increase in Analysed state (per day):", mean(dy3))

plt.plot(ScheduledTickets)
plt.title("Number of Tickets per day - Scheduled queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()
dy4 = np.gradient(ScheduledTickets)
print("Rate of increase in Scheduled state (per day):", mean(dy4))

plt.plot(ImplementedTickets)
plt.title("Number of Tickets per day - Implemented queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()
dy5 = np.gradient(ImplementedTickets)
print("Rate of increase in Implemented state (per day):", mean(dy5))

plt.plot(VerifiedTickets)
plt.title("Number of Tickets per day - Verified queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()
dy6 = np.gradient(VerifiedTickets)
print("Rate of increase in Verified state (per day):", mean(dy6))

fig, ax = plt.subplots()
ax.plot(engg_1, label='Engg 1')
ax.plot(engg_2, label='Engg 2')
ax.plot(engg_3, label='Engg 3')
ax.plot(engg_4, label='Engg 4')
ax.plot(engg_5, label='Engg 5')
ax.plot(engg_6, label='Engg 6')
ax.plot(engg_7, label='Engg 7')
ax.set_xlabel('Days')
ax.set_title('Engineer Queues')
ax.set_ylabel('Tickets')
ax.legend()
plt.show()

dy_engg1 = np.gradient(engg_1)
print("\n\nRate of increase Engg 1:", mean(dy_engg1))
dy_engg2 = np.gradient(engg_2)
print("Rate of increase Engg 2:", mean(dy_engg2))
dy_engg3 = np.gradient(engg_3)
print("Rate of increase Engg 3:", mean(dy_engg3))
dy_engg4 = np.gradient(engg_4)
print("Rate of increase Engg 4:", mean(dy_engg4))
dy_engg5 = np.gradient(engg_5)
print("Rate of increase Engg 5:", mean(dy_engg5))
dy_engg6 = np.gradient(engg_6)
print("Rate of increase Engg 6:", mean(dy_engg6))
dy_engg7 = np.gradient(engg_7)
print("Rate of increase Engg 7:", mean(dy_engg7))

```

```
print("\nMean number of tickets that go through the system in a day: ", mean(
    nrDays_vs_Tickets["Tickets"]))


data = {
        'engg 1': mean(dy_engg1),
        'engg 2': mean(dy_engg2),
        'engg 3': mean(dy_engg3),
        'engg 4': mean(dy_engg4),
        'engg 5': mean(dy_engg5),
        'engg 6': mean(dy_engg6),
        'engg 7': mean(dy_engg7),
        'open': mean(dy1),
        'accepted': mean(dy2),
        'analysed': mean(dy3),
        'scheduled': mean(dy4),
        'implemented': mean(dy5),
        'verified': mean(dy6),
        'mean nr. of tickets through the system': mean(nrDays_vs_Tickets["Tickets"
    ])
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/rate of increase.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    rate of increase.xlsx', index=False)

print("\nMean time spent by each priority in the system - ")

if len(priority_0) > 0:
    print(" Blocking '0' : ", mean(priority_0)/3600, " for ", len(set(
    count_priority_0)), " tickets")

if len(priority_1) > 0:
    print(" Urgent '1' : ", mean(priority_1)/3600, " for ", len(set(
    count_priority_1)), " tickets")

if len(priority_2) > 0:
    print(" High '2' : ", mean(priority_2)/3600, " for ", len(set(count_priority_2)
    ), " tickets")

if len(priority_3) > 0:
    print(" Normal '3' : ", mean(priority_3)/3600, " for ", len(set(
    count_priority_3)), " tickets")

if len(priority_4) > 0:
    print(" Low '4' : ", mean(priority_4)/3600, " for ", len(set(count_priority_4))
    , " tickets")

data = {
        'blocking - time': mean(priority_0)/3600,
        'blocking - nr. Of tickets': len(set(count_priority_0)),
        'urgent - time': mean(priority_1)/3600,
        'urgent - nr. Of tickets': len(set(count_priority_1)),
        'high - time': mean(priority_2)/3600,
        'high - nr. Of tickets': len(set(count_priority_2)),
        'normal - time': mean(priority_3)/3600,
        'normal - nr. Of tickets': len(set(count_priority_3)),
        'low - time': mean(priority_4)/3600,
        'low - nr. Of tickets': len(set(count_priority_4)),
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/priority - time and tickets.xlsx')
df = df.append(data, ignore_index=True)
```

```
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    priority - time and tickets.xlsx', index=False)

a = 1
print("\nService times:")
for i in servDist:
    print(" Queue ", a, ": ", np.mean(i.rvs(365*years)))
    a = a+1

# Storing data

data = {
        0: res.queueResults[0].getMeanWaitingTime(),
        1: res.queueResults[1].getMeanWaitingTime(),
        2: res.queueResults[2].getMeanWaitingTime(),
        3: res.queueResults[3].getMeanWaitingTime(),
        4: res.queueResults[4].getMeanWaitingTime(),
        5: res.queueResults[5].getMeanWaitingTime()
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/waiting times.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    waiting times.xlsx', index=False)

data = {
        0: res.queueResults[0].getMeanQueueLength(),
        "0 - std dev": res.queueResults[0].getVarianceQueueLength() ** (1/2),
        1: res.queueResults[1].getMeanQueueLength(),
        "1 - std dev": res.queueResults[1].getVarianceQueueLength() ** (1/2),
        2: res.queueResults[2].getMeanQueueLength(),
        "2 - std dev": res.queueResults[2].getVarianceQueueLength() ** (1/2),
        3: res.queueResults[3].getMeanQueueLength(),
        "3 - std dev": res.queueResults[3].getVarianceQueueLength() ** (1/2),
        4: res.queueResults[4].getMeanQueueLength(),
        "4 - std dev": res.queueResults[4].getVarianceQueueLength() ** (1/2),
        5: res.queueResults[5].getMeanQueueLength(),
        "5 - std dev": res.queueResults[5].getVarianceQueueLength() ** (1/2)
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/queue lengths.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    queue lengths.xlsx', index=False)

data = {
        'mean sojourn time': mean(res.sojournTimes)/3600,
        's1': s1.workTime/3600,
        's1 - tickets solved': len(set(s1.nrTickets)),
        1: engg[0].workTime/3600,
        '1 - tickets solved': len(set(engg[0].nrTickets)),
        2: engg[1].workTime/3600,
        '2 - tickets solved': len(set(engg[1].nrTickets)),
        3: engg[2].workTime/3600,
        '3 - tickets solved': len(set(engg[2].nrTickets)),
        4: engg[3].workTime/3600,
        '4 - tickets solved': len(set(engg[3].nrTickets)),
        5: engg[4].workTime/3600,
        '5 - tickets solved': len(set(engg[4].nrTickets)),
        6: engg[5].workTime/3600,
        '6 - tickets solved': len(set(engg[5].nrTickets)),
        7: engg[6].workTime/3600,
        '7 - tickets solved': len(set(engg[6].nrTickets)),
        'tester': tester.workTime/3600,
        'tester - tickets solved': len(set(tester.nrTickets))
```

```
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/sojourn times servers and mean.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    sojourn times servers and mean.xlsx', index=False)

data = {
        's1': s1.idleTime/3600,
        1: engg[0].idleTime/3600,
        2: engg[1].idleTime/3600,
        3: engg[2].idleTime/3600,
        4: engg[3].idleTime/3600,
        5: engg[4].idleTime/3600,
        6: engg[5].idleTime/3600,
        7: engg[6].idleTime/3600,
        'tester': tester.idleTime/3600,
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/idle times.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    idle times.xlsx', index=False)

data = {
        1: OpenTickets
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/eod queue lengths open.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    eod queue lengths open.xlsx', index=False)

data = {
        1: AcceptedTickets
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/eod queue length accepted.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    eod queue length accepted.xlsx', index=False)

data = {
        1: AnalysedTickets
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/eod queue lengths analysed.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    eod queue lengths analysed.xlsx', index=False)

data = {
        1: ScheduledTickets
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/eod queue lengths scheduled.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    eod queue lengths scheduled.xlsx', index=False)

data = {
```

```
        1: ImplementedTickets
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/eod queue lengths implemented.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    eod queue lengths implemented.xlsx', index=False)

data = {
        1: VerifiedTickets
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/eod queue lengths verified.xlsx')
df = df.append(data, ignore_index=True)
df.to_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/
    eod queue lengths verified.xlsx', index=False)

data = {
        1: engg_1,
        2: engg_2,
        3: engg_3,
        4: engg_4,
        5: engg_5,
        6: engg_6,
        7: engg_7
        }

df = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/
    prob 1/eod queue length engineers.xlsx')
df = df.append(data, ignore_index=True)
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

```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 01 19:17:25 2023

@author: 20181301
"""


import csv
import pandas as pd
import numpy as np
from scipy.stats import t
import math
import matplotlib . pyplot as plt
import seaborn as sns
import ast

def calc_CI(m):
    n = len(m)
    X = np.mean(m)
    s = np.std(m, ddof=1)
    confidence_level = 0.95
    dof = n - 1
    t_value = t.ppf((1 + confidence_level) / 2, dof)
    error = t_value * (s / np.sqrt(n))
    l_bound = X - error
    u_bound = X + error
    return (l_bound, u_bound)
```

```python


print("############ WAITING TIMES ###############")
waitingTimes_states = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/waiting times.xlsx')
for i in range(0,6):
    if i == 0:
        print("###### Open State #######")
    elif i == 1:
        print("###### Accepted State #######")
    elif i == 2:
        print("###### Analysed State #######")
    elif i == 3:
        print("###### Scheduled State #######")
    elif i == 4:
        print("###### Implemented State #######")
    elif i == 5:
        print("###### Verified State #######")
    print("\n",waitingTimes_states[i].describe())
    print("\nConfidence interval: ", calc_CI(waitingTimes_states[i]), "\n")

print("############ QUEUE LENGTHS ###############")
queueLengths_states = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/queue lengths.xlsx')
for i in range(0,6):
    if i == 0:
        print("###### Open State #######")
        print("\n",queueLengths_states[i].describe())
        print("\nFor standard dev.:",queueLengths_states["0 - std dev"].describe())
        print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
    elif i == 1:
        print("###### Accepted State #######")
        print("\n",queueLengths_states[i].describe())
        print("\nFor standard dev.:",queueLengths_states["1 - std dev"].describe())
        print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
    elif i == 2:
        print("###### Analysed State #######")
        print("\n",queueLengths_states[i].describe())
        print("\nFor standard dev.:",queueLengths_states["2 - std dev"].describe())
        print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
    elif i == 3:
        print("###### Scheduled State #######")
        print("\n",queueLengths_states[i].describe())
        print("\nFor standard dev.:",queueLengths_states["3 - std dev"].describe())
        print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
    elif i == 4:
        print("###### Implemented State #######")
        print("\n",queueLengths_states[i].describe())
        print("\nFor standard dev.:",queueLengths_states["4 - std dev"].describe())
        print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")
    elif i == 5:
        print("###### Verified State #######")
        print("\n",queueLengths_states[i].describe())
        print("\nFor standard dev.:",queueLengths_states["5 - std dev"].describe())
        print("\nConfidence interval: ", calc_CI(queueLengths_states[i]), "\n")


print("############ SOJOURN TIMES, WORK TIMES & IDLE TIMES ###############")
work_sojourn_times = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/sojourn times servers and mean.xlsx')
idle_sojourn_times = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/idle times.xlsx')

total_time = 5*365*8

print("\nMean Sojourn Time: ", np.mean(work_sojourn_times["mean sojourn time"]), " and CI: ", calc_CI(work_sojourn_times["mean sojourn time"]))
```

```
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"]))
print("S1 idle time: ", np.mean(idle_sojourn_times["s1"]), "(", np.mean(idle_sojourn_times["s1"])*100/total_time, "%)")

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"]))
print("Engg 1 idle time: ", np.mean(idle_sojourn_times[1]), "(", np.mean(idle_sojourn_times[1])*100/total_time, "%)")

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"]))
print("Engg 2 idle time: ", np.mean(idle_sojourn_times[2]), "(", np.mean(idle_sojourn_times[2])*100/total_time, "%)")

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"]))
print("Engg 3 idle time: ", np.mean(idle_sojourn_times[3]), "(", np.mean(idle_sojourn_times[3])*100/total_time, "%)")

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"]))
print("Engg 4 idle time: ", np.mean(idle_sojourn_times[4]), "(", np.mean(idle_sojourn_times[4])*100/total_time, "%)")

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"]))
print("Engg 5 idle time: ", np.mean(idle_sojourn_times[5]), "(", np.mean(idle_sojourn_times[5])*100/total_time, "%)")

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"]))
print("Engg 6 idle time: ", np.mean(idle_sojourn_times[6]), "(", np.mean(idle_sojourn_times[6])*100/total_time, "%)")

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"]))
print("Engg 7 idletime: ", np.mean(idle_sojourn_times[7]), "(", np.mean(idle_sojourn_times[7])*100/total_time, "%)")

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"]))
print("Tester idle time: ", np.mean(idle_sojourn_times["tester"]), "(", np.mean(idle_sojourn_times["tester"])*100/total_time, "%)")

print("\n############ RATE OF INCREASE OF QUEUES ###############")
rates_of_increase = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/Simulation/prob 1/rate of increase.xlsx')
print("\nOpen state: ", np.mean(rates_of_increase["open"]))
print("Accepted state: ", np.mean(rates_of_increase["accepted"]))
print("Analysed state: ", np.mean(rates_of_increase["analysed"]))
print("Scheduled state: ", np.mean(rates_of_increase["scheduled"]))
print("Implemented state: ", np.mean(rates_of_increase["implemented"]))
print("Verified state: ", np.mean(rates_of_increase["verified"]))
print("Engg 1: ", np.mean(rates_of_increase["engg 1"]))
print("Engg 2: ", np.mean(rates_of_increase["engg 2"]))
print("Engg 3: ", np.mean(rates_of_increase["engg 3"]))
print("Engg 4: ", np.mean(rates_of_increase["engg 4"]))
```

```
print("Engg 5: ", np.mean(rates_of_increase["engg 5"]))
print("Engg 6: ", np.mean(rates_of_increase["engg 6"]))
print("Engg 7: ", np.mean(rates_of_increase["engg 7"]))
print("Average number of tickets going through the system (per day): ", np.mean(
    rates_of_increase["mean nr. of tickets through the system"]))

print("\n############ PRIORITY TICKETS ##############")
priority = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/priority - time and tickets.xlsx')
print("\nBlocking: average time spent in system : ", np.mean(priority["blocking -
    time"]), " for ", np.mean(priority["blocking - nr. Of tickets"]))
print("Urgent: average time spent in system : ", np.mean(priority["urgent - time"])
    , " for ", np.mean(priority["urgent - nr. Of tickets"]))
print("High: average time spent in system : ", np.mean(priority["high - time"]), "
    for ", np.mean(priority["high - nr. Of tickets"]))
print("Normal: average time spent in system : ", np.mean(priority["normal - time"])
    , " for ", np.mean(priority["normal - nr. Of tickets"]))
print("Low: average time spent in system : ", np.mean(priority["low - time"]), "
    for ", np.mean(priority["low - nr. Of tickets"]))


### Plotting EOD queues

def return_avg_eod_queue(df):
    result = []
    for i in range(df.shape[0]):
        for j in range(df.shape[1]):
            cell_list = df.iloc[i, j]
            cell = ast.literal_eval(cell_list)
            if result == []:
                result = [0]*len(cell)
            for i in range(len(cell)):
                result[i] = result[i] + cell[i]
    n = len(df)
    result = [r/n for r in result ]
    return result

OpenTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue lengths open.xlsx')
AcceptedTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue length accepted.xlsx')
AnalysedTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue lengths analysed.xlsx')
ScheduledTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP
    /Simulation/prob 1/eod queue lengths scheduled.xlsx')
ImplementedTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/
    BFP/Simulation/prob 1/eod queue lengths implemented.xlsx')
VerifiedTickets = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue lengths verified.xlsx')

engineers = pd.read_excel('C:/Users/20181301/Desktop/APPLIED MATH/YEAR-3/BFP/
    Simulation/prob 1/eod queue length engineers.xlsx')
engg_1 = pd.DataFrame(engineers[1])
engg_2 = pd.DataFrame(engineers[2])
engg_3 = pd.DataFrame(engineers[3])
engg_4 = pd.DataFrame(engineers[4])
engg_5 = pd.DataFrame(engineers[5])
engg_6 = pd.DataFrame(engineers[6])
engg_7 = pd.DataFrame(engineers[7])

plt.plot(return_avg_eod_queue(OpenTickets))
plt.title("Number of Tickets per day - Open queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()

plt.plot(return_avg_eod_queue(AcceptedTickets))
```

```
plt.title("Number of Tickets per day - Accepted queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()

plt.plot(return_avg_eod_queue(AnalysedTickets))
plt.title("Number of Tickets per day - Analysed queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()

plt.plot(return_avg_eod_queue(ScheduledTickets))
plt.title("Number of Tickets per day - Scheduled queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()

plt.plot(return_avg_eod_queue(ImplementedTickets))
plt.title("Number of Tickets per day - Implemented queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()

plt.plot(return_avg_eod_queue(VerifiedTickets))
plt.title("Number of Tickets per day - Verified queue", fontsize = 14)
plt.xlabel("Day", fontsize = 12)
plt.ylabel("Tickets", fontsize = 12)
plt.show()

fig, ax = plt.subplots()
ax.plot(return_avg_eod_queue(engg_1), label='Engg 1')
ax.plot(return_avg_eod_queue(engg_2), label='Engg 2')
ax.plot(return_avg_eod_queue(engg_3), label='Engg 3')
ax.plot(return_avg_eod_queue(engg_4), label='Engg 4')
ax.plot(return_avg_eod_queue(engg_5), label='Engg 5')
ax.plot(return_avg_eod_queue(engg_6), label='Engg 6')
ax.plot(return_avg_eod_queue(engg_7), label='Engg 7')
ax.set_xlabel('Days')
ax.set_title('Engineer Queues')
ax.set_ylabel('Tickets')
ax.legend()
plt.show()
```