Human vs. Automated Coding Style Grading in Computing Education

James Perretta, Westley Weimer, and Andrew DeOrio,
University of Michigan

ASEE Annual Conference and Exposition, June 2019

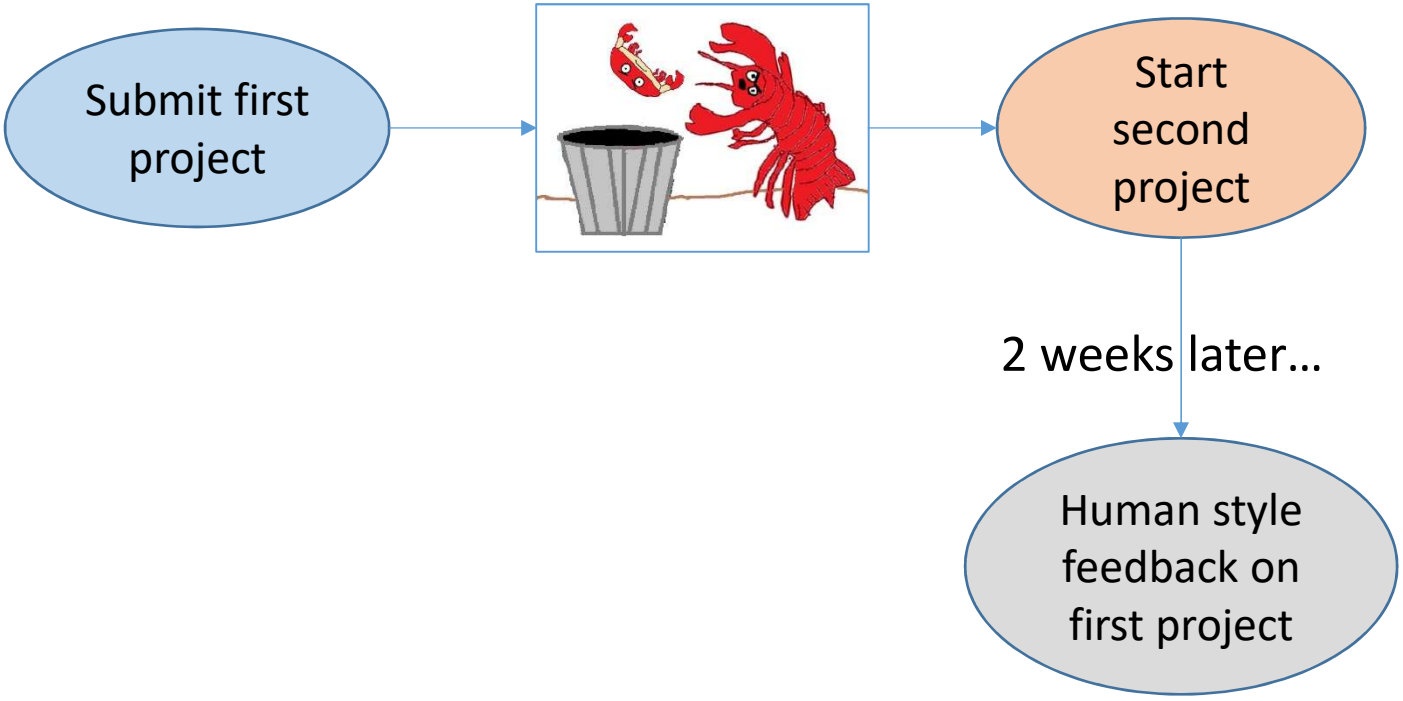
Motivation

- Code review and good coding style are important for writing maintainable software (McIntosh 2014)
  - Style grading can be worth up to 5% of overall CS1/CS2 course grade
- Grading style is time consuming, difficult to scale, usually manual process
- Can automated static analysis tools help?

Code Style

- For CS1/CS2 students, build good habits and discourage common bad ones
  - e.g. Indent source code, use good variable names, don't copy-paste
- Modern tools exist to enforce specific coding standards
  - e.g. pycodestyle (Python), checkstyle (Java), OCLint (C++)

Problem with Human Style Grading



Style Grading: Desirable Qualities

Speed

- Students benefit from frequent, actionable feedback (Edwards 2003)

Accuracy

- Free from false positives, students should get the right grade

Clarity

- Students can learn from feedback and make changes

Research Questions

1. Do human graders provide style grading scores consistent with each other?
2. Are human coding style evaluation scores consistent with static analysis tools?
3. Which style grading criteria are more effectively evaluated with existing static analysis tools and which are more effectively evaluated by human graders?

Goal: Identify code inspections from off-the-shelf static analysis tools that provide high-quality style-grading feedback.

Methods: Course Overview

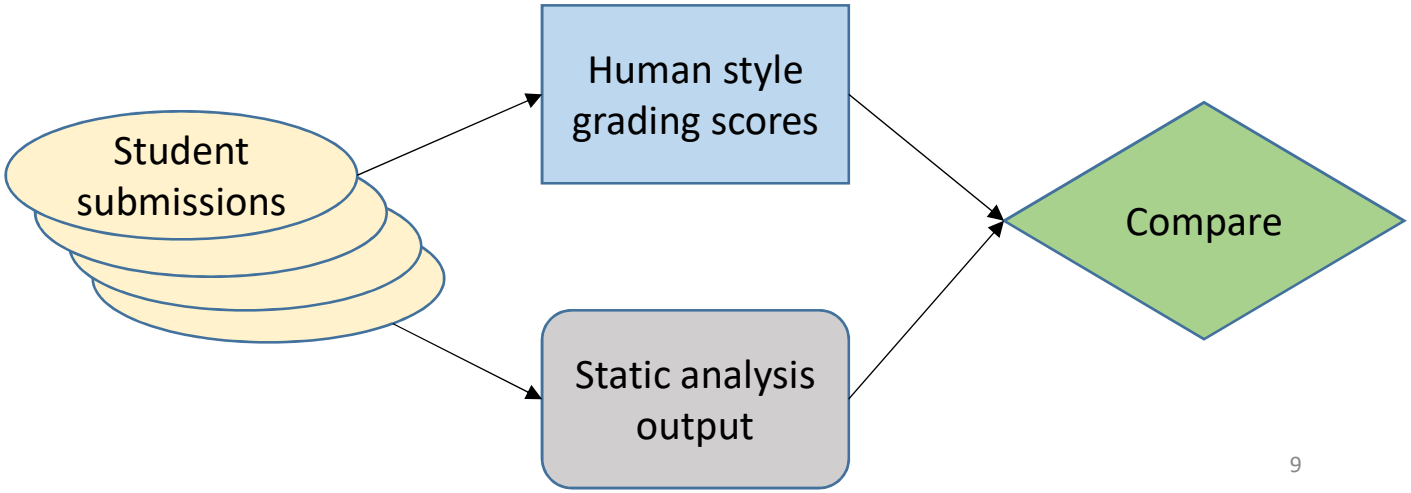
- 943 students in one semester of a CS2 course at the University of Michigan
- 3 hrs lecture and 2 hrs lab per week
- 5 projects, students write C++ code according to specification
- Students could work alone or with a partner

Methods: Programming Project

- Examined one programming project with two components:
  - Implement several abstract data types (ADTs)
  - Implement an open-ended command-line program using the ADTs
- Instructor solution 595 lines of code
- Average student solution 857 lines of code
- Correctness feedback from automated grading system
- Style grading evaluated manually after deadline

Methods: Data Collected

- **621** distinct assignment final submissions
- Style grading scores assigned by human graders
- Static analysis post hoc



Methods: Human Style Grading

- Hired student graders, 42 submissions each over 2 weeks
- Style rubric, 3-value scale
  - Full Credit, Partial Credit, No Credit
- Written instructions on how to apply criteria

Methods: Style Grading Rubric

- Criteria represent common guidelines in intro programming courses, e.g.
  - *Helper functions used where appropriate*
  - *Lines are not too long*
  - *Functions and variables have descriptive names*
  - *Effective, consistent, and readable line indentation is used*
  - *Code is not too deeply nested in loops and conditionals*

Methods: Static Analysis Inspections

- Tools must:
  - Support C++
  - Have configurable thresholds
  - Have easy-to-parse output
- We selected tools that detect:
  - Lines too long
  - Blocks too deeply nested
  - Functions too long
  - Duplicated code

Results: Human Grader Consistency

	Human Grader #														
	1*	2*	3*	4†	5†	6*	7*	8*	9*	10*	11*	12*	13†	14†	15‡
Mean	20	20	20	20	19	20	20	21	20	20	20	21	20	20	17
Stdev	2.1	1.7	1.9	1.8	2.5	2.0	1.7	1.8	2.7	2.1	2.7	1.5	1.8	1.9	4.7
Median	21	20	20	20	20	21	21	21	22	21	21	21	20	19	18

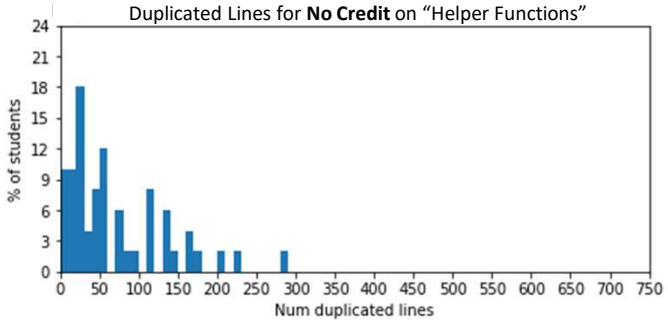
- Scores out of 22 points possible
- A third of our human graders did not assign style scores consistently when compared to the other two thirds

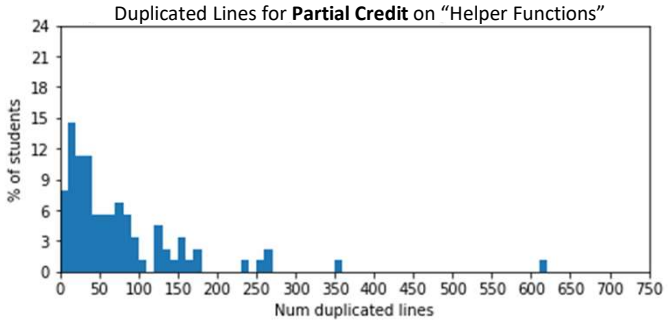
Results: Static Analysis vs. Human Style Grading Scores

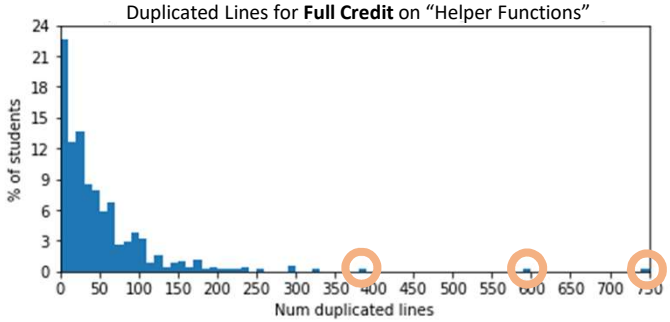
Style Criterion	Static Analysis Inspection	Pearson r
Line Length	OCLint LongLine	**-0.22**
Nesting	OCLint DeepNestedBlock	**-0.21**
Helper Functions	OCLint HighNcssMethod	**-0.07**
Helper Functions	PMD Copy/Paste Detector	**-0.12**

- Human style scores are weakly correlated, if at all, with the number of static analysis warnings

Results: Distributions of Static Analysis Warnings







Scores	U	p-value
No Credit vs **Partial Credit**	2204	0.46
Partial Credit vs **Full Credit**	17084	0.0005

- Significant difference between Partial Credit and Full Credit, but not b/w No Credit and Partial Credit
- Many students were either **unfairly penalized** or **should have been penalized**
  - 10% of students who received **No Credit** had **no reported duplicated code**
  - 13% of students with **Full Credit** had **at least 100 duplicated lines**

Static Analysis Limitations

- Some style criteria are too specific to be covered by general-purpose tools
- Others are too complicated, e.g. analyzing variable names (length isn't enough)
  - `e` and `i` generally accepted for caught exception and loop counter

```
void set_players(string arg1, string arg2,
                 string arg3, string arg4,
                 string arg5, string arg6,
                 string arg7, string arg8) {…}
```

Limitations of the Study

- Hired graders were undergraduate students with limited training
  - Historical data, so we relied on training provided by the course
- Student submissions contained identifiers, potential for grader bias
- No double-marking, limits conclusions about inter-rater reliability

Conclusions and Recommendations

- Human graders do not make a consistent distinction between students who made some mistakes and those who made many mistakes

- Up to 10% of submissions were either unfairly penalized by humans or should have been penalized but were not

- Static analysis tools perform faster, more consistently, and more accurately than humans when a style criterion can be evaluated with simple rules

Conclusions and Recommendations

- Prefer static analysis for style criteria that can be evaluated with simple abstract syntax tree rules
- Prefer a binary scale unless hired graders can be thoroughly trained
- With static analysis evaluating simple aspects of code style, a few well-trained graders can focus on more complex aspects
- Students should be able to address static analysis feedback and resubmit
  - Better yet, let them run the tools on their own!