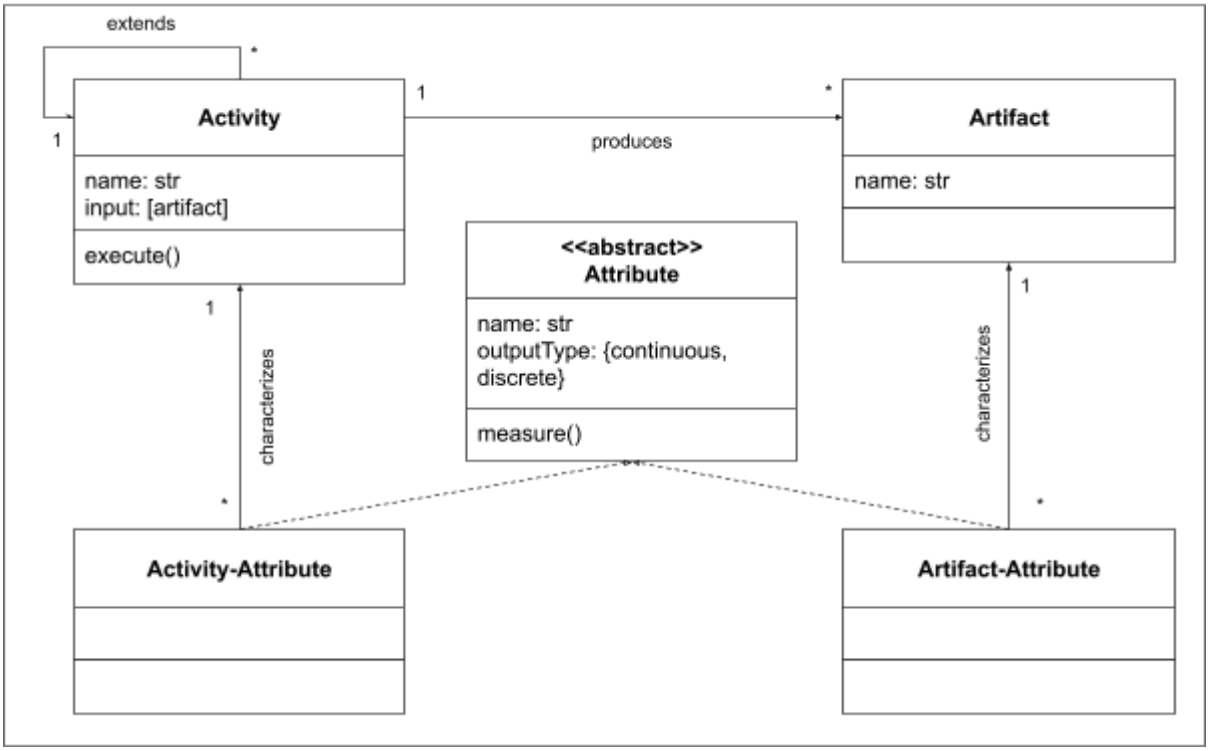
Requirements-affected Activities & their Attributes

Metamodel and Coding Guideline

This document specifies the meta-model of the model under development and the extraction guideline, i.e., the means of development.

Overview

The following class diagram visualizes the meta-model.



Activity

An activity is a software engineering process that considers some input artifacts and produces some output artifacts. If one of the input artifacts is a requirements specification, the activity is requirements-affected.

Coding activities

The extractions of experimental tasks, interview statements about requirements use, and software process steps involving requirements are coded to each represent one activity.

We code each activity how its source describes it. This means we do not differentiate if an activity conflates multiple activities. For example, if a source describes an experiment investigating participants ability of **understanding** a requirement by letting them specify a model from the requirement, we code it just as **understanding**, not as understanding *and* specifying. This conflation has implications on the measurement (i.e., the activity of understanding is impossible to distinguish from specifying) which will be discussed at another time.

Structure

Properties

An activity has the following properties.

Property	Type	Description	Example
name	string	Identifier of the activity	implementing, comprehending
input	list of artifacts	Artifacts necessary to perform the activity	[requirements specification, architecture]
produces	list of artifacts	Artifacts produced when executing the activity	[code]

Methods

An activity has the following methods.

Method	Description
execute()	Processes the input artifacts and produces the output artifacts.

Attribute

An attribute of an activity is a property of the activity that informs about its success. Attributes can be variables used in key performance indicators (KPIs) and are interesting when evaluating how well an activity was performed.

Coding attributes

We code the extractions of dependent variables, KPIs of reported requirements reuse, and performance indicators of software process steps as attributes.

Types of attributes

An attribute represents a property of either an activity, the artifact produced by the activity, or both. Consider the following examples:

- Property of an activity: if an experiment measures the time it takes participants to complete the task, this is the **duration** attribute of the activity.
- Property of an artifact: if an experiment measures the redundancy of produced models, this is how **redundant** the artifact is.
- Properties of activities and artifacts: if an experiment evaluates the implementing activity by how well the resulting implementation covers the requirements used as an input, it characterizes how **complete** the implementing activity is and how **valid** the resulting implementation artifact is.

A few studies make this distinction explicit (e.g., E41, E171)

Exclusion of non-valuating attributes

The attributes of relevance for the final model must be *valuating*, i.e., a value of the attribute must inform about the quality of the activity. This usually means that the two extremes of the range of possible values of an attribute are associated to *good* or *bad* quality. For example, for the duration attribute, lower values are good (i.e., the activity is quick) and higher values are bad (i.e., the activity takes a long time). While it is **not** necessary that values are precisely associated to levels of quality (i.e., how many minutes of duration may an activity take to still be considered quick), the sign of a difference should have a clear implication (i.e., more minutes is worse, less minutes is better).
This necessitates the exclusion of non-valuating attributes, i.e., attributes whos values do not inform about the quality of their respective activity. Two criteria disqualify attributes as non-valuating:

V1. Attributes that are not compared to a ground truth. An attribute for which a ground truth exists (e.g., effort estimation, for which the actual effort is known) can be valuated based on how close the attribute value is to the ground truth. If no such ground truth exists or is used, then the attribute cannot make a claim about quality.
V2. Attributes with no equivalent to established qualities (e.g., compared to the items in ISO/IEC 25010). Even without a clear ground truth, attributes that resemble quality attributes from ISO/IEC 25010 (e.g., duration resembles time behavior, correctness resembles functional correctness, etc.) have an established association between the sign of changes of their value (more/less) and quality (e.g., shorter time behavior is good, higher functional correctness is good).

Consider the following example: for the activity of estimating implementation effort based on a requirement [E252]:

- The dependent variable **precision** (i.e., distance between estimated and actual effort) is an attribute, because the higher the precision the better.
- The dependent variable **estimated effort** is not an attribute, because different values of effort do not entail good or bad.

A workaround that some experimental studies found towards the first exclusion criterion is to use the standard deviation of attribute values of the experimental sample (i.e., the variation of effort estimation values) rather than the absolute values as a dependent variable. In this case, the attribute avoids the exclusion criterion because there now is a clear optimum (i.e., a low standard deviation means that the activity of estimating effort is precise).

Structure

Properties

An attribute has the following properties.

Property	Type	Description	Example
name	string	Identifier of the attribute	duration, complete, correct
outputType	{continuous, discrete}	Data type of the scale onto which the attribute maps the activity	

Methods

An attribute has the following methods.

Method	Description
measure()	Quantify the activity in regard to the attribute and produce a measurement on the output scale.