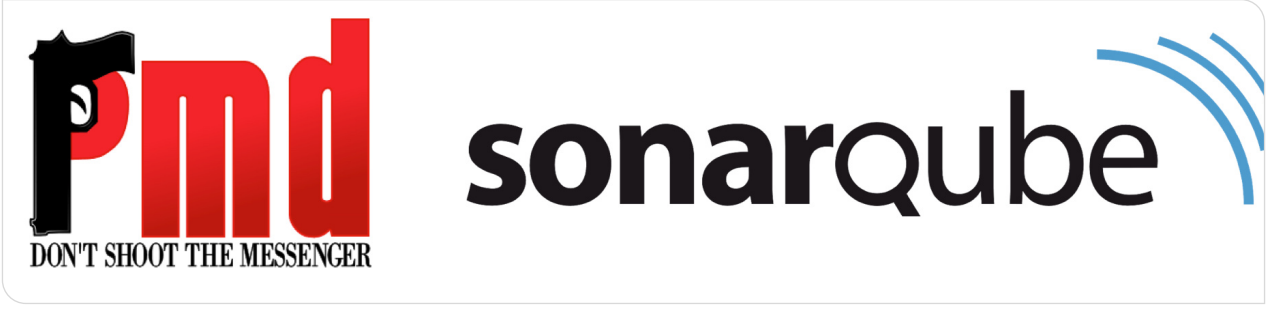
Karlsruher Institut für Technologie

Automatisiertes Testen von Programmiermethodik

Ingenieursmäßige Softwareentwicklung

David Laubenstein | 20. März 2023



Motivation: Aktueller Stand

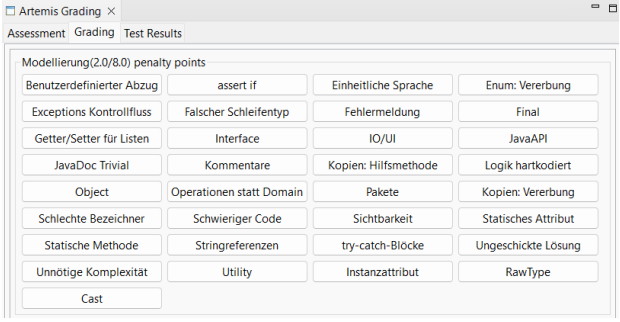
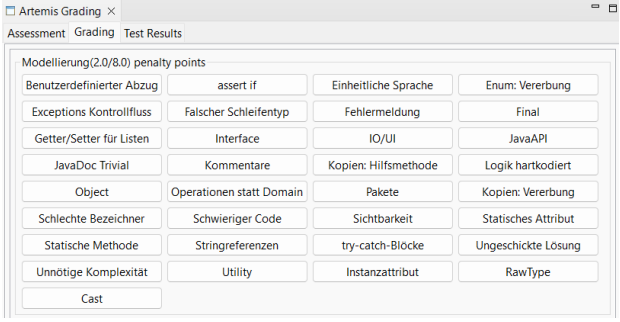


Abbildung: *Grading Edition in Eclipse Artemis* 2023

Motivation: Aktueller Stand



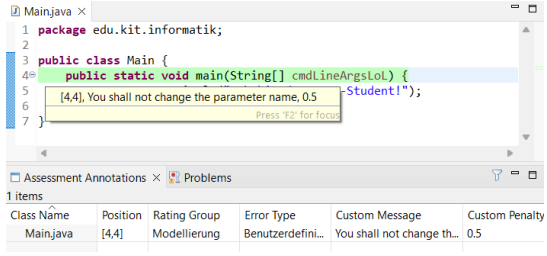


Abbildung: *Grading Edition in Eclipse Artemis* 2023

Motivation: Ziel

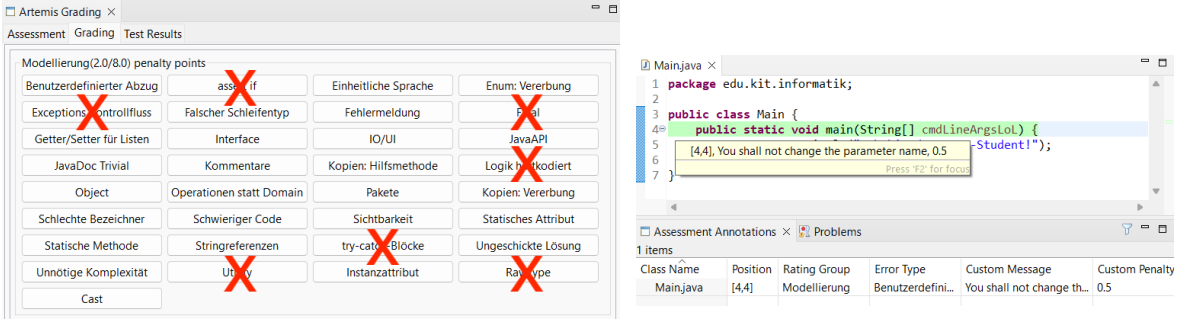


Abbildung: *Grading Edition in Eclipse Artemis* 2023

Ansatz

- Gegeben: 42 Bewertungsrichtlinien für ‘Programmieren‘
- Aufgabe: Automatisierung durch Tests

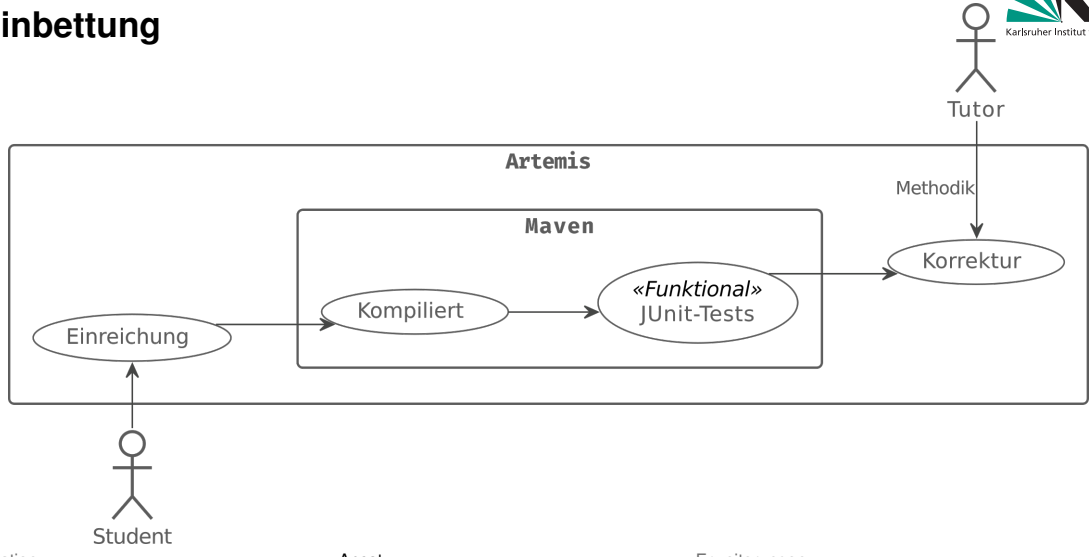
Ansatz

- Gegeben: 42 Bewertungsrichtlinien für ‘Programmieren‘
- Aufgabe: Automatisierung durch Tests
- 2 Fragestellungen:
  - Welche Tools gibt es?
  - Welche Richtlinien können automatisiert werden?
- Tools
  - SonarQube $\rightarrow$ 650 Regeln
  - PMD $\rightarrow$ 325 Regeln
  - etc.

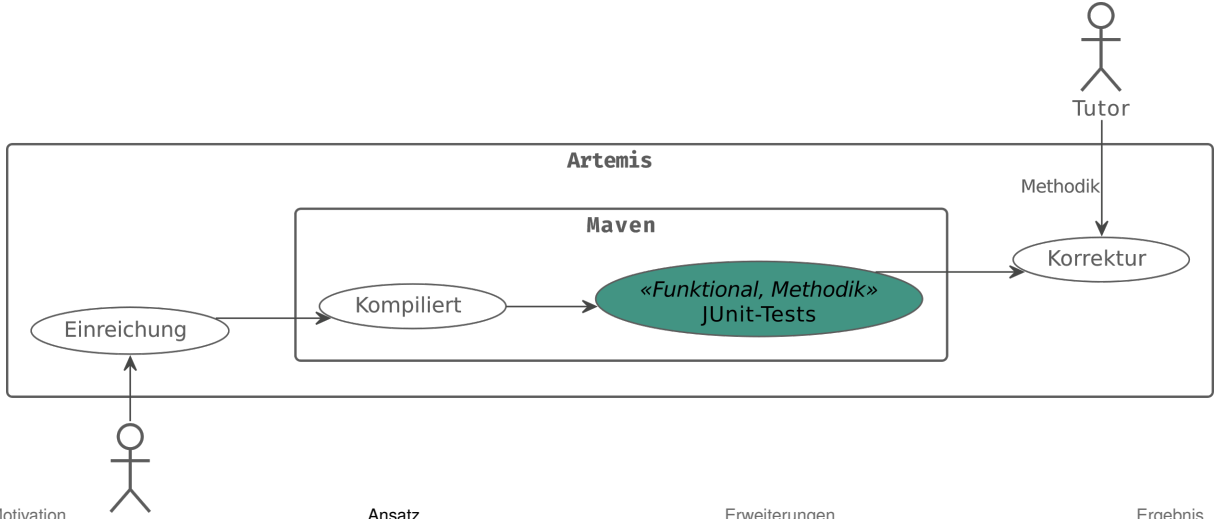
Ansatz

- Gegeben: 42 Bewertungsrichtlinien für ‘Programmieren‘
- Aufgabe: Automatisierung durch Tests
- 2 Fragestellungen:
  - Welche Tools gibt es?
  - Welche Richtlinien können automatisiert werden?
- Tools
  - SonarQube $\rightarrow$ 650 Regeln
  - PMD $\rightarrow$ 325 Regeln
  - etc.
- PMD + SonarQube als Quelle
- Abbilden der existierenden Regeln zu den eigenen Bewertungsrichtlinien von ‘Programmieren‘

Einbettung


Tutor
Artemis
Methodik
Maven
Korrektur
«Funktional»
JUnit-Tests
Kompiliert
Einreichung
Student

Einbettung

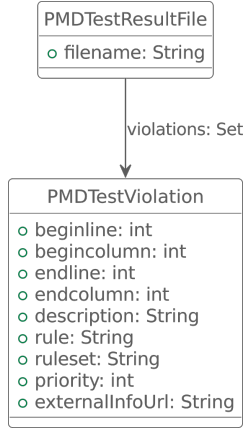

Tutor
Artemis
Methodik
Maven
Korrektur
«Funktional, Methodik»
JUnit-Tests
Kompiliert
Einreichung
Student

Ansatz

```java
public class PMDTests {
    private PMDTestResult issues;

    @BeforeAll
    public void setUpBeforeClass() throws Exception {
        // start PMD analysis
        // map PMD result to issues
    }

    @DisplayName("Test Codebase")
    @ParameterizedTest(...)
    @MethodSource("getTestTypeParameters")
    public void testCodeBase(List<String> relevantIssueNumbers) {
        checkOccurringIssues(findOccurringIssues(relevantIssueNumbers));
    }
}
```



Raw Type (Nr. 3740)

Raw types should not be used

Code Smell | Major | pitfall

Generic types shouldn't be used raw (without type parameters) in variable declarations or return values. Doing so bypasses generic type checking, and defers the catch of unsafe code to runtime.

Noncompliant Code Example

```
List myList; // Noncompliant
Set mySet; // Noncompliant
```

Compliant Solution

```
List<String> myList;
Set<? extends Number> mySet;
```

```
1  public class RawTypeBadExample {
2      List badRawTypeUsage;
3      List<String> goodRawTypeUsage;
4      Set mySet;
5  }
```

Verknüpfen existierender Regeln

- Es matchen von PMD + SonarQube zusammen 7 Regeln
  - *Utility class*
  - *Empty constructor*
  - *Empty block*
  - *Unused element*
  - *Final modifier*
  - *Raw type*
  - *Class instead of interface*
- zudem (+4) Regeln, die Teilmenge der eigentlichen Richtlinie abdecken
  - *Code duplication*
  - *Exceptions for control flow (empty catch block)*
  - *Hardcoded logic*
  - *Wrong loop type*

Erweiterungen

- PMD + SonarQube als Quelle reicht nicht aus
- PMD bietet Möglichkeit, eigene Regeln zu erstellen

Eigene Regeln

```
<?xml version="1.0"?>
<ruleset name="My Custom PMD Ruleset">
    <description>...</description>
    <rule name="CustomJavaPMDRule"
            ...
            message="This is the message, the rule will return"
            class="customPMDRule.CustomJavaPMDRule">
        ...
        <example>
            ...
        </example>
    </rule>
    <rule>
        ...
    </rule>
    ...
</ruleset>
```

Eigene Regeln

```java
public class CustomJavaPMDRule extends AbstractJavaRule {
    @Override
    public Object visit(ASTMethodStatement node, Object data) {
        if (ruleFoundViolation) {
            addViolation(data, node);
        }
        return super.visit(node, data);
    }
}
```

Eigene Regeln

- *Assert vs If*
- *Class of constants*
- *Line separator*
- *Public enum in class or interface*
- *Try catch block size*

Eigene Regeln: Public enum in class or interface

```java
public class PublicEnumInClassOrInterfaceRule extends AbstractJavaRule {
    @Override
    public Object visit(ASTEnumDeclaration node, Object data) {
        if (node.isPublic() && node.getParent() instanceof ASTClassOrInterfaceBodyDeclaration) {
            addViolation(data, node);
        }
        return super.visit(node, data);
    }
}
```

Ergebnis

- Automatisierung von 16 Bewertungsrichtlinien
- Beim Korrigieren:
  - Zeitersparnis
  - Höhere Genauigkeit
  - Fairness
- Future work:
  - Integration in Artemis
    - Studenten haben vor Korrektur auch Einsicht darauf
  - Weitere ’custom rules’ entwickeln!

Eigene Regeln: AssertVsIF

```java
public class AssertStatementFirstInPublicFunctionRule extends AbstractJavaRule {
    @Override
    public Object visit(ASTMethodDeclaration node, Object data) {
            if (node.isPublic()
                && node.getNumChildren() > 2
                && node.getChild(2).getNumChildren() > 0
                && node.getChild(2).getChild(0).getNumChildren() > 0
        ) {
            // Get the first statement of the method
            Node firstChild = node.getChild(2).getChild(0).getChild(0);
            if (firstChild instanceof ASTAssertStatement) {
                addViolation(data, node);
            }
        }
        return super.visit(node, data);
    }
}
```

Eigene Regeln: ClassOfConstants

```java
public class ClassOfConstants extends AbstractJavaRule {
    @Override
    public Object visit(ASTClassOrInterfaceDeclaration node, Object data) {
        int fieldCount = 0;
        int constantCount = 0;
        for (ASTFieldDeclaration field : node.findDescendantsOfType(ASTFieldDeclaration.class)) {
            fieldCount++;
            if (field.isFinal() && field.isStatic()) {
                constantCount++;
            }
        }
        if (node.findDescendantsOfType(ASTMethodDeclaration.class).isEmpty()
            && fieldCount != 0
            && fieldCount == constantCount) {
            addViolation(data, node);
        }
        return super.visit(node, data);
    }
}
```

Eigene Regeln: LineBreakRule

```java
public class LineBreakRule extends AbstractJavaRule {
    @Override
    public Object visit(ASTCompilationUnit node, Object data) {
        for (ASTLiteral literal : node.findDescendantsOfType(ASTLiteral.class)) {
            if (literal.getImage() != null) {
                String[] lines = literal.getImage().split("\\r?\\n|\\r");
                for (String line: lines) {
                    if (line.contains("\\r") || line.contains("\\n")) {
                        addViolation(data, literal);
                    }
                }
            }
        }
        return super.visit(node, data);
    }
}
```

Eigene Regeln: PublicEnumInClassOrInterface

```java
public class PublicEnumInClassOrInterfaceRule extends AbstractJavaRule {
    @Override
    public Object visit(ASTEnumDeclaration node, Object data) {
        if (node.isPublic() && node.getParent() instanceof ASTClassOrInterfaceBodyDeclaration) {
            addViolation(data, node);
        }
        return super.visit(node, data);
    }
}
```

Eigene Regeln: TryCatchBlockSize

```java
public class TryCatchBlockSizeRule extends AbstractJavaRule {
    private static final int MAX_LENGTH = 10;

    @Override
    public Object visit(ASTTryStatement node, Object data) {
        int length = node.getChild(0).getNumChildren();
        if (length > MAX_LENGTH) {
            addViolation(data, node);
        }
        return super.visit(node, data);
    }
}
```

Parameterized Test: getTestTypeParameters

```java
private Stream<Arguments> getTestTypeParameters() {
    return Stream.of(
            Arguments.of("Test SystemDependentLineBreak", List.of("SystemDependentLineBreakNotAllowed")),
            Arguments.of("Test ConcreteClassInsteadOfInterface", List.of("LooseCoupling")),
            Arguments.of("Test AssertInsteadOfIfLoop", List.of("AssertStatementFirstInPublicFunction")),
            Arguments.of("Test UnusedElement",
                List.of(
                    "UnusedPrivateField",
                    "UnusedPrivateMethod",
                    "UnusedLocalVariable",
                    "UnusedFormalParameter"
                )
            ),
            ...
    );
}
```

PMD: BeforeAll

```java
@BeforeAll
public void setUpBeforeClass() throws Exception {
    PMDConfiguration configuration = new PMDConfiguration();
    configuration.addInputPath(PMD_REPORT_INPUT_FILE_PATH);

    configuration.setRuleSets(new ArrayList<>(PMD_RULE_SET_FILE_PATHS.values()));

    configuration.setReportFormat(PMD_REPORT_FILE_FORMAT);
    configuration.setReportFile(PMD_REPORT_FILE_PATH);

    PMD.runPmd(configuration);

    ObjectMapper objectMapper = new ObjectMapper();
    issues = objectMapper.readValue(new File(PMD_REPORT_FILE_PATH), PMDTestResult.class);
}
```

Literatur

Titelbilder Quelle:

Das Titelbild wurde aus den 2 Bildern von (*PMD Java Logo* 2023) und (*SonarQube Logo* 2023) zusammengesetzt.

RawType Quelle:

Der Screenshot stammt von der offiziellen SonarQube Documentation Website (*SonarQube RawType Java Rule* 2023).

Literatur

[1] *Grading Edition in Eclipse Artemis*. https://kit-sdq.github.io/programming-lecture-eclipse-artemis.docs/_images/gradingedition_grading.png. Zugriff am 07.03.2023. 2023.

[2] *PMD Java Logo*. https://pmd.github.io/img/pmd_logo.png. Zugriff am 07.03.2023. 2023.

[3] *SonarQube Logo*. https://assets-eu-01.kc-usercontent.com/844a25be-b07a-010a-7ca0-4069669f847d/12e3974b-220d-4cde-8f17-2ff9fa9d9c27/SonarQube_Logo.svg. Zugriff am 07.03.2023. 2023.

[4] *SonarQube RawType Java Rule*. https://rules.sonarsource.com/java/RSPEC-3740. Zugriff am 15.03.2023. 2023.