2. Case description

when arrived... or may go further to another Task...

How other things are created, linked ?

This MLM challenge involves representing universal properties of process types along w[illegible] types, actor types and their various [illegible] (Section 2.2), and an application of [illegible] in the scope of a particular software engineering process (Section 2.3). Submitted solutions should include bottom-level instances, at least for key types, exemplifying all attributes mentioned in the challenge description. Deviations from the case as described here should be documented in submissions. The case description may be extended but respective rationales should then be provided.

Hence there is a perform relationship

2.1. Overview

Does that mean there is an actual duration ? i.e a start time ?

all instances should be critical or some of them? I choose the

Model de Fahad ?

Process management is charac[illegible] of rules concerning the executio[illegible] cesses, tasks, actions or activities[illegible] ulating and keeping track of processes, i.e. the enactments of process types, with such pro[illegible] referred to as "process instances[illegible] literature. For example, in the s[illegible] it may be necessary to keep tr[illegible]ting, e.g., the fact whether or not code has been *tested*, etc. Further rules impo[illegible] the participation of business actors (hu[illegible] and artifacts (equipment, documents, to[illegible] and specifies dependen[illegible] the software engineering domain: (i[illegible] t[illegible] *sign*; (ii) *test case desi[illegible]er[illegible]* employs a *requiremen[illegible]* results in *test cases*; and (iii) *testing* is performed by a *tester*, employs *test cases*, and produces a *test* [illegible]

P10: so many interpretations!

P11-P1x reduce interpretation scope...

P11:task instance!

In other contexts, such as [illegible] may be a need to keep track of [illegible] *ted an insurance claim*, when it [illegible] *analyst authorized payment* o[illegible] response to the claim, how muc[illegible]

ambiguous. p16/p14. exists (weak-prefered) or all (strong) or in between ?

Any constraints on TT-auth- A et TT -auth- A?

Submissions to the challenge should focus on the software engineering domain. They may optionally include the insurance domain as well. In the following, we are using the insurance domain for illustrative purposes only.

2.2. Processes, tasks, actors and artifacts

The following general rules pertaining to processes, tasks, actors and artifacts apply for the challenge:

P1) A *process type* (such as *claim handling*) is defined by the composition of one or more *task types* (*receive claim*, *assess claim*, *pay premium*) and their relations.

P2) Ordering constraints between *task types* of a *process type* are established through *gateways*, which may be *sequencing*, *and-split*, *or-split*, *and-join* and *or-join*.

[illegible]*cess type* has one *initial task type* (with which all [illegible]ecutions begin), and one or more *final task types* (with which all its executions end).

P4) Each *task type* is *created by* an *actor*, who will not necessarily perform it. For example, *Ben Boss created* the task type *assess claim*.

P5) For each *task type*, one may stipulate a set of *actor types* whose instances are the only ones that may *perform* instances of that *task type*. For example, in the *XSure* insurance company, only a *claim handling manager* or a *financial officer* may *authorize payments*.

see S1

P6) A *task type* may alternatively be assigned to a particular set of *actors* who are authorized (e.g., *John Smith* and *Paul Alter* may be the only *actors* who are allowed to *assess claims*).

see S13

P9:how? use/ produce /other ?

P7) For each *task type* (such as *authorize payment*) one may stipulate the *artifact types* which are *used* and *produced*. For example, *assess claim* uses a *claim* and produces a *claim payment decision*.

P8) *Task types* have an *expected duration* (which is not necessarily respected in particular occurrences).

P9) *Critical task types* are those whose instances are *critical tasks*; each of the latter must be *performed* by a *senior actor* and the artifacts they produce must be associated with a *validation task*.

Not arte fact type

P9:an insta nce or a type ?

P10) Each *process type* may be enacted multiple times.

P11) Each *process* comprises one or more *tasks*.

P12) Each *task* has a *begin date* and an *end date*. (e.g., *Assessing Claim 123* has *begin date 01-Jan-19* and *end date 02-Jan-19*).

P13) *Tasks* are associated with *artifacts used* and *produced*, along with *performing actors*.

P14) Every *artifact used* or *produced* in a *task* must instantiate one of the *artifact types* stipulated for the *task type*.

P15) An *actor* may have more than one *actor type* (e.g., *Senior Manager* and *Project Leader*.)

P16) Likewise, an *artifact* may have more than one *artifact type*.

P17) An *actor* who *performs* a *task* must be authorized for that task. Typically, a class of actors is automatically authorized for certain classes of tasks.

class = type or a new concept ? Choice ==

P18) *Actor types* may *specialize* other *actor types* in which case all the rules that apply to instances of the specialized *actor type* must apply to instances of the specializing *actor type*. For example, if a *manager* is allowed to *perform tasks* of a certain *task type*, so is a *senior manager*.

P19) All modeling elements, at all levels, must have a *last updated* value of type *time stamp*. This feature should be defined as few times as possible, ideally only once. Respective definitions are exempt from the requirement to have a *last updated* value. Note th[illegible] differs from the respective version i[illegible]

??

names are impicit see S0

Note that it is not necessary for every *type* in the model to have an instance. It is useful, however, to illustrate the design with a number of instances.

In the world, in the company, in the team? Lets interpret in the known system.

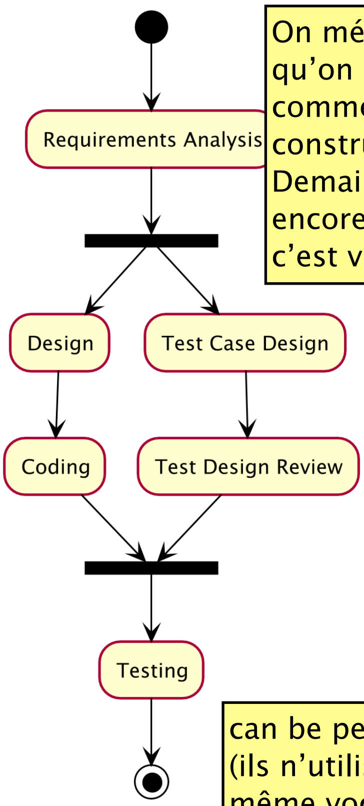
2.3. Software engineering process

An application of the above described process management must be defined to cap

A single instance of Ann Smith exists (I know many of thems)

of software engineering proc *Software Development Compar* *opment process* is composed o *sign*, *coding*, *test case design*, *test design review* and *testing* (conforming to the constraints indicated i the bars represent an *and-split* and an *an*

défini comment?



On mélange... ce qu'on dévrit et comment c'est construit. Demain, ce sera encore vrai ? ou c'est vrai now !

S0

can be performed (ils n'utilisent pas le même vocabulaire.)

Figure 1. The Acme software engineering process.

The following rules for the software engineering domain apply:

S1) A *requirements analysis* is *performed by* an *analyst* and *produces* a *requirements specification*.

S2) A *test case design* is *performed by* a *developer* or *test designer* and *produces test cases*. Note that this requirement from [3] conflicts with S13. We have maintained it here for the record but ask submitters to let the information in S13 override S2, i.e. only *senior analysts* may perform a *test case design*. Note that *test case designs* still *produce test cases*.

S3) An occurrence of *coding* is *performed by* a *developer* and *produces code*. It must furthermore reference one or more *programming languages* employed.

It = code artefact or coding task?

Task! see S4

S4) *Code* must reference the *programming language(s)* in which it was *written*.

S5) *Coding in COBOL* always produces *COBOL code*.

S6) All *COBOL code* is written in *COBOL*.

S7) *Ann Smith* is a *developer*; she is the only one allowed to perform *coding in COBOL*.

S8) *Testing* is *performed by* a *tester* and *produces* a *test report*.

S9) Each *tested artifact* must be associated to its *test report*.

forever, as soon as it exists?

S10) *Software engineering artifacts* have a responsible *actor* and a *version number*. This applies to *requirements specification*, *code*, *test case*, *test report*, but also to any future types of *software engineering artifacts*.

S11) *Bob Brown* is an *analyst* and *tester*. He has *created* all *task types* in this *software development process*.

S12) The *expected duration* of *testing* is *9 days*.

S13) *Designing test cases* is a *critical task* which must be performed by a *senior analyst*. *Test cases* must be validated by a *test design review*.

3. Solution presentation requirements

Submissions responding to the challenge should describe a multi-level model conforming to the case description, including justifications for non-trivial design decisions. In order to foster comparability between solutions, respondents are asked to make sure that concepts of the case description are explicitly represented by one or more model elements. Conformance of the model elements to each of the requirements (P1–P19 and S1–13) must be documented in a dedicated section of the article.

3.1. Mandatory discussion aspects

Challenge respondents must discuss their multilevel model solution with regard to the following aspects, each of which should be treated in a specific sub-section of the "Assessment" section of the article:

- **Basic modeling constructs**: Explain the basic modeling constructs used in the solution.
- **Levels (or other model content organization schemes employed)**: Explain the nature of "levels" in the model, how model elements are arranged on these levels and which relationships (such as "instance-of") may feature between elements at different levels. The nature of levels should be captured by explicitly stating the level segregation and the level cohesion principles used [5]. Avoid vague language such as "higher level concepts are more abstract" if the inter-level relationship is more specific. If the inter-level relationship is deliberately allowed to be vague, state this explicitly.
- **Number of levels**: Elaborate whether the submitted solution could have had more or fewer levels and explain how any potentially existing degrees of freedom were resolved.
- **Cross-level relationships**: Discuss if and how associations and links can connect model elements at different levels. State well-formedness constraints, if any apply.

1. As mentioned before, an urance domain is purely optional.