**Business Process Engineering**

**Project Report**

**Course Instructor**

Dr. Behjat Zuhaira

**Group Members**

Muhammad Awaimer Zaeem…………………………………………(22I-2616)

Abdullah Daoud……………………………………………………….(22I-2626)

Azghan Ahmed……………………………………………….……….(22I-2667)

Usman Ali………………………………………………………….….(22I-2725)

Faizan Rasheed………………………………………………………..(22I-2734)

Amna Asif…………………………………………………………….(22I-8777)

**Section**

SE-E

**Date**

**Sunday, May 5, 2024**

**Spring 2024**



**Department of Software Engineering**

FAST – National University of Computer & Emerging Sciences

Islamabad Campus

Table of Contents

[1. Executive Summary 1](#_Toc165826766)

[2. Introduction 1](#_Toc165826767)

[3. Assignment – 4: XPDL Generation and Parsing 2](#_Toc165826768)

[(a). Description of BPMN Models 2](#_Toc165826769)

[(i). Core Process (Internet Service Provision) 2](#_Toc165826770)

[(ii). Support Process (Network Architecture) 3](#_Toc165826771)

[(b). Generation Process 3](#_Toc165826772)

[(c). Development and Functionality of the Parsing Tool 4](#_Toc165826773)

[(i). File Loading 4](#_Toc165826774)

[(ii). Extraction of Elements 4](#_Toc165826775)

[(iii). Analysis of Data 4](#_Toc165826776)

[(d). Results 5](#_Toc165826777)

[(i). Event Counts 5](#_Toc165826778)

[(ii). Activity Counts 5](#_Toc165826779)

[(iii). Gateway and Artifact Counts 5](#_Toc165826780)

[(iv). Connecting Objects 6](#_Toc165826781)

[(v). Swim Lanes 6](#_Toc165826782)

[(vi). Anomalies Detected 6](#_Toc165826783)

[4. Project: Process Modification and Variant Creation 6](#_Toc165826784)

[(a). Dynamic Modification of XPDL Files 6](#_Toc165826785)

[(i). addActivityTime Algorithm 6](#_Toc165826786)

[(ii). calculateCT Algorithm 7](#_Toc165826787)

[(iii). Results 8](#_Toc165826788)

[(b). Creation of Process Variants 9](#_Toc165826789)

[(c). Conclusion 13](#_Toc165826790)

[(i). Challenges 13](#_Toc165826791)

[(ii). Key Findings 13](#_Toc165826792)

[(iii). Future Project Recommendations 13](#_Toc165826793)

[(d). Appendices 14](#_Toc165826794)

[5. References 27](#_Toc165826795)

# 1. Executive Summary

The purpose of this report is to present the key findings and achievements from assignment – 4 and the project on the Business Process Model Notation (BPMN) and XML Process Definition Language (XPDL). The project covers two important sections:

* Assignment – 4
* Project

Assignment – 4 included converting our BPMN Models into XPDL Formats and parsing them through a tool for analysis whereas the Main Project involved assigning random times to specific activities in those BPMN Models, calculating there cycle times, and creating process variants to understand variability that can occur in a project

The Project ended with us gaining numerous significant insights, such as the significance of precise and accurate BPMN modeling and efficient XPDL parsing for process analysis. We have realized that many projects place an emphasis on reliable tools that can help organizations understand how to make a business process adaptable and explore all possibilities by creating process variants. Additional materials, such as diagrams and code, are included in the appendices and references for guidance

# 2. Introduction

The purpose of this project was to investigate the meaning of XML Process Definition Language (XPDL) and Business Process Model and Notation (BPMN) in relation to process modeling. BPMN is a popular graphical notation for business process modeling, and XPDL is an XML-based format for machine-readable representation of BPMN diagrams. This combination makes it a vital tool in business process management because it enables a smooth transition from process design to execution and analysis

In Assignment 4, we generated and parsed XPDL files to extract and analyze BPMN elements used in the BPMN Diagrams of our Core and Support processes. We developed a parsing tool by writing code in Java that provided insights into the composition of the process diagrams. The results revealed a comprehensive breakdown of the BPMN elements, including events, activities, gateways, and artifacts, along with detailed information on the parsing process and all the elements used in the BPMN Models

For the project, we expanded on the assignment – 4 and focused on dynamic modifications of XPDL files by implementing an algorithm called "addActivityTime" that assigns random times to specific activities. This procedure made it possible to add variability to the model, showing how business procedures can change to adapt to specific circumstances. Based on this procedure, we calculated the cycle time of an entire process and developed a number of process variants that provide a more thorough understanding of how various different configurations of a process can affect total cycle times and efficiency of a process. These changes demonstrate how flexible BPMN and XPDL are in terms of accommodating changes in various business processes

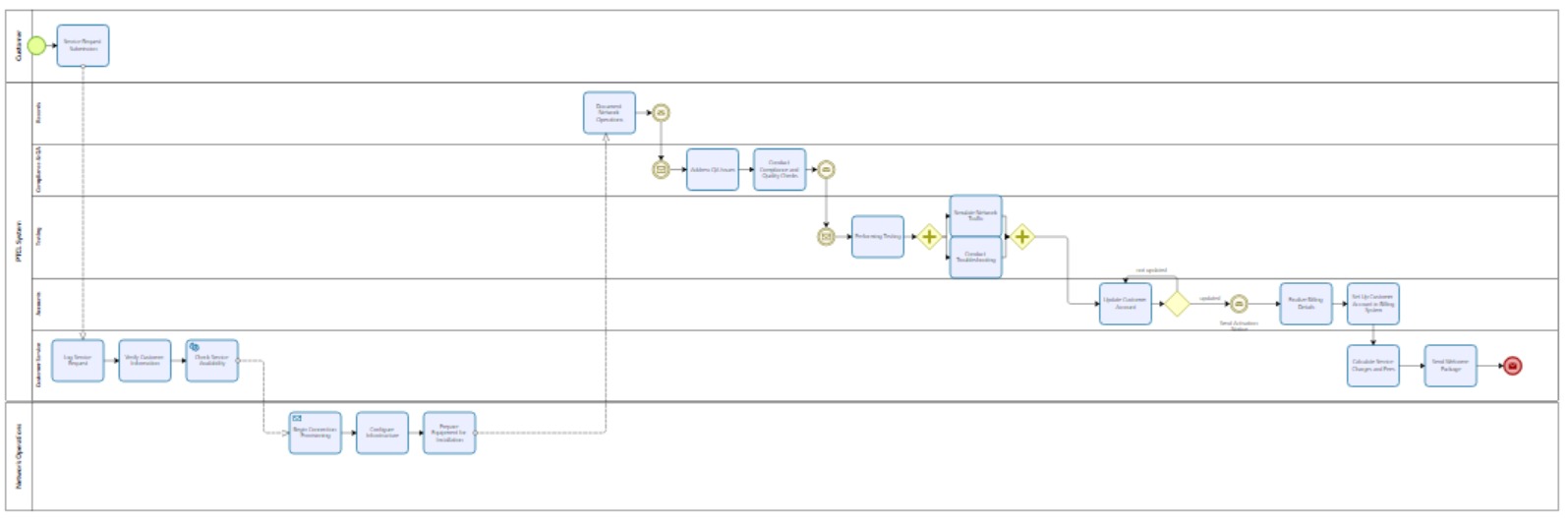
The report includes all details about the project's results, technical difficulties faced, and implemented methods and techniques. Additionally, there are appendices and references section that include the specific source code and tools used for this entire project

# 3. Assignment – 4: XPDL Generation and Parsing

## (a). Description of BPMN Models

### (i). Core Process (Internet Service Provision)

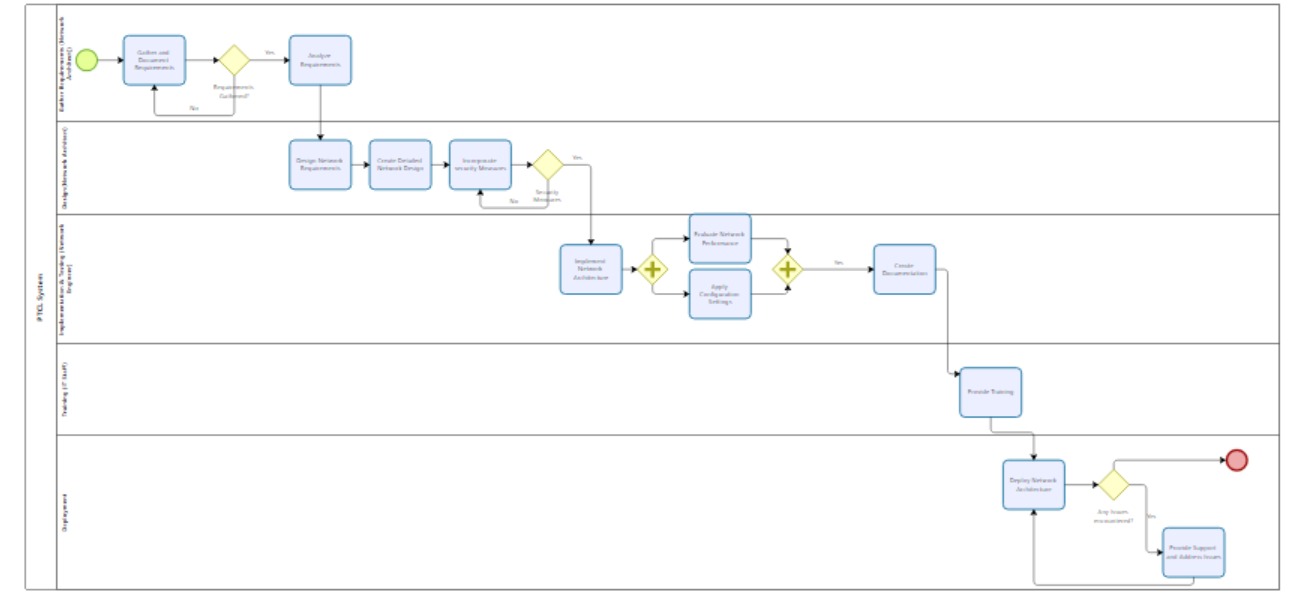
Our Core Process is delivering internet service, the BPMN Model developed for the core process encompasses all necessary activities required to carry out this process



The process starts with PTCL logging the service request made by a customer and verifies the service availability, after that the equipment is prepared for installation by carrying out compliance, QA checks and testing, then finally billing details are sent to the customer which is payed for and the customer receives a welcome message of the service package

### (ii). Support Process (Network Architecture)

Our Support Process is the creation of the network architecture which serves a basis for our core process, the BPMN Model developed for the support process displays all the steps carried out in sequence to develop the network architecture



The process starts with gathering and analyzing the requirements of the network architecture, after that the design is constructed by implementing security measures as a precaution, and lastly, the whole procedure is documented and the network architecture is deployed with constant maintenance to make sure there is minimum downtime of network services

## (b). Generation Process

The tools used for converting our BPMN Models into XPDL files were bpmn.io and Bizagi

We created our BPMN Models for both the core and support process on bpmn.io, but the tool didn’t include an option for exporting or converting it into an XPDL file, so instead we used Bizagi

We imported our BPMN Models created on bpmn.io into Bizagi, made some modifications in our BPMN Models and then converted and exported our BPMN Models from Bizagi into XPDL formats

## (c). Development and Functionality of the Parsing Tool

The parsing tool is designed to read, analyze, and extract information from XML Process Definition Language (XPDL) files, which are used to represent Business Process Model and Notation (BPMN) diagrams in a machine-readable format. Developed using Java, the tool leverages the Document Object Model (DOM) to parse, traverse and extract valuable information from the XPDL structure

The methodology employed by the tool includes the following key steps:

### (i). File Loading

The tool first loads the specified XPDL file by giving its path and creates a DOM tree structure using the DocumentBuilderFactory and DocumentBuilder classes. This process involves parsing the XML content and normalizing it to ensure a consistent structure

### (ii). Extraction of Elements

Using the self-made getElementCount and getElementCountWithAttribute methods, the tool extracts specific elements from the DOM tree based on their tag names and attributes. These methods allow the tool to count various BPMN elements, like events, activities, gateways, artifacts, and connecting objects

### (iii). Analysis of Data

Once the elements are extracted and counted, the tool proceeds to analyze the data, providing counts for each type of BPMN element and additional attributes like triggers for events and types of tasks in a report format

During the development of the parsing tool, several challenges were faced, which included:

* Making sure that the tool accurately identified and counted the correct elements and attributes, since an XPDL File contains multiple different variations of attributes and tags. This required careful validation and error handling to address these issues when traversing the XML File
* The usage of built-in functions which are getDocumentElement and getElementsByTagName. Since the built-in functions were generating incorrect results, it meant we had to create our own versions of the built-in functions of the DOM structure that implemented these built-in functions in a new way

Solutions implemented to overcome these challenges included:

* Exception handling, allowing the tool to continue operating even when encountering unexpected data or errors.
* Implementing our own versions of the DOM Structures built-in functions which are getElementCount and getElementCountWithAttribute. These functions were created by analyzing and understanding how the built-in functions of the DOM Structure worked in the first place

## (d). Results

The parsing tool's outcomes are presented in a detailed report that includes counts of various different BPMN elements. The results provide a comprehensive breakdown of the XPDL file's contents, allowing users to understand the structure of the BPMN diagram.

The tool's results included the following key findings:

### (i). Event Counts

The total number of events, including specific counts for start, intermediate, end events and the total events. This information provides insights into the BPMN diagram's overall workflow

### (ii). Activity Counts

Detailed breakdowns of the various activities within the BPMN diagram, including total activities, tasks and sub-processes, and user tasks. It also includes additional attributes, such as task implementation types like user, service, script, manual and receive tasks which describes the specific tasks and activities being carried out in the whole process

### (iii). Gateway and Artifact Counts

Counts for different types of gateways and artifacts, like total gateways, exclusive, parallel, inclusive gateways and data objects, groups and annotations show the complexity and branching logic within the BPMN diagram

### (iv). Connecting Objects

Analysis of sequence flows, message flows, and associations, demonstrate how different BPMN elements are connected with each other in the process

### (v). Swim Lanes

Clear Segragation of Swim lanes like pools and lanes showcase the different departments/teams or individuals involved in the whole process

### (vi). Anomalies Detected

During the parsing process, several anomalies were detected, such as missing or unexpected attributes, and incorrect element counts. These anomalies were investigated, with solutions implemented to ensure the tool's robustness and accuracy. The report provides a detailed explanation of these findings and the steps taken to resolve them

# 4. Project: Process Modification and Variant Creation

## (a). Dynamic Modification of XPDL Files

### (i). addActivityTime Algorithm

The purpose of the addActivityTime function is to assign a random time estimating between 5 to 15 minutes to each task used in our Core and Support BPMN Models by analyzing the XPDL Files. By doing a thorough analysis of the XPDL Files, we noticed a pattern that could allow us to implement this function in a quick and efficient way. The pattern was that all the tasks in the BPMN Models used two specific tags inside the activity tag in the XPDL file, which are <Implementation> and <Task> tags, using this information, we developed the function in a relatively quick way

**Steps:**

* Initialization: The function retrieves a NodeList containing all elements with the tag "Activity" from the provided Document object. It also creates an instance of the Random class to generate random numbers.
* Looping through Activities: The function uses a for loop to iterate over the list of "Activity" elements. For each "Activity", it checks if the node is an element node (Node.ELEMENT\_NODE)
* Inside the loop, the function initializes a boolean variable, hasTaskInImplementation, to false. This variable tracks whether the "Activity" contains an "Implementation" tag with a nested "Task" tag.
* The function then obtains the list of child nodes for the current "Activity" using NodeList children = activity.getChildNodes()
* It uses a nested for loop to iterate through the child nodes. If a child node is an element node and its tag name is "Implementation", the algorithm enters a second nested loop to check the child nodes of this "Implementation" tag.
* In the second nested loop, the function checks each child node for an element node with the tag name "Task". If found, it sets hasTaskInImplementation to true and breaks the loop
* After the nested loops, if hasTaskInImplementation is true, the function generates a random time between 5 and 15 minutes. It does this by adding a base value of 5 to a random integer from 0 to 10 (5 + random.nextInt(11)), ensuring an inclusive range of 5 to 15
* The function then assigns this random time as the value for the "Duration" attribute on the "Activity" tag (activity.setAttribute("Duration", randomTime + " minutes"))

### (ii). calculateCT Algorithm

The calculateCT function calculates the total cycle time of a process based on the "Duration" attribute in the "Activity" elements that was assigned by the addActivityTime function before it. The way it works is:

**Steps:**

* Initialization: The function creates two maps:
* activityDurations: Stores the duration of each activity, with the activity's "Id" as the key and its duration as the value
* predecessors: Maps each activity's "Id" to a list of the "Id"s of its incoming flows
* It loops through all "Activity" elements in the XPDL document
* For each "Activity", it retrieves the "Id" and "Duration" attributes. If the "Duration" is not empty, it extracts the numerical value and adds it to activityDurations
* It also gathers the list of incoming flows for each activity and stores them in predecessors
* It initializes a variable totalDuration to 0.
* The function then loops through each activity in activityDurations. It calculates the maximum path duration for each activity using the getMaxPathDuration function
* It updates totalDuration with the calculated maximum path duration for that activity, thererfore the total cycle time considers the longest path in the process
* The function returns totalDuration as the total cycle time for the business process.
* The getMaxPathDuration function is used to calculate the duration of the longest path leading up to a given activity
* If the activity has no predecessors (it is a starting activity), it returns its own duration from activityDurations.
* If the activity has predecessors, the function loops through each predecessor "Id" and recursively calls getMaxPathDuration to get the duration of the longest path from the predecessor to the current activity
* It keeps track of the longest predecessor path duration.
* The function returns the sum of the longest predecessor path duration and the current activity's duration

### (iii). Results

* The addActivityTime function introduced variability into the BPMN models by assigning random durations between 5 and 15 minutes to specific activities. This modification was based on a pattern observed in the XPDL files where tasks were identified by the presence of both "Implementation" and nested "Task" tags within the "Activity" tag. After implementing this function, the modified XPDL files contained a "Duration" attribute for each relevant "Activity," providing additional flexibility and variability in the process models.
* After that, using the modified XPDL files, the calculateCT function calculated the total cycle time of the business process by adding the durations of sequential and parallel tasks, taking into account the longest paths from start to finish. The function analyzed the process flow by taking into account the predecessor relationships between activities, resulting in an accurate calculation of the process's total cycle time
* The results showed that the total cycle time was influenced by the longest path among all activities, representing the most extended duration from start to finish. This outcome provided valuable insights into the overall efficiency and potential bottlenecks in the business process
* The results of modifying the XPDL files and calculating cycle times showed that BPMN models are flexible and adaptable when representing real-world business processes. The ability to assign random durations and calculate cycle times allows organizations to analyze process variations and understand how different configurations affect efficiency and resource allocation. These key findings are useful for ongoing process improvement and business process management

## (b). Creation of Process Variants

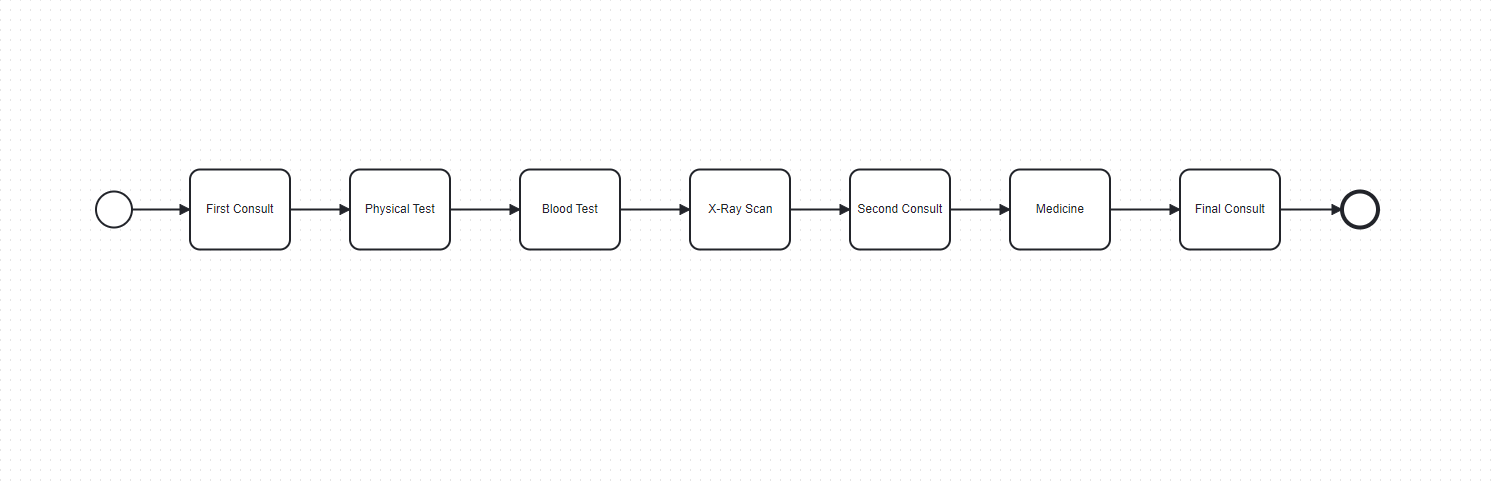
The Event log is provided as a .csv file containing information regarding patient treatment in hospitals and the numerous tasks that were performed on several patients for their treatment.

The Tasks performed on each patient included first consult, second consult, surgery etc. The Rationale behind the variant process creation was that each of the tasks was being performed for each patient in a different order at different times, so if we sorted the event log and mapped out the whole process, it would help us understand the process variants in a better way

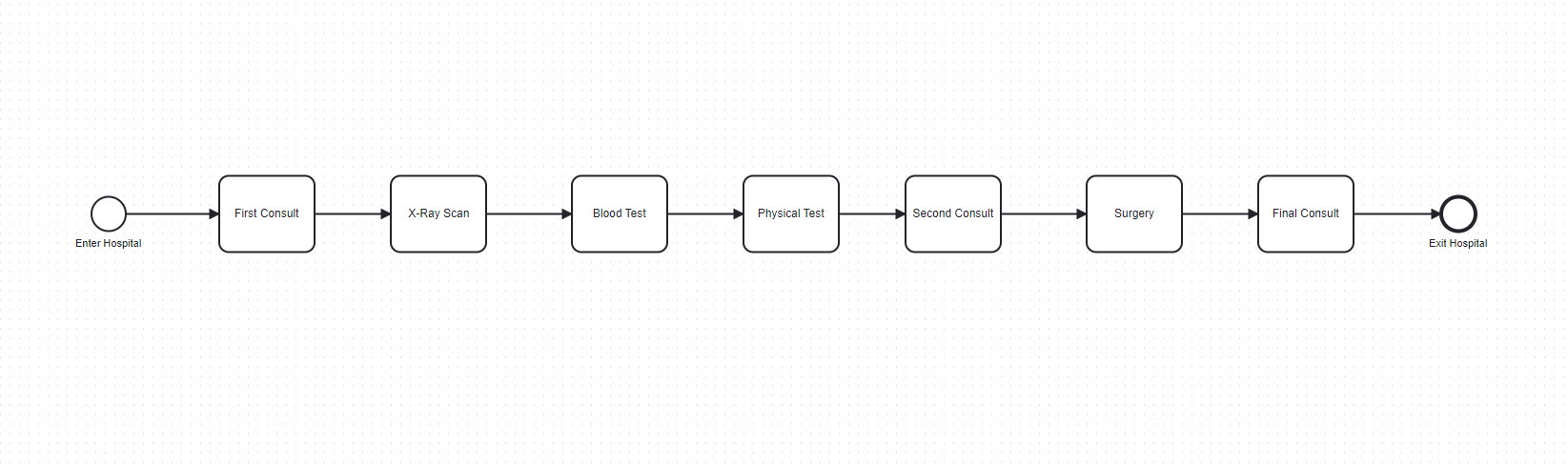
**Steps:**

* The first step we took was picking out specific patients from the event log (one for each group member who would be working on its process variant creation)
* Then we sorted the .csv file based on the patients data (from patient 0 to patient 99)
* After that each group member sorted the tasks being performed on the patients based on time e.g first consult at 11:30 then Blood Test att 11:45 etc.
* From there, each group member created the BPMN Process Variant Model in bpmn.io to accurately represent the process variant
* The Results included creation of 6 Process Variants (corresponding to each group member) which represented how the same process can be performed in different ways and how it can have an effect on the activity and cycle times

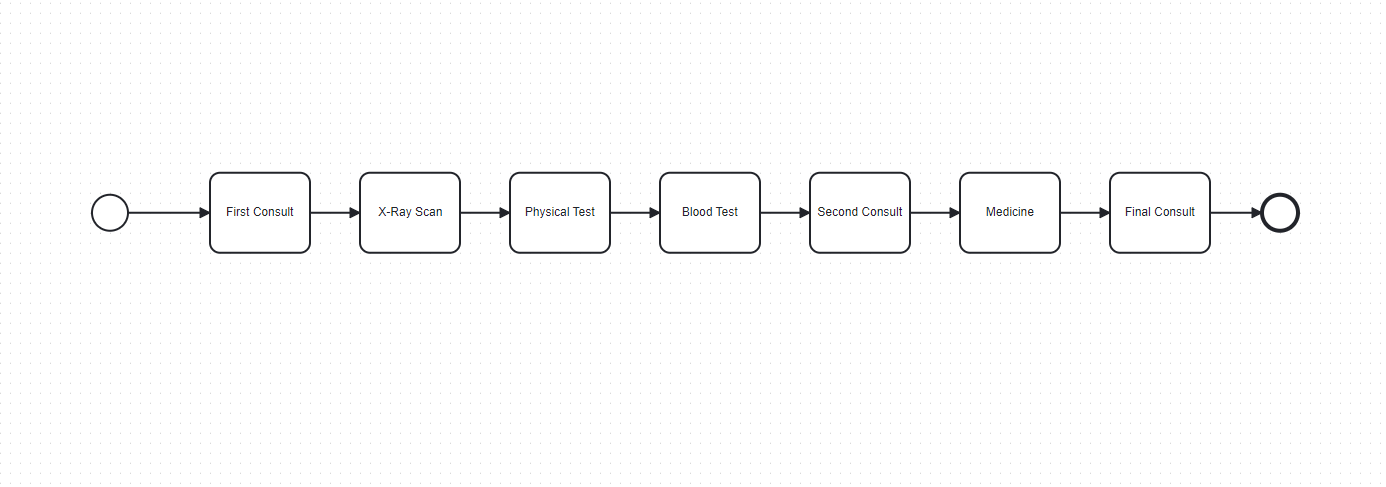
Muhammad Awaimer Zaeem’s (22I-2616) Process Variant 🡪 Patient 10



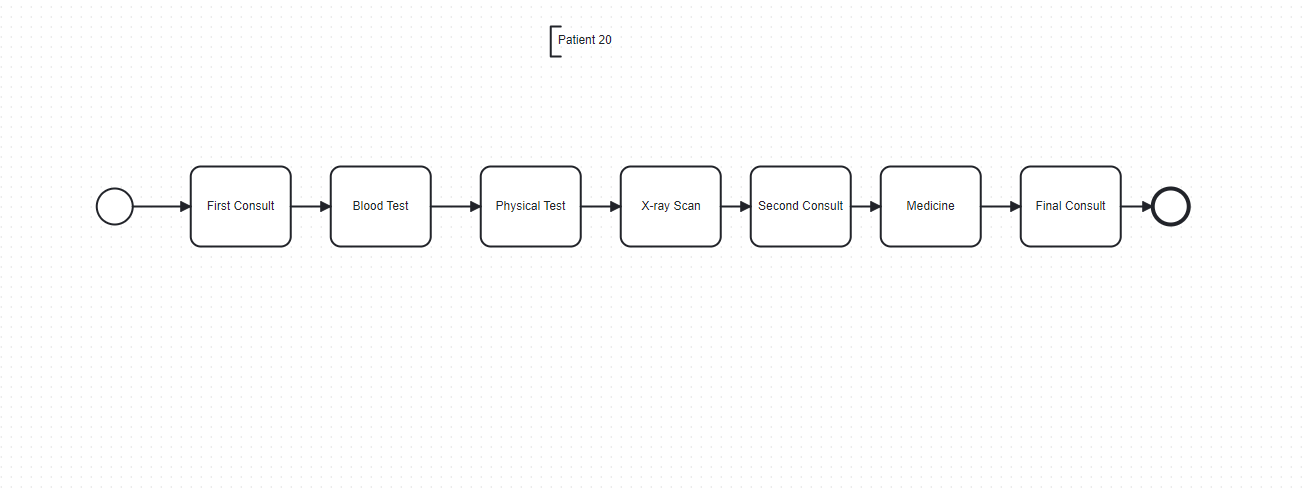
Abdullah Daoud’s (22I-2626) Process Variant 🡪 Patient 21



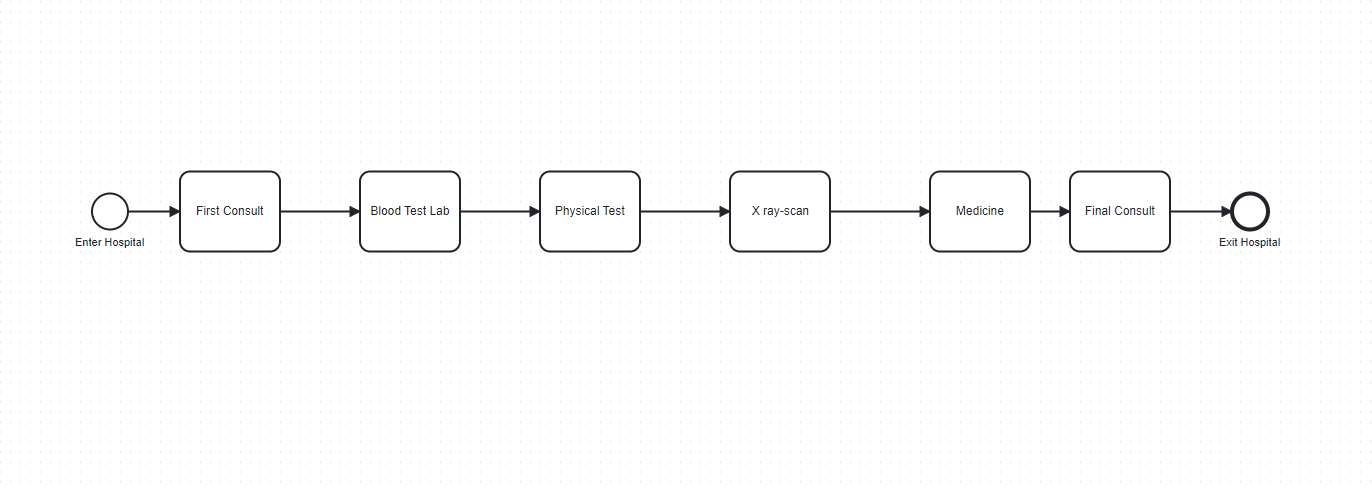
Azghan Ahmed’s (22I-2667) Process Variant 🡪Patient 26



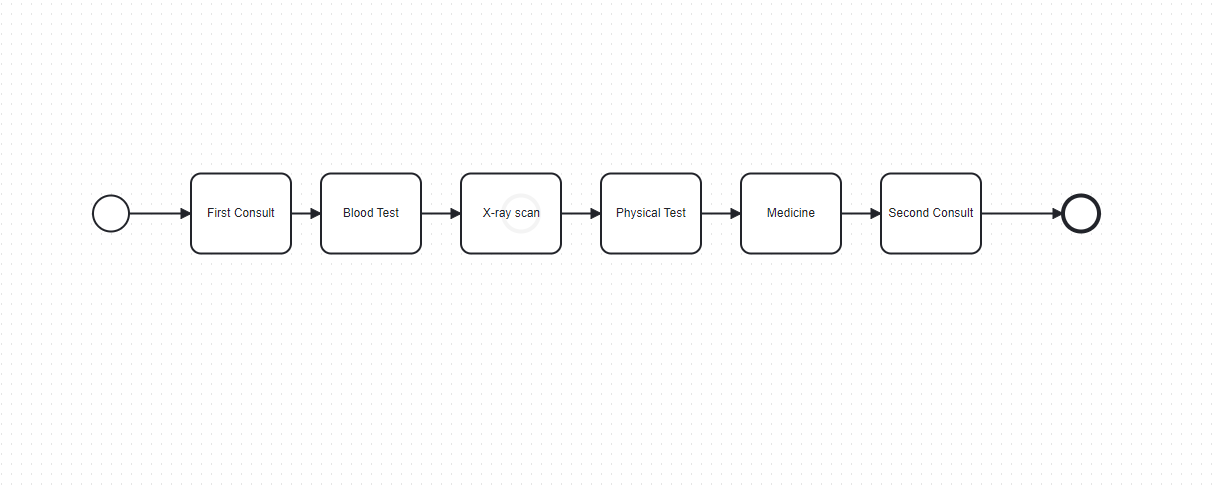
Usman Ali’s (22I-2725) Process Variant 🡪 Patient 20



Faizan Rasheed’s (22I-2734) Process Variant 🡪Patient 22



Amna Asif’s (22I-8777) Process Variant 🡪Patient 38



## (c). Conclusion

The completion of Assignment 4 and the main project on Business Process Model and Notation (BPMN) and XML Process Definition Language (XPDL) provided several valuable insights. Many challenges were faced throughout the assignment and project which lead to creation of new solutions by utilizing our knowledge. Many Key findings were identified which explain the flexibility and adaptability of Business Processes

### (i). Challenges

One of the main challenges we faced was ensuring that the BPMN Models were converted accurately, failure to tackle this challenge would have lead to an inaccurate extraction of crucial data, we tackled this by using tools like bpmn.io and Bizagi to convert our BPMN Models into XPDL Files

Another challenge involved creating a robust parsing tool that could accurately extract and analyze various BPMN elements from the XPDL files. The built-in functions in Java's Document Object Model (DOM) library often generated incorrect results, which lead to us developing our own methods. We implemented error handling and validation techniques to ensure consistent and reliable results during parsing

### (ii). Key Findings

Overall, the project outcomes were positive, demonstrating the importance of precise BPMN modeling and effective XPDL parsing for process analysis. The project also highlighted the flexibility of BPMN Models and XPDL Files, allowing for process modifications and the creation of process variants. These findings are crucial for organizations that are looking to adapt their business processes to changing requirements

### (iii). Future Project Recommendations

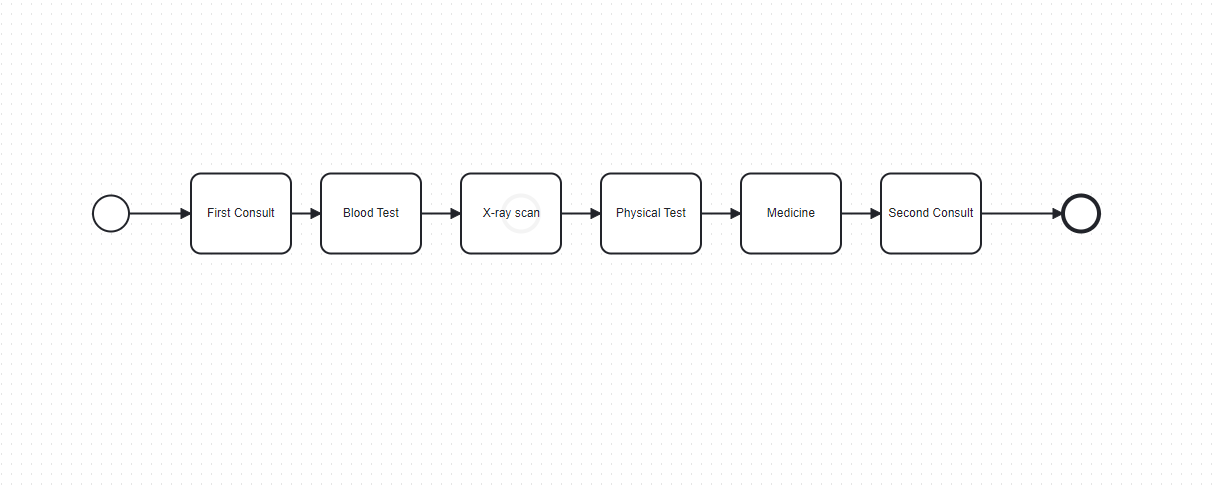
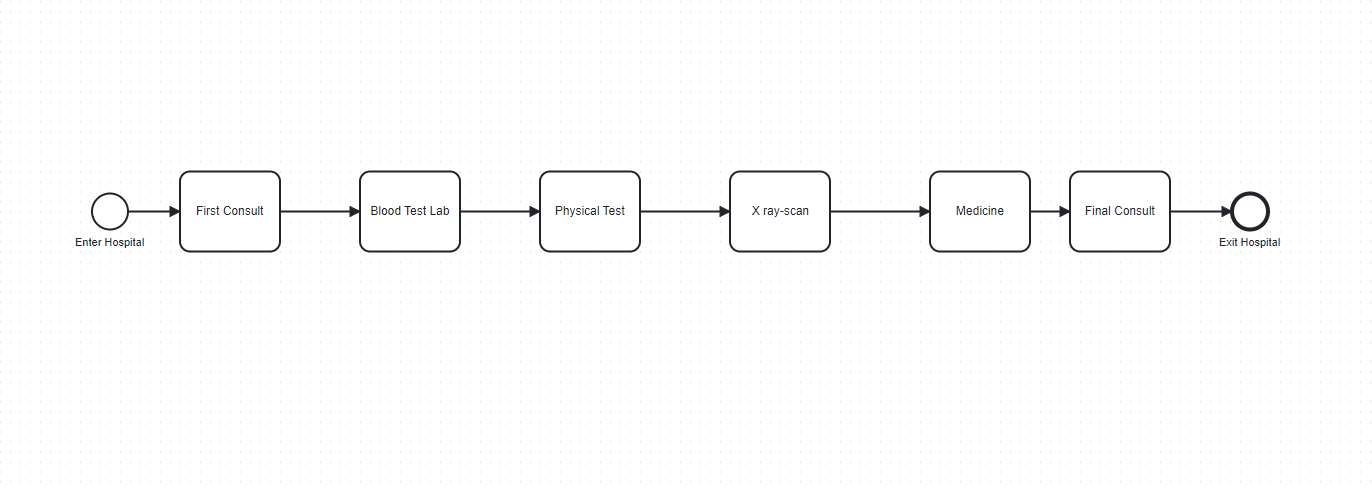
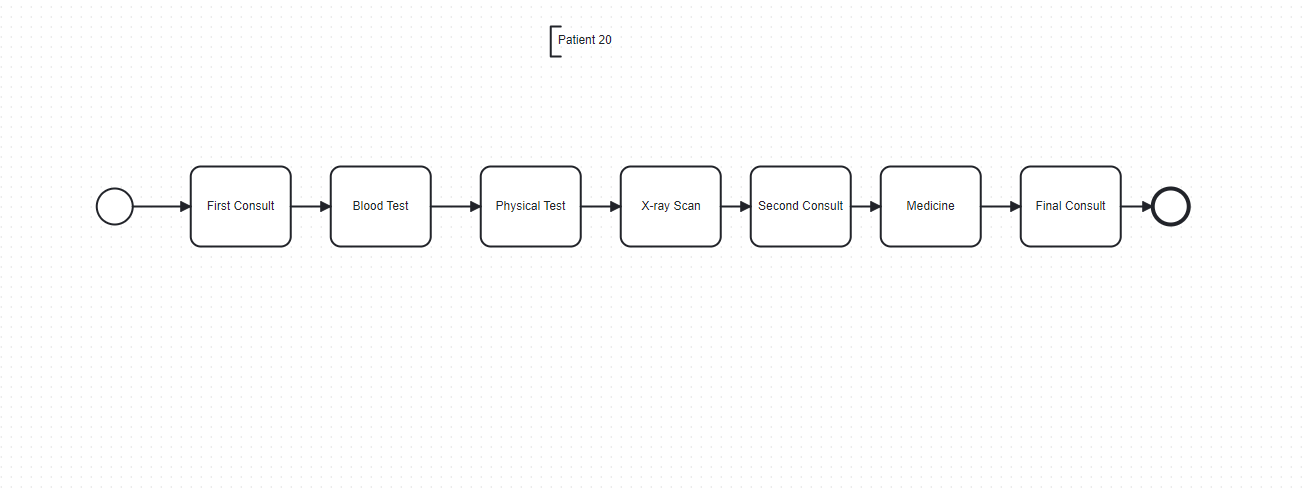
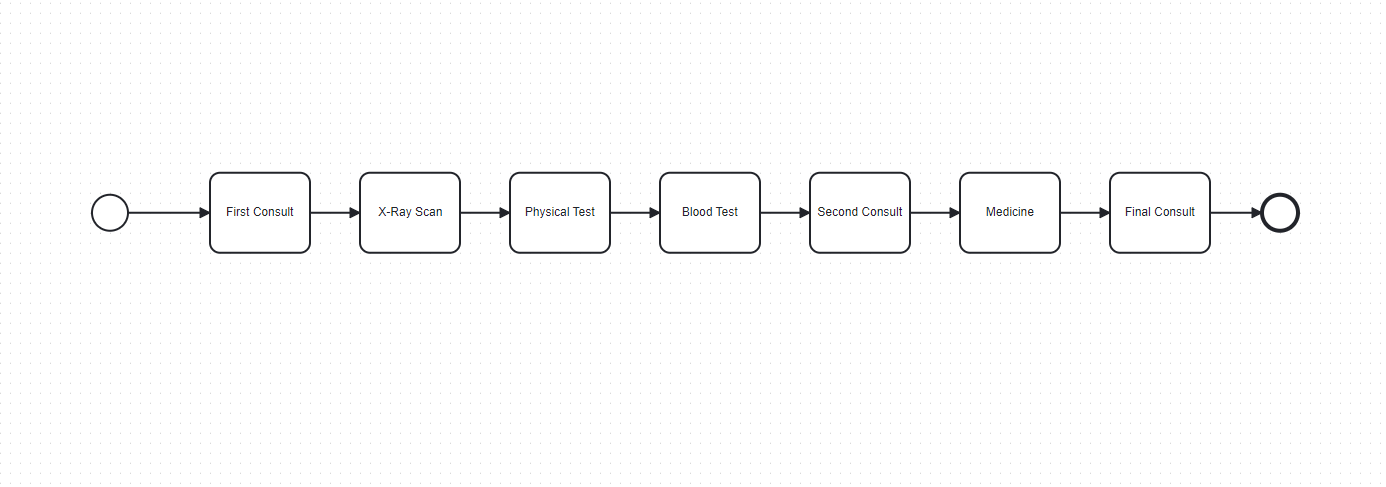
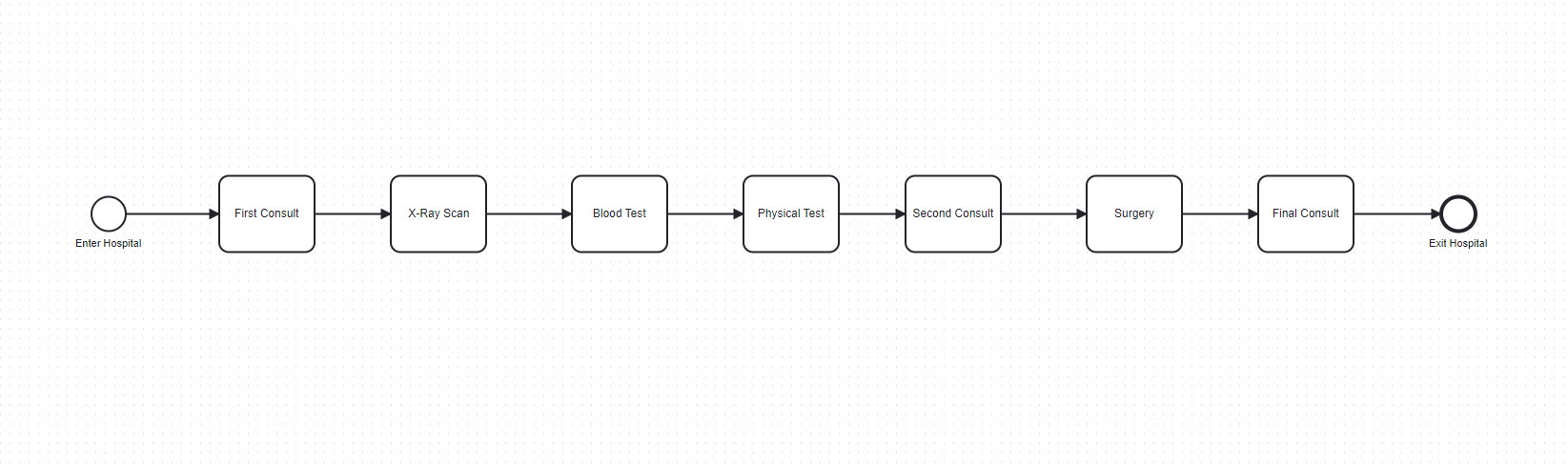
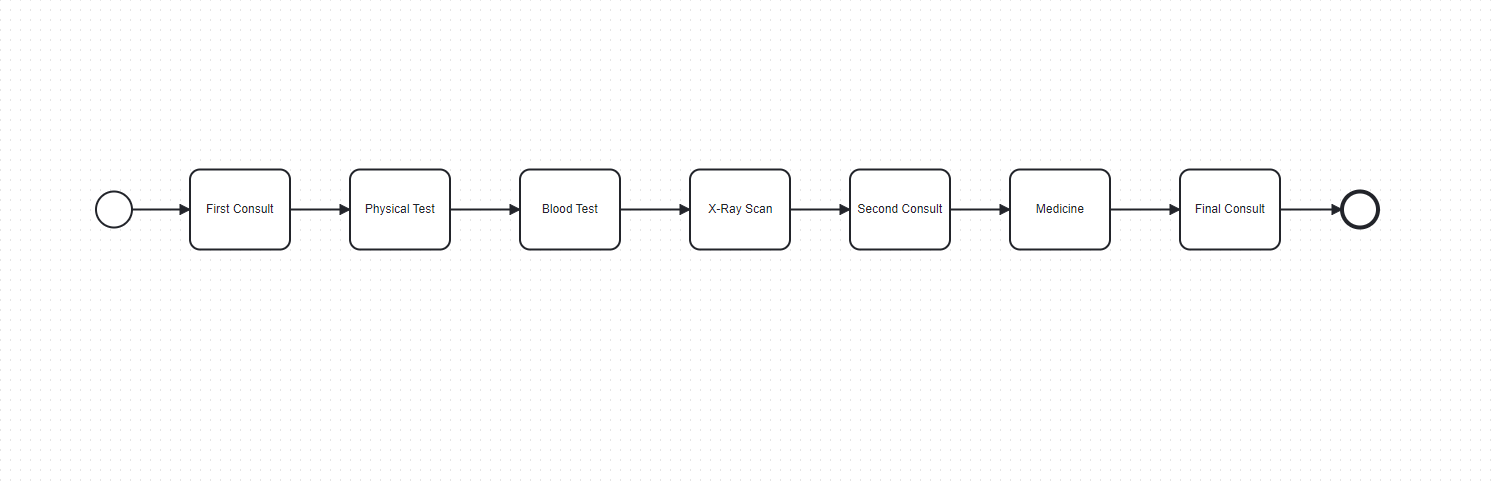
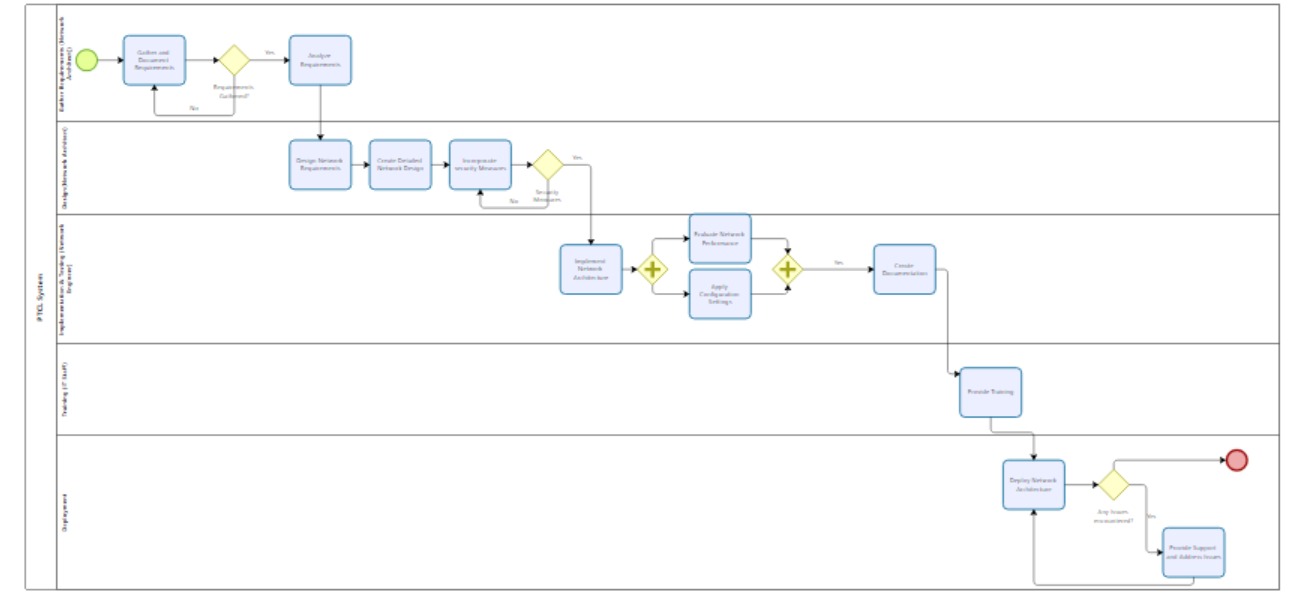
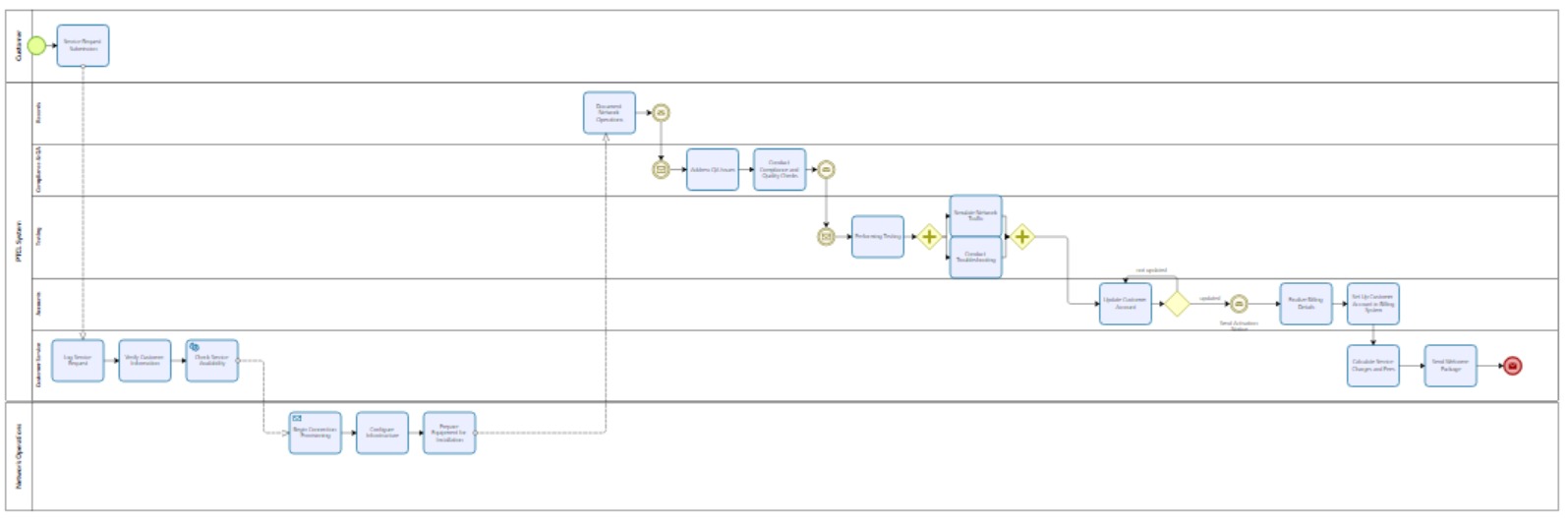
For future projects, we recommend the following solutions:

* Enhanced Tools: Develop more advanced tools for BPMN-to-XPDL conversion to minimize errors and improve compatibility with various process modeling platforms.
* Advanced Parsing Techniques: Explore advanced parsing algorithms to handle complex XPDL structures so that they can be integrated with other potential tools.
* Focus on Process Variants: Focus on process variants to understand how different compositions of processes can impact business process efficiency and cycle times. This can lead to a more dynamic approach to business process management

These recommendations provide a foundation for future research and development in the field of business process modeling, with the goal of contributing to the development of more adaptable and efficient business processes

## (d). Appendices

All material used for Assignment – 4 and Project are listed below:



**Core.java**

**import** org.w3c.dom.\*;  
**import** javax.xml.parsers.\*;  
**import** javax.xml.transform.OutputKeys;  
**import** javax.xml.transform.Transformer;  
**import** javax.xml.transform.TransformerFactory;  
**import** javax.xml.transform.dom.DOMSource;  
**import** javax.xml.transform.stream.StreamResult;  
**import** java.io.\*;  
**import** java.util.\*;  
  
**public class** Core  
{  
 **public static void** main(String[] args)  
 {  
 **try** {  
 // Loading the XML file  
 File xmlFile = **new** File("G:/My Drive/Classroom/Semester - 4/SE-4032 BPE E and G/Project/Core.xpdl");  
 DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder = factory.newDocumentBuilder();  
 Document document = builder.parse(xmlFile);  
 document.getDocumentElement().normalize();  
  
 // Assigning random times to all Tasks  
 *addActivityTime*(document);  
  
 // Calculating total Cycle Time  
 **int** cycleTime = *calculateCT*(document);  
 System.*out*.println("Total Cycle Time: " + cycleTime + " minutes");  
  
 // Saving the updated XML document  
 *saveDocument*(document, "G:/My Drive/Classroom/Semester - 4/SE-4032 BPE E and G/Project/Updated\_Core\_7.xpdl");  
  
  
 // Counting all BPMN elements used in the model  
 **int** totalEventCount = *getElementCount*(document, "Event");  
 **int** startEventCount = *getElementCountWithAttribute*(document, "StartEvent", "Trigger", "None");  
 **int** intermediateEventCount = *getElementCountWithAttribute*(document, "IntermediateEvent", "Trigger", "Message");  
 **int** endEventCount = *getElementCount*(document, "EndEvent");  
  
 **int** totalActivityCount = *getElementCount*(document, "Activity");  
 **int** taskCount = *getElementCount*(document, "Implementation");  
 **int** userTaskCount = *getElementCountWithAttribute*(document, "TaskUser", "Implementation", "Unspecified");  
 **int** serviceTaskCount = *getElementCountWithAttribute*(document, "TaskService", "", "");  
 **int** scriptTaskCount = *getElementCountWithAttribute*(document, "Task", "Type", "Script");  
 **int** manualTaskCount = *getElementCountWithAttribute*(document, "Task", "Type", "Manual");  
 **int** receiveTaskCount = *getElementCount*(document, "TaskReceive"); // New count for receive tasks  
 **int** subProcessCount = *getElementCount*(document, "SubProcess");  
  
 **int** totalGatewayCount = *getElementCount*(document, "Gateway") + *getElementCount*(document, "Route"); // Count both Gateways and Routes  
 **int** exclusiveGatewayCount = *getElementCountWithAttribute*(document, "Route", "GatewayType", "Exclusive");  
 **int** parallelGatewayCount = *getElementCountWithAttribute*(document, "Route", "GatewayType", "Parallel");  
 **int** inclusiveGatewayCount = *getElementCountWithAttribute*(document, "Route", "GatewayType", "Inclusive");  
  
 **int** totalArtifactCount = *getElementCount*(document, "Artifact");  
 **int** dataObjectCount = *getElementCountWithAttribute*(document, "Artifact", "Type", "DataObject");  
 **int** groupCount = *getElementCountWithAttribute*(document, "Artifact", "Type", "Group");  
 **int** annotationCount = *getElementCountWithAttribute*(document, "Artifact", "Type", "Annotation");  
  
 **int** totalConnectingObjectCount = *getElementCount*(document, "SequenceFlow") + *getElementCount*(document, "MessageFlow") + *getElementCount*(document, "Association");  
  
 **int** sequenceFlowCount = *getElementCount*(document, "Transition");  
 **int** messageFlowCount = *getElementCount*(document, "MessageFlow");  
 **int** associationCount = *getElementCount*(document, "Association");  
  
 **int** totalSwimlaneCount = *getElementCount*(document, "Lane") + *getElementCount*(document, "Pool");  
 **int** poolCount = *getElementCount*(document, "Pool");  
 **int** laneCount = *getElementCount*(document, "Lane");  
  
 // Generating Report  
 System.*out*.println("BPMN Model Elements:");  
 System.*out*.println("Total Events: " + totalEventCount);  
 System.*out*.println("Start Events: " + startEventCount);  
 System.*out*.println("Intermediate Events: " + intermediateEventCount);  
 System.*out*.println("End Events: " + endEventCount);  
 System.*out*.println("Total Activities: " + totalActivityCount);  
 System.*out*.println("Tasks: " + taskCount);  
 System.*out*.println("User Tasks: " + userTaskCount);  
 System.*out*.println("Service Tasks: " + serviceTaskCount);  
 System.*out*.println("Script Tasks: " + scriptTaskCount);  
 System.*out*.println("Manual Tasks: " + manualTaskCount);  
 System.*out*.println("Receive Tasks: " + receiveTaskCount);  
 System.*out*.println("Sub Processes: " + subProcessCount);  
 System.*out*.println("Total Gateways: " + totalGatewayCount);  
 System.*out*.println("Exclusive Gateways (XOR): " + exclusiveGatewayCount);  
 System.*out*.println("Parallel Gateways (AND): " + parallelGatewayCount);  
 System.*out*.println("Inclusive Gateways (OR): " + inclusiveGatewayCount);  
 System.*out*.println("Total Artifacts: " + totalArtifactCount);  
 System.*out*.println("Data Objects: " + dataObjectCount);  
 System.*out*.println("Groups: " + groupCount);  
 System.*out*.println("Annotations: " + annotationCount);  
 System.*out*.println("Total Connecting Objects: " + totalConnectingObjectCount);  
 System.*out*.println("Sequence Flows: " + sequenceFlowCount);  
 System.*out*.println("Message Flows: " + messageFlowCount);  
 System.*out*.println("Associations: " + associationCount);  
 System.*out*.println("Total Swimlanes: " + (totalSwimlaneCount-1));  
 System.*out*.println("Pools: " + (poolCount-1));  
 System.*out*.println("Lanes: " + laneCount);  
  
 }  
 **catch** (Exception e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 // addActivityTime function assigns random times between 5 to 15 minutes to all activity tags containing both "Implementation" and "Task" Tags  
 **private static void** addActivityTime(Document document)  
 {  
 NodeList activities = document.getElementsByTagName("Activity");  
 Random random = **new** Random();  
  
 **for** (**int** i = 0; i < activities.getLength(); i++)  
 {  
 Node node = activities.item(i);  
 **if** (node.getNodeType() == Node.*ELEMENT\_NODE*)  
 {  
 Element activity = (Element) node;  
  
 // Boolean variable to check for "Implementation" and "Task" tags inside "Activity" tags  
 **boolean** hasTaskInImplementation = **false**;  
  
 // Looping through children of "Activity" tags to find "Implementation" and "Task" tags  
 NodeList children = activity.getChildNodes();  
  
 **for** (**int** j = 0; j < children.getLength(); j++)  
 {  
 Node child = children.item(j);  
 **if** (child.getNodeType() == Node.*ELEMENT\_NODE* && child.getNodeName().equals("Implementation"))  
 {  
 NodeList implementationChildren = child.getChildNodes();  
 **for** (**int** k = 0; k < implementationChildren.getLength(); k++)  
 {  
 Node implementationChild = implementationChildren.item(k);  
 **if** (implementationChild.getNodeType() == Node.*ELEMENT\_NODE* && implementationChild.getNodeName().equals("Task"))  
 {  
 hasTaskInImplementation = **true**;  
 **break**;  
 }  
 }  
 }  
 }  
  
 // If both "Implementation" and "Task" are found, assign a random time between 5 to 15 minutes  
 **if** (hasTaskInImplementation)  
 {  
 **int** randomTime = 5 + random.nextInt(11); // 11 for inclusive range [5, 15]  
 activity.setAttribute("Duration", randomTime + " minutes");  
 }  
 }  
 }  
 }  
  
 // Helper function to save the updated XML document  
 **private static void** saveDocument(Document document, String outputFilePath) **throws** Exception  
 {  
 TransformerFactory transformerFactory = TransformerFactory.*newInstance*();  
 Transformer transformer = transformerFactory.newTransformer();  
 transformer.setOutputProperty(OutputKeys.*INDENT*, "yes");  
 DOMSource source = **new** DOMSource(document);  
 StreamResult result = **new** StreamResult(**new** File(outputFilePath));  
 transformer.transform(source, result);  
 }  
  
 // Helper function to count elements by tag name  
 **private static int** getElementCount(Document document, String tagName)  
 {  
 NodeList nodeList = document.getElementsByTagName(tagName);  
 **return** nodeList.getLength();  
 }  
  
 // Helper function to count elements with a specific attribute value  
 **private static int** getElementCountWithAttribute(Document document, String tagName, String attributeName, String attributeValue)  
 {  
 NodeList nodeList = document.getElementsByTagName(tagName);  
 **int** count = 0;  
  
 **for** (**int** i = 0; i < nodeList.getLength(); i++)  
 {  
 Node node = nodeList.item(i);  
  
 **if** (node.getNodeType() == Node.*ELEMENT\_NODE*)  
 {  
 Element element = (Element) node;  
 String value = element.getAttribute(attributeName);  
  
 **if** (value.equals(attributeValue))  
 {  
 count++;  
 }  
 }  
 }  
 **return** count;  
 }  
  
 // Function to calculate Total Cycle Time (CT) of the process  
 **private static int** calculateCT(Document document)  
 {  
 Map<String, Integer> activityDurations = **new** HashMap<>();  
 Map<String, List<String>> predecessors = **new** HashMap<>();  
  
 // Populating activity durations and predecessors  
 NodeList activities = document.getElementsByTagName("Activity");  
  
 **for** (**int** i = 0; i < activities.getLength(); i++)  
 {  
 Element activity = (Element) activities.item(i);  
 String activityId = activity.getAttribute("Id");  
 String durationStr = activity.getAttribute("Duration");  
  
 **if** (!durationStr.isEmpty())  
 {  
 **int** duration = Integer.*parseInt*(durationStr.split("\\s+")[0]);  
 activityDurations.put(activityId, duration);  
 }  
  
 NodeList incomingFlows = activity.getElementsByTagName("Incoming");  
 List<String> predList = **new** ArrayList<>();  
  
 **for** (**int** j = 0; j < incomingFlows.getLength(); j++)  
 {  
 Element incomingFlow = (Element) incomingFlows.item(j);  
 String sourceId = incomingFlow.getAttribute("Source");  
 predList.add(sourceId);  
 }  
  
 predecessors.put(activityId, predList);  
 }  
  
 // Calculating total duration considering parallel paths  
 **int** totalDuration = 0;  
  
 **for** (String activityId : activityDurations.keySet())  
 {  
 **int** maxPathDuration = *getMaxPathDuration*(activityId, activityDurations, predecessors);  
 totalDuration = Math.*max*(totalDuration, maxPathDuration);  
 }  
  
 **return** totalDuration;  
 }  
  
 // Helper function to calculate maximum path duration  
 **private static int** getMaxPathDuration(String activityId, Map<String, Integer> activityDurations, Map<String, List<String>> predecessors)  
 {  
 **if** (!predecessors.containsKey(activityId) || predecessors.get(activityId).isEmpty())  
 {  
 **return** activityDurations.get(activityId);  
 }  
  
 **int** maxPredecessorDuration = 0;  
  
 **for** (String predId : predecessors.get(activityId))  
 {  
 **int** predDuration = *getMaxPathDuration*(predId, activityDurations, predecessors);  
 maxPredecessorDuration = Math.*max*(maxPredecessorDuration, predDuration);  
 }  
  
 **return** maxPredecessorDuration + activityDurations.get(activityId);  
 }  
}

**Support.java**

**import** org.w3c.dom.\*;  
**import** javax.xml.parsers.\*;  
**import** javax.xml.transform.OutputKeys;  
**import** javax.xml.transform.Transformer;  
**import** javax.xml.transform.TransformerFactory;  
**import** javax.xml.transform.dom.DOMSource;  
**import** javax.xml.transform.stream.StreamResult;  
**import** java.io.\*;  
**import** java.util.\*;  
  
**public class** Support  
{  
 **public static void** main(String[] args)  
 {  
 **try** {  
 // Loading the XML file  
 File xmlFile = **new** File("G:/My Drive/Classroom/Semester - 4/SE-4032 BPE E and G/Project/Support.xpdl");  
 DocumentBuilderFactory factory = DocumentBuilderFactory.*newInstance*();  
 DocumentBuilder builder = factory.newDocumentBuilder();  
 Document document = builder.parse(xmlFile);  
 document.getDocumentElement().normalize();  
  
 // Assigning random times to all Tasks  
 *addActivityTime*(document);  
  
 // Calculating total Cycle Time  
 **int** cycleTime = *calculateCT*(document);  
 System.*out*.println("Total Cycle Time: " + cycleTime + " minutes");  
  
 // Saving the updated XML document  
 *saveDocument*(document, "G:/My Drive/Classroom/Semester - 4/SE-4032 BPE E and G/Project/Updated\_Support\_7.xpdl");  
  
  
 // Counting all BPMN elements used in the model  
 **int** totalEventCount = *getElementCount*(document, "Event");  
 **int** startEventCount = *getElementCountWithAttribute*(document, "StartEvent", "Trigger", "None");  
 **int** intermediateEventCount = *getElementCountWithAttribute*(document, "IntermediateEvent", "Trigger", "Message");  
 **int** endEventCount = *getElementCount*(document, "EndEvent");  
  
 **int** totalActivityCount = *getElementCount*(document, "Activity");  
 **int** taskCount = *getElementCount*(document, "Implementation");  
 **int** userTaskCount = *getElementCountWithAttribute*(document, "TaskUser", "Implementation", "Unspecified");  
 **int** serviceTaskCount = *getElementCountWithAttribute*(document, "TaskService", "", "");  
 **int** scriptTaskCount = *getElementCountWithAttribute*(document, "Task", "Type", "Script");  
 **int** manualTaskCount = *getElementCountWithAttribute*(document, "Task", "Type", "Manual");  
 **int** receiveTaskCount = *getElementCount*(document, "TaskReceive"); // New count for receive tasks  
 **int** subProcessCount = *getElementCount*(document, "SubProcess");  
  
 **int** totalGatewayCount = *getElementCount*(document, "Gateway") + *getElementCount*(document, "Route"); // Count both Gateways and Routes  
 **int** exclusiveGatewayCount = *getElementCountWithAttribute*(document, "Route", "GatewayType", "Exclusive");  
 **int** parallelGatewayCount = *getElementCountWithAttribute*(document, "Route", "GatewayType", "Parallel");  
 **int** inclusiveGatewayCount = *getElementCountWithAttribute*(document, "Route", "GatewayType", "Inclusive");  
  
 **int** totalArtifactCount = *getElementCount*(document, "Artifact");  
 **int** dataObjectCount = *getElementCountWithAttribute*(document, "Artifact", "Type", "DataObject");  
 **int** groupCount = *getElementCountWithAttribute*(document, "Artifact", "Type", "Group");  
 **int** annotationCount = *getElementCountWithAttribute*(document, "Artifact", "Type", "Annotation");  
  
 **int** totalConnectingObjectCount = *getElementCount*(document, "SequenceFlow") + *getElementCount*(document, "MessageFlow") + *getElementCount*(document, "Association");  
  
 **int** sequenceFlowCount = *getElementCount*(document, "Transition");  
 **int** messageFlowCount = *getElementCount*(document, "MessageFlow");  
 **int** associationCount = *getElementCount*(document, "Association");  
  
 **int** totalSwimlaneCount = *getElementCount*(document, "Lane") + *getElementCount*(document, "Pool");  
 **int** poolCount = *getElementCount*(document, "Pool");  
 **int** laneCount = *getElementCount*(document, "Lane");  
  
 // Generating Report  
 System.*out*.println("BPMN Model Elements:");  
 System.*out*.println("Total Events: " + totalEventCount);  
 System.*out*.println("Start Events: " + startEventCount);  
 System.*out*.println("Intermediate Events: " + intermediateEventCount);  
 System.*out*.println("End Events: " + endEventCount);  
 System.*out*.println("Total Activities: " + totalActivityCount);  
 System.*out*.println("Tasks: " + taskCount);  
 System.*out*.println("User Tasks: " + userTaskCount);  
 System.*out*.println("Service Tasks: " + serviceTaskCount);  
 System.*out*.println("Script Tasks: " + scriptTaskCount);  
 System.*out*.println("Manual Tasks: " + manualTaskCount);  
 System.*out*.println("Receive Tasks: " + receiveTaskCount);  
 System.*out*.println("Sub Processes: " + subProcessCount);  
 System.*out*.println("Total Gateways: " + totalGatewayCount);  
 System.*out*.println("Exclusive Gateways (XOR): " + exclusiveGatewayCount);  
 System.*out*.println("Parallel Gateways (AND): " + parallelGatewayCount);  
 System.*out*.println("Inclusive Gateways (OR): " + inclusiveGatewayCount);  
 System.*out*.println("Total Artifacts: " + totalArtifactCount);  
 System.*out*.println("Data Objects: " + dataObjectCount);  
 System.*out*.println("Groups: " + groupCount);  
 System.*out*.println("Annotations: " + annotationCount);  
 System.*out*.println("Total Connecting Objects: " + totalConnectingObjectCount);  
 System.*out*.println("Sequence Flows: " + sequenceFlowCount);  
 System.*out*.println("Message Flows: " + messageFlowCount);  
 System.*out*.println("Associations: " + associationCount);  
 System.*out*.println("Total Swimlanes: " + (totalSwimlaneCount-1));  
 System.*out*.println("Pools: " + (poolCount-1));  
 System.*out*.println("Lanes: " + laneCount);  
  
 }  
 **catch** (Exception e)  
 {  
 e.printStackTrace();  
 }  
 }  
  
 // addActivityTime function assigns random times between 5 to 15 minutes to all activity tags containing both "Implementation" and "Task" Tags  
 **private static void** addActivityTime(Document document)  
 {  
 NodeList activities = document.getElementsByTagName("Activity");  
 Random random = **new** Random();  
  
 **for** (**int** i = 0; i < activities.getLength(); i++)  
 {  
 Node node = activities.item(i);  
 **if** (node.getNodeType() == Node.*ELEMENT\_NODE*)  
 {  
 Element activity = (Element) node;  
  
 // Boolean variable to check for "Implementation" and "Task" tags inside "Activity" tags  
 **boolean** hasTaskInImplementation = **false**;  
  
 // Looping through children of "Activity" tags to find "Implementation" and "Task" tags  
 NodeList children = activity.getChildNodes();  
  
 **for** (**int** j = 0; j < children.getLength(); j++)  
 {  
 Node child = children.item(j);  
 **if** (child.getNodeType() == Node.*ELEMENT\_NODE* && child.getNodeName().equals("Implementation"))  
 {  
 NodeList implementationChildren = child.getChildNodes();  
 **for** (**int** k = 0; k < implementationChildren.getLength(); k++)  
 {  
 Node implementationChild = implementationChildren.item(k);  
 **if** (implementationChild.getNodeType() == Node.*ELEMENT\_NODE* && implementationChild.getNodeName().equals("Task"))  
 {  
 hasTaskInImplementation = **true**;  
 **break**;  
 }  
 }  
 }  
 }  
  
 // If both "Implementation" and "Task" are found, assign a random time between 5 to 15 minutes  
 **if** (hasTaskInImplementation)  
 {  
 **int** randomTime = 5 + random.nextInt(11); // 11 for inclusive range [5, 15]  
 activity.setAttribute("Duration", randomTime + " minutes");  
 }  
 }  
 }  
 }  
  
 // Helper function to save the updated XML document  
 **private static void** saveDocument(Document document, String outputFilePath) **throws** Exception  
 {  
 TransformerFactory transformerFactory = TransformerFactory.*newInstance*();  
 Transformer transformer = transformerFactory.newTransformer();  
 transformer.setOutputProperty(OutputKeys.*INDENT*, "yes");  
 DOMSource source = **new** DOMSource(document);  
 StreamResult result = **new** StreamResult(**new** File(outputFilePath));  
 transformer.transform(source, result);  
 }  
  
 // Helper function to count elements by tag name  
 **private static int** getElementCount(Document document, String tagName)  
 {  
 NodeList nodeList = document.getElementsByTagName(tagName);  
 **return** nodeList.getLength();  
 }  
  
 // Helper function to count elements with a specific attribute value  
 **private static int** getElementCountWithAttribute(Document document, String tagName, String attributeName, String attributeValue)  
 {  
 NodeList nodeList = document.getElementsByTagName(tagName);  
 **int** count = 0;  
  
 **for** (**int** i = 0; i < nodeList.getLength(); i++)  
 {  
 Node node = nodeList.item(i);  
  
 **if** (node.getNodeType() == Node.*ELEMENT\_NODE*)  
 {  
 Element element = (Element) node;  
 String value = element.getAttribute(attributeName);  
  
 **if** (value.equals(attributeValue))  
 {  
 count++;  
 }  
 }  
 }  
 **return** count;  
 }  
  
 // Function to calculate Total Cycle Time (CT) of the process  
 **private static int** calculateCT(Document document)  
 {  
 Map<String, Integer> activityDurations = **new** HashMap<>();  
 Map<String, List<String>> predecessors = **new** HashMap<>();  
  
 // Populating activity durations and predecessors  
 NodeList activities = document.getElementsByTagName("Activity");  
  
 **for** (**int** i = 0; i < activities.getLength(); i++)  
 {  
 Element activity = (Element) activities.item(i);  
 String activityId = activity.getAttribute("Id");  
 String durationStr = activity.getAttribute("Duration");  
  
 **if** (!durationStr.isEmpty())  
 {  
 **int** duration = Integer.*parseInt*(durationStr.split("\\s+")[0]);  
 activityDurations.put(activityId, duration);  
 }  
  
 NodeList incomingFlows = activity.getElementsByTagName("Incoming");  
 List<String> predList = **new** ArrayList<>();  
  
 **for** (**int** j = 0; j < incomingFlows.getLength(); j++)  
 {  
 Element incomingFlow = (Element) incomingFlows.item(j);  
 String sourceId = incomingFlow.getAttribute("Source");  
 predList.add(sourceId);  
 }  
  
 predecessors.put(activityId, predList);  
 }  
  
 // Calculating total duration considering parallel paths  
 **int** totalDuration = 0;  
  
 **for** (String activityId : activityDurations.keySet())  
 {  
 **int** maxPathDuration = *getMaxPathDuration*(activityId, activityDurations, predecessors);  
 totalDuration = Math.*max*(totalDuration, maxPathDuration);  
 }  
  
 **return** totalDuration;  
 }  
  
 // Helper function to calculate maximum path duration  
 **private static int** getMaxPathDuration(String activityId, Map<String, Integer> activityDurations, Map<String, List<String>> predecessors)  
 {  
 **if** (!predecessors.containsKey(activityId) || predecessors.get(activityId).isEmpty())  
 {  
 **return** activityDurations.get(activityId);  
 }  
  
 **int** maxPredecessorDuration = 0;  
  
 **for** (String predId : predecessors.get(activityId))  
 {  
 **int** predDuration = *getMaxPathDuration*(predId, activityDurations, predecessors);  
 maxPredecessorDuration = Math.*max*(maxPredecessorDuration, predDuration);  
 }  
  
 **return** maxPredecessorDuration + activityDurations.get(activityId);  
 }  
}

# 5. References

(n.d.). Retrieved from BPMN.io: https://bpmn.io

(n.d.). Retrieved from Bizagi: https://www.bizagi.com/en

*HTML DOM Document getDocumentElement*. (n.d.). Retrieved from W3Schools: https://www.w3schools.com/jsref/prop\_document\_documentelement.asp

*HTML DOM Document getElementsByTagName*. (n.d.). Retrieved from W3Schools: https://www.w3schools.com/jsref/met\_document\_getelementsbytagname.asp

IntelliJ for writing Java Code