Table 4 lists some of the JUnit tests that are available.

**NOTE:** The AMIL database is cleared before each test beings.

Table 4. Available JUnit tests

|  |  |
| --- | --- |
| **Test** | **Description** |
| MetricsTestBase.java | This tests getting the existing AMILInterface for the localhost. If the interface does not exist it tests creating one. |
| TestBBNMetrics.java | Tests executing the BBN Metric. |
| TestComplexityAMILmetric.java | This could be considered a gold standard metric test as it tests the simple graph based metric AMILComplexity which counts nodes and edges. |
| TestCostDesignConceptMetric | This tests the procurementCost metric which implements the binning of components into categories of cost by weight for different categories as an AMIL. |
| TestDesignConceptVariationMetric.java | Executing the DesignConcept metric, which is the DesignCurvature metric entirely implemented in AMIL. This was the ArrowWebServices implemented version capturing the MaxTorque |
| TestDesignCurvatureMetric.java | This explores the DesignCurvature metric, which was one of our early explorations at evoking metrics in AMIL. This metric was originally based on the Requirements Space Curvature metric of the Phase1A Final Report page 53, but then simplified to measure parameter sensitivity. Vary the input parameters of an executable node according to (exhaustive, search, sampling algorithm} and calculate a metric of how outputs change in relation to how much the inputs were changed. These changes can multidimentional (below test). This metric is small when larger inputs produce little change in outputs, and large when small changes in inputs produce large changes in outputs. |
| TestDesignCurvatureMetricMultiDim.java | See above description. Note that the curse of dimensionality results in exponential growth of execution time since all possible parameter combinations over the number of steps are considered. |
| TestEvaluateNodesByAttribute.java | This tests the EvaluateTransport metric which extracts and assesses transports from the SYSML requirements. |
| TestExtractValueMeasureMetric.java | This tests extracting a value from an AMIL node and summing it as the EstimateWeightMeasure metric. |
| TestPARCMetrics.java | Runs the PARC envisioner PCC metric calculation. |
| TestRefineArchetypeMetric.java | The RefineArchetype metric is looking for the “best fit” SteeringSuspensionTrack type that meets the requirements for the “NGV”. It pulls requirement data from a feature node (“SteeringSuspensionTrack”) and compares to required value names of a targetNode (“NGV”) graph to determines if each lower and upper feature value meet the requirement, producing a score, which is minimized to determine the best reasoned component (see the SYSML section). |
| TestRobustZScore.java | This tests the RobustZScore metric which consists of dual evaluators, a measure, and a statistic. |
| TestWeightMetric.java | The EstimateWeight metric sums the weight values gleaned from the connected nodes to the targetNode. It is currently configured to use the demo model but it can also be configured to sum the weights off of the ECTO model by 1) commenting out the clear AMIL database of this test driver. 2) Changing the fields in the demo.AMILmetrics.weight.txt file to: "targetNode":"IFV", "ltypesOfInterest":"[hasA]", and  "fieldsOfInterest":"[mass]". |