LUND UNIVERSITY

Trisotech Tutorial

Steen, Odd

2023

Link to publication

Citation for published version (APA):
Steen, O. (2023, Nov 29). Trisotech Tutorial.

Total number of authors:
1

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Trisotech Tutorial

INFN50 | Spring ‘24 | Department of Informatics

Odd Steen©

Content

Part I 1
Introduction 1
Useful Resources 1
Step 1: Login to Trisotech Enterprise Suite 2
Part II 4
Test Grading Workflow and Decisioning 4
The Regulation for the Type of Exam We Will Work with Is the Following. 4
The Top-level Workflow 4
Step 1: Model the Workflow 5
A Short Aside on the Difference Between Process and Workflow, and Black-Boxed Pools 6
Back to Business… 7
A Short Aside on Flow Arrows and Message Arrows 8
Back to Business… 8
A Short Aside on Exam Media 8
Back to Business… 8
A Short Aside on Pools and Lanes 11
Back to Business… 12
A Short Aside on Start and End Events in Sub-Processes 13
Back to Business… 14
A Short Aside on Data Store and Data Object 14
Back to Business… 15
Managing Sub-Processes 17
Step 2: Modelling a Sub-Process 17
A Short Aside on Timer Event 18
Back to Business… 19
Part III 30
Modelling the Business Decisions 30
Step 1: Adding a DRD to a Business Rule Task 30
Step 2: Creating Data Types for the Decisions 31
Step 3: Designing the DRD 34
Step 4: Does the Student Have Score for All Tasks? 36
Step 5: Test Your Decision 42
Step 6: Is the Achieved Score for Each Task Greater Than or Equal to the Task Pass Score? 45
Step 7: Test the Decision 46

Step 8: Another Way to Do It 48
Step 8.1: Yet Another Way to Do It..49
Step 9: Are All Test Tasks Done and Passed? 50
Step 10: Test Your Decision 54
Step 11: What Is the Student's Grade Calculated from The Achieved Total Score on the Test?..........55
Step 11.1: Test the DL 57
Step 11.2: Infer a Letter Grade from the Achieved Test Percentage 57
Step 11.3: Test Your Decision 62
Step 12: Awarded grade...66
Step 13: Test Your Decision 68
Step 14: Add Knowledge Sources to the Model 71
Part IV 73
Connecting Decisions and Workflow 73
Step 1: Connect Decision Tasks in BPM with Decisions in DMN 73
Step 2: End 75
References...76

Part I

Introduction

This tutorial will help you start to use the **Trisotech Enterprise Suite** for modelling BPMN and DMN to orchestrate a workflow using automated decision points. The workflow is about a teacher correcting exams and using a grade system to store the results and sending result lists to the secretary for input into Ladok. The grade system uses business rules to generate student exam grades based on assignment grades, test task scores, etc.

Useful Resources

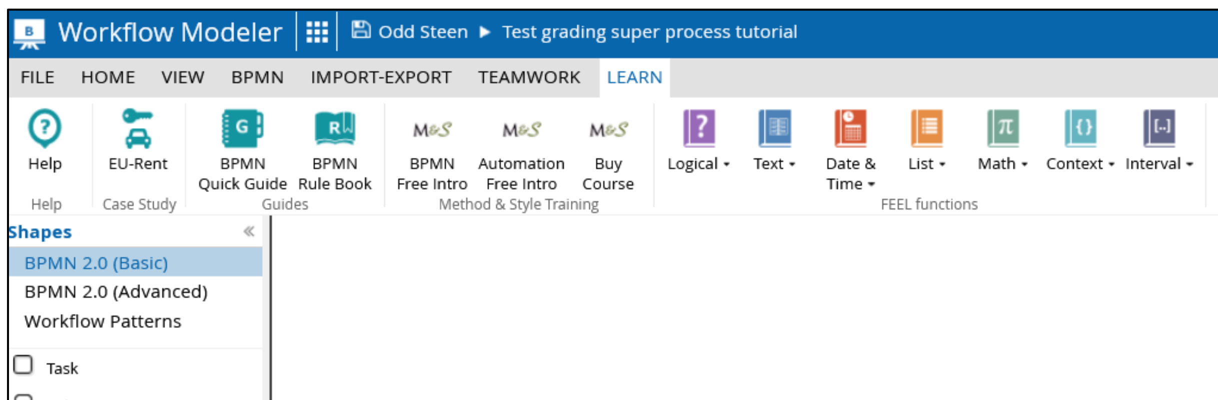
You will find several videos and other material at https://www.trisotech.com/webinars/

There is extensive documentation for the Trisotech Enterprise Suite at: https://lund.trisotech. com/help/

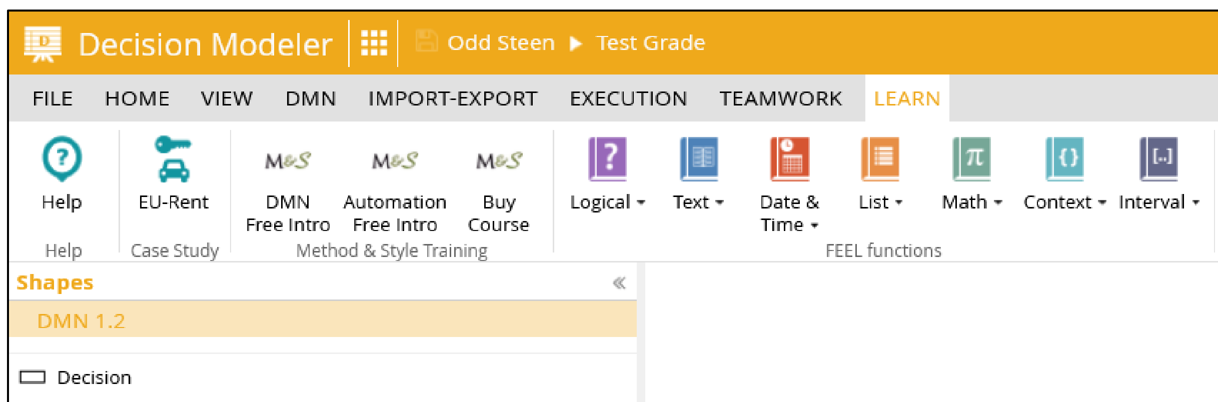
You could start by viewing this recorded webinar: https://www.trisotech.com/how-to-capture-business-decisions-using-dmn/

In the modelling environment you also have several resources under the **LEARN** ribbon.

Workflow



Decision

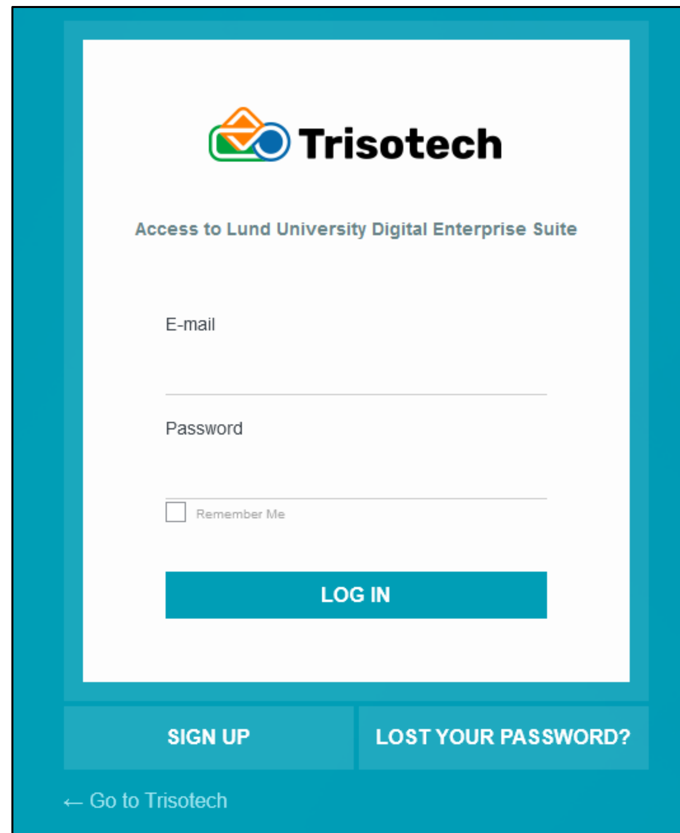


You could start by viewing the "DMN Free Intro" with Bruce Silver. Of some reason it wouldn't start in FF so I used Chrome to watch it.

It might come in handy to have a short list of all FEEL functions in Trisotech: https://www.trisotech.com/feel-functions/

Step 1: Login to Trisotech Enterprise Suite

When you have received the confirmation email, go to the login page as shown in the picture below.



When you have successfully logged in, you should have something like the picture below. Since I have done this earlier, I have already models under my "Places". You should however at least have **EU-Rent** and **Trisotech Examples**.

Lund University Digital Enterpris
Trisotech Help
https://lund.trisotech.com
Sök
Inkorgen (1 992) - odd... Bank och försäkring | ... Översikt E-post–odd.steen@ics... TimeEdit Lunds univer...
Lund University Digital ...
Places
Search Place
Odd Steen
EU-Rent
Trisotech Examples
Odd Steen Odd Steen
Name
Test grading super process

Part II

Test Grading Workflow and Decisioning

In this tutorial you will model one workflow and one decision for managing students' grades on one type of exam. The level of modelling is more for automation than for conceptual or semantic models.

The Regulation for the Type of Exam We Will Work with Is the Following.

A written exam includes one or more tasks where each task has a max score and a pass score. The exam has a max score which is the sum of the tasks' max scores. The grade scale for the test is UA.

- Test max score = sum(task max scores)

The following applies for each task:

- 0 ≤ Task pass score ≤ Task max score
- If Task score < Task pass score then task = fail (U)

The following LUSEM policy applies to calculate the grade of the exam:

- If total score % < 50 then the grade is U
- If total score % [50..100] and all tasks are passed then the grade is calculated according to the policy (53 = grade E, etc.)
- If total score % [50..100] and at least one task is failed then the grade is U

Test Grade Scale UA

A: 85-100%

B: 75-84%

C: 65-74%

D: 55-64%

E: 50-54%

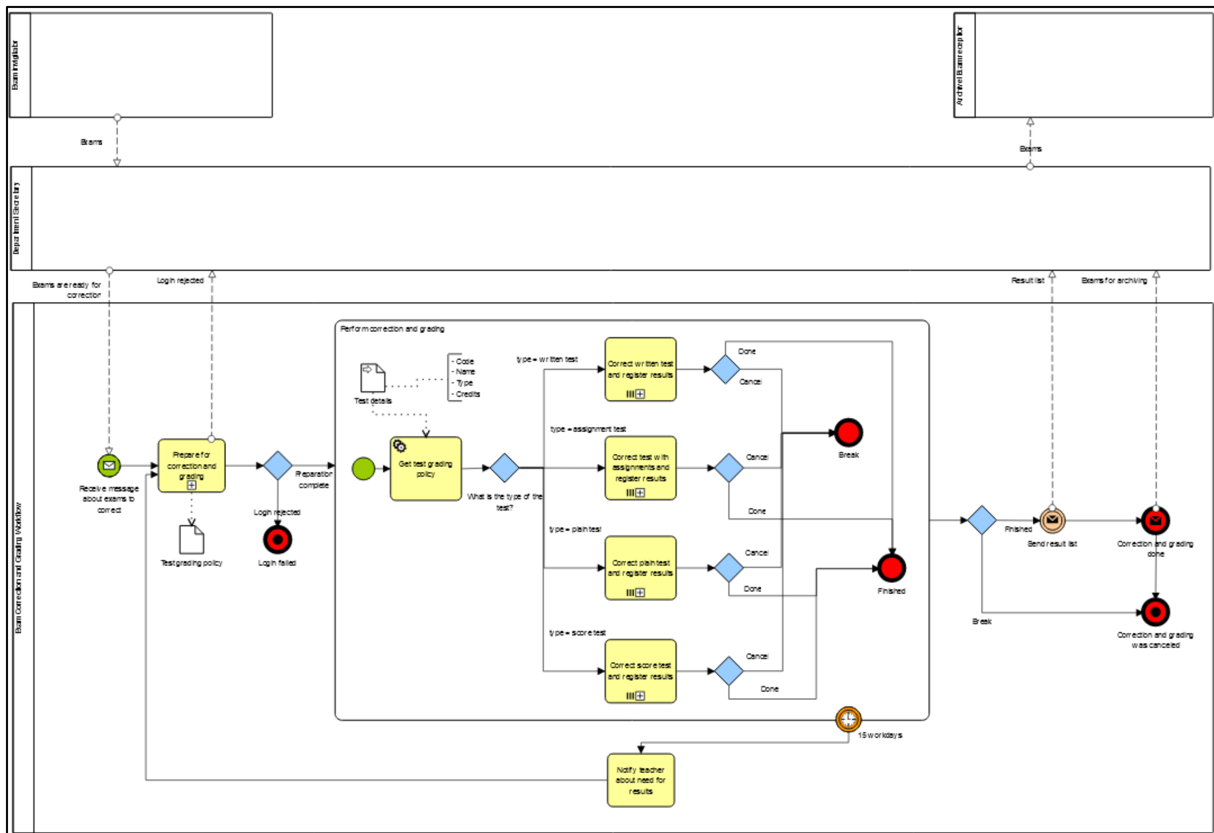
U: 0-49%

The Top-level Workflow

The overarching (top-level) workflow for managing the grading of students is shown below.

The real correction and grading work is not as prescribed and structured as in this tutorial. There is for instance no requirement that the teachers should use a special system to manage grades before they are entered in Ladok. It could be done on paper, using spreadsheet or word processor, or perhaps sometimes even directly into Ladok. It would probably be better to model this using Case Management Model and Notation (CMMN) instead, but the tight integration between BPMN and DMN suggests to use BPMN.

Ladok is a pure record keeping system and not a work support system. Normally, anything below a test, like an assignment or item in an exam, is very hard to handle in Ladok. There is also no automation of grading built on rules in Ladok, meaning e.g., that there is no automation for calculating grade B on a test using a grade model, achieved score, and max test score. So, we could enter 75% of test max score and grade D when it should be B.



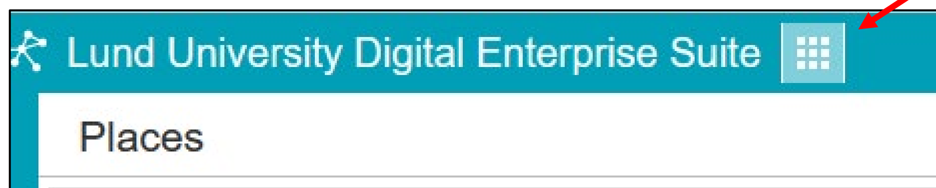
This tutorial therefore assumes that a special system is used and if you are not authorized to use it the process terminates. And that is quite simplified!

Normally, a test should be corrected and graded within 15 workdays.

Step 1: Model the Workflow

First, you need to model this workflow in Trisotech Workflow Modeler.

On the Lund University Digital Enterprise Suite page, you click the on the matrix-like symbol to the right:



This will open the palette of modelling tools, settings, help, etc. Select **Workflow Modeler**.

This should open a new tab for you with a blank canvas to the right and a tool palette to the left. Go to the **File** menu and do **Save As…** and name the diagram "Test grading super process". Select the **Pool** symbol in the tool palette by holding down the left mouse button and drag it the on to the canvas. Double-click in the name field of the pool and name it "Exam Correction and Grading Workflow".

A Short Aside on the Difference Between Process and Workflow, and Black-Boxed Pools

Since we are designing the inner workings of a workflow and in fact are prescribing how this work should be carried out with actors, tasks, events, etc. the pool will subsequentially be filled with such shapes. For this kind of internal and specified workflow the name should reflect that the pool is a process or workflow.

Other pools that are external or black-boxed in relation to this process or workflow should *not* prescribe the inner workings. Such a pool is empty or black-boxed and is always an external actor and the name of the pool should reflect that it is an actor and *not* a workflow.

Hence, workflows that we both can and should detail and prescribe, because they are internal and in focus of our design and development effort, belong in pools named to reflect the work. External workflows that we should not, could not, or are uninterested in detailing and prescribing their inner workings, are always actors and the name should reflect that.

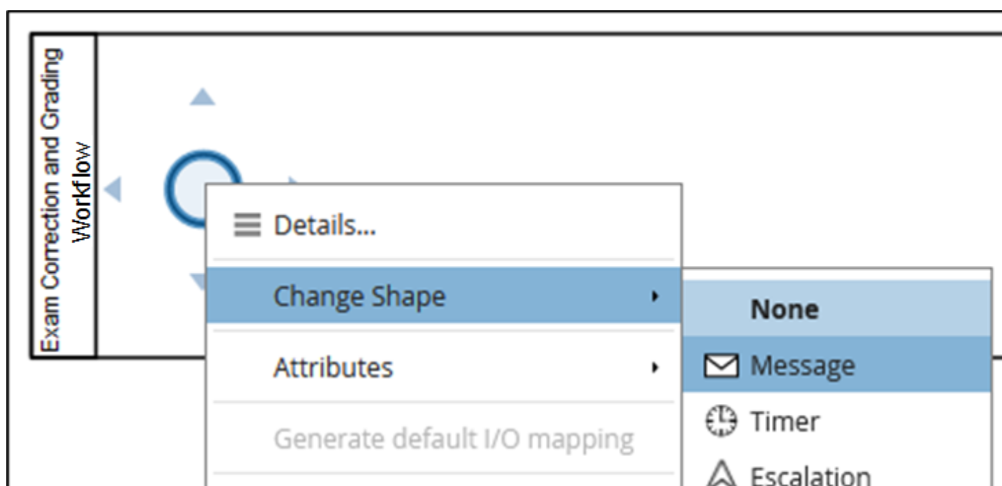
One way to separate a workflow from a process is to use the Zachman Framework for EA (ZEF) where a process, i.e., several work tasks in a directed flow, is in the How column (column two) whereas a workflow, i.e., several work tasks in set order that are carried out by actors, is in the Who column (column four). It could be discussed if a BPMN model that includes actors (Who) is a composite of How and Who, or a primitive model in Who. Anyway, a process pure should not consider or design data (What – column one), actors (Who – column four), geography (Where – column three), or reason (Why – column six). It could consume and produce data (C1) and could consider timing (C5) as at least start and end events, but should thus not consider, i.e., model and specify, who does what,

where does it happen, why does it happen, and how should the persistent data be structured.

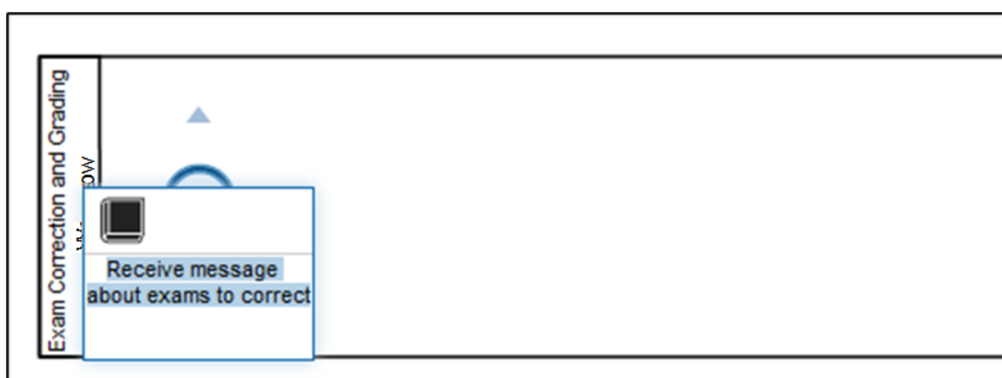
The terminology in BPMN is not that clear. I would still consider a process with actors to be a workflow with lanes for actors and roles. Without any lanes for actors and roles it could be considered a business process. But in that case, I would find it strange with tasks that signifies an actor (Manual, User, Service, etc.) in a process. According to me should a proper *business process* (not workflow) not model and name tasks that are tied to a needed actor, role, or other relevant entity, which could be e.g., a department, business unit, human, or system.

Back to Business...

Select and drag the **Start Event** symbol on to the canvas. Right-click the circle symbol and change the shape into a **Message Start Event**.



Double-click the start symbol and enter "Receive message about exams to correct" as the name.



Since this process starts with a received message, the message must be received from somewhere outside of the process. Hence, the origin of the message is an external actor, i.e., a black-boxed actor in the form of a pool.

I use colours in my diagrams to see more easily which is what. To colour the shapes, you select the artifact in the diagram and use **Fill** on the **HOME** ribbon.

A Short Aside on Flow Arrows and Message Arrows

BPMN is not strict in its syntax and many things can be done in many ways. One thing, however, is quite strict in BPMN: there cannot *be any sequence arrows between pools* since pools communicate through messages. Likewise, there cannot *be any messages flows inside pools* since the sequence is the communication.

Back to Business…

The message received in the workflow must therefore emanate from an external actor that sends the message to our workflow. As you just learned, an external actor is a black-boxed pool. We thus need to add that to our diagram.

A Short Aside on Exam Media

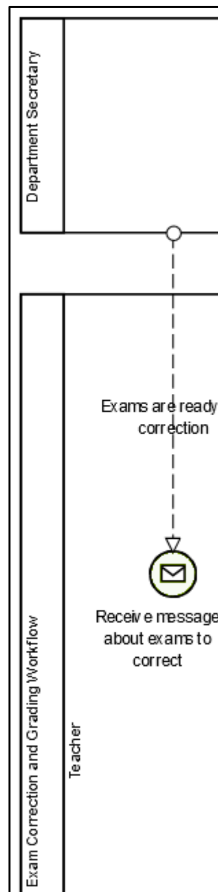
Historically it was the department secretary at the reception desk that was handed the pack of exams by the invigilator. She (during my time here it has always been a 'she') emailed or told the 'Teacher' in question that exams had arrived and were ready to be picked up for correction and grading. When the correction was done the teacher handed the secretary a printed document or a file with the grades so she could input these in the Ladok system. If the exam is paper-based, this will still be the procedure.

However, these days are many of the exams digitalized using exam software such as Inspera. When the exam is closed in Inspera, the teacher is notified that correction may begin, alternatively the teacher checks this him-/herself in the system.

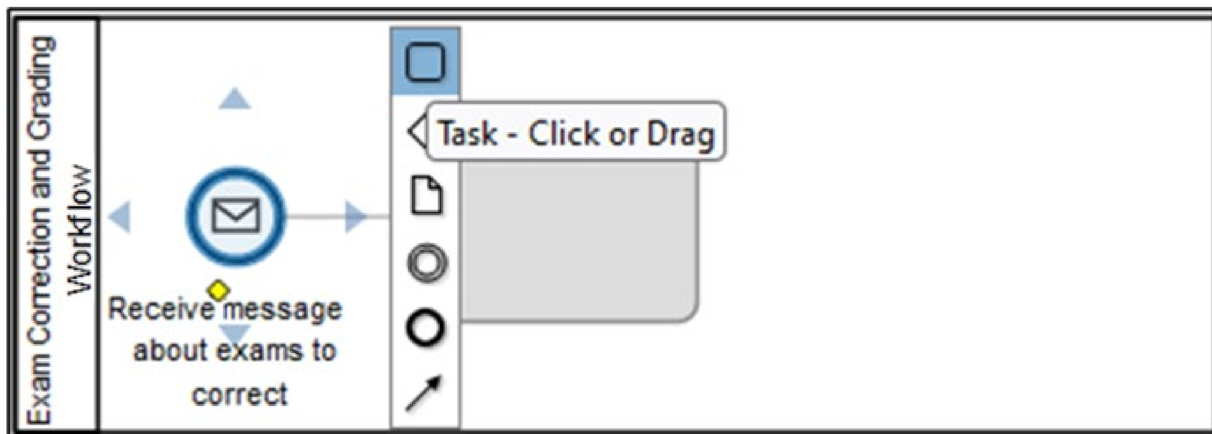
Whether paper-based or digitalized, the correction process or workflow is the same: The teacher corrects and assesses the exams and give them a grade and maybe also a score, and then sends the result list to the secretary for input in Ladok.

Back to Business…

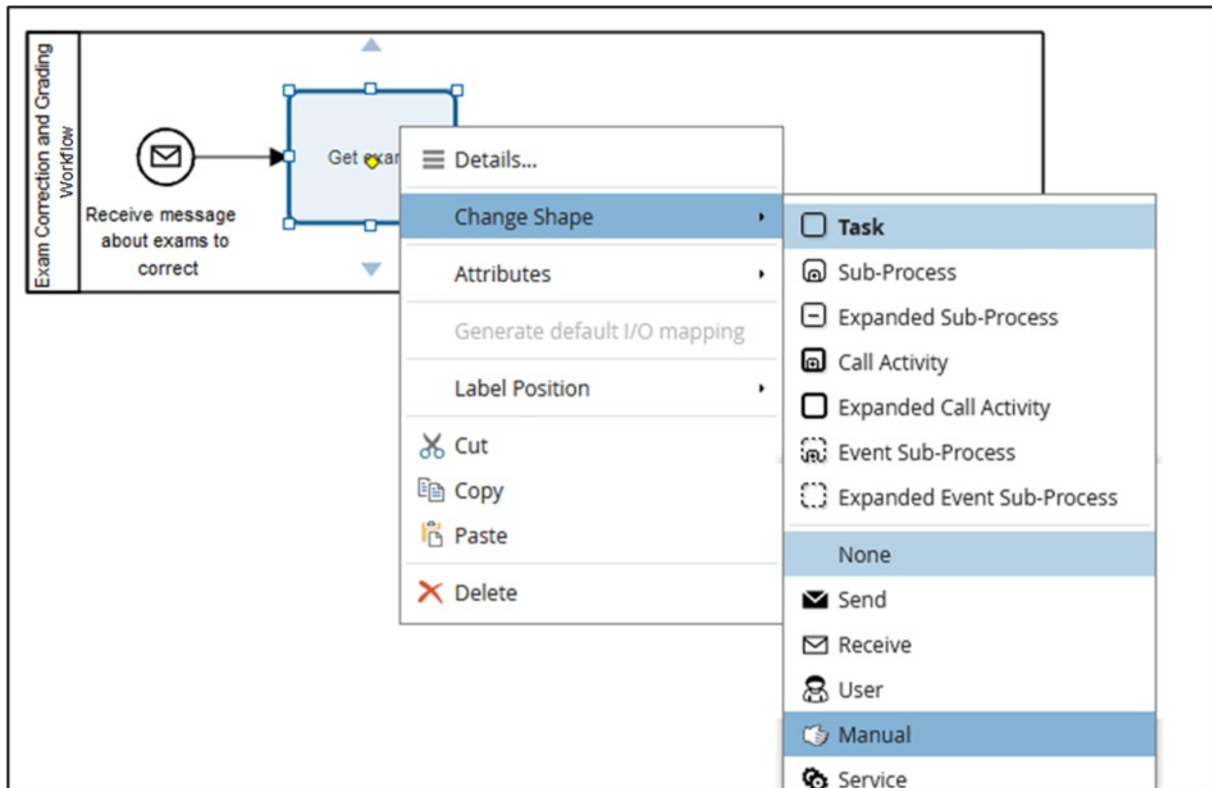
Connect a message arrow from the external actor edge going to the message start event in the workflow.



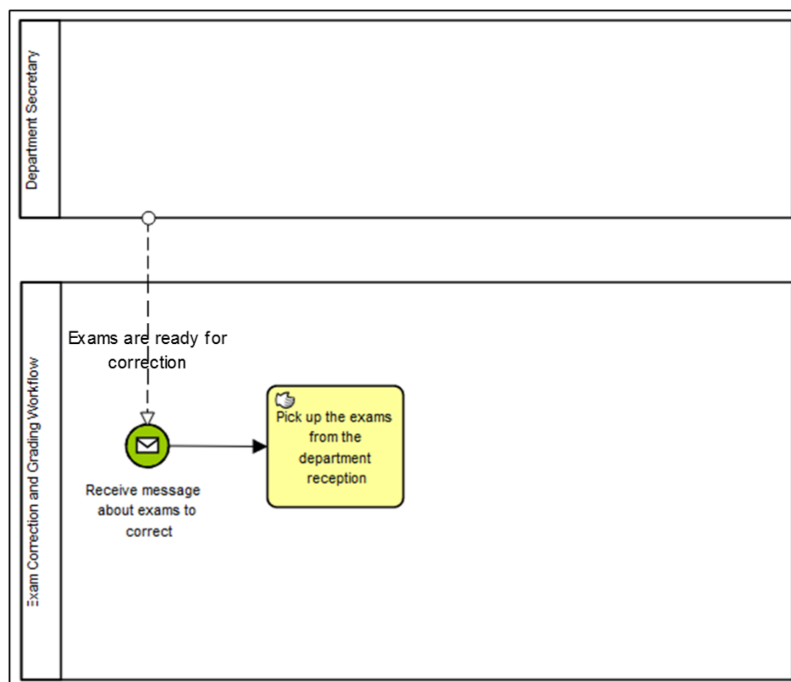
Select the newly created **Start Event** and click on the blue arrow pointing to the right and select the task symbol from the pop-up menu.



Double-click the **Task** and enter “Pick up the exams from the department reception” as name. Change the task shape into a **Manual Task**:



The **Manual Task** is explained as this in the BPMN 2.0.2 spec: "A **Manual Task** is a **Task** that is expected to be performed without the aid of any business process execution engine or any application." (Object Management Group, 2013, p. 161). This means that the 'Teacher' in question strolls over to the reception desk and picks up the pack of exams to correct and grade. It could also be a **User Task** accessing the exam system (e.g., Inspera). If you want to, you can select the shapes and fill them with colour using the **Fill** function in the top menu. The workflow model should now look like this:



One thing that is missing in the workflow is the 'Teacher' actor. That actor is the one picking up the exams from the department reception or opening the Inspera system, correcting and grading the exams, entering the grades into the 'Student Grade System', sending or emailing the list of grades to the secretary for registration in Ladok, and mailing/handing the corrected exams to the exam archive/Reception desk in case of paper-based exams.

A Short Aside on Pools and Lanes

In BPMN an actor is modelled using a lane inside a pool on parent level (however, the BPMN spec. allows you to draw a lane without a pool on any level) and without any pool on child level. A white-box pool contains the end-to-end process/workflow and is named that way. A lane inside the pool is an actor or role (maybe a business unit, department, human, or system, but not an individual!) performing tasks in the process flow/workflow.

A black-box pool represents an external actor and how they should work is not our task to specify or is out of scope in the current design situation. They should always be named according to the actor/role and should never have any shapes inside.

Since actors/roles represent details of the work, it might be better to use them in child (sub-) levels and only have the pool at the parent (super-levels). Since a child process is part of the parent pool it should not have any pool of its own. If the parent process pool has lanes, then the child process may only expand the lane it resides in. It may not suddenly cross lanes or add lanes that are not considered subsumed under the lane of the parent process pool.

Since the hierarchy of parent and child processes/workflows represent drill down and up in the models and the design problem, lanes may be added to the child level to specify more detail than on the parent level.

However, this must be consistent between parent and child levels. If the parent level for instance has a 'Teacher' lane with a sub-process in it, that sub-process may not add other lanes that are not specialisations of 'Teacher'. Proper lanes could be 'Bachelor Teacher' and 'Master Teacher' or 'Course Director', but probably not 'Secretary', unless you consider a secretary to be a kind of 'Teacher' (which I do not).

Unless necessary and a good design decision, I would not have any lanes at the top level. In addition, it might be complicated to have a mix of atomic tasks and sub-processes at the top level and use lanes, at least if the tasks are inconsistent from an actor perspective. If they are in the same lane, the sub-process may not add lanes that are not consistent with the atomic task at the parent level.

The question is also whether we should have pools in child diagrams to, for instance, model sub-processes. It would seem natural to make a pool in a sub-process and name that pool after the sub-process. However, a pool signifies an end-to-end process and since a sub-process is part of that *it cannot itself be an end-to-end process* too – you should thus *not have pools in pools*. There are other reasons too that you can read about in e.g., Silver (2011).

Back to Business…

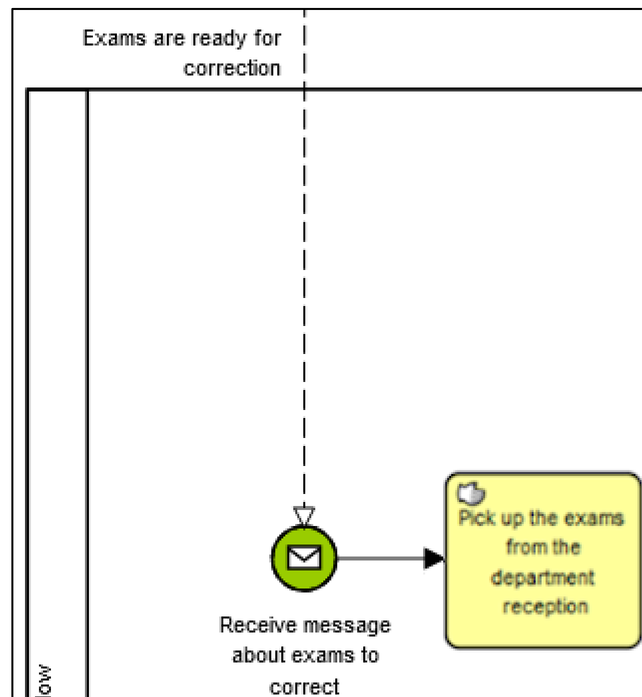
It seems better to avoid lanes on the top-level diagram, even if it is not forbidden, and have the same granularity for all the tasks of the top-level process/workflow. Actors and roles may be specified in sub-processes using lanes.

However, three external actors could be added to the diagram as black-boxed pools: The 'Department secretary', the 'Invigilator' that delivers the pack of exams to the department secretary in case of paper-based exams, and the 'Exam Reception' that stores the corrected paper-based exams and hand them to the students. They communicate with each other and the end-to-end process using message flows.

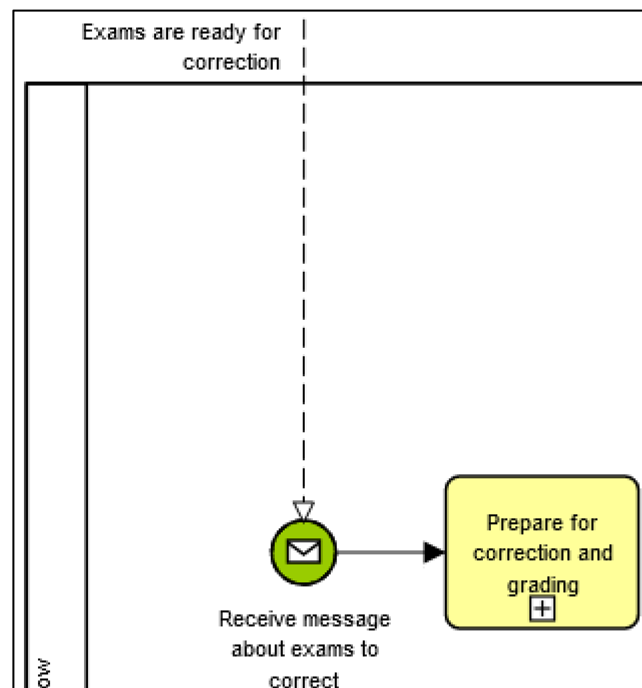
To continue modelling the top-level process we will add tasks for the 'Teacher': picking up the exams from the department deception desk, logging in to the 'Student Grade System' which handles grades on a granular level under Ladok and uses business rules to calculate grades, correcting and grading the exams, using the 'Student Grade System' to store and render grades, producing a list of grades and sending it to the secretary for registration in Ladok, and finally mailing/handing the exams to the exam desk/archive, unless it is an Inspera exam..

To avoid the situation above of atomic and sub-processes at the same level, is it better to treat all tasks that involves some actor (human or system) at the top-level as sub-processes. In this case, there are two types of sub-processes: 'Preparation and Correction', and 'Grading'. But we need a task after the 'Preparation' sub-process that checks what type of test is handled, since that controls the procedure and rules of the correction. This task is in this case a **Script Task** (could also be a **Service Task**) carried out by the 'Student Grade System'.

We had reached the model in the picture below but understand, based on the discussion above, that instead of atomic tasks followed by sub-processes, we should only have tasks of the same type on the top-level diagram without lanes.



Hence, we don't continue to model like this and instead move the 'Pick up…' task to a sub-process. Therefore, we add a task after the 'Receive message…' **Message Start Event**, name it "Prepare for correction and grading", and change it into a sub-process.



Click the "+" symbol of the sub-process and model the sub-process like this:

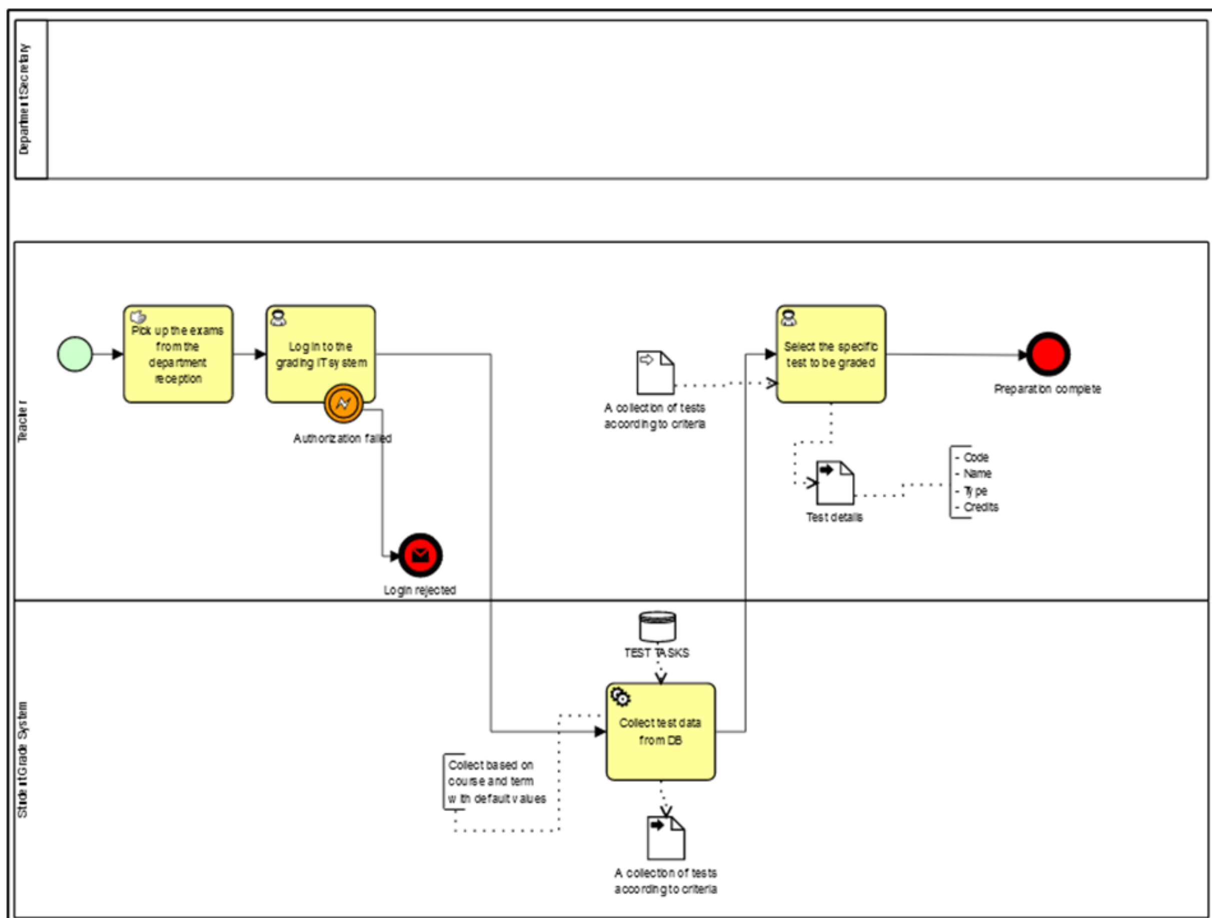
A Short Aside on Start and End Events in Sub-Processes

Since a sub-process (child level) is part of an end-to-end process on the parent level, it is always started by the sequence flow on the parent level going to the sub-process. Hence, a sub-process may not have any qualified start and end events, so these are always of

None type. A sub-process should also only have one start event but may of course have several end events. According to Silver (2011), if there are at least two end states (events) in the sub-process, the parent level process must consume the end states with a gateway branching based on the end state conditions. By the way – all events need to be named properly to catch their business meaning.

Back to Business…

In the sub-process above I added an **Intermediate Event** to the edge of the 'Log in…' tasks to catch an exception generated by the system when a user tries to login but is not authorized to use the system. It is on the edge of the task since we must not wait until the task is completed but need to catch the exception during the execution of the task. If the exception is caught, it leads the flow to a **Message End Event** named "Login rejected", and we need to process that end state on the parent level. We of course also need to, based on the discussion above, process the 'Preparation complete' **None End Event.**



A Short Aside on Data Store and Data Object

The disk pack shape in the diagram above denotes a record in a persistent data store that the workflow can perform CRUD (Create, Read, Update, Delete) operations on (depending on access rights). This is probably a record in a centralized database that is not tailored to the workflow, but to many different processes and workflows. It could be a datastore for an ERP or a CRM. The datastore is never designed in BPMN. The architecture of the data belongs to the What (column one) in ZEF and is often designed in EER or UML using

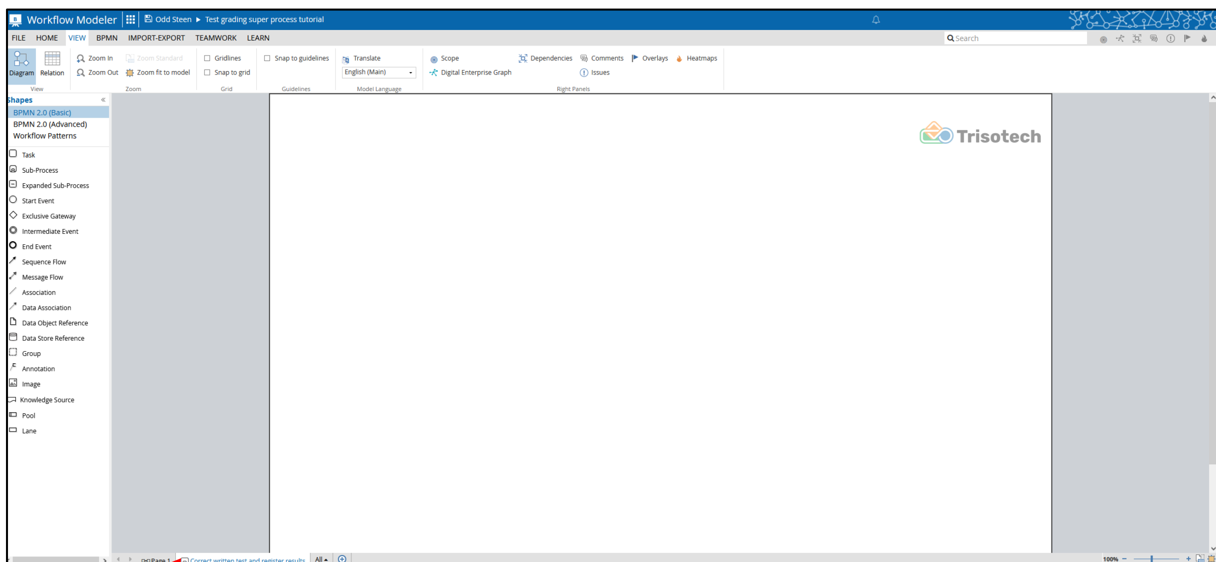
a data design tool such as DB-Main or semantic modelling or ORM modelling in a tool like System Architect. It is also possible to devise such a model in Semantics of Business Vocabulary and Rules (SBVR) with a tool that supports that.

The dog-eared paper shape denotes a data object. Data objects are never persistent and only live during the duration of the session or workflow instance. As soon as the session or instance reaches its end, the data objects are killed and ready for garbage collection. If data of a data object should be persisted, a service or script task in the workflow is needed to update one or several datastore record/s.

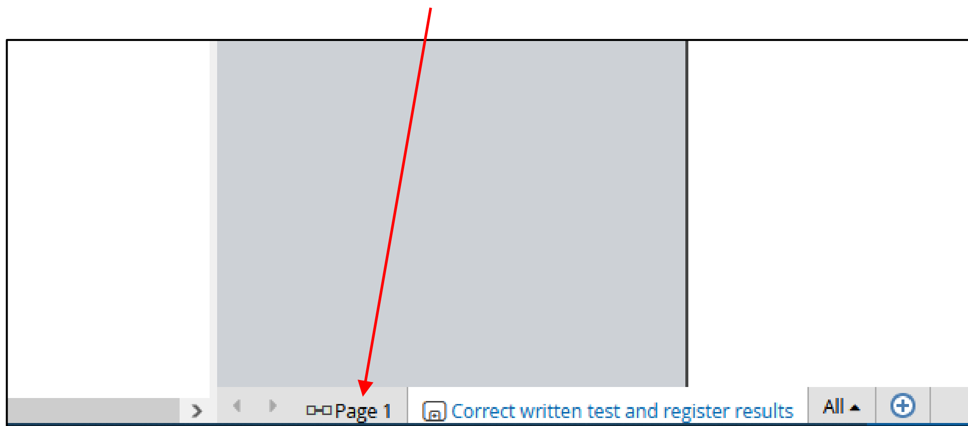
Data objects do not need to be in a 1:1 relationship with entities in a data model. They are more likely to only hold data that is needed by the workflow since holding and moving redundant data is poor from a data quality perspective and should be avoided. It could also be that a data object is the combination of several data entities and sources through e.g., views with joins and unions. You will discover further on that the workflow is responsible for providing the decision tasks with the needed data. It could therefore be an idea to model the required data in a model or to detail and explain the data objects in a dictionary.

Back to Business...

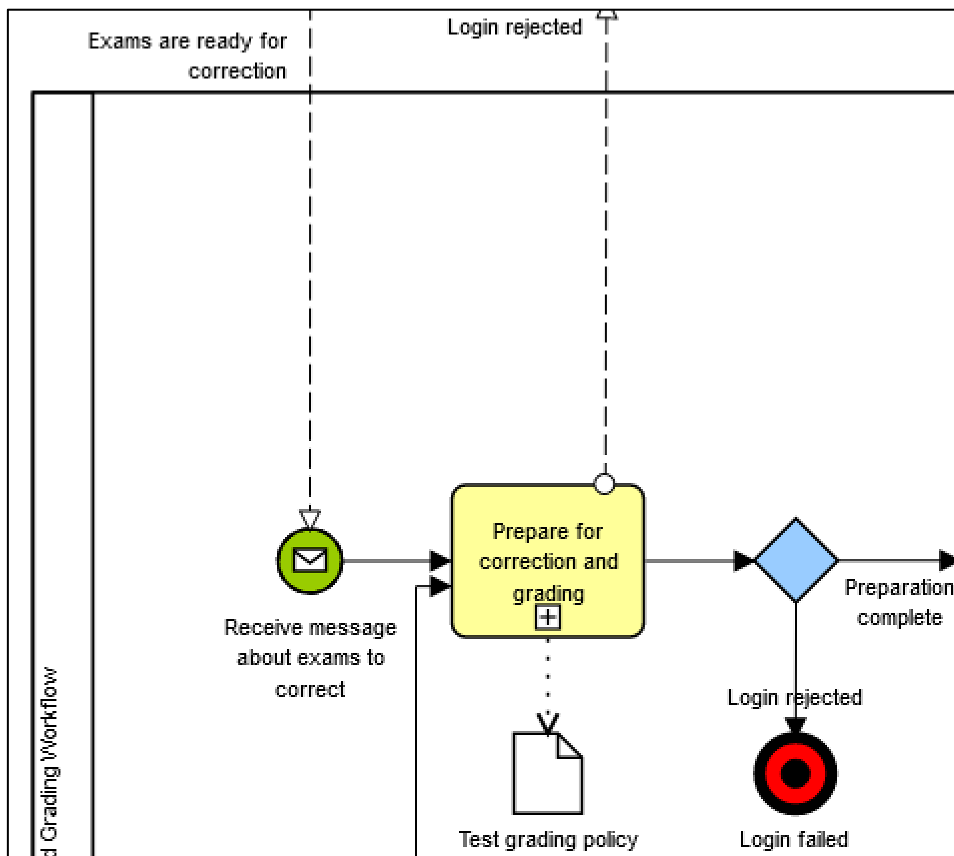
Go back to the super process by moving up one level to Page 1.



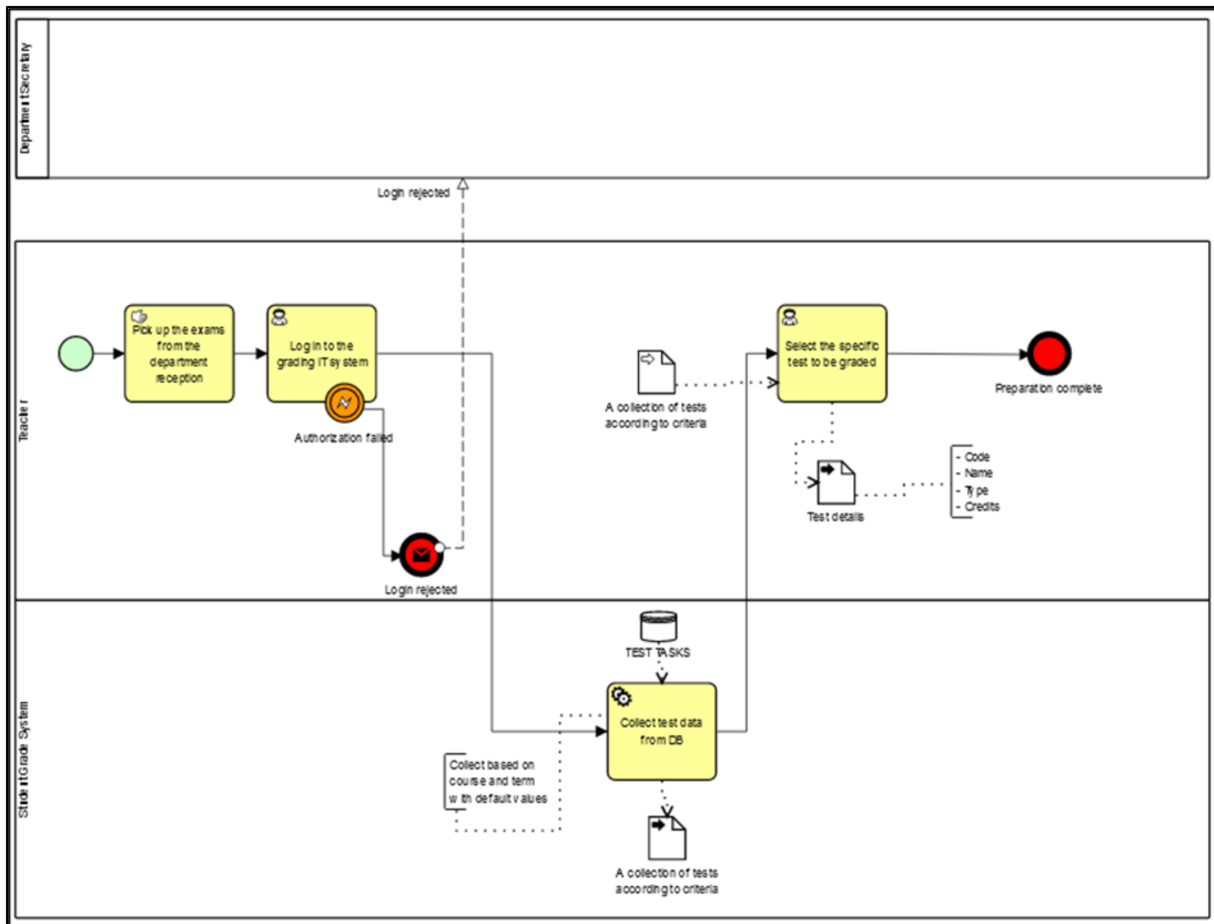
To move up one level, click the link (Page 1 in this case) in the bottom-left of the screen:



In the super process, continue modelling until you have the following:



According to Bruce Silver Method&Style, message flows must be consistent between parent and child levels. Hence, the message from the collapsed sub-process of 'Prepare for correction and grading' in the parent level must be replicated in the child level. We therefore need to add that lane to the sub-process and add a message out from the sub-process to the 'Department Secretary' lane.



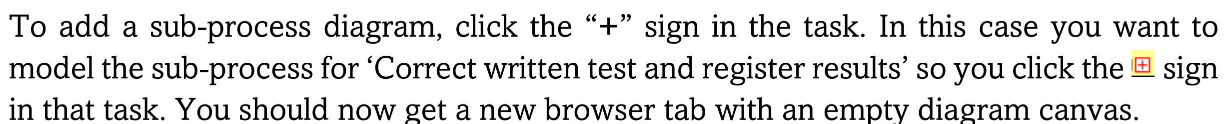
The next step is to model the needed sub-processes.

Managing Sub-Processes

There are four different types of tests with different rules for how a grade is calculated based on a student's performance and different data to enter – score, grade, etc. To manage this, the flow needs to branch based on the type of the test.

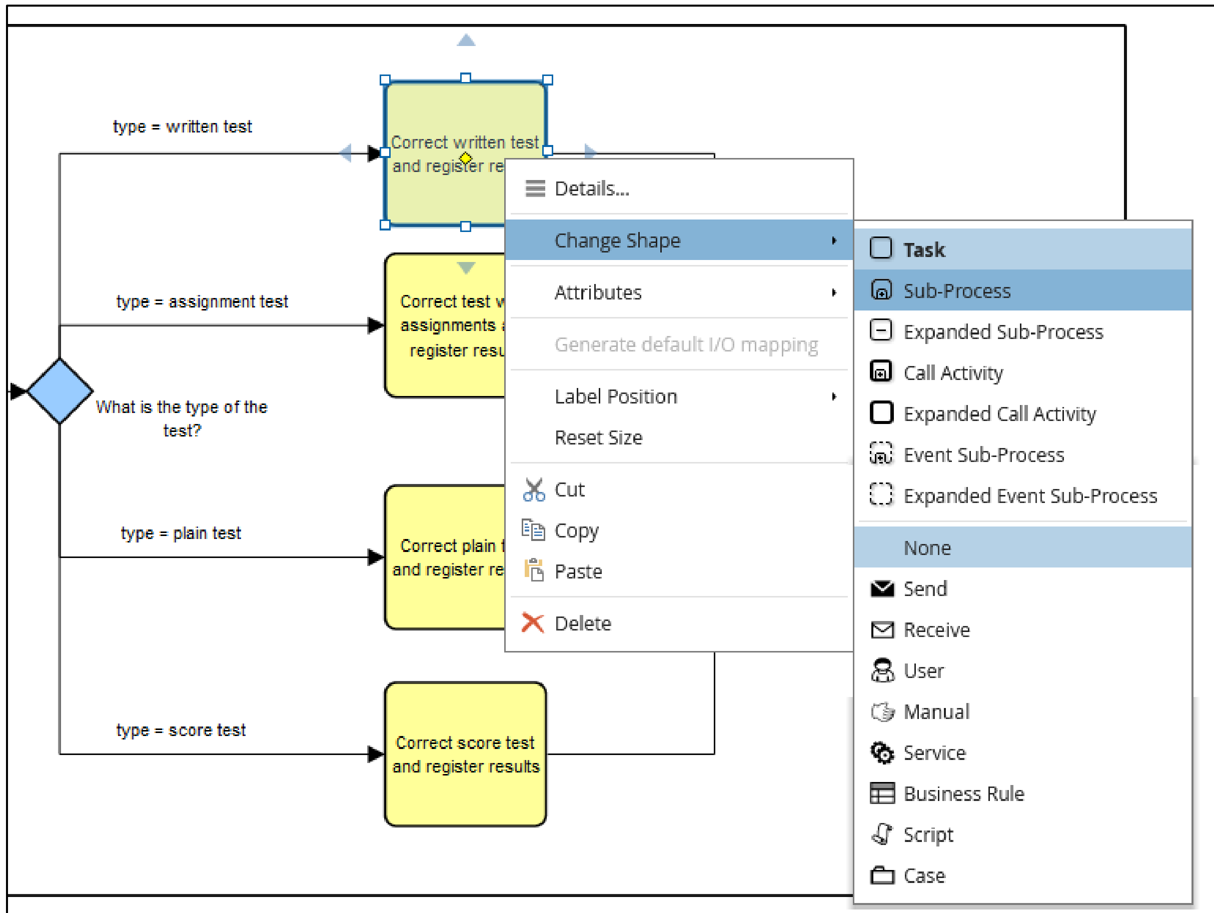
Each of the four sequences after the 'What is the type of the test?' gateway is too complex to show in the same diagram without making it cluttered and hard to read. Therefore, we will introduce four sub-processes but for the tutorial you only need to develop one of them.

Step 2: Modelling a Sub-Process

To add a sub-process diagram, click the "+" sign in the task. In this case you want to model the sub-process for 'Correct written test and register results' so you click the  sign in that task. You should now get a new browser tab with an empty diagram canvas.

Select the 'What is the type of the test?' gateway in the model and add a new task. Name it "Correct written test and register results". Iterate this three times and name the tasks respectively "Correct test with assignments and register results", "Correct plain test and register results", and "Correct score test and register results". Order the layout of the model so it looks like the model in the picture below.

Now you need to turn the new tasks into collapsed sub-processes. You do that by right clicking each task and select **Sub-Process** in the contextual menu.

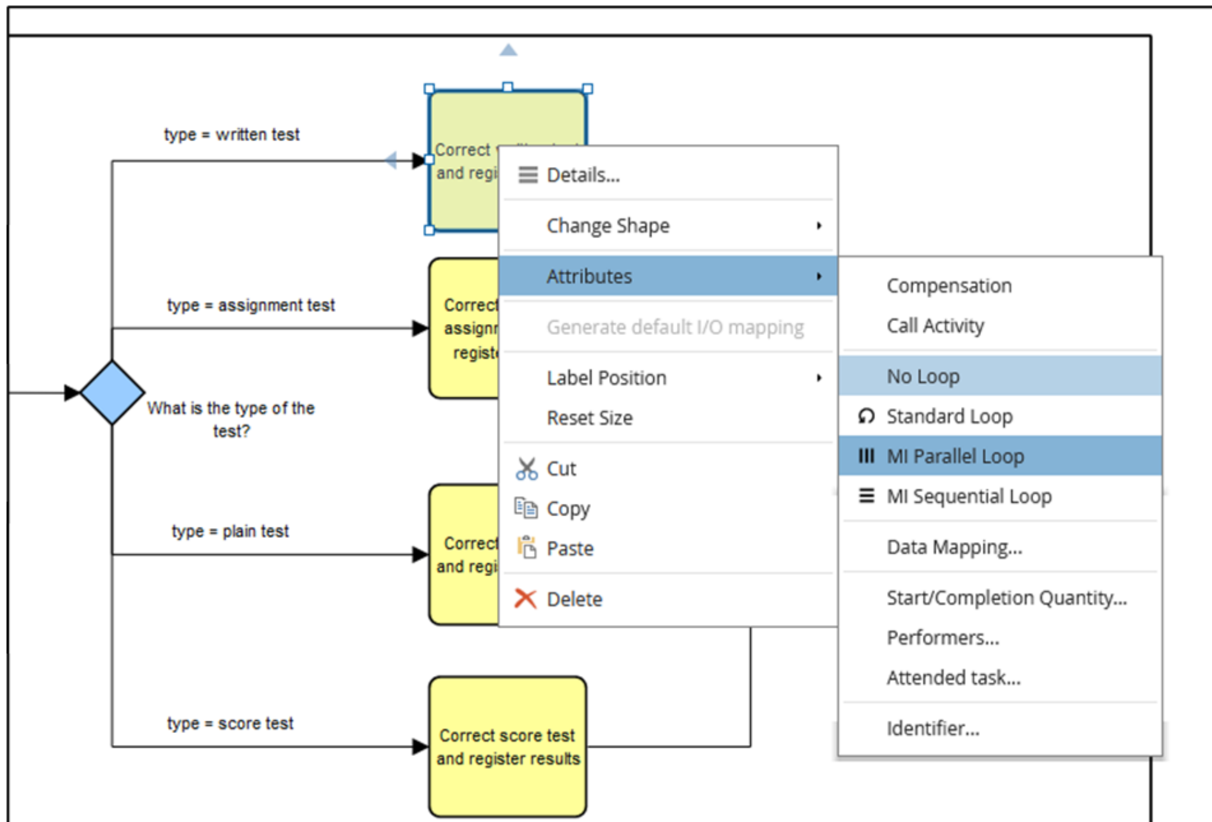


After this you must change the sub-processes into parallel ones.

Since the policy says that a test should normally be corrected and graded within 15 workdays, we need to show that somehow in the model. You do that by dragging an **Intermediate Event** shape from the palette and attach that to the edge of the first collapsed sub-process. If it can be attached to the edge of the sub-process the edge will turn thick green. When done change the colour to orange and add “15 workdays” as name/label.

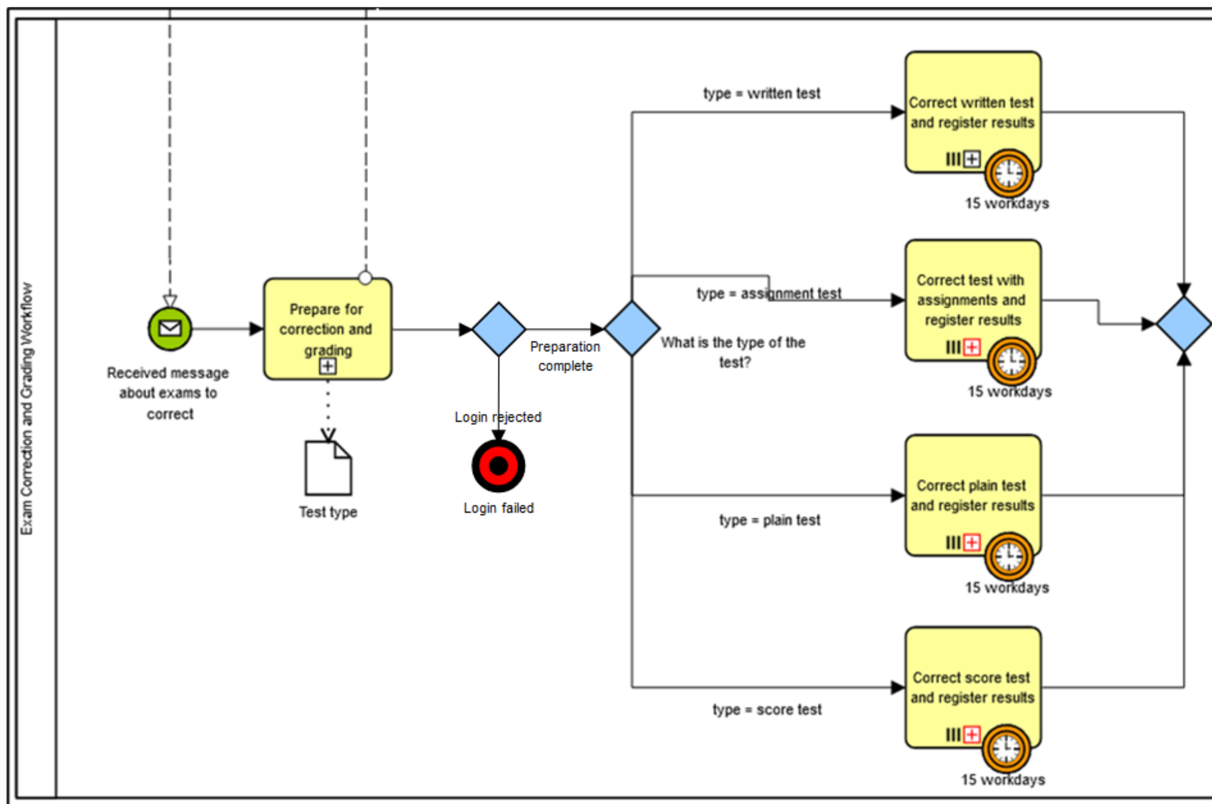
A Short Aside on Timer Event

The **Timer Event** in BPMN is not used to indicate or record how long a task does or will take. You can almost think of it as some kind of exception or rather deadline, since it prescribes what will happen when the time limit is up. Hence, you need to remember it is a type of event and that an *event takes place when the condition is met.*

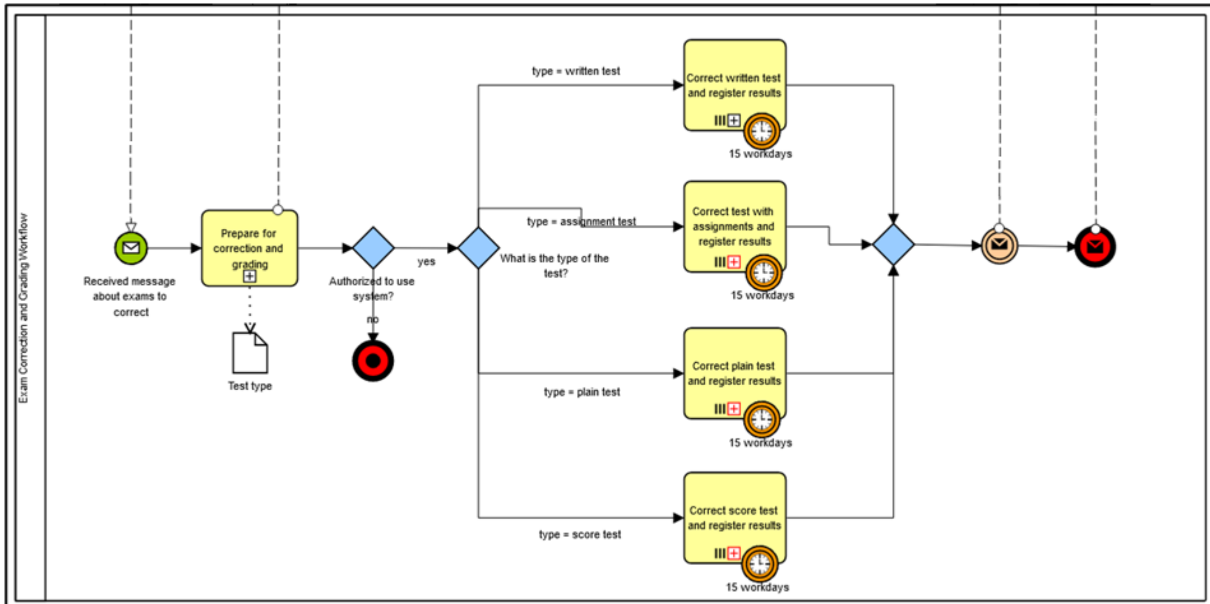


Back to Business…

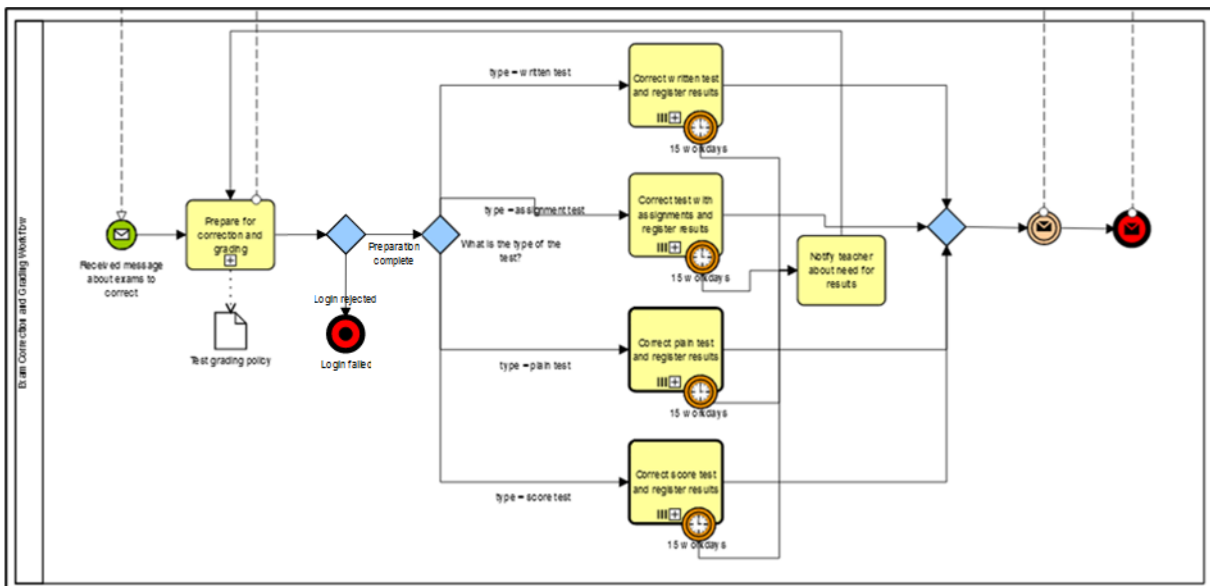
Do this for all four tasks. You should then have the following model:



Continue adding symbols to the model until it looks like this:



Since we have an intermediate **Timer Event** on each correction and grading sub-process, we need to take care of the flows out of them. The timer event means that the sub-process starts and continues until done, unless the 15 workdays timer is triggered: That is, a timer is started when the sequence flow reaches the sub-process and if 15 workdays are used before the flow continues out of the sub-process, the timer will "ring" and the flow will be directed out of the timer event and not the sub-process itself. Hence, we need to add a flow out from the timers to a task that takes care of the activity flowing from the timer event. So, let us add that task and simply let it notify the teacher about the delay and flow back to the 'Prepare…' sub-process:

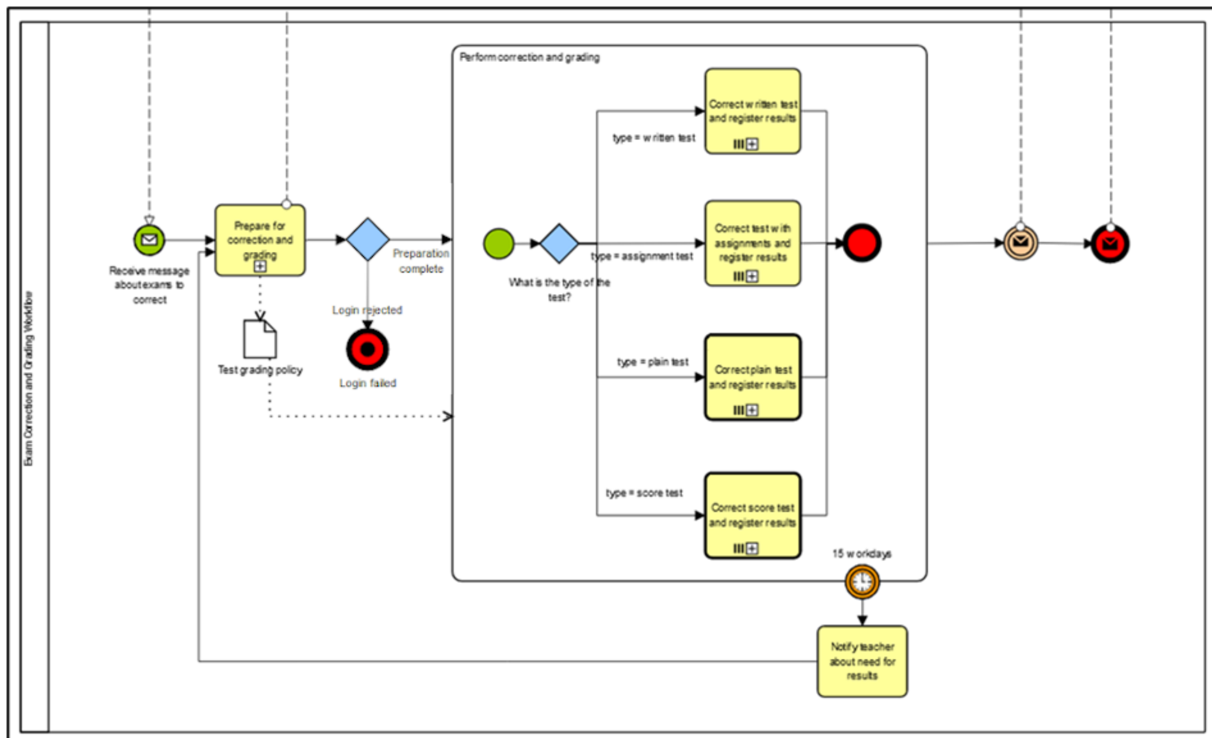


That does not look good! Since the timer event is the exact same for the four sub-processes, we should not add a timer event to each sub-process and have four sequence flows to one task to handle the event outflow. Instead, we will put the all the correction and grading work into a new, expanded sub-process and attach the timer event on its edge.

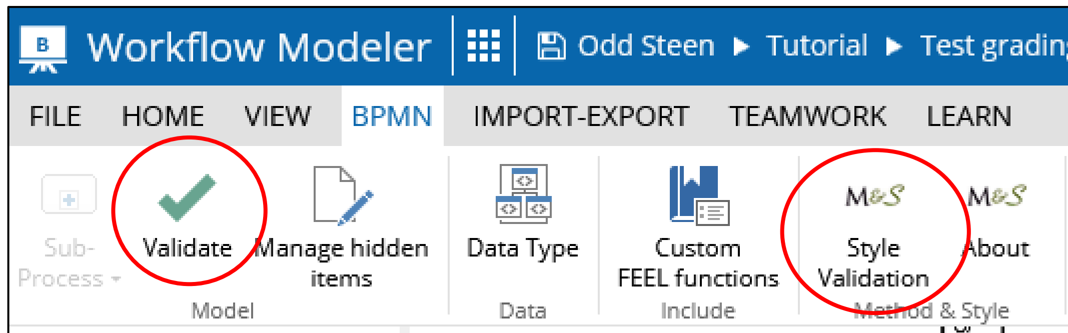
Hence, select the **Expanded Sub-Process** in the palette and place it in the pool. Name the new sub-process “Perform correction and grading”. Select everything between the second XOR gateway and the intermediary message event and drag it into the new sub-process. Remove all the timer events from the edges of the sub-processes. Add a **None Start Event** in the left of the sub-process and add a sequence flow to the ‘What is the type of the test?’ gateway. Add a **None End Event** in the right of the sub-process and connect all the correction sub-processes with sequence flows to it. Remember that a sub-process always starts with a **None Start Event** and ends with at least one **None End Event**, and that the sequence arrows of the parent process always connect to the edges of the sub-process.

Select an **Intermediary Event** from the palette and add it to the edge of the expanded sub-process. Change it into a **Timer Event** and name it ”15 workdays”. Add a new task from the timer event and name it “Notify teacher about need for results”. Finally, add a sequence flow from this task to the left edge of the ‘Prepare for correction and grading’ task.

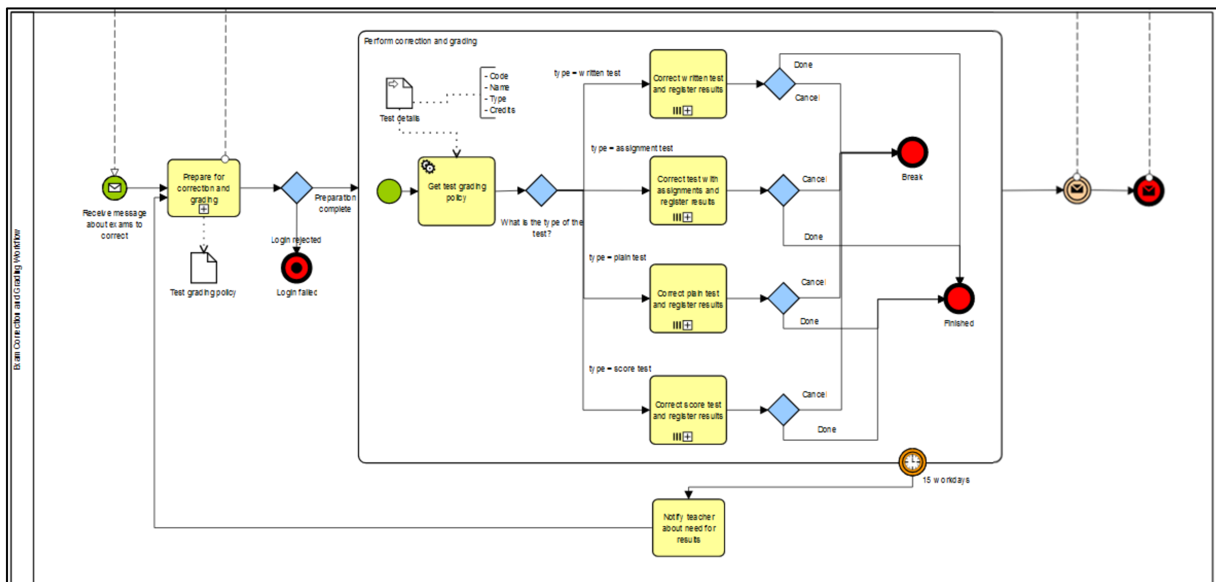
Now, the workflow says that the ‘Perform correction and grading’ sub-process has a timer alarm that goes off after 15 workdays if the sub-process is not already completed. The alarm starts a task to inform the teacher that the normal correction time is out and the results are requested. The workflow then backtracks to the first task in the workflow and the work could proceed or commence, depending on the actual state of the correction and grading work.



We use the **Validate** and **M&S Style Validation** under the **BPMN** ribbon to check the model for compliance with the BPMN specification rules and the rules of Method&Style.

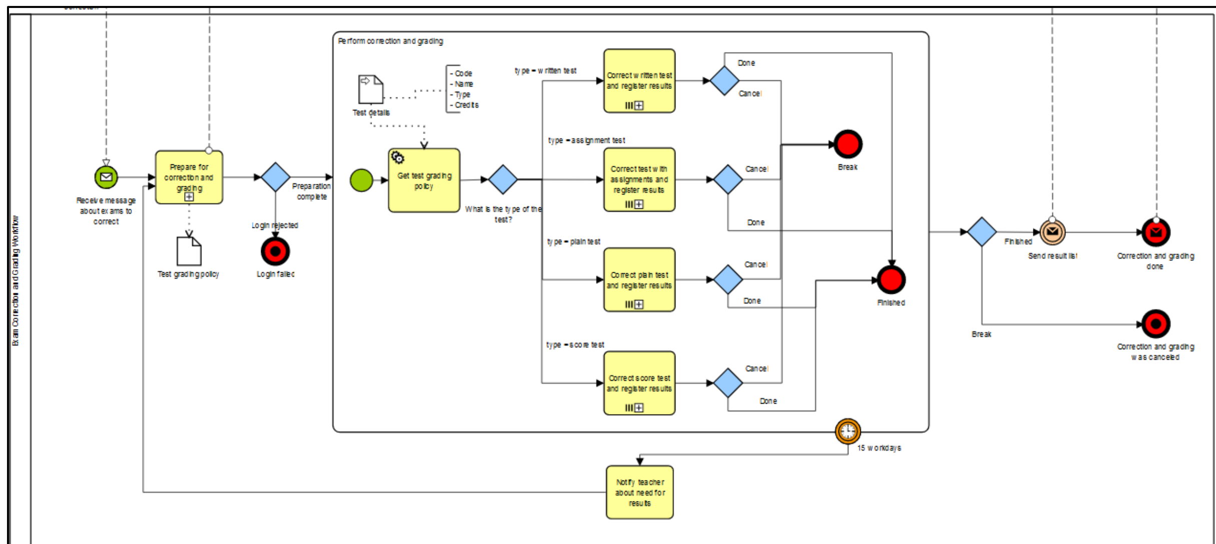


We will get several warnings about modelling errors that we need to fix. Since the exam correction sub-processes all have more than one end state, the **Style Validation** will warn that the parent process 'Perform correction and grading' lacks XOR gateways to consume these end states. We need to fix that by adding XOR gateways after each sub-process that leads to two end states: 'Break' and 'Finished'.



A new check using **M&S Style Validation** reveals that we have the same problem with the super process since the sub-process of 'Perform correction and grading' now have two end states. We also learn that the intermediate message event and the end message event both lack labels.

We need to fix that too by adding an XOR gateway after the 'Perform correction and grading' expanded sub-process with tow sequence flows out of it, leading to two end states of 'Correction and grading done' and 'Correction and grading cancelled'. And we add a label to the intermediate message event:

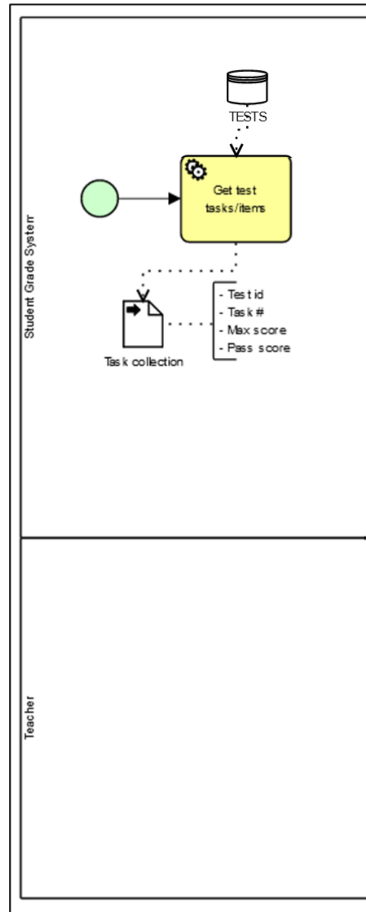


Now we need to model the sub-processes, but in this tutorial, it suffices to model the 'Correct written test and register the results' sub-process. This sub-process will eventually use several **Business Rule Tasks** for automatic and rule-based grade calculations based on each student's achievement and details of the test.

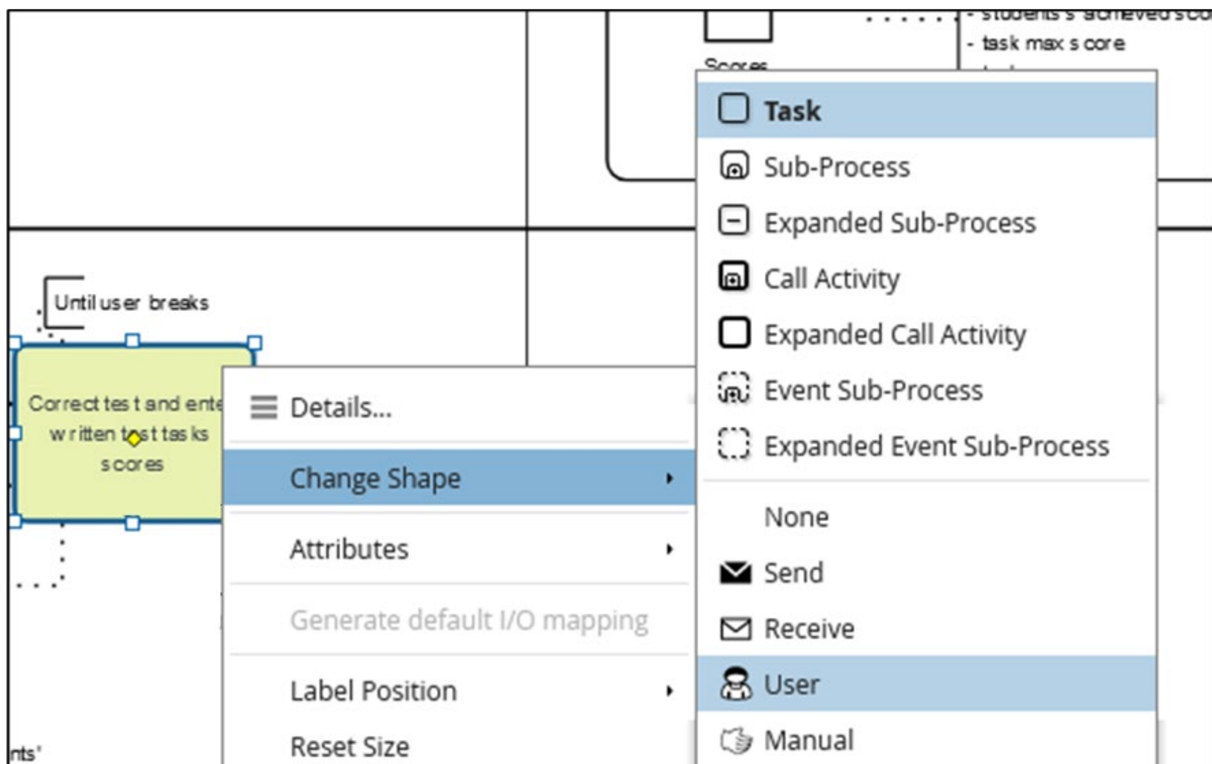
Click on the "+" symbol in the 'Correct written test and register results' to open a new empty diagram to model the sub-process.

Begin by adding two lanes (no pool!) and name them "Student Grade System" and "Teacher" respectively. Drag a **Start Event** and place it to the left in the 'Student Grade System' lane.

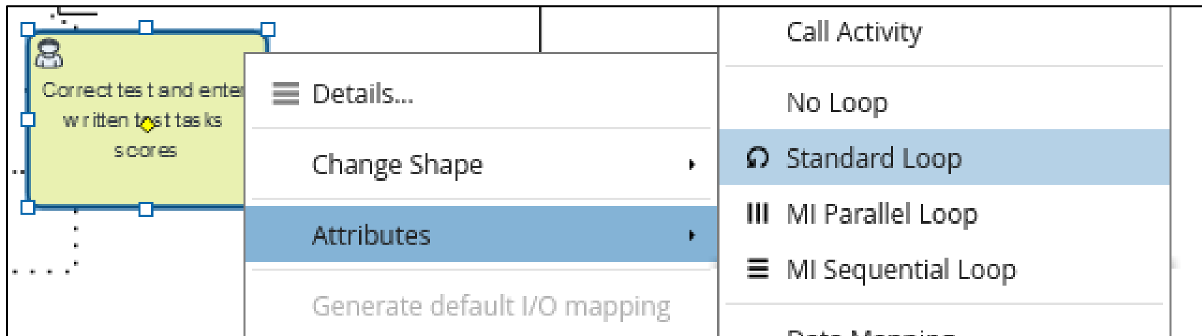
The first work task that needs to be performed is to get the tasks or items in the test, i.e., what are the questions of the test? We need to know this to decide whether a particular student has answered or done all tasks in the test and if the student has passed all mandatory tasks. We also need to know the max score of each task since the sum of them entails the max score of the whole test. That max score (normally 100) is used to derive the grade given the percentage of the max score as in the table on page 5: 75 out of 100 means 75% and a B while 75 out of 120 means 62.5% and a D.



Add a new task in the 'Teacher' lane and name it "Correct test and enter written test tasks scores". Change it into a **User Task**:

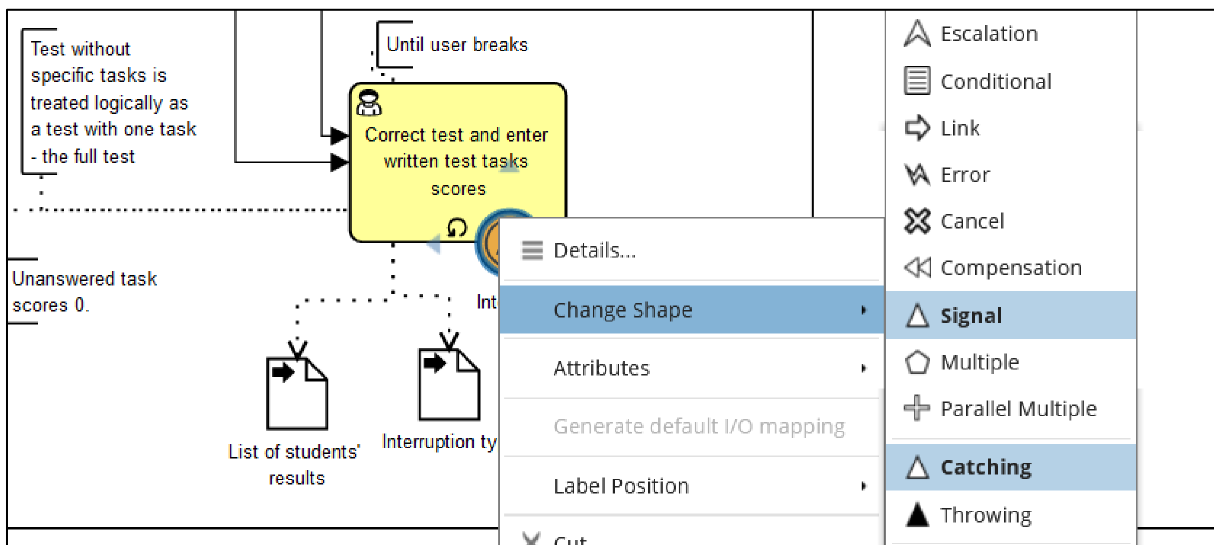


And then into a looping task:

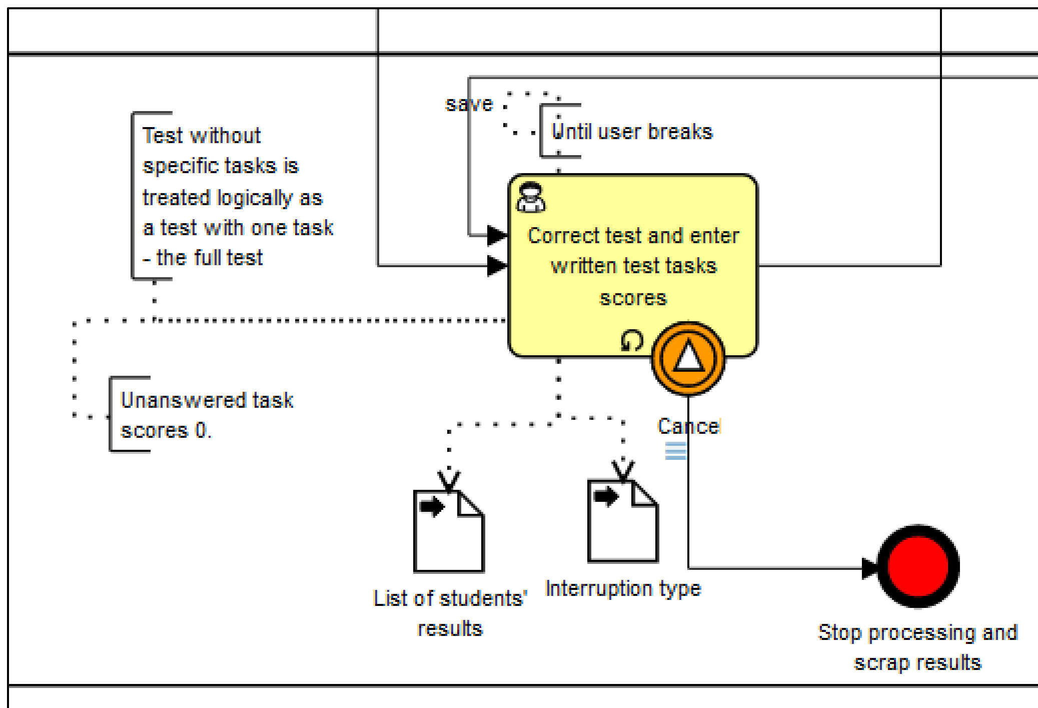


The idea is that the user of the grading system, i.e., a 'Teacher', corrects each test in the manner he or she wants to (most often Q1 for each student, then Q2…, etc.) On a form in the system the 'Teacher' registers each student's score for each task. Whenever he or she wants to 'Save', 'Save and close' (i.e., 'Done'), or 'Cancel', the person clicks on the respective button in the dialog. The system then applies all the necessary controls, calculates the students' test grades based on the applicable business rules, and saves the data to the database. If the 'Teacher' clicked 'Save' he or she just continues with the form open. If the 'Teacher' clicked 'Done' the system runs the same procedures as for 'Save' but closes the form when done. If the 'Teacher' clicked 'Cancel' the form is closed without processing any data (you should have a warning here about unsaved work and if you want to save it first, which, if you chose to save first, tells the system to do the same as for 'Done'. We skip that here though.)

Since this click of a button could happen anytime the form is up and in focus, it is not possible to wait until the 'Correct…' task is done before handling the UI event. Hence, we cannot have a sequence arrow to some task that takes care of the UI event, since that would mean that the full task must be done before we reach that event. Instead, we need to throw (or maybe catch) that event within the ongoing task whenever it happens. To do so we need to add an event to the edge of the task by dragging an event shape and place it on the task shape's edge. The event will be an intermediate **Catching Signal Event**:



This will have a sequence flow to an end event:



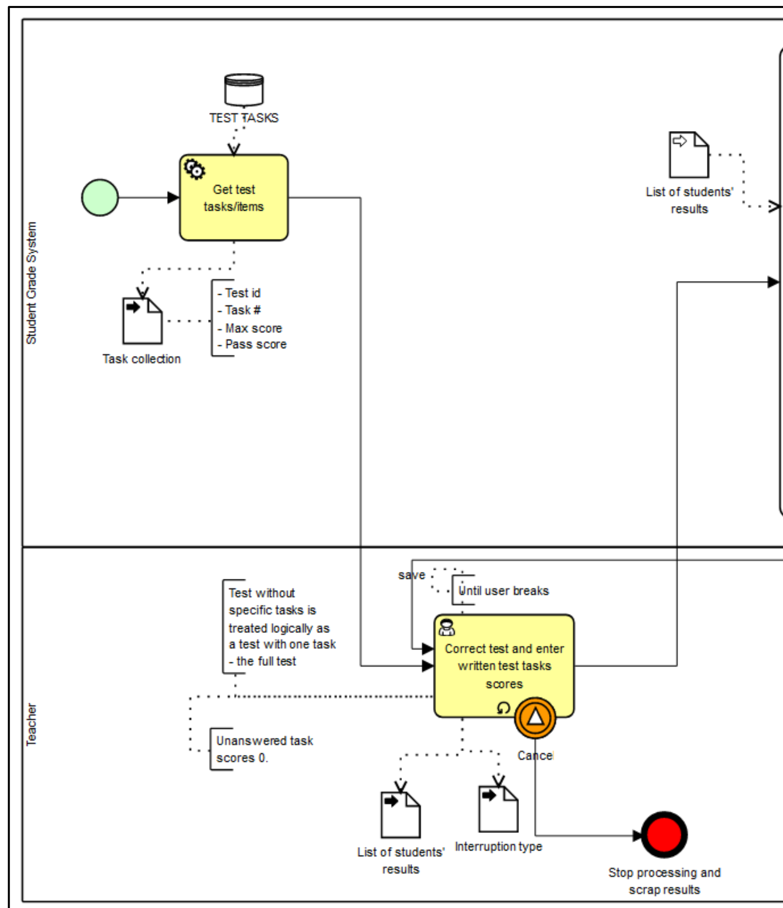
The 'Correct…' task must produce two data objects: one with the students and their score per test task (so programmatically probably a collection of student result objects containing another collection of task score objects) and one to keep track of the interrupt type.

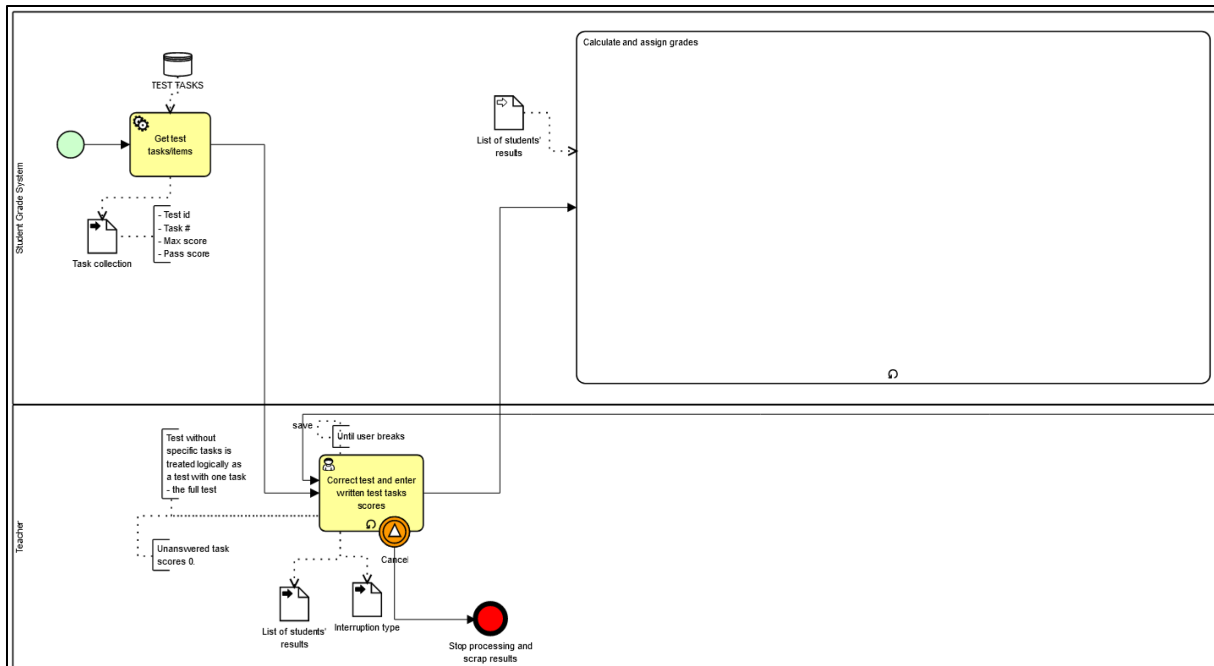
Having done this, we must connect the 'Get test…' task in the 'Student Grade System' lane to the 'Correct…' task in the 'Teacher' lane.

The workflow has now prepared data objects that could be used to calculate test grades for the students and store them in the database. The next step in the flow will be to react to the interrupt event and, depending on the type of interrupt, process the produced data.

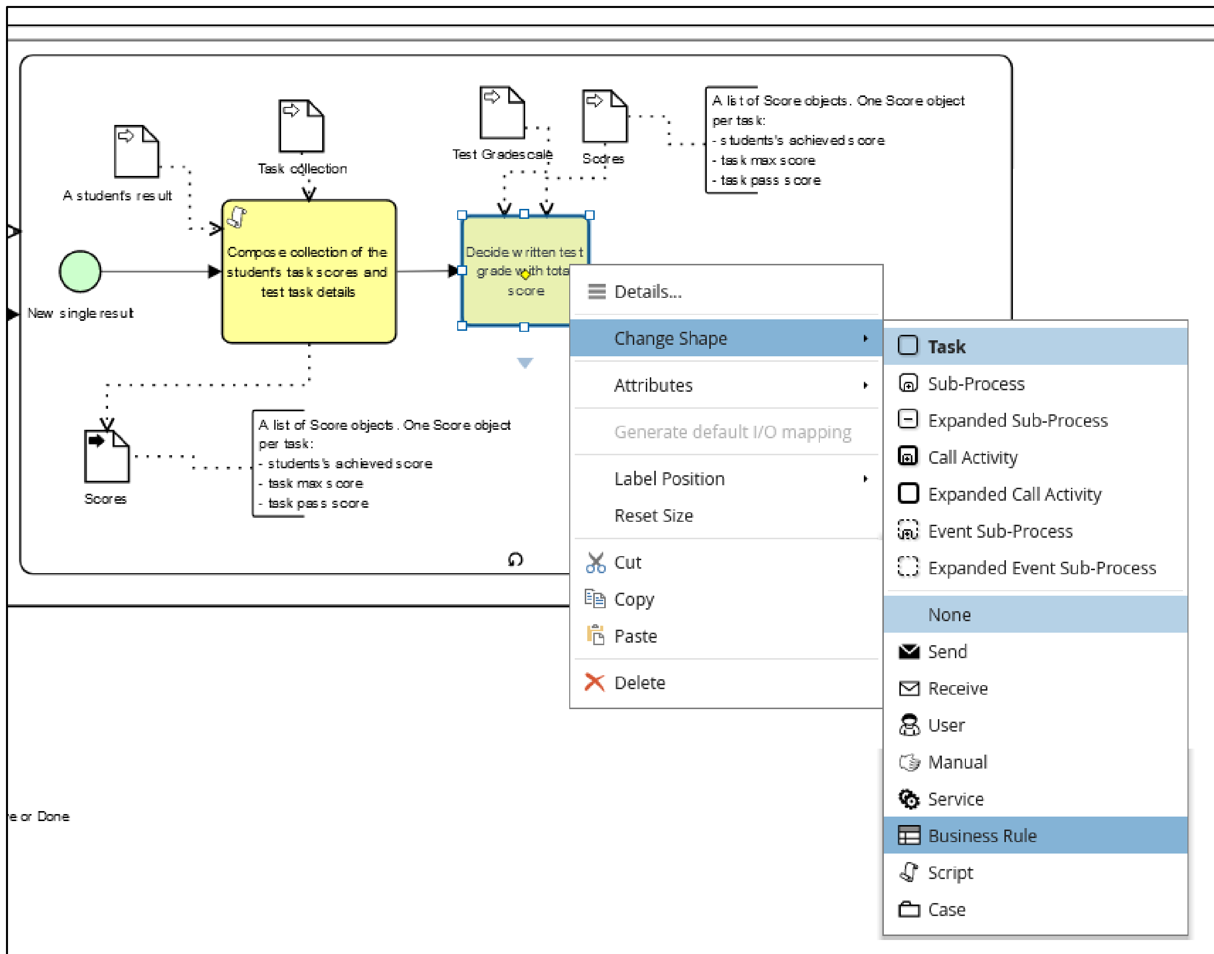
Since we have a collection of data, the 'Student Grade System' needs to process each student and his or her results iteratively in a loop. So, first add a task to the 'Student Grade System' lane and change it into a **Regular** and **Expanded Sub-Process** and set its attributes to **Standard Loop**. Name it "Calculate and assign grades". Finally, add an input data object named 'List of students' results' and connect that to the sub-process. You should now have a model looking like the one below.

Add a **None Start Event** inside the sub-process and name it "New single result". Then add a task named "Compose collection of the student's task scores and test task details" and another task named "Decide test grade with total score".





The first task will compose the needed data objects for the subsequent decision service, and it should probably be a **Script Task**. The second task will execute a decision service to receive the test grade based on the scores on the task in the test per student. This test grade will be stored as the student's grade in the system.



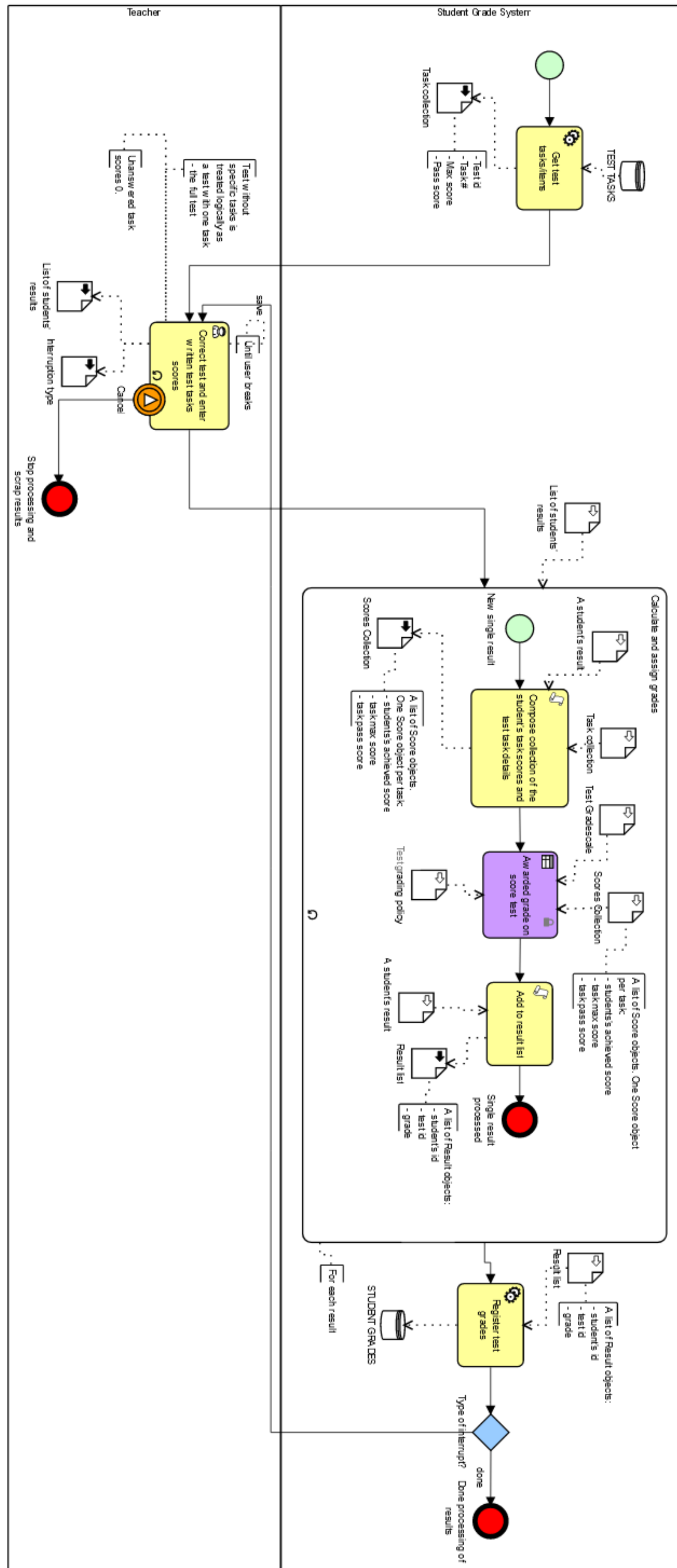
For this to work, the new task must be changed into a **Business Rule Task**. Add all the necessary data input and output objects that are needed to calculate the test grade per student. In addition, add annotations to explain the content in the data objects.

I like to change the colour of BR tasks in BPMN and decisions in DMN diagrams to light purple to easily separate them from other tasks in a BPMN diagram and other shapes in a DMN diagram.

Each time the BR task is done it will produce a test grade as a letter (e.g., “A”) and a text (e.g., “Excellent”). This grade needs to be stored properly in the database. We therefore need another task after the BR task that composes an output data object of a result list with a collection of Test id, Student id, Grade, Teacher signature, Task Scores, and Result Date. This task could probably be a **Script Task**.

The expanded sub-process will loop until there are no more results to process and then continue to a **Service Task** that uses the produced result list to create and execute proper update commands for the records in the database.

Finally, you should have the following complete model for the ‘Correct written test and register results’ sub-process:


Teacher
Student Grade System
TEST TASKS
Get test tasks/items
Task collection
- Test id
- Task#
- Max score
- Pass score
Correct test and enter written test tasks scores
Until user breaks
save
Test without specific tasks is treated logically as a test with one task - the full test
Unanswered task scores 0.
List of students' results
Interruption type
Cancel
Stop processing and scrap results
Calculate and assign grades
A student's result
New single result
Compose collection of the student's task scores and test task details
Task collection
Test Gradescale
Scores Collection
A list of Score objects. One Score object per task:
- student's achieved score
- task max score
- task pass score
Awarded grade on score test
Scores Collection
Test grading policy
Add to result list
A student's result
Result list
A list of Result objects:
- student's id
- test id
- grade
Single result processed
Result list
Register test grades
STUDENT GRADES
For each result
Type of interrupt?
done
Done processing of results

Part III

Modelling the Business Decisions

You should now have workflow models that show the flow of registering results, calculating test grades based on those results, and storing them in the 'Student Grade System'.

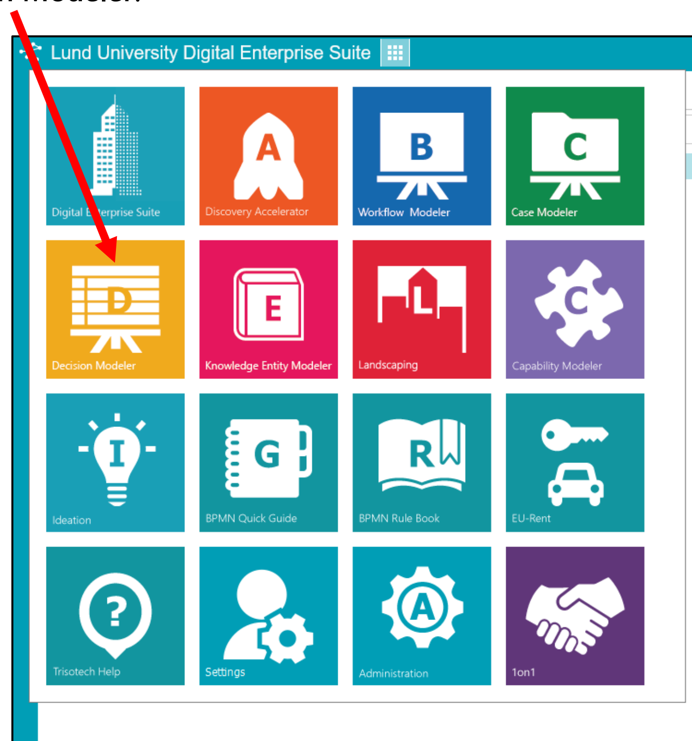
In the diagram you now have one **Business Rule Task** that is responsible for generating test results based on input and decision logic. The very idea of Business Decision Management is that models and logic for decision making must be kept separate from process models, workflow models, data models, and so on. We should thus *not* try and model the required decisions using BPMN. Instead, we should use the DMN standard and design DRDs and DL that take care of the decision making needed.

The order of modelling is seldom as sequential as it is portrayed here. Probably you would model workflows and decisions in parallel. If we are working with a decision-centric or rules-rich workflow/process, we should start in the decision end. As it happened, this time in this tutorial it was easier to start with workflow modelling.

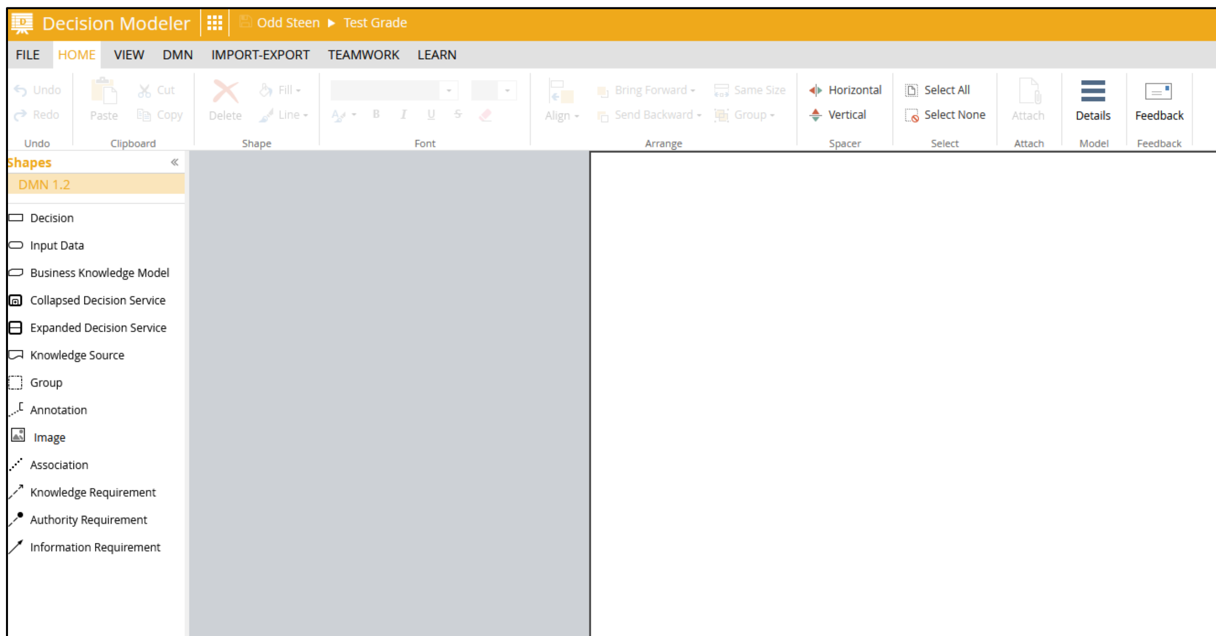
Step 1: Adding a DRD to a Business Rule Task

In this case, you will design the decisions requirements and decision logic for the **Business Rule Task** 'Decide written test grade with total score and pass scores'.

To add a decision diagram to control the decision in the workflow task, go to your Lund Digital Enterprise Suit tab of your browser and click the matrix like symbol to the right and select **Decision Modeler**:



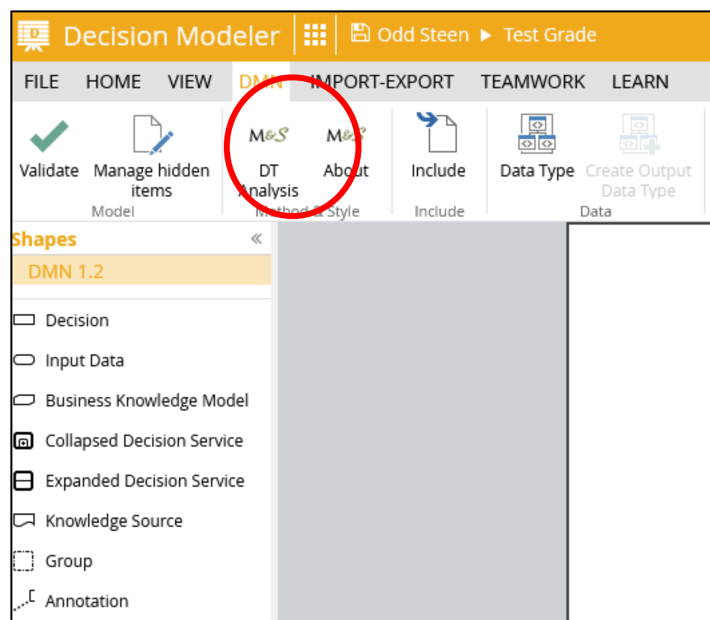
This will open a new tab with an empty diagram canvas to the right and the DMN 1.5 palette to the left. Name the diagram “Test Grade”. You should now have a graphical editor looking like this:



Before we do any modelling of decisions, input data, etc. we need to create the necessary data types that the decision making will need.

Step 2: Creating Data Types for the Decisions

A DRD models decisions that take input and use that to produce output. Input can be either data or the output from a preceding decision. Decision logic may be designed, and it details on what grounds a certain output is generated from the input.

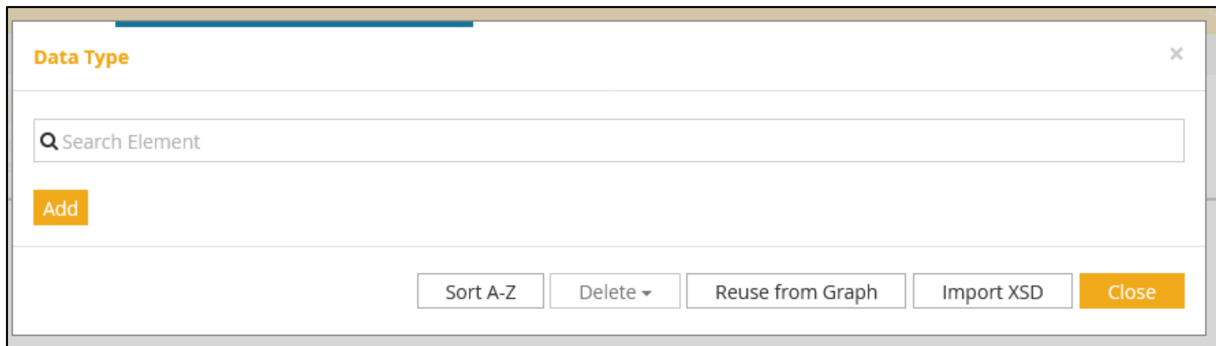


The BR task in the workflow inputs two data objects to the decision service: ‘Test Gradescale’ and ‘Scores’. ‘Scores’ is a collection of ‘Score’ objects where each score object

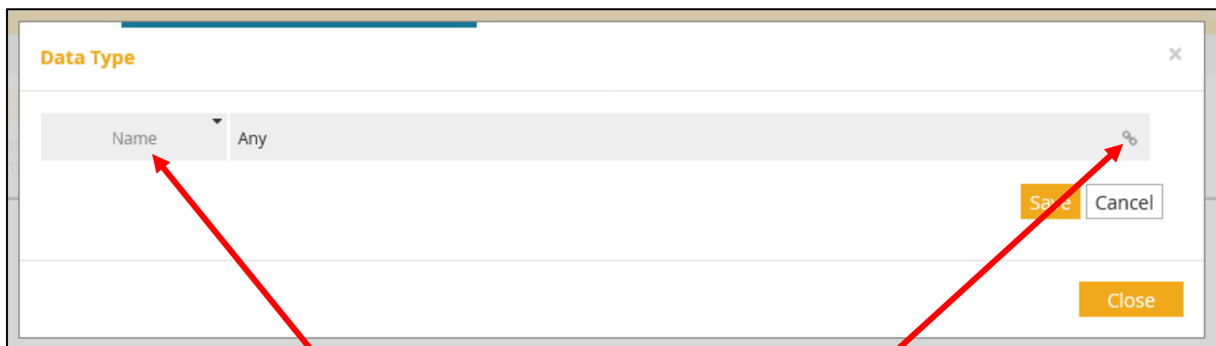
holds the pass and max score for the test task and the student's achieved score for the test task.

To decide on the student's test grade for this kind of test means to check if the student has results on all tasks in the test and that the score per task is at least equal to 'passScore'. If true, the grade is calculated as percentage of the sum of the student's task scores.

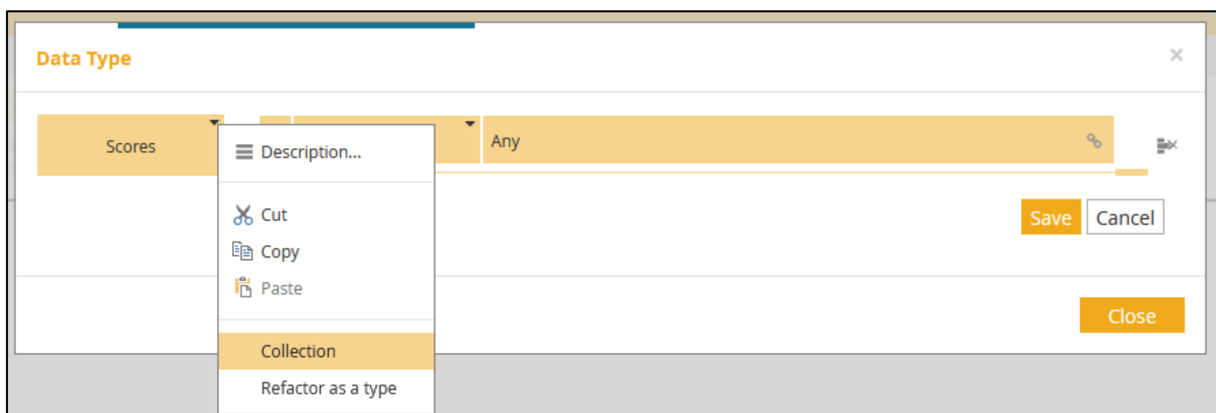
In your DMN tab click **Data Type**. This will open a new dialogue where you specify the new data type.

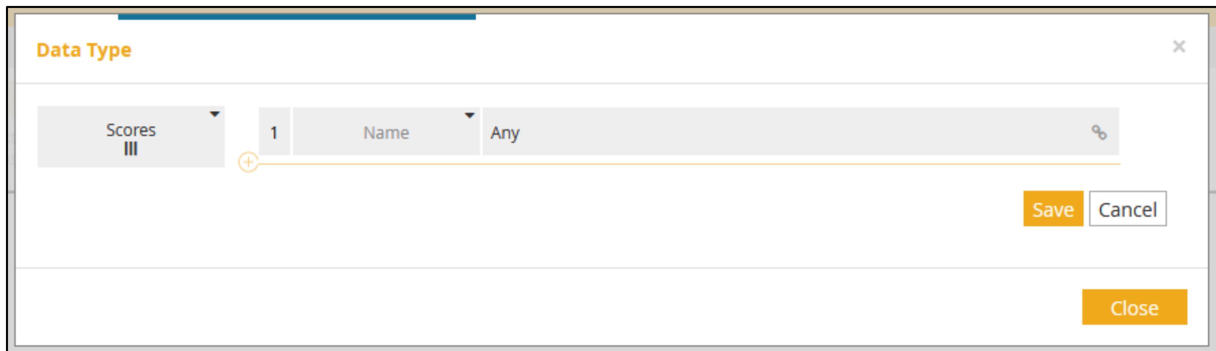


Here you click **Add.**

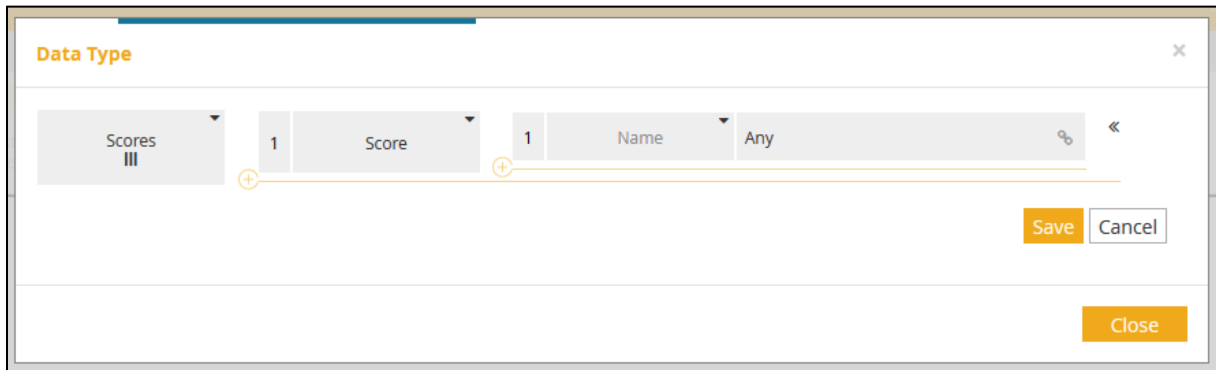


Write "Scores" in the **Name** field. Then click on the little link symbol to the right and in the pop-up menu select **Structure.** Turn 'Scores' into a **Collection.**

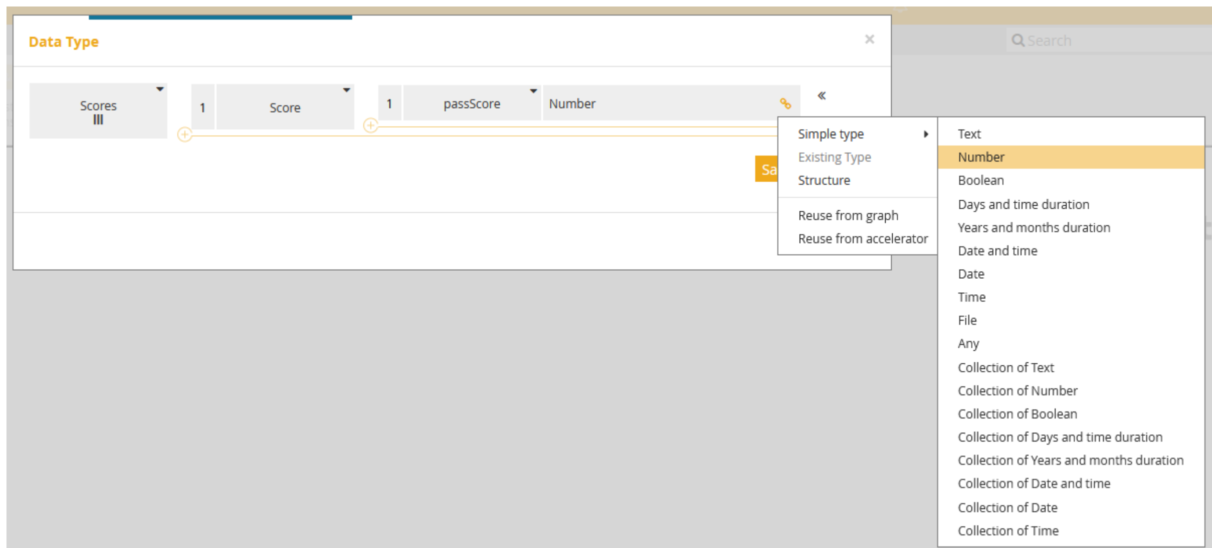




In the **Name** field of the first item write “Score”. Also turn that into a **Structure**.



Add members to the ‘Score’ structure and type them as **Number**:



Continue until you have this:

Step 3: Designing the DRD

To decide on the final test grade for a student we first need to know if all the tasks of the test are passed by the student. Since this is a written test, we also need to calculate the grade based on the percentage of the maximum test score achieved by the student. Hence, the decision on final test grade is preceded by the decision on the student having passed all test tasks and the decision on which grade the student has achieved based on the sum of task scores:

1. Test grade
    - 1.1. Are all test tasks passed?
    - 1.2. What is the grade given the total score percent?

OK, but to decide whether all tasks are passed we must decide – per task – if the achieved task score is greater than or equal to the task pass score. In addition, if a student does not have scores for all tasks included in the test, he or she has not passed the test. We then have:

1. Test grade
    - 1.1. Are all test tasks passed?
        - 1.1.1. Is the achieved score for each task greater than or equal to the task pass score?
        - 1.1.2. Does the student have score for all tasks?
    - 1.2. What is the student's grade given his or her total score percent?

To decide the grade based on achieved total score means that we need to know the sum of the achieved scores and compare that to the maximum test score to render a percentage. This percentage is the ground for deciding on the grade. We then have:

1. Test grade
    - 1.1. Are all test tasks passed?
        - 1.1.1. Is the achieved score for each task greater than or equal to the task pass score?
        - 1.1.2. Does the student have score for all tasks?
    - 1.2. What is the student's grade given his or her total score percent?
        - 1.2.1. What is the sum of the student's scores for the tasks?

- 1.2.2. How many percent of the test maximum score is the sum of the student's scores for the tasks?
- 1.2.3. What grade matches that sum?

To decide whether an achieved task score is greater than or equal to or below the task pass score, we must calculate that difference. If task score – pass score < 0 then the task is failed, otherwise it is passed. We then have:

1. Test grade
   - 1.1. Are all test tasks passed?
     - 1.1.1. Is the achieved score for each task greater than or equal to the task pass score?
       - 1.1.1.1. What is the student's score on a task?
       - 1.1.1.2. Is that score minus the task pass score below zero or not?
     - 1.1.2. Does the student have score for all tasks?
   - 1.2. What is the student's grade given his or her total score percent?
     - 1.2.1. What is the sum of the student's scores for the tasks?
     - 1.2.2. How many percent of the test maximum score is the sum of the student's scores for the tasks?
     - 1.2.3. What grade matches that sum?

We also need to know if the student has results for all tasks in the test. In this case of a written exam, not answering a question would mean 0 points for that task. If you hand in a blank written exam, it will thus mean that you will score 0 on all included tasks. Not handing in a written exam would mean no scores at all.

But when we have a test of scored assignments, not handing in one of them does not mean 0 as score. That instead means that you will have no score for that assignment. To handle that in an IS could be to set the score to -1 to signal lack of result (since the value needs to be numeric)

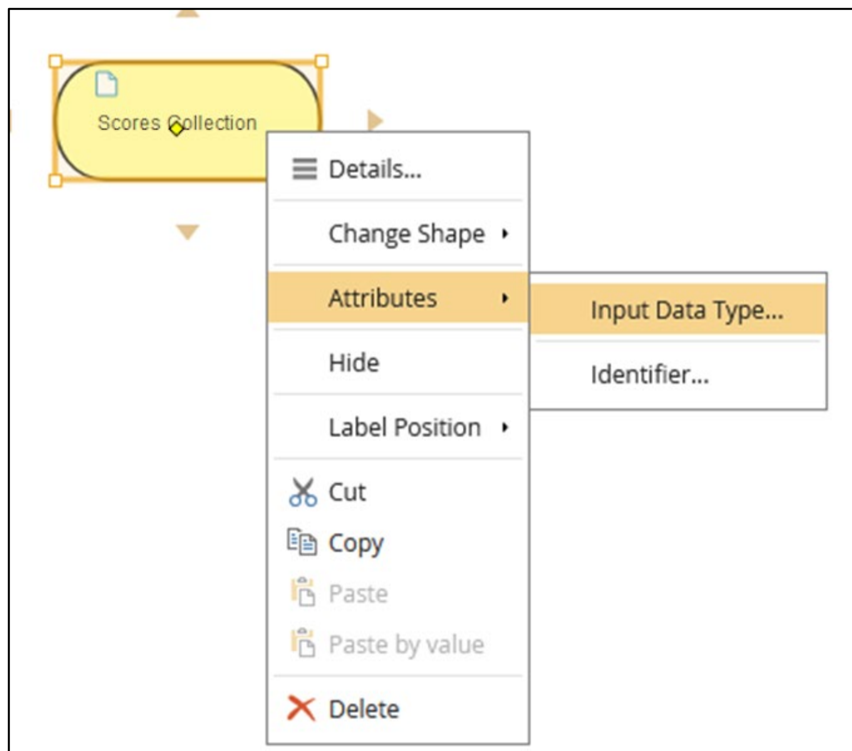
Finally, we have:

1. Test grade
   - 1.1. Are all test tasks passed?
     - 1.1.1. Is the achieved score for each task greater than or equal to the task pass score?
       - 1.1.1.2. What is the student's score on a task?
       - 1.1.1.3. Is that score minus the task pass score below zero or not?
     - 1.1.2. Does the student have score for all tasks?
       - 1.1.2.2. Is the score greater than or equal to zero?
   - 1.2. What is the student's grade given his or her total score percent?
     - 1.2.1. What is the sum of the student's scores for the tasks?
     - 1.2.2. How many percent of the test maximum score is the sum of the student's scores for the tasks?
     - 1.2.3. What grade matches that sum?

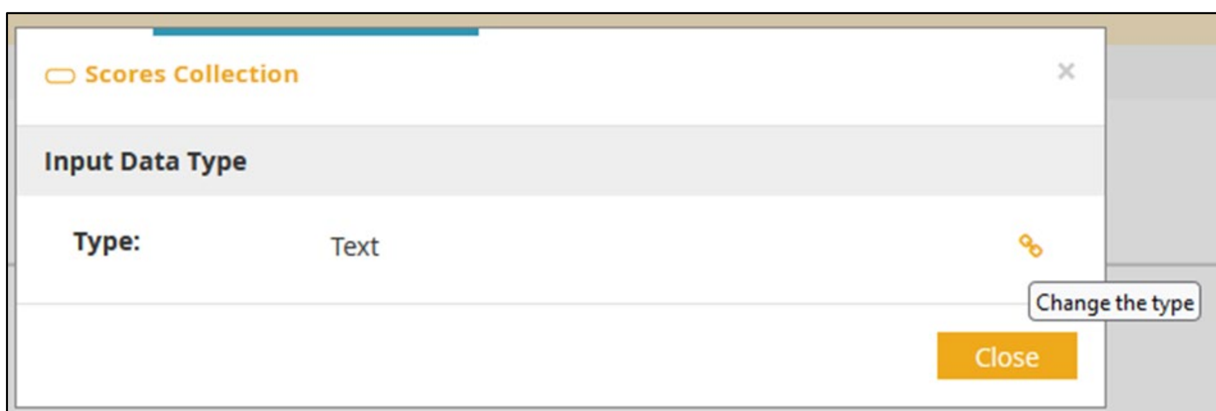
Step 4: Does the Student Have Score for All Tasks?

Let us begin with the simple question of item 1.1.2 in the list above. To decide on this, we need to iterate through the collection of the student's task scores and test each score to see whether it is greater than or equal to 0.

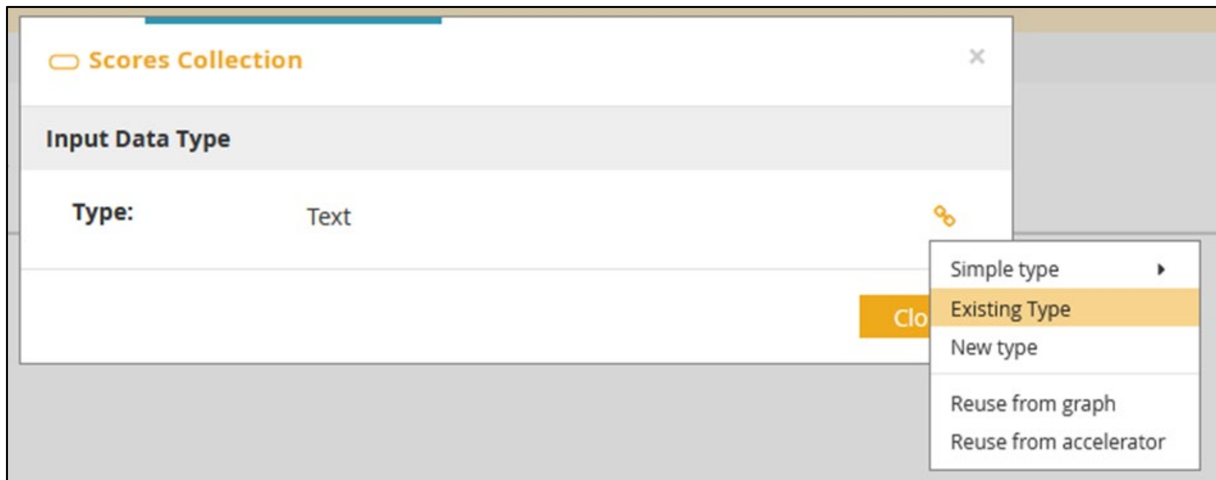
We need one data input: a list of the 'Scores' objects according to the data model and dictionary. Drag an **Input Data** symbol onto the canvas, change its colour to light yellow and change its **Input Data Type...**



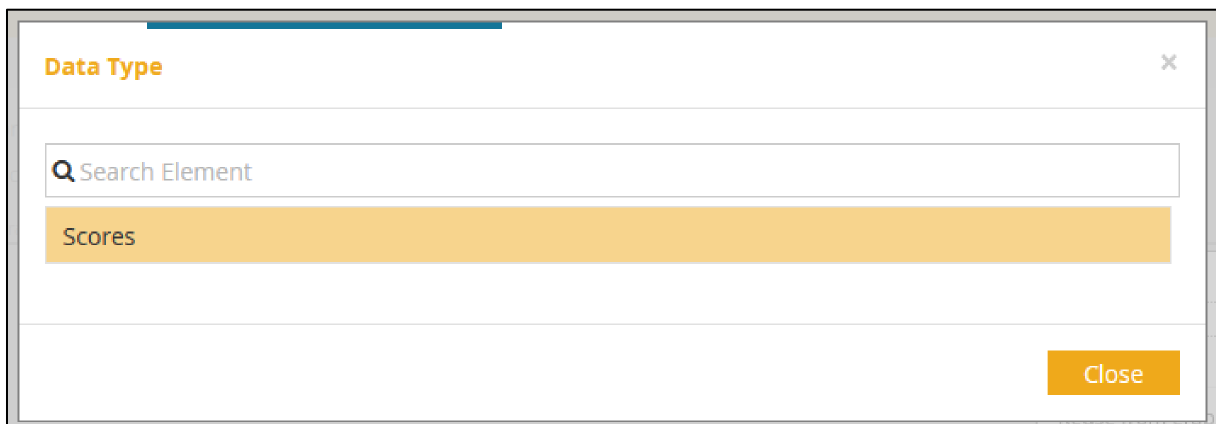
Click the little link symbol to the far right of the **Type:** field:



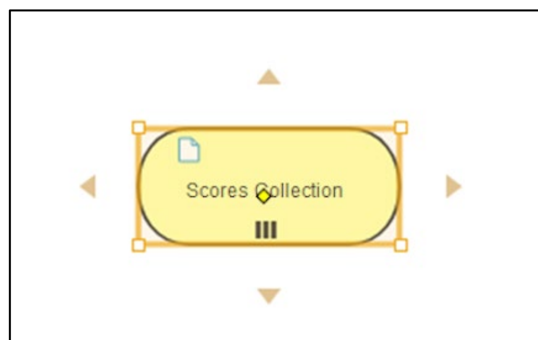
In the pop-up menu select **Existing Type.**



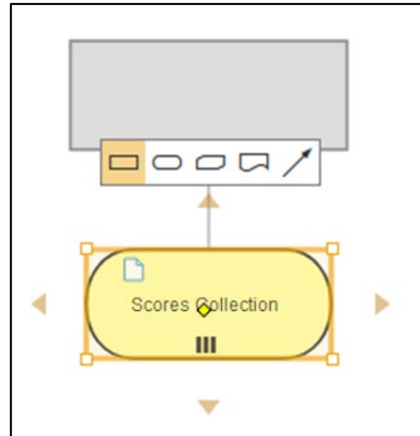
Select your previously specified data type 'Scores'.



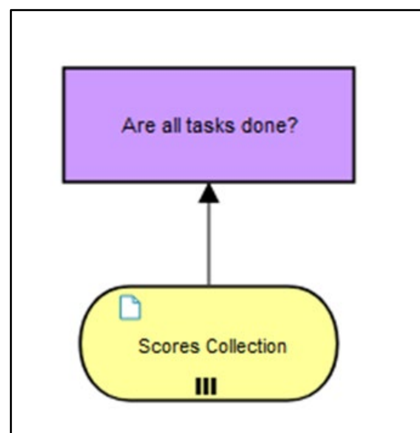
Close the dialogue and you should have the following on your canvas:



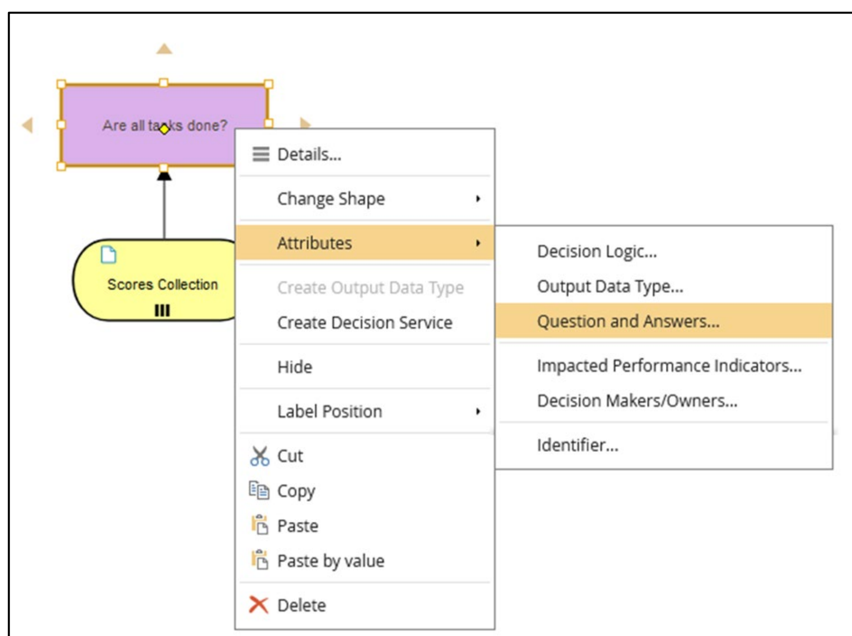
The three vertical bars denotes that the input data is in the form of a collection. Use the north arrow to add a decision to the model.



Change its colour into purple and name it “Are all tasks done?” That naming convention will tell you that the decision outcome is either yes or no and hence a Boolean output.



Now we will add the decision logic to the decision, i.e., the precise way to get to the outcome from the input data. But before that we will describe and document the question this decision will answer and the all the possible answers.



Are all tasks done?

Question and Allowed Answers

Question

Has the student done all the tasks in the test? For instance, if there are four questions/tasks in the test and the student has provided three answers then he or she has not done all tasks.

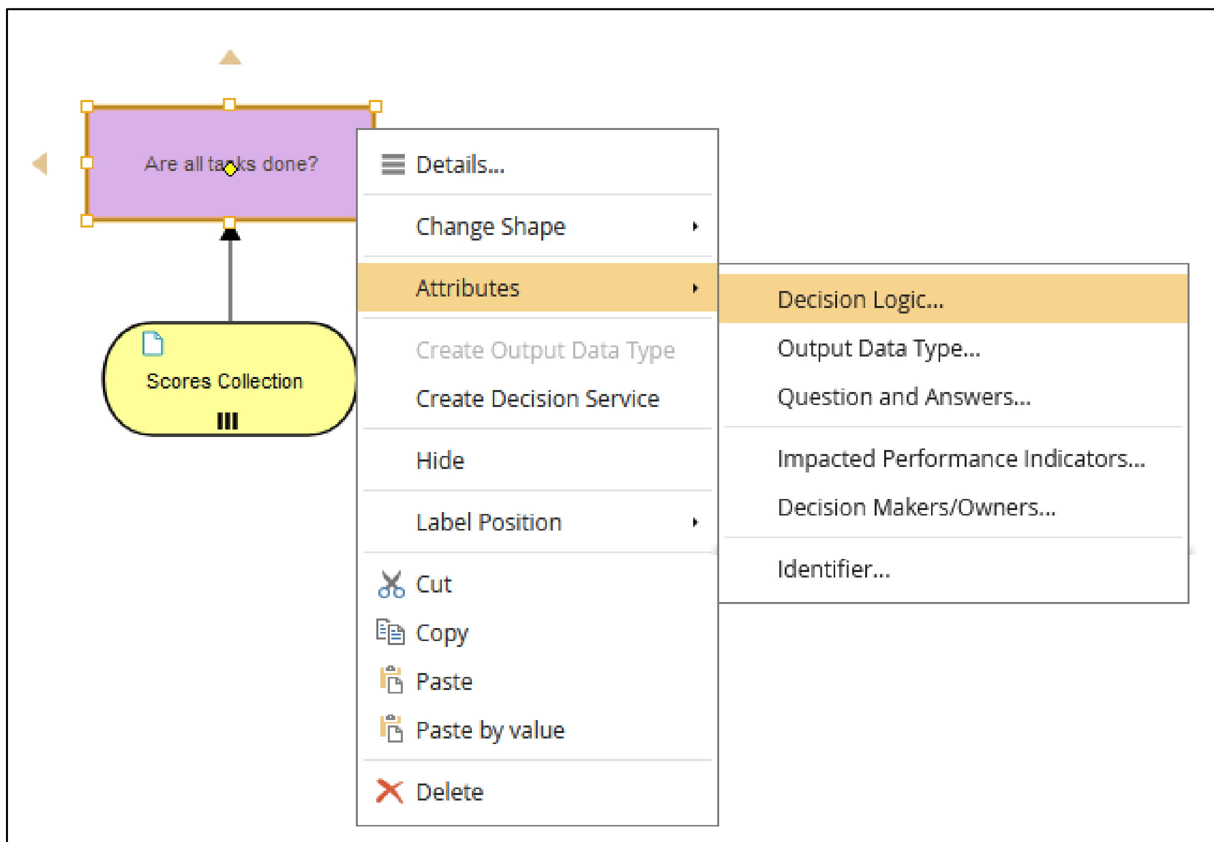
Allowed Answers

Yes or No.

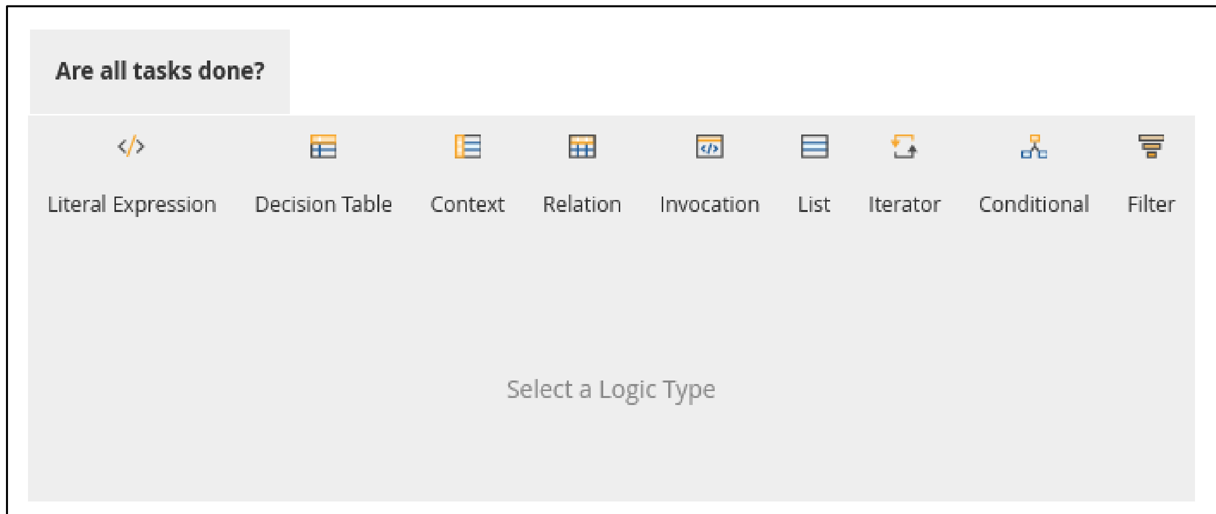
Use Data Output Type as allowed answers

Close

Now we add the decision logic for the decision. Right-click the purple decision shape in your model, expand **Attributes** and select **Decision Logic...**



You should get this DL modeler:



The logic we want to specify is whether a student has results on all the tasks included in the test. To do this we need to know the number of tasks in the test and the number of tasks the student has done: achievedScore - passScore per task in the student's result is greater than or equal to 0 (i.e., greater than -1 which denotes absence of a result). When we have these counts, we can check the following:

If
  The number of test tasks > number of student result tasks where achievedScore > 0
Then
    All tasks are done? = false
Else
    All tasks are done? = true

But I couldn't make this work in FEEL. So, I had to revert to another solution.

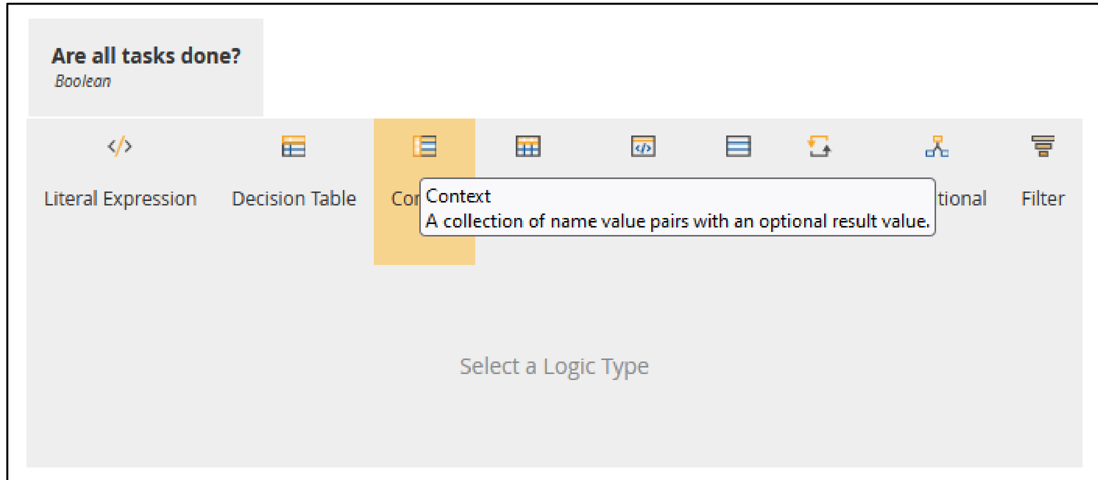
There are several list operations in FEEL that could possibly be used to work with the list of scores. Two of these are **all(list)** and **any(list)**. The **all(list)** function returns `false` if any element in the list is false, `true` if all elements are true, and `null` otherwise. The **any(list)** function returns `true` if any element is true, `false` if all elements are false, and `null` otherwise.

First of all we need to establish whether the scores collection contains any score element where score.achievedScore is below zero (< 0). To do this we use the **for…in…return** statement, which is used to iterate (**for**) over a collection (**in**) and return (**return**) a new list fulfilling the conditions. We can then test the existence of `true/false` in the new list with **all(list)** or **any(list)**.

Thus, we iterate over `Score` Collection.Score and pick each `score item` in the list and return a new list 'Undone tasks' where score item.achievedScore is less than zero.

The next test will be to use a conditional expression that checks whether 'Undone tasks' contains any `true` value, which means that at least one tasks is undone. If so, 'All tasks are done' is false, or else its true.

Let's begin with the iteration over the list of score elements. Create a new **Context**. Name it "Done tasks".

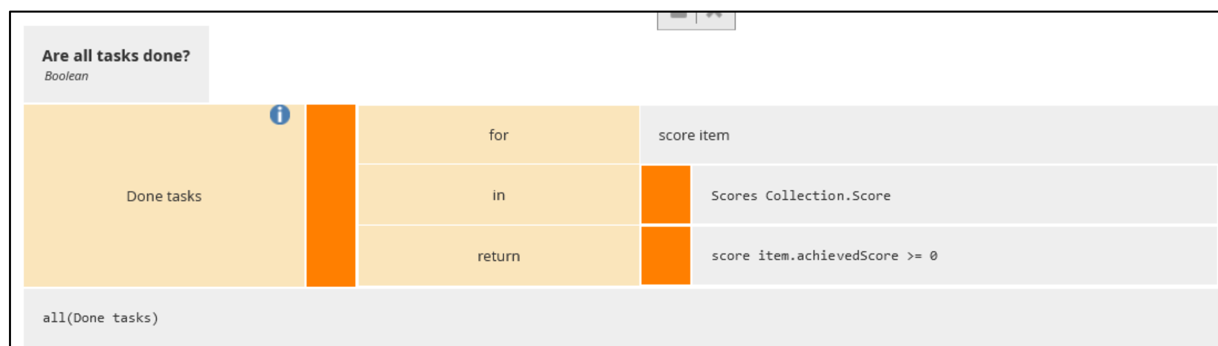


In the new **Context** add an **Iteration.**

Enter **Literal Expressions** in each of the **Iterator** statement's parts. Use auto completion for the variables.

Now we need a test to find out if the 'Done tasks' list contains any false elements. Click on the yellow +-sign to add a new row below. Name the variable "Are all tasks done?" and select **Conditional.**

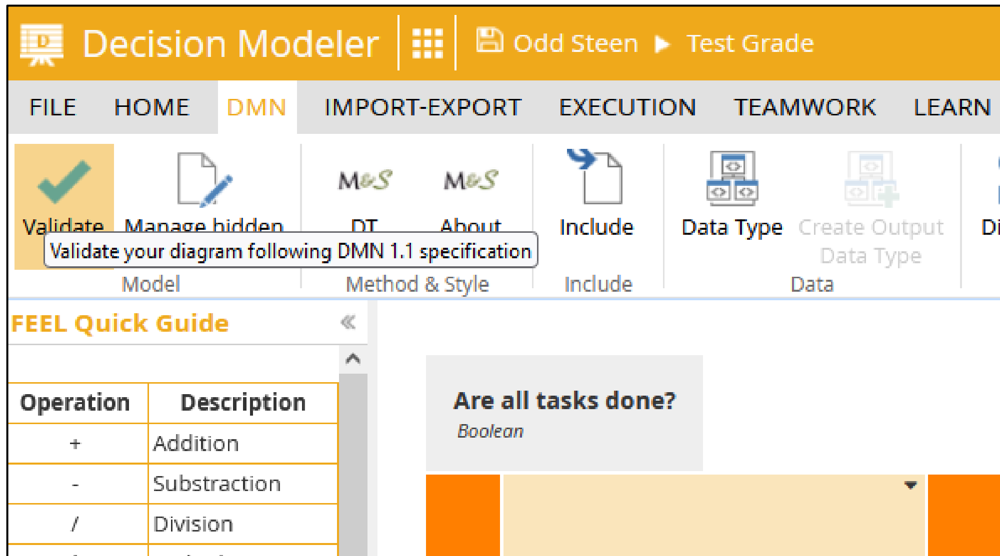
Use **Literal Expressions** to specify the logic in the **Conditional** expression. Set the output to be 'Are all tasks done?'



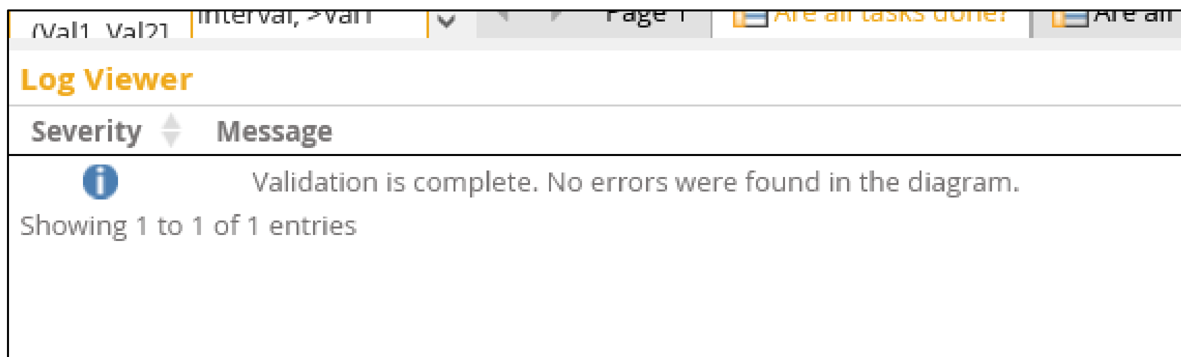
Step 5: Test Your Decision

You should now have a first complete decision!

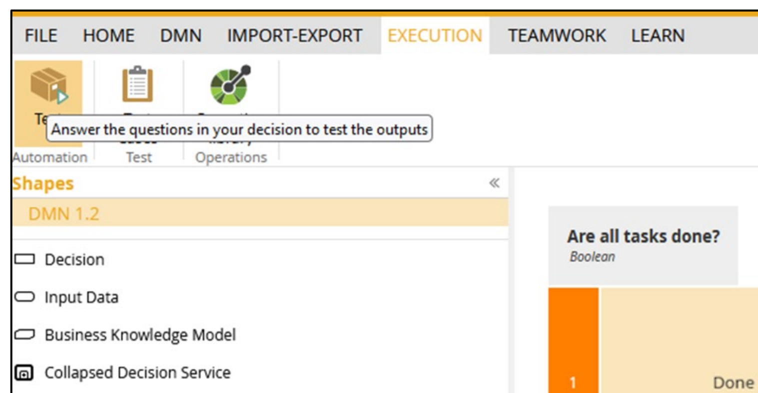
Now we will first see if it contains any errors. Click on **Validate** in the **DMN** ribbon.



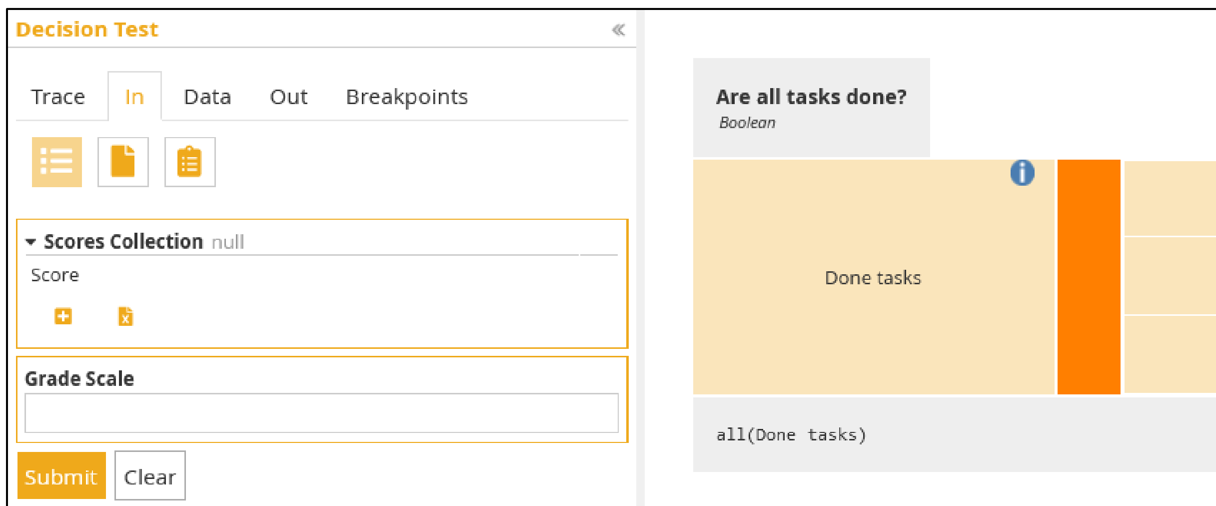
You should get this message at the bottom of the screen:



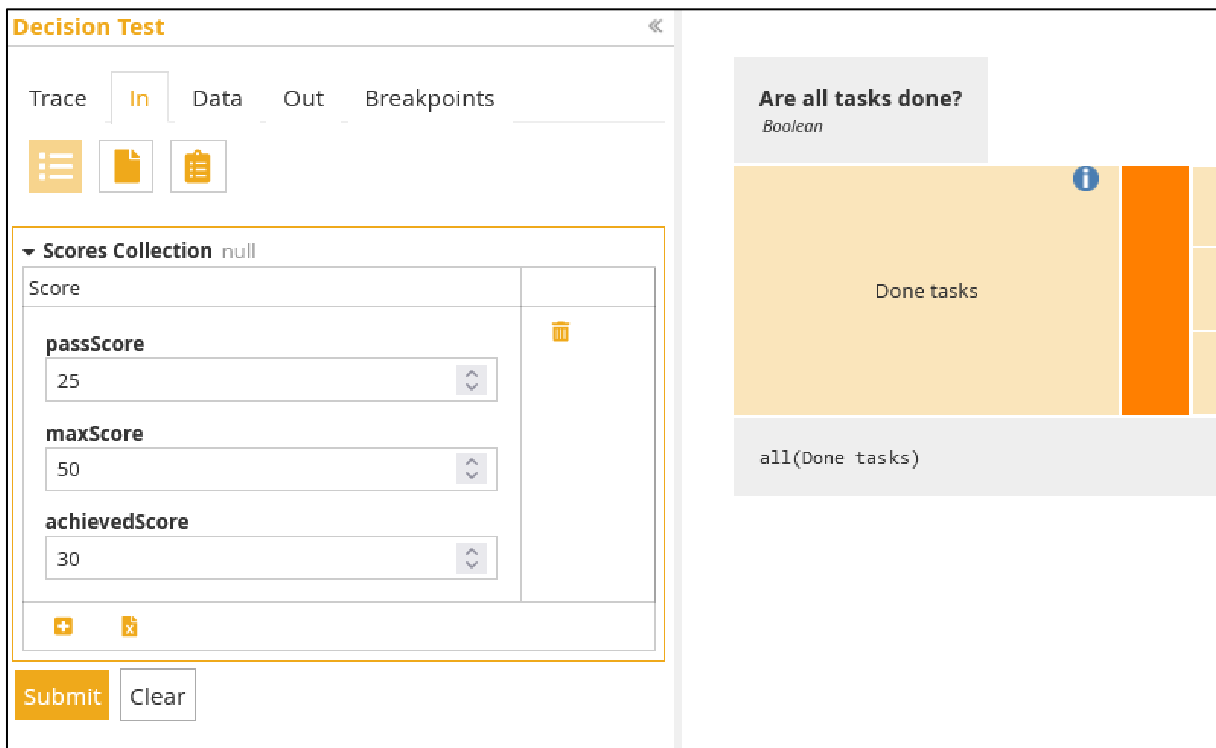
So, we have no formal errors. Now we will test the DL to see if it works the way it should. Go to **Execution** and select **Test.**



Click on the +-sign under 'Score' to add an element to the collection.



Enter values for the fields of Score in Score Collection:

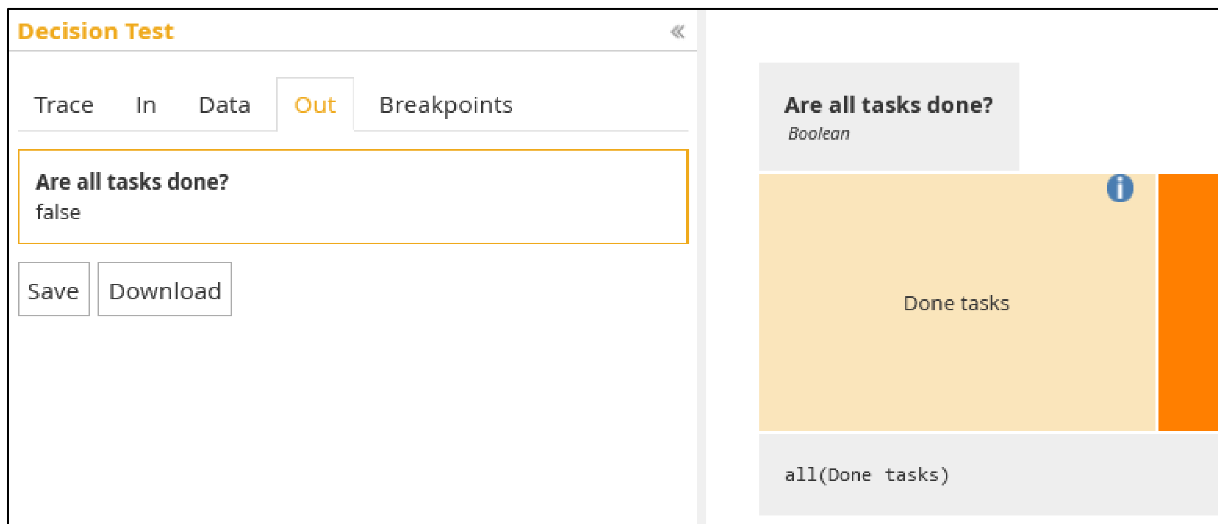


Click **Submit** to run the logic. The output should be this:

You could **Save** this test for later tests, so you don't need to construct the same test data repeatedly.

Run a new test with "-1" for achievedScore which means that there is no achievedScore for this task.

The result should be:

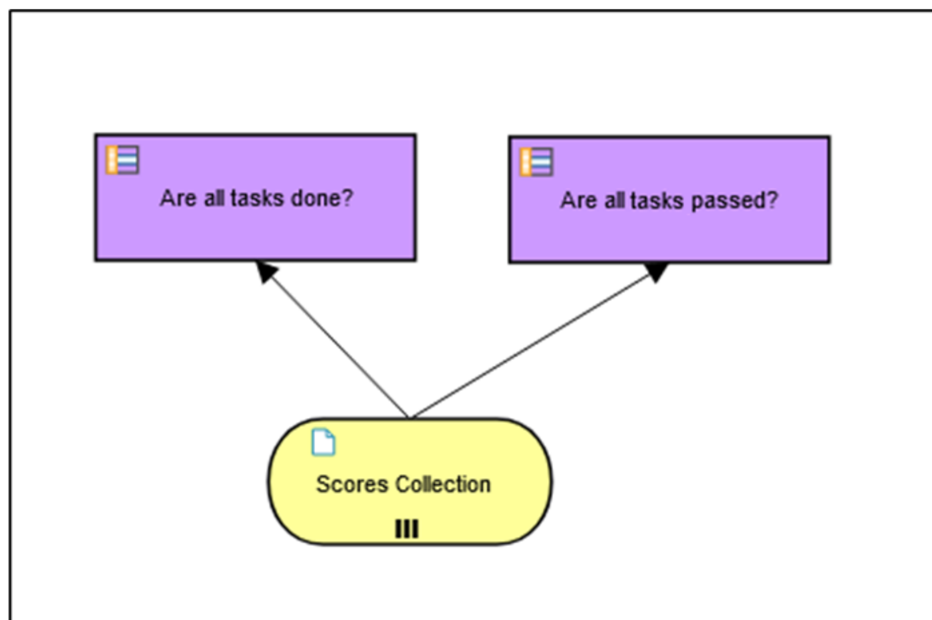


Great! This seems to work as intended. Save the test as “Test case 2: 25,50,-1”.

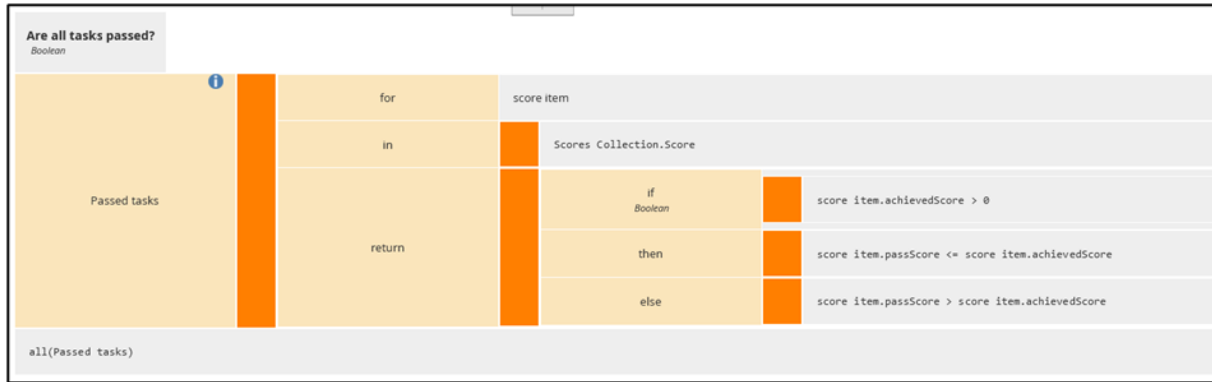
Step 6: Is the Achieved Score for Each Task Greater Than or Equal to the Task Pass Score?

The next step would be to check whether the student has passed all the tasks he or she has results on. Again, it is an iteration over the collection of the student’s results and a comparison with the tasks of the test. In this case, the comparison is between the student’s scores on tasks and the pass limit of each task. If the score is below the pass limit the task is failed.

Add a new decision shape from the data input and name it “Are all tasks passed?”



Specify its DL like this:



Step 7: Test the Decision

First do **Validate** to see if there are any formal errors. To test the DL, use the same input for testing this DL as the previous DL above. Go to your list of test cases on the **Decision Test** page and pick your intended test:

Decision Test

Trace In Data Out Breakpoints

Test Case

Pick the test case to load

Test case 1: 25,50,30

Load

Load it. Hit **Submit.**

Correctly 'All tasks are passed?' is `true` as is 'Are all tasks done?'. It works.

Load the "Test case 2: 25,50,-1":

That worked too.

Let's try [[25,50,30],[25,50,20]]. That should give true and false.

Let's try [[25,50,30],[25,50,25],[25,50,-1]]. That should give false and true.

Step 8: Another Way to Do It

There are several ways to set up FEEL expressions to do the same thing. The two solutions above are clear and easy to understand, but maybe a bit long and verbose for quite simple decisions.

You will achieve the same logic by changing the FEEL expressions like this:

Are all tasks done?
Boolean

1	Done tasks	`Scores Collection.Score[achievedScore > -1]`

```
count(Done tasks) = count(Scores Collection.Score)
```

And this:

Are all tasks passed?
Boolean

1	Passed tasks	`Scores Collection.Score[achievedScore >= passScore]`

```
count(Passed tasks) = count(Scores Collection.Score)
```

You do this by creating a **Context** with **Literal Expression** per decision that filters out elements in `Scores` Collection.Score that fulfil the expression inside the "[]" part and create a new list with those elements. In the output, the number of elements in this new list is compared to the number of elements in the original score collection list. If the numbers are equal, the result is true otherwise false. This is a more compact way to do the same things as above, but perhaps a tad more "programming".

Step 8.1: Yet Another Way to Do It

The most compact way to do it is to use the '**every {range variable} in {list expression} satisfies {Boolean expression with range variable}**' function as **Literal Expression**. The 'element' evaluates to a Score object in the Score Collection and thus has the achievedScore, maxScore, and passScore attributes.

To decide if all tasks are done, we therefore check the Score Collection list that every Score object in the list has achievedScore greater than -1. If so, the result is true, else it is false.

Are all tasks done?
Boolean

```
every element in Scores Collection.Score satisfies element.achievedScore > -1
```

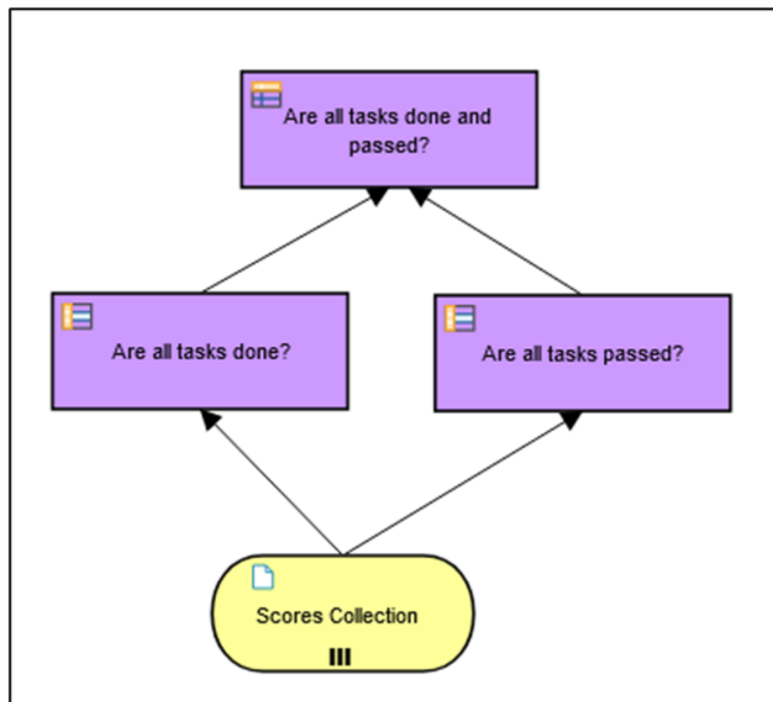
To decide if all tasks are passed, we do the similar: Every Score object in the list must have achievedScore which is greater than or equal to passScore. If so, the result is true, else it is false.

Are all tasks passed?
Boolean

```
every element in Scores Collection.Score satisfies element.achievedScore >= element.passScore
```

Step 9: Are All Test Tasks Done and Passed?

Now we can set up the decision and DL for deciding if all the tasks are done and all the done tasks are passed. Add a new decision to your DRD and name it “Are all tasks done and passed?”



We have three possible outcomes from this decision. Fill in the Q&A of the decision.

Are all tasks done and passed?

Question and Allowed Answers

Question

Has the student done all tasks in the test and has he or she passed all the done tasks? There are three possibilities: 1. He or she has completed the test and has passed; 2. He or she has completed the test but has failed; 3. He or she has not completed the test.

Allowed Answers

1. Complete, pass
2. Complete, fail
3. Incomplete

☐ Use Data Output Type as allowed answers

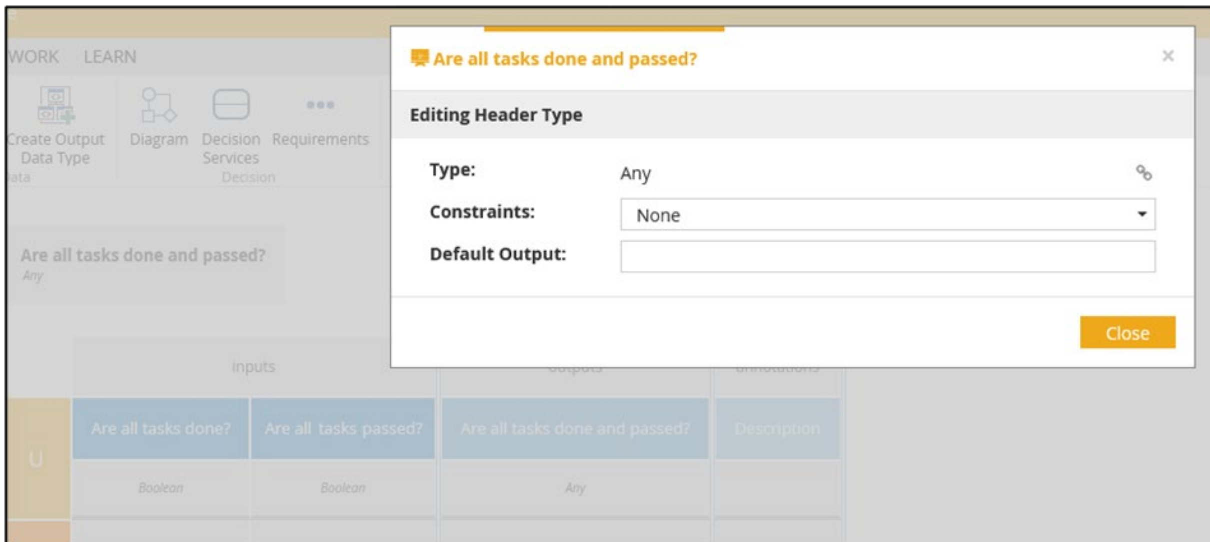
Close

From the **Answer** of the Q&A we see that we need an enumeration as output of the decision. We will specify the enumeration for the output column of the **Decision Table** that will be the DL of this decision.

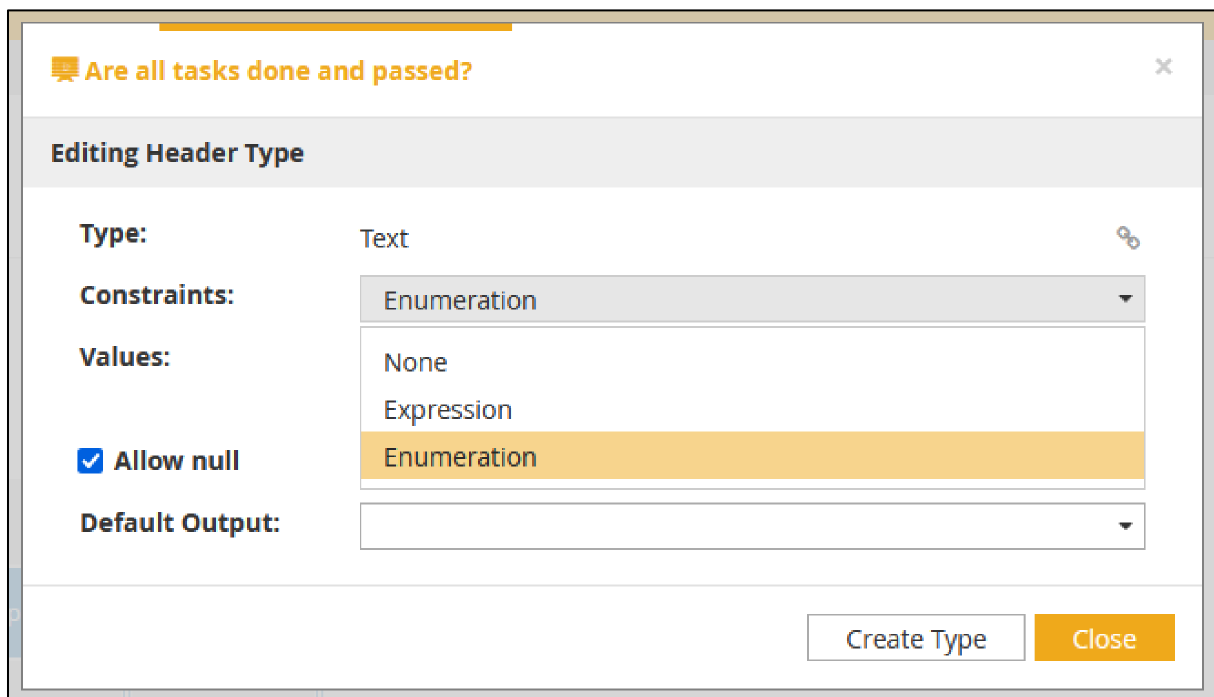
Are all tasks done and passed?
Any

	inputs		outputs	annotations
U	Are all tasks done?	Are all tasks passed?	Are all tasks done and passed?	Description
	Boolean	*Boolean*	*Any*	
1				

Click on **Any** in the header of 'Are all tasks done and passed?' to open the header type editor.



Change the **Type** to **Text** and set the **Constraints** to **Enumeration**.



Add the three output options as strings of the enumeration.

Close the dialogue and you should get this:

Specify the first business rule in row one by clicking in the cells and selecting the right alternatives until you have this:

Are all tasks done and passed? *Text*				
	inputs		outputs	annotations
U	Are all tasks done?	Are all tasks passed?	Are all tasks done and passed?	Description
	Boolean	*Boolean*	*Text* "complete, pass", "complete, fail", "incomplete"	
1	true	true	"complete, pass"	
2	true	false	"complete, fail"	
3	false	-	"incomplete"	

Step 10: Test Your Decision

Use the same input as above and you should get:

[25,50,30]

Are all tasks done and passed? *Text*				
U	Are all tasks done?	Are all tasks passed?	Are all tasks done and passed?	Description
	Boolean	*Boolean*	*Text* "complete, pass", "complete, fail", "incomplete"	
1	true	true	"complete, pass"	
2	true	false	"complete, fail"	
3	false	-	"incomplete"	

[[25,50,30],[25,50,20]]

Are all tasks done and passed? Text				
U	Are all tasks done?	Are all tasks passed?	Are all tasks done and passed?	Description
	Boolean	Boolean	Text "complete, pass", "complete, fail", "incomplete"	
1	true	true	"complete, pass"	
2	true	false	"complete, fail"	
3	false	-	"incomplete"	

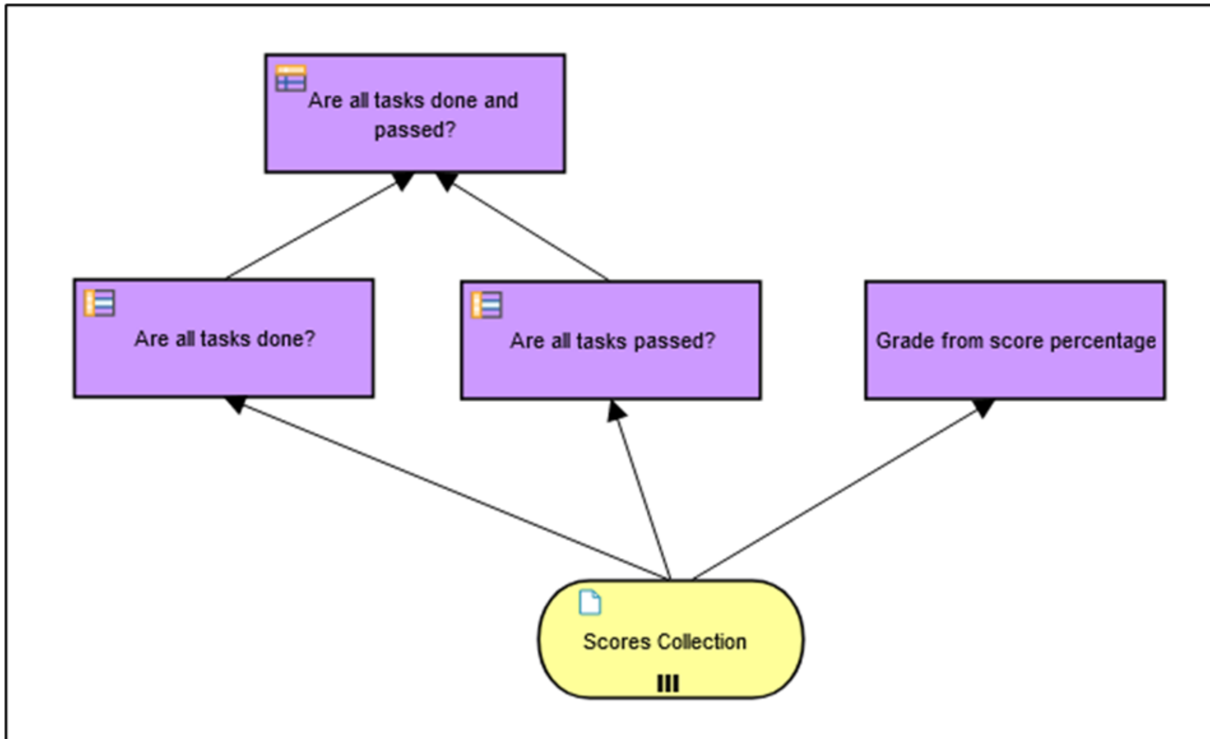
[[25,50,30],[25,50,25],[25,50,-1]]

Are all tasks done and passed? Text				
U	Are all tasks done?	Are all tasks passed?	Are all tasks done and passed?	Description
	Boolean	Boolean	Text "complete, pass", "complete, fail", "incomplete"	
1	true	true	"complete, pass"	
2	true	false	"complete, fail"	
3	false	-	"incomplete"	

Step 11: What Is the Student's Grade Calculated from The Achieved Total Score on the Test?

It's not very likely that a result is graded "complete, pass" or the like. Rather, a grade according to a set policy is calculated/inferred from the score of a test. The first decision is thus to generate a letter grade from a score.

Drag a decision shape onto your canvas, set its colour to purple, and name it "Grade from score percentage".



Specify the Q&A of the decision.

Grade from score percentage

Paragraph
Open Sans 14px

A grade for a score test is calculated as the rounded achieved score percentage of the test max score. The percentage is used to match a letter grade given a grade scale table.

At LUSEM the table for a graded (UA) test looks like this:

A: 85-100%

B: 75-84%

C: 65-74%

D: 55-64%

E: 50-54%

U: 0-49%

Example: Achieved score = 63. Test max score = 100. Percentage = 63% which is in the interval for grade D.

Allowed Answers

A letter grade inferred from the achieved score, the test max score, and the test grade scale.

☐ **Use Data Output Type as allowed answers**

Close

The math is quite simple:

round up(100 * (Total achieved score of the test / Total score of the test))

Add the DL to the decision as three **Contexts**.

Achieved test percentage

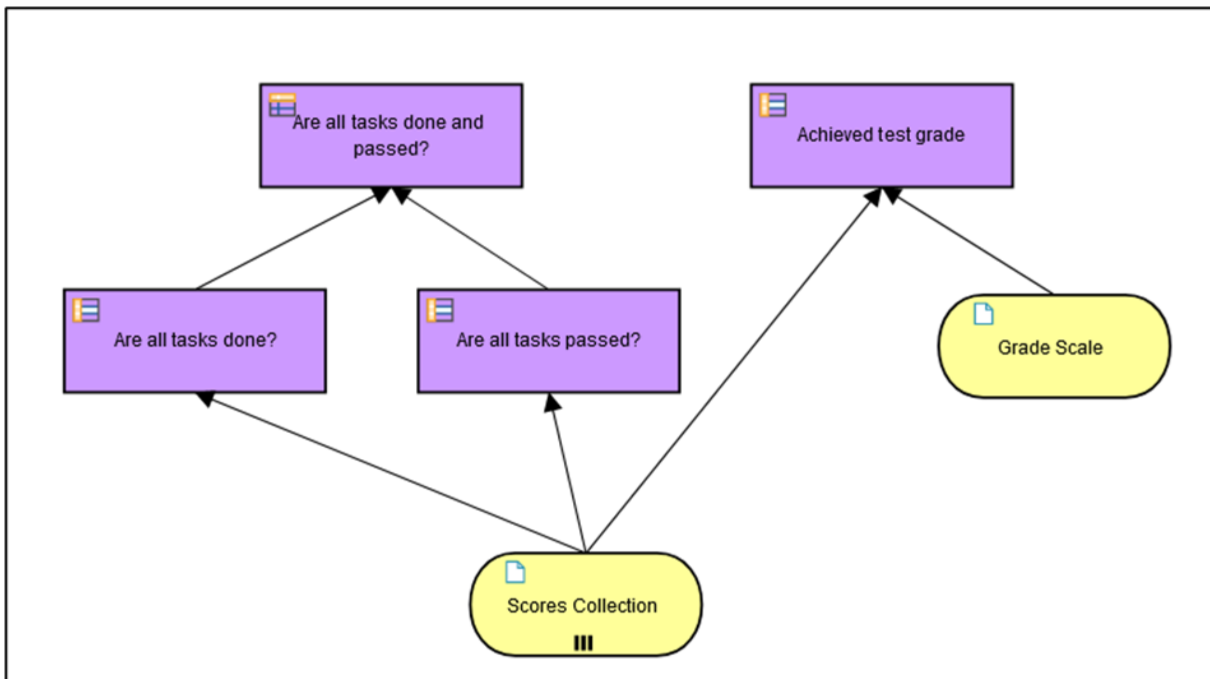
#	Context entry	Expression
1	Total score of the test	`sum(Scores Collection.Score.maxScore)`
2	Total achieved score on the test	`sum(Scores Collection.Score.achievedScore)`
3	Achieved test percentage	`round up(100 * (Total achieved score on the test / Total score of the test))`
		`Achieved test percentage`

Step 11.1: Test the DL

Use [25,50,30]. The achieved test percentage is 60.

Step 11.2: Infer a Letter Grade from the Achieved Test Percentage

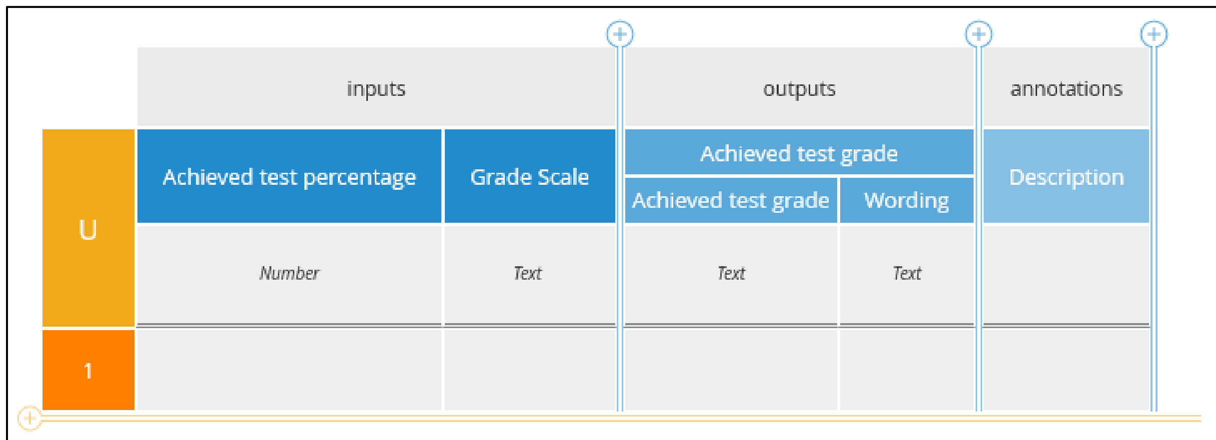
Here we could create a new decision that infers the grade from the achieved test percentage calculated by the preceding decision. But we will not do that. Instead, we will edit the DL above to decide on the letter grade with the contexts already in the DL. First, we need to add the 'Grade Scale' input to the decision and *change the name* of the decision to reflect what is decided. We do not need to update the Q&A to match the added logic, since we obviously wanted this decision from the start.



Click on the DL symbol in the upper left corner of the decision and add a new row named “Achieved test grade”. Specify the DL as a **Decision Table.** Change the final output from “Achieved test percentage” to “Achieved test grade”.

You should now have the following:

Set up the decision table to look like this:



Click the *Text* attribute of the ‘Grade Scale’ header and set its **Constraints** to **Enumeration**.

Grade Scale

Editing Header Type

Type: Text

Constraints: None

None
Expression
Enumeration

sum(Scores Collection.Score.achievedScore)

round up(100 * (Total achieved score on the test / Total score of the test))

Add the enum values and close the dialogue.

Grade Scale

Editing Header Type

Type: Text

Constraint type: Enumeration

Values:
"UA"
"UV"
"1-9"

Allow null

Create Type Close

Edit the **Decision Table** to look like in the picture on the page 62. Before you go there, merge input cells with equal content. Select the cells to be merged and merge them:

chieved test percentage	Grade Scale	Achieved score percentage Number	Test gr Text
>= 85	"UA"	Achieved test percentage	"A"
[75..84]	"UA"	Achieved test percentage	"B"
[65..74]	"UA"	Achieved test percentage	"C"
[55..64]	"UA"	Achieved test percentage	"D"
[50..54]	"UA	entage	"E"

Merge

The cells will be merged, and it increases the readability.

Achieved test percentage	Grade Scale	Achieved score percentage Number	Test g Te
>= 85	"UA"	Achieved test percentage	"A
[75..84]		Achieved test percentage	"B
[65..74]		Achieved test percentage	"C
[55..64]		Achieved test percentage	"D
[50..54]		Achieved test percentage	"E

It is probably also easier to read if the merged cells are to the left, so put the mouse over the header of the column, press down the left mouse button, and drag the column to the far left. The ‘Achieved test percentage’ and the ‘Grade Scale’ columns should swap places.

Achieved test grade

1	Total score of the test	sum(Scores Collection.Score.maxScore)
2	Total achieved score on the test	sum(Scores Collection.Score.achievedScore)
3	Achieved test percentage *Number*	round up(100 * (Total achieved score on the test / Total score of the test))
4	Achieved test grade	(decision table below)

	inputs		outputs		annotations
U	Grade Scale	Achieved test percentage	Achieved test grade: Achieved test grade	Achieved test grade: Wording	Description
	Text "UA", "UV", "1-9"	*Number*	*Text*	*Text*	
1	"UA"	>= 85	"A"	"Excellent"	
2		[75..84]	"B"	"Very Good"	
3		[65..74]	"C"	"Good"	
4		[55..64]	"D"	"Fair"	
5		[50..54]	"E"	"Satisfactory"	
6	"UV"	>= 80	"VG"	"Excellent"	
7		[50..79]	"G"	"Good"	
8	"1-9"	>= 90	"9"	"Excellent"	
9		[80..89]	"8"	"Very Good"	
10		[70..79]	"7"	"Good"	
11		[60..69]	"6"	"Fair"	
12		[50..59]	"5"	"Satisfactory"	
13	-	< 50	"U"	"Fail"	

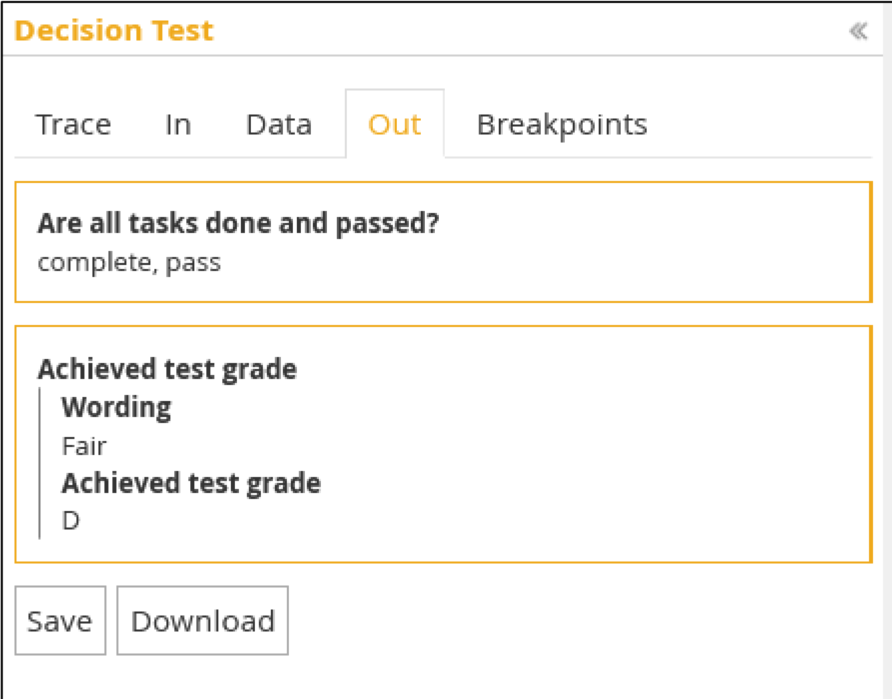
Achieved test grade

Save your model.

Step 11.3: Test Your Decision

Use the same test data as above and “UA” as grade scale.

[25,50,30]. The achieved test percentage is 60.



It worked as intended. It is probably a good idea to output the score percentage too, so let us edit the decision table and add a new output 'Achieved score percentage' as in the next picture.

Achieved test grade
Any

#	Variable	Expression
1	Total score of the test	`sum(Scores Collection.Score.maxScore)`
2	Total achieved score on the test	`sum(Scores Collection.Score.achievedScore)`
3	Achieved test percentage *Number*	`round up(100 * (Total achieved score on the test / Total score of the test))`
4	Achieved test grade	(decision table below)

U	inputs		outputs			annotations
	Grade Scale	Achieved test percentage	Achieved test grade			Description
			Achieved score percentage	Test grade	Wording	
	Text "UA", "UV", "1-9"	*Number*	*Number*	*Text*	*Text*	
1	"UA"	>= 85	Achieved test percentage	"A"	"Excellent"	
2		[75..84]	Achieved test percentage	"B"	"Very Good"	
3		[65..74]	Achieved test percentage	"C"	"Good"	
4		[55..64]	Achieved test percentage	"D"	"Fair"	
5		[50..54]	Achieved test percentage	"E"	"Satisfactory"	
6	"UV"	>= 80	Achieved test percentage	"VG"	"Excellent"	
7		[50..79]	Achieved test percentage	"G"	"Good"	
8	"1-9"	>= 90	Achieved test percentage	"9"	"Excellent"	
9		[80..89]	Achieved test percentage	"8"	"Very Good"	
10		[70..79]	Achieved test percentage	"7"	"Good"	
11		[60..69]	Achieved test percentage	"6"	"Fair"	
12		[50..59]	Achieved test percentage	"5"	"Satisfactory"	
13	-	< 50	Achieved test percentage	"U"	"Fail"	

`Achieved test grade`

Run again [25,50,30]. The achieved test percentage is 60.

Decision Test

Trace In Data Out Breakpoints

Are all tasks done and passed?
complete, pass

Achieved test grade
Wording
Fair
Achieved score percentage
60
Achieved test grade
D

Save Download

Try with [[25,50,30],[25,50,20]]

Decision Test

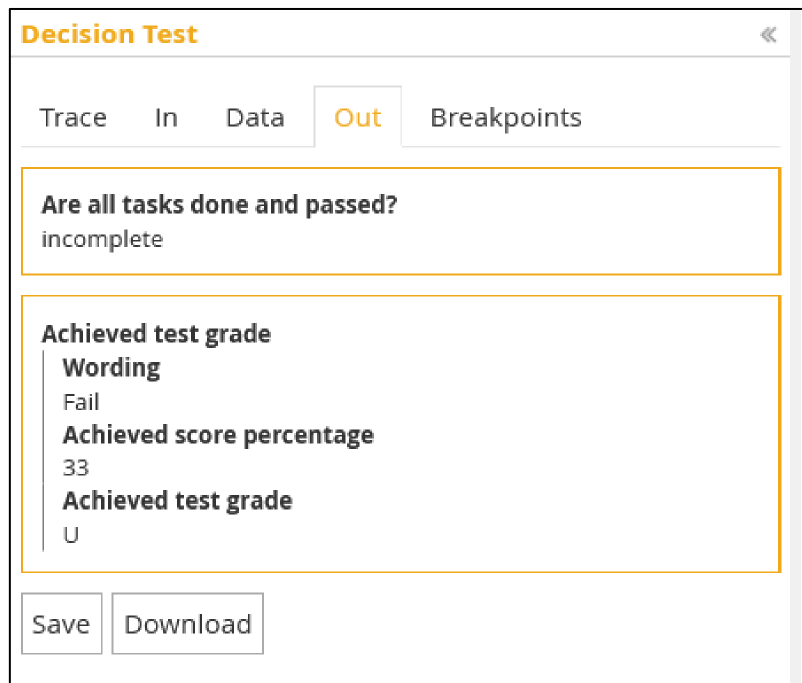
Trace In Data Out Breakpoints

Are all tasks done and passed?
complete, fail

Achieved test grade
Wording
Satisfactory
Achieved score percentage
50
Achieved test grade
E

Save Download

Try with [[25,50,30],[25,50,25],[25,50,-1]]



Test other values to see what happens.

As you can see, the two decisions are not integrated. You are not supposed to get e.g., grade "E" when another decision is "complete, fail". Hence, we need to change our model to work the right way.

Step 12: Awarded grade

In this step you will complete the decision model.

Drag a **Decision** symbol from the palette and place it at the top of the present model. Name it "Awarded Grade" and change its colour to purple. Connect it to the 'Are all tasks done and passed?' and 'Achieved test grade' decisions. These decisions will precede the 'Awarded Grade' decision. Specify the Q&A of the decision.

Awarded grade

×

Question and Allowed Answers

Question

What is the final grade awarded to a student for a test? It is based on preceding decisions on score percentage to grade, if the student has completed all the tasks in the test, and if the student has passed the completed tasks.

Allowed Answers

The outcome of the preceding decision on score to grade.

☐ **Use Data Output Type as allowed answers**

Close

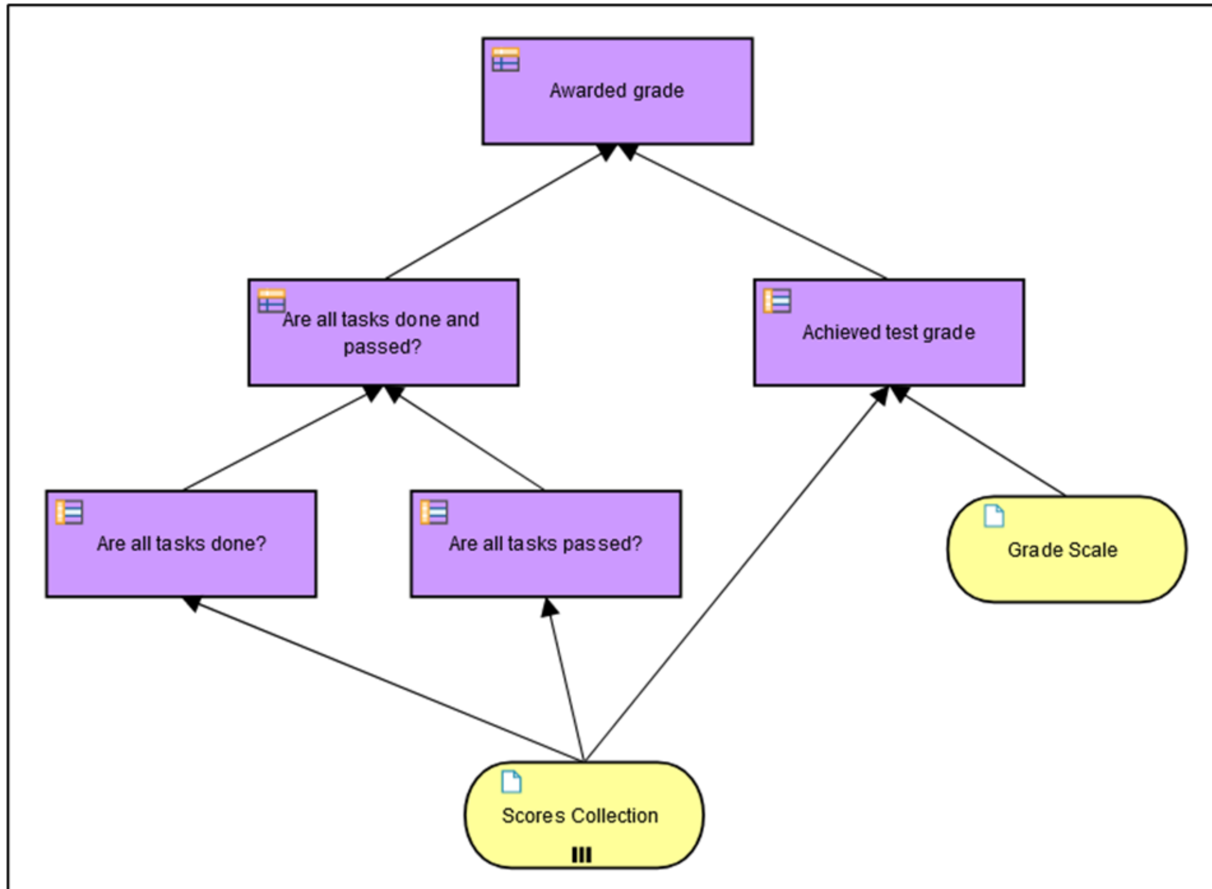
Open the dialog for decision logic of the ‘Awarded Grade’ decision. Edit the **Decision Table** to look like in the picture below.

Awarded grade
Any

	inputs	outputs	annotations
U	Are all tasks done and passed? *Text* *"complete, pass", "complete, fail", "incomplete"*	Awarded grade *Any*	Description
1	"complete, pass"	Achieved test grade	
2	"complete, fail"	Achieved test grade	
3	"incomplete"	"Not graded"	

Step 13: Test Your Decision

Now the decision model is complete!



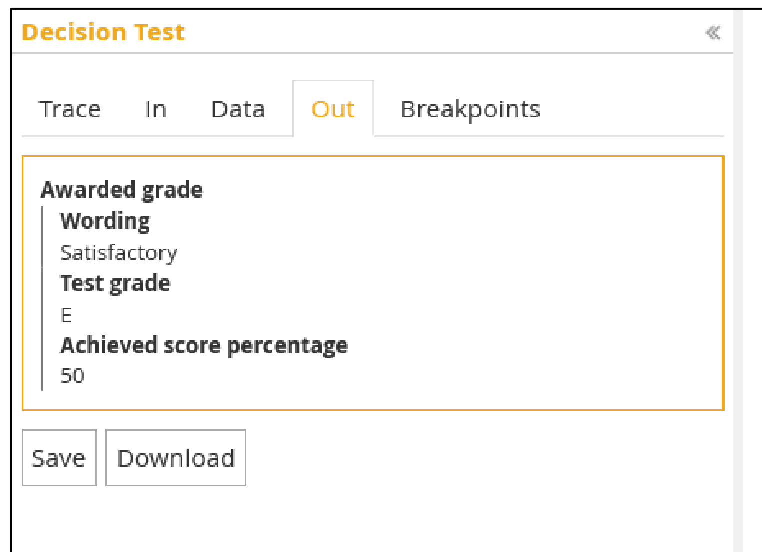
Test the model in **Execution, Test** mode using the same values as above.

It seems to work!

Change the score on one of the tasks to be below the pass limit score: [[25,50,30],[25,50,25],[25,50,20]]

Awarded grade
Any

U	Are all tasks done and passed?	Awarded grade	Description
	Text *"complete, pass", "complete, fail", "incomplete"*	*Any*	
1	"complete, pass"	Achieved test grade	
2	"complete, fail"	Achieved test grade	
3	"incomplete"	"Not graded"	



That did not work! You should not get “complete, fail” and “E” at the same time. The error is in row two, column two of the decision table. ‘Are all tasks done and passed?’ results in “complete, fail” while ‘Achieved test grade’ in ‘Awarded grade’ results in “E”.

The ‘Achieved test grade’ decision only derives a letter grade based on the score percentage and grade scale without caring about whether all required tasks are passed. This is tested in another branch of the DRD.

The upside of this is that we can use the ‘Achieved test grade’ decision as a general decision for all tests with total score in other workflows. It could possibly be turned into a **BKM**. The downside is that we do not only get the ‘Achieved score percentage’ from ‘Achieved test grade’ but all outputs. Maybe this can be controlled somehow, but I could not find out how.

Given this, we need to change the top-level decision logic. To make the output clearer we also add an ‘Explanation’ output.

Awarded grade

U	inputs	outputs		annotations
	Are all tasks done and passed?	Awarded grade		Description
		Awarded grade	Explanation	
	Text *"complete, pass", "complete, fail", "incomplete"*	*Any*	*Text*	
1	"complete, pass"	Achieved test grade		
2	"complete, fail"	"U"	"Some tasks are mandatory and have a pass score greater than zero. The score achieved for at least one task in the test is below the pass score of that task. All mandatory tasks need to be passed to pass the test, even if the achieved total score is at least equal to the pass score of the full test"	
3	"incomplete"	"Not graded"	"Only tests where all included tasks are done are graded. At least one task in the test is not done and the test is thus incomplete."	

When we run the same test as above, we get the following:

Decision Test

Trace In Data Out Breakpoints

Awarded grade

Awarded grade

U

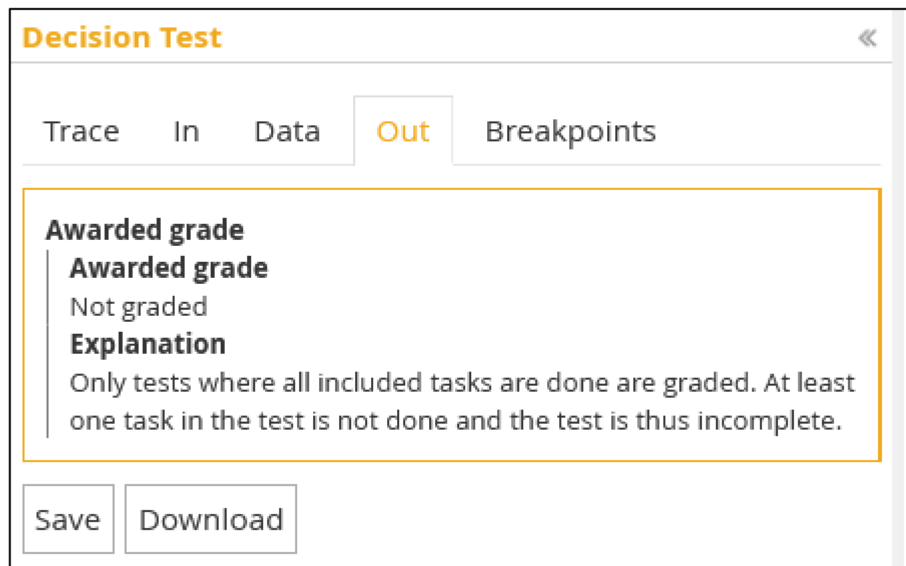
Explanation

Some tasks are mandatory and have a pass score greater than zero. The score achieved for at least one task in the test is below the pass score of that task. All mandatory tasks need to be passed to pass the test, even if the achieved total score is at least equal to the pass score of the full test

Save Download

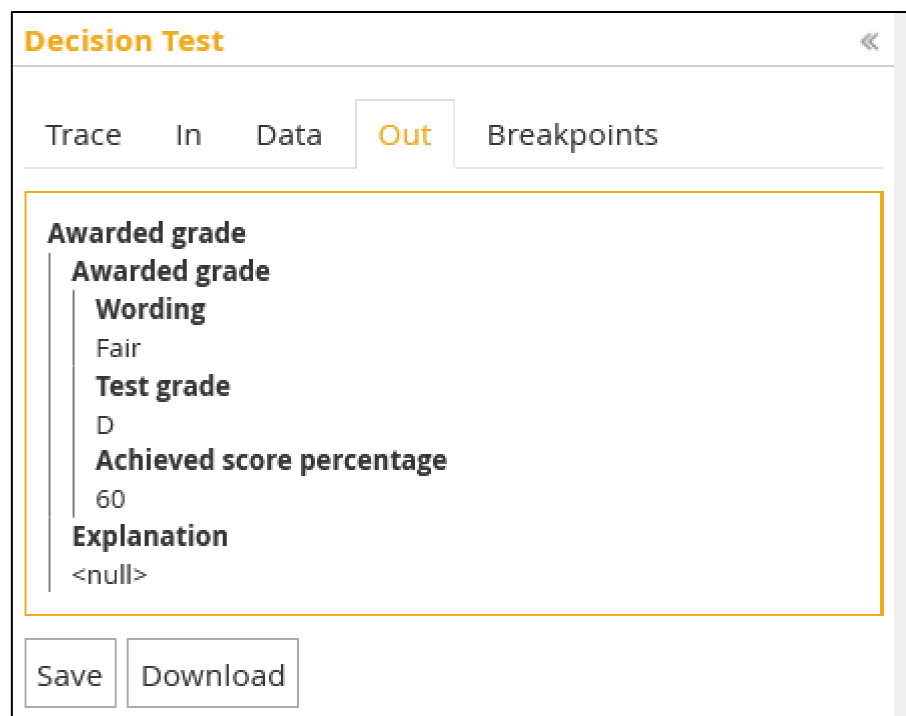
That works fine!

Try with [[25,50,30],[25,50,25],[25,50,-1]]



Works as it should!

Try with [[25,50,30],[25,50,25],[25,50,35]]



Try other different combination of input data and vary the number of score triplets, their scores, and the grade scales.

Step 14: Add Knowledge Sources to the Model

The decisions in the model are based on policies at the school and department. One very nice and important feature of the DMN standard is that this can be recorded in the DRD itself. It is called a **Knowledge Source.**

Drag two **Knowledge Source** symbols onto the canvas.

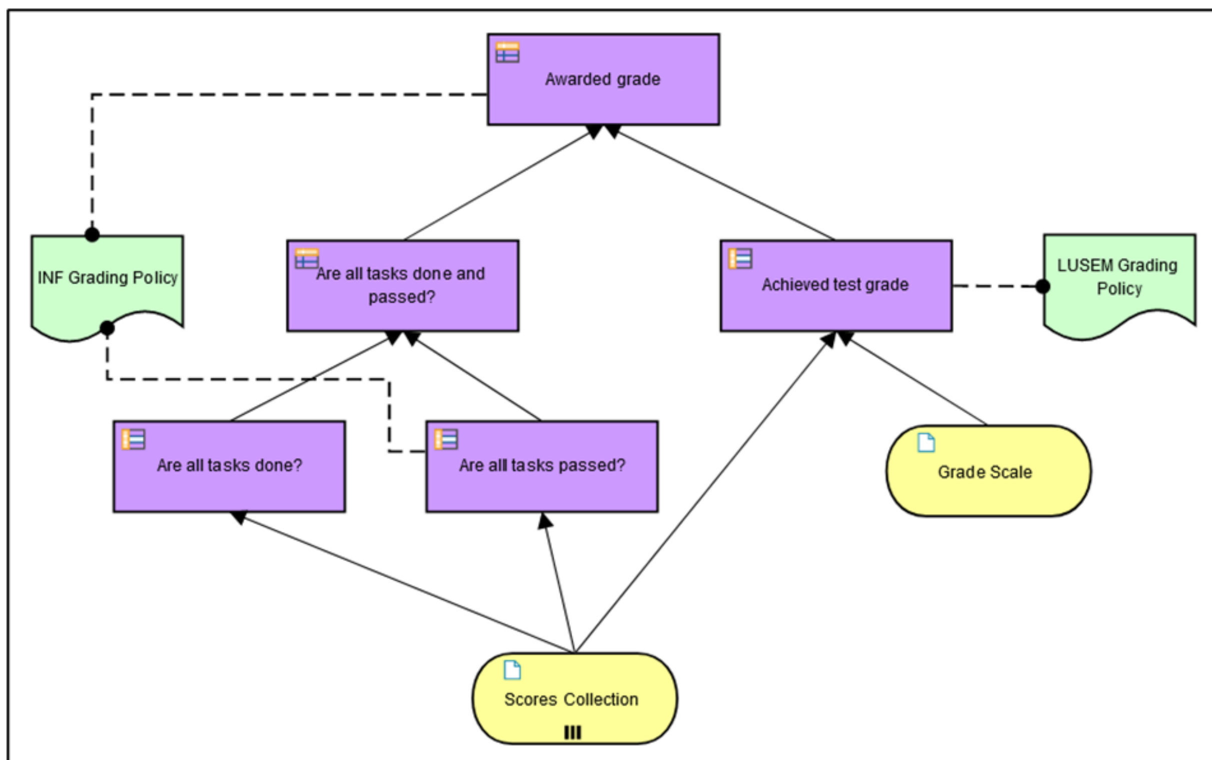
One should be to the left and be named “INF Grading Policy“. The other to the right and be named “LUSEM Grading Policy”.

Change the colour of the **Knowledge Source** symbols to light green.

Connect the ‘INF Grading Policy’ knowledge source to both the ‘Awarded grade’ and the ‘Are all tasks passed?’ decisions.

Connect the ‘LUSEM Grading Policy’ knowledge source to the ‘Achieved test grade’ decision.

When you are done your model should look like in the picture below:

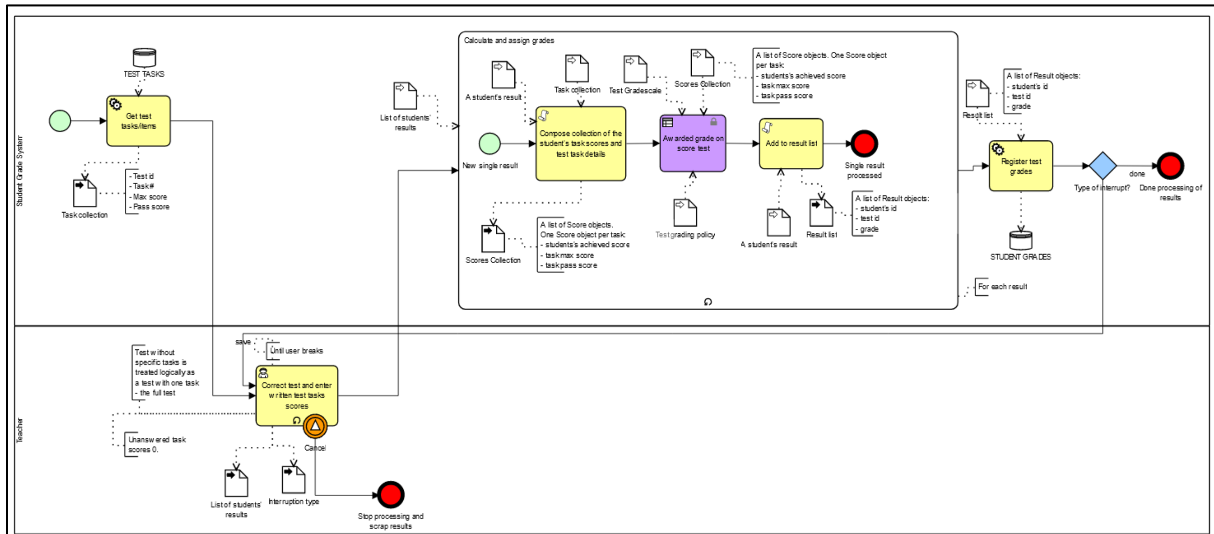


Part IV

Connecting Decisions and Workflow

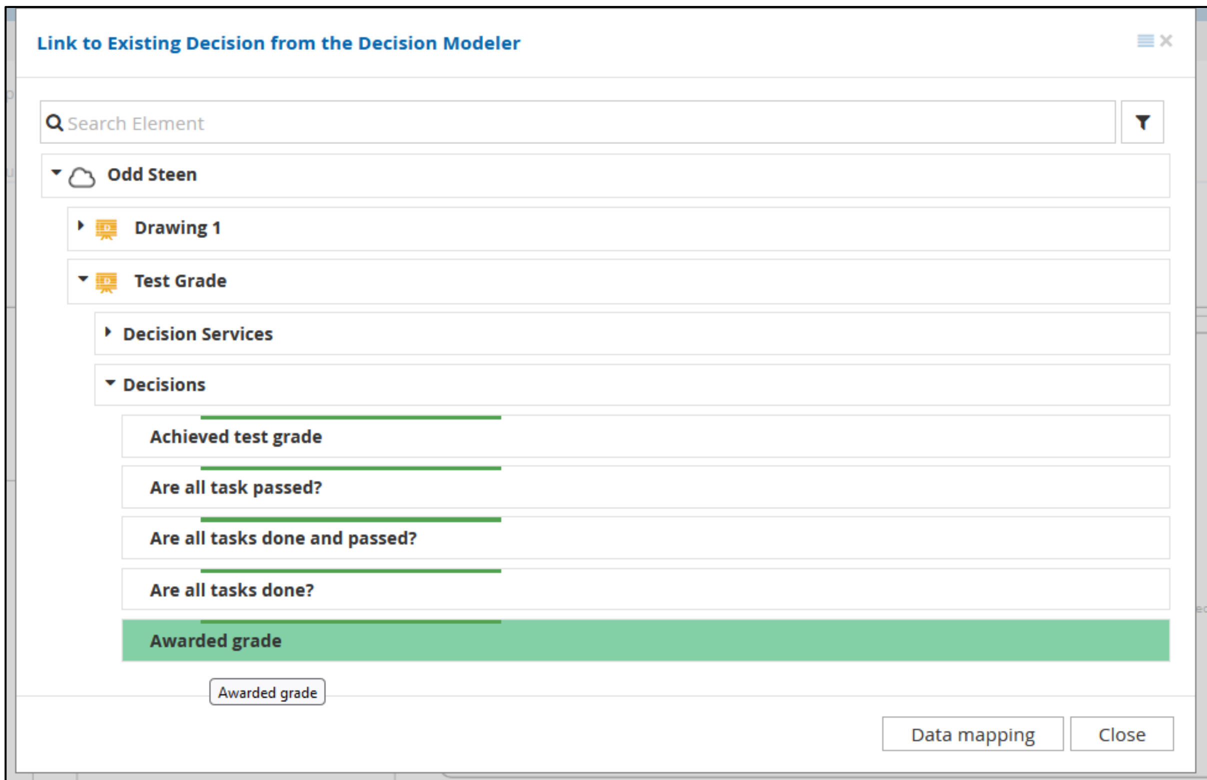
Finally, we need to connect our decisions to the workflow we modelled before.

The workflow should look like in the picture below.

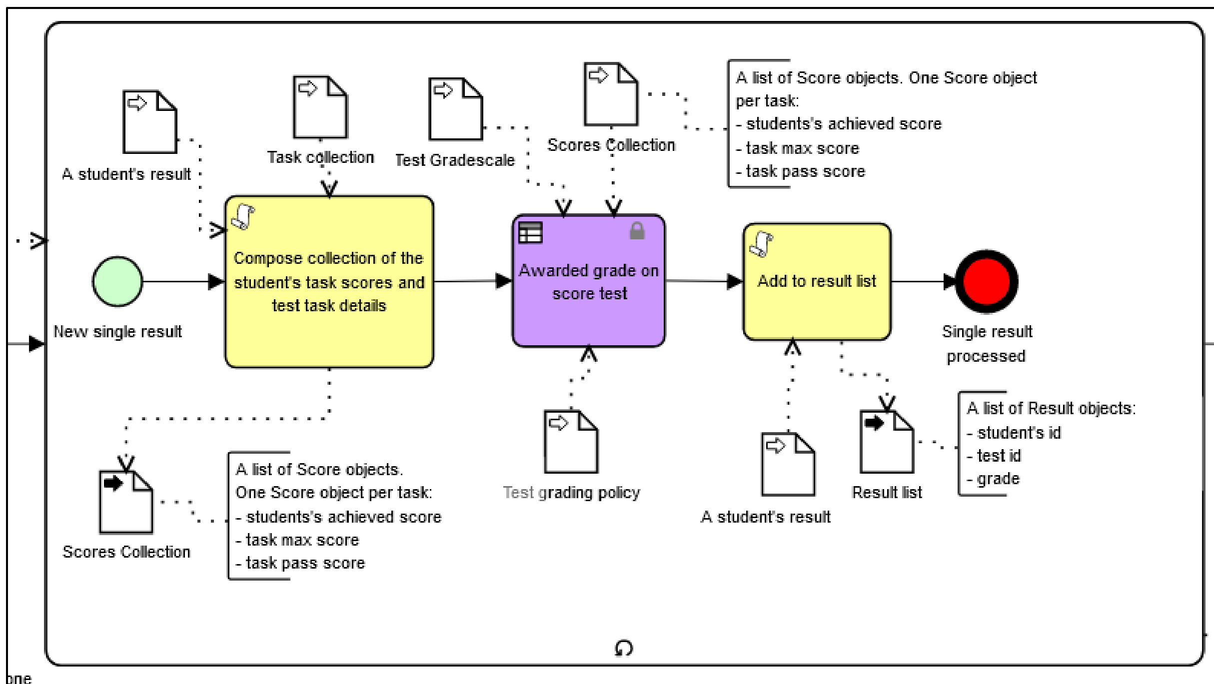


Step 1: Connect Decision Tasks in BPM with Decisions in DMN

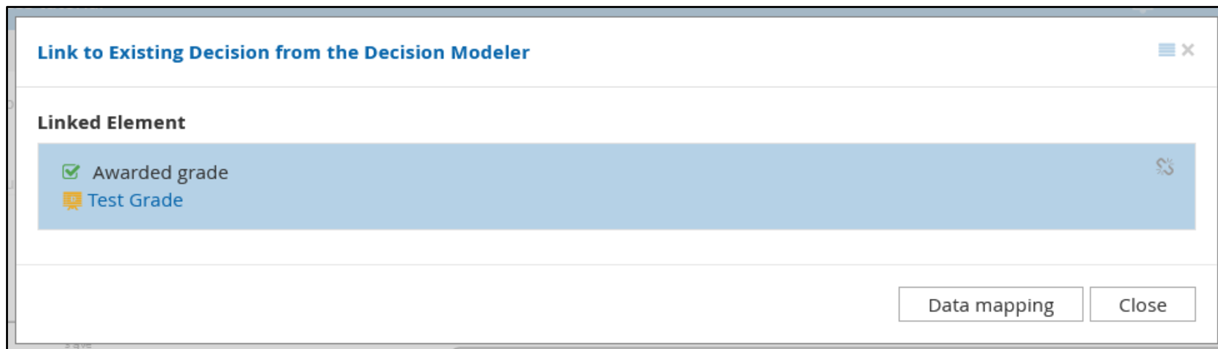
If you click on the table-like symbol in the top left corner of the 'Decide written test grade with total score' rules task a pop-up window will show which diagram that is linked to the task.



Navigate to your top-level decision for awarding a grade on at test with max score and pass scores. Select that decision and click **Close.** As you can see, the name of the rules task was changed to the name of the decision you linked.



When you click on the table-like symbol in the rule task you should see this:



Step 2: End

Congratulations! You are now done with the tutorial.

References

Object Management Group. (2013). *Business Process Model and Notation (BPMN) version 2.0.2*. Retrieved from: https://www.omg.org/spec/BPMN/2.0.2/PDF

Silver, B. (2011). *BPMN Method and Style: With BPMN Implementer's Guide*. Cody-Cassidy Press.