SOFTWARE ENGINEERING
MEDIENINFORMATIK

# Introducing GitHub Classroom into a Formal Methods Module

Soaibuzzaman and Jan Oliver Ringert

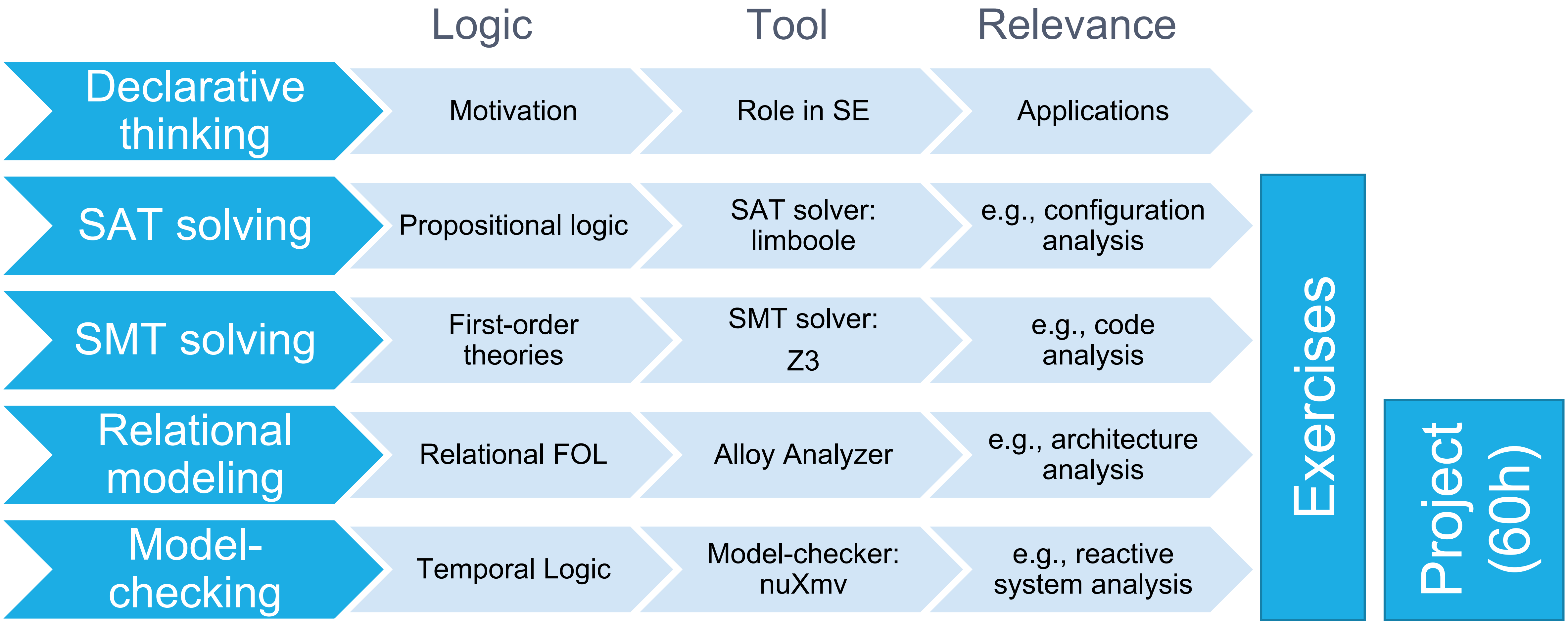
Bauhaus-Universität
Weimar

# Context

- Formal Methods for Software Engineering module of 6 ECTS

- Students:
  - MSc Digital Engineering
    - Computer science or engineering background
    - Majority: Civil, electrical, or mechanical engineering
  - MSc Computer Science for Digital Media
    - Classic computer science background
  - MSc Human-Computer Interaction
    - Mixed backgrounds (computer science, psychology, etc.)

# Module: Formal Methods for Software Engineering



|  | Logic | Tool | Relevance |
|---|---|---|---|
| Declarative thinking | Motivation | Role in SE | Applications |
| SAT solving | Propositional logic | SAT solver: limboole | e.g., configuration analysis |
| SMT solving | First-order theories | SMT solver: Z3 | e.g., code analysis |
| Relational modeling | Relational FOL | Alloy Analyzer | e.g., architecture analysis |
| Model-checking | Temporal Logic | Model-checker: nuXmv | e.g., reactive system analysis |

# Assessment of Module

- Students work (individually or) in pairs
  - Winter 22/23 we had 80 students and let them work in pairs/groups
  - Winter 23/24 we wanted to move to individual submissions, but we forgot to remove this from the slides, so students worked in pairs

- Passing each assignment is mandatory to do a project
  - assignments are intended to prepare students for projects
  - Examples:
    - Enumeration of interesting SAT/SMT solutions
    - deep vs. shallow embedding of feature models with cardinalities into Alloy

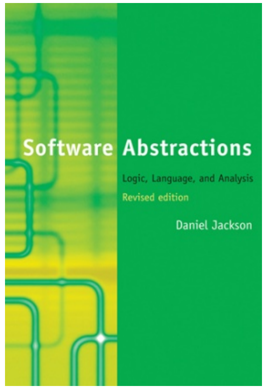
- Mark is 100% based on project results
  - Implementation, Report, Presentation

# Assignments

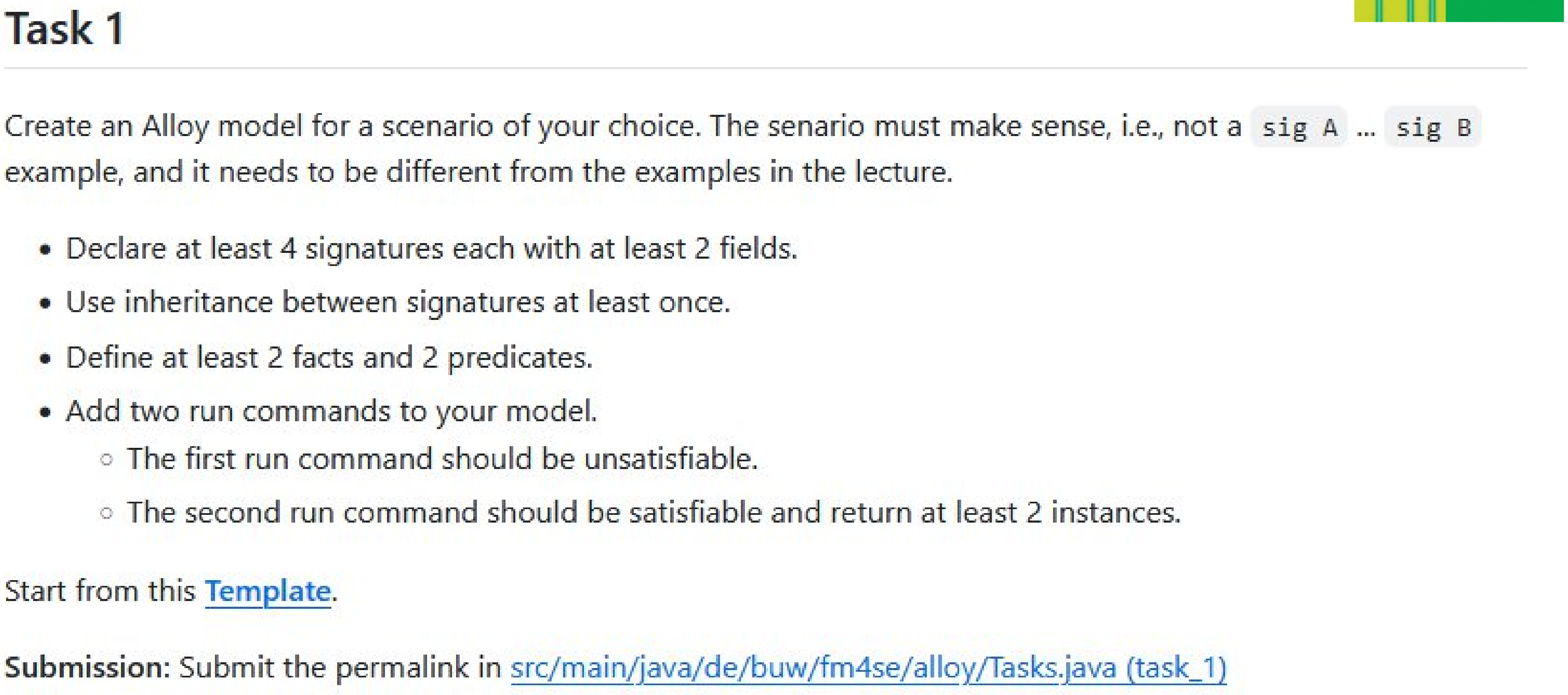
- Assignments come in alternating categories
  - spec: Manual Specification writing
  - impl: Automating a translation of problems to specifications

1. SAT spec: formulas, checking conclusions, verifying Role-Based-Access
2. SAT impl: Feature Model analysis [21], dead features, product preservation
3. SMT spec: Agatha puzzle [24, P.55], math puzzle, PC configuration
4. SMT impl: PC configuration from CSV-files, budget and purpose
5. Alloy spec: domain model, Agatha puzzle [24, P.55], Trash can [19]
6. Alloy impl: Analysis of Alloy modules: dead signatures, minimal scopes
7. nuXmv spec: LTL equivalence, counterexamples, chess knight moves

# Example Tasks



- Alloy spec:



## Task 1

Create an Alloy model for a scenario of your choice. The senario must make sense, i.e., not a `sig A` ... `sig B` example, and it needs to be different from the examples in the lecture.

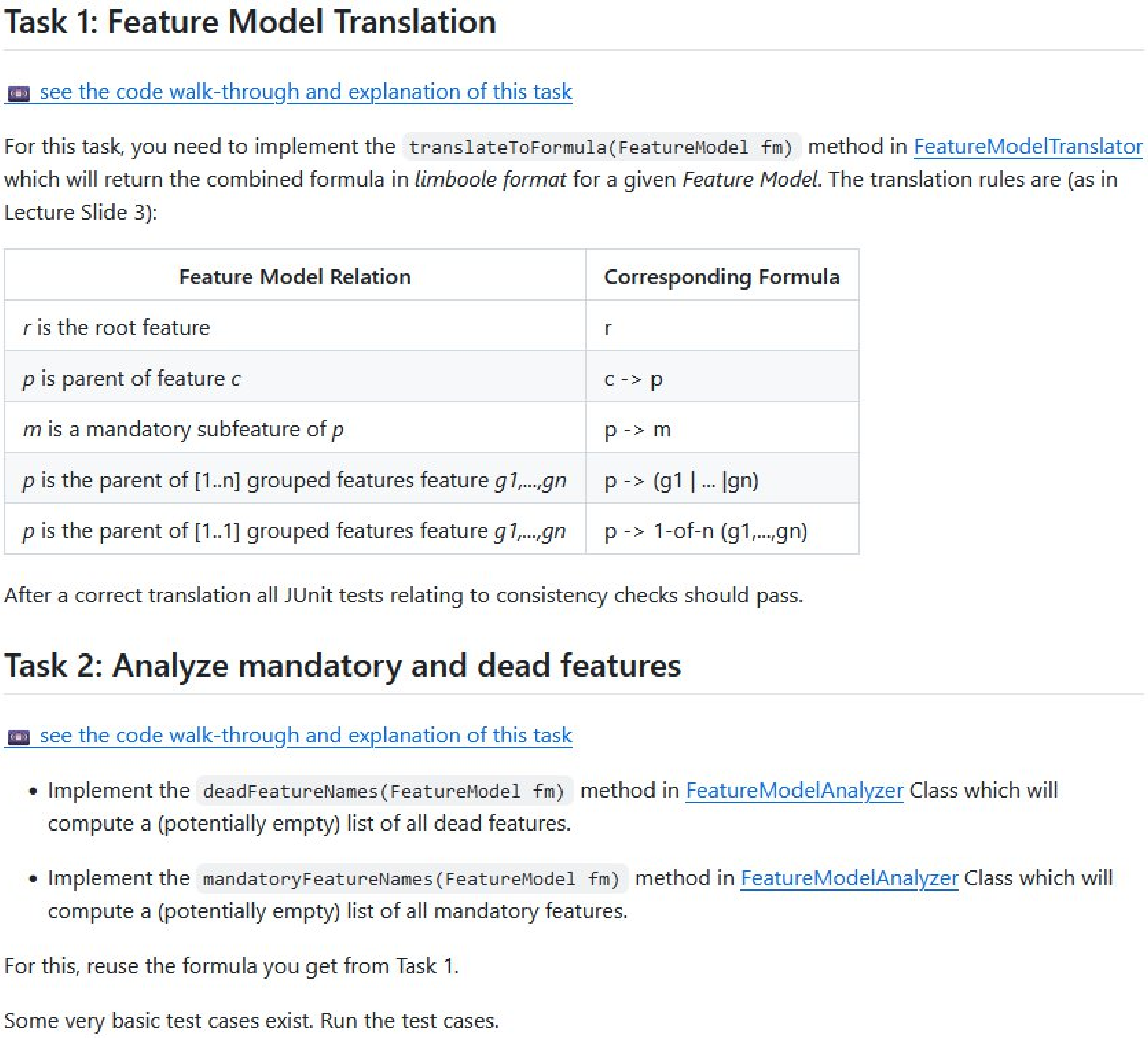
- Declare at least 4 signatures each with at least 2 fields.
- Use inheritance between signatures at least once.
- Define at least 2 facts and 2 predicates.
- Add two run commands to your model.
  - The first run command should be unsatisfiable.
  - The second run command should be satisfiable and return at least 2 instances.

Start from this **Template**.

**Submission:** Submit the permalink in src/main/java/de/buw/fm4se/alloy/Tasks.java (task_1)

# Example Tasks

- SAT impl



## Task 1: Feature Model Translation

see the code walk-through and explanation of this task

For this task, you need to implement the `translateToFormula(FeatureModel fm)` method in FeatureModelTranslator which will return the combined formula in *limboole format* for a given *Feature Model*. The translation rules are (as in Lecture Slide 3):

| Feature Model Relation | Corresponding Formula |
|---|---|
| *r* is the root feature | r |
| *p* is parent of feature *c* | c -> p |
| *m* is a mandatory subfeature of *p* | p -> m |
| *p* is the parent of [1..n] grouped features feature *g1,...,gn* | p -> (g1 \| ... \|gn) |
| *p* is the parent of [1..1] grouped features feature *g1,...,gn* | p -> 1-of-n (g1,...,gn) |

After a correct translation all JUnit tests relating to consistency checks should pass.

## Task 2: Analyze mandatory and dead features

see the code walk-through and explanation of this task

- Implement the `deadFeatureNames(FeatureModel fm)` method in FeatureModelAnalyzer Class which will compute a (potentially empty) list of all dead features.
- Implement the `mandatoryFeatureNames(FeatureModel fm)` method in FeatureModelAnalyzer Class which will compute a (potentially empty) list of all mandatory features.

For this, reuse the formula you get from Task 1.

Some very basic test cases exist. Run the test cases.

# GitHub Classroom

- Platform to create assignments for students
  - Creates task GIT repositories for each student who takes the assignment
  - Students submit by pushing to their repository
  - Supports automation for grading
  - Supports synchronization with Learning Management Systems, e.g., moodle

- Provided free as part of GitHub Education for teachers

# Goals of Migration

- Reduce turnaround time (submission, marking, feedback, resubmission)
- Reduce the number of resubmissions
- Reduce marking effort

- Provide fast and actionable feedback to students during assignments

# Our GitHub Classroom Setup

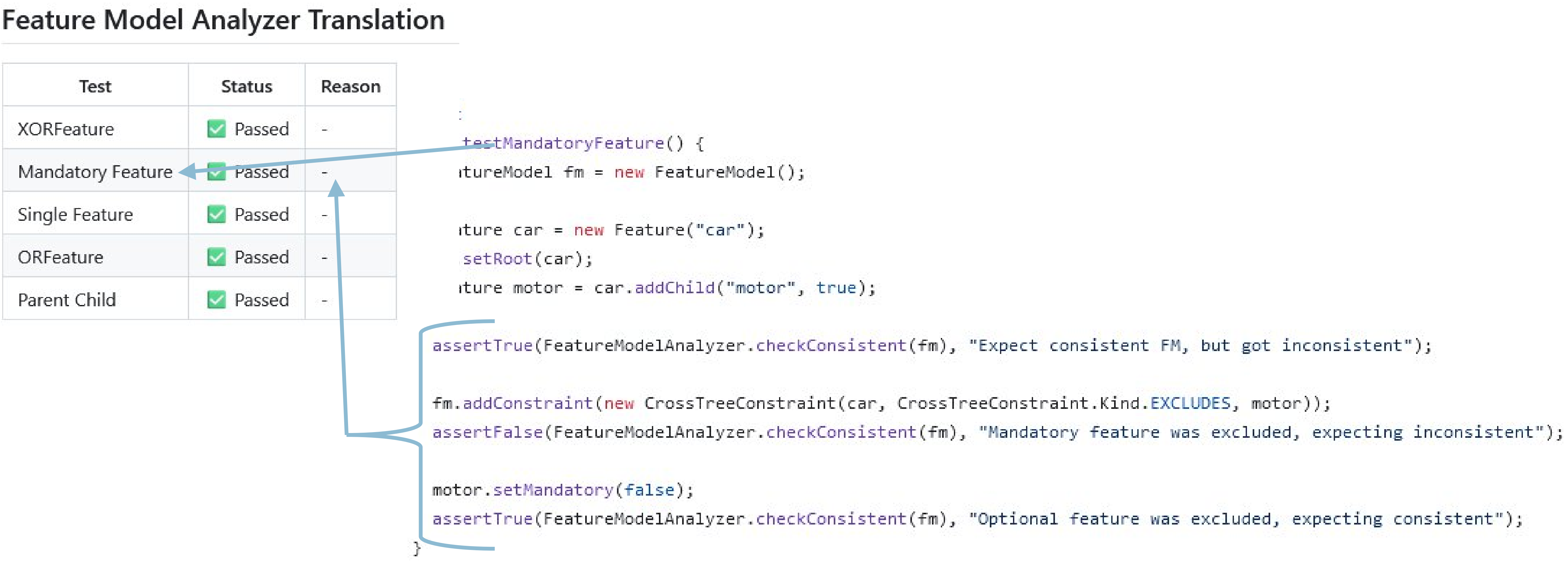
- GitHub Actions set up the execution environment, install necessary software, run maven build scripts to execute JUnit tests,

- Python script generates reports (standard are execution logs on console)

- Students submit a link to their repository on the LMS (decoupled for data protection)

- Alternatives:
  - Repository creation possible with scripts, e.g., in GitLab
  - Other continuous integration systems can easily replace GitHub Actions

# Generated Feedback Reports



## Feature Model Analyzer Translation

| Test | Status | Reason |
|---|---|---|
| XORFeature | ✅ Passed | - |
| Mandatory Feature | ✅ Passed | - |
| Single Feature | ✅ Passed | - |
| ORFeature | ✅ Passed | - |
| Parent Child | ✅ Passed | - |

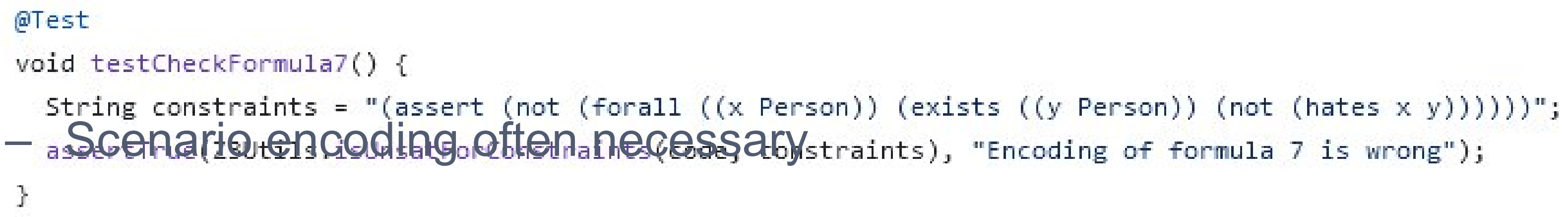
```
testMandatoryFeature() {
tureModel fm = new FeatureModel();

ture car = new Feature("car");
setRoot(car);
ture motor = car.addChild("motor", true);

    assertTrue(FeatureModelAnalyzer.checkConsistent(fm), "Expect consistent FM, but got inconsistent");

    fm.addConstraint(new CrossTreeConstraint(car, CrossTreeConstraint.Kind.EXCLUDES, motor));
    assertFalse(FeatureModelAnalyzer.checkConsistent(fm), "Mandatory feature was excluded, expecting inconsistent");

    motor.setMandatory(false);
    assertTrue(FeatureModelAnalyzer.checkConsistent(fm), "Optional feature was excluded, expecting consistent");
}
```

Bauhaus-Universität Weimar

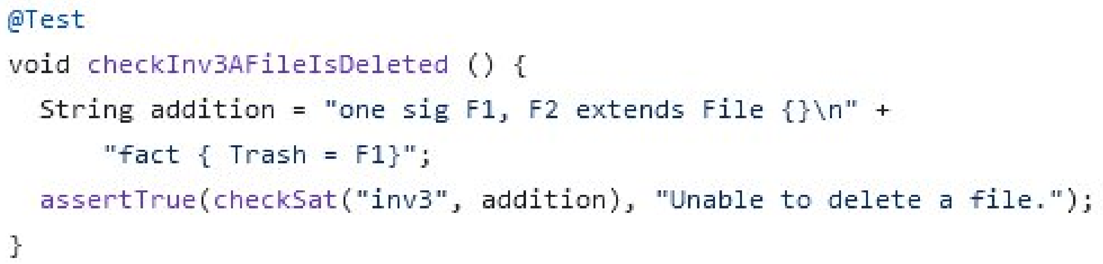
# Observed Challenges

# Migration Challenges

- Free-response questions
  - Difficult to test all aspects, e.g., model is meaningful
- Testability vs. problem encoding
  - Difficulty to write assertions when variable names/types not known
- Solutions in test cases
  - Scenario encoding often necessary



```
@Test
void testCheckFormula7() {
  String constraints = "(assert (not (forall ((x Person)) (exists ((y Person)) (not (hates x y))))))";
  assertTrue(ISUtils.isUnsatForConstraints(code, constraints), "Encoding of formula 7 is wrong");
}
```



```
@Test
void checkInv3AFileIsDeleted () {
  String addition = "one sig F1, F2 extends File {}\n" +
      "fact { Trash = F1}";
  assertTrue(checkSat("inv3", addition), "Unable to delete a file.");
}
```

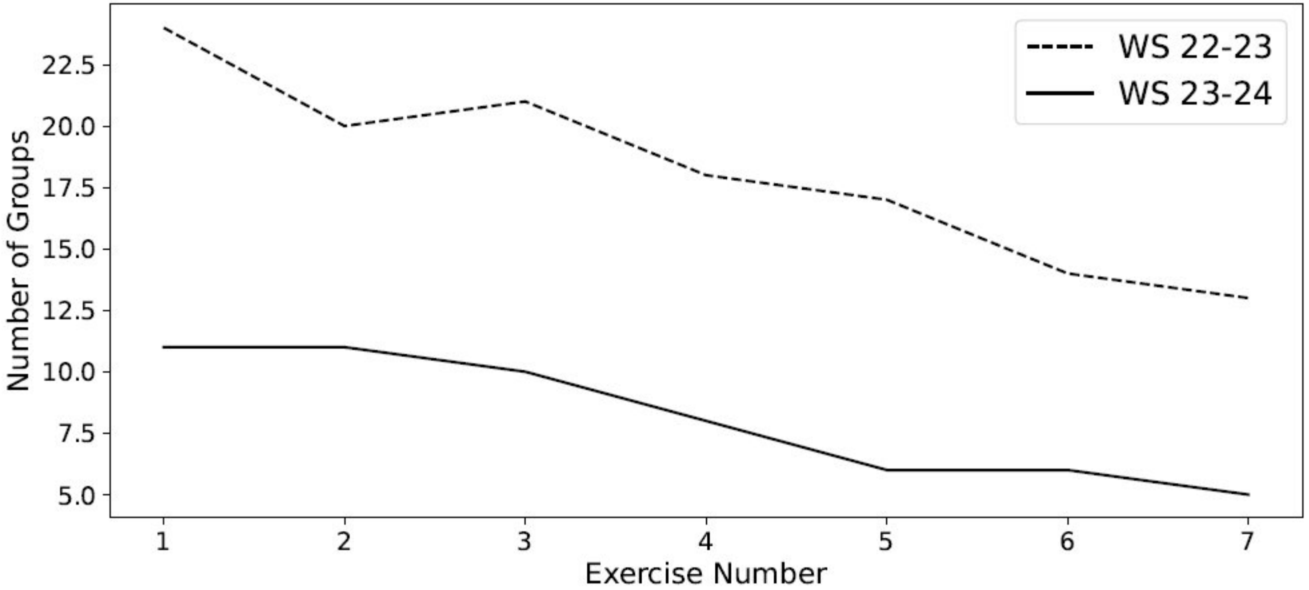
# Migration Challenges

- Submission format
  - PDF not suitable for autograding
- False positives
  - Empty implementation passes tests
- Task dependencies
  - errors in problem encoding may lead to errors in analyses encoding
- Order and number of test cases
  - Achieve early positive feedback for students

# Surveys

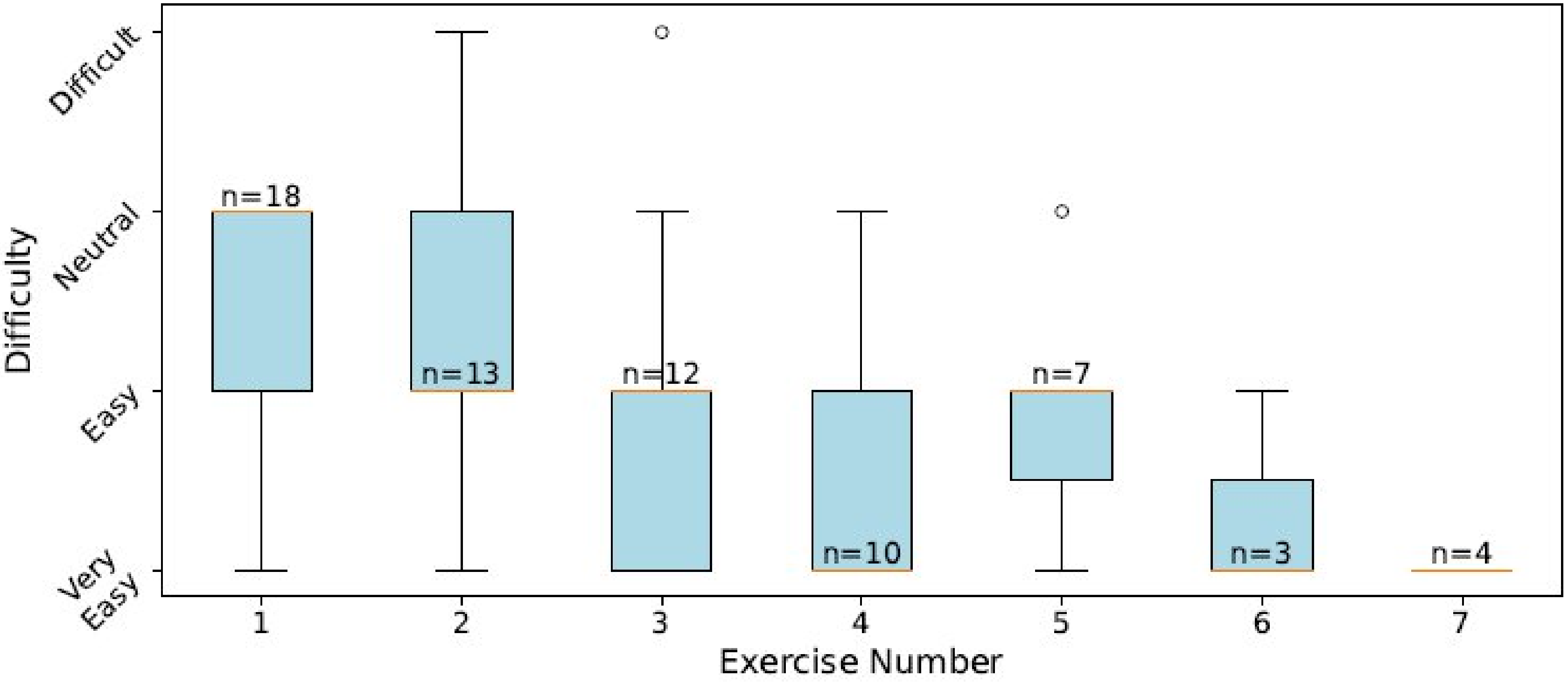
Conducted after each submission on paper

# Submissions per Exercise

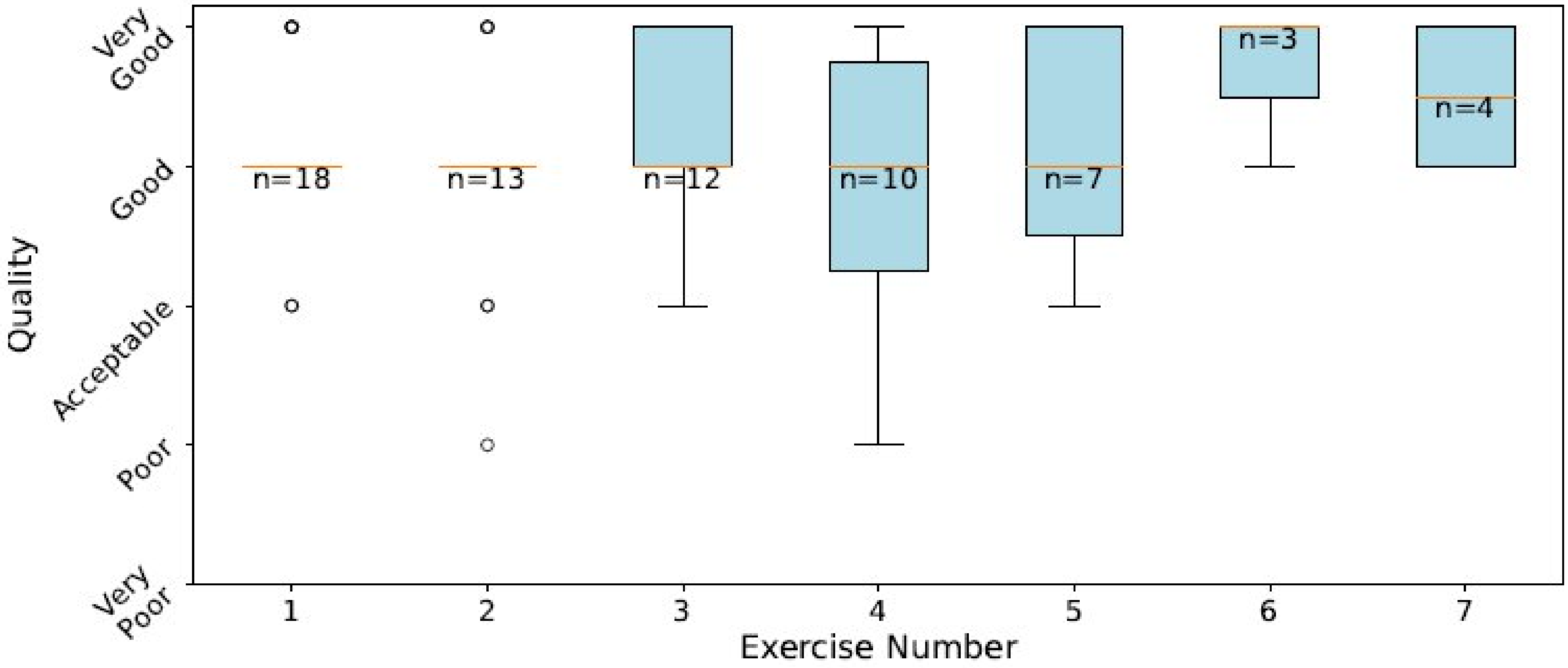


- Overall, com
- Similar to previous iteration without GitHub ClassRoom

# Difficulty of using GitHub for exercise submissions



# Quality of the Automated Feedback



# Free text comments

- The **automated feedback was very important** to evaluate in our cases it was very nice **to know what improvements can be made** in the code further.
- The feedback really helps with the process of completing and understanding the tasks. If a problem is encountered, the feedback helps in identifying the topic of **concept that needs to be revised for completion**.
- Continuous feedback on each statement helped **me compare and understand the assignment better**

- The automated tests **didn’t test for multiple components of some category**, which should not be possible.
- **Provide more information on why the test case has failed** and also the exact errors.
- Maybe include test cases or in this case the **LTLSPECS in the playground template for easy access**.

- There can be **better infrastructure assignments for group submissions** of the assignments.

# Conclusion

- Achievement of goals:
  - **Reduce turnaround time (submission, marking, feedback, resubmission)**
  - **Reduce the number of resubmissions**
  - **Reduce marking effort, at high cost of assignment creation**
  - **Provide fast and actionable feedback to students during assignments**

- Assignment setting (concrete task formulation) and autograding not independent
- Creativity and efforts needed to automate marking
- Manual checks of submissions are still necessary