Guide to understanding

Mistake Detection System

by Prabhsimran Singh

Topics Covered

- What is Mistake Detection System?

- How does it work?

- Which Mistakes it covers?

- A Poster explaining the system

- How is the system tested?

- Understanding the test cases.
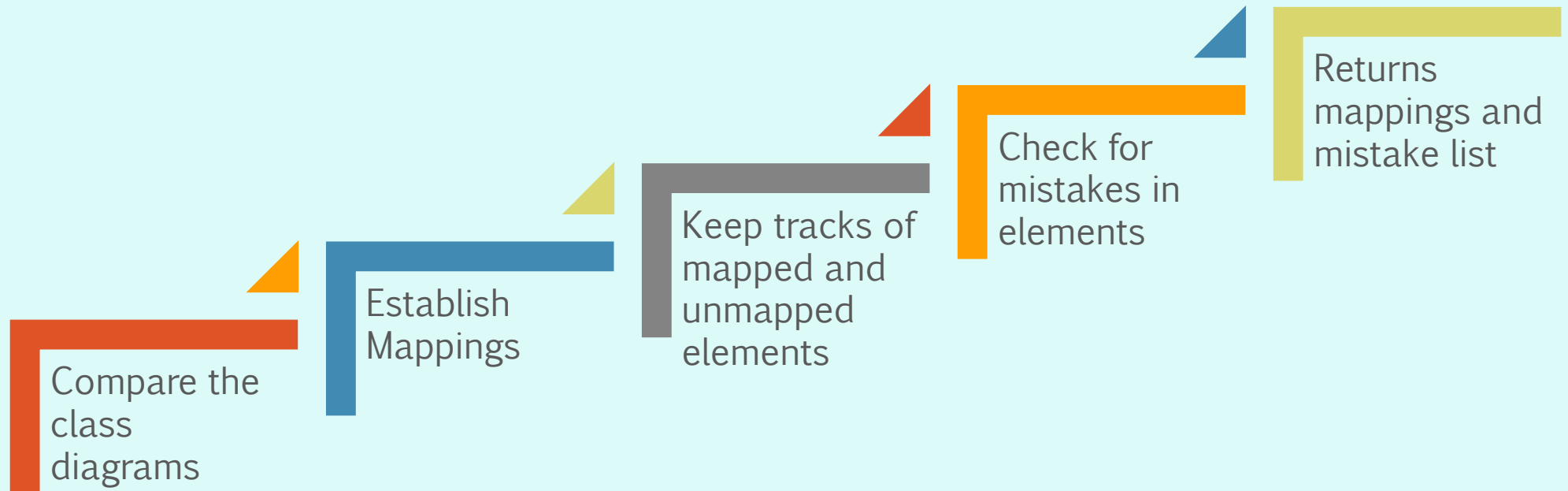
- Writing a sample test.

What is Mistake Detection System?

- The Mistake Detection System automates the process of detecting modelling mistakes in domain models by comparing student solutions with a correct solution.

- The automated feedback provided by the mistake detection system allows a larger number of students to receive valuable feedback compared to a manual approach, which is constrained by the availability of teaching staff.

How does it work?

Compare the class diagrams

Establish Mappings

Keep tracks of mapped and unmapped elements

Check for mistakes in elements

Returns mappings and mistake list
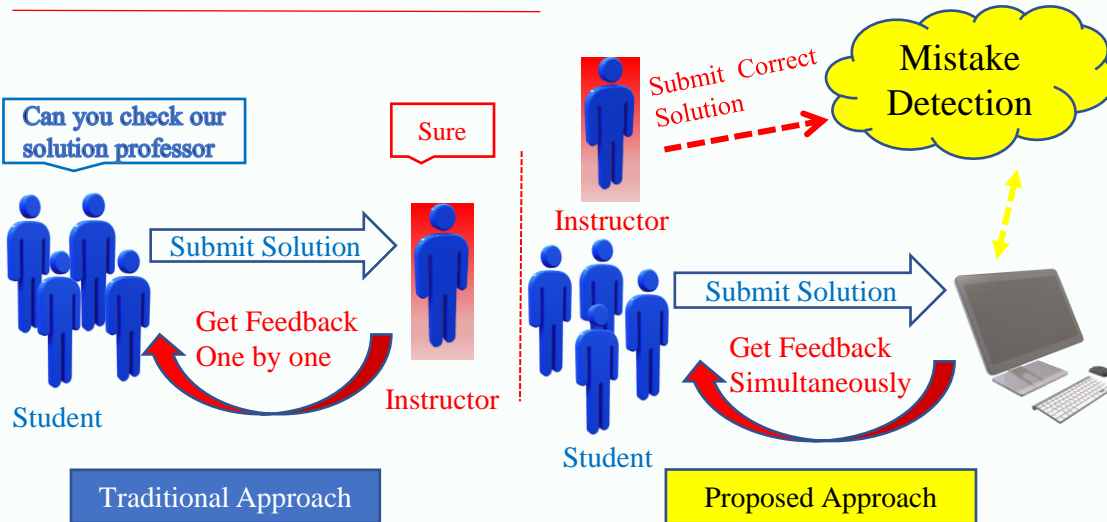
Which Mistakes it covers?

- It will cover 53 different types of Mistakes.

- At present code for 12 mistakes have been written and tested.

 (be aware this number will increase as I work more ☺)

Domain Modelling Mistake Detection System

Prabhsimran Singh, Gunter Mussbacher, McGill University, Montreal, Canada
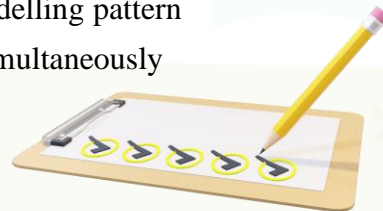
Introduction

Knowledge about the system's problem domain is crucial for successful system development. Hence, domain modelling with class diagrams is an important Requirement Engineering activity, which is routinely taught in undergraduate and graduate programs. The proposed system automates the process of detecting modelling mistakes in domain models by comparing student solutions with a correct solution.
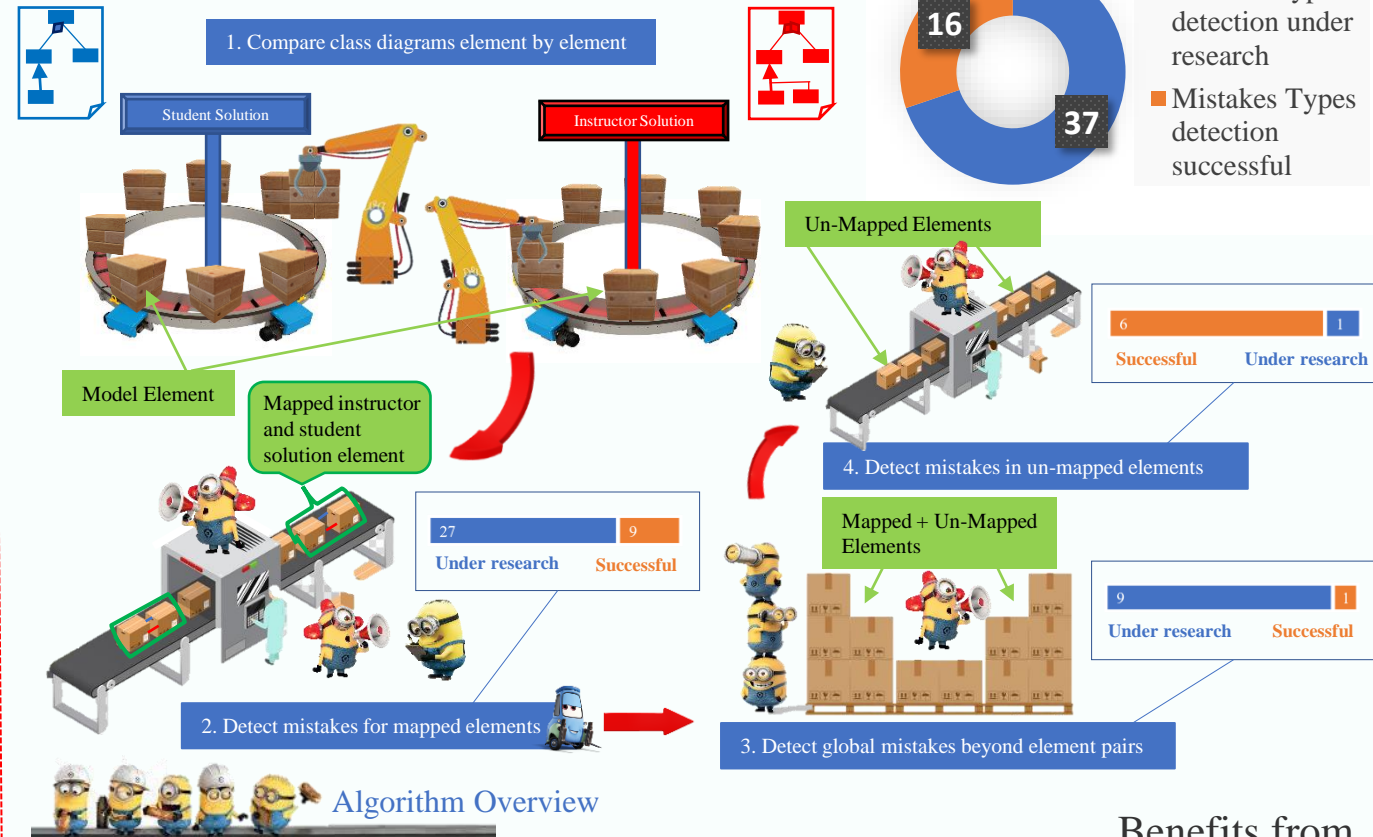
Problem Statement

Can you check our solution professor

Sure

Submit Correct Solution

Mistake Detection

Student

Submit Solution

Instructor

Get Feedback One by one

Instructor

Traditional Approach

Student

Submit Solution

Get Feedback Simultaneously

Proposed Approach

Main Goals and Objectives

✓ Detect different types of mistakes in class diagrams such as missing elements, incorrect elements, or incorrect application of modelling pattern

✓ Provide valuable feedback to multiple students simultaneously

✓ Automate the traditional approach

✓ Save student's and instructor's time

✓ Scale to large number of students

Current and upcoming state of the research project

1. Compare class diagrams element by element

Student Solution

Instructor Solution

Model Element

Mapped instructor and student solution element

Un-Mapped Elements

Successful 6   Under research 1

4. Detect mistakes in un-mapped elements

Mapped + Un-Mapped Elements

Under research 9   Successful 1

Under research 27   Successful 9

2. Detect mistakes for mapped elements

3. Detect global mistakes beyond element pairs

Algorithm Overview

53 Identified Mistake Types

16   37

- Mistake Type detection under research
- Mistakes Types detection successful

Relevance

Class diagrams are used to describe domain models that help analyze and specify problem domains and requirements. Consequently, modelling skills for class diagrams are a fundamental skill that needs to be learned by engineering students early on in their curriculum. This proposed approach aims to help students practice and improve their modelling skills through an improved learning experience. In addition, the catalog of common domain modelling mistakes may lead to a better understanding of domain modelling.

Benefits from participation

FEEDBACK

Testing of Mistake Detection
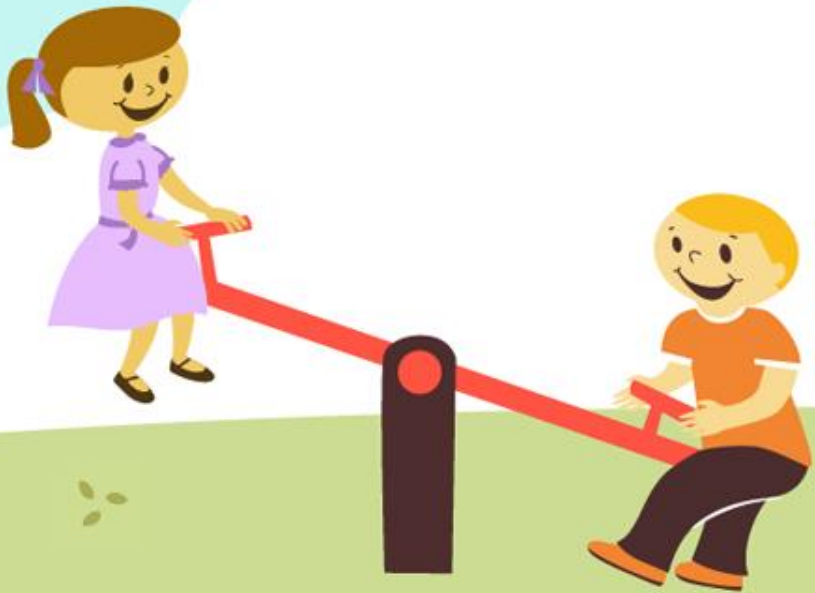
How is the system tested?

- To test the system multiple JUnit tests are created.

- Tests check the correctness of mapping, the number of mistakes, correctness of types of mistakes and the properties of every mistake.

Understanding the test

- In the next few slides will learn about the structure of tests

- Every test will have a body, some initializations, function call and assertions

Code to load cdm file

Name of instructor solution class diagram file

```java
@Test public void testMistakePluralClassName() {
    ClassdiagramPackage.eINSTANCE.eClass();
    var cdmFile = "../mistakedetection/testModels/InstructorSolution/flolderName/Instructor_MistakePluralClassName.domain_model.cdm";
    var resource = ResourceHelper.INSTANCE.loadResource(cdmFile);
    var classDiagram = (ClassDiagram) resource.getContents().get(0);
    var maf = ModelingassistantFactory.eINSTANCE;
    var modelingAssistant = maf.createModelingAssistant();
    var solution = maf.createSolution();
    solution.setModelingAssistant(modelingAssistant);
    solution.setClassDiagram(classDiagram);

    ClassdiagramPackage.eINSTANCE.eClass();
    var cdmFile1 = "../mistakedetection/testModels/InstructorSolution/flolderName/Student_MistakePluralClassName.domain_model.cdm";
    var resource1 = ResourceHelper.INSTANCE.loadResource(cdmFile1);
    var classDiagram1 = (ClassDiagram) resource1.getContents().get(0);
    var maf1 = ModelingassistantFactory.eINSTANCE;
    var modelingAssistant1 = maf1.createModelingAssistant();
    var solution1 = maf1.createSolution();
    solution1.setModelingAssistant(modelingAssistant1);
    solution1.setClassDiagram(classDiagram1);
    var student = maf1.createStudent();
    solution1.setStudent(student);
```

Relative path of cdm file

Similarly, load student solution class diagram file

In coming slides you will learn to create a cdm file!!!

Sample test structure (1/3)

Define the classifier and Attributes you created in cdm file

Initialize the defined variables

Similarly, repeat the same for student solution

```java
Classifier instructorTestClass = null;
Attribute instructorTestClassAttributeTestAttribute = null;

for (var c : classDiagram.getClasses()) {
  if ("Test".equals(c.getName())) {
    instructorTestClass = c;
    for (Attribute a : c.getAttributes()) {
      if ("testAttribute".equals(a.getName())) {
        instructorTestClassAttributeTestAttribute = a;
      }
    }
  }
}

Classifier studentTestClass = null;
Attribute studentTestClassAttributeTestAttribute = null;

for (var c : classDiagram.getClasses()) {
  if ("Tests".equals(c.getName())) {
    studentTestClass = c;
    for (Attribute a : c.getAttributes()) {
      if ("testAttribute".equals(a.getName())) {
        studentTestClassAttributeTestAttribute = a;
      }
    }
  }
}
```

Sample test structure (2/3)

Call the compare function of Mistake Detection System, it will return an object of comparison class which contains the hashmaps and lists

```
comparison = MistakeDetection.compare(solution, solution1);

assertEquals(comparison.notMappedInstructorClassifier.size(), 0);
assertEquals(comparison.extraStudentClassifier.size(), 0);
assertEquals(comparison.mappedClassifier.size(), 1);

assertEquals(comparison.mappedClassifier.get(instructorTestClass),studentTestClass);

assertEquals(comparison.notMappedInstructorAttribute.size(), 0);
assertEquals(comparison.extraStudentAttribute.size(), 0);
assertEquals(comparison.duplicateStudentAttribute.size(), 0);
assertEquals(comparison.mappedAttribute.size(), 1);

assertEquals(comparison.mappedAttribute.get(instructorTestClassAttributeteTestAttribute),
    studentTestClassAttributeteTestAttribute);

assertEquals(comparison.newMistakes.size(), 1);
assertEquals(solution1.getMistakes().size(), 1);

for (Mistake m : solution1.getMistakes()) {
  if (m.getMistakeType() == MistakeTypes.USING_PLURAL_OR_LOWERCASE
      && m.getStudentElements().get(0).getElement() == studentTestClass) {
    assertEquals(m.getStudentElements().get(0).getElement(), studentTestClass);
    assertEquals(m.getNumDetectionSinceResolved(), 0);
    assertEquals(m.getNumDetection(), 1);
    assertTrue(m.isResolved());
  }
}
```

These check the correctness of mapping in terms of classifier (class).

This check if correct elements are mapped

Similar in case of attributes

Asserting size of detected mistake and Mistakes introduced in class diagram

Check the correctness of mistake type and its solution element

Sample test structure (3/3)

```
assertEquals(comparison.notMappedInstructorClassifier.size(), 0);
assertEquals(comparison.extraStudentClassifier.size(), 0);
assertEquals(comparison.mappedClassifier.size(), 1);
```

Contains Classes existing in Instructor solution and not in student solution

Contains Classes existing in student solution and not in instructor solution

Contains Classes existing in both Instructor student solution

```
assertEquals(comparison.notMappedInstructorAttribute.size(), 0);
assertEquals(comparison.extraStudentAttribute.size(), 0);
assertEquals(comparison.duplicateStudentAttribute.size(), 0);
assertEquals(comparison.mappedAttribute.size(), 1);
```

Similar case with attribute hash maps

```
assertEquals(comparison.newMistakes.size(), 1);

assertEquals(solution1.getMistakes().size(), 1);
```

Number of mistakes returned by MDS

Number of mistakes associated with a student solution

Explanation of Hash maps and lists used in previous slides

Writing the test

- In the next few slides will learn to write a sample test

- You will also learn to create a cdm file using TouchCORE

This is the home screen of TouchCORE

Step 1: Create a cdm using TouchCORE.
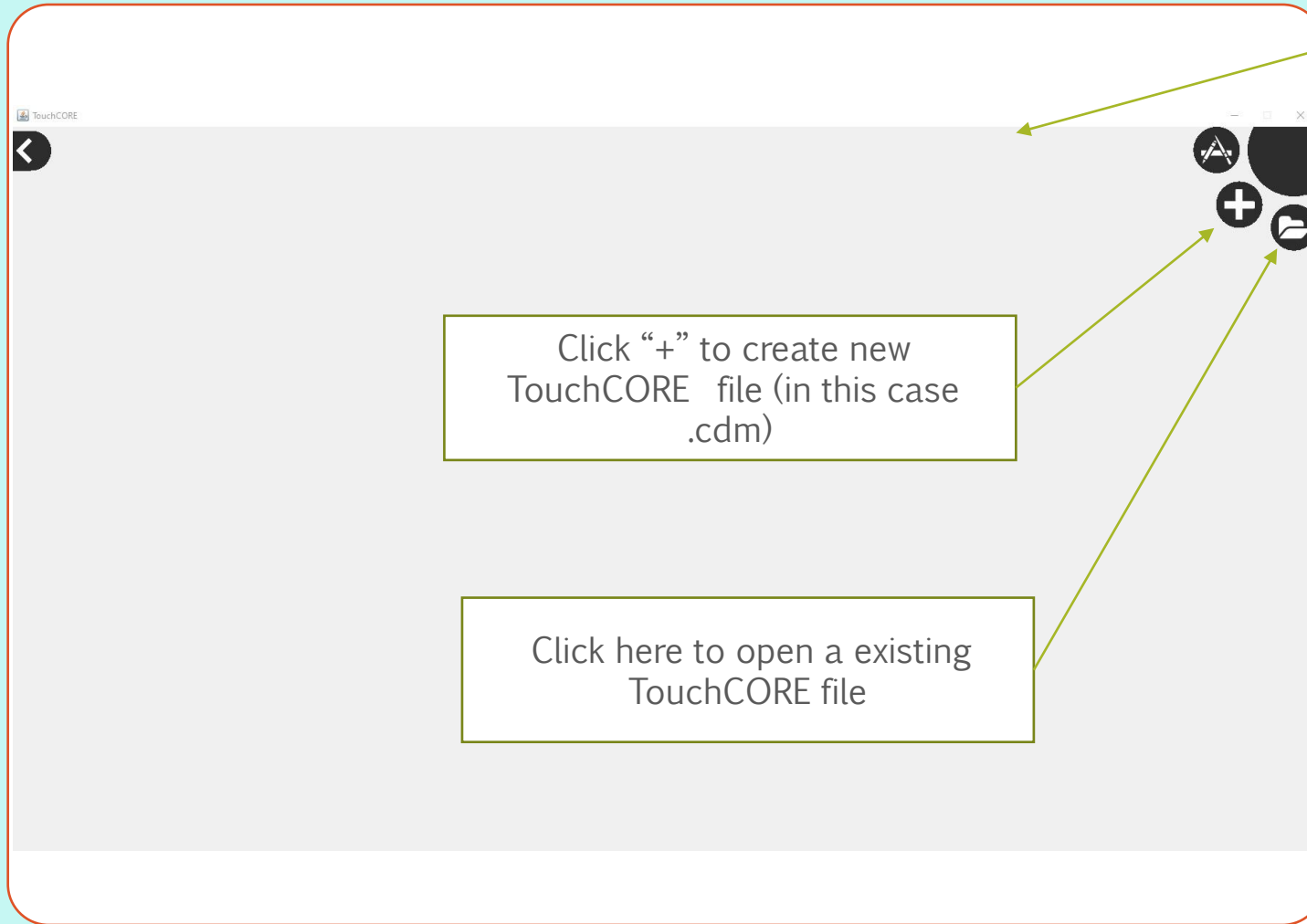
Click "+" to create new TouchCORE file (in this case .cdm)

Click here to open a existing TouchCORE file

Browse to the location where you want to store the file.

Test\

The folder is empty.
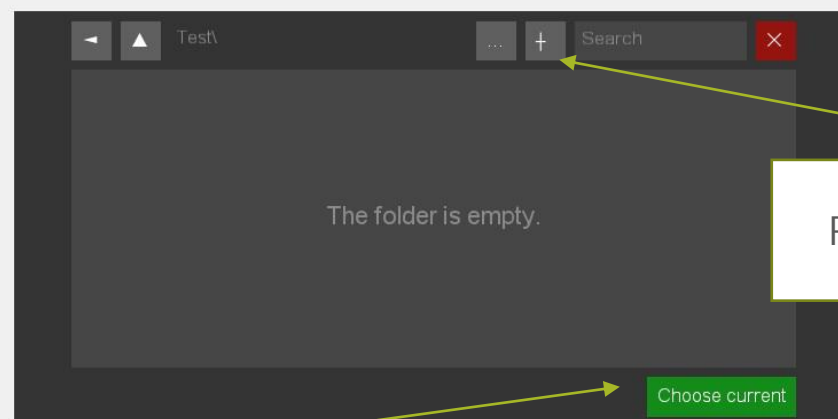
Press "+" to create new folder

Choose current

Click "Choose Current" to continue at that location

Enter the name of cdm file

instructor_MistakePluralClassNam ✓ ✗

Choose current
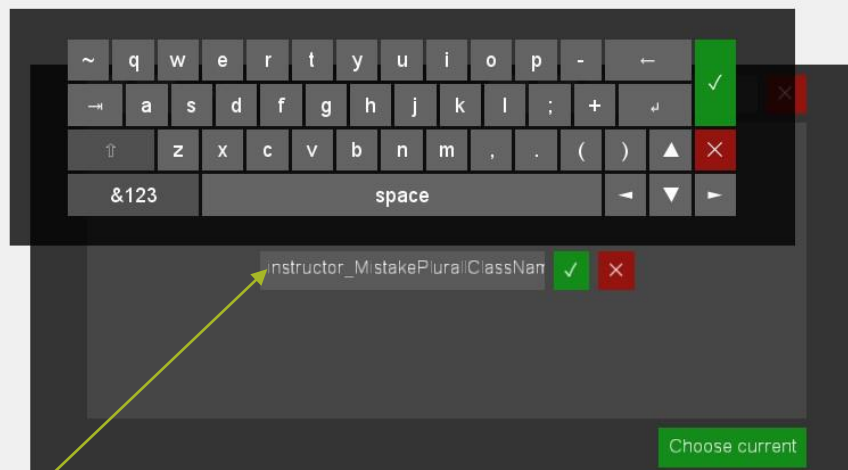
!! Name the class according to its projected use. In this example "instructor_MistakePluralClassName" is used as a name, this tells us that, this is an instructor solution and will be used to test Plural class name mistake

Instructor_MistakePlurallClassName

Instructor_MistakePlurallClassName

New Realization Model
Import Realization Model
Associate Conflict Resolution Model
Add Optional
Add Mandatory
Add Xor
Add Or

2. Hold left mouse button and select "New Realization Model"

1 .Select "Domain Modellng"

Concern Reuses

Perspective
Design Modelling
Design Modelling with RAM
Domain Modelling
Rest Interface Modelling
Use Case Modelling

Goals

Legend

Constraints

Step 1: Create a cdm using TouchCORE.

1. Hold left mouse button and select "Create Class"

Step 1: Create a cdm using TouchCORE.

TouchCORE

Instructor_MistakePlurallClassName <Instructor_MistakePlurallClassName> : Instructor_MistakePlurallClassName

Test

~	q	w	e	r	t	y	u	i	o	p	-	←	✓
→	a	s	d	f	g	h	j	k	l	;	+	↵	
⇧	z	x	c	v	b	n	m	,	.	(	)	▲	✕
&123		space							◄	▼	►	

1. Name the Class

2. Save

Extended Design Models
Concern Reuses

Step 1: Create a cdm using TouchCORE.

TouchCORE

Test

1. To Add attribute long press (hold left mouse button) the class

2. Press "A*"

Extended Design Models

Concern Reuses
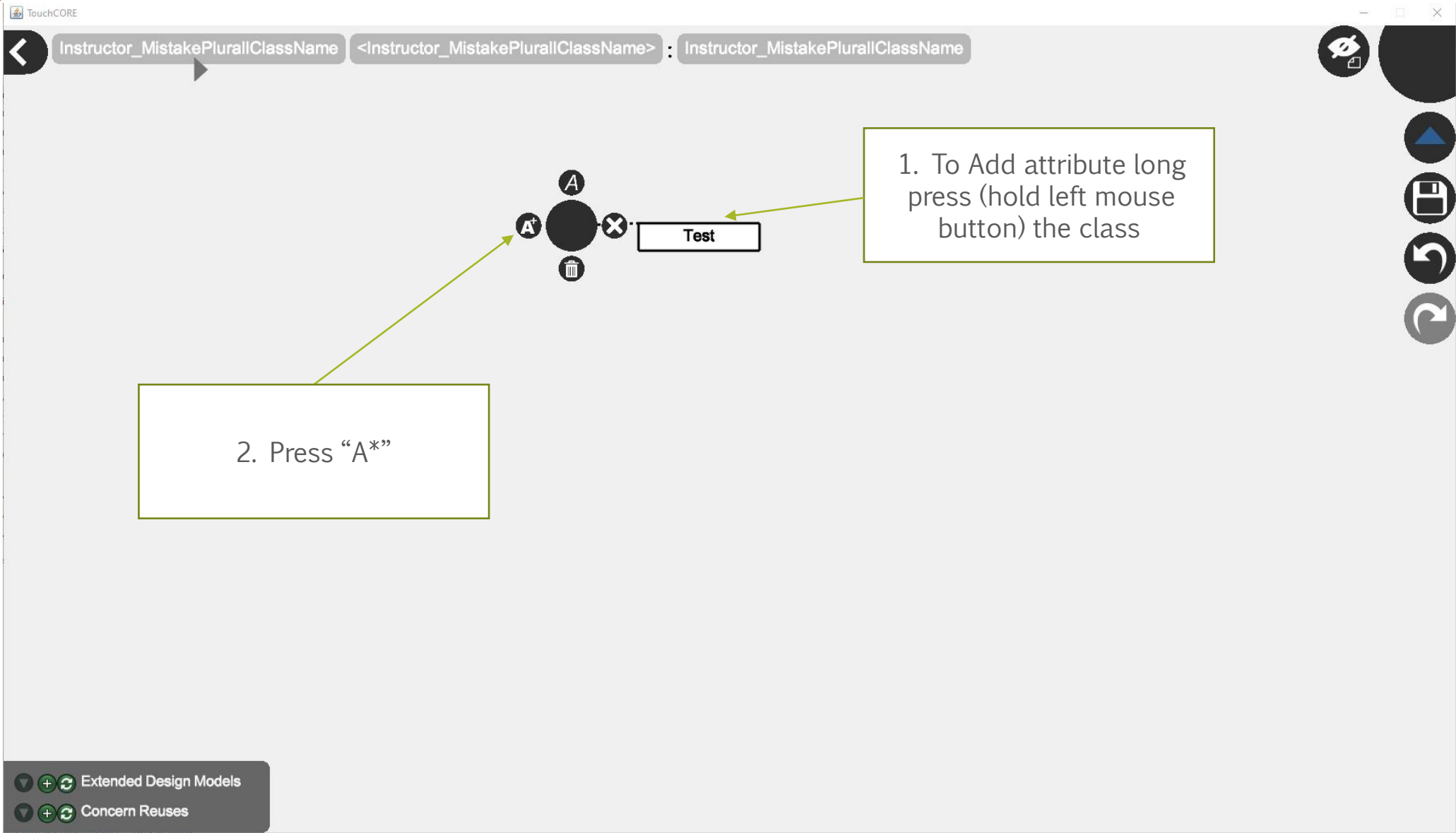
Step 1: Create a cdm using TouchCORE.

Instructor_MistakePlurallClassName <Instructor_MistakePlurallClassName> : Instructor_MistakePlurallClassName

Test

int testAttribute
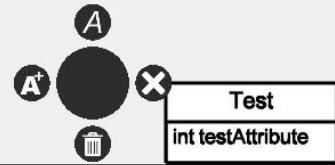
1. Enter the data type and name of the attribute
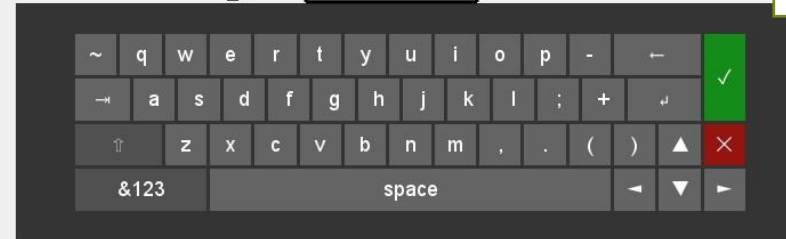
2. Save

Extended Design Models

Concern Reuses

Step 1: Create a cdm using TouchCORE.

TouchCORE

Test

int testAttribute

Important !!
1. Take a screenshot of the created class diagram. This will help to look at the class diagram in future
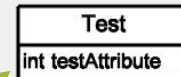
Bravo !! your class diagram is now ready .

2. Save

Note: You can create more classes, in similar way

Extended Design Models

Concern Reuses

Step 1: Create a cdm using TouchCORE.

Now browse to a new location and create a cdm for student solution

student_MistakePluralClassName

Choose current
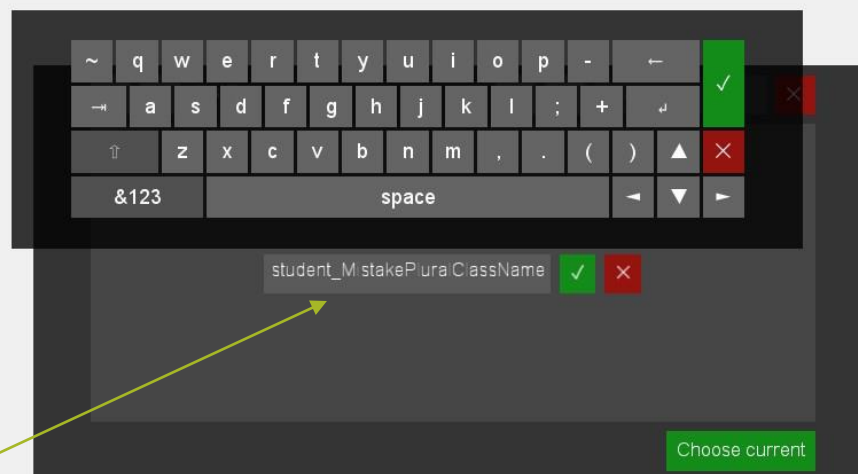
!! Name the class according to its projected use. In this example "student_MistakePluralClassName" is used as a name, this tells us that, this is an student solution and will be used to test Plural class name mistake

Step 1: Create a cdm using TouchCORE.

Step 1: Create a cdm using TouchCORE.

This is how the folders will look at the saved location

The files we require for testing (.cdm)

Step 1: Create a cdm using TouchCORE.

mistakedetection [modeling-assistant PrabhsimranSingh-mistake-detectic
- Project Dependencies
- src
- JRE System Library [JavaSE-11]
- Plug-in Dependencies
- Referenced Libraries
- META-INF
- model
- taggers
- testModels
  - InstructorSolution
    - ModelsToTestAttribute
    - ModelsToTestClass
    - ModelsToTestPattern
    - ModelsToTestRelationship
    - One
    - two(withAttributes)
    - InstructorSolution.core
  - StudentSolution
    - four(WrongAttibuteType)
    - ModelsToTestAttribute
    - ModelsToTestClass
    - ModelsToTestPattern
    - ModelsToTestRelationship
    - One
    - three(withAttributes)
    - two(withAttribute)
- .classpath
- .project
- build.properties
- stanford-postagger.jar

Copy the .cdm to the folder according to the mistake to be tested

In this example the instructor solution file will go to: testModels->InstructorSolution->ModelsToTestClass

And the student solution file will go to: testModels->StudentSolution->ModelsToTestClass

Step 2: Copying File to project

Go to the Mistake appropriate java File to write a test

In this example, you will write test in "MistakeDetectionWrongClassTest"

Step 3: Writing Test

1. Load cdm file

Name of instructor solution class diagram file

```java
@Test public void testMistakePluralClassName() {
    ClassdiagramPackage.eINSTANCE.eClass();
    var cdmFile = "../mistakedetection/testModels/InstructorSolution/flolderName/Instructor_MistakePluralClassName.domain_model.cdm";
    var resource = ResourceHelper.INSTANCE.loadResource(cdmFile);
    var classDiagram = (ClassDiagram) resource.getContents().get(0);
    var maf = ModelingassistantFactory.eINSTANCE;
    var modelingAssistant = maf.createModelingAssistant();
    var solution = maf.createSolution();
    solution.setModelingAssistant(modelingAssistant);
    solution.setClassDiagram(classDiagram);

    ClassdiagramPackage.eINSTANCE.eClass();
    var cdmFile1 = "../mistakedetection/testModels/InstructorSolution/flolderName/Student_MistakePluralClassName.domain_model.cdm";
    var resource1 = ResourceHelper.INSTANCE.loadResource(cdmFile1);
    var classDiagram1 = (ClassDiagram) resource1.getContents().get(0);
    var maf1 = ModelingassistantFactory.eINSTANCE;
    var modelingAssistant1 = maf1.createModelingAssistant();
    var solution1 = maf1.createSolution();
    solution1.setModelingAssistant(modelingAssistant1);
    solution1.setClassDiagram(classDiagram1);
    var student = maf1.createStudent();
    solution1.setStudent(student);
```

Relative path of cdm file

Similarly, load student solution class diagram file

Step 3: Writing a test

```
Classifier instructorTestClass = null;
Attribute instructorTestClassAttributeTestAttribute = null;

for (var c : classDiagram.getClasses()) {
  if ("Test".equals(c.getName())) {
    instructorTestClass = c;
    for (Attribute a : c.getAttributes()) {
      if ("testAttribute".equals(a.getName())) {
        instructorTestClassAttributeTestAttribute = a;
      }
    }
  }
}

Classifier studentTestClass = null;
Attribute studentTestClassAttributeTestAttribute = null;

for (var c : classDiagram.getClasses()) {
  if ("Tests".equals(c.getName())) {
    studentTestClass = c;
    for (Attribute a : c.getAttributes()) {
      if ("testAttribute".equals(a.getName())) {
        studentTestClassAttributeTestAttribute = a;
      }
    }
  }
}
```

3. Initialize the defined variables

4. Similarly, repeat the same for student solution

Step 3: Writing a test

```java
comparison = MistakeDetection.compare(solution, solution1);

assertEquals(comparison.notMappedInstructorClassifier.size(), 0);
assertEquals(comparison.extraStudentClassifier.size(), 0);
assertEquals(comparison.mappedClassifier.size(), 1);

assertEquals(comparison.mappedClassifier.get(instructorTestClass),studentTestClass);

assertEquals(comparison.notMappedInstructorAttribute.size(), 0);
assertEquals(comparison.extraStudentAttribute.size(), 0);
assertEquals(comparison.duplicateStudentAttribute.size(), 0);
assertEquals(comparison.mappedAttribute.size(), 1);

assertEquals(comparison.mappedAttribute.get(instructorTestClassAttributeteTestAttribute),
    studentTestClassAttributeteTestAttribute);

assertEquals(comparison.newMistakes.size(), 1);
assertEquals(solution1.getMistakes().size(), 1);

for (Mistake m : solution1.getMistakes()) {
  if (m.getMistakeType() == MistakeTypes.USING_PLURAL_OR_LOWERCASE
      && m.getStudentElements().get(0).getElement() == studentTestClass) {
    assertEquals(m.getStudentElements().get(0).getElement(), studentTestClass);
    assertEquals(m.getNumDetectionSinceResolved(), 0);
    assertEquals(m.getNumDetection(), 1);
    assertTrue(m.isResolved());
  }
}
```

5. Write assertions to check the mapping (Test = Tests).

This check if correct elements are mapped

Similar in case of attributes

6. Assert the size of detected mistake and Mistakes introduced in class diagram

7. Check the correctness of mistake type and its solution element

Step 3: Writing a test

Thank you and have fun using Mistake Detection System