Personal Prof

Automatic Code Review For Java Assignments

Markus Klinik, Pieter Koopman, Rick van der Wal

November 23, 2021

Institute for Computing
and Information Sciences
Radboud University

Paper!

Markus Klinik, Pieter Koopman, and Rick van der Wal. 2021. Personal Prof: Automatic Code Review for Java Assignments. In *The 10th Computer Science Education Research Conference (CSERC 21), November 2223, 2021, Virtual Event, Netherlands.* ACM, New York, NY, USA, 8 pages.

https://doi.org/10.1145/3507923.3507930

Assessing Assignments of Large Courses

The Course Object-Oriented Programming

400 students, CS and AI

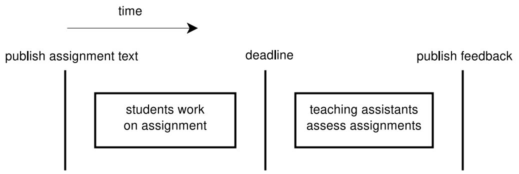
2 teachers

17 teaching assistants (TAs)

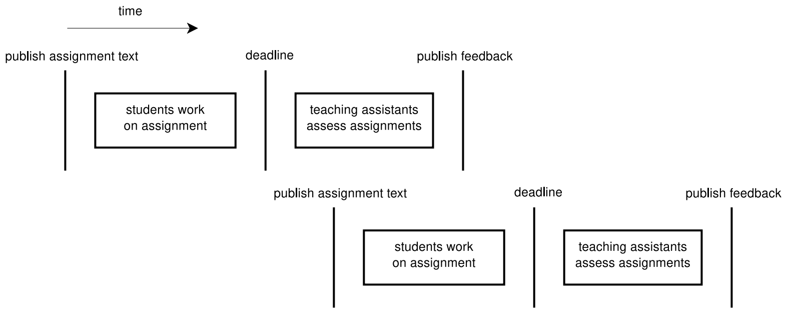
1 assignment coordinator

14 weekly assignments

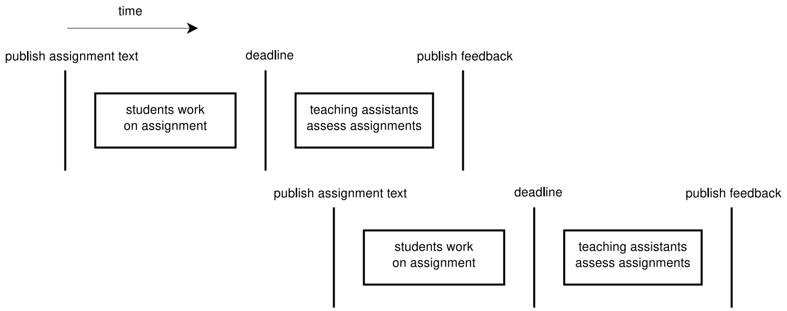
The Course Object-Oriented Programming


time
publish assignment text
deadline
publish feedback
students work on assignment
teaching assistants assess assignments

The Course Object-Oriented Programming


time
publish assignment text
deadline
publish feedback
students work on assignment
teaching assistants assess assignments
publish assignment text
deadline
publish feedback
students work on assignment
teaching assistants assess assignments

The Course Object-Oriented Programming


time
publish assignment text
deadline
publish feedback
students work on assignment
teaching assistants assess assignments
publish assignment text
deadline
publish feedback
students work on assignment
teaching assistants assess assignments

Problem: Students get feedback too late

The are busy with the next assignment

Side Note

The Point of Having Assignments

Encourage students to do some programming

Not gatekeeping the final exam

Assignment grades only for student’s self-evaluation

How To Give Timely Feedback?

Automate It!

- Personal Prof
- Assignment-specific rules
- Based on the source code's object model
  - AST and symbol tables
- Instant Feedback

Example Rule

Excerpt of our grading manual

> There should be an abstract class **Question**. This class should have one attribute *score* and a setter for the score that checks that the argument is valid.

Example Rule

Excerpt of our grading manual

> There should be an abstract class **Question**. This class should have one attribute *score* and a setter for the score that checks that the argument is valid.
>
> There should be three classes **OpenQuestion**, **MultipleChoiceQuestion**, and **ThisThatQuestion**.

Example Rule

Excerpt of our grading manual

> There should be an abstract class **Question**. This class should have one attribute *score* and a setter for the score that checks that the argument is valid.
>
> There should be three classes **OpenQuestion**, **MultipleChoiceQuestion**, and **ThisThatQuestion**.
>
> **OpenQuestion** and **MultipleChoiceQuestion** should extend **Question**.

Example Rule

Excerpt of our grading manual

> There should be an abstract class **Question**. This class should have one attribute *score* and a setter for the score that checks that the argument is valid.
>
> There should be three classes **OpenQuestion**, **MultipleChoiceQuestion**, and **ThisThatQuestion**.
>
> **OpenQuestion** and **MultipleChoiceQuestion** should extend **Question**.
>
> **ThisThatQuestion** should extend **MultipleChoiceQuestion**.

Example Rule

Excerpt of our grading manual

> There should be an abstract class **Question**. This class should have one attribute *score* and a setter for the score that checks that the argument is valid.
>
> There should be three classes **OpenQuestion**, **MultipleChoiceQuestion**, and **ThisThatQuestion**.
>
> **OpenQuestion** and **MultipleChoiceQuestion** should extend **Question**.
>
> **ThisThatQuestion** should extend **MultipleChoiceQuestion.**
>
> Every question class is different, and therefore should implement the three functions *toString*, *isCorrect* and *correctAnswer*.

Personal Prof Goals And Non-Goals

Goals

Give code-review-like feedback

Non-Goals

Give numerical grades

Personal Prof Goals And Non-Goals

Goals

Give code-review-like feedback

Software architecture

Non-Goals

Give numerical grades

Correctness

Personal Prof Goals And Non-Goals

Goals

- Give code-review-like feedback
- Software architecture
- Fast feedback cycle

Non-Goals

- Give numerical grades
- Correctness
- Replace human feedback

Personal Prof Goals And Non-Goals

Goals

- Give code-review-like feedback
- Software architecture
- Fast feedback cycle
- Focus on uncontroversial faults

Non-Goals

- Give numerical grades
- Correctness
- Replace human feedback
- Cover the whole grading manual

Implementation

Rascal Metaprogramming Language

`https://www.rascal-mpl.org/`

Developed at CWI Amsterdam

Programming language for writing *compilers*

Includes everything a compiler writer needs

- Parsers: grammars also define syntax tree data type
- Language primitives: deep pattern-matching
- Data types: source locations
- Standard library: rich symbol tables

What Can Rascal Do For Personal Prof?

How can we implement the rule

OpenQuestion should extend **Question**

What Can Rascal Do For Personal Prof?

How can we implement the rule

OpenQuestion should extend **Question**

```
int main(list[str] args) {
  loc projectDir = |file:///| + args[0];
  M3 m = createM3FromDirectory(projectDir,
      javaVersion="11");
}
```

The M3 Meta Model

Generic

Java-specific

The M3 Meta Model

Generic

messages `set[Message]`

Java-specific

The M3 Meta Model

Generic

messages `set[Message]`

names `rel[str simpleName, loc qualifiedName]`

Java-specific

The M3 Meta Model

Generic

- **messages** `set[Message]`
- **names** `rel[str simpleName, loc qualifiedName]`
- **containment** `rel[loc from, loc to]`

Java-specific

The M3 Meta Model

Generic

- **messages** `set[Message]`
- **names** `rel[str simpleName, loc qualifiedName]`
- **containment** `rel[loc from, loc to]`
- **modifiers** `rel[loc definition, Modifier modifier]`

Java-specific

The M3 Meta Model

Generic

- **messages** `set[Message]`
- **names** `rel[str simpleName, loc qualifiedName]`
- **containment** `rel[loc from, loc to]`
- **modifiers** `rel[loc definition, Modifier modifier]`
- **types** `rel[loc name, TypeSymbol typ]`

Java-specific

The M3 Meta Model

Generic

messages `set[Message]`

names `rel[str simpleName, loc qualifiedName]`

containment `rel[loc from, loc to]`

modifiers `rel[loc definition, Modifier modifier]`

types `rel[loc name, TypeSymbol typ]`

Java-specific

classes `set[loc]`

The M3 Meta Model

Generic

messages `set[Message]`

names `rel[str simpleName, loc qualifiedName]`

containment `rel[loc from, loc to]`

modifiers `rel[loc definition, Modifier modifier]`

types `rel[loc name, TypeSymbol typ]`

Java-specific

classes `set[loc]`

extends `rel[loc from, loc to]`

The M3 Meta Model

Generic

- **messages** `set[Message]`
- **names** `rel[str simpleName, loc qualifiedName]`
- **containment** `rel[loc from, loc to]`
- **modifiers** `rel[loc definition, Modifier modifier]`
- **types** `rel[loc name, TypeSymbol typ]`

Java-specific

- **classes** `set[loc]`
- **extends** `rel[loc from, loc to]`
- **fieldAccess** `rel[loc from, loc to]`

The M3 Meta Model

Generic

- **messages** `set[Message]`
- **names** `rel[str simpleName, loc qualifiedName]`
- **containment** `rel[loc from, loc to]`
- **modifiers** `rel[loc definition, Modifier modifier]`
- **types** `rel[loc name, TypeSymbol typ]`

Java-specific

- **classes** `set[loc]`
- **extends** `rel[loc from, loc to]`
- **fieldAccess** `rel[loc from, loc to]`
- ...

Check That A Class Exists

OpenQuestion should extend **Question**

Check That A Class Exists

OpenQuestion should extend **Question**

```
str getName(M3 model, loc id) {
  set[str] candidates = invert(model.names)[id];
  // not shown: error handling
  return getOneFrom(candidates);
}
```

Check That A Class Exists

OpenQuestion should extend **Question**

```
loc findClass(M3 model, str className) {
  set[loc] candidates =
    { cls | cls <- classes(model)
    , /^<className>$/ := getName(model, cls)
    };
  if(size(candidates) > 1) {
    throw error("Need at most one <className>", |file:///|);
  }
  if(size(candidates) < 1) {
    throw error("Need at least one <className>", |file:///|);
  }
  return getOneFrom(candidates);
}
```

Check That A Class Extends Another Class

OpenQuestion should extend **Question**

```
set[Message]
extendsClass(M3 model, str classNameA, str classNameB) {
  try {
    loc classA = findClass(model, classNameA);
    loc classB = findClass(model, classNameB);
    if( ! (classB in model.extends[classA]) )
    {
      return { error("<classNameA> should extend <classNameB>",
        classA) };
    }
  }
  catch e:error(_,_): {
    return { e };
  }
  return {};
}
```

The Class Hierarchy For The Quiz Assignment

```
set[Message] aQuiz_question_types(M3 model) =
  extendsClass(model, "OpenQuestion", "Question") +
  extendsClass(model, "MultipleChoiceQuestion", "Question") +
  extendsClass(model, "ThisThatQuestion", "MultipleChoiceQuestion");
```

Some Other Rules Personal Prof Can Check

All I/O should only happen in the view class.

Some Other Rules Personal Prof Can Check

All I/O should only happen in the view class.

StringBuilder should be used to construct the guessed word.

Some Other Rules Personal Prof Can Check

All I/O should only happen in the view class.

StringBuilder should be used to construct the guessed word.

Interface *Geometric* should extend *Comparable*

Some Other Rules Personal Prof Can Check

All I/O should only happen in the view class.

StringBuilder should be used to construct the guessed word.

Interface *Geometric* should extend *Comparable*

Circle and *Rectangle* should implement *Comparable.compareTo*

Some Other Rules Personal Prof Can Check

All I/O should only happen in the view class.

StringBuilder should be used to construct the guessed word.

Interface *Geometric* should extend *Comparable*

Circle and *Rectangle* should implement *Comparable.compareTo*

List traversal should use iterators. No indices or foreach-loops should be used.

Some Other Rules Personal Prof Can Check

All I/O should only happen in the view class.

StringBuilder should be used to construct the guessed word.

Interface *Geometric* should extend *Comparable*

Circle and *Rectangle* should implement *Comparable.compareTo*

List traversal should use iterators. No indices or foreach-loops should be used.

No loops should be used at all

Some Other Rules Personal Prof Can Check

All I/O should only happen in the view class.

StringBuilder should be used to construct the guessed word.

Interface *Geometric* should extend *Comparable*

Circle and *Rectangle* should implement *Comparable.compareTo*

List traversal should use iterators. No indices or foreach-loops should be used.

No loops should be used at all

Instead of the recursive function call, a new thread should be started

Some Other Rules Personal Prof Can Check

All I/O should only happen in the view class.

StringBuilder should be used to construct the guessed word.

Interface *Geometric* should extend *Comparable*

Circle and *Rectangle* should implement *Comparable.compareTo*

List traversal should use iterators. No indices or foreach-loops should be used.

No loops should be used at all

Instead of the recursive function call, a new thread should be started

There should be exactly one Lock and two Conditions in the whole project

Some Other Rules Personal Prof Can Check

All I/O should only happen in the view class.

StringBuilder should be used to construct the guessed word.

Interface *Geometric* should extend *Comparable*

Circle and *Rectangle* should implement *Comparable.compareTo*

List traversal should use iterators. No indices or foreach-loops should be used.

No loops should be used at all

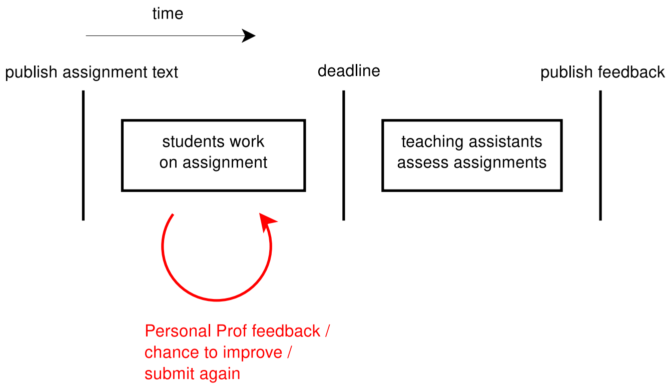
Instead of the recursive function call, a new thread should be started

There should be exactly one Lock and two Conditions in the whole project

Only *Station* may use synchronization

Results

Fast Feedback


time
publish assignment text
deadline
publish feedback
students work on assignment
teaching assistants assess assignments
Personal Prof feedback / chance to improve / submit again

Limitations

Human review is still needed for checking some learning goals

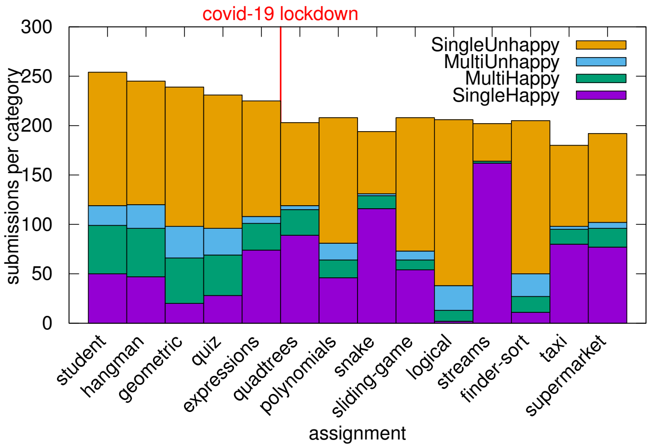
Limitations

- Human review is still needed for checking some learning goals
- False positives are unavoidable
  - Some work required to fine-tune the rules
  - When in doubt, relax the rules

Limitations

- Human review is still needed for checking some learning goals
- False positives are unavoidable
  - Some work required to fine-tune the rules
  - When in doubt, relax the rules
- Adapt assignments to permit automated checking
  - More specific task descriptions
  - More precise requirements
  - Stipulate class names

Impact: Resubmission Behavior



SingleUnhappy: students who should have resubmitted but did not.

Institute for Computing and Information Sciences
Radboud University

Impact

No measurable impact on grades, but corona . . .

Biggest observable change:

No more student complaints about late feedback

Project Website

Source Code

Installation guide

Integration with our online learning environment Brightspace

`https://gitlab.science.ru.nl/pieter/personal-prof-public-repository`