

NL2ProcessOps evaluation

Welcome to the questionnaire for the human evaluation of NL2ProcessOps, a novel approach leveraging Large Language Models (LLMs) and concepts such as Retrieval Augmented Generation, agents, and tools for code generation to streamline process deployment operations.

The proposed approach is designed to work with textual process descriptions and focuses on the different operations of process deployment, from extracting the control flow in terms of a process model, over retrieving required tools associated with each task, to generating executable code for manual refinement purposes and deployment.

A core assumption of the proposed approach is that the Python code contains invocation to external tools that simulate the tasks of a given process. The tools are available and only those related to the process at hand are retrieved.

The user will find a textual process description, a Python code and specification of the tools imported by the code. Each tool specify the input and output parameters and the module containing the relative function.

The user has to evaluate three criteria for each code - from 1 (very bad) to 5 (very good).

The criteria are:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

1. Are you an expert in BPM and/or Python development? *

BPM

Python

BPM and Python

None of them

Other _____

2. What is your job? *

- MSc student
- PhD student
- Researcher
- Professor
- Other _____

Part 1.1

Process description:

When a pallet arrives at the working station, the system empties the scan results. Then the worker scans the order. Afterwards the system displays the scanning UI to the worker and in parallel, the worker assembles the part.

Code:

```
from tools.working_station_is import EmptyScan
from tools.working_station_is import ScanOrder
from tools.working_station_is import DisplaysScanningUI
from tools.worker import AssembleParts
import threading

# Assuming the tools are already imported and available for use as
described
# EmptyScan, ScanOrder, DisplaysScanningUI, AssembleParts

def display_scanning_ui_and_assemble_parts(order_id):
    # Display scanning UI
    DisplaysScanningUI.call(order_id=order_id)
    # Assemble parts in parallel
    AssembleParts.call()

def process_pallet_arrival():
    # Empty scan results
    EmptyScan.call()

    # Scan order
    order_id = ScanOrder.call()

    # Create threads for parallel execution of displaying scanning UI and
    # assembling parts
    thread_ui = threading.Thread(target=DisplaysScanningUI.call, args=
(order_id,))
    thread_assemble = threading.Thread(target=AssembleParts.call)

    # Start threads
    thread_ui.start()
    thread_assemble.start()

    # Wait for both threads to complete
    thread_ui.join()
    thread_assemble.join()

    return "Process completed."

if __name__ == "__main__":
    result = process_pallet_arrival()
    print(result)
```

```

Tools:
EmptyScan = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "working_station_is"
}
ScanOrder = {
    "input_parameters": [],
    "output_parameters": ["order_id:int"],
    "module": "working_station_is"
}
DisplaysScanningUI = {
    "input_parameters": ["order_id:int"],
    "output_parameters": [],
    "module": "working_station_is"
}
AssembleParts = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "worker"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

3. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Very good

4. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

5. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Ver: Very good

Part 1.2

Process description:

When a pallet arrives at the working station, the system empties the scan results. Then the worker scans the order. Afterwards the system displays the scanning UI to the worker and in parallel, the worker assembles the part.

Code:

```
from threading import Thread
from working_station_is import PalletArrives, EmptyScan, ScanOrder,
DisplaysScanningUI
from worker import AssembleParts

def process():
    # Pallet arrives at the working station
    PalletArrives.call()

    # The system empties the scan results
    EmptyScan.call()

    # The worker scans the order
    order_id = ScanOrder.call()

    # The system displays the scanning UI to the worker
    ui_thread = Thread(target=DisplaysScanningUI.call, args=(order_id,))
    ui_thread.start()

    # In parallel, the worker assembles the part
    AssembleParts.call()

    # Wait for the UI thread to finish
    ui_thread.join()
```

Tools:

```
PalletArrives = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "pallet"
}
EmptyScan = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "working_station_is"
}
ScanOrder = {
    "input_parameters": [],
    "output_parameters": ["order_id:int"],
    "module": "working_station_is"
}
DisplaysScanningUI = {
```

```

    "input_parameters": ["order_id:int"],
    "output_parameters": [],
    "module": "working_station_is"
}
AssembleParts = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "worker"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

6. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Very Very good

7. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Very Very good

8. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 2.1

Process description:

The calibration process of a cardboard production consists of continuously capturing a photo of the cardboard being produced. Each photo is analyzed to check if all the markers identified are ok. If markers are not ok, the calibration process continues. If the markers are ok, the speed of the die cutting machine is set to 10000 RPM and the process ends.

Code:

```
from tools.camera import CaptureImage
from tools.worker import StoreCardboardRoll
from tools.vision_is import CheckMarkers
from tools.die_machine import SetSpeedDieMachine
import numpy as np

# Assuming the tools are already imported and available for use
# from tools import CaptureImage, CheckMarkers, SetSpeedDieMachine

def calibration_process():
    while True:
        # Capture photo of the cardboard
        captured_image = CaptureImage.call()

        # Analyze photo to check if all markers identified are ok
        markers_ok = CheckMarkers.call(image=captured_image)

        # If markers are not ok, the calibration process continues (loop)
        if not markers_ok:
            continue

        # If markers are ok, set the speed of the die cutting machine to
        # 10000 RPM and end the process
        speed_set = SetSpeedDieMachine.call(speed=10000)
        if speed_set:
            print("Machine speed set to 10000 RPM successfully.")
        else:
            print("Failed to set machine speed.")
        break

if __name__ == "__main__":
    calibration_process()
```

Tools:

```
CaptureImage = {
    "input_parameters": [],
    "output_parameters": ['image:np.matrix'],
    "actor": "camera"
}
StoreCardboardRoll = {
```

```

    "input_parameters": ["location:str"],
    "output_parameters": [],
    "module": "worker"
}
CheckMarkers = {
    "input_parameters": ["image:np.matrix"],
    "output_parameters": ["markers_ok:bool"],
    "module": "vision_is"
}
SetSpeedDieMachine = {
    "input_parameters": ["speed:int"],
    "output_parameters": ['speed_set:bool'],
    "module": "die_machine"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

9. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Very Very good

10. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Very Very good

11. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 2.2

Process description:

The calibration process of a cardboard production consists of continuously capturing a photo of the cardboard being produced. Each photo is analyzed to check if all the markers identified are ok. If markers are not ok, the calibration process continues. If the markers are ok, the speed of the die cutting machine is set to 10000 RPM and the process ends.

Code:

```
from vision_is import CheckMarkers
from die_machine import SetSpeedDieMachine

def calibration_process():
    while True:
        # Capture a photo of the cardboard
        image = capture_photo()  # This function needs to be implemented

        # Analyze the photo to check if all the markers are ok
        markers_ok = CheckMarkers.call(image)

        if markers_ok:
            # If the markers are ok, set the speed of the die cutting
            # machine to 10000 RPM
            SetSpeedDieMachine.call(10000)
            break
```

Tools:

```
CheckMarkers = {
    "input_parameters": ["image:np.matrix"],
    "output_parameters": ["markers_ok:boolean"],
    "module": "vision_is"
}
SetSpeedDieMachine = {
    "input_parameters": ["speed:int"],
    "output_parameters": ['speed_set:boolean'],
    "module": "die_machine"
}
```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

12. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

13. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

14. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Ver: Very good

Part 3.1

Process description:

The manufacturing process of spindles in HSD company is fully automated. When a new order for a spindle arrives at the sales department, a new process instance is initiated. The warehouse system retrieves the necessary raw materials, and in parallel, the L12 line is set up for the assembly of the ordered spindle. Once the warehouse successfully retrieves the raw materials and the L12 line is set up, the spindle is assembled over the L12 line. Subsequently, the spindle undergoes testing and running-in in the smart tester. If the outcome of the test is negative, the spindle is sent to maintenance. Then, the process ends.

Code:

```
from tools.l12 import L12AssembleSpindle
from tools.wms_is import RetrieveRawMaterials
from tools.wms_is import OrderRawMaterial
from tools.wms_is import RetrieveRawMaterial
from tools.l12 import L12SetUp
from tools.smart_tester import TestSpindle
import threading

# Assuming the tools are already imported and available for use as
described

def process(part_list, product_id):
    # Step 1: Retrieve raw materials and set up L12 line in parallel
    def retrieve_raw_materials():
        return RetrieveRawMaterials.call(part_list=part_list)

    def setup_l12_line():
        return L12SetUp.call()

    retrieve_thread = threading.Thread(target=retrieve_raw_materials)
    setup_thread = threading.Thread(target=setup_l12_line)

    retrieve_thread.start()
    setup_thread.start()

    retrieve_thread.join()
    setup_thread.join()

    # Both tasks completed, proceed to assemble spindle
    assembly_result = L12AssembleSpindle.call(part_list=part_list)

    if not assembly_result['assembled']:
        return False # Assembly failed, end process

    # Test and run-in spindle
    test_result = TestSpindle.call(product_id=product_id)

    if not test_result['passed']:
```

```

        # Test failed, send to maintenance
        # Assuming maintenance does not affect the outcome of the process
        return True # Process ends after maintenance

    return True # Process successfully completed

if __name__ == "__main__":
    part_list = ['part1', 'part2', 'part3'] # Example part list
    product_id = 123 # Example product ID
    process_completed = process(part_list, product_id)
    print(f"Process completed: {process_completed}")

```

Tools:

```

L12SetUp = {
    "input_parameters": [],
    "output_parameters": ['set_up:bool'],
    "module": "l12"
}

RetrieveRawMaterials = {
    "input_parameters": ["part_list:list"],
    "output_parameters": ["retrieved:bool"],
    "module": "wms_is"
}

RetrieveRawMaterial = {
    "input_parameters": ["part:str"],
    "output_parameters": ["retrieved:bool"],
    "module": "wms_is"
}

OrderRawMaterial = {
    "input_parameters": ["part:str"],
    "output_parameters": [],
    "module": "wms_is"
}

L12AssembleSpindle = {
    "input_parameters": ['part_list:list'],
    "output_parameters": ['assembled:bool'],
    "module": "l12"
}

TestSpindle = {
    "input_parameters": ['product_id:int'],
    "output_parameters": ['passed:bool'],
    "module": "smart_tester"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

15. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

16. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

17. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Ver: Very good

Part 3.2

Process description:

The manufacturing process of spindles in HSD company is fully automated. When a new order for a spindle arrives at the sales department, a new process instance is initiated. The warehouse system retrieves the necessary raw materials, and in parallel, the L12 line is set up for the assembly of the ordered spindle. Once the warehouse successfully retrieves the raw materials and the L12 line is set up, the spindle is assembled over the L12 line. Subsequently, the spindle undergoes testing and running-in in the smart tester. If the outcome of the test is negative, the spindle is sent to maintenance. Then, the process ends.

Code:

```
from l12 import L12SetUp, L12AssembleSpindle
from smart_tester import TestSpindle
from wms_is import RetrieveRawMaterials
from cmms_is import ProductMaintenance

def manufacture_spindle(part_list):
    # Retrieve raw materials
    retrieved = RetrieveRawMaterials.call(part_list)
    # Set up the L12 line
    set_up = L12SetUp.call()

    # Wait for both the raw materials to be retrieved and the L12 line to
    be set up
    if retrieved and set_up:
        # Assemble the spindle
        assembled = L12AssembleSpindle.call(part_list)
        if assembled:
            # Test the spindle
            passed = TestSpindle.call(assembled)
            if not passed:
                # If the test failed, send the spindle to maintenance
                ProductMaintenance.call(assembled)
        else:
            print("Failed to retrieve raw materials or set up the L12 line.")
```

Tools:

```
L12SetUp = {
    "input_parameters": [],
    "output_parameters": ['set_up:bool'],
    "module": "l12"
}
L12AssembleSpindle = {
    "input_parameters": ['part_list:list'],
    "output_parameters": ['assembled:bool'],
    "module": "l12"
}
RetrieveRawMaterials = {
```

```

    "input_parameters": ["part_list:list"],
    "output_parameters": ["retrieved:bool"],
    "module": "wms_is"
}
TestSpindle = {
    "input_parameters": ['product_id:int'],
    "output_parameters": ['passed:bool'],
    "module": "smart_tester"
}
ProductMaintenance = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "cmms_is"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

18. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

19. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

20. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 4.1

Process description:

When craving a custom 3D model, as a first step you have to instruct an artist to create a project for you. First you have to send him several sketches, and then tell him what to change until you are satisfied with the result. After that, you choose a plastic color you want to use for 3D printing. If you have the color at home (in stock), you check how much color you have left. If it's under 100 grams, you put it on your shopping list. If you do not have the color at home, you order it. When you have the plastic, you can turn the printer on and heat up the bed and the extruder (it continuously sends you back its status). While doing that, you can generate the gcode file for your printer out of the STL sent to you by the artist. After that you print the model.

Code:

```
from tools.manufacturer import SendSketches
from tools.manufacturer import ChoosePlasticColor
from tools.manufacturer import CheckColorQuantity
from tools.manufacturer import CheckColorAvailability
from tools.manufacturer import OrderColor
from tools.printer_3d import HeatUpBedExtruder
from tools.printer_3d import TurnOn
from tools.manufacturer import GenerateGCode
from tools.printer_3d import Print
def instruct_artist():
    project_description = SendSketches.call()
    return project_description

def choose_plastic_color():
    color = ChoosePlasticColor.call()
    return color

def check_and_order_color(color):
    availability = CheckColorAvailability.call(color=color)
    if not availability:
        OrderColor.call(color=color)
    else:
        quantity = CheckColorQuantity.call(color=color)
        if quantity < 100:
            print("Adding color to shopping list.")
            # Add to shopping list (Assuming a function or a process for
            adding to shopping list exists)
        # If quantity is above 100g, no action needed as per process
        description

def prepare_printer():
    TurnOn.call()
    HeatUpBedExtruder.call()

def generate_gcode(project):
```

```

gcode = GenerateGCode.call(project=project)
return gcode

def print_model(gcode):
    Print.call(GCodeFile=gcode)

def process():
    project_description = instruct_artist()
    color = choose_plastic_color()
    check_and_order_color(color)
    prepare_printer()
    gcode = generate_gcode(project_description)
    print_model(gcode)
    return "Process completed."

if __name__ == "__main__":
    result = process()
    print(result)

```

Tools:

```

SendSketches = {
    "input_parameters": [],
    "output_parameters": ["project:str"],
    "module": "manufacturer"
}
ChoosePlasticColor = {
    "input_parameters": [],
    "output_parameters": ["color:int"],
    "module": "manufacturer"
}
CheckColorQuantity = {
    "input_parameters": ["color:int"],
    "output_parameters": ["quantity:int"],
    "module": "manufacturer"
}
CheckColorAvailability = {
    "input_parameters": ["color:int"],
    "output_parameters": ["availability:bool"],
    "module": "manufacturer"
}
OrderColor = {
    "input_parameters": ["color:int"],
    "output_parameters": [],
    "module": "manufacturer"
}
TurnOn = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "printer_3d"
}
HeatUpBedExtruder = {
    "input_parameters": []
}
```

```

    "output_parameters": [],
    "module": "printer_3d"
}
Print = {
    "input_parameters": ["GCodeFile:str"],
    "output_parameters": [],
    "module": "printer_3d"
}
GenerateGCode = {
    "input_parameters": ["project:str"],
    "output_parameters": ["gcode:str"],
    "module": "manufacturer"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

21. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

22. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

23. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 4.2

Process description:

When craving a custom 3D model, as a first step you have to instruct an artist to create a project for you. First you have to send him several sketches, and then tell him what to change until you are satisfied with the result. After that, you choose a plastic color you want to use for 3D printing. If you have the color at home (in stock), you check how much color you have left. If it's under 100 grams, you put it on your shopping list. If you do not have the color at home, you order it. When you have the plastic, you can turn the printer on and heat up the bed and the extruder (it continuously sends you back its status). While doing that, you can generate the gcode file for your printer out of the STL sent to you by the artist. After that you print the model.

Code:

```
from manufacturer import SendSketches, ChoosePlasticColor,
CheckColorAvailability, OrderColor, CheckColorQuantity,
UpdateShoppingList, GenerateGCode
from printer_3d import TurnOn, HeatUpBedExtruder, Print

def create_custom_3d_model(sketches, changes, color, project):
    # Send sketches to the artist and tell him what to change
    SendSketches.call(sketches)
    for change in changes:
        # Assume that there is a function to send changes to the artist
        SendChangesToArtist.call(change)

    # Choose a plastic color for 3D printing
    ChoosePlasticColor.call(color)

    # Check if the color is available and if there is enough quantity
    if CheckColorAvailability.call(color):
        if CheckColorQuantity.call(color) < 100:
            UpdateShoppingList.call(color)
        else:
            OrderColor.call(color)

    # Turn the printer on and heat up the bed and the extruder
    TurnOn.call()
    HeatUpBedExtruder.call()

    # Generate the gcode file for the printer
    gcode = GenerateGCode.call(project)

    # Print the model
    Print.call(gcode)
```

Tools:

```
SendSketches = {
    "input_parameters": [] ,
```

```

    "output_parameters": ["project:str"],
    "module": "manufacturer"
}
ChoosePlasticColor = {
    "input_parameters": [],
    "output_parameters": ["color:int"],
    "module": "manufacturer"
}
CheckColorQuantity = {
    "input_parameters": ["color:int"],
    "output_parameters": ["quantity:int"],
    "module": "manufacturer"
}
CheckColorAvailability = {
    "input_parameters": ["color:int"],
    "output_parameters": ["availability:bool"],
    "module": "manufacturer"
}
OrderColor = {
    "input_parameters": ["color:int"],
    "output_parameters": [],
    "module": "manufacturer"
}
TurnOn = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "printer_3d"
}
UpdateShoppingList = {
    "input_parameters": ["color:int"],
    "output_parameters": [],
    "module": "manufacturer"
}
HeatUpBedExtruder = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "printer_3d"
}
Print = {
    "input_parameters": ["GCodeFile:str"],
    "output_parameters": [],
    "module": "printer_3d"
}
GenerateGCode = {
    "input_parameters": ["project:str"],
    "output_parameters": ["gcode:str"],
    "module": "manufacturer"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

24. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

25. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

26. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Ver: Very good

Part 5.1

Process description:

Plastic injection molding is a manufacturing process for producing a variety of parts by injecting molten plastic material into a mold, and letting it cool and solidify into the desired end product. Our interest is in the quality assurance process which involves real-time monitoring of injection molding parameters. As each batch of molten plastic enters the mold, sensors capture data on temperature, pressure, and fill rates. The system analyzes this data to ensure that the molding parameters align with the specified standards. If any deviations are detected, the system triggers adjustments to the injection molding machine settings, allowing for immediate correction. Once the parameters are within the acceptable range, the system authorizes the production run to continue. This dynamic monitoring process guarantees the consistency and quality of the plastic molded components, minimizing the risk of defects and ensuring adherence to precise manufacturing specifications.

Code:

```
from tools.mold_is import SensorMeasure
from tools.mold_is import AnalyzeMold
from tools.mold_is import AuthorizeProduction
def monitor_and_adjust_parameters():
    # Monitor Injection Molding Parameters
    temperature, pressure, fill_rate = SensorMeasure.call()

    # Analyze the parameters to check for deviations
    deviation = AnalyzeMold.call(temperature=temperature,
                                 pressure=pressure, fill_rate=fill_rate)

    if deviation:
        # Adjust Machine Settings if deviations are detected
        # Assuming there's a tool or method to adjust the settings which
        # is not specified
        # For the sake of example, we'll assume a placeholder function
        # AdjustSettings.call() exists
        # AdjustSettings.call()
        # Since the actual tool for adjustment is not provided, we'll
        # just print a message
        print("Adjusting machine settings...")
        return False # Indicating that adjustments were made and
        # monitoring needs to be repeated
    else:
        # Authorize Production Run if parameters are acceptable
        AuthorizeProduction.call()
        return True # Indicating that the process can proceed

def main_process():
    while True:
        process_completed = monitor_and_adjust_parameters()
        if process_completed:
```

```

        break # Exit the loop once the production run is authorized

if __name__ == "__main__":
    main_process()

Tools:
SensorMeasure = {
    "input_parameters": [],
    "output_parameters": ['temperature:int', 'pressure:int',
'fill_rate:int'],
    "module": "mold_is"
}
AnalyzeMold = {
    "input_parameters": ['temperature:int', 'pressure:int',
'fill_rate:int'],
    "output_parameters": ['deviation:bool'],
    "module": "mold_is"
}
AdjustMold = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "mold_is"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

27. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

Other

28. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Very Very good

29. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 5.2

Process description:

Plastic injection molding is a manufacturing process for producing a variety of parts by injecting molten plastic material into a mold, and letting it cool and solidify into the desired end product. Our interest is in the quality assurance process which involves real-time monitoring of injection molding parameters. As each batch of molten plastic enters the mold, sensors capture data on temperature, pressure, and fill rates. The system analyzes this data to ensure that the molding parameters align with the specified standards. If any deviations are detected, the system triggers adjustments to the injection molding machine settings, allowing for immediate correction. Once the parameters are within the acceptable range, the system authorizes the production run to continue. This dynamic monitoring process guarantees the consistency and quality of the plastic molded components, minimizing the risk of defects and ensuring adherence to precise manufacturing specifications.

Code:

```
from mold_is import SensorMeasure, AnalyzeMold, AdjustMold,
AuthorizeProduction

def quality_assurance_process():
    # Measure the temperature, pressure and fill rates of the mold
    temperature, pressure, fill_rate = SensorMeasure().call()

    # Analyze the measured data
    deviation = AnalyzeMold().call(temperature, pressure, fill_rate)

    # If any deviations are detected, adjust the settings
    if deviation:
        AdjustMold().call()

    # Once the parameters are within the acceptable range, authorize the
    # production run to continue
    else:
        AuthorizeProduction().call()
```

Tools:

```
SensorMeasure = {
    "input_parameters": [],
    "output_parameters": ['temperature:int', 'pressure:int',
    'fill_rate:int'],
    "module": "mold_is"
}

AnalyzeMold = {
    "input_parameters": ['temperature:int', 'pressure:int',
    'fill_rate:int'],
    "output_parameters": ['deviation:boolean'],
    "module": "mold_is"
}
```

```

AdjustMold = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "mold_is"
}
AuthorizeProduction = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "mold_is"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

30. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

31. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

32. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 6.1

Process description:

The production of custom metal brackets begins with order processing. The warehouse department evaluates the parts lists and in parallel the production planning department configures the robotic assembly line accordingly. The automated precision machine cuts the metal and the welding machine assembles the parts into brackets. A computer vision inspection system then checks for quality assurance. If defective brackets are detected, the process ends. After inspection, a coating system enhances durability. Finally, the process is complete.

Code:

```
from tools.crm_is import AcceptOrder
from tools.wms_is import RetrieveRawMaterials
from tools.assembly_line import ConfigureAssemblyLine
from tools.precision_machine import CutMetal
from tools.welding_machine import AssembleParts
from tools.worker import AssembleParts
from tools.vision_is import CheckQualityBrackets
from tools.coating_machine import EnhanceProduct
import threading

# Assuming the tools are already imported as per the guidelines
# from tools import AcceptOrder, RetrieveRawMaterials,
ConfigureAssemblyLine, CutMetal, AssembleParts, CheckQualityBrackets,
EnhanceProduct

def process_order(product_id, part_list):
    # Accept the order
    order_accepted = AcceptOrder.call(product_id=product_id)
    if not order_accepted:
        return False

    # Parallel execution for evaluating parts list and configuring the
    # assembly line
    def evaluate_parts():
        global parts_retrieved
        parts_retrieved = RetrieveRawMaterials.call(part_list=part_list)

    def configure_assembly_line():
        ConfigureAssemblyLine.call()

    thread1 = threading.Thread(target=evaluate_parts)
    thread2 = threading.Thread(target=configure_assembly_line)

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()
```

```

    if not parts_retrieved:
        return False

    # Sequential tasks for cutting metal, assembling brackets, and
    # inspecting them
    CutMetal.call()
    AssembleParts.call()
    quality_ok = CheckQualityBrackets.call()

    if not quality_ok:
        return False

    # Apply coating if the brackets pass the quality inspection
    EnhanceProduct.call()

    return True

if __name__ == "__main__":
    product_id = 123  # Example product ID
    part_list = ['part1', 'part2', 'part3']  # Example part list
    process_completed = process_order(product_id, part_list)
    if process_completed:
        print("Process completed successfully.")
    else:
        print("Process terminated due to an error or quality issue.")

```

Tools:

```

AcceptOrder = {
    "input_parameters": ["product_id:int"],
    "output_parameters": ["order_accepted:boolean"],
    "module": "crm_is"
}
RetrieveRawMaterials = {
    "input_parameters": ["part_list:list"],
    "output_parameters": ["retrieved:boolean"],
    "module": "wms_is"
}
ConfigureAssemblyLine = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "assembly_line"
}
CutMetal = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "precision_machine"
}
AssembleParts = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "welding_machine"
}

```

```

}

AssembleParts = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "worker"
}
CheckQualityBrackets = {
    "input_parameters": [],
    "output_parameters": ["quality_ok:boolean"],
    "module": "vision_is"
}
EnhanceProduct = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "coating_machine"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

33. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

34. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

35. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 6.2

Process description:

The production of custom metal brackets begins with order processing. The warehouse department evaluates the parts lists and in parallel the production planning department configures the robotic assembly line accordingly. The automated precision machine cuts the metal and the welding machine assembles the parts into brackets. A computer vision inspection system then checks for quality assurance. If defective brackets are detected, the process ends. After inspection, a coating system enhances durability. Finally, the process is complete.

Code:

```
from manufacturer import OrderParts
from wms_is import RetrieveRawMaterials
from assembly_line import ConfigureAssemblyLine
from precision_machine import CutMetal
from welding_machine import AssembleParts
from vision_is import AnalyzeWarehouse
from coating_machine import ApplyCoating

def produce_brackets():
    # Order processing
    OrderParts.call()

    # Warehouse department evaluates the parts lists
    parts_retrieved = RetrieveRawMaterials.call(part_list)

    # Production planning department configures the robotic assembly line
    ConfigureAssemblyLine.call()

    if not parts_retrieved:
        return "Parts not retrieved. Process ended."

    # The automated precision machine cuts the metal
    CutMetal.call()

    # The welding machine assembles the parts into brackets
    AssembleParts.call()

    # A computer vision inspection system checks for quality assurance
    inspection_passed = AnalyzeWarehouse.call()

    if not inspection_passed:
        return "Defective brackets detected. Process ended."

    # A coating system enhances durability
    ApplyCoating.call()

return "Process complete."
```

```

Tools:

OrderParts = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "actor": "manufacturer"
}

RetrieveRawMaterials = {
    "input_parameters": ["part_list:list"],
    "output_parameters": ["retrieved:bool"],
    "module": "wms_is"
}

ConfigureAssemblyLine = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "assembly_line"
}

CutMetal = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "precision_machine"
}

AssembleParts = {
    "input_parameters": [],
    "output_parameters": [],
    "module": "welding_machine"
}

AnalyzeWarehouse = {
    "input_parameters": ["image:np.matrix", "type_cardboard:str"],
    "output_parameters": ["location:str"],
    "module": "vision_is"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

36. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

37. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

38. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Ver: Very good

Part 7.1

Process description:

You produce airplanes. While the base model is always the same, your customer likes to pimp their airplane with fancy interior. You let them decide between a number of 5 different bars, the number of seats, their color, and the amount of water in the toilets of their plane (you can come up with additional stuff). After you received the specifications you send the requirements to different teams to manufacture the parts. E.g. manufacturing the vodka bar is done by the Russian team, the whiskey bar is manufactured by an Irish team. After receiving the individual bits and pieces, the interior of the plane is assembled, and the plane is sent on a test flight. During the flight you create a test protocol which is sent to you and the customer. The plane is delivered to the customer, which has to confirm.

Code:

```
from tools.crm_is import ReceiveOrder
from tools.manufacturer import BuySnacks
from tools.manufacturer import InformStorehouseEngineering
from tools.manufacturer import AssembleInterior
from tools.manufacturer import OrderParts
from tools.manufacturer import CreateTestProtocol
import threading

# Assuming the tools are already imported and available for use

def manufacture_parts(part_list, product_id):
    # Inform storehouse and engineering departments about the parts and
    product id
    InformStorehouseEngineering.call(part_list=part_list,
product_id=product_id)
    # Manufacturer orders the parts online
    OrderParts.call(part_list=part_list)

def assemble_and_test(part_list, product_id):
    # Assemble the interior of the plane
    AssembleInterior.call(part_list=part_list, product_id=product_id)
    # Create the test protocol for the product
    test_protocol = CreateTestProtocol.call(part_list=part_list,
product_id=product_id)
    return test_protocol

def process():
    # Sales department receives a new order specification from customer
    part_list, product_id = ReceiveOrder.call()

    # Manufacture parts in parallel
    threads = []
    for _ in range(4): # Assuming there are 4 different parts to
manufacture in parallel
```

```

        thread = threading.Thread(target=manufacture_parts, args=
(part_list, product_id))
            threads.append(thread)
            thread.start()

    for thread in threads:
        thread.join()

    # Assemble interior and test
    test_protocol = assemble_and_test(part_list, product_id)

    # Deliver plane to customer and wait for confirmation
    # Assuming a function DeliverPlane exists for delivery and
confirmation
        # This is a placeholder as the actual delivery and confirmation
process is not detailed in the tools
        # DeliverPlane.call(product_id=product_id,
test_protocol=test_protocol)

    return "Process completed successfully."

if __name__ == "__main__":
    result = process()
    print(result)

Tools:
ReceiveOrder = {
    "input_parameters": [],
    "output_parameters": ["part_list:list", "product_id:int"],
    "module": "crm_is"
}
BuySnacks = {
    "input_parameters": ["people:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
InformStorehouseEngineering = {
    "input_parameters": ["part_list:list", "product_id:int"],
    "output_parameters": [],
    "module": "manufacturer"
}
AssembleInterior = {
    "input_parameters": ["part_list:list", "product_id:int"],
    "output_parameters": [],
    "module": "manufacturer"
}
OrderParts = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
CreateTestProtocol = {

```

```
"input_parameters": ["part_list:list", "product_id:int"],  
"output_parameters": ["test_protocol:str"],  
"module": "manufacturer"  
}
```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

39. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Very Very good

40. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Very Very good

41. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 7.2

Process description:

You produce airplanes. While the base model is always the same, your customer likes to pimp their airplane with fancy interior. You let them decide between a number of 5 different bars, the number of seats, their color, and the amount of water in the toilets of their plane (you can come up with additional stuff). After you received the specifications you send the requirements to different teams to manufacture the parts. E.g. manufacturing the vodka bar is done by the Russian team, the whiskey bar is manufactured by an Irish team. After receiving the individual bits and pieces, the interior of the plane is assembled, and the plane is sent on a test flight. During the flight you create a test protocol which is sent to you and the customer. The plane is delivered to the customer, which has to confirm.

Code:

```
from manufacturer import AssembleInterior, SendRequirements,
CreateTestProtocol

class AirplaneManufacturer:
    def __init__(self):
        self.teams = {
            'vodka_bar': 'Russian team',
            'whiskey_bar': 'Irish team',
            # Add other teams as needed
        }

    def produce_airplane(self, specifications: dict):
        # Send requirements to different teams
        for part, team in self.teams.items():
            if part in specifications:
                SendRequirements.call([part])

        # Wait for parts to be manufactured and delivered
        # This can be simulated with a delay or you can implement a real
        waiting mechanism
        # time.sleep(60)

        # Assemble the interior of the plane
        AssembleInterior.call(specifications['parts'],
specifications['product_id'])

        # Send the plane on a test flight and create a test protocol
        test_protocol = CreateTestProtocol.call()

        # Send the test protocol to the manufacturer and the customer
        # send_test_protocol(test_protocol, manufacturer, customer)

        # Deliver the plane to the customer and wait for confirmation
        # deliver_plane(customer)
```

```

# wait_for_confirmation(customer)

Tools:
AssembleInterior = {
    "input_parameters": ["part_list:list", "product_id:int"],
    "output_parameters": [],
    "module": "manufacturer"
}
SendRequirements = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
CreateTestProtocol = {
    "input_parameters": ["part_list:list", "product_id:int"],
    "output_parameters": ["test_protocol:str"],
    "module": "manufacturer"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

42. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Very Very good

43. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Very Very good

44. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 8.1

Process description:

The warehouse of Grimaldi is a warehouse that stores cardboard rolls. A cardboard roll is used to produce cardboards. There exists two types of cardboard: the white cardboard and the brown cardboard. The warehouse stores the cardboard rolls depending on the type of cardboard. When a new cardboard roll arrives at the warehouse, the worker checks the type of cardboard and enter this information inside the WMS system. The system automatically capture an image of the current status of the warehouse. By analyzing the image, the system identifies the location where the cardboard roll should be stored. Then the worker stores the cardboard rool in the identified location in the warehouse and the system updates the stock of that cardboard rolls in the warehouse.

Code:

```
from tools.worker import StoreCardboardRoll
from tools.vision_is import AnalyzeWarehouse
from tools.worker import CheckTypeCardboard
from tools.worker import InsertCardboardTypeInfo
from tools.wms_is import UpdateStock
import numpy as np

# Assuming the necessary tool classes are already imported as per the
guidelines.

def process_new_cardboard_roll(image: np.matrix):
    # Check the type of the cardboard
    type_cardboard = CheckTypeCardboard.call()

    # Enter the type of cardboard into the WMS system
    InsertCardboardTypeInfo.call(type_cardboard=type_cardboard)

    # Capture an image of the current warehouse status (assuming image is
already provided)
    # Note: The process description mentions capturing an image, but
since it's not a tool and the image is already an input, we'll proceed
with the given image.

    # Identify the storage location based on the image and type of
cardboard
    location = AnalyzeWarehouse.call(image=image,
type_cardboard=type_cardboard)

    # Store the cardboard roll in the identified location
    StoreCardboardRoll.call(location=location)

    # Update the stock in the system
    UpdateStock.call(product_type=type_cardboard)

if __name__ == "__main__":
```

```

# Example usage
# Assuming an example image of the warehouse is provided as a numpy
matrix
example_image = np.matrix([[0, 0, 1], [1, 0, 0], [0, 1, 0]]) # Simplified representation
process_new_cardboard_roll(example_image)

```

Tools:

```

StoreCardboardRoll = {
    "input_parameters": ["location:str"],
    "output_parameters": [],
    "module": "worker"
}
AnalyzeWarehouse = {
    "input_parameters": ["image:np.matrix", "type_cardboard:str"],
    "output_parameters": ["location:str"],
    "module": "vision_is"
}
CheckTypeCardboard = {
    "input_parameters": [],
    "output_parameters": ["type_cardboard:str"],
    "module": "worker"
}
InsertCardboardTypeInfo = {
    "input_parameters": ["type_cardboard:str"],
    "output_parameters": [],
    "module": "worker"
}
UpdateStock = {
    "input_parameters": ["product_type:str"],
    "output_parameters": [],
    "module": "wms_is"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

45. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

46. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

47. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Ver: Very good

Part 8.2

Process description:

The warehouse of Grimaldi is a warehouse that stores cardboard rolls. A cardboard roll is used to produce cardboards. There exists two types of cardboard: the white cardboard and the brown cardboard. The warehouse stores the cardboard rolls depending on the type of cardboard. When a new cardboard roll arrives at the warehouse, the worker checks the type of cardboard and enter this information inside the WMS system. The system automatically capture an image of the current status of the warehouse. By analyzing the image, the system identifies the location where the cardboard roll should be stored. Then the worker stores the cardboard rool in the identified location in the warehouse and the system updates the stock of that cardboard rolls in the warehouse.

Code:

```
from vision_is import AnalyzeWarehouse
from worker import CheckTypeCardboard, StoreCardboardRoll
from wms_is import UpdateStock

def process_new_cardboard_roll(image: np.matrix):
    # Worker checks the type of cardboard
    type_cardboard = CheckTypeCardboard().call()

    # System identifies the location where the cardboard roll should be
    # stored
    location = AnalyzeWarehouse().call(image, type_cardboard)

    # Worker stores the cardboard roll in the identified location
    StoreCardboardRoll().call(location)

    # System updates the stock of that cardboard rolls in the warehouse
    UpdateStock().call(type_cardboard)
```

Tools:

```
StoreCardboardRoll = {
    "input_parameters": ["location:str"],
    "output_parameters": [],
    "module": "worker"
}

AnalyzeWarehouse = {
    "input_parameters": ["image:np.matrix", "type_cardboard:str"],
    "output_parameters": ["location:str"],
    "module": "vision_is"
}

CheckTypeCardboard = {
    "input_parameters": [],
    "output_parameters": ["type_cardboard:str"],
    "module": "worker"
}

UpdateStock = {
```

```
"input_parameters": ["product_type:str"] ,  
"output_parameters": [] ,  
"module": "wms_is"  
}
```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

48. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

49. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

50. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Ver: Very good

Part 9.1

Process description:

You want to build a tree house. First you collect your requirements, and send them to a tree house architect. The architect sends you back a draft, which you refine multiple times with additional requirements. You then create the list of needed materials from the plan. These materials fall into several categories, you order them from several online stores. While the order is processed, you send messages to several of your friends to build the house. After the house is built, you send invitations for a tree house party to your friends. In order to buy the snacks for the party, a list of people that attend the party is created.

Code:

```
from tools.manufacturer import GenerateRequirementTreeHouse
from tools.manufacturer import RefineRequirementsTreeHouse
from tools.manufacturer import SendRequirementsArchitect
from tools.manufacturer import SendRequirements
from tools.manufacturer import OrderParts
from tools.manufacturer import AssembleTreeHouse
from tools.manufacturer import BuySnacks
import threading

def collect_requirements():
    return GenerateRequirementTreeHouse.call()

def refine_requirements(part_list):
    return RefineRequirementsTreeHouse.call(part_list=part_list)

def send_requirements_to_architect(part_list):
    SendRequirementsArchitect.call(part_list=part_list)

def order_materials(part_list):
    OrderParts.call(part_list=part_list)

def send_messages_to_friends():
    # Assuming there's a predefined list of friends
    friends = ["Alice", "Bob", "Charlie"]
    SendRequirements.call(part_list=friends)

def build_tree_house(part_list):
    AssembleTreeHouse.call(part_list=part_list)

def send_invitations():
    # Assuming there's a predefined list of friends
    friends = ["Alice", "Bob", "Charlie"]
    # Sending invitations to the same friends who helped build the tree
    house
    SendRequirements.call(part_list=friends)

def create_list_for_party_snacks():
```

```

# Assuming there's a predefined list of friends who confirmed their
attendance
attending_friends = ["Alice", "Bob"]
BuySnacks.call(people=attending_friends)

def process():
    part_list = collect_requirements()
    refined_part_list = part_list
    additional_requirements = True

    while additional_requirements:
        send_requirements_to_architect(refined_part_list)
        # Simulate receiving a draft and deciding if more refinement is
needed
        # For simplicity, we assume there's a condition that eventually
becomes False
        additional_requirements = False # This would be determined by
some logic
        if additional_requirements:
            refined_part_list = refine_requirements(refined_part_list)

    # Once no additional requirements, create list of materials
    final_part_list = refined_part_list

    # Ordering materials and sending messages to friends in parallel
    order_thread = threading.Thread(target=order_materials, args=
(final_part_list,))
    message_thread = threading.Thread(target=send_messages_to_friends)
    order_thread.start()
    message_thread.start()
    order_thread.join()
    message_thread.join()

    # Building the tree house
    build_tree_house(final_part_list)

    # Sending invitations for the tree house party
    send_invitations()

    # Creating list for party snacks
    create_list_for_party_snacks()

if __name__ == "__main__":
    process()

Tools:
GenerateRequirementTreeHouse = {
    "input_parameters": [],
    "output_parameters": ["part_list:list"],
    "module": "manufacturer"
}
SendRequirementsArchitect = {

```

```

    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
SendRequirements = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
RefineRequirementsTreeHouse = {
    "input_parameters": ["part_list:list"],
    "output_parameters": ["refined_part_list:list"],
    "module": "manufacturer"
}
OrderParts = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
AssembleTreeHouse = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
BuySnacks = {
    "input_parameters": ["people:list"],
    "output_parameters": [],
    "module": "manufacturer"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

51. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

52. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Very Very good

53. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 9.2

Process description:

You want to build a tree house. First you collect your requirements, and send them to a tree house architect. The architect sends you back a draft, which you refine multiple times with additional requirements. You then create the list of needed materials from the plan. These materials fall into several categories, you order them from several online stores. While the order is processed, you send messages to several of your friends to build the house. After the house is built, you send invitations for a tree house party to your friends. In order to buy the snacks for the party, a list of people that attend the party is created.

Code:

```
from manufacturer import GenerateRequirementTreeHouse,
RefineRequirementsTreeHouse, SendRequirementsArchitect, OrderParts,
SendMessage, AssembleTreeHouse, CreateListOfPeople, SendInvitations,
BuySnacks

def build_tree_house():
    # Generate requirements
    generate_req = GenerateRequirementTreeHouse()
    part_list = generate_req.call()

    # Send requirements to architect
    send_req_architect = SendRequirementsArchitect()
    send_req_architect.call(part_list)

    # Refine requirements
    refine_req = RefineRequirementsTreeHouse()
    refined_part_list = refine_req.call(part_list)

    # Order parts
    order_parts = OrderParts()
    order_parts.call(refined_part_list)

    # Send messages to friends to build the house
    send_message = SendMessage()
    send_message.call()

    # Assemble tree house
    assemble_tree_house = AssembleTreeHouse()
    assemble_tree_house.call(refined_part_list)

    # Create list of people for the party
    create_list_of_people = CreateListOfPeople()
    people = create_list_of_people.call()

    # Send invitations
    send_invitations = SendInvitations()
    send_invitations.call(people)
```

```

# Buy snacks
buy_snacks = BuySnacks()
buy_snacks.call(people)

Tools:
GenerateRequirementTreeHouse = {
    "input_parameters": [],
    "output_parameters": ["part_list:list"],
    "module": "manufacturer"
}
SendRequirementsArchitect = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
SendRequirements = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
RefineRequirementsTreeHouse = {
    "input_parameters": ["part_list:list"],
    "output_parameters": ["refined_part_list:list"],
    "module": "manufacturer"
}
OrderParts = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
AssembleTreeHouse = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
SendMessage = {
    "input_parameters": ["message:str"],
    "output_parameters": [],
    "module": "manufacturer"
}
CreateListOfPeople = {
    "input_parameters": [],
    "output_parameters": ["people:list"],
    "module": "manufacturer"
}
SendInvitations = {
    "input_parameters": ["people:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
BuySnacks = {

```

```
"input_parameters": ["people:list"],  
"output_parameters": [],  
"module": "manufacturer"  
}
```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

54. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Very Very good

55. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Very Very good

56. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 10.1

Process description:

A small company manufactures customized bicycles. Whenever the sales department receives an order, a new process instance is created. A member of the sales department can then reject or accept the order for a customized bike. In the former case, the process instance is finished. In the latter case, the storehouse and the engineering department are informed. The storehouse immediately processes the part list of the order and checks the required quantity of each part. If the part is available in-house, it is reserved. If it is not available, it is back-ordered. This procedure is repeated for each item on the part list. In the meantime, the engineering department prepares everything for the assembling of the ordered bicycle. If the storehouse has successfully reserved or back-ordered every item of the part list and the preparation activity has finished, the engineering department assembles the bicycle. Afterwards, the sales department ships the bicycle to the customer and finishes the process instance.

Code:

```
from tools.crm_is import ReceiveOrder
from tools.crm_is import AcceptOrder
from tools.manufacturer import InformStorehouseEngineering
from tools.manufacturer import SendRequirements
from tools.manufacturer import OrderParts
from tools.manufacturer import AssembleBicycle
import threading

def process_part_list(part_list):
    for part in part_list:
        # Simulate checking if part is available and either reserving or
        # back-ordering it
        # This is a simplification, as the actual logic for checking
        # availability and reserving/back-ordering is not provided
        # Assume a function check_and_process_part(part) exists for this
        # purpose
        check_and_process_part(part)
    # After processing all parts, inform that all parts are processed
    print("All parts processed.")

def prepare_for_assembling():
    # Simulate preparing for assembling
    # This is a simplification, as the actual logic for preparation is
    # not provided
    # Assume a function prepare_assembly() exists for this purpose
    prepare_assembly()
    print("Preparation for assembling finished.")

def process_order():
    part_list, product_id = ReceiveOrder.call()
    order_accepted = AcceptOrder.call(product_id=product_id)
```

```

    if not order_accepted:
        print("Order rejected.")
        return

    InformStorehouseEngineering.call(part_list=part_list,
product_id=product_id)

    # Start parallel tasks for processing part list and preparing for
assembling
    part_list_thread = threading.Thread(target=process_part_list, args=
(part_list,))
    prepare_assembling_thread =
threading.Thread(target=prepare_for_assembling)

    part_list_thread.start()
    prepare_assembling_thread.start()

    # Wait for both threads to complete
    part_list_thread.join()
    prepare_assembling_thread.join()

    # After both tasks are completed, proceed with assembling the bicycle
    AssembleBicycle.call(part_list=part_list)
    print("Bicycle assembled and ready to be shipped.")

    # Simulate shipping the bicycle
    # This is a simplification, as the actual logic for shipping is not
provided
    # Assume a function ship_bicycle(product_id) exists for this purpose
    ship_bicycle(product_id)
    print("Bicycle shipped to customer.")

if __name__ == "__main__":
    process_order()

```

Tools:

```

ReceiveOrder = {
    "input_parameters": [],
    "output_parameters": ["part_list:list", "product_id:int"],
    "module": "crm_is"
}

AcceptOrder = {
    "input_parameters": ["product_id:int"],
    "output_parameters": ["order_accepted:boolean"],
    "module": "crm_is"
}

InformStorehouseEngineering = {
    "input_parameters": ["part_list:list", "product_id:int"],
    "output_parameters": [],
    "module": "manufacturer"
}

SendRequirements = {

```

```

    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
OrderParts = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}
AssembleBicycle = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

57. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

58. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

59. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Very Very good

Part 10.2

Process description:

A small company manufactures customized bicycles. Whenever the sales department receives an order, a new process instance is created. A member of the sales department can then reject or accept the order for a customized bike. In the former case, the process instance is finished. In the latter case, the storehouse and the engineering department are informed. The storehouse immediately processes the part list of the order and checks the required quantity of each part. If the part is available in-house, it is reserved. If it is not available, it is back-ordered. This procedure is repeated for each item on the part list. In the meantime, the engineering department prepares everything for the assembling of the ordered bicycle. If the storehouse has successfully reserved or back-ordered every item of the part list and the preparation activity has finished, the engineering department assembles the bicycle. Afterwards, the sales department ships the bicycle to the customer and finishes the process instance.

Code:

```
from crm_is import ReceiveOrder, AcceptOrder
from manufacturer import InformStorehouseEngineering, AssembleBicycle
from wms_is import OrderRawMaterial

def process_order():
    # Sales department receives a new order
    part_list, product_id = ReceiveOrder.call()

    # Sales department can then reject or accept the order for a
    # customized bike
    order_accepted = AcceptOrder.call(product_id)

    if not order_accepted:
        # If the order is rejected, the process instance is finished
        return "Order rejected."

    # If the order is accepted, the storehouse and the engineering
    # department are informed
    InformStorehouseEngineering.call(part_list, product_id)

    # The storehouse immediately processes the part list of the order and
    # checks the required quantity of each part
    for part in part_list:
        # If the part is not available, it is back-ordered
        OrderRawMaterial.call(part)

    # The engineering department prepares everything for the assembling
    # of the ordered bicycle
    # If the storehouse has successfully reserved or back-ordered every
    # item of the part list and the preparation activity has finished, the
    # engineering department assembles the bicycle
```

```

AssembleBicycle.call(part_list)

    # Afterwards, the sales department ships the bicycle to the customer
    # and finishes the process instance
    return "Bicycle assembled and shipped to the customer."

```

Tools:

```

ReceiveOrder = {
    "input_parameters": [],
    "output_parameters": ["part_list:list", "product_id:int"],
    "module": "crm_is"
}

AcceptOrder = {
    "input_parameters": ["product_id:int"],
    "output_parameters": ["order_accepted:bool"],
    "module": "crm_is"
}

InformStorehouseEngineering = {
    "input_parameters": ["part_list:list", "product_id:int"],
    "output_parameters": [],
    "module": "manufacturer"
}

AssembleBicycle = {
    "input_parameters": ["part_list:list"],
    "output_parameters": [],
    "module": "manufacturer"
}

OrderRawMaterial = {
    "input_parameters": ["part:str"],
    "output_parameters": [],
    "module": "wms_is"
}

```

Evaluate the code based on:

- the control and data flows with respect to the process description (and parameters of the tools),
- the correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import),
- presence of additional functions instead of using available tools.

60. Criterion 1 *

Control and data flows with respect to the process description (and parameters of the tools)

1 2 3 4 5

Ver: Very good

61. Criterion 2 *

Correctness of the code (no syntax errors such as indentation, invalid functions, invalid calls, invalid import)

1 2 3 4 5

Ver: Very good

62. Criterion 3 *

Presence of additional functions instead of using available tools.

1 2 3 4 5

Ver: Very good
