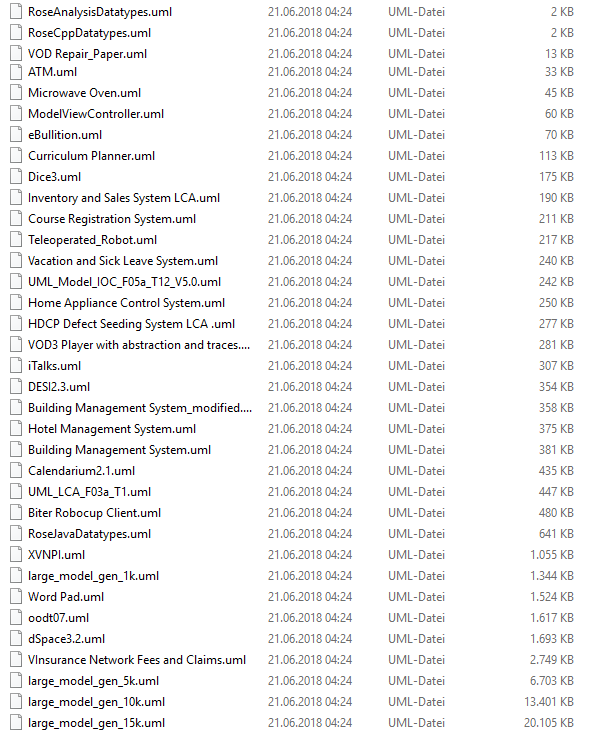
***Models 2019 – Evaluation Data***

**Selected Consistency Rules**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Rule** | **Context** | **G/L** |
| CR1 | **"Every lifeline has to have a corresponding class";** | uml:Lifeline | G |
| CR2 | **"Every transition has to have a corresponding message"** | uml:Transition | G |
| CR3 | **“Statechart Action must be defined as an Operation in the Owner’s Class”** | uml:Transition | G |
| CR4 | **"Message Action must be defined as an Operation in Receiver’s Class"** | uml:Message | G |
| CR5 | **"Operation Parameters must have unique Names"** | uml:Operation | L |
| CR6 | **"An Operation has at most one return Parameter"** | uml:Operation | L |
| CR7 | **“An interface can have at most one generalization”** | uml:Interface | L |
| CR8 | **"An Interface can only contain Public Operations and no Attributes"** | uml:Interface | L |
| CR9 | **"No two Class Operations may have the same Signature"** | uml:Class | L |
| CR10 | **"Parent Class should not have an Attribute referring to a Child Class"** | uml:Class | L |
| CR10 | **“No two fields may have the same name”** | uml:Class | L |

**Projects sorted by size**



**Select 10!**

***Models 2019 – Evaluation Process***

**Goal**

We show that both a local and a global view on engineering artifacts can display X rule evaluations, whereas a timestamped local and global view can show Y(-z) rule evaluations, based on 10 consistency rules in 10 different models with W to V model elements.

**Approach**

The following steps are initiated for the evaluation

* Initialize typed artifacts for model elements.
* Step by step instantiate model elements from chosen files. This build up simulates changes! Mock-Up user class “feeds” changes (change state increment) to workspaces and commits them after a single model element (e.g. an entire class) is finished.
* For multiple tool simulation: Each user is responsible for one diagram type.
* For single tool simulation: Model Elements are split up between users.
* On each relevant change the consistency checker re-evaluates, logs the change state, and increases the amount of total rule evaluations in each workspace.
* So in the end each workspace will have a total amount of rule evaluations run for it.
* This scenario is run through for each view on each file

We only count the amount of Rule Evaluation artifacts in each scenario! Whether these rule evaluations hold or not is irrelevant since in each complementation with additional timestamped artifacts may introduce or remove inconsistencies. So the amount of broken or held consistency rules does not add any sort of information. The only thing that is interesting is the amount of potential evaluation a certain view on the data can provide.

**Final Tables**

**Model Elements**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Total** | **Classes** | **Interfaces** | **Operations** | **Message** | **Transition** | **Lifelines** | **Other** |
| **M1** | M1(T) | M1(C) | M1(I) | M1(O) | M1(M) | M1(T) | M1(L) | M1(Ot) |
| **M2** | M2(T) | M2(C) | M2(I) | M2(O) | M2(M) | M2(T) | M2(L) | M2(Ot) |
| **….** | … | … | … | … | … | … | … | … |

**Number of Evaluations**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Local + Tstmp** | **Local** | **Global + Tstmp** | **Global** |
| **M1** | M1(X) | M1(X’) | M1(Y) | M1(Y’) |
| **M2** | M2(X) | M2(X’) | M2(Y) | M2(Y’) |
| **….** | … | … | … | … |

**Discussion?**

**Sources:**

https://github.com/jku-isse/modelanalyzer/blob/master/test/src/at/jku/isse/modelanalyzer/test/util/DesignRules.java

|  |  |  |
| --- | --- | --- |
| **Rule** | **CT** | **Inter/Intra** |
| |  | | --- | |  | |  |  |  | | --- | | String dr = "let children:Set(NamedElement) = self.namespace.oclAsType(Package).packagedElement->select(pe:PackageableElement|pe.oclIsTypeOf(Class )and pe.oclAsType(Class).allParents()->includes(self)) in self.ownedAttribute->forAll(p:Property|p.type.oclIsTypeOf(Class) implies children->excludes(p.type.oclAsType(Class )))"; | |  |  |  | | --- | | String desc = "**Parent Class should not have an Attribute referring to a Child Class**"; | |  |  |  | | --- | | String name = "DR1"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Class"), name, dr, desc, repairable, true, | |  |   dataModel)); | Class | Model |
| |  | | --- | | dr = "let children:Set(NamedElement) = self.namespace.oclAsType(Package).packagedElement->select(pe:PackageableElement|pe.oclIsTypeOf(Class) and pe.oclAsType(Class).allParents()->includes(self)) in self.ownedOperation->forAll(o:Operation|o.ownedParameter->forAll(p:Parameter|p.type.oclIsTypeOf(Class) implies children->excludes(p.type.oclAsType(Class))))"; | |  |  |  | | --- | | desc = "**Parent Class should not have a Method with a Parameter referring to a Child Class**"; | |  |  |  | | --- | | name = "DR2"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Class"), name, dr, desc, repairable, true, | |  |  |  | | --- | | dataModel)); | |  | |  | | Class | Model |
| |  | | --- | | String dr3 = "self.receiveEvent.oclAsType(InteractionFragment).covered->forAll(represents.type.oclAsType(Class).ownedOperation->exists(name=self.name))"; | |  |  |  | | --- | | desc = **"Message Action must be defined as an Operation in Receiver’s Class"**; | |  |  |  | | --- | | name = "DR3"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Message"), name, dr3, desc, repairable, | |  |   true, dataModel)); | Message | Inter |
| |  | | --- | | String dr4 = "self.receiveEvent.oclAsType(InteractionFragment).covered->exists(let rc:Class=represents.type.oclAsType(Class) in self.sendEvent.oclAsType(InteractionFragment).covered-> exists (let sc:Class=represents.type.oclAsType(Class) in sc.ownedAttribute->exists(association <> null implies type=rc)))"; | |  |  |  | | --- | | desc = "**Message Direction must match Class Association**"; | |  |  |  | | --- | | name = "DR4"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Message"), name, dr4, desc, repairable, | |  |   true, dataModel)); | Message | Model |
| |  | | --- | | dr = "self.memberEnd <> null and self.memberEnd->forAll(p | p.type <> null and p.type.namespace = self.namespace)"; | |  |  |  | | --- | | desc = "**The connected Classifier of the Association End should be included in the Namespace of the Association**"; | |  |  |  | | --- | | name = "DR5"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Association"), name, dr, desc, repairable, | |  |   true, dataModel)); | Association | Model |
| |  | | --- | | dr = "self.memberEnd->forAll(p1,p2 : Property | p1 <> p2 implies p1.name <> p2.name)"; | |  |  |  | | --- | | desc = "**AssociationEnds must have unique Names within the Association**"; | |  |  |  | | --- | | name = "DR6"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Association"), name, dr, desc, repairable, | |  |   true, dataModel)); | Association | Model |
| |  | | --- | | dr = "self.memberEnd->size() > 0 implies self.memberEnd->select(p | p.aggregation <> AggregationKind::none )->size() <= 1"; | |  |  |  | | --- | | desc = "**At most one AssociationEnd may be an Aggregation or Composition**"; | |  |  |  | | --- | | name = "DR7"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Association"), name, dr, desc, repairable, | |  |   true, dataModel)); | Association | Model |
| |  | | --- | | dr = "self.allParents()->forAll(c:Classifier | c.oclAsType(Class).ownedAttribute->forAll(p:Property | p.class.ownedAttribute->collect(name)->excludes(p.name)))"; | |  |  |  | | --- | | desc = "**A Classifier may not declare an Attribute that has been declared in Parent Classifiers**"; | |  |  |  | | --- | | name = "DR8"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Class"), name, dr, desc, repairable, true, | |  |   dataModel)); | Class | Model |
| |  | | --- | |  | |  |  |  | | --- | | dr = "self.ownedAttribute->forAll(p1,p2:Property | p1 <> p2 implies p1.name <> p2.name)"; | |  |  |  | | --- | | desc = "**A Class may use Unique Attribute Names**"; | |  |  |  | | --- | | name = "DR9"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Class"), name, dr, desc, repairable, true, | |  |   dataModel)); | Class | Model |
| |  | | --- | | dr = "(self.association <> null and self.aggregation=AggregationKind::composite) implies (self.upper >= 0 and self.upper <= 1)"; | |  |  |  | | --- | | desc = "**A Classifier may not belong by Composition to more than one Composite Classifier**"; | |  |  |  | | --- | | name = "DR10"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Property"), name, dr, desc, repairable, | |  |   true, dataModel)); | Property | Model |
| |  | | --- | | dr = "self.packagedElement->forAll(e1,e2 : PackageableElement | (e1 <> e2) implies ( e1.name <> e2.name))"; | |  |  |  | | --- | | desc = "**A Classifier may not belong by Composition to more than one Composite Classifier**"; | |  |  |  | | --- | | name = "DR11"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Package"), name, dr, desc, repairable, | |  |   true, dataModel)); | Package | Model |
| |  | | --- | | dr = "self.ownedAttribute->forAll(pr : Property | pr.association <> null or pr.visibility = VisibilityKind::public) and self.ownedOperation->forAll(o : Operation | o.visibility = VisibilityKind::public)"; | |  |  |  | | --- | | desc = "**An Interface can only contain Public Operations and no Attributes**"; | |  |  |  | | --- | | name = "DR12"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Interface"), name, dr, desc, repairable, | |  |   true, dataModel)); | Interface | Model |
| |  | | --- | | dr = "self.ownedOperation->forAll( o1,o2 : Operation | o1 <> o2 implies (o1.name <> o2.name or o1.ownedParameter->size() <> o2.ownedParameter->size() or let ops1 : Collection(Type) = o1.ownedParameter->collect(type) in let ops2 : Collection(Type) = o2.ownedParameter->collect(type) in ops1->exists(t : Type | ops2->excludes(t)) or ops2->exists(t:Type | ops1->excludes(t))))"; | |  |  |  | | --- | | desc = "**No two Class Operations may have the same Signature**"; | |  |  |  | | --- | | name = "DR13"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Class"), name, dr, desc, repairable, true, | |  |   dataModel)); | Class | Model |
| |  | | --- | |  | |  |  |  | | --- | | dr = "self.ownedParameter->forAll(p1,p2 : Parameter | p1 <> p2 implies p1.name <> p2.name)"; | |  |  |  | | --- | | desc = "**Operation Parameters must have unique Names**"; | |  |  |  | | --- | | name = "DR14"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Operation"), name, dr, desc, repairable, | |  |   true, dataModel)); | Operation | Model |
| |  | | --- | | dr = "self.ownedParameter->forAll(p : Parameter | p.type <> null implies p.type.namespace = self.owner.oclAsType( Class ).namespace)"; | |  |  |  | | --- | | desc = "**The Type of Operation Parameters must be included in the Namespace of the Operation Owner**"; | |  |  |  | | --- | | name = "DR15"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Operation"), name, dr, desc, repairable, | |  |   true, dataModel)); | Operation | Model |
| |  | | --- | | dr = "self.source->forAll(e1: Element | e1.oclIsKindOf( NamedElement ) implies self.target->forAll(e2 : Element | e2.oclIsKindOf( NamedElement ) and e1.oclAsType(NamedElement).namespace = e2.oclAsType(NamedElement ).namespace))"; | |  |  |  | | --- | | desc = "**The Parent must be included in the Namespace of the Generalizable Element**"; | |  |  |  | | --- | | name = "DR16"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Generalization"), name, dr, desc, | |  |   repairable, true, dataModel)); | Generalization | Model |
| |  | | --- | |  | |  |  |  | | --- | | dr = "not self.allParents()->includes(self)"; | |  |  |  | | --- | | desc = "**No circular Inheritance allowed**"; | |  |  |  | | --- | | name = "DR17"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Class"), name, dr, desc, repairable, true, | |  |   dataModel)); | Class | Model |
| |  | | --- | | dr = "self.owner.oclAsType( Region ).stateMachine <> null implies let classifier : BehavioredClassifier=self.owner.oclAsType (Region).stateMachine.context in classifier.oclIsTypeOf( Class ) implies classifier.oclAsType( Class ).ownedOperation->exists( o : Operation | o.name = self.name)"; | |  |  |  | | --- | | desc = "**Statechart Action must be defined as an Operation in the Owner’s Class**"; | |  |  |  | | --- | | name = "DR18"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Transition"), name, dr, desc, repairable, | |  |   true, dataModel)); | Transition | Inter |
| |  | | --- | | dr = "self.ownedParameter->select(p : Parameter | p.direction = ParameterDirectionKind::return)->size() <= 1"; | |  |  |  | | --- | | desc = "**An Operation has at most one return Parameter**"; | |  |  |  | | --- | | name = "DR19"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Operation"), name, dr, desc, repairable, | |  |   true, dataModel)); | Operation | Model |
| |  | | --- | |  | |  |  |  | | --- | | String dr20 = "self.interaction.package.packagedElement->exists(p|p.oclIsTypeOf(Class) and p.oclAsType(Class).name = self.name)"; | |  |  |  | | --- | | desc = **"Every lifeline has to have a corresponding class"**; | |  |  |  | | --- | | name = "DR20"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Lifeline"), name, dr20, desc, repairable, | |  |  |  | | --- | | true, dataModel)); | |  | | Lifeline | Inter |
| |  | | --- | |  | |  |  |  | | --- | | String dr21 = "self.coveredBy->asSequence()->first().oclAsType(Class).ownedBehavior->asSequence()->first().oclAsType(StateMachine).region->asSequence()->first().transition->forAll(t:Transition | self.coveredBy->exists(m|m.oclIsTypeOf(MessageOccurrenceSpecification) implies m.oclAsType(MessageOccurrenceSpecification).message.name = t.name))"; | |  |  |  | | --- | | desc = **"Every transition has to have a corresponding message";** | |  |  |  | | --- | | name = "DR21"; | |  |  |  | | --- | | rules.put(name, RuleDefinition.readDesignRule(Util.getContextElement("Lifeline"), name, dr21, desc, repairable, | |  |   true, dataModel)); | Lifeline | Inter |