Integrative Project Assignment

sem3-pi-2024-25 - version 1.7

November 28, 2024

Abstract

This document describes the practical work to be developed within the Integrative Project for the 3rd Semester courses: ARQCP, BDDAD, ESINF, FSIAP, and LAPR3. The assignment involves the development of a computer solution to support production management in an industrial facility. This document briefly introduces the problem to be solved, and some fundamental concepts of Production Management, in general, and of Product Engineering, in particular. Finally, the functional and non-functional requirements of the solution to be developed are presented as well as the operational and technical details of the Integrative Project.

Table 1: Version History

Version	**Description**
1.0	Kick-off version
1.1	Clarifications on ESINF/Sprint 1, workstations and machines
1.2	US for sprint 2 for ARQCP, BDDAD and FSIAP
1.3	US for sprint 2 for ESINF and LAPR3
1.4	New section with a full example for BOO and BOM; US for ESINF updated; .cvs formats added (BDDAD, ESINF, LAPR3)
1.4.1	Format for boo.csv updated to clarify operations dependencies
1.5	Added US for sprint 3 - BDDAD and ESINF
1.6	Added US for sprint 3 - FSIAP and ARQCP
1.7	Added US for sprint 3 - LAPR3 - and typos corrections

Contents

1 Integrative Project

In this project, students should be able to analyze, design, and implement a computer-based solution to support production management in industrial facilities, focused on short-run production and project-based manufacturing. This project serves as a proof-of-concept covering a set of relevant modules for production management, such as Product Engineering, Production Planning, and Production Control. To develop these modules, it will be necessary, among other aspects, to support the appropriate management of the plant floor, products, machines, and operations; to manage orders and production orders within the production plan context; to develop tools for simulating production scenarios; to support project management comprising the definition of tasks and critical paths; to monitor machines in a production context; and to plan the equipment preventive maintenance.

In line with the best practices learned in the course, particularly in the subjects Computer Architecture (ARQCP), Databases (BDDAD), Information Structures (ESINF), Applied Physics (FSIAP), and Laboratory-Project III (LAPR3), an iterative and incremental process will be used. Therefore, an agile approach based on SCRUM will be applied in team management, structured in three sprints of four weeks each.

The solution to be developed should consist of a set of applications created in PL/SQL, Java, and C/Assembly, depending on the requirements. To enhance the solution's maintainability and align with software development best practices, the implementation must follow a Test-Driven Development (TDD) approach.

2 Description of the Problem

This section describes the problem to be solved: the development of a software solution that addresses critical aspects of planning, managing, and controlling production in an industrial facility. The industrial management problem is thus presented, along with its specificities, and the most relevant associated concepts, particularly aspects related to: Product Engineering, such as the structure of the facility (e.g., products, machines, operations, and layouts); Production Planning, processing customer orders, generating production orders; and Production Control, such supervising machine operation and scheduling preventing maintenance.

2.1 Product Engineering

The company KeepItSimple Solutions (KS) is specialized in creating software solutions for a wide range of businesses. Recently, an opportunity was identified to develop a solution for the planning, management, and production control of flexible industrial units suited to short-run production and/or project-based manufacturing. The system could be used across multiple sectors (e.g., furniture, metalworking, cork, footwear, cutlery). As a proof of concept, the goal is to develop a solution focused on Production Management.

Product Engineering encompasses the activities related to the conception and design of the product and its components, as well as load analysis and planning. These

elements are crucial for determining the products/projects' technical and economic feasibility.

2.2 Factory Facilities

The factory plant is designed according to multiple factors, such as type of industry, safety issues, common sequences of operations, efficiency concerns, space limitations resulting from the organic evolution of the facility over time, access to raw materials warehouses, components, and/or finished products (see figure 1).

Figure 1: Manufacturing Plant

In a factory, one or numerous products can be produced, depending on the market demand, optimization, production costs, and flexibility intended for the facility. The classification of the facilities can vary from "Process" (a limited number of products, high optimization) to Project (unique products, typically with complex structures, and long lead times), including JobShops (a wide variety of products, short runs, high flexibility). Flow-shop is a specific case of a JobShop (where products are limited to a restricted number of product families). In the context of the Integrative Project, the focus is on Project-and-JobShop-oriented facilities.

2.3 Products, Components and Raw Materials

The items produced in a manufacturing facility can be described using a tree-like structure (see figure 2), named BOM (Bill of Materials). The tree represents final products (root), raw materials (leaf nodes), and components (intermediate nodes), as well as the required quantities of each of those elements to produce the final product (see figure 3; note that, in this diagram, the assembly operations are represented on the branches).

It is relevant to note that the classification is not rigid; it depends on the manufacturing process. For example, in producing a bicycle, the factory uses a component (e.g., front brake) that was a final product in another manufacturing unit. Similarly, raw material is the result of a Process industry that performs one or more operations of

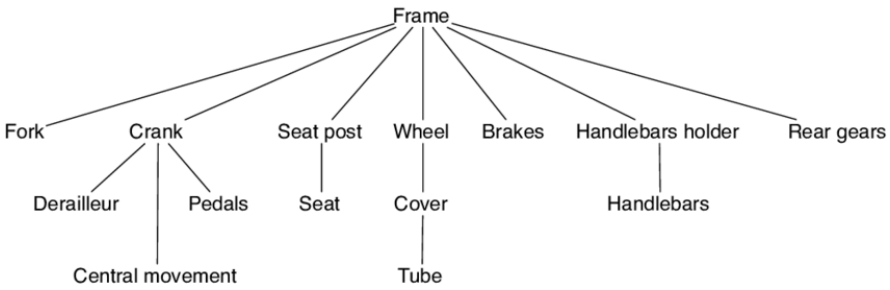

Figure 2: Product Structure of a Bicycle

extraction, refining, processing, and packaging of that raw material. The raw material in question is the final product for those manufacturing units.

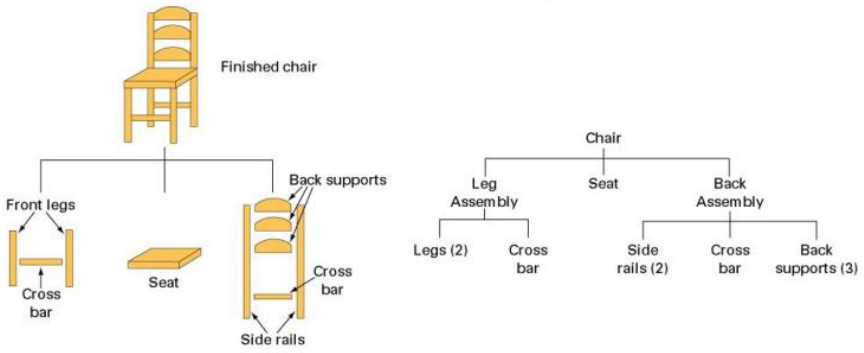

Figure 3: Product Structure of a Chair

2.4 Operations, Stations and Layouts

The plant floor comprises a set of stations organized according to the layout (see figure 4 for an example). These stations can be composed of robotic units, automated machines, machines operated by human operators, or human operators' workstations. The stations are capable of performing a set of manufacturing operations that depend on the type of industry; for example, in the furniture industry, the most common operations include cutting, drilling, glueing, assembly, polishing, varnishing, and painting.

A particular product is subjected to several operations with one or more possible sequences throughout its production process. The different sequences can be represented by a tree-like structure known as the Bill of Operations (BOO). Products can be organized into families, where a family consists of products with a similar operation sequence.

2.5 Production Planning

The production planning module generates the production plan for a specific time. The production plan is defined based on the aggregated needs of customer orders, and it is limited by the constraints set in the product tree (BOM) and in the operational ranges of the items to be produced (BOO). The production plan can be instantiated as

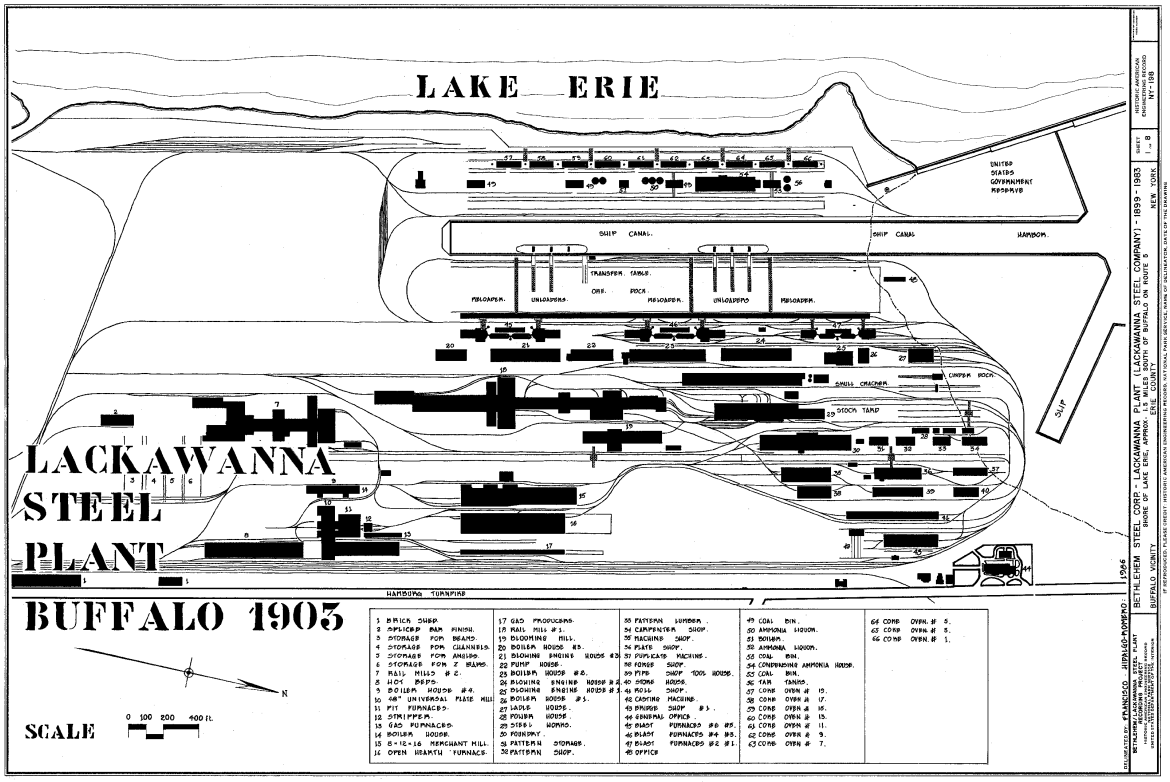

Figure 4: Lackawana Steel Plant

a set of orders related to the multiple items to be manufactured, and the sequences of operation/station to be used.

2.6 Orders, Production Plan and Production Orders

In the production planning module, customer orders are considered, with each order potentially including multiple items. A product can have variants in terms of sizes and colours, meaning that each variant corresponds to a distinct item. However, it is important that each item/variant can be easily associated with the main product, which essentially aggregates all the variants.

Internally, an order has a number, the customer's identification, a list of products, and the expected delivery date and location. To simplify, all items are assumed to be delivered on the same date and location. Customers are identified by their tax identification number (NIF). The customers' names, addresses, and contact details are also stored. Customers can be categorized into two types: individual or company.

An order can lead to one or more production orders (there may be more than one production order for the same item, but different items must have distinct production orders). When generating production orders, it is essential to consider the BOM (Bill of Materials) defined in Product Engineering, as this structure outlines the components/raw materials needed for producing the item. On the other hand, the production order must include the list of operations necessary for completing the respective item (according to the BOO), and potentially also the stations where those operations will be carried out.

Figure 5: Simple Bench Assembling

2.7 Case Study

In this section, the production of a simple bench made of solid wood is detailed focusing on its Bill of Operations and Bill of Materials.

Consider the production of a solid wood bench, polished and varnished. This bench is made up of a seat top and four legs; the legs are attached by using a threaded rod inserted into the leg and secured into a female fitting (nut) attached to the seat top (as represented in 5).

The Bill of Operations for this simple bench is shown in figure 6. In this diagram, the operations are represented by rectangles, the components by ellipses, and the raw materials by diamonds.

Note that from the Bill of Operations, it is possible to derive the Bill of Materials as shown in figure 7. In this diagram, the components are represented by ellipses and the raw materials by diamonds.

It should be noted that each item/product has a specific BOO and BOM, even if they belong to the same product family. Thus, considering a family like 'Simple Bench,' we could consider a variant, such as a bench with a similar structure but painted instead of varnished. This would mean a new BOO that includes a painting operation (instead of varnishing) and a new BOM that includes paint (instead of varnish). In this context, a BOO template is a common structure that can be used to enhance the user experience, for example, by facilitating the creation of variations within a given family.

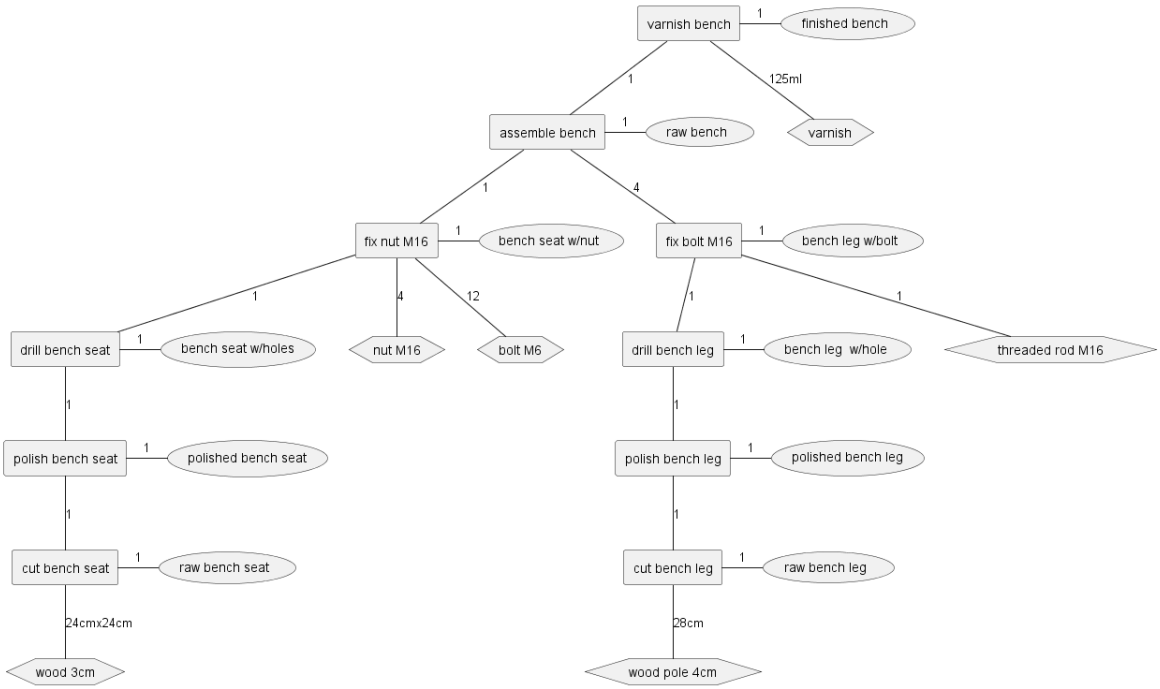

Figure 6: Simple Bench - Bill of Operations

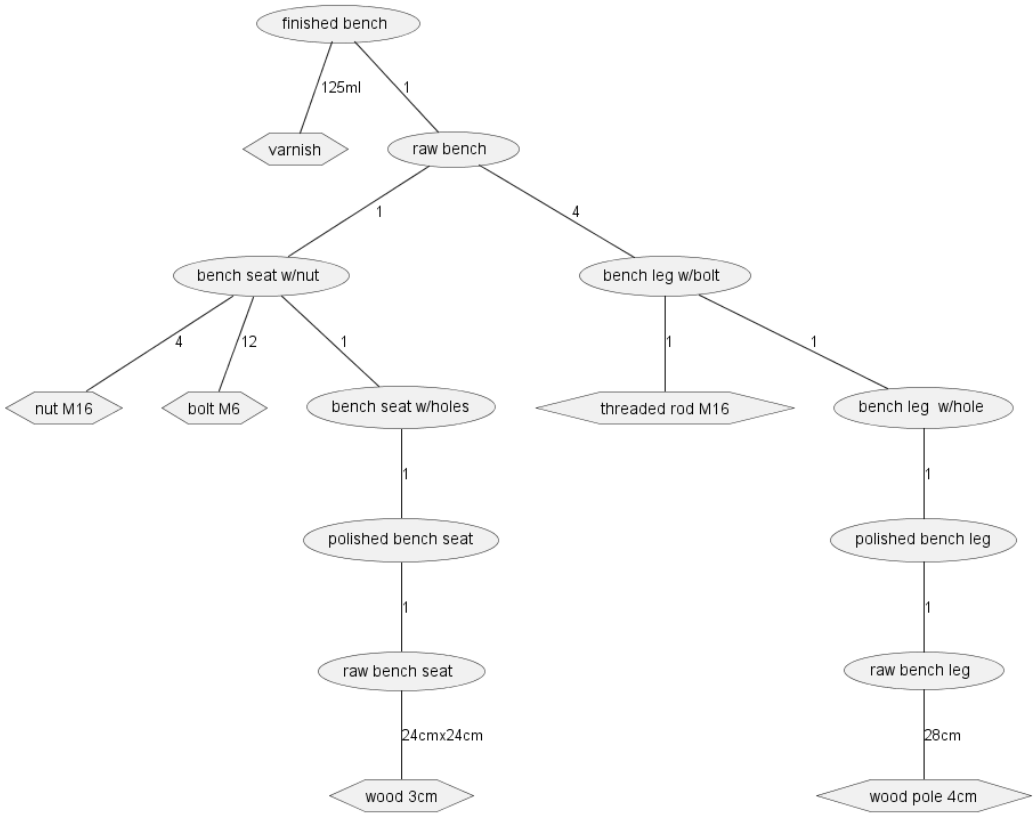

Figure 7: Simple Bench - Bill of Materials

2.8 System Users

Below, the most common user profiles and respective functions are introduced:

- Production Manager: responsible for maintaining information related to products and used raw materials, and for controlling and managing information associated with production orders;
- Plant Floor Manager: responsible for specifying the factory production lines and their respective machines;
- Administrator: responsible for managing (e.g., creating, deactivating) the various system users and their respective permissions, as well as for the overall system configuration.

3 Minimum Viable Product (MVP)

The project to be developed must consider the architecture presented in the diagram of the components in figure 8.

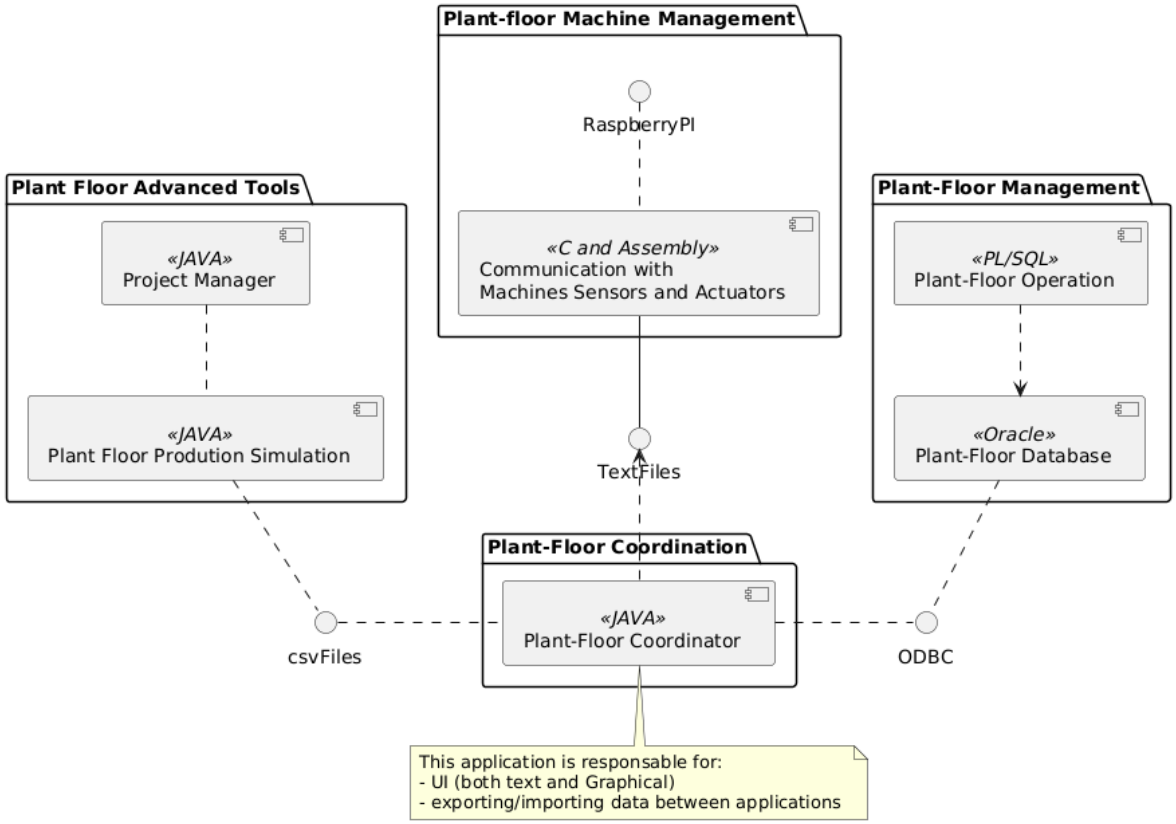

Figure 8: Architecture suggested for the solution to be developed

This project aims to develop a Minimum Viable Product, iteratively and incrementally, hence, the assignment is divided into three Sprints:

- Sprint 1 – weeks 3 to 6 – from 1/October to 27/October
- Sprint 2 – weeks 7 to 10 – from 28/October to 24/November
- Sprint 3 – weeks 11 to 14 – from 25/November to 5/January

A description of the MVP is provided for each sprint. The teams must follow the provided User Stories (US) and consider their sequencing and respective dependencies. By the end of each sprint, each team should be able to meet the specified requirements. The teams should be able to add the US to the Backlog, estimate them appropriately, and distribute them among team members. For easier reading, the US are separated by Course Unit (Subject).

3.1 Computer Architecture (ARQCP)

The logical architecture of the Plant-floor Machine Management system is presented in Fig. 9:

Figure 9: Logical architecture.

3.1.1 UI - User Interface Component

The purpose of the UI component is to be the user interface of the Plant-floor Machine Management system. This is a console-based interface.

3.1.2 Machine Component

The Raspberry PI Pico will be used to emulate a production machine. It is equipped with five LEDs plus the built-in LED and temperature and humidity sensor.

The purpose of the built-in LED is to show the state of the machine:

- *Blinking* means that the machine is operating a production operation;
- *On* means that the machine is ready to operate;
- *Off* means that the machine is not operational.

The purpose of the five LEDs is to present the number (using binary representation) of the production operation being executed by the machine. The sensor will be used to get the temperature and humidity of the production machine at a given time instant.

Regarding the software, the device must wait for commands, it sets the LEDs according to the received command and returns the temperature and humidity values (unless the machine is not operational).

Thus, this device will be used to emulate several machines; whenever it receives a command, it should turn off all the LEDs after two seconds.

3.1.3 MachManager Component

The purpose of this component is to manage the set of machines that compose the plant floor. It forwards the commands from the UI to Machine components and manages the plant floor data. For instance, it should have information about what each machine is operating at a given time instant.

3.1.4 Sprint 2

This Sprint's purpose is to code a set of functions using **assembly**. These functions will be integrated into the software to be developed in Sprint 3.

- USAC01 - Implement the `int extract_data(char* str, char* token, char* unit, int* value)` function.

  It receives as input two strings, `str` and `token`, and for output, it receives two pointers, one for a string, `unit` and another for an integer, `value`. The content of the `str` is formatted as follows:

  - *TOKEN*`&unit:`*xxxxxxx*`&value:`*xx*`#`*TOKEN*`&unit:xxxxxxxx&value:xx`, where *TOKEN* could be `TEMP` or `HUM` for temperature or humidity, respectively.

  This function extracts value and unit data from `str` according to the `token`. It should return `1` if it succeeds, `0` otherwise (in this case, it should set `value` to zero and `unit` to empty string, `""`).

```
char str [] ="TEMP&unit:celsius&value:20#HUM&unit:percentage&value:80";
char token [] ="TEMP";
char unit[20];
int value;
int res =  extract_data(str,token, unit, &value);
printf("%d:%s,%d\n", res, unit, value); //1: celsius,20

char token2 [] ="AAA";
res =  extract_data(str,token2, unit, &value);
printf("%d:%s,%d\n", res, unit, value); //0: ,0
```

- USAC02 - Implement the `int get_number_binary(int n, char* bits)` function.

  It receives as input a number in the range[0, 31] and for output a string, `bits` This function should represent the received number (that is in base 10) in base 2, using a byte for each binary digit. It should return `1` if it succeeds, `0` otherwise.

```
    int value = 26;//0b11010
    char bits[5];
    int res = get_number_binary(value, bits);
    printf("%d: %d, %d, %d, %d, %d\n",res, bits[4], bits[3],bits[2],bits[1],bits
        [0]); //1: 1,1,0,1,0
```

- USAC03 - Implement the `int get_number(char* str, int * n)` function. It receives as input a string `bits` and for output an integer pointer, `n`. This function should convert the string into a number. It should return `1` if it succeeds, `0` otherwise.

```
int value;
char str[] = "  89  ";
int res = get_number(str, &value);
printf("%d: %d\n",res, value); //1: 89
char str2[] = "8--9";
res = get_number(str, &value);
printf("%d: %d\n",res, value); //0:
```

- USAC04 - Implement the `int format_command(char* op, int n, char *cmd)` function.

  It receives as input a string, `op`, and an integer, `n`, and for output it gets a string, `cmd`. This function should format the output string as follows:

  - *`CMD`*`,x,x,x,x,x`, where *`CMD`* could be `OP`, `ON`, or `OFF`, and the five `x` (separated by comma) are the representation of the `n` in base 2. Moreover, the *`CMD`* should be trimmed and capitalized.

  It should return `1` if it succeeds, `0` otherwise.

```
int value = 26;
char str[] = "  oN  ";
char cmd[20];
int res = format_command(str, value, cmd);
printf("%d: %s\n",res, cmd); //1: ON,1,1,0,1,0
char str2[] = "aaa";
int res = format_command(str, value, cmd);
printf("%d: %s\n",res, cmd); //0:
```

A circular buffer is an array of constant length, and it is used to store data in a continuous loop. It is also known as a ring buffer because it stores the data circularly. The circular buffer has two pointers, one for the head of the buffer and another for the tail. The head pointer points to the location where we will insert the next element, while the tail pointer points to the location of the oldest element in the buffer. The length of the array is also required.

- USAC05 - Implement the `int enqueue_value(int* buffer, int length, int* tail, int* head, int value)` function that insert the `value` into buffer. However, if the buffer is full, the oldest element should be removed to insert the new one. It should return `1` if the buffer is full after insertion, `0` otherwise.

- USAC06 - Implement the `int dequeue_value(int* buffer, int length, int* tail, int* head, int *value)` function that removes the oldest element from the buffer and outputs it in the `value` integer pointer. It should return `1` if it succeeds, `0` otherwise.

- USAC07 - Implement the `int get_n_element(int* buffer, int length, int* tail, int* head)` function that returns the number of elements stored in the buffer.

- USAC08 - Implement the `int move_n_to_array(int* buffer, int length, int *tail, int *head, int n, int* array)` function that remove the oldest `n` elements from buffer to `array`. It should return `1` if it succeeds, `0` otherwise (if there are no `n` elements to move).
- USAC09 - Implement the `int sort_array(int* vec, int length, char order)` function that sorts the array taking into account the `order` parameter, `1` one for ascending, `0` for descending. It should return `1` if it succeeds, `0` otherwise (if the `length` is less or equal to zero).
- USAC10 - Implement the `int median(int* vec, int length, int *me)` function that sorts the array and outputs the median value in the `me` parameter. It should return `1` if it succeeds, `0` otherwise (if `length` is less or equal to zero).

4 Sprint 3

According to the architecture defined for the Plant-floor Machine Management System (Fig. 9), the implementation of the three components is required: `Machine`, `MachManager`, and `UI`.

4.1 Machine

As described in Subsection 3.1.2, the purpose of this component is to emulate a plant-floor machine. The `Machine` device's generic algorithm is as follows:

```
while(1){
    cmd = wait_for_command_from_mach_manager() //cmd = "ON,1,1,0,1,0"
    temp = read_temp_from_sensor()
    hum = read_hum_from_sensor()
    str ="TEMP&unit:celsius&value:temp#HUM&unit:percentage&value:hum";
    send_data(str)
    turn_on_leds(cmd)
    sleep(2 secs)
    turn_off_leds()
}
```

The programming language used for implementation must be C.

4.2 MachManager

As described in Subsection 3.1.3, the purpose of this component is to manage the set of machines that compose the plant floor and also bridge the interactions between `UI` and the `Machine components`.

Each machine holds a numerical identifier. All operations performed by a machine must be saved in a container, where each entry holds the following information:

- State, could be "OP", "ON" or "OFF" (meaning, in operation, available to be used, and inoperational due to malfunction or maintenance procedure);

- Operation designation, for instance, "cutting"
- Operation number, between 0 and 31;
- Timestamp of the beginning of the operation;
- Temperature and humidity.

Each machine has an alert mechanism that triggers a message when the temperature and/or humidity values are outside the recommended levels (minimum or maximum). Sensor data is error-prone, so to deal with such errors the Moving Median[1] filter technique should be used. Such a technique collects the last `n` values, sorts them, and picks up the median. Alerts should be triggered when the median value is outside the respective levels.

In the implementation of this component:

- All arrays must be dynamically allocated and freed when they are no longer in use.
- It is mandatory to invoke/use the functions coded in Sprint 2 (if any is missing you must implement it in assembly).

The `MachManager` component generic algorithm is as follows:

```
while(1){
    inst = wait_for_instructions_from_ui()
    update_internal_data(inst)
    cmd = get_cmd_from internal_data()
    send_cmd_to_machine(cmd);
    str = wait_for_data_from_machine()
    extract_data(str, data)
    update_internal_data(data)
    check_for_alerts()
}
```

4.3 `UI` - User Interface

As described in Subsection 3.1.1, the purpose of this component is to be the user interface, a console-based interface. This user interface should be robust, which means it should be able to handle malicious/error inputs (*e.g.,* entering a character when a number was supposed to be inserted).

4.4 User Stories

- USAC11 - Implement and assembly the `Machine` device.
- USAC12 - As a Product Owner, I want to perform the data structure initial setup for each machine using a text file.

[1] https://reference.wolfram.com/language/ref/MovingMedian.html

  - For setting up the machine, the following information is required:
    * An identifier
    * A name (or designation)
    * Temperature - minimal and maximum level
    * Humidity - minimal and maximum level
    * Circular Buffer length
    * Moving median window length
  - Each line in the text file should have the required information to set up a machine.

- USAC13 - As a Product Owner, I want to create a text file (.csv file) with the sequence of operations performed by a specific machine.
- USAC14 - As a Product Owner, I want to manage the plant-floor machinery.
  - Add and remove a machine from the plant floor.
    * The machines can only be removed if they are not operating.
  - Read the current status of a machine on the plant floor.
- USAC15 - As a Product Owner, I want to assign an operation to a machine on the plant floor.
- USAC16 - As a Product Owner, whenever I manage a specific machine, I want to show its state in the `Machine` device for 2 seconds.
- USAC17 - As a Product Owner, I want to feed the system with a list of instructions.
  - This list must be implemented through a text file.
- USAC18 - As a Product Owner, I want to be notified if the machine temperature and/or humidity values are outside the levels.

4.5 Databases (BDDAD)

In this component, a Database should be designed to model an industrial facility that would support the following User Stories (USs).

4.5.1 Sprint 1

- USBD01 - As a Product Owner, I want a data dictionary/glossary to be created.
- USBD02 - As a Product Owner, I want the relational model to be created (logical level).

  Acceptance criteria:

  - The data model should cover the industrial facility's activity described in the document, as well as any additional requirements resulting from the provided user stories.
  - Minimum expected requirement: a complete model in Visual Paradigm, including all connections and constraints, which allows the physical model script to be automatically generated.
  - Minimum requirement above the expected: the presentation of a conceptual model developed in Visual Paradigm in addition to the expected level.

- USBD03 - As a Product Owner, I want the relational model to be instantiated (physical level).
  - It will be shown on Oracle LiveSQL.
  - Minimum expected requirement: automatic generation from Visual Paradigm (centralized change management).

- USBD04 - As a Product Owner, I want to import data from a legacy system and deliver it on a spreadsheet.
  - Minimum expected requirement: manual creation of the data input scripts.
  - Minimum requirement above the expected: automatic generation of SQL input code from the spreadsheet (e.g., Excel formulas, scripts in any other language, etc.).

- USBD05 - As a Production Manager, I want to know, for each product, the orders to be delivered (customer, product, quantity, date) within a given time frame.

- USBD06 - As a Production Manager, I want to know the types of workstations that can be used in a given order.

- USBD07 - As a Production Manager, I want to know the materials/components to be ordered to fulfil a given production order, including the quantity of each material/component.

- USBD08 - As a Plant Floor Manager, I want to know the different operations the factory supports.

- USBD09 - As a Plant Floor Manager, I want to get the operations sequence as well as get the respective type of workstation, from a BOO of a given product.

  Acceptance Criteria for USBD05 to BD09:
  - Minimum acceptance criteria: only the User Stories with data allowing their proper functioning will be evaluated.
  - Minimum expected requirement: demonstrated with data imported from the legacy system.
  - Minimum requirement above the expected: demonstrated with data provided for Sprint 1 evaluation.

4.5.2 Sprint 2

- USBD10 As a Product Owner, I want an updated relational data model of the system / logical and physical level).
- USBD11 As a Product Owner, I want to insert the provided data into the system.
- USBD12 As a Production Manager, I want to get the list of parts used in a product.

  Acceptance criteria: A function should return a cursor with all the product parts and their quantity. The individual components should be included when a part is a subproduct made at the factory.
- USBD13 As a Production Manager, I want to get a list of operations involved in the production of a product, as well as each workstation type.

  Acceptance criteria: A function should return a cursor with all product operations. When a part is a subproduct made at the factory, its list of operations should be included. For each operation, the inputs and outputs should be included.
- USBD14 As a Plant Manager, I want to know which product uses all types of machines available in the factory.
- USBD15 As a Plant Floor Manager, I want to register a workstation in the system.

  Acceptance criteria: A function should be used to create the workstation and to return success or an error.
- USBD16 As a Production Manager, I want to register a product in the system. **Acceptance criteria**: A function should be used to create the product and to return success or an error.
- USBD17 As a Production Manager, I want to register an order in the system. Acceptance criteria: A function should be used to create the order, and to return the Order ID or an error. An order must be from an active customer and a product in the current line-up.
- USBD18 As a Production Manager, I want to deactivate a customer from the system. Acceptance criteria: A function should be used to deactivate the customer and return success or an error. A customer with orders that have not yet been delivered/fulfilled cannot be deactivated.
- USBD19 As a Production Manager, I want to know which product has more operations in its BOO.

  USBD10 to UBD14 are mandatory for all teams. USB15 to USB19 are individual. Teams of 4 or less members do not work on USBD19.

4.5.3 Sprint 3

Each component and raw material has a minimum stock and can have multiple suppliers. Each supplier has a minimum order size for each material/component and a unit cost. The cost and minimum order size can change with time.

An operation has an expected execution time. For each operation, a workstation type has a maximum execution time and a setup time that applies to the whole batch (the setup time is the period required to set up the machine to produce a batch).

- USBD20 As a Product Owner, I want an updated relational data model at the logical level. All constraints must be in the model.
- USBD21 As a Product Owner, I want to automatically generate the physical level SQL code of the database from the Visual Paradigm model.
- USBD22 As a Product Owner, I want the developed PL/SQL code to be thoroughly tested. The Product Owner will supply some testing data. Other may have to be created by the team.
- USBD23 As a Production Manager, I want to make sure that the expected execution time of an operation is not greater than the maximum execution time of every workstation type where it may be run. A trigger should be developed to avoid this issue in both insert and update operations.
- USBD24 As a Production Manager, I don't want it to be possible to add a product as an input in its own BOO. A trigger should be developed to avoid such circular references. This doesn't apply to the BOOs of subproducts.
- USBD25 As a Production Manager, I want to get a list of a product's operations necessary for the production plan module.

  Acceptance criteria: A function should return a cursor with all product operations. When a part is a subproduct made at the factory, its list of operations should be included. For each operation, the execution time, inputs and outputs should be included.
- USBD26 As a Production Manager, I want to know if we have the necessary materials and components in stock to fulfil a given order.
- USBD27 As a Production Manager, I want to reserve the necessary materials and components to fulfil a given order. The reserved materials and components should be registered in the database, but not automatically deducted from stock. The reservation should be created only if the whole order can be fulfilled.
- USBD28 As a Production Manager, I want to have a list of all reserved materials and components, their quantity and the ID of the supplier.
- USBD29 As a Factory Manager, I want to know the workstation types not used in any BOO.

- USBD30 As a Factory Manager, I want to use/consume a material/component, i.e. to deduct a given amount from the stock. The operation should not be allowed if the remaining stock falls below the currently reserved quantity.

Acceptance criteria for USBD26-USBD30: A PL/SQL function must be developed. It may use other functions and/or procedures.

USBD20 to UBD24 are mandatory for all teams. USB26 to USB30 are individual. Teams of 4 or less members do not work on USBD30.

USBD25 is only to be done by groups involved in LAPR3. It replaces USBD29.

4.6 Information Structures (ESINF)

In the Product Engineering process within Production Management, planning and evaluating production capacity is of great importance. Simulation tools generate statistical information that allows for the quick identification of relevant concepts such as machine/station utilization, time spent on operations, bottlenecks in the production flow, and underutilized resources. Consider a factory where simple items that do not require assembly are produced; that is, simple items like chair legs and tabletops, which will be subsequently joined together (e.g. on a plant floor) or sold in that format (e.g. marketing model IKEA-type). In its production, each item is subject to a set of sequential operations (e.g., cutting, drilling, polishing, painting, labelling, inspection). These operations are carried out by machines/stations installed in the factory. There may be different machines performing the same operation, possibly with different execution times. At this stage, for simplification, a given machine only performs one operation. These stations can be robots, automated machines, or operated by humans.

4.6.1 Sprint 1

Consider the development of a basic simulation tool that consumes the following files:

```
- articles.csv with the format:
<id_item, priority, name_oper1, name_oper2, ...,  name_operN>

- workstations.csv with the format:
<id_workstation, name_oper, time>
```

- USEI01 - Define the adequate data structures to store the information imported from the files.
- USEI02 - Implement a simulator that processes all the items according to the following criteria:
  - AC1: The simulator should create a preliminary queue for each operation, containing all the items, whose next operation (according to the sequential production process) is that of the specified queue.
  - AC2: The items in the queue should be assigned based on the processing availability to the available machine capable of performing the required operation faster, in the order of their entry into the queue.

- USEI03 - Calculate the total production time for the items.
- USEI04 - Calculate execution times by each operation.
- USEI05 - Present a list of workstations with total time of operation, and percentages relative to the operation time and total execution time, sorted in ascending order of the percentage of execution time relative to the total time.
- USEI06 - Present average execution times per operation and the corresponding waiting times.
- USEI07 - Produce a listing representing the flow dependency between workstations. The listing should be sorted in descending order of processed items. Consider the following example: the items a, b, c, d and e were processed in the workstation m1, m2, m3, m4, m5:

  ```
  a: m1 -> m5
  b: m1 -> m2 -> m4 -> m5
  c: m1 -> m2 -> m3 -> m5
  d: m1 -> m4 -> m3
  e: m1 -> m3 -> m5
  ```

  After the complete processing of these items, the following listing should be produced:

  ```
  m1 : [(m2,2),(m5,1),(m3,1),(m4,1)]
  m2 : [(m4,1),(m3,1)]
  m3 : [(m5,2)]
  m4 : [(m5,1),(m3,1)]
  ```

- USEI02B - Consider an improvement to the simulator developed in USEI02 that takes into account a processing order based on priority, according to the following criteria:
  - AC1 - The items in the queue should be assigned, according to their priority (high, normal, low) to the available machine that can perform the required operation in the shortest time.
  - AC2 - Statistical measures should be produced similarly for this simulator variant.

4.6.2 Sprint 2

Consider these files as input data for the following US:

```
- boo.csv with the format:
<op_id,item_id, item_qtd,
   (op_id1, op_qtd1, ..., op_idN, op_qtdN),
   (item_id1, item_qtd1, ..., item_idN, item_qtdN) >
```

```
- items.csv with the format:
 <id_item, item_name>

- operations.csv with the format:
 <op_id, op_name>
```

- USEI08 - Build the complete production tree of a product As a user, I want to import data from a CSV file containing the operations, components, materials and quantities, for a product, to create a production tree. In this production tree, each node represents either an operation or a material associated with a specific stage in the production process. Notice that each subcomponent has its own tree and a respective entry in the boo.csv file, instead of the materials that are always leafs in the production tree. The figure 6 should be used as a reference.

  For a more complex product like a bicycle, where each part (*e.g.,* the frame, wheels, brakes, chain, etc.) has its own assembly or manufacturing process, you could have multiple sub-trees within the main production tree.

- USEI09 - Efficiently search for specific operations or materials

  As a user, I want to be able to search for any specific operation or material in the production tree to view its details quickly.

  - AC1 - Implement a map to associate each operation and material with a unique identifier.
  - AC2 - The map should allow for efficient retrieval of nodes by name or ID.
  - AC3 - The search function should return the node type (Operation or Material), quantity (for materials), and parent operation if applicable.

- USEI10 - Tracking material quantities (Binary Search Tree)

  As a user, I want to track and organise materials by quantity to see all materials used in the production, sorted by their quantities, so I can easily find the most or least used materials. Having multiple materials with the same quantity, associate that quantity with a list of materials considering that quantity.

  - AC1 - Materials should be stored in a BST sorted by quantity.
  - AC2 - The user should be able to display the materials in increasing or decreasing order of quantity.

- USEI11 - Prioritize Quality Checks (Heap Priority Queue)

  As a user, I want to prioritize quality checks based on their position in the production process so that essential checks are completed first. For example, quality checks closer to the final product should have higher priority.

  - AC1 - Use a priority queue to manage quality check operations, where each quality check node has a priority level based on its production stage.

  - AC2 - The user should be able to view quality checks in order of priority and perform them one at a time from highest to lowest priority.

- USEI12 - Update material quantities in the production tree

  As a user, I want to adjust material quantities in the production tree if changes are required and see these updates reflected in real time.

  - AC1 - Allow users to specify a new quantity for a material, which should then update the respective node in the production tree.
  - AC2 - The system should handle any cascading updates or dependencies efficiently.

- USEI13 - Calculate the total quantity of materials and time needed for the production.

  As a user, I want to calculate the quantity of materials in the production tree, using the quantity of each material to understand the overall production expense.

  - AC1 - Use a recursive traversal method to sum up the quantity of all materials and time needed to perform the operations in the production tree.
  - AC2 - The final output should display the totals per material and per operation.

- USEI14 - Identify and prioritize critical path operations (Heap and Binary Tree)

  As a user, I want to identify and prioritize operations that have the longest sequence or are the most essential so that I can allocate resources efficiently.

  - AC1 - Use a heap-based priority queue to store operations by their dependency levels (depth in the production tree).
  - AC2 - Display critical path operations in order of their importance based on depth.
  - AC3 - The user should be able to view the critical path in sequence.

- USEI15 - Put the components into production in the simulator (AVL Tree)

  As a user, I want to extract the BOO tree from the production tree using a hierarchy identifier strategy and, with an adequate traversal of this binary tree, put the components into production in the simulator.

  - AC1 - Use a BST/AVL to store the operations by their dependency levels.

4.6.3 Sprint 3

The Program Evaluation and Review Technique (PERT) and the Critical Path Method (CPM) are essential tools for managing and scheduling complex projects. These techniques help project managers identify the critical tasks that directly affect the project's timeline and provide insights into scheduling flexibility for other tasks.

In PERT/CPM, project activities and their dependencies are represented as a directed graph, where:

- Nodes represent individual activities, associated with details such as duration, description, and fixed and variable costs, dependent on the resources and time involved in its implementation.
- Edges represent dependencies between activities, defining the order in which they must be executed.

For small project networks, finding all the paths and determining the longest path(s) is a possible method to identify the critical path(s). However, this is not an efficient procedure for larger projects.

PERT/CPM uses a considerably more efficient procedure based on forward and backward pass calculations rather than finding all paths in the graph. The forward and backward pass method used in PERT/CPM is a systematic, efficient, and informative mode to manage project schedules. By avoiding the need to enumerate all paths, that procedure allows for practical application in large-scale projects while yielding detailed scheduling insights critical for decision-making.

- USEI17 - Build a PERT-CPM graph - As a user, I want to import data from a CSV file structured as follows:

  ```
  - activities.csv with the format:
  <act_id,act_descr,duration,duration_unit,cost,cost_unit,
  prev_act_id1,...,prev_act_idN>
  ```

  And build one PERT/CPM graph, considering the necessary validations for activity dependencies and data integrity.

- USEI18 - Detect Circular Dependencies - As a user, I want to validate the project graph to ensure there are no circular dependencies among activities, as this would make the project unable to be scheduled.
- USEI19 - Topological sort of project activities - As a user, I want to perform a topological sort of the project's activities. This will provide a feasible sequence for executing the activities, respecting all dependency constraints, and help validate the graph's structure.
- USEI20 - Calculate Earliest and Latest Start and Finish Times - As a user, I want to calculate the earliest start (ES) and the latest finish (LF) times for each activity, based on dependencies. This will help identify slack for non-critical activities and ensure accurate scheduling.
- USEI21 - Export Project Schedule to CSV - As a user, I want to export the entire project schedule, including start times, finish times, and slack for each activity, to a CSV file for record-keeping and further analysis.

  ```
  - schedule.csv with the format:
  <act_id,cost,duration,es,ls,ef,lf,prev_act_id1,...,prev_act_idN>
  ```

- USEI22 - Identify the Critical Path - As a user, I want to analyse the PERT-CPM graph to identify the critical path(s) and calculate the total project duration. This should account for dependencies and ensure that all activities on the critical path are presented with their key metrics: earliest and latest (start/finish) times.
- USEI23 - Identify Bottlenecks Activities in the Project Graph - As a user, I want to identify bottleneck activities in the project graph, which are activities with the highest number of dependent activities or that appear on most paths, to better understand where delays are most likely to propagate.
- USEI24 - Simulate Project Delays and their Impact - As a user, I want to simulate delays in specific activities by increasing their durations, and I want to automatically recalculate the critical path, total project duration, and slack times to assess the potential impact of these delays on the overall project schedule.

4.7 Applied Physics (FSIAP)

This project's component goal is to create a scaled-down plan for a ventilation and exhaust system, using the provided electronic hardware components as the main resources.

4.7.1 Sprint 2

In this project phase, you should: begin assembling the hardware with the provided sensors to simulate the operation of the final system; set up the equipment to communicate with the sensors, and obtain some readings from them. At this stage, LEDs will represent the ventilation and exhaust elements.

- USFA01 - Assemble and program the board to read the provided sensors.
  - Use the Arduino IDE interface and load the libraries to read the respective sensors.
  - Read some temperature values, and touch on the temperature sensor to check if it varies correctly.
  - Place a LED to signal the operation of the temperature sensor; it should only light up when the temperature is 5 º higher than the initial reading.
  - Add a second LED to indicate the operation of the humidity sensor, which should only light up when the humidity value is more than 5% above the initial reading.
  -
- USFA02 - Monitor the space in which you are located for at least 15 minutes, recording the temperature and humidity values.
  - The temperature and humidity values records should be saved in an easily accessed file and indicate when the 'flags' that triggered the LEDs occurred (.txt).

4.7.2 Sprint 3

Using a prototype box with two small built-in fans, designed exclusively for classroom use, you must demonstrate the operation of a ventilation and exhaust system.

- USFA03 - Add another sensor to the previously created system, in this case, a gas sensor. The use of this sensor will be limited as it can only occur during the respective class period. Check the characteristics to ensure the board communicates with the sensor. The fans' operation should follow this sequence:
  - When the temperature sensor registers a 5^{o} value above the initial reading, it must:
    1. Turn on the exhaust fan first for 5 seconds;
    2. Then turn on the ventilation fan for 5 seconds.
  - When the humidity registers a value 5% above the initial reading, it should:
    1. First turn on the ventilation fan for 10 seconds;
    2. Afterwards, turn on the exhaust fan for 10 seconds.
  - When the gas sensor registers a value 2% above the initial reading value, it should:
    1. Turn on both fans simultaneously for 10 seconds;

    Sprint's added value item:
    1. Consider activating the fans for sudden sensor changes exceeding 30% per minute.

4.8 Laboratory-Project 3 (LAPR3)

4.8.1 Sprint 1

- USLP01 - As a Product Owner, I want the domain model to be created (conceptual level). This model will be an essential communication element between all the stakeholders. The domain model diagram is a "dynamic" document, which should continuously reflect the stakeholders' shared understanding of the domain.

  Acceptance Criteria:
  - The data model should cover the activity of the industrial facility, excluding the simulation tool.
  - What is expected: a "basic" model shared in Visual Paradigm or the documentation in GitHub (e.g., PlantUML).
- USLP02 - As a Product Owner, I intend to present the simulation tools' functionality and results through a user-friendly text-based interface.

4.8.2 Sprint 2

- USLP03 - As a Product Owner, I want to graphically visualize the Product Structure (BOM) for a given article (use figure 7 as reference).
- USLP04 - As a Product Owner, I want to graphically visualize the Operation Structure (BOO) for a given article (use figure 6 as reference).
- USLP05 - As a Product Owner, I want to import from Oracle SGBD (using ODBC) the information needed (*e.g.,* BOO and BOM) from a given article and generate the .csv files required to fulfil from USEI08 to USEI15.

4.8.3 Sprint 3

- USLP06 - As a Product Owner, I want to read the file `orders.csv` and with previously developed features, namely the ones resulting from USEI15 and USEI12, simulate the production of the items required to accomplish the orders. The workstation operation times should be imported from Oracle SGBD (using ODBC).

  ```
  - orders.csv with the format:
  <order_id,item_id,priority,qtd>
  ```

- USLP07 - As a Product Owner, I want to update/define the average production times, in the Oracle DB (using ODBC). This average is obtained with the use of the simulator and it should consider the waiting times.
- USLP08 - As a Product Owner, I want to export the production planner simulator results to a text file, able to be processed by USAC17.

4.9 Non-functional Requirements

This section describes some of the non-functional requirements that must be considered in the implementation of the project.

- The validation of business rules that must be followed during data registration and updating.
- The database will be the main repository of information for the system and must reflect the necessary data integrity. The information should be persisted in a remote DBMS (Database Management System).
- To enhance interoperability between existing or developing systems, the main application will be developed in Java. However, some components will need to be developed in other languages; the creation and management of the database will use PL/SQL and the interaction with stations will be developed in C/Assembly. A significant part of the integration will be carried out through files.
- The class structure should be designed to allow for easy maintenance and the addition of new functionalities, in line with best practices in object-oriented programming (OOP).

- During the system development, the team must: (i) adopt best practices for identifying requirements, and for OO software analysis and design; (ii) adopt recognized coding standards (e.g., CamelCase); (iii) use Javadoc to generate useful documentation for Java code.
- All the images/figures produced during the software development process should be recorded in SVG format.