Test Case Outline

Introduction

This document outlines essential test cases for our system, aligning with the strategic directives outlined in the Software Configuration Management (SCM) plan. Tests play a crucial role in ensuring the reliability and functionality of our system. As per the SCM plan, a robust testing strategy is imperative for success. This guide provides a concise overview of test cases, emphasizing their critical role in fortifying the system against defects and uncertainties. Each outlined test case contributes to the overall quality assurance process, aligning with our commitment to delivering a resilient and high-performing system.

Index

References

-SCM plan

Glossary

-ECP - Equivalence Class Partitioning
-BVA - Boundary Value Analysis

ProductSystem

1-addProcess

1.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-"Product System" Instantiated 2-"Process" Instantiated	1-"ProductSystem" not Instantiated 2-"Process" not Instantiated
Number of Inputs	1	! 1
Type	Process	!Process
Restrictions	1-Process can't be null 2-Process can't be already in the ProductSystem	Process is null

1.2- BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME
#01_addProcess_valid	addProcess	1. Null Process 2. Product System must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	process	1	Process is added to the processes list
#02_addProcess_null	addProcess	1. Non-null Process 2. Product System must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	null	1	"Input is null"

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME
#03_addProcess_duplicate	addProcess	1. process must be already added 2. Product System must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	process	1	"This Input already exists"

1.3- Scope

The tests for the addProcess function encompass various scenarios to ensure robust functionality. They include cases for valid, null, and duplicate inputs, along with boundary tests to assess the function's behavior with different numbers of processes. Additionally, performance tests were conducted to verify the efficiency of the function when adding a substantial number of processes. These tests aim to ensure that the function operates consistently and meets the system requirements.

2-addFlow

2.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-"Product System" Instantiated 2-"Flow" Instantiated	1-"ProductSystem" not Instantiated 2-"Flow" not Instantiated
Number of Inputs	1	! 1
Type	Flow	!Flow
Restrictions	1-Flow can't be null 2-Flow can't be already in the ProductSystem	Flow is null

2.2-BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME	EX DA
#04_addFlow_valid	addFlow	1. Non-null Flow 2. Product System must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	flow	1	Flow is added to the flows list	10
#05_addFlow_null	addFlow	1. Null Flow 2. Product System must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	null	1	"Input is null"	10
#06_addFlow_duplicate	addFlow	1. flow must be already added 2. Product System must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	flow	1	"This Input already exists"	10

2.3-Scope

Tests for the addFlow function cover scenarios such as valid input, handling null flows, and preventing the addition of duplicate flows. Boundary testing assesses the function's versatility with different flow quantities, while performance testing evaluates its efficiency with a substantial number of flows. The objective is to ensure that addFlow functions consistently, securely, and efficiently, meeting system

3-removeProcess

3.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-Product System Instantiated 2-"Process" Instantiated 3-Process already added in the System	ProductSystem not Instantiated
Number of Inputs	1	! 1

Criteria	Valid Class	Invalid Class
Type	String	!String
Restrictions	1-Process can't be null 2-Process is already added in ProductSystem 3- The name of the process can't be null	1-Process is already added in ProductSystem 2-Process can't be null 3-Process isn't in the ProductSystem 4-The name of the process is null

3.2-BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPE OUT
#07_removeProcess_valid	removeProcess	1. ProductSystem must be instantiated 2.process must be already added	1. Call the function 2. Initiate the function with the appropriate parameter	processName	1	"proc retur
#08_removeProcess_null	removeProcess	1. ProductSystem must be instantiated 2.name must be null 3. process list must not be empty	1. Call the function 2. Initiate the function with the appropriate parameter	null	1	"Nam must be nu
#09_removeProcess_nonExisting	removeProcess	1. ProductSystem must be instantiated 2.process must not be added 3. process list must not be empty	1. Call the function 2. Initiate the function with the appropriate parameter	processName	1	"Proc not f
#10_removeProcess_empty	removeProcess	1. ProductSystem must be instantiated 2. process list must be empty	1. Call the function 2. Initiate the function with the appropriate parameter	processName	1	"Emp list"

3.3-Scope

The removeProcess function is tested for its ability to efficiently remove processes under diverse conditions. The scope includes scenarios with instantiated productSystem and processes, checking the function's accuracy in handling the presence or absence of processes in the system. Testing covers both singular and invalid input cases, examining the function's behavior with one or zero inputs. Type testing is conducted for valid and invalid string inputs, while restrictions ensure proper handling of null processes, processes already added to the product system, and null process names. The objective is to validate the consistent and secure performance of the removeProcess function across various situations, aligning with system

4-removeFlow

3.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-Product System Instantiated 2-"Flow" Instantiated 3-Flow already added in the System	ProductSystem not Instantiated
Number of Inputs	1	! 1
Type	String	!String
Restrictions	1-Flow can't be null 2-Flow is already added in ProductSystem 3- The name of the Flow can't be null	1-Flow is already added in ProductSystem 2-Flow can't be null 3-Flow isn't in the ProductSystem 4-The name of the Flow is null

3.2-BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME
#11_removeFlow_valid	removeFlow	1. ProductSystem must be instantiated 2.flow must be already added	1. Call the function 2. Initiate the function with the appropriate parameter	flowName	1	"flow "is returned
#12_removeFlow_null	removeFlow	1. ProductSystem must be instantiated 2.name must be null 3. flow list must not be empty	1. Call the function 2. Initiate the function with the appropriate parameter	null	1	"Name must not be null"
#13_removeFlow_nonExisting	removeFlow	1. ProductSystem must be instantiated 2.flow must not be added 3. flows list must not be empty	1. Call the function 2. Initiate the function with the appropriate parameter	flowName	1	"Flow not found"
#14_removeFlow_empty	removeFlow	1. ProductSystem must be instantiated 2. flow list must be empty	1. Call the function 2. Initiate the function with the appropriate parameter	flowName	1	"Empty list"

3.3-Scope

The removeFlow function is subjected to a comprehensive scope of tests to ascertain its effectiveness in removing flows across different scenarios. The testing encompasses situations involving instantiated product systems and flows, evaluating the function's precision in managing flows present or absent in the system. Tests cover both singular and invalid input cases, assessing the function's behavior with one or zero inputs. Type testing is executed for valid and invalid string inputs, while restrictions ensure proper handling of null flows, flows already added to the product system, and null flow names. The primary objective is to validate the consistent and secure performance of the removeFlow function across diverse scenarios, aligning with the specified system requirements.

4-saveToJson

4.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-Product System Instantiated 2-"Process" and "Flow"Instantiated 3-Process and Flow already added in the System	ProductSystem not Instantiated
Number of Inputs	1	! 1
Type	String	!String
Restrictions	Product System Instantiated Process's and flows already added to the system	Product System not Instantiated Dont have precosses or flows in the system

4.2-BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME	EXE DAT
#95_saveToJson_valid	saveToJson	1. ProductSystem must be instantiated 2.Already Process's and Flow's add to the System	1. Call the function			"Conteudo do arquivo JSON:..."	14/(

4.3-Scope

The testing scope for the `saveToJson` function involves validating essential functionalities. It includes confirming successful creation of the specified JSON file, ensuring non-empty content, and verifying the expected structure with key attributes like "flowName." The test assesses the function's accuracy in capturing and saving the product system state and checks its behavior with different inputs, including empty systems. Additionally, it evaluates the function's ability to overwrite existing JSON files with proper warnings, handles errors, and performs cleanup by deleting created files. Performance testing considers varied system sizes, and compatibility testing ensures functionality across different environments. If applicable, integration tests are conducted to verify interactions with other system components. The overarching goal is to ensure the `saveToJson` function consistently and securely saves the product system state according to specified requirements across diverse scenarios.

Process

1-addProcessFlow
1.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-"Process" Instantiated 2-"ProcessFlow" Instantiated	1-"Process" not Instantiated 2-"ProcessFlow" not Instantiated
Number of Inputs	1	! 1
Type	ProcessFlow	!ProcessFlow
Restrictions	1-ProcessFlow can't be null 2-Flow can't be already in the Process	ProcessFlow is null

1.2-BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECT OUTCO
#15_addProcessFlow_valid	addProcessFlow	1. Non-null ProcessFlow 2. Process must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1	Process added t process list
#16_addProcessFlow_null	addProcessFlow	1. Null ProcessFlow 2. Process must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	null	1	"Input is null"
#17_addProcessFlow_duplicate	addProcessFlow	1. processFlow must be already added 2. Process must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1	"This Inp already exists"

2.3-Scope
Tests for the addFlow function cover scenarios such as valid input, handling null flows, and preventing the addition of duplicate flows. Boundary testing assesses the function's versatility with different flow quantities, while performance testing evaluates its efficiency with a substantial number of flows. The objective is to ensure that addFlow functions consistently, securely, and efficiently, meeting system

2-removeProcessFlow
2.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-Product System Instantiated 2-"Process" Instantiated 3-Process already added in the System	ProductSystem not Instantiated
Number of Inputs	1	! 1
Type	String	!String
Restrictions	1-Process can't be null 2-Process is already added in ProductSystem 3- The name of the process can't be null	1-Process is already added in ProductSystem 2-Process can't be null 3-Process isn't in the ProductSystem 4-The name of the process is null

2.2-BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPE OUT
#18_removeProcess_valid	removeProcess	1. ProductSystem must be instantiated 2.process must be already added	1. Call the function 2. Initiate the function with the appropriate parameter	processName	1	"proc retur
#19_removeProcess_null	removeProcess	1. ProductSystem must be instantiated 2.name must be null 3. process list must not be empty	1. Call the function 2. Initiate the function with the appropriate parameter	null	1	"Nam must be nu
#20_removeProcess_nonExisting	removeProcess	1. ProductSystem must be instantiated 2.process must not be added 3. process list must not be empty	1. Call the function 2. Initiate the function with the appropriate parameter	processName	1	"Proc not f
#21_removeProcess_empty	removeProcess	1. ProductSystem must be instantiated 2. process list must be empty	1. Call the function 2. Initiate the function with the appropriate parameter	processName	1	"Emp list"

2.3-Scope

The testing scope for the addProcessFlow function, nested within the Process context, involves validating its effectiveness in adding flows to a process under various conditions. The function is tested within the context of an instantiated process, ensuring its accurate handling of scenarios where the process is already associated with flows or not. The number of inputs is assessed for both singular and invalid cases, examining the function's behavior when provided with one or zero inputs. Type testing includes valid and invalid string inputs. Restrictions are enforced to verify the function's ability to handle scenarios such as null flows, flows already added to the process, and null flow names. The goal is to ensure that the addProcessFlow function performs consistently and securely within the Process context, meeting the requirements defined for this specific functionality.

CircularityCalculator

1-containsByname
1.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-CicularityCalculator Instantiated 2-CircularityFlow already added in the CircularityFlows	CicularityCalculator not Instantiated
Number of Inputs	1	! 1
Type	String	!String

1.2-BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME
#22_containsByname_true	containsByname	1. Non-null name 2. CircularityCalculator must be instantiated 3. A circulrityFlow with the same name must be added to the circularityFlows list	1. Call the function 2. Initiate the function with the appropriate parameter	"name"	1	true

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME
#23_containsByname_false	containsByname	1. Non-null name 2. CircularityCalculator must be instantiated 3. A circulrityFlow without the same name Must be added to the circularityFlows list	1. Call the function 2. Initiate the function with the appropriate parameter	"name"	1	false
#24_containsByname_null	containsByname	1. Null name 2. CircularityCalculator must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	null	1	"Name is null"

1.3-Scope The testing scope for the containsByname functionality comprises three test cases evaluating its behavior in different scenarios. In the first case, a non-null name is provided, the CircularityCalculator is instantiated, and a CircularityFlow with the same name is present in the circularityFlows list. The expected outcome is a return value of true. The second case involves a similar setup, but with a CircularityFlow not sharing the same name, leading to an expected outcome of false. The third case examines the function's response when a null name is provided, expecting the function to handle this case appropriately and return "Name is null." These test cases aim to ensure the correctness and reliability of the containsByname feature under diverse conditions.

2-searchCircularityFlow
2.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-CicularityCalculator Instantiated 2-CircularityFlow already added in the CircularityFlows	CicularityCalculator not Instantiated
Number of Inputs	0	! 0
Type	N/A	!N/A
2.2-BVA		

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS
#25_searchCircularityFlow_nullName	searchCircularityFlow	1. Null productName 2. CircularityCalculator must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#26_searchCircularityFlow_nullProductSystem	searchCircularityFlow	1. Non-null productName 2. CircularityCalculator must be instantiated 3. productSystem must be null	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#27_searchCircularityFlow_processesEmpty	searchCircularityFlow	1. Non-null productName 2. CircularityCalculator must be instantiated 3. productSystem must not be null 4. processes list must be empty	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#28_searchCircularityFlow_processFlowsEmpty	searchCircularityFlow	1. Non-null productName 2. CircularityCalculator must be instantiated 3. processFlows list must be empty	1. Call the function 2. Initiate the function with the appropriate parameter	-	0

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS
#29_searchCircularityFlow_invalidFlowType	searchCircularityFlow	1. Non-null productName 2. CircularityCalculator must be instantiated 3. productSystem must not be null 4. processes list must not be empty 5. FlowType must be invalid	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#30_searchCircularityFlow_flowEnergy	searchCircularityFlow	1. Non-null productName 2. CircularityCalculator must be instantiated 3. productSystem must not be null 4. processes list must not be empty 5. FlowType must be Energy	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#31_searchCircularityFlow_flowMaterialNew	searchCircularityFlow	1. Non-null productName 2. CircularityCalculator must be instantiated 3. productSystem must not be null 4. processes list must not be empty 5.Flow must be type Material 4. A circularity flow with the same name of the flow must not exist in the circularityFlows list	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#32_searchCircularityFlow_flowServicelNew	searchCircularityFlow	1. Non-null productName 2. CircularityCalculator must be instantiated 3. productSystem must not be null 4. processes list must not be empty 5.Flow must be type Service 4. A circularity flow with the same name of the flow must not exist in the circularityFlows list	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#33_searchCircularityFlow_flowMaterial	searchCircularityFlow	1. Non-null productName 2. CircularityCalculator must be instantiated 3. productSystem must not be null 4. processes list must not be empty 5.Flow must be type Material 4. circularityFlow with the same name of the flow must already exist in the circularityFlows list	1. Call the function 2. Initiate the function with the appropriate parameter	-	0

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS
#34_searchCircularityFlow_flowService	searchCircularityFlow	1. Non-null productName 2. CircularityCalculator must be instantiated 3. productSystem must not be null 4. processes list must not be empty 5.Flow must be type Service 4. circularityFlow with the same name of the flow must already exist in the circularityFlows list	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#35_searchCircularityFlow_noProcessForProduct	searchCircularityFlow	1. Non-null productName 2. CircularityCalculator must be instantiated 3. productSystem must not be null 4. processes list must not be empty 5. No processes must be added with the same name introduced	1. Call the function 2. Initiate the function with the appropriate parameter	-	0

2.3-Scope

The testing scope for the searchCircularityFlow functionality involves a set of diverse test cases to ensure the accurate retrieval of Circularity Flows under various conditions. These include scenarios with null or non-null product names, instantiation of the Circularity Calculator, presence or absence of a product system, and the state of processes and process flows. Test cases account for different flow types such as Energy, Material, and Service, considering the existence or absence of corresponding circularity flows in the circularityFlows list. The tests aim to validate that the function behaves as expected, adds or updates circularity flows appropriately, and handles cases where there are no processes with the introduced name. The comprehensive test suite ensures the reliability and correctness of the searchCircularityFlow feature in various scenarios.

3-updateCircularityFlows
1.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-CicularityCalculator Instantiated 2-CircularityFlow already added in the CircularityFlows	CicularityCalculator not Instantiated
Number of Inputs	1	! 1
Type	ProcessFlow	!ProcessFlow
3.2-BVA		

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	IN
#36_updateCircularityFlows_null	updateCircularityFlows	1. Null processFlow 2. CircularityCalculator must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#37_updateCircularityFlows_nullprocessFlowState	updateCircularityFlows	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. processFlow state must be null	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	IN
#38_updateCircularityFlows_emptyCircularityFlows	updateCircularityFlows	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. processFlow state must not be null 4. CircularityFlows must be empty	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#39_updateCircularityFlows_nullProcessType	updateCircularityFlows	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. Process type must be null	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#40_updateCircularityFlows_V	updateCircularityFlows	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. ProcessFlow state must be Virgin	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#41_updateCircularityFlows_Ri	updateCircularityFlows	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. ProcessFlow state must be Recycled 4. Process type must be primary 5. IOFlow must be Input	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#42_updateCircularityFlows_R	updateCircularityFlows	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. ProcessFlow state must be Recycled 4. Process type must be recycling 5. IOFlow must be Input	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#43_updateCircularityFlows_Rr	updateCircularityFlows	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. ProcessFlow state must be Recycled 4. Process type must be recycling 5. IOFlow must be Output	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#44_updateCircularityFlows_Wf	updateCircularityFlows	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. ProcessFlow state must be Waste 4. Process type must be primary	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#45_updateCircularityFlows_Wc	updateCircularityFlows	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. ProcessFlow state must be Waste 4. Process type must be recycling	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1

3.3-Scope The testing scope for the updateCircularityFlows functionality encompasses a series of test cases to validate the proper updating of Circularity Flows under diverse conditions. Test scenarios include handling null process flows, ensuring correct behavior with different process flow states, managing empty Circularity Flows, validating against null process types, and verifying the appropriate updating of Circularity Flows based on specific conditions such as Virgin, Recycling Input, Recycling Output, and Waste states. The tests aim to confirm that the function produces the expected outcomes and maintains accurate updates to Circularity Flows in various scenarios, ensuring the reliability and correctness of the updateCircularityFlows feature.

4-searchProcessType
4.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-CicularityCalculator Instantiated 2-CircularityFlow already added in the CircularityFlows 3.Process added in the processes list	CicularityCalculator not Instantiated
Number of Inputs	1	! 1
Type	ProcessFlow	!ProcessFlow
4.2-BVA		

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS
#46_searchProcessType_null	searchProcessType	1. Null processFlow 2. CircularityCalculator must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	null	1
#47_searchProcessType_nullProductSystem	searchProcessType	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. ProductSystem must be null	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#48_searchProcessType_emptyProcesses	searchProcessType	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. Processes must be empty	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#49_searchProcessType_nullProcessType	searchProcessType	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. Process type is null	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#50_searchProcessType_invalidProcessName	searchProcessType	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. Processes must have a process with the same name as the processFlow processName processType must not be null 4. Processes must not have a process with the same name as the processFlow processName	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1
#51_searchProcessType_primary	searchProcessType	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. Processes must have a process with the same name as the processFlow processName processType must not be null 4. Process type must be primary	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS
#52_searchProcessType_recycling	searchProcessType	1. Non-null processFlow 2. CircularityCalculator must be instantiated 3. Processes must have a process with the same name as the processFlow processName processType must not be null 4. Process type must be recycling	1. Call the function 2. Initiate the function with the appropriate parameter	processFlow	1

4.3-Scope The scope of tests for this table involves evaluating the searchProcessType functionality under various conditions. Test cases cover scenarios such as a null process flow, null product system, empty processes, null process type, invalid process name, and valid process types (both primary and recycling). Each test case specifies the expected outcome based on the given inputs. The objective is to verify the correctness and robustness of the searchProcessType function across different input conditions. The tests aim to ensure that the function can handle various situations and provide the expected results as specified in the test cases.

5-searchFlow
5.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-CicularityCalculator Instantiated 2-CircularityFlow already added in the CircularityFlows 3.Flowadded in the Flows list	CicularityCalculator not Instantiated
Number of Inputs	1	! 1
Type	String	!String
5.2-BVA		

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPEC OUTC
#53_searchFlow_null	searchFlow	1. Null name 2. CircularityCalculator must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	null	1	"Flow r null"
#54_searchFlow_nullProductSystem	searchFlow	1. Non-null name 2. CircularityCalculator must be instantiated 3. ProductSystem must be null	1. Call the function 2. Initiate the function with the appropriate parameter	"name"	1	"Produ is null"
#55_searchFlow_emptyFlows	searchFlow	1. Non-null name 2. CircularityCalculator must be instantiated 3. Flows list must be empty	1. Call the function 2. Initiate the function with the appropriate parameter	"name"	1	"Empty list"
#56_searchFlow_invalidName	searchFlow	1. Non-null name 2. CircularityCalculator must be instantiated 3. Flows list must not be empty 4. Flows list must not have a flow with the same name as the introduced	1. Call the function 2. Initiate the function with the appropriate parameter	"name"	1	"Flow c exist"

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPEC OUTC
#57_searchFlow_valid	searchFlow	1. Non-null name 2. CircularityCalculator must be instantiated 3. Flows list must not be empty 4. Flows list must have a flow with the same name as the introduced	1. Call the function 2. Initiate the function with the appropriate parameter	"name"	1	Flow is returne

5.3-Scope The scope of tests for this table involves evaluating the searchFlow functionality under different conditions. Test cases cover scenarios such as null input for the flow name, null product system, empty flow list, invalid flow name, and a valid flow name. Each test case specifies the expected outcome based on the given inputs. The objective is to verify the correctness and robustness of the searchFlow function across various input conditions. The tests aim to ensure that the function can handle different situations and provide the expected results as specified in the test cases.

6-calculateCircularityFlow
6.1-ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-CicularityCalculator Instantiated 2-CircularityFlow already added in the CircularityFlows 3.Flow added in the Flows list 4.Process added to the processes list	CicularityCalculator not Instantiated
Number of Inputs	0	! 0
Type	N/A	!N/A
6.2-BVA		

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS
#58_calculateCircularityFlow_null	calculateCircularityFlow	1. Null name 2. CircularityCalculator must be instantiated	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#59_calculateCircularityFlow_nullProductSystem	calculateCircularityFlow	1. Non-null name 2. CircularityCalculator must be instantiated 3. ProductSystem must be null	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#60_calculateCircularityFlow_emptyProcesses	calculateCircularityFlow	1. Non-null name 2. CircularityCalculator must be instantiated 3. ProductSystem must not be null 4. Processes must be empty	1. Call the function 2. Initiate the function with the appropriate parameter	-	0
#61_calculateCircularityFlow_emptyFlows	calculateCircularityFlow	1. Non-null name 2. CircularityCalculator must be instantiated 3. ProductSystem must not be null Processes must not be empty	1. Call the function 2. Initiate the function with the appropriate parameter	-	0

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS
#62_calculateCircularityFlow_valid	calculateCircularityFlow	1. Non-null name 2. CircularityCalculator must be instantiated 3. ProductSystem must not be null 4. Processes must not be empty 5. Constant y (all M combined from circularityFlows)!=0 6. 0 < result < 1	1. Call the function 2. Initiate the function with the appropriate parameter	-	0

6.3-Scope The scope of tests for this table involves evaluating the calculateCircularityFlow functionality under various conditions. Test cases cover scenarios such as null input for the product name, null product system, empty process list, and empty flow list. Additionally, there are tests to ensure the function handles valid cases where the constant y (all M combined from circularityFlows) is not equal to zero, and the result falls within the range of 0 to 1. Each test case specifies the expected outcome based on the given inputs. The objective is to validate the correctness and robustness of the calculateCircularityFlow function across different input conditions.

CircularityFlow

1-calculateW

1.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1. CircularityFlow is instantiated	1. CircularityFlow is not instantiated
Number of Inputs	0	!0
Type	N/A	N/A
Restrictions	N/A	N/A

1.2- BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME	EXEC DAT
#63_calculateW_valid	calculateW	1. CircularityFlow must be instantiated	1. Call the function		0	returns W	11/0

1.3- Scope

The scope of tests for the calculateW function involves evaluating its functionality in various scenarios. Test cases cover valid conditions situations. The objective is to verify the correctness and robustness of the calculateW function. The tests aim to ensure that the function provides the expected results as specified in the test cases.

2-calculateM

2.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1. CircularityFlow is instantiated	1. CircularityFlow is not instantiated
Number of Inputs	0	!0
Type	N/A	N/A
Restrictions	N/A	N/A

2.2- BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME	EXEC DAT
#64_calculateM_valid	calculateW	1. CircularityFlow must be instantiated	1. Call the function		0	returns M	11/0

2.3- Scope

The scope of tests for the calculateM function involves evaluating its functionality in various scenarios. Test cases cover valid conditions situations. The objective is to verify the correctness and robustness of the calculateM function. The tests aim to ensure that the function provides the expected results as specified in the test cases.

3-calculateEp

3.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1. CircularityFlow is instantiated 2. CircularityFlow.R is different than 0	1. CircularityFlow isn't instantiated 2. CircularityFlow.R is equal to 0
Number of Inputs	0	!0
Type	N/A	N/A
Restrictions	N/A	N/A

3.2- BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME
#65_calculateEp_valid	calculateEp	1. CircularityFlow must be instantiated 2. CircularityFlow.R should be different than 0	1. Call the function		0	returns Ep
#66_calculateEp_R_equals_0	calculateEp	1. CircularityFlow must be instantiated 2. CircularityFlow.R should be equal to 0	1. Call the function		0	"Divison by zero"

3.3- Scope

The scope of tests for the calculateEp function involves evaluating its functionality in various scenarios. Test cases cover scenarios such as denominator equals 0 and valid conditions situations. The objective is to verify the correctness and robustness of the calculateEp function. The tests aim to ensure that the function provides the expected results as specified in the test cases.

4-calculateEs

4.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1. CircularityFlow is instantiated 2. CircularityFlow.V is different than 0	1. CircularityFlow isn't instantiated 2. CircularityFlow.V is equal to 0
Number of Inputs	0	!0
Type	N/A	N/A
Restrictions	N/A	N/A

4.2- BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME
#67_calculateEs_valid	calculateEs	1. CircularityFlow must be instantiated 2. CircularityFlow.V should be different than 0	1. Call the function		0	returns Es
#68_calculateEs_V_equals_0	calculateEs	1. CircularityFlow must be instantiated 2. CircularityFlow.V should be equal to 0	1. Call the function		0	"Division by zero"

4.3- Scope

The scope of tests for the calculateEs function involves evaluating its functionality in various scenarios. Test cases cover scenarios such as denominator equals 0 and valid conditions situations. The objective is to verify the correctness and robustness of the calculateEs function. The tests aim to ensure that the function provides the expected results as specified in the test cases.

5-calculateX

5.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1. CircularityFlow is instantiated 2. Lavg and Uavg are bigger than 0	1. CircularityFlow isn't instantiated 2. Lavg and Uavg are smaller or equal to 0
Number of Inputs	4	4
Type	int, double, int, double	!int, !double, !int, !double
Restrictions	N/A	N/A

5.2- BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS
#69_calculateX_valid	calculateX	1. CircularityFlow must be instantiated 2. Lavg and Uavg should be bigger than 0	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg	4
#70_calculateX_Lavg_equals_0	calculateX	1. CircularityFlow must be instantiated 2. Lavg should be equal to 0 3. Uavg should be bigger than 0	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg	4
#71_calculateX_Uavg_equals_0	calculateX	1. CircularityFlow must be instantiated 2. Uavg should be equal to 0 3. Lavg should be bigger than 0	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg	4
#72_calculateX_Lavg_and_Uavg_equals_0	calculateX	1. CircularityFlow must be instantiated 2. Lavg and Uavg should be equal to 0	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg	4

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS
#73_calculateX_Lavg_smallerThan_0	calculateX	1. CircularityFlow must be instantiated 2. Lavg should be smaller than 0 3. Uavg should be bigger than 0	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg	4
#74_calculateX_Uavg_smallerThan_0	calculateX	1. CircularityFlow must be instantiated 2. Uavg should be smaller than 0 3. Lavg should be bigger than 0	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg	4
#75_calculateX_Lavg_and_Uavg_smallerThan_0	calculateX	1. CircularityFlow must be instantiated 2. Lavg and Uavg should be smaller than 0	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg	4

5.3- Scope

The scope of tests for the calculateX method involves evaluating its functionality under different conditions. Test cases cover scenarios such as valid input values for CircularityFlow calculations, situations where Lavg or Uavg is zero, and cases where Lavg or Uavg is less than zero. Additionally, there are tests for combinations of these conditions. Each test case specifies the expected outcome based on the given inputs. The objective is to verify the correctness and robustness of the calculateX method across various input conditions.

6-calculateFx

6.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1. CircularityFlow is instantiated 2. CircularityFlow.x is bigger than 0	1. CircularityFlow isn't instantiated 2. CircularityFlow.x is smaller or equal to 0
Number of Inputs	4	4
Type	int, double, int, double	!int, !double, !int, !double
Restrictions	N/A	N/A

6.2- BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME
#76_calculateFx_valid	calculateFx	1. CircularityFlow must be instantiated 2. CircularityFlow.x should be bigger than 0	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg	4	returns Fx
#77_calculateFx_x_equals_0	calculateFx	1. CircularityFlow must be instantiated 2. CircularityFlow.x should be equal to 0	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg	4	"Division by zero"

6.3- Scope

The scope of tests for the calculateFx method involves evaluating its functionality under different conditions. Test cases cover scenarios such as valid input values for CircularityFlow calculations and situations where the attribute CircularityFlow.x is 0. Each test case specifies the expected outcome based on the given inputs. The objective is to verify the correctness and robustness of the calculateFx method across various input conditions.

7-calculateLFI

7.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1. CircularityFlow is instantiated 2. CircularityFlow.M is bigger than 0 3. CircularityFlow.Wf is bigger than CircularityFlow.Wc	1. CircularityFlow isn't instantiated 2. CircularityFlow.M is equal to 0 3. CircularityFlow.Wf is smaller or equal to CircularityFlow.Wc
Number of Inputs	0	0
Type	N/A	N/A
Restrictions	N/A	N/A

7.2- BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXP OUT
#78_calculateLFI_valid	calculateLFI	1. CircularityFlow must be instantiated 2. CircularityFlow.M should be bigger than 0 3. CircularityFlow.Wf should be bigger than CircularityFlow.Wc	1. Call the function		0	retu
#79_calculateLFI_negative	calculateLFI	1. CircularityFlow must be instantiated 2. CircularityFlow.M should be bigger than 0 3. CircularityFlow.Wf should be bigger than CircularityFlow.Wc 4. CircularityFlow.Rr should be bigger than the double of CircularityFlow.V	1. Call the function		0	"LFI be beet and
#80_calculateLFI_biggerThan1	calculateLFI	1. CircularityFlow must be instantiated 2. CircularityFlow.M should be bigger than 0 3. CircularityFlow.Wc should be bigger than CircularityFlow.Wf 4. The difference beetwen the double of CircularityFlow.V minus CircularityFlow.Rr should be bigger than the double of CircularityFlow.M plus the difference beetwen CircularityFlow.Wf and CircularityFlow.Wc divided by 2	1. Call the function		0	"LFI be beet and
#81_calculateLFI_denominatorEquals0	calculateLFI	1. CircularityFlow must be instantiated 2. CircularityFlow.M should be equal to 0 3. CircularityFlow.Wc should be equal to CircularityFlow.Wf	1. Call the function		0	"Divi by z
#82_calculateLFI_Wc_BiggerThan_Wf	calculateLFI	1. CircularityFlow must be instantiated 2. CircularityFlow.M should be bigger than to 0 3. CircularityFlow.Wc should be bigger than CircularityFlow.Wf	1. Call the function		0	"Wc be b than

7.3- Scope

The scope of tests for the LFI method involves evaluating its functionality under different conditions. Test cases cover scenarios such as valid CircularityFlow calculations, situations where the resulting LFI is negative or exceeds 1, cases where the denominator is zero, and instances where CircularityFlow.Wc is greater than CircularityFlow.Wf. Each test case specifies the expected outcome based on the given inputs. The objective is to verify the correctness and robustness of the LFI method across various input conditions.

8-calculateMCIp

8.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1. CircularityFlow is instantiated 2. CircularityFlow.Rr plus CircularityFlow.Wc is equal to the absolute value of CircularityFlow.R 3.CircularityFlow.M is equal or bigger than CircularityFlow.V 4. CircularityFlow.W is smaller or equal to CircularityFlow.M 5. CircularityFlow.R is smaller than CircularityFlow.W 6. CircularityFlow.Wc is smaller or equal to CircularityFlow.Wf 7. CircularityFlow.R is equal to 0 so CircularityFlow.Rr is equal to 0 8. CircularityFlow.LFI is valid 9. CircularityFlow.Fx is valid	1. CircularityFlow isn't instantiated 2. CircularityFlow.Rr plus CircularityFlow.Wc is different than the absolute value of CircularityFlow.R 3.CircularityFlow.M is smaller than CircularityFlow.V 4. CircularityFlow.W is bigger than CircularityFlow.M 5. CircularityFlow.R is equal or bigger than CircularityFlow.W 6. CircularityFlow.Wc is bigger than CircularityFlow.Wf 7. CircularityFlow.R is equal to 0 so CircularityFlow.Rr is different than 0 8. CircularityFlow.LFI isn't valid 9. CircularityFlow.Fx isn't valid
Number of Inputs	4	4
Type	int, double, int, double	!int, !double, !int, !double
Restrictions	N/A	N/A

8.2- BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE
#83_calculateMCIp_valid	calculateMCIp	1. CircularityFlow must be instantiated 2. CircularityFlow.Rr plus CircularityFlow.Wc must be equal to the absolute value of CircularityFlow.R 3.CircularityFlow.M must be equal or bigger than CircularityFlow.V 4. CircularityFlow.W must be smaller or equal to CircularityFlow.M 5. CircularityFlow.R must be smaller than CircularityFlow.W 6. CircularityFlow.Wc should be smaller or equal to CircularityFlow.Wf 7. CircularityFlow.R is equal to 0 and CircularityFlow.Rr is different than 0 8. CircularityFlow.LFI must be valid 9. CircularityFlow.Fx must be valid	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg
#84_calculateMCIp_RrAndWc_DifferentThan_R	calculateMCIp	1. CircularityFlow must be instantiated 2. CircularityFlow.Rr plus CircularityFlow.Wc must be equal to the absolute value of CircularityFlow.R	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE
#85_calculateMCIp_M_smallerThan_V	calculateMCIp	1. CircularityFlow must be instantiated 2. CircularityFlow.Rr plus CircularityFlow.Wc must be equal to the absolute value of CircularityFlow.R 3. CircularityFlow.M must be smaller than CircularityFlow.V	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg
#86_calculateMCIp_W_biggerThan_M	calculateMCIp	1. CircularityFlow must be instantiated 2. CircularityFlow.Rr plus CircularityFlow.Wc must be equal to the absolute value of CircularityFlow.R 3. CircularityFlow.M must be equal or bigger than CircularityFlow.V 4. CircularityFlow.W must be bigger than CircularityFlow.M	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg
#87_calculateMCIp_R_BiggerThan_W	calculateMCIp	1. CircularityFlow must be instantiated 2. CircularityFlow.Rr plus CircularityFlow.Wc must be equal to the absolute value of CircularityFlow.R 3. CircularityFlow.M must be equal or bigger than CircularityFlow.V 4. CircularityFlow.W must be smaller or equal to CircularityFlow.M 5. CircularityFlow.R must be bigger than CircularityFlow.W	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg
#88_calculateMCIp_R_EqualTo_W	calculateMCIp	1. CircularityFlow must be instantiated 2. CircularityFlow.Rr plus CircularityFlow.Wc must be equal to the absolute value of CircularityFlow.R 3. CircularityFlow.M must be equal or bigger than CircularityFlow.V 4. CircularityFlow.W must be smaller or equal to CircularityFlow.M 5. CircularityFlow.R must be equal to CircularityFlow.W	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE
#89_calculateMCIp_R_equalTo0_RrShouldBeDifferentThan_0	calculateMCIp	1. CircularityFlow must be instantiated 2. CircularityFlow.Rr plus CircularityFlow.Wc must be equal to the absolute value of CircularityFlow.R 3. CircularityFlow.M must be equal or bigger than CircularityFlow.V 4. CircularityFlow.W must be smaller or equal to CircularityFlow.M 5. CircularityFlow.R must be smaller than CircularityFlow.W 6. CircularityFlow.Wc should be smaller or equal to CircularityFlow.Wf 7. CircularityFlow.R is equal to 0 and CircularityFlow.Rr is different than 0	1. Call the function 2. Initiate the function with the appropriate parameter	l, lavg, u, uavg

8.3- Scope

The scope of tests for the calculateMCIp method involves evaluating its functionality under different conditions. Test cases cover scenarios such as valid CircularityFlow calculations, situations where specific conditions are violated, and various invalid attribute combinations. Each test case specifies the expected outcome based on the given inputs. The objective is to verify the correctness and robustness of the calculateMCIp method across various input conditions.

Parser

1-loadFromCsvs

1.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	A CSV file exists with valid data.	A CSV file does not exist or is malformed.
Number of Inputs	1 (File name).	1 (Null file name).
Type	String (File name).	Null.
Restrictions	The CSV file follows the expected structure.	The CSV file is empty or has invalid data.

1.2- BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPEC
#90_loadParser_valid	Load from a valid CSV file.	Valid CSV file exists.	Specify the valid file name.	"valid.csv"	1	data ac
#91_loadParser_null	Load from a null CSV file.	Valid CSV name exists.	Specify null file name.	"invalid csv file name"	1	IllegalA is throv
#92_loadParser_empty	Load from an empty CSV file.	Empty CSV file exists.	Specify the empty file name.	"empty.csv"	1	IllegalA is throv
#93_loadParser_malformed	Load from a malformed CSV file.	Malformed CSV file exists.	Specify the malformed file name.	"malformed.csv"	1	IllegalA is throv
#94_loadParser_invalidData	Load from a CSV file with invalid data.	CSV file with invalid data exists.	Specify the file name with invalid data.	Load from a CSV file with invalid data.	1	IllegalA is throv

1.3- Scope The tests for the loadFromCsv function encompass various scenarios to ensure robust functionality. They include cases for valid inputs, null inputs, and invalid csvs files, along with boundary tests to assess the function's behavior with different numbers of processes, flows and process flows.

Exporter

1-saveToCsv

1.1- ECP

Criteria	Valid Class	Invalid Class
Pre Condition	1-Product System Instantiated 2-"Process" and "Flow"Instantiated 3-Process and Flow already added in the System	ProductSystem not Instantiated
Number of Inputs	1	! 1
Type	String	!String
Restrictions	Product System Instantiated Process's and flows already added to the system	Product System not Instantiated Dont have processes or flows in the system

4.2-BVA

TEST CASE ID	USE/CASE/FUNCTIONALITY	PRECONDITIONS/INITIAL STATE	PROCEDURE/STEPS FOR THE USER FOR THIS FUNCTION	INPUTS VALUE	INPUTS	EXPECTED OUTCOME	EXEC DATI
#95_saveToCsv_Valid	saveToCsv	1. ProductSystem must be instantiated 2.Already Process's and Flow's add to the System	1. Call the function			"Arquivo CSV criado com sucesso!"	13/0

1.3- Scope

The testing scope for the `saveToCsv` function involves validating the accurate export of product system data to a CSV file using the `DataExporter` class. It includes verifying successful file creation, checking the content's adherence to the expected structure for flows, processes, and process flows. The scope extends to validating data integrity, handling various product system configurations, and ensuring proper file deletion post-test. Additionally, the function's error handling and performance with different system sizes are assessed, along with compatibility across diverse environments. Integration tests, if applicable, are conducted to confirm seamless interaction with other system components. The primary goal is to ensure the `saveToCsv` function consistently and reliably exports product system data according to specified requirements.