TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Software- und Multimediatechnik, Lehrstuhl für Softwaretechnologie

Extending Artemis With a Rule-Based Approach for Automatically Assessing Modeling Tasks

Franz Rodestock

franz.rodestock@tu-dresden.de
Born on: 9th May 2000 in Zwenkau
Course: Informatik
Matriculation number: 4883445
Matriculation year: 2019

Bachelor Thesis

to achieve the academic degree

Bachelor of Science (B.Sc.)

Supervisor
Dr. Sebastian Götz

Supervising professor
Prof. Dr. rer. nat habil. Uwe Aßmann

Submitted on: 20th May 2022

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

E-learning systems have become an integral part of university education in recent years. The *Technische Universität Dresden* has multiple projects in use. The *Chair of Software Technology* uses *Inloop* to teach students object-oriented programming through automatic feedback [12]. In the last years, interest has grown in giving students automated feedback on modeling tasks. This is why there was an extension developed by Hamann to automate the assessment of modeling tasks in 2020 [8].

The *TU Dresden* currently has plans to replace *Inloop* with *Artemis*, a comparable system developed by the *TU Munich*. This decision was made in favor of a more extensive range of functions, and more rapid feature development due to the higher number of developers [11]. Artemis currently supports the semi-automatic assessment of modeling exercises, which feedback is therefore not instantaneous for students. In contrast, the system proposed by Hamann, called Inloom, is based on a rule-based approach and provides instant feedback. A rule-based system has certain advantages over a similarity-based system. One advantage is the mostly better feedback that these systems generate. The feedback is essential for the learning process of students.

To give instructors more flexibility and choice, this work tries to identify possible ways of extending *Artemis* with Inloom. In the second step, this thesis will provide a proof of concept implementation. Furthermore, a comparison between different systems is developed to help instructors choose the best suitable system for their use case.

1.2 Research Questions

This thesis establishes multiple research questions:

RQ1 What are the current technologies used to automate the feedback for modeling exercises?

RQ2 What are ways to integrate Inloom into Artemis?

RQ3 What is the best way for the integration of Inloom?

RQ4 How does the newly integrated system performs when used by students?

The first question (RQ1) is needed to get the required background knowledge to properly evaluate and compare the already existing semi-automated assessment with the implemented rule-based assessment. This question is answered by a literature survey of already existing methods for automatic model assessment. To answer the second question (RQ2), this work presents multiple approaches for implementing the rule-based assessment system. The found approaches are going to be compared and ranked against each other. The third question (RQ3) is answered by proposing a prototypical implementation. The last question (RQ4) deals with the newly implemented system and how it may affect the student's process of learning and improving their modeling skills.

2 Background

This section gives a brief overview of technology and concepts needed to understand this thesis. Among other things, the design of Artemis and Inloom is explained. Inloom is the system that will be integrated into Artemis in this thesis.

2.1 Artemis and Semi-Automatic Assessment of Modeling Exercises

Artemis is an automatic assessment management system for interactive learning developed by the TU Munich [11]. Artemis is an acronym for **A**u**T**omated ass**E**ssment **M**anagement **S**ystem for interactive learning. The goal of Artemis is to provide students in larger classes with instant feedback. Artemis is deployed by eleven different universities[1], primarily used in software engineering courses. Artemis supports different types of exercises. It currently supports programming- and modeling exercises, questionnaires, text-based- and upload exercises[2], but the main focus is clearly set on the programming exercises. It does not depend on the programming language and currently supports nine programming languages. Figure 2.1 shows Artemis' top-level design. Students' solutions are pushed to a version control system (VCS) which are then evaluated by a continuous integration (CI) pipeline. This process ensures a maximum scalability. Artemis offers an online editor. The editor entirely hides the implementation of the automatic assessment, which offers a low-key possibility for students to participate in these kinds of tasks. More advanced students may push their solutions directly to the version control system without the need to interact with Artemis.

Artemis also supports modeling exercises. These exercises are currently assessed in a semi-automatic way using supervised machine learning [10]. Modeling tasks do not use the CI pipeline for assessment. Compared to manual assessments, the goal is to speed up the assessment, increase the fairness and the quality of provided feedback. Students create their solutions in a graphical editor called *Apollon*[4]. The core concept is to assess the first few solution models students have proposed manually. While assessing the models, the system learns correct elements and appropriate feedback.

[1] `https://github.com/ls1intum/Artemis`, 13.05.2022
[2] `https://docs.artemis.ase.in.tum.de`, 13.05.2022
[3] `https://docs.artemis.ase.in.tum.de/_images/TopLevelDesign.png`, 13.05.2022
[4] `https://github.com/ls1intum/Apollon`, 13.05.2022

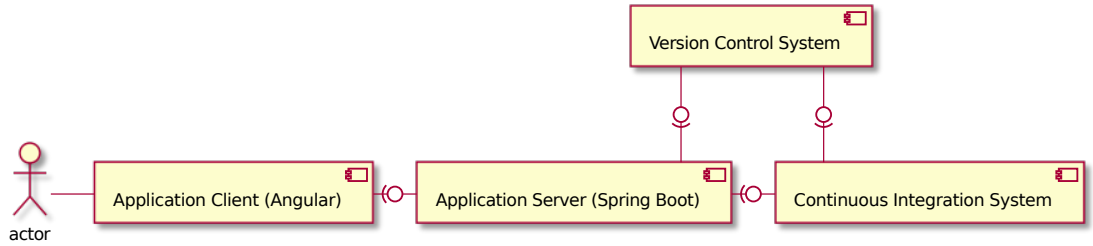

Figure 2.1 An overview of Artemis' top-level design. The figure is based on the top-level design figure from the Artemis documentation[3].

When grading more solutions, the system gives the reviewer suggestions on assessing certain elements of the model.

2.2 Inloop and Rule-Based Assessment Using Inloom

Inloop is another representative system for automatic assessments of programming exercises [12]. It was developed at TU Dresden. Inloop stands for **IN**teractive **L**earning center for **O**bject-**O**riented **P**rogramming. Currently, Inloop supports Java programming exercises. While the scope of functions is not comparable to Artemis, the structure is, in principle, similar. Tasks are also defined using a version control system, and the students' submissions are checked by running them in Docker containers as Artemis does.

Inloom [8] was developed as an extension for Inloop to support the automatic grading of modeling exercises. As of 2022, it is not actively used at TU Dresden. Inloom was used for testing purposes and was ready for use. In fall 2021, however, the decision was made to switch to Artemis.

In contrast to the currently used semi-automated assessment by Artemis, Inloom uses an alternative way of assessing modeling tasks. Inloom utilizes a rule-based approach. Models are represented within the Eclipse Modeling Framework[5] (EMF). Inloom creates the constraint-based test set (evaluation rules) out of an example solution given by instructors. The instructors can then adjust this generated test set. Task artifacts like the task description, the test engine or the test set are stored in a Git repository.

Figure 2.2 shows Inloop's top-level design. After a student submits a solution, the solution will be built by background workers. The workers pull the necessary artifacts from the task repository. Afterward, they build the submission and execute the test using Docker containers. In Inloom, the test engine is called instead. The test engine will match the students' solutions against the previously created test set. Inloom also grades the model and gives the students generated feedback.

[5] `https://www.eclipse.org/modeling/emf/`, 13.05.2022

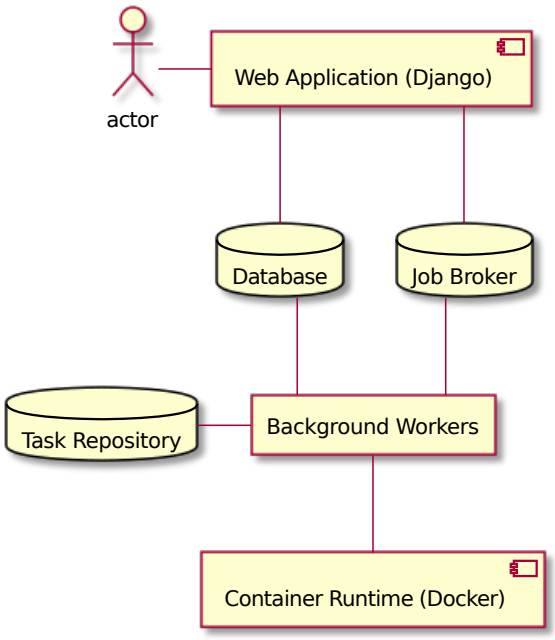

Figure 2.2 An overview of Inloop's top-level design. The figure is based on the top-level overview from the work of Morgenstern and Demuth [12]

3 Related Work

This chapter will give an overview of concepts and systems currently used for the automatic assessment of modeling exercises. Furthermore, various technologies will be compared, which might serve as a guideline to help instructors choose the best assessment method for their objective or university course. The goal of this chapter is to answer Research Question RQ1.

The literature research was done using keywords like "automatic", "assessment", "grading", "evaluation", "machine learning", "rule-based", "UML", "class diagram" or any combination of them querying popular archives like *Google Scholar*/*IEEE* or *Researchgate*. Afterward, a reference search was conducted on the found literature to increase the search radius by following promising references further. This chapter's primary focus will be on systems proposed after 2018, the year of Hamann's work to mitigate duplications [8].

When discussing modeling tasks, this chapter refers to modeling exercises of class diagrams. This limitation to class diagrams is due to the literature's primary focus on these diagrams.

Table 3.1 lists related work categorized for the approach used and the kind of grading system used. The grading is an essential characteristic for instructors to know because the grading defines the system's potential use.

The following sections will introduce and explain the two main approaches to automatic model assessment. Furthermore, the different approaches are further compared in the Conclusion section.

3.1 Systems for Automatic Modeling Feedback

There are two main approaches for automatic assessment of modeling exercises. One subsection will deal with similarity-based systems, another with rule-based systems.

3.1.1 Assessment Using a Similarity-Based Approach

Matching a student's solution to one or multiple sample solutions is the most common approach in most recent literature. These systems create and use an abstract similarity measure for determining the similarity of the models. The abstract value can be created heuristically or by machine learning algorithms.

Table 3.1 An overview of model assessment systems found in the literature. The table is sorted by date.

Author	Year	Approach	Feedback and Grading
Krusche [10]	2022	manual assessment supported with machine learning	generated feedback and point-based grading system
Anas [1]	2021	similarity on graphs	similarity measure
Boubekeur [5]	2020	machine learning	prediction of grades (A-E)
Hamann [8]	2020	rule-based	feedback and point-based grading system
Reischmann [13, 14]	2019	similarity-based supported with machine learning	generated feedback
Ichinohe [9]	2019	similarity-based	similarity measure
Cheers [7]	2019	machine learning	generated feedback
Bian [4]	2019	rule-based	generated feedback and point-based grading system
Stikkolorum [15]	2019	machine learning	classification into categories (e.g. good, pass, fail)
Striewe [16]	2011	rule-based	generated feedback

Anas et al. first creates an UML graph and then measures syntactic, structural and semantic similarity to an expert solution [1]. Based on this, a similarity value to the sample solution is generated, which serves as feedback for students.

An example of a similarity approach using a machine learning algorithm was proposed by Boubekeur et al. [5]. They use different machine learning approaches to predict a grade (A-E) for the submitted solution.

Another system, based on similarity but working differently, is the currently used system in Artemis proposed by Krusche, which uses machine learning to assist tutors while evaluating the students' solutions, and is therefore not automatic [10]. The functionality of this system is further described in Chapter 2.

3.1.2 Assessment Using a Rule-Based Approach

Another way of automatic assessment is to grade the student's solution by matching it to an expert solution by algorithmic element matching, introducing rules or constraints, or using graph techniques.

This approach is less common in literature than the similarity approach. This was different in the past, so older works were also reviewed.

One system was proposed by Bian et al. [4]. A student's solution is matched to the example solution by using syntactic, semantic, and structural information. Striewe and Goedicke proposed another system [16]. They interpret UML diagrams as graphs. The instructor can define rules which are then validated using graph queries. Inloom, which this thesis is going to integrate into Artemis, is also representative of the rule-based approach [8]. The functionality of Inloom has already been described in Chapter 2.

3.2 Grading and Feedback

Systems provide different kinds of feedback. Feedback can be in the form of textual feedback, for example, on correct or missing elements. Explanatory feedback is *"Class Profile was found"* or *"Operation accept of Friendship was found"*. The two sentences are taken from Inlooms output. The systems proposed by Reischmann, Cheers are further representatives of systems with this kind of feedback [7, 14].

Feedback can also quantitatively predict a grade or assign the student's submission a number of points. Representatives of systems that give quantitative feedback were proposed by Boubekeur or by Hamann [5, 8].

The feedback the systems provide is crucial for defining the areas of application a system has. For teaching purposes, the quality of textual feedback is essential, while for exams, the grading system has to be of high quality. For exams, the origin of the feedback has to be traceable. Similarity-based approaches, especially those supported by machine learning, do not provide that.

3.3 Comparison Between Similarity-Based and Rule-Based Approaches

This section compares the two main approaches used. This is to support the decision made by instructors on which approach to take for a specific use case. Table 3.3 compares the similarity-based with the rule-based approach. Non-self-explanatory attributes are explained in this section.

Table 3.3 A comparison of similarity-based approaches with a rule-based approach.

	Similarity/Machine Learning-Based Approach	Rule-Based Approach
Setup	Easy as providing an example solution is sufficient	Might be more complicated due to higher complexity for understanding rules
Grading	Similarity value or prediction of grade	Point based system
Feedback	Mostly no textual feedback	Textual feedback on model elements
Assessing Alternative Solutions	Might be hard to incoporate	
Assessment Time	Basically instantaneous	

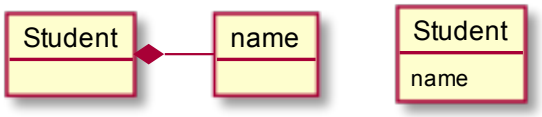

Figure 3.1 An example of two models describing the same domain but modeled differently.

Preparing and setting up a task with similarity systems is mostly pretty straightforward since only one or more example solutions are needed, against which the students' solutions are compared. The preparation of a rule-based system can be more tedious since a set of rules must be prepared. This effort can be reduced by auto-generating a set of rules from example solutions, which then can be edited and refined by instructors. Inloom leverages exactly that process [8].

One general drawback of fully automated systems is that it is hard to assess alternative solutions correctly. Alternative solutions are also correct, but the instructor has not thought of them. Modeling class attributes as a second class with a relation is a typical example. Figure 3.1 shows that example. Rule-based systems can manage that by predefining rules matching specific alternative solutions. Furthermore, a set of alternative solutions can be stored with similarity and rule-based systems, against which the student solutions are matched.

Finding alternative solutions can be done by searching the students' solutions, which have performed poorly, to find valid alternative solutions. The found solutions can then be integrated as another sample solution, or the rules can be updated. This tedious task could be diminished by automatically generating alternative solutions by refactoring and transforming the model. Such a system does not currently exist and is left for future work [3, p. 373].

One of the main differences between both approaches is the quality of created feedback and the grading they provide. Similarity-based systems sometimes only provide a predicted grade or a similarity score. However, feedback on the correct/erroneous model elements is necessary for learning progress. Rule-based assessment systems offer precisely that. When the system is going to be used for bonus points or even grading of exams, the extra effort of creating and maintaining the rules for a rule-based system may be worth it. This is because the rule-based feedback is mostly more helpful to students than the output of a similarity-based system.

The decision for a system is not easily made. Both approaches have their advantages and disadvantages. The recommendation is to test the desired system thoroughly before using it in software engineering courses.

4 Analysis

This thesis aims to present the best possible integration of Inloom into Artemis. In this chapter, Artemis is analyzed to find starting points for the integration. This will lay the foundation to answer Research Question RQ2.

Firstly, the following section introduces criteria with which the identified approaches for the integration are compared. Secondly, three potential approaches get further analyzed. In the conclusion section, an approach is identified, which this thesis will pursue further.

4.1 Evaluation Criteria

All presented approaches are compared using following criteria.

EC1 User Experience:
A good user experience is crucial for the future success of the proposed system. Above all, this means how easy it is to use the system.

EC2 Maintainability:
How maintainable a system is, decides on the future maintenance effort and thus directly on the costs of the system.

EC3 Invasiveness:
The criterion invasiveness describes how profoundly the program has been adjusted. Less invasiveness of the integration leads most likely to less maintenance effort and is therefore desirable.

EC4 Implementation Effort:
This thesis has to balance the approach's implementation effort with the anticipated outcomes. Less implementation effort is desirable since it lowers the development cost in a real-world setting.

4.2 Replacing the Built in Assessment System for Modeling Tasks

In Artemis' modeling tasks student use the modeling editor *Apollon*[1] to create and submit models. These models are then evaluated by the automatic assessment component *Compass*.
This approach intends to replace this system with Inloom. At first sight, this approach looks feasible, but unfortunately, it would come with drawbacks.

One major drawback is that the Compass system does not accept any input. However, Inloom uses a test set to match the student's solution to the example solution. This test set differs between tasks and therefore needs to be given as input for the test engine. Replacing the assessment system with Inloom would ultimately lead to many changes in Artemis' code and, therefore, to a not easily maintainable solution.

4.3 Adding a Continuous Integration Pipeline for Modeling Tasks

The second approach is to add a continuous integration (CI) pipeline for the modeling tasks. This approach was identified because Inloom currently also uses a build pipeline, which is similar to a CI pipeline. This approach has the same advantage as the reusability of the modeling editor, but currently, modeling tasks are not connected to the CI pipeline. Currently, only programming exercises are connected to the CI pipeline. Adding a pipeline to modeling tasks can only be achieved with many backend changes and potentially duplicated code.

4.4 Reusing Programming Exercises

The third approach is to treat the modeling exercise like a programming exercise. Artemis' programming exercises are already integrated into a CI pipeline, which hopefully can be used without customization.

One drawback is that the modeling exercises currently do not support the graphical modeling editor *Apollon*. This drawback can be combated by enabling the upload of externally created models, for example, created with the *Eclipse Modeling Tools*. Inloom currently supports the models students can create with an editor plugin written for Eclipse Modeling Tools.

If it becomes apparent that this approach has not enough flexibility needed for the integration of Inloom, it would also be possible to create a new type of exercise, extending the programming exercises.

4.5 Conclusion

The decision on which approach to follow further was not easily made. Since no approach fits perfectly, there are compromises made. Figure 4.1 shows the main differences between all exercise types Artemis offers. This includes the exercise types, which were not featured in more detail in this chapter. Table 4.1 compares the presented approaches more precisely with the introduced design criteria.

[1] `https://github.com/ls1intum/Apollon`, 13.05.2022

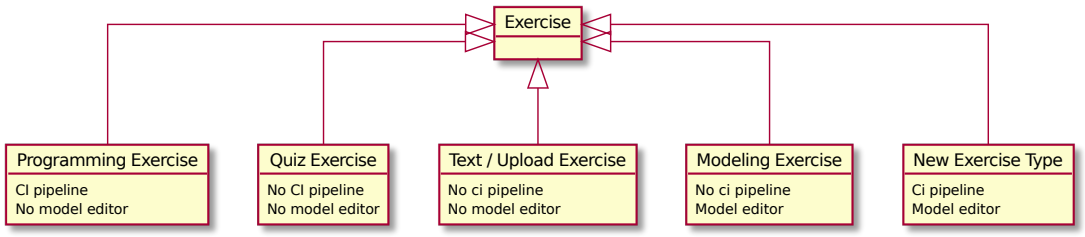

Figure 4.1 An overview of supported exercises by Artemis with advantages and limitations.

Table 4.1 Weighing the different presented approaches by the different evaluation criteria. The scale ranges from strongly negative (- -) to neutral (o) to strongly positive (++).

	EC1	EC2	EC3	EC4
Replacing Compass System	++	-	- -	- -
Adding a CI Pipeline to Modeling Tasks	++	- -	-	- -
Reuse Programming Exercises	o	+	++	+

This thesis is going to pursue the third approach further. The reason for this decision is the excellent modular ability to add a new type of programming language. Hopefully, the implementation should be straightforward and outweighs the predicted nonperfect user experience of a missing online model editor.

The following chapter will focus on the prototypical system design of the chosen approach.

5 System Design

This chapter will discuss and propose a way of integrating Inloom into Artemis. By doing this, Research Question RQ3 is going to be answered. While this thesis was being written, the pipeline behind Inloom was renamed to Mars. Mars is an acronym for **M**odel-driven **A**ssisted **R**ule-based assessment **S**ystem.

The previous chapter concluded that the approach of adding Mars as a new type of programming language is the best. This approach is modular since the already existing codebase is not extensively modified.

5.1 Proposed Integration

Figure 5.1 shows the proposed pipeline for the integration of Mars. The parts of the steps underlined will be modified by this thesis. The following sections describe what is needed for the integration and what considerations have been made.

5.2 Adding Mars as a New Type of Programming Language

Artemis programming exercises have the attribute *type*. The programming exercises currently support Java, C, Haskell, and many more. A new type of programming language must be added. Adding Mars as a programming language includes adding it to the back end and front end.

Allowing users to choose Mars in the new exercise form is ultimately needed to create a new programming exercise of the type Mars.

Figure 5.2 shows the data model of Artemis. It omits details in favor of better readability. The slight adjustment of adding Mars as a new programming language is highlighted in the diagram.

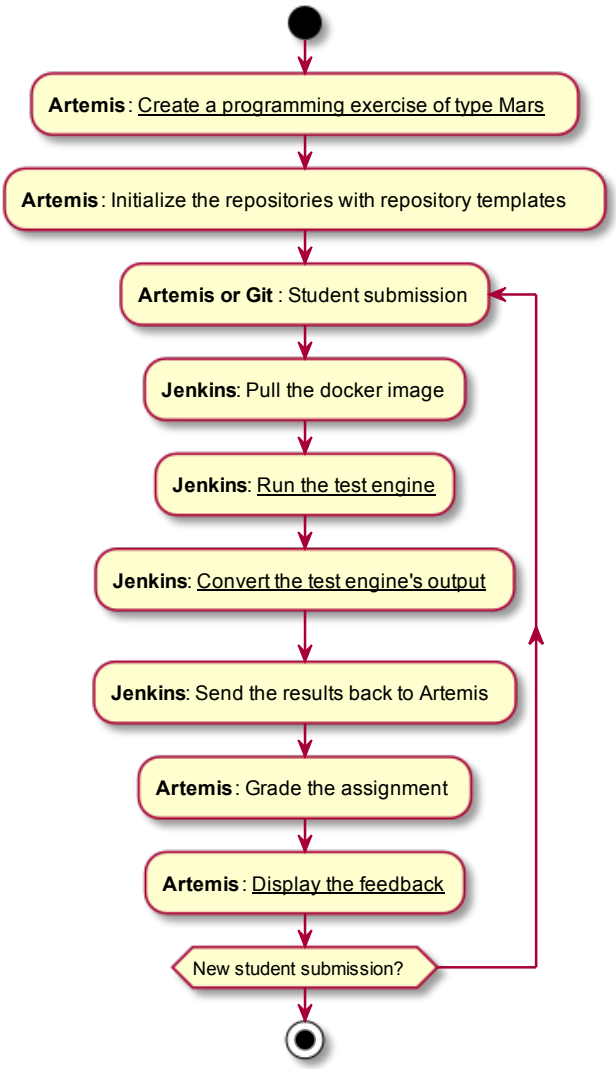

Figure 5.1 An overview of the proposed pipeline for the integration of Mars. The program responsible for the step is highlighted. Underlined steps of the pipeline are modified by this thesis.

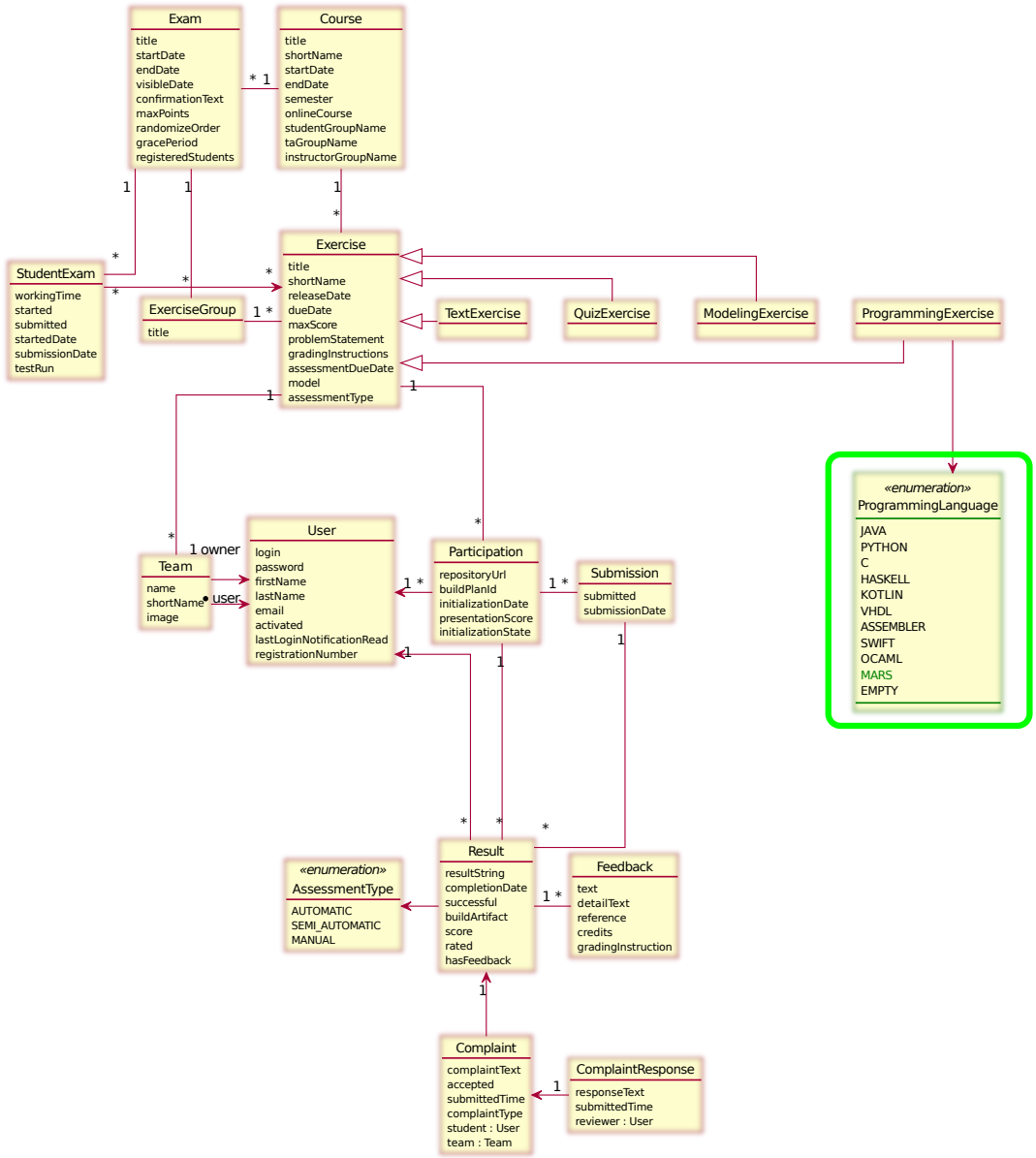

Figure 5.2 An overview of Artemis' exercise relevant data model with the point of integration highlighted. The data model is based on the data model in the Artemis documentation[1]

5.3 Specifying Repository Templates for an Effortless Task Setup

Every programming exercise has an explanatory task to make the setup process easier for instructors. This task is predefined in the repository templates. Artemis separates tasks into three Git repositories. The repositories' naming and functionality are similar to the repositories of other programming languages.

The repositories are the following:

Exercise
: Whenever students start an exercise, their repository is initialized with the exercise repository. All files included in the exercise repository are copied into the newly created student repository.

Solution
: This repository contains an example solution.

Test
: The test repository contains all files necessary for grading the students' assignments. For Java programming exercises, these are, for example, the JUnit tests.

In the process of integrating Mars, all three templates must be defined.

5.4 Submitting Models by Leveraging the Online Editor

Artemis integrated existing online file editor has the necessary functionality to submit students' assignments. The editor includes the creation and editing of files. Students will create their models using the offline modeling editor provided as a plugin for the *Eclipse Modeling Tools*[2]. Afterward, they will upload their model using the online file editor or alternatively by using Git. The creation and uploading the file is similar to the process proposed by Hamann for Inloom [8].

5.5 Jenkins Continuous Integration Pipeline

Reusing programming exercises as the point of extension allows for the reuse of the continuous integration (CI) pipeline. Artemis runs the checks on student repositories with the help of a continuous integration system. Artemis currently supports two CI systems, namely *Bamboo*[3] and *Jenkins*[4]. The TU Dresden is using Jenkins. That is why the focus of this thesis is on Jenkins. The prototypical implementation will only support Jenkins because Artemis' Atlassian stack could not be tested.

A new Jenkins pipeline must be created to process the Mars assignments. Jenkins will be responsible for checking out the repository, running the test engine, converting the test engine's output to a format Artemis can handle, and sending back the results to Artemis. A more detailed explanation will be given in Chapter 6.

[1] `https://docs.artemis.ase.in.tum.de/dev/system-design/#id4`, 13.05.2022
[2] `https://www.eclipse.org/downloads/packages/release/2022-03/r/eclipse-modeling-tools`, 13.05.2022
[3] `https://www.atlassian.com/de/software/bamboo`, 13.05.2022
[4] `https://www.jenkins.io/`, 13.05.2022

5.6 Using Docker Containers for Increased Security

Whenever third-party code is executed, security aspects must be considered. The student submissions are such third-party code. That is why Artemis checks the student submissions within Docker containers. This encapsulation ensures that student assignments do not have direct access to the host system hence increasing security.

The Docker images contain all the necessary software for running the test engine and converting the output. A new Docker image must be created for the exercises of type Mars.

5.7 Converting the Test Engine's Output

Artemis uses test cases to determine the correctness of a submission compared to the given solution. The *Jenkins Server Notification Plugin* therefore expects some test cases. Currently, the test engine's output is not in the form of test cases. Therefore the test engine's output needs to be converted to match the test case layout of Artemis.

5.7.1 Current Test Engine Output

The test engine's output is currently an XML file. It contains some metadata about the version and different metamodels used. However, more importantly, the output contains a list of all results and information about the achieved points. The following excerpt shows two example result objects. The full output can be found in the appendix 9.4.

Listing 5.1 Excerpt of the test engine's output.

```
1  <Result>
2   <ExpertObject>Student</ExpertObject>
3   <ExpertType>Class</ExpertType>
4   <TestObject>Student</TestObject>
5   <TestType>Class</TestType>
6   <Rule>Rule_R010101</Rule>
7   <Category>CORRECT</Category>
8   <Points>1.0</Points>
9   <Msg>Class Student was found.</Msg>
10 </Result>
11 <Result>
12  <ExpertObject>Student_name</ExpertObject>
13  <ExpertType>Property</ExpertType>
14  <TestObject>Student_name</TestObject>
15  <TestType>Property</TestType>
16  <Rule>Rule_R020101</Rule>
17  <Category>CORRECT</Category>
18  <Points>0.5</Points>
19  <Msg>Property name of Student were found.</Msg>
20 </Result>
```

The code example shows that the test engine's output mainly consists of positive messages ("Class Student was found"). There are no negative messages like "Class student was **not** found" because messages of this kind would give away information about the solution and might enable students to reverse engineer the model.

5.7.2 Accepted Output by Artemis

Results are sent back to Artemis by the *Jenkins Server Notification Plugin*[5]. There are two options of formats the plugin accepts. The plugin accepts either JUnit XML reports or a custom format having one file for each test result.

The following code shows how an example JUnit output looks like.

Listing 5.2 Example JUnit results file.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <testsuite name="Mars Output">
3     <testcase name="Successful Testcase 1"/>
4     <testcase name="Successful Testcase 2"/>
5     <testcase name="Failed Testcase 3">
6         <failure>Test Failure Message</failure>
7     </testcase>
8 </testsuite>
```

One advantage of using JUnit is that the results file's creation is pretty simple and can be easily done by modifying the test engine's output file. However, using JUnit has a significant disadvantage since it only allows for messages passed in failed tests. That is why the second option was developed, which allows for messages, also in successful tests.

The following code shows an example output that follows the standard set by the Artemis Notification Plugin.

Listing 5.3 Example .json results file.

```
1 {
2     "name": "Rule: 1",
3     "successful": true,
4     "message": "CORRECT 1.0 Points: Class Student was found."
5 }
```

The test engine's output mainly consists of positive messages. This is another reason why this thesis will pursue the second option of using the JSON files, even though the implementation effort is higher than just creating a JUnit result.
To create these JSON files an executable is needed. This executable will be called *Mars Output Converter*.

5.8 Sending the Results Back to Artemis

The *Jenkins Server Notification Plugin* is responsible for aggregating all files from the previous step and sending them back to Artemis. The plugin is used as is and will not be modified to increase the maintainability by not altering the transfer process of results.

[5] `https://github.com/ls1intum/jenkins-server-notification-plugin`, 13.05.2022

5.9 Grading the Assignment and Displaying the Feedback

Artemis grades the exercise by comparing the successful tests of the students' submission with the tests successful in the solution repository.

This means that if three out of ten test cases are successful, the student gets 30% of the points assigned to the exercise. Artemis also offers the possibility of changing the influence of single tests on the overall rating, but this is not necessary for the implementation of Mars.

Artemis already has the ability to display feedback. This functionality of Artemis will be reused. Slight alterations in the front end may be necessary for a more structured and organized view.

6 Implementation

This chapter will explain the steps needed for the integration in more detail. First, the prerequisites are presented. Then the creation of test cases from the output of the test engine is described. Furthermore, the structure of the pipeline for the evaluation of the student models is explained.

6.1 Prerequisites

Mars is going to be integrated using Artemis with the Jenkins and Gitlab stack.

The development was done on a branch named *rule_based_assessment*. The base commit for the branch is part of Artemis version 5.4.3 and has the commit hash *#20d3116*[1].

6.2 Adding Mars as a New Type of Programming Language

Adding Mars as a new programming language had to be done in the front and back end. This included adding it to the *programmingLanguage*[2] enumeration to allow for the creation of Mars programming exercises. Instructors are now able to create new programming exercises of the type Mars.

6.3 Specifying Repository Templates for an Effortless Task Setup

Every programming exercise has a template task to simplify creating a task for instructors. There is the exercise, solution, and test repository.

The following listing shows the repository directory structure with their included files. The listings omits some files in favor of better readability. The repositories will be further explained in the following sections.

[1] `https://Github.com/ls1intum/Artemis/commit/20d3116052918a1a3382e11883dade6745afbb3b`, 13.05.2022

[2] `https://github.com/ls1intum/Artemis/blob/d19b274081d5e3bb609a5863fa2c2a5ccac1351f/src/main/java/de/tum/in/www1/artemis/domain/enumeration/ProgrammingLanguage.java`, 13.05.2022

Listing 6.1 Directory structure of repository templates for Mars exercises. Folders and files are shown.

```
├─ Exercise
│  └─ baseModel.ooa_classdiagram
├─ Solution
│  └─ solutionModel.ooa_classdiagram
└─ Test
   ├─ Metamodels
   │  ├─ gradingmodel.ecore
   │  └─ oaa_classdiagram.ecore
   ├─ Tests
   │  ├─ test_app.evl
   │  ├─ rule_class_1.evl
   │  ├─ rule_class_2.evl
   │  └─ lib_output.evl
   ├─ TestEngine.jar
   └─ MarsOutputConverter.jar
```

6.3.1 Exercise

Purpose The exercise repository is the template for all student assignments. Whenever a student starts the exercise, a new student's repository is initialized with the contents of the exercise repository. When tested, all tests shall fail.

Structure The exercise repository contains only one pre-initialized Eclipse EMF model file (.ooa_classdiagram). Students shall put their assignments in this file. Besides the model file, a complete pre-initialized modeling project could be distributed using this repository.

6.3.2 Solution

Purpose The solution repository is repository with an model a student would submit. It contains an example solution. When tested, all tests shall pass.

Structure The solution repository also contains the Eclipse EMF model file (model.ooa_classdiagram). But this time, the file is **not** empty but contains the solution for the given task.

6.3.3 Test

Purpose The test repository is different from the former two repositories. It does not contain any models submitted by users. Instead, the repository contains all files necessary for checking and grading the student's assignments.

Structure The test repository includes all files necessary for grading the assignments. This includes a folder with the tests, the metamodels, the test engine, and the output converter. The tests are generated or written beforehand and then pushed to the test repository. The Jenkins pipeline then uses the tests to generate the grading.

6.4 Using Docker Containers for Increased Security

The newly created Mars exercises need a Docker container to run separated from the host system. The Docker image created is as minimalistic as possible.

Listing 6.2 Dockerfile to create the Docker image for running Mars exercises.

```
1 # Base on Ubuntu 20.04 LTS
2 FROM ubuntu:20.04
3 # Make sure all sources are up to date
4 RUN apt-get update
5 # Install apache ant
6 RUN apt install -y ant
```

As seen in the Dockerfile, the image is based on Linux Ubuntu and provides the software *Apache Ant*[3]. Apache Ant is used to run the rule-based test engine.

6.5 Specification of the Mars Output Converter

The *Mars Output Converter* is the necessary adapter between the test engine and the grading of Artemis. The goal is to convert the output to a format Artemis can handle.

6.5.1 Test Engine's Feedback Classes

The test engine provides five different feedback classes for the elements matched by the test engine. The following classes provide points to the solution Correct, Warning and the Error class. The other two classes do not provide points and are for informational purposes: Missing / Wrong and Info.

The feedback class will be prepended to the test message to allow students to receive the feedback.

6.5.2 Creation of Test Cases From the Results Objects

Artemis grades student submission using the concept of successful test cases, but since the test engine's output is not assignable to individual test cases, a method of converting the output to a list of test cases had to be developed. Test cases do not have points attached to them and can only succeed or fail.

The decision was made to treat the number of test cases as points. Each successful test case is worth a certain amount of points (e.g., 1 point). The test engine uses variable grading schemas. The smallest reachable points by the test is used as a base value.

Tests that do not provide points to the solution (Missing / Wrong, Info) will be marked as failed, but the message will be appended to the result JSON. Failed tests do not affect Artemis grading, but the messages are shown to the student.

When the student does not reach the maximum amount of points, another test with the status failed is appended, which carries the point result as a message -> *INFO: 12.5 / 23.5 Points reached*.

Table 6.1 shows the concept for three example submissions.

[3] `https://ant.apache.org/`, 13.05.2022

Table 6.1 Number of test cases sent back to Artemis for real-world student submissions of the *Social Network* task.

Assignment	Student Solution 1	Student Solution 2	Example Solution
Points Reached	11.5	16	23.5
Successful Test Cases	46	64	94
Unsuccessful Test Cases	1	1	0
Total Test Cases	47	65	95

Example

In the following example, we will use the grading schema of 0 points, 0.25 points, 0.5 points, and 1 point for correct tests. That means that for every one-point result, there must be four 0.25 points tests sent back to Artemis.

Listing 6.3 shows an example test engine result.

Listing 6.3 Example test engine results object.

```
1   <Result>
2    <ExpertObject>Student</ExpertObject>
3    <ExpertType>Class</ExpertType>
4    <TestObject>Student</TestObject>
5    <TestType>Class</TestType>
6    <Rule>Rule_R010101</Rule>
7    <Category>CORRECT</Category>
8    <Points>1.0</Points>
9    <Msg>Class Student was found.</Msg>
10  </Result>
```

The example result object is pointed with one point (line 8) and will therefore be converted to four JSON test case files. The four test case files are shown in the following listing. Only one test has a message attached to it (line 5). The others are successful but do not have a message. This allows later for a better display of the results.

Listing 6.4 Example JSON tests generated from the example results object.

```
1  Rule 1_0.json:
2  {
3      "name": "Rule: 1",
4      "successful": true,
5      "message": "CORRECT 1.0 Points: Class Student was found."
6  }
7
8  Rule 1_1.json / Rule 1_2.json / Rule 1_3.json
9  Example for Rule 1_1.json:
10 {
11     "name": "Rule: 1_1",
12     "successful": true,
13 }
```

6.6 Jenkins Pipeline

Jenkins is used to running the checker on the solutions the students have provided. The Jenkins pipeline is pretty similar to the pipelines already integrated into Artemis. The Jenkinsfile uses variables that Artemis replaces before creating a building plan in the actual Jenkins service. Variables are specified with an **hash sign** in front of the variable name (e.g. *#dockerImage*) A Jenkins pipeline consists of multiple stages. The following sections describe the stages of tasks in detail.

6.6.1 Specifying the Docker Image

The first step is to specify the agent, Jenkins uses Docker images to run the students submissions. In this case we are using an Docker agent. The URL for the Docker image is specified by the *#dockerImage* variable (line 3). The image for the Mars exercises is then automatically pulled from *Dockerhub*[4].
The following listing shows this step in detail. This step is identical to the Java Jenkins pipeline.

Listing 6.5 Step of specifying the agent

```
1  agent {
2      docker {
3          image '#dockerImage'
4          label 'docker'
5      }
6  }
```

6.6.2 Checking Out the Git Repositories

The next step is to check out the student's assignment repository and the tests repository to create one workspace to run the tests. The repository URLs *#testRepository* and *#assignmentRepository* are replaced by the corresponding Git repositories. The variables are used because the repositories differ by task and user.
This step is identical to the step in the Java Jenkins pipeline and shown in detail in the following listing.

Listing 6.6 Checking out the Git repositories

```
1      stage('Checkout') {
2          steps {
3              checkout([$class: 'GitSCM', branches: [[name: '*/master']],
                   doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [],
                    userRemoteConfigs: [[credentialsId: '#gitCredentials', name: 'tests', url: '#
                   testRepository']]])
4              dir('#assignmentCheckoutPath') {
5                  checkout([$class: 'GitSCM', branches: [[name: '*/master']],
                       doGenerateSubmoduleConfigurations: false, extensions: [],
                       submoduleCfg: [], userRemoteConfigs: [[credentialsId: '#gitCredentials',
                       name: 'assignment', url: '#assignmentRepository']]])
6              }
7          }
```

[4] `https://hub.docker.com/repository/docker/rfranzr/artemis-mars-docker`, 13.05.2022

6.6.3 Running the Test Engine and Converting the Output

The student's submission is run against the checker in the build phase. The functionality of the test engine was further described in Chapter 2.

Before the student assignment is checked, the output directory is created (line 7). The test engine is called using *Apache Ant*. The test engine's output is placed in the output directory.
The way how the test engine is executed, which model is taken as input, and where the output is placed is similar to the way Hamann proposed it [8].

Next, the *Mars Output Converters* output folder is created. Its contents are sent back to Artemis at the end of the pipeline using the *Jenkins Notification Plugin*. Afterward, the test engine's output file name is renamed to the input filename of the Mars Output Converter (line 13). The renaming increases the stability of cases where students have renamed their submission files. Finally, the test engine's output is converted by the Mars Output Converter to comply with Artemis's format for grading the assignment (line 14).

The following listing shows the described process in detail.

Listing 6.7 Running the test engine and converting the output with the Mars Output Converter.

```
1   stage('Build') {
2       steps {
3           timestamps {
4               sh '''
5                   cd $WORKSPACE
6                   rm -rf output
7                   mkdir -p output
8                   ant -f run.xml
9                   '''
10              sh '''
11                  rm -rf customFeedbacks
12                  mkdir customFeedbacks
13                  cp `ls -1 output/*.xml | head -1` output/testEngineOutput.xml
14                  java -jar MarsOutputConverter.jar
15                  '''
16          }
17      }
```

6.6.4 Notify Artemis with the Test Results

In the last step, the test results are sent to Artemis. The results from the feedbacks directory are aggregated and then sent back to Artemis using the *Jenkins Server Notification Plugin*[5]. The step of sending back the results is identical to the Java Jenkins Pipeline. The *notification plugin* was not modified for the implementation of Mars. This increases the maintainability.

The following listing shows how the *notification plugin* is called from within the Jenkins pipeline (line 3).

[5] `https://github.com/ls1intum/jenkins-server-notification-plugin`, 13.05.2022

Listing 6.8 Sending back the results by calling the Jenkins Server Notification Plugin

```
1        post {
2            cleanup {
3                sendTestResults credentialsId: '#jenkinsNotificationToken', notificationUrl: '#
                     notificationsUrl'
4                cleanWs()
5            }
6        }
```

6.7 Displaying the Results to Students

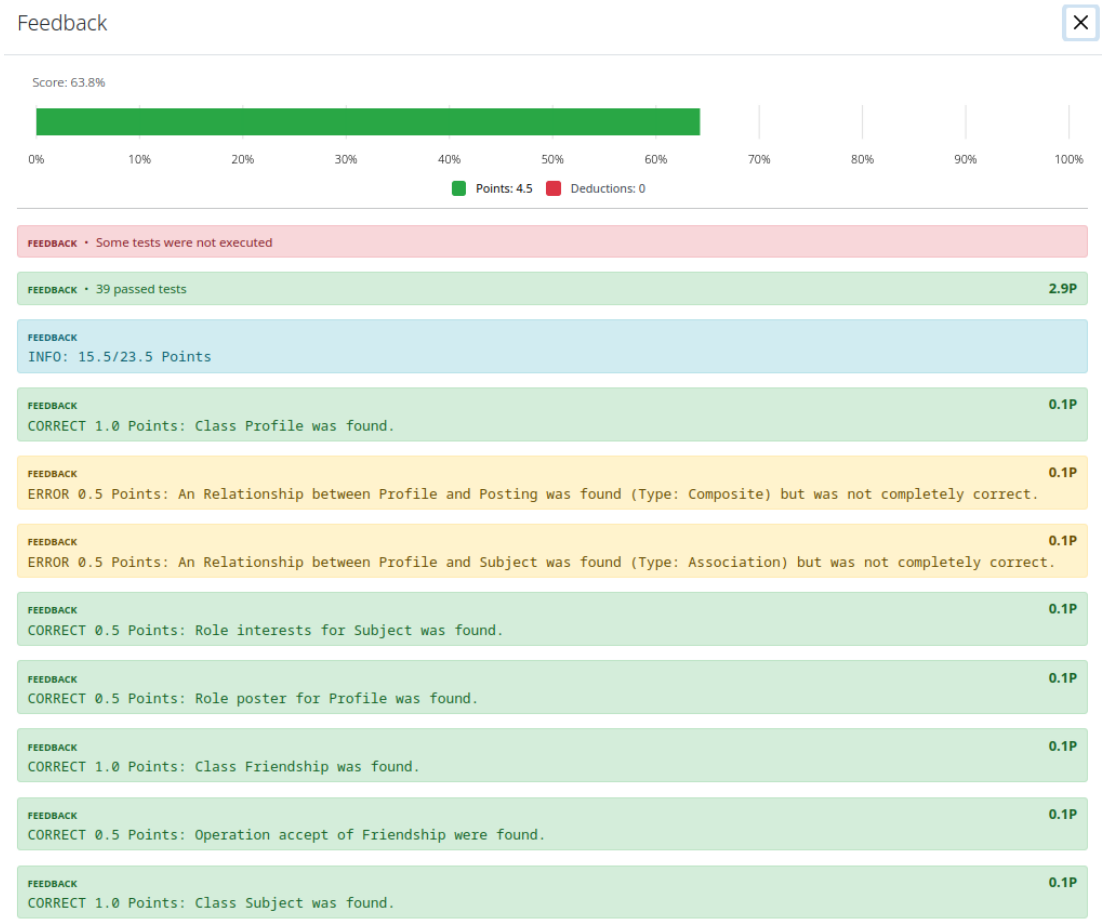

Figure 6.1 An example feedback page for a submission a student has made.

The results are displayed like they were test cases for a programming exercise. Slight adjustments were made to visualize the feedback more appealing and to not distract students from the essential feedback. To achieve this, all successful tests, which have no message are aggregated into one feedback. The same aggregation is done with tests not in the results object and therefore marked as not executed. The different feedback classes are visually differentiated by assigning them different colors. Figure 6.1 shows an example feedback page for a result a student has submitted. It shows how the feedback is presented and how the categories are distinguished.

6.8 Creating a Test Set to Ensure Stability

A test set was created to test the Mars Output Converter and the whole pipeline. Five example solutions were modeled for each of the two explanatory tasks. Both tasks were given in former exams of the Software Engineering Course at the TU Dresden. The tasks are called *Social Network* and *Module Chaos*.

The goal was to create models which are as different as possible. The model scores are all roughly 50% of the maximum achievable score. The test set includes the models and the associated test engine outputs. The pipeline was manually tested by submitting the models and asserting the feedback. Furthermore, the output converter is tested with unit tests using the test engine outputs as an input for the converter.

7 Evaluation

This chapter is going to evaluate the integration of the newly implemented rule-based assessment system Mars. This is trying to answer the fourth research question RQ4.

7.1 Design

This section will explain the evaluation design. A quantitative analysis that benchmarks the run time of Mars and a qualitative analysis evaluating the user experience were conducted.

7.1.1 Quantitative Analysis of the Pipeline Run Time

Assessment speed is an essential criterion since it directly affects the student's motivation to learn. Therefore the assessment should be as fast as possible. All successful runs of two example Mars exercises are compared to the run time of a Java exercise. The run times are measured by the Jenkins continuous integration server. The Jenkins server runs on a machine with an eight-core Intel Core i7-8550U, 16 gigabytes of ram, and with the operating system Manjaro Linux. Artemis is installed across two computers. Jenkins and Gitlab run on the already mentioned machine, Artemis Client / Server, and the MySQL database runs on a second machine. Both machines are connected to the same local area network.

7.1.2 Qualitative Analysis of the User Experience

A user survey was conducted to evaluate the quality of the proposed integration from a user's perspective. The user's task was to create a Mars exercise using the template exercise and then try to solve the created exercise. The *Eclipse Modeling Tools*[1], containing the editor plugin used to create models, were preinstalled. The exercise used in this study is an old exam exercise from the software engineering module at TU Dresden and was given as an Artemis template.

The survey consisted of a system usability scale (SUS). The SUS was proposed already 1995 by Brooke. It consists of ten questions and has a reliabilty score of 0.85. The system usability scale tries to make usability comparable across various contexts [6].

[1] `https://www.eclipse.org/downloads/packages/release/juno/sr1/eclipse-modeling-tools`, 13.05.2022

Table 7.1 Overview of longest, shortest, and average pipeline run times for different tasks.

Task	Mars: Module Chaos	Mars: Social Network	Java: Sorting Strategies
Number of model elements	34	31	-
Longest pipeline run	14.0s	13.0s	16.0s
Shortest pipeline run	8.7s	8.4s	9.4s
Average run time	9.7s	9.9s	11.6s

Furthermore, the participants were asked what they like and what they do not like about Mars using free-text answers. Five participants completed the survey. All participants had prior experience in modeling and had already passed the software engineering module *Software Technology 1*[2] at TU Dresden.

The free text questions included, but were not limited to:

- What are your opinions on creating a Mars exercise?
- How do you like the feedback?
- What do you like the most?
- What problems occurred during the use of Mars?
- What would you like to improve?

7.2 Results

This section shows the results and findings for the study conducted. Quantitative- and qualitative analysis are presented separately. Tables and diagrams are used to present the results more visually.

7.2.1 Results of the Quantitative Analysis of the Pipeline Run Time

The pipeline run time depends on pulling and building the Docker image, running the test engine, converting the output, and sending back the test results to Artemis. The pipeline run time depends on pulling and building the Docker image, running the test engine, converting the output, and sending back the test results to Artemis. This thesis integration adds the run time of the test engine and the output converter and does not influence the use of the Docker image and the notification plugin.

Table 7.1 compares the pipeline run times between two Mars tasks and the Artemis Java default task. Both Mars exercises are tasks taken from old exams held at TU Dresden. The scope of the Java exercise is smaller, therefore, not large enough to be asked in an exam. The table shows that the average run time of the pipeline does not strongly depend on the number of model elements. Furthermore, the average run time of the Mars exercises was lower than the average run time of the Java example. This is an important finding and concludes that the system is scalable and practical enough for tasks of a typical size. That the scope of the Mars exercises is greater than the scope of the Java exercise is a significant benefit of the proposed integration.

[2] https://tu-dresden.de/ing/informatik/smt/st/studium/lehrveranstaltungen?leaf=1&lang=de&subject=446&embedding_id=47eddfa7c5a54ed5be49042aff35a31b, 13.05.2022

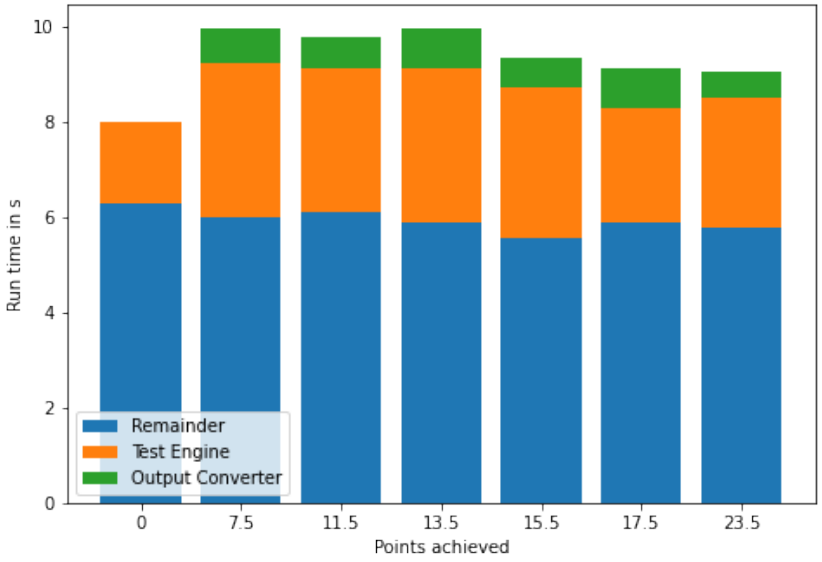

Figure 7.1 Run time differences between the different components over the achieved points by the submitted solution. The chart is ordered by the points achieved.

Figure 7.1 shows the pipeline run time over the submissions points. The remainder includes the pipeline and Docker setup. The Git repository checkout and the runtime of the server notification plugin. Zero points represent a submission with syntactical errors, which failed the build. The chart shows a slight decrease in run time, with greater points achieved on successful builds. This is expected and property of rule-based assessment systems.

The run time of the test engine fluctuates between 2.4s (17.5 point run) and 3.3s (7.5 point run). The worse the output, the more additional tests, and alternatives have to be run. The Mars Output Converters' run time is more constant between 0.5s and 0.8s. This is to be expected. The run time directly depends on the number of result objects which need to be converted.

The chart also shows that running the test engine and the Output Converter roughly takes 40% of the whole pipeline run. That means the run time is dominated by the setup of the Docker image and the notification plugin, both not altered by the proposed integration of this thesis.

7.2.2 Results of the Qualitative Analysis of the User Experience

Identifying and organizing the comments relevant to the integration of Mars was not easily done. This is because the evaluating students came in contact with a lot of different systems, and the proposed integration has not always control over the other systems. An example is that missing feedback for the models was criticized, but the feedback is solely dependent on the test set used and not directly part of the Mars integration.

System Usability Scale

The system usability scale gives a general overview of the system's usability. It helps identify the existence of usage problems, but not where the problems are concretely located. The scale ranges from 0 to 100 points.

After calculating the results, the average score is 74.5 with the lowest score of 70 and the highest score of 80. According to Bangor et al., a score of 74.5 is considered "good" [2, pp. 591–593].

Free Text Answers

Some questions were targeted to the creation of exercises. Only a tiny amount of instructors do this step. Students will not come into contact with that process. Therefore, creating a new exercise is not as important as the process of students submitting models for grading. The more essential students' user experience was surveyed too.

The following list represents a selection of the most meaningful answers. Answers which are not directly linked to the integration of Mars are excluded. The answers were rephrased for a better understanding. The number of evaluators supporting the comment is specified behind the answer.

Positive Feedback

- Setting up the Mars exercise (as tutor):
  - The example task helps in understanding how to create own tasks. (2/5)
- Submitting the Mars exercise (as student):
  - Independent modeling with immediate feedback helps me improve my modeling skills. (5/5)
  - In my opinion, the feedback is intuitive. (3/5)
  - The different coloring helps me to categorize the feedback better. (2/5)

Negative Feedback

- Setting up the Mars exercise (as tutor):
  - I was expecting the creation under the modeling exercise category. Why is Mars a programming exercise? (5/5)
- Submitting the Mars exercise (as student):
  - I couldn't upload a file. I had to copy and paste the contents. (5/5)
  - Submitting the model's works, but I am missing a graphical model editor. (1/5)

All evaluators were not expecting Mars to be created under the programming exercises. This can easily be remedied with comprehensive documentation or a message displayed in the creation of a modeling exercise pointing to the programming exercises.

The first evaluating person mentioned that the feedback given was unorganized. Afterward, the feedback displayed was adjusted by better aggregating failed and successful tests. The following evaluations did not mention the feedback to be unorganized anymore. Further improvements can be made by grouping the feedback by matched element type (e.g., class, relation).

Unexpectedly only one out of the five evaluators mentioned an online editor as missing. *Jack*, an evaluation system of the University of Duisburg Essen, also uses the same method of downloading and uploading the submission[3]. All evaluators were unanimous in that independent learning with instant feedback positively affects their modeling skills. This shows that an online editor is **not** necessary for the successful use of Mars in university courses.

The negative feedback given will be further addressed in future work, Chapter 8.

7.3 Discussion of the Results

The evaluation shows that Mars is a seamless integration to Artemis, works well, and can already be used in software engineering courses. This is backed by the result of the system usability scale and the positive feedback from the free-text answers. Regarding displaying feedback, minor adjustments like categorizing the feedback by element type (class, relation) or hiding the points Artemis assigns to a test might clear all ambiguities.

Furthermore, the run time for the grading of Mars exercises is compelling as it even outperforms the run time of Java exercises.

The necessary part of designing and creating own tasks was not evaluated since it is not within the scope of this thesis. It seems that the rules and tests of the test engine are rather hard to grasp with a high complexity for instructors. One bright spot is that task design only needs to be done once.

7.4 Limitations of the Evaluation

This section tries to analyze the limitations of the evaluation regarding the validity of the results. The focus lies on qualitative feedback.

The first limitation follows from the small number of evaluators. The number is minimal and not enough to obtain reliable results. Furthermore, only one task was evaluated, and therefore potential usability issues with other tasks were not identified. A more extensive survey catered to students using the system in production might be conducted to solve this limitation.

Another limitation is concerned with the wording and understanding of the questions asked. This was limited by conducting the evaluations in presence. The evaluators had the chance to ask when something was unclear.

That the whole pipeline was evaluated brings the downside of not easily getting feedback regarding the Mars integration since the feedback is also targeting other parts of the system out of the control of Mars.

The use of the system usability scale brings all their limitations, like no concrete hints on weaknesses of the system or the nondifferentiability between single tests with it.

[3] `https://jack-community.org/wiki/index.php/Aufgabentyp_UML`, 13.05.2022

8 Conclusion and Future Work

Previous chapters have compared current systems (Chapter 3) and identified a suitable starting point for the integration of the rule-based evaluation system *Mars* in *Artemis* (Chapter 4). It has been shown that the new evaluation system could be successfully integrated into Artemis. After the description of the implementation in Chapter 6 and the subsequent evaluation in Chapter 7, a conclusion can be drawn. The Research Questions are answered and potential future work is presented.

8.1 Research Questions

The following research questions got defined in Chapter 1. This thesis's success is measured by answering the research questions, which are summarized in this section.

RQ1 What are the current technologies used to automate the feedback for modeling exercises?

An answer to this question can be found in Chapter 3. Ten different systems were analyzed and categorized by their approach and the feedback they output. Seven systems were similarity-based. The other three systems use a rule-based approach to assess models. Furthermore, a comparison was developed to help future instructors decide on the best system which suits their needs.

RQ2 What are ways to integrate Inloom into Artemis?

In Chapter 4, three potential approaches for the integration were identified. These approaches were then compared against each other using previously defined evaluation criteria. Potential weaknesses were addressed as well as strengths of the proposed approaches. The approach chosen is to add Mars as a new type of programming language. The following chapters were then implementing this approach.

RQ3 What is the best way for the integration of Inloom?

The identified approach of adding Mars as a new type of programming language from Chapter 4 was now being implemented. Chapter 5 gives an overview of the proposed system design for the integration, while the following Chapter 6 concretely implements the rule-based system. Technical details are explained here. For example, a converter converting the test engine's output to test cases was written. Furthermore, a continuous integration pipeline assessing students' submissions was created. A usable system was designed and implemented at the end of the implementation chapter.

RQ4 How does the newly integrated system performs when used by students?

The goal of this research question was to get real-world feedback and evaluate the proposed integration of the feedback. This research question was therefore answered in Chapter 7. Five students tested the system. As a result of the feedback, minor adjustments in displaying the feedback were already made. The integration got a total system usability score of 74.5. Potential future work got also derived from the student's feedback.

8.2 Future Work

A few topics could not be addressed during this thesis or came up while writing this thesis. These topics may be a good starting point for improvements or future research.

8.2.1 Deploy Web Based UML Editor

Even though only one out of the five evaluators has wished for an online model editor, an editor is great for not having to download files on their computer and upload them again. An online editor would improve the overall user experience flow and minimize the need for lengthy introductions. The *Apollon* web modeling editor, which is already integrated with Artemis, might be reused for that purpose. The editor already supports different model types, which the automatic assessment can support all. The editor's output has to be in EMF form or converted.

8.2.2 Improving the Creation of the Mars Exercises

Currently, Mars exercises are just created within the same dialog as programming exercises. In the future, Mars exercises should be created in their own dialog. This would also allow the ability to create the necessary test set for Mars exercises. Creating the test set is currently done offline on the instructor's machine. The new dialog could leverage the created web-based UML editor to automatically create the test set out of the example solution. The new creation dialog directly mitigates the problem of instructors not finding the correct dialog to create Mars exercises.

8.2.3 Future Work Regarding the Rule-Based Assessment System

Future work regarding the rule-based assessment system proposed by Hamann [8] is still relevant. Future work, mentioned in this thesis, includes creating a "Test Suite for Master Constraint Sets" or the "Assessment of Textual Task Descriptions".

8.3 Conclusion

E-Learning systems have become an integral part of university education in recent years. Modeling is one of the foundational pillars of software engineering. It is becoming more important at universities, and therefore the demand for automatic assessment systems has grown.

This thesis gives an overview of the already existing systems and compares the two main approaches to help instructors of software engineering courses to choose the right system suitable to their needs.

The Faculty of Computer Science at Technische Universität Dresden will use Artemis, an interactive learning system for software engineering courses in the future. Unfortunately, Artemis currently does not support the automatic assessment system with instant feedback for models. Inloom is such a rule-based assessment system with instantaneous feedback, which this thesis successfully implemented into Artemis.

Course instructors now have the option to create modeling tasks that are assessed by the Inloom test engine. This thesis provides a ready for use task template to simplify the setup process. Students now can submit their models and receive instant feedback with a grade of their submission. The models currently have to be created using an offline modeling editor. Integrating an online modeling editor is left for future work.

However, only one out of five students mentioned the missing editor, so the new system, called Mars, is ready to be tested in software engineering courses.

Bibliography

[1] Outair Anas, Tanana Mariam, and Lyhyaoui Abdelouahid. “New method for summative evaluation of UML class diagrams based on graph similarities”. In: *International Journal of Electrical and Computer Engineering* 11 (2 Apr. 2021), pp. 1578–1590.

[2] Bangor et al. “The System Usability Scale (SUS): an Empirical evaluation”. In: *International Journal of Human-Computer Interaction* 24 (Aug. 2008), pp. 574–594.

[3] Weiyi Bian, Omar Alam, and Jörg Kienzle. “Is automated grading of models effective?: Assessing automated grading of class diagrams”. In: Association for Computing Machinery, Inc, Oct. 2020, pp. 365–376.

[4] Weiyi Bian, Omar Alam, and Jorg Kienzle. “Automated grading of class diagrams”. In: Institute of Electrical and Electronics Engineers Inc., Sept. 2019, pp. 700–709.

[5] Younes Boubekeur, Gunter Mussbacher, and Shane McIntosh. “Automatic assessment of students’ software models using a simple heuristic and machine learning”. In: Association for Computing Machinery, Inc, Oct. 2020, pp. 84–93.

[6] John Brooke. “SUS: A quick and dirty usability scale”. In: *Usability Eval. Ind.* 189 (Nov. 1995), pp. 189–194.

[7] Hayden Cheers et al. “Exploring a Comprehensive Approach for the Automated Assessment of UML”. In: Institute of Electrical and Electronics Engineers Inc., July 2019, pp. 133–139.

[8] Markus Hamann. “AUTOMATIC FEEDBACK FOR UML MODELING EXERCISES AS AN EXTENSION OF INLOOP”. 2020.

[9] Yuta Ichinohe et al. “Effectiveness of automated grading tool utilizing similarity for conceptual modeling”. In: vol. 108. Springer Science and Business Media Deutschland GmbH, 2019, pp. 117–126.

[10] Stephan Krusche. *Semi-Automatic Assessment of Modeling Exercises using Supervised Machine Learning*. Jan. 2022.

[11] Stephan Krusche and Andreas Seitz. “ArTEMiS - An automatic assessment management system for interactive learning”. In: vol. 2018-January. Association for Computing Machinery, Inc, Feb. 2018, pp. 284–289.

[12] Martin Morgenstern and Birgit Demuth. *Continuous Publishing of Online Programming Assignments with INLOOP*. 2018, pp. 32–33.

[13] Tobias Reischmann and Herbert Kuchen. “A web-based e-assessment tool for design patterns in UML class diagrams”. In: vol. Part F147772. Association for Computing Machinery, 2019, pp. 2435–2444.

[14] Tobias Reischmann and Breno Menezes. “Application of Swarm-intelligent Methods to Optimize Error-tolerant Graph Matching for Automatic E-Assessment”. In: Institute of Electrical and Electronics Engineers Inc., Nov. 2019.

[15] Dave R. Stikkolorum et al. “Towards automated grading of UML class diagrams with machine learning”. In: *BNAIC/BENELEARN*. Vol. 2491. CEUR-WS, 2019.

[16] M Striewe and M Goedicke. *Automated Checks on UML Diagrams*. ACM, 2011, pp. 38–42.

9 Appendix

9.1 Setup

This section explains how to setup the assessment of Mars exercises.

9.1.1 Creation of Mars Exercises

- Install Artemis[1] using the Gitlab / Jenkins Stack. The Atlassian stack is **not** supported. Documentation on how to install Artemis on your machine can be found within the Artemis repository. Some Dockerfiles got altered to get the setup working. These files can be found in the digital attachments for this thesis.
- Login in Artemis as administrator and create a new *Course* under the course management tab.
- Enter the course overview and click *create new exercise*.
- Select *programming exercise* as the exercise type and enter the dialog.
- Select *Mars* as the *programming language*, follow the dialog, and adopt the preloaded Mars task template

9.1.2 Assess a Model Using Mars

- Go to the *Course Overview* and join the previously created course
- Start the previously created exercise
- Open the online code editor or clone the repository
- Replace the model.ooa_classdiagram file with a submission (Do not rename the file!)
- After submitting, the feedback is displayed in the feedback section of the editor

An example submission is provided in the next section. For setup of the eclipse model editor, files provided by Hamann are needed. Information for the setup can be found in the appendix of Hamanns work [8].

[1] `https://github.com/ls1intum/Artemis`, 13.05.2022

9.1.3 List of Files Modified by This Thesis

- .jhipster/ProgrammingExercise.json
- artemis.jh
- src/main/java/de/tum/in/www1/artemis/domain/enumeration/ProgrammingLanguage.java
- src/main/java/de/tum/in/www1/artemis/service/connectors/ContinuousIntegrationService.java
- src/main/java/de/tum/in/www1/artemis/service/connectors/bamboo/BambooBuildPlanService.java
- src/main/java/de/tum/in/www1/artemis/service/connectors/jenkins/JenkinsBuildPlanService.java
- src/main/java/de/tum/in/www1/artemis/service/connectors/jenkins/JenkinsProgrammingLanguageFeatureService.java
- src/main/java/de/tum/in/www1/artemis/service/programming/ProgrammingExerciseService.java
- src/main/java/de/tum/in/www1/artemis/service/programming/TemplateUpgradePolicy.java
- src/main/resources/templates/jenkins/mars/regularRuns/Jenkinsfile
- src/main/resources/templates/mars
- src/main/webapp/app/exercises/programming/manage/update/programming-exercise-update.component.html
- src/main/webapp/app/exercises/programming/manage/update/programming-exercise-update.component.ts
- src/main/webapp/app/exercises/programming/shared/code-editor/file-browser/supported-file-extensions.ts
- src/main/webapp/app/exercises/shared/result/result-detail.component.ts
- src/main/webapp/app/exercises/shared/result/result-detail.scss
- src/main/webapp/i18n/de/programmingLanguage.json
- src/main/webapp/i18n/en/programmingLanguage.json

9.2 Digital Attachments

Further files like the program code of Artemis and the Mars Output Converter can be found in the digital attachments of this thesis. This also includes all files created during the evaluation of this thesis.

9.3 Complete Students Submission

This example submission is for the *Social Network* task, since the template task is the same.

```
<?xml version="1.0" encoding="ASCII"?>
<ooa_classdiagram:OOAClassModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi
      ="http://www.w3.org/2001/XMLSchema-instance" xmlns:ooa_classdiagram="http://www.inloom.
      org/ooa_classdiagram" ID="SocialNetworks">
  <classes xsi:type="ooa_classdiagram:Class" name="Profile">
    <properties name="username"/>
    <operations name="invite"/>
    <operations name="accept"/>
    <operations name="decline"/>
    <operations name="follow"/>
    <operations name="post"/>
    <operations name="comment"/>
  </classes>
  <classes xsi:type="ooa_classdiagram:Class" name="Friendship"/>
  <classes xsi:type="ooa_classdiagram:Enumeration" name="Status">
    <literalgroups>
      <literals name="PENDING"/>
      <literals name="ACCEPTED"/>
      <literals name="DECLINED"/>
    </literalgroups>
  </classes>
  <classes xsi:type="ooa_classdiagram:Class" name="Posting">
    <properties name="text"/>
```

```
  </classes>
  <classes xsi:type="ooa_classdiagram:Class" name="Comment">
    <properties name="text"/>
  </classes>
  <classes xsi:type="ooa_classdiagram:Class" name="Subject">
    <properties name="name"/>
  </classes>
  <relationships id="r1">
    <relationshipEnds upper="1" lower="1" class="Profile" id="r1e1">
      <role name="inviter"/>
    </relationshipEnds>
    <relationshipEnds upper="1" lower="1" class="Profile" id="r1e2">
      <role name="invitee"/>
    </relationshipEnds>
    <associationclassEnd class="Friendship"/>
  </relationships>
  <relationships id="r4">
    <relationshipEnds upper="1" lower="1" type="shared" class="Posting" id="r4e1">
      <role name="commented"/>
    </relationshipEnds>
    <relationshipEnds upper="-1" class="Comment" id="r4e2"/>
  </relationships>
  <relationships name="" id="r5">
    <relationshipEnds upper="-1" class="Comment" id="r5e1"/>
    <relationshipEnds upper="1" lower="1" type="shared" class="Profile" id="r5e2">
      <role name="commenter"/>
    </relationshipEnds>
  </relationships>
  <relationships id="r6">
    <relationshipEnds upper="-1" type="shared" class="Profile" id="r6e1"/>
    <relationshipEnds upper="-1" class="Subject" id="r6e2">
      <role name="interests"/>
    </relationshipEnds>
  </relationships>
  <relationships id="r3">
    <relationshipEnds upper="1" lower="1" type="shared" class="Profile" id="r3e1">
      <role name="poster"/>
    </relationshipEnds>
    <relationshipEnds upper="-1" class="Posting" id="r3e2"/>
  </relationships>
  <relationships id="r2">
    <relationshipEnds upper="1" lower="1" type="shared" class="Friendship" id="r2e1"/>
    <relationshipEnds upper="1" lower="1" class="Status" id="r2e2">
      <role name="status"/>
    </relationshipEnds>
  </relationships>
</ooa_classdiagram:OOAClassModel>
```

9.4 Example Test Engine Output

```
<?xml version="1.0" encoding="UTF-8"?>
<TestResult>
 <TestData>
  <ExpertModel>New_OOA_Class_Model</ExpertModel>
  <TestModel>New_OOA_Class_Model</TestModel>
  <MetaModel>OOAClassModel</MetaModel>
  <MCSIdentifier>mcs_inloom_ooa_class</MCSIdentifier>
  <MCSVersion>1.1.0</MCSVersion>
 </TestData>
 <Results>
  <Result>
```

```
    <ExpectObject>Chair</ExpectObject>
    <ExpectType>Class</ExpectType>
    <TestObject>Chair</TestObject>
    <TestType>Class</TestType>
    <Rule>Rule_R010101</Rule>
    <Category>CORRECT</Category>
    <Points>1.0</Points>
    <Msg>Class Chair was found.</Msg>
   </Result>
   <Result>
    <ExpectObject>Student</ExpectObject>
    <ExpectType>Class</ExpectType>
    <TestObject>Student</TestObject>
    <TestType>Class</TestType>
    <Rule>Rule_R010101</Rule>
    <Category>CORRECT</Category>
    <Points>1.0</Points>
    <Msg>Class Student was found.</Msg>
   </Result>
   <Result>
    <ExpectObject>Student_name</ExpectObject>
    <ExpectType>Property</ExpectType>
    <TestObject>Student_name</TestObject>
    <TestType>Property</TestType>
    <Rule>Rule_R020101</Rule>
    <Category>CORRECT</Category>
    <Points>0.5</Points>
    <Msg>Property name of Student were found.</Msg>
   </Result>
   <Result>
    <ExpectObject>r3</ExpectObject>
    <ExpectType>Relationship</ExpectType>
    <TestObject>r3</TestObject>
    <TestType>Relationship</TestType>
    <Rule>Rule_R080101</Rule>
    <Category>CORRECT</Category>
    <Points>1.0</Points>
    <Msg>A Relationship between Student and Chair was found (Type: Association).</Msg>
   </Result>
   <Result>
    <ExpectObject>r2e2_chairs</ExpectObject>
    <ExpectType>Role</ExpectType>
    <TestObject>r2e2_chairs</TestObject>
    <TestType>Role</TestType>
    <Rule>Rule_R100101</Rule>
    <Category>CORRECT</Category>
    <Points>0.5</Points>
    <Msg>Role chairs for Chair was found.</Msg>
   </Result>
  </Results>
  <ResultPoints>
   <MaxPoints>9.5</MaxPoints>
   <TestPoints>7.5</TestPoints>
  </ResultPoints>
</TestResult>
```

9.5 Example jUnit Results XML

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="Mars Output">
      <testcase name="Successful Testcase 1"/>
      <testcase name="Successful Testcase 2"/>
      <testcase name="Failed Testcase 3">
```

```
        <failure>Test Failure Message</failure>
    </testcase>
</testsuite>
```

9.6 Full Jenkins Pipeline

```
// ARTEMIS: JenkinsPipeline

pipeline {
    options {
        timeout(time: #jenkinsTimeout, unit: 'MINUTES')
    }
    agent {
        docker {
            image '#dockerImage'
            label 'docker'
        }
    }
    stages {
        stage('Checkout') {
            steps {
                checkout([$class: 'GitSCM', branches: [[name: '*/master']],
                    doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [],
                    userRemoteConfigs: [[credentialsId: '#gitCredentials', name: 'tests', url: '#
                    testRepository']]])
                dir('#assignmentCheckoutPath') {
                    checkout([$class: 'GitSCM', branches: [[name: '*/master']],
                        doGenerateSubmoduleConfigurations: false, extensions: [],
                        submoduleCfg: [], userRemoteConfigs: [[credentialsId: '#gitCredentials',
                        name: 'assignment', url: '#assignmentRepository']]])
                }
            }
        }
        stage('Build') {
            steps {
                timestamps {
                    sh '''
                        cd $WORKSPACE
                        rm -rf output
                        mkdir -p output
                        ant -f run.xml
                        '''
                    sh '''
                        rm -rf customFeedbacks
                        mkdir customFeedbacks
                        cp `ls -1 output/*.xml | head -1` output/testEngineOutput.xml
                        java -jar MarsOutputConverter.jar
                        '''
                }
            }
        }
    }
    post {
        cleanup {
            sendTestResults credentialsId: '#jenkinsNotificationToken', notificationUrl: '#
                notificationsUrl'
            cleanWs()
        }
    }
}
```