Measuring the Software Development Process to Enable Formative Assessments

Ayaan M. Kazerouni <ayaan@vt.edu>

Computer Science, Virginia Tech

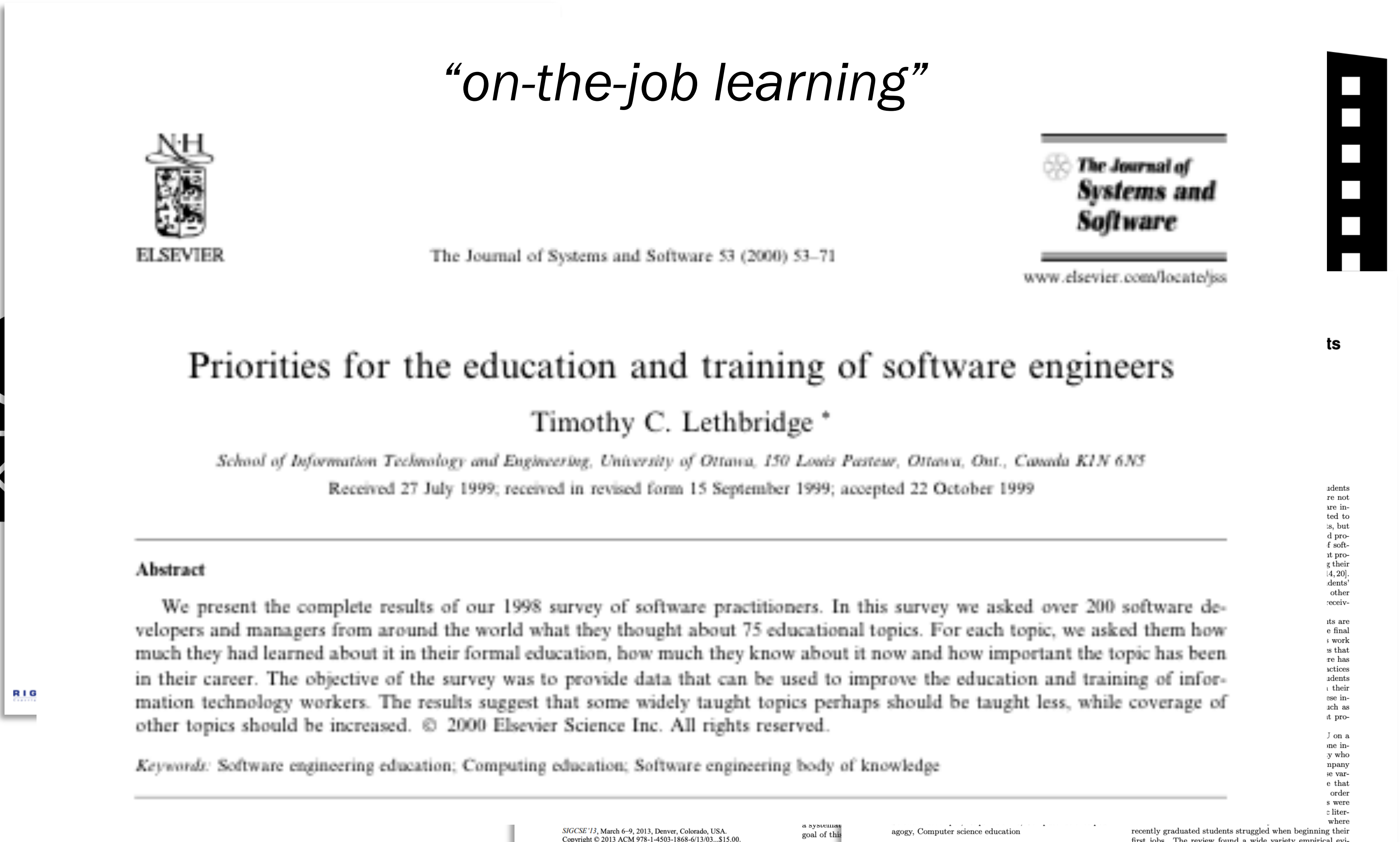
Friday, 18 December 2020

Advisory Committee

Cliff Shaffer, Steve Edwards, Francisco Servant, Dennis Kafura, Jaime Spacco

Graduating CS students tend to face difficulties upon entering the work-force

"on-the-job learning"


ELSEVIER

The Journal of Systems and Software 53 (2000) 53–71

The Journal of Systems and Software

www.elsevier.com/locate/jss

Priorities for the education and training of software engineers

Timothy C. Lethbridge *

School of Information Technology and Engineering, University of Ottawa, 150 Louis Pasteur, Ottawa, Ont., Canada K1N 6N5

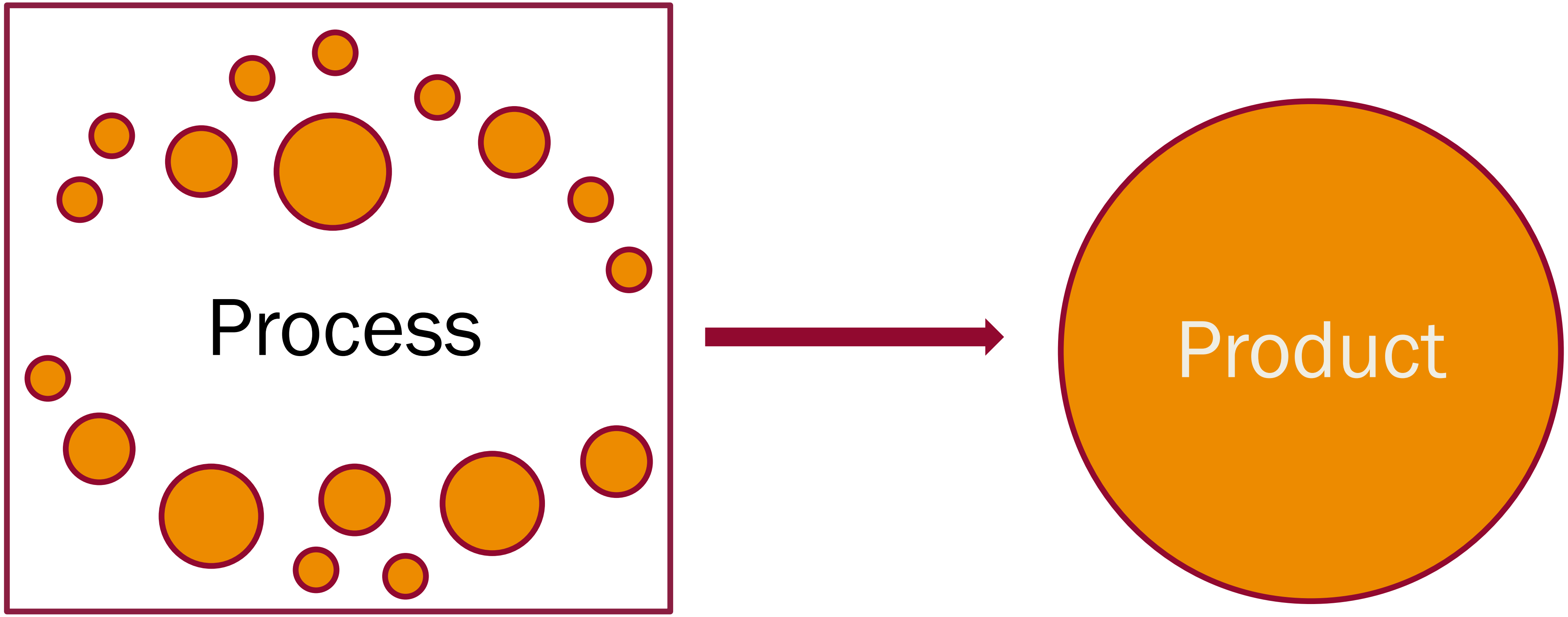
Received 27 July 1999; received in revised form 15 September 1999; accepted 22 October 1999

Abstract

We present the complete results of our 1998 survey of software practitioners. In this survey we asked over 200 software developers and managers from around the world what they thought about 75 educational topics. For each topic, we asked them how much they had learned about it in their formal education, how much they know about it now and how important the topic has been in their career. The objective of the survey was to provide data that can be used to improve the education and training of information technology workers. The results suggest that some widely taught topics perhaps should be taught less, while coverage of other topics should be increased.

Keywords: Software engineering education; Computing education; Software engineering body of knowledge

Focus is on the *engineered product*, and ignores the *engineering process*



Time management
Software testing
Test quality

Correctness
Code style
Code coverage
e.g., Web-CAT, CI/CD

Overarching hypothesis

Formative feedback about software development will help student developers achieve better project outcomes.

Thesis addressed in this talk

Measurable differences in students' software development processes can explain differences in their project outcomes.

Outline

Motivation

Time Management

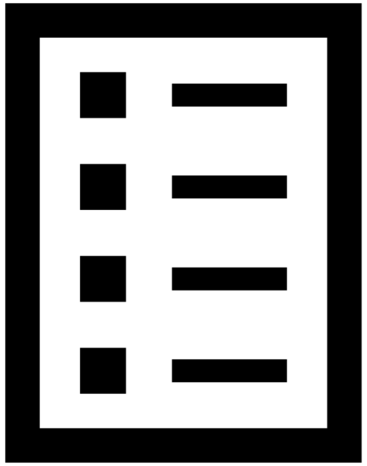
Infrastructure

Software Testing

- Test Process
- Test Quality

Context

CS 2
Software Design & Data Structures



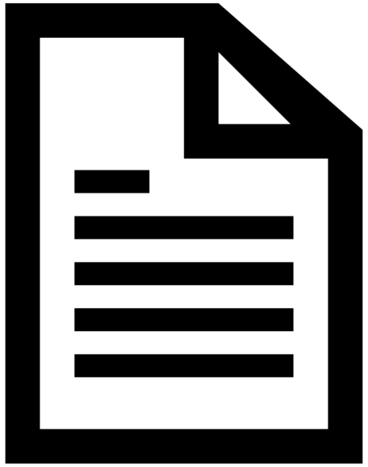
Simpler

Smaller

Scaffolded

~1–2 weeks

CS 3
Data Structures & Algorithms



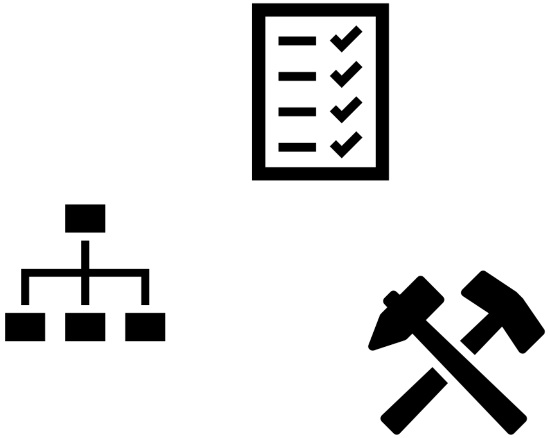
Relatively complex

Larger

Un-scaffolded

~3–4 weeks

"on-the-job learning"



Failure rates

Fall 2016: 22%

Fall 2018: 28%

Better Feedback on Software Development

Programming effort

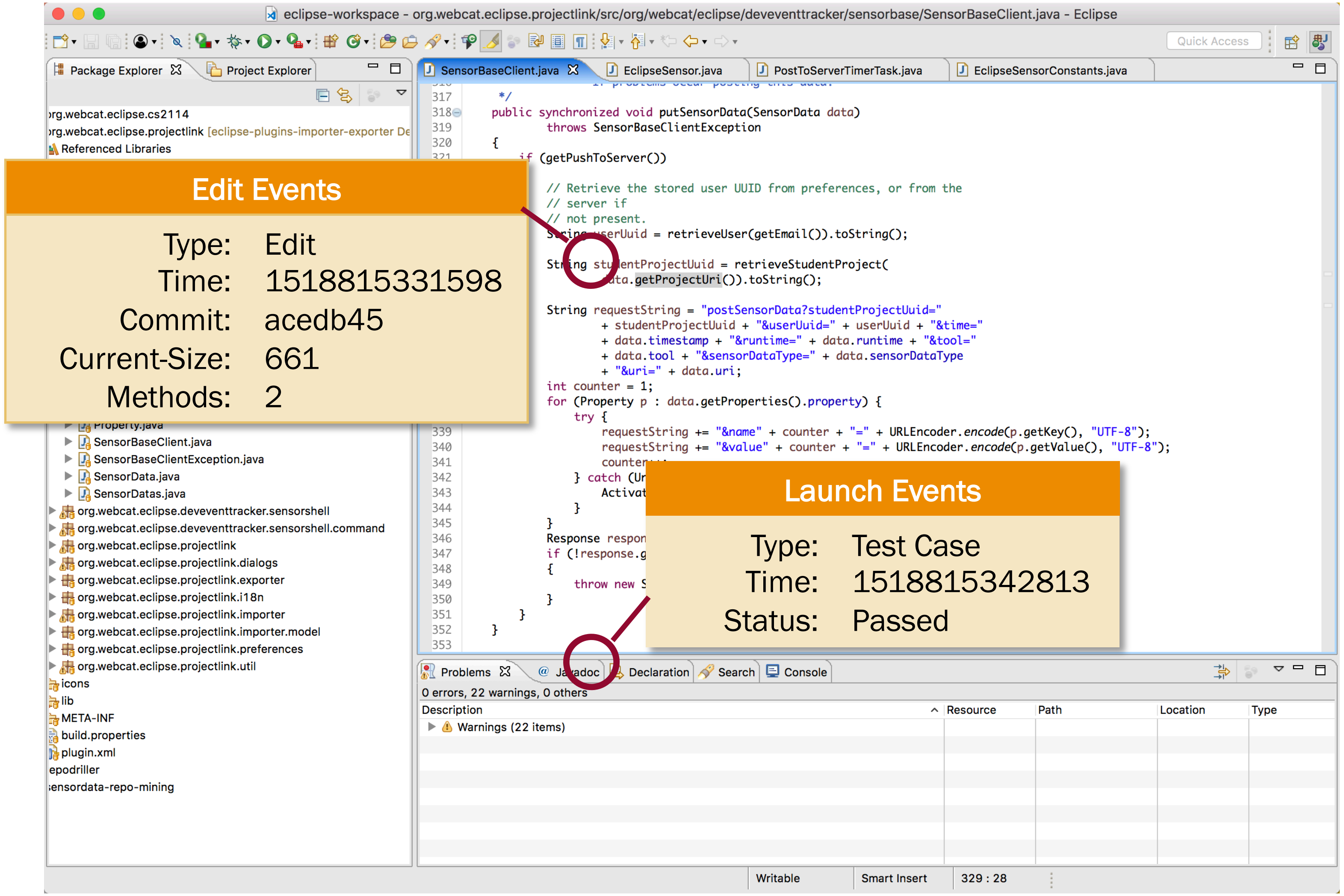
Time →

Feedback

Correctness: 100%

Code coverage: 89%

How do we observe a ~30-hour development process carried out at home?



Events emitted for IDE actions

- Edit
- Program execution
- Test execution
- Debugger step

* one of 4 developers

Time Management

Better Feedback on Software Development

Programming effort

Feedback

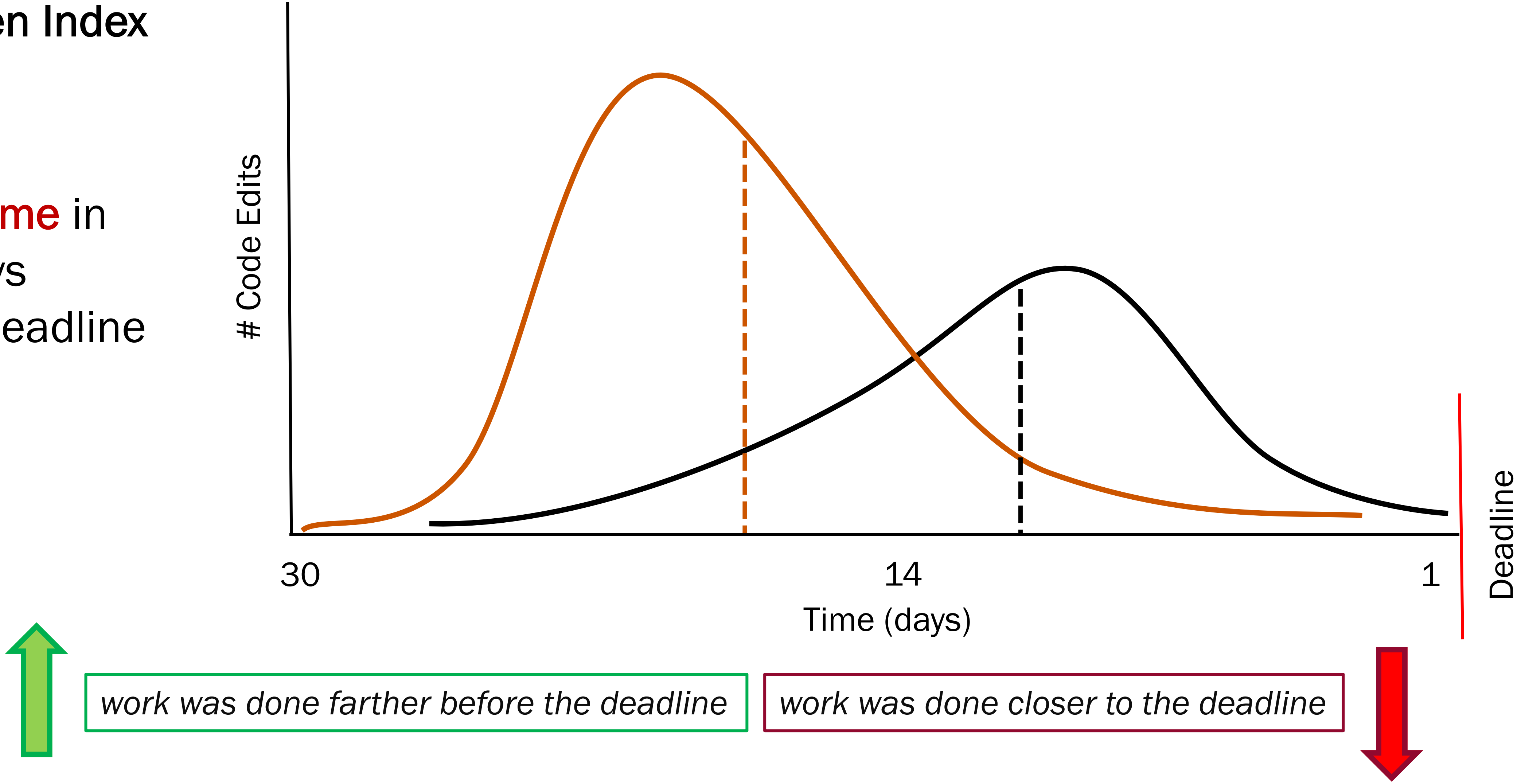
Correctness:	100%
Code coverage:	89%

Procrastination

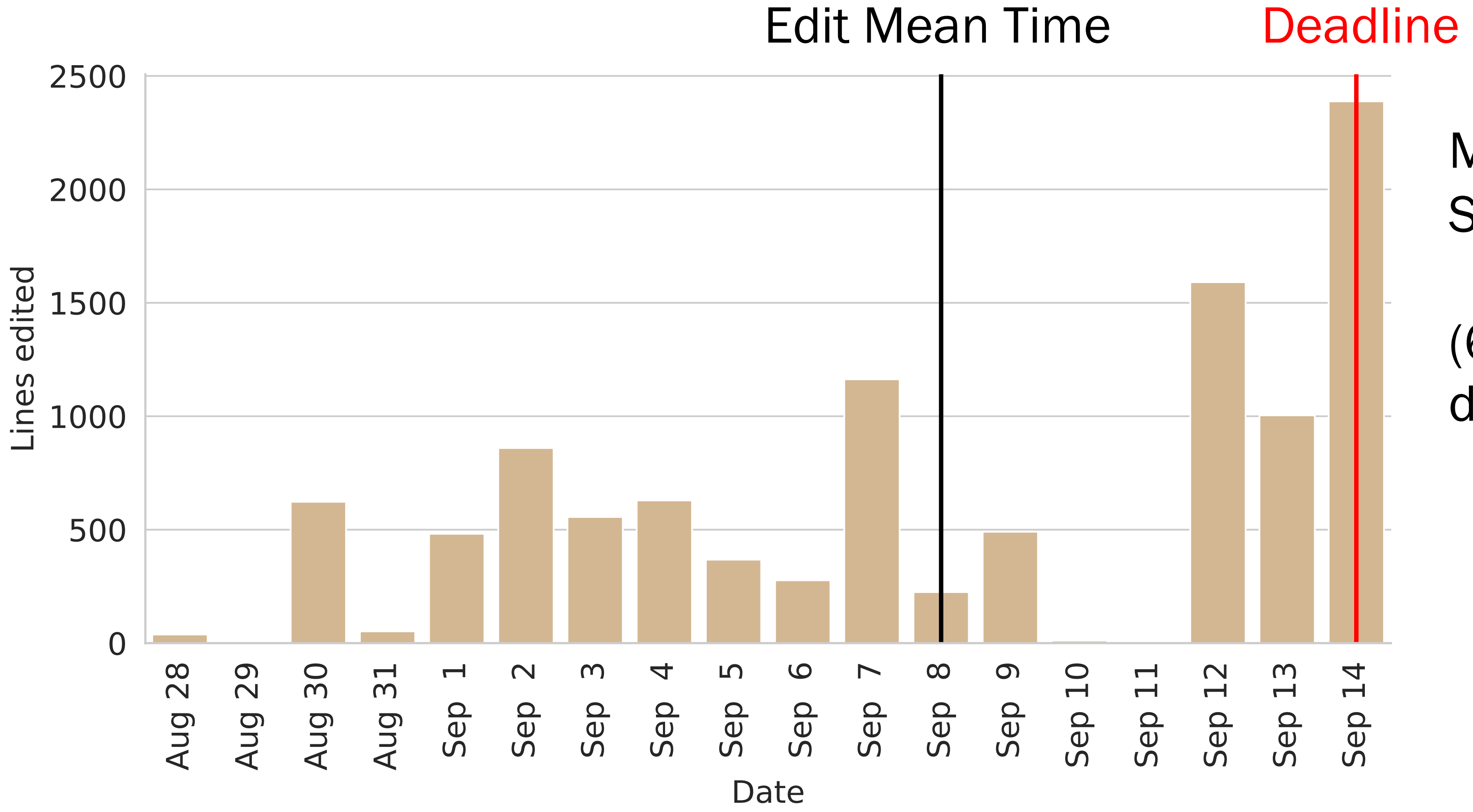
Proposed Measure of Working Early and Often

- **Early/Often Index**

Mean edit time in terms of days before the deadline



Early/Often Index: Example from Project 1 in Fall 2016



Mean edit time is September 8

(6 days before the deadline)

Validating the Early/Often Index

No readily available oracle to help measure accuracy.

Interviews with students $n = 7$

Manual inspection of Git histories $n = 12$

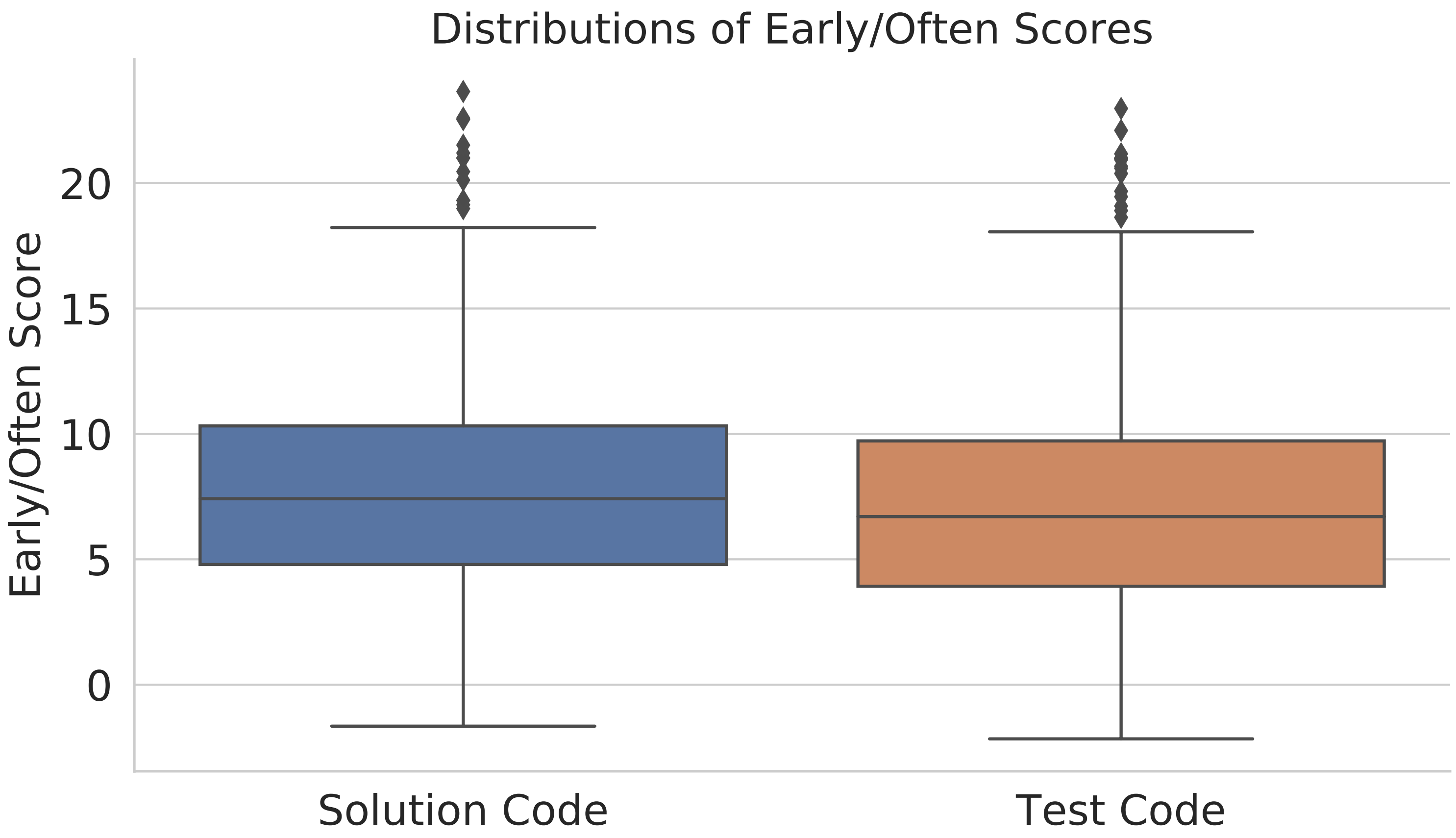
Agreement with

- Students' own perceptions of their process
- Project evolution observed in change histories

Identified differences between

- Individual students
- Individual assignments for the same student

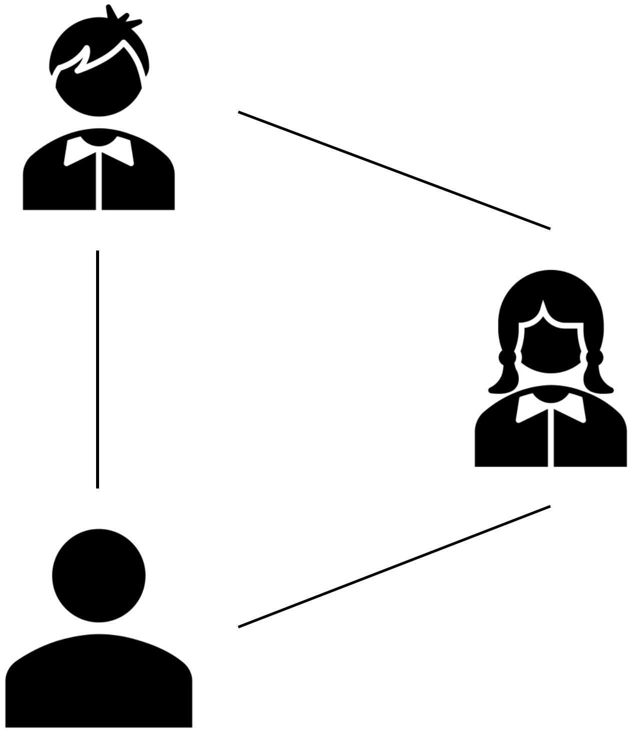
Students tend to work on projects <10 days before the deadline



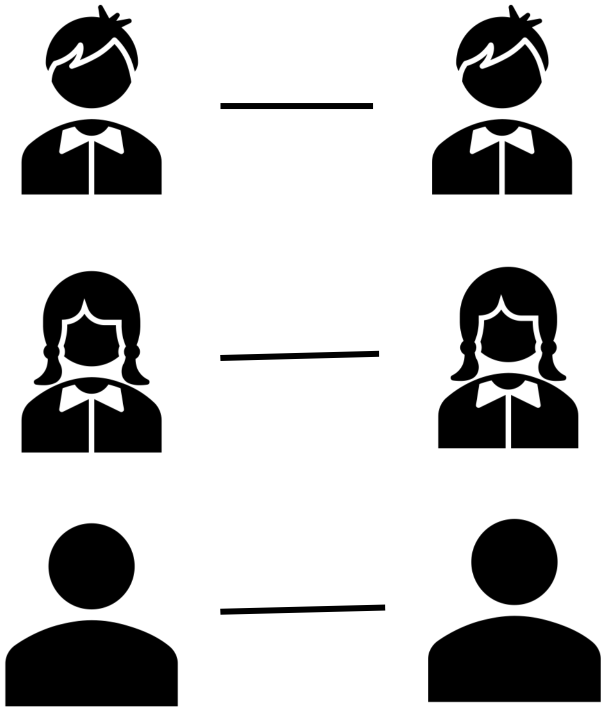
Similar distributions observed for

- Solution code editing
- Test code editing
- Program and test executions
- Debugger use

Research Method



Repeated Measures



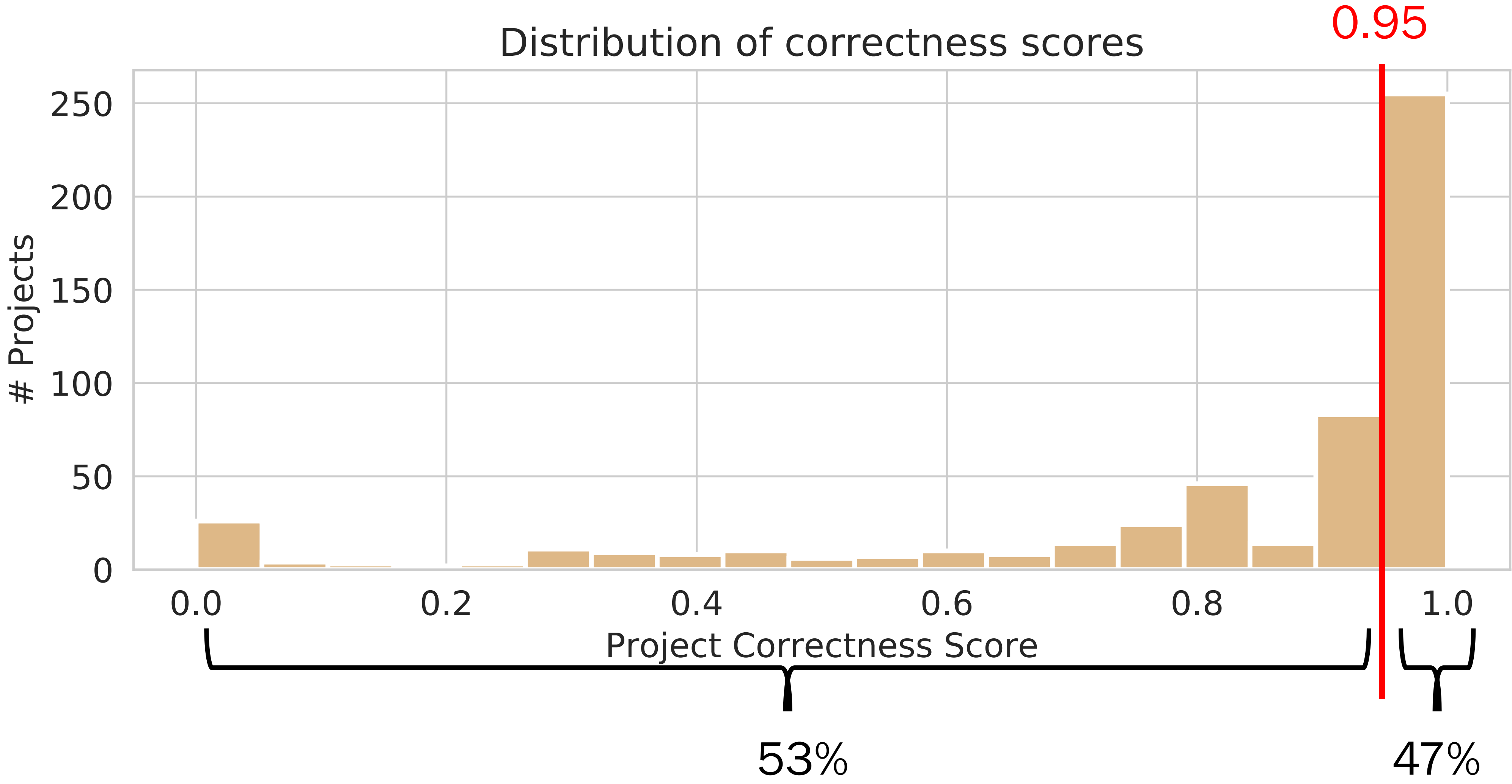
Mixed-model ANCOVA

Fixed effects: Development process metrics

Random effects: Individual students

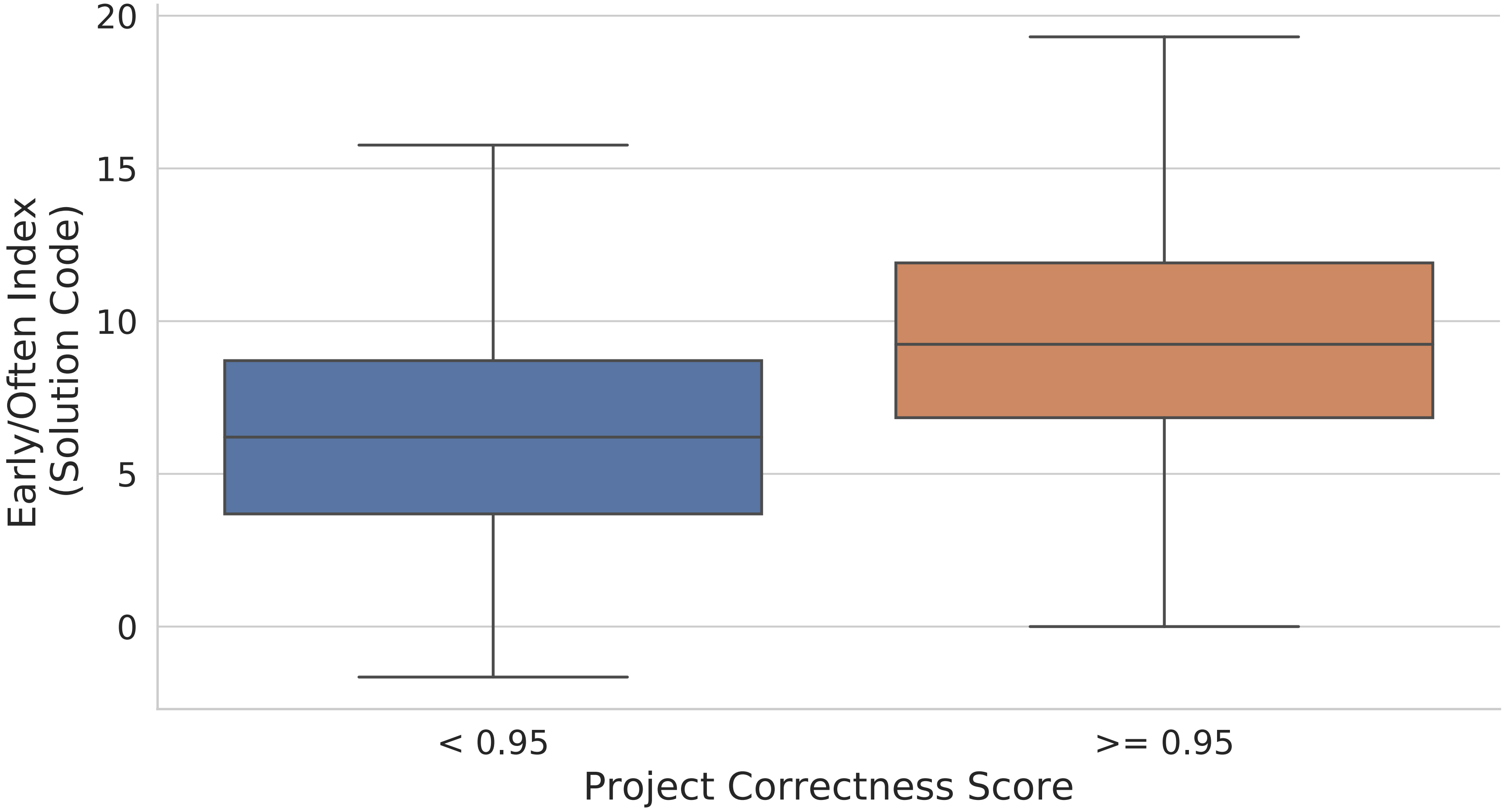
Project correctness

Students produced projects with higher correctness when they worked earlier and more often.



Project correctness

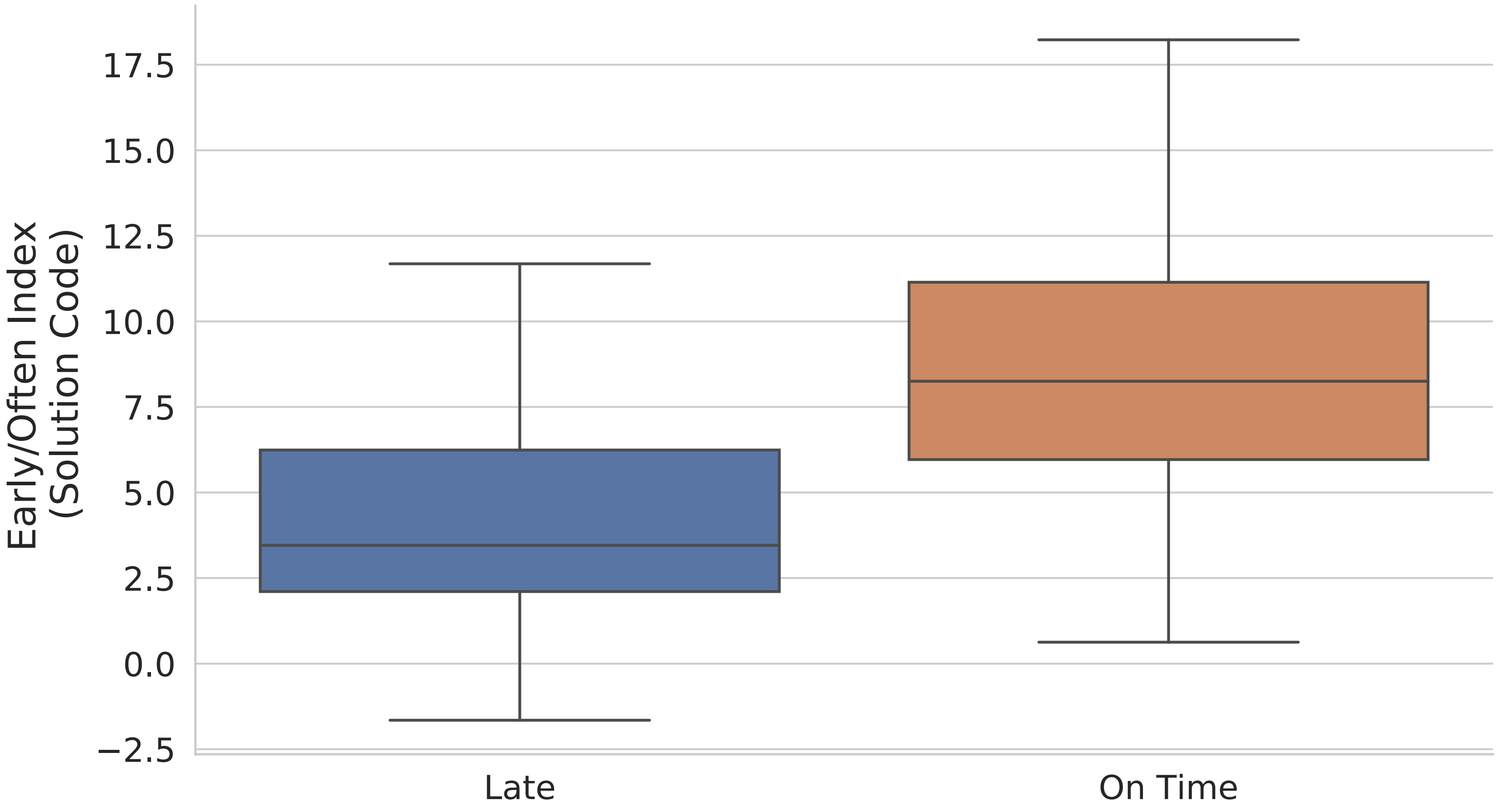
Students produced projects with higher correctness when they worked earlier and more often.



Cohen's $d = 0.69$

Time of submission

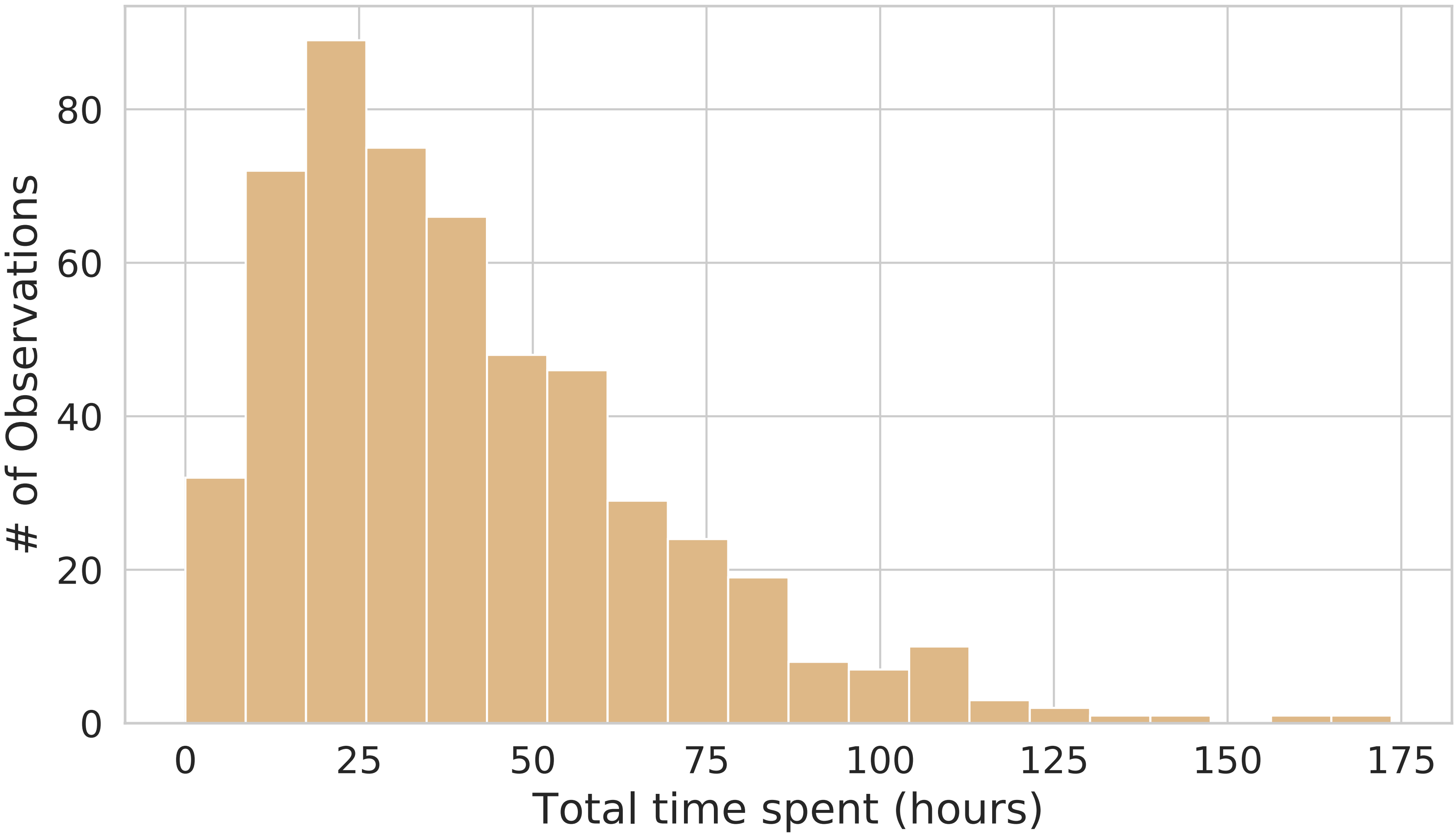
Students had earlier finish times and reduced likelihoods of late submission when they worked earlier and more often.



Cohen's $d = 1.10$

Total time spent on the project

Measured by adding up the lengths of individual work sessions

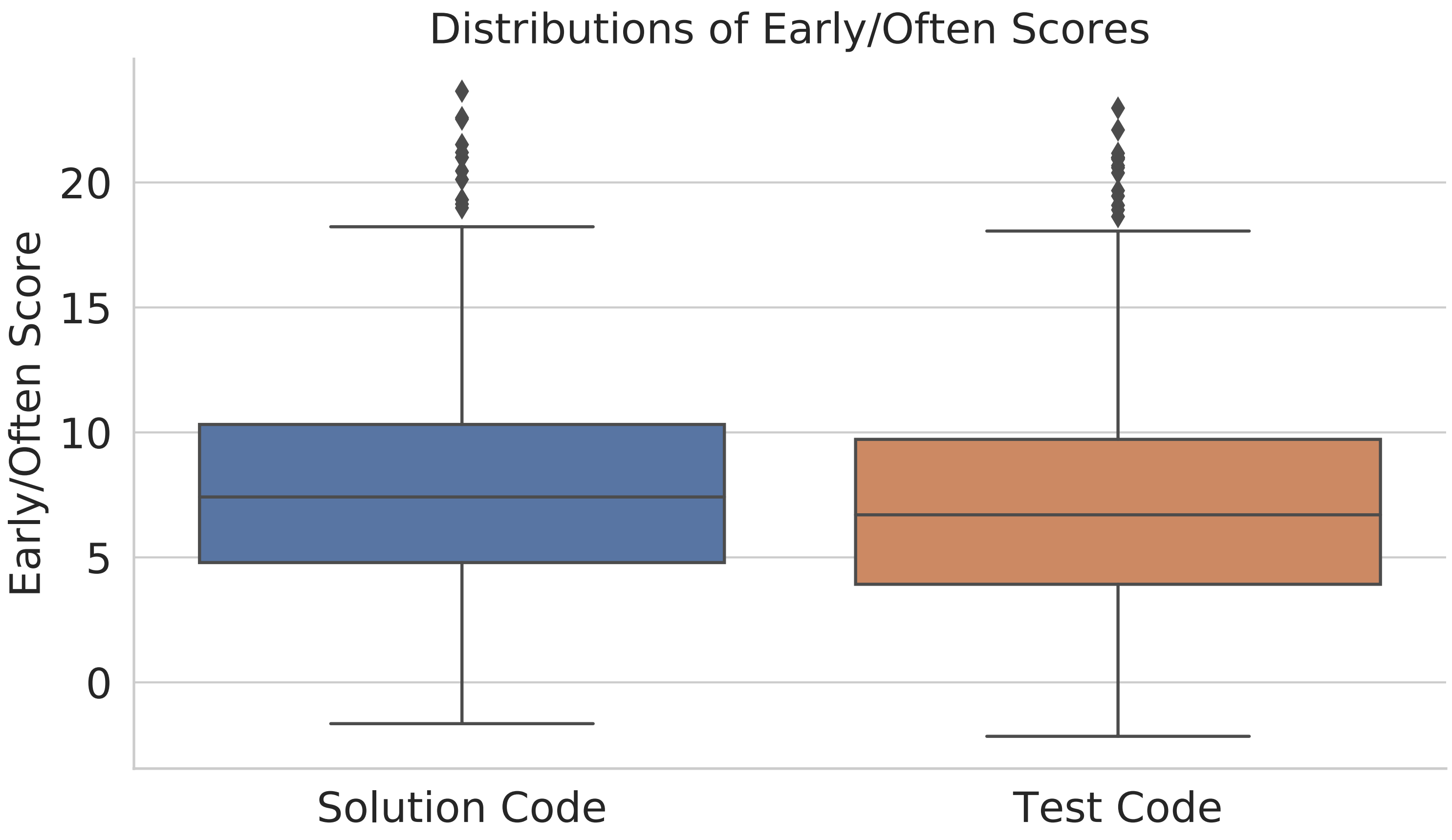


Students spent a median of 34.45 hours on each project.

No relationship with

- Solution edit mean time
- Project correctness

Students tend to work on projects <10 days before the deadline



Similar distributions observed for

- Solution code editing
- Test code editing
- Program and test executions
- Debugger use

Summary: Time Management on Software Projects

When students worked earlier and more often, projects

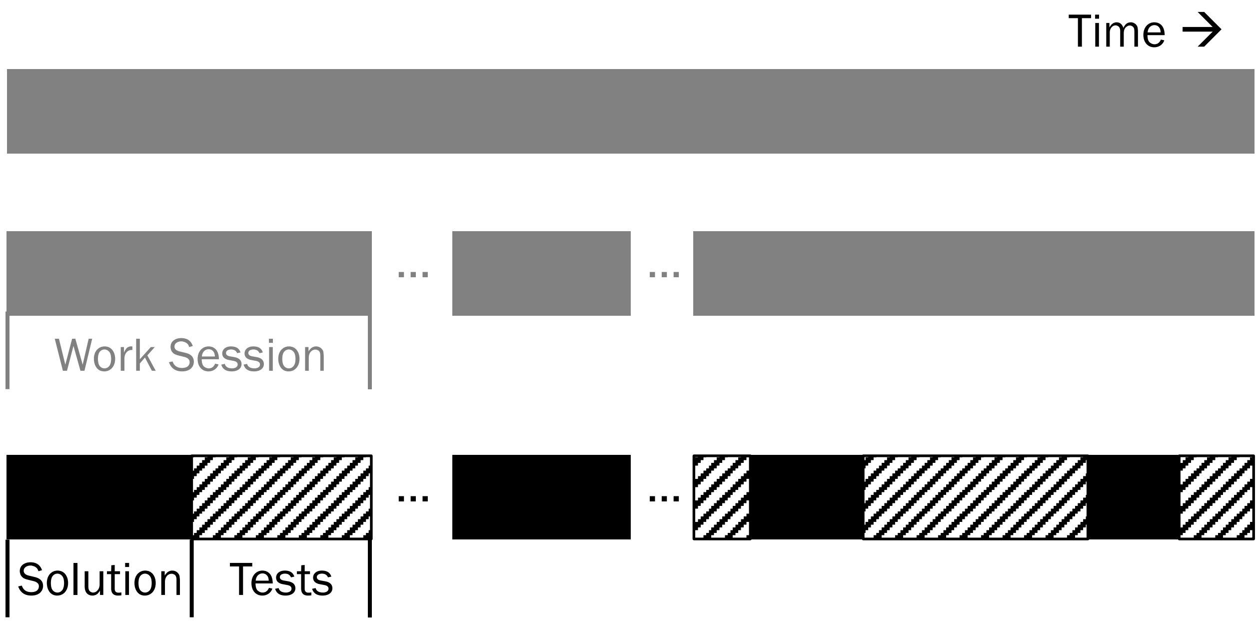
+ Were more correct
+ Were completed earlier

? Took the same amount of time to complete

Software Test Process

Better Feedback on Software Development

Programming effort



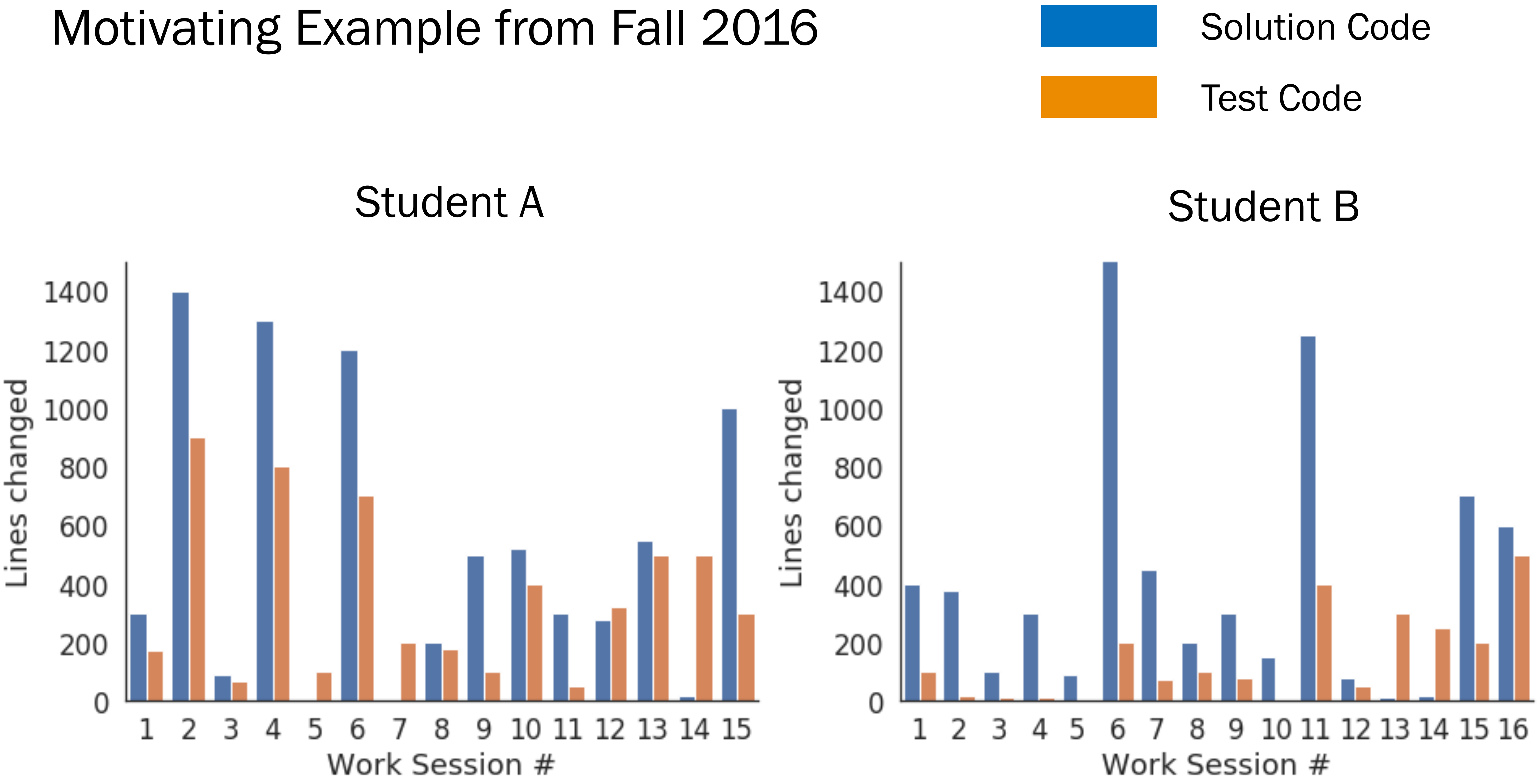
Feedback

Correctness: 100%
Code coverage: 89%

Procrastination

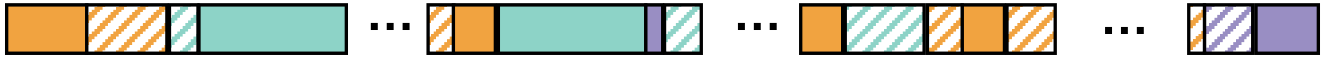
Balance/sequence of testing

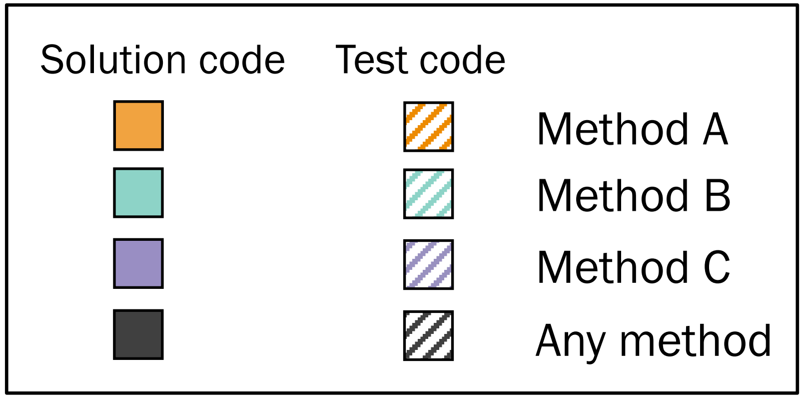
Motivating Example from Fall 2016



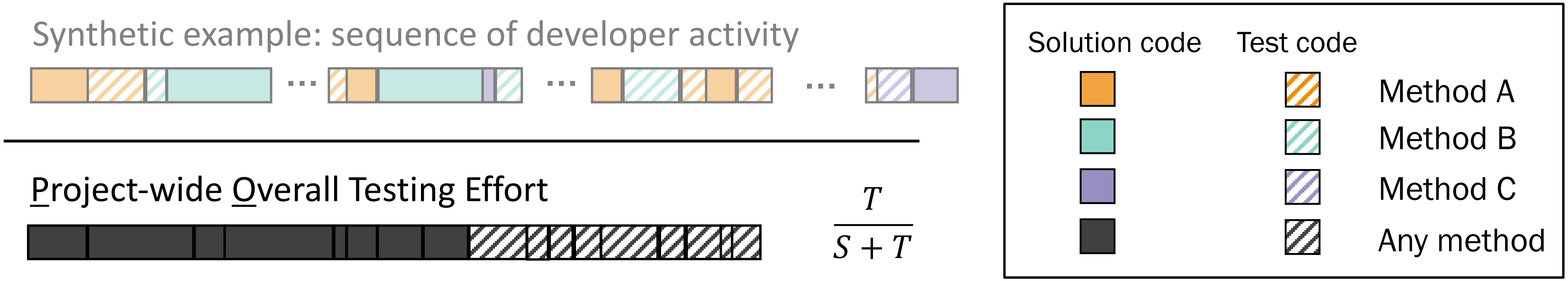
Proposed Metrics of Testing Effort

Synthetic example: sequence of developer activity

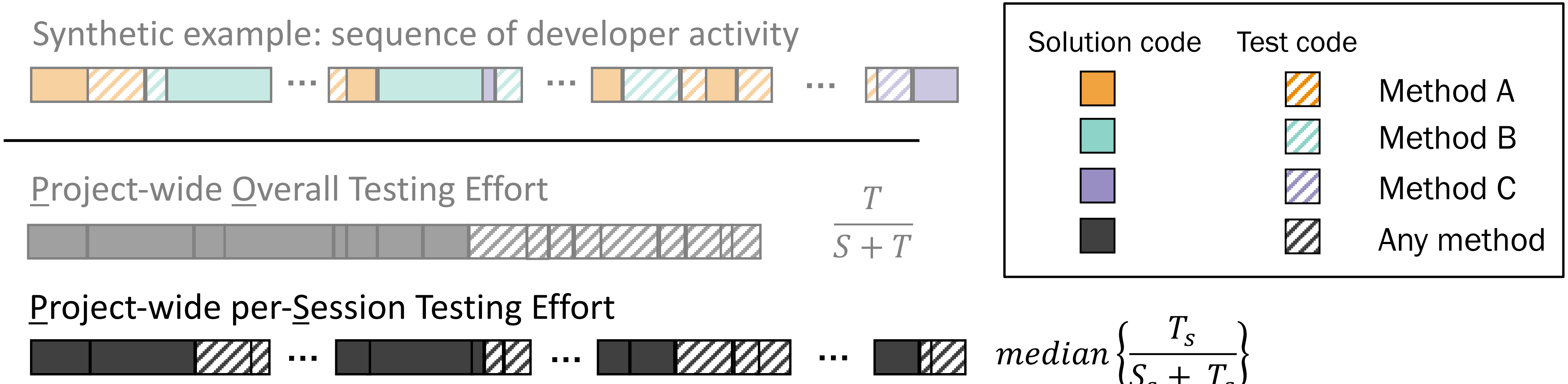




Proposed Metrics of Testing Effort



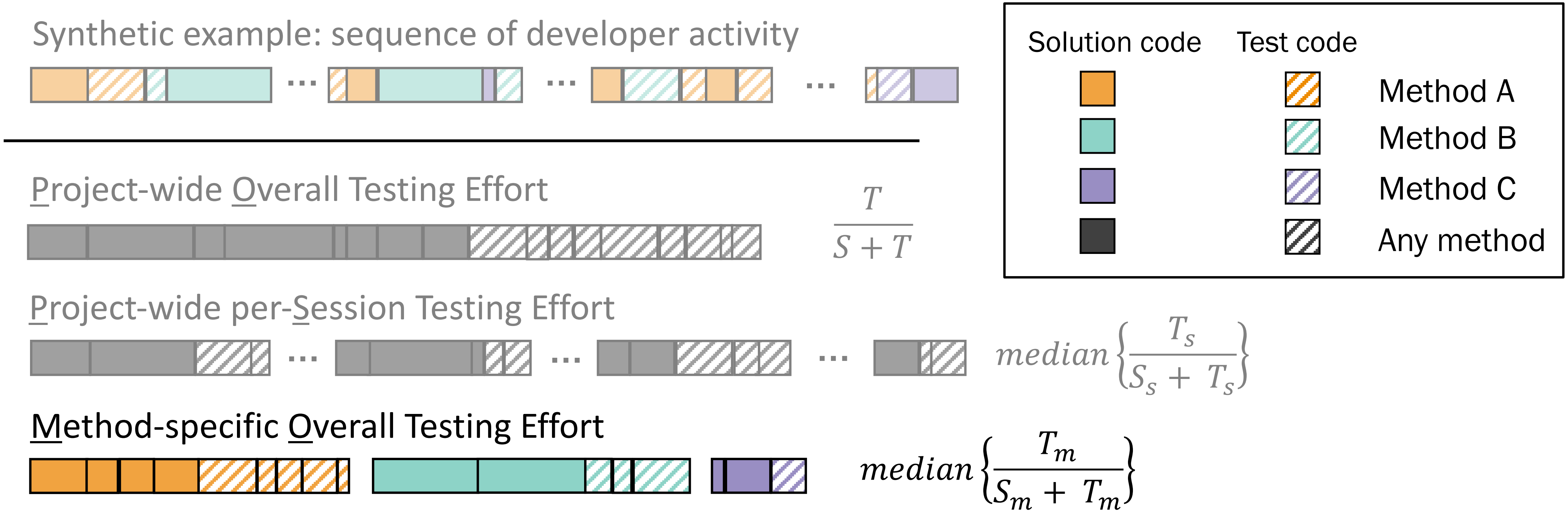
Proposed Metrics of Testing Effort



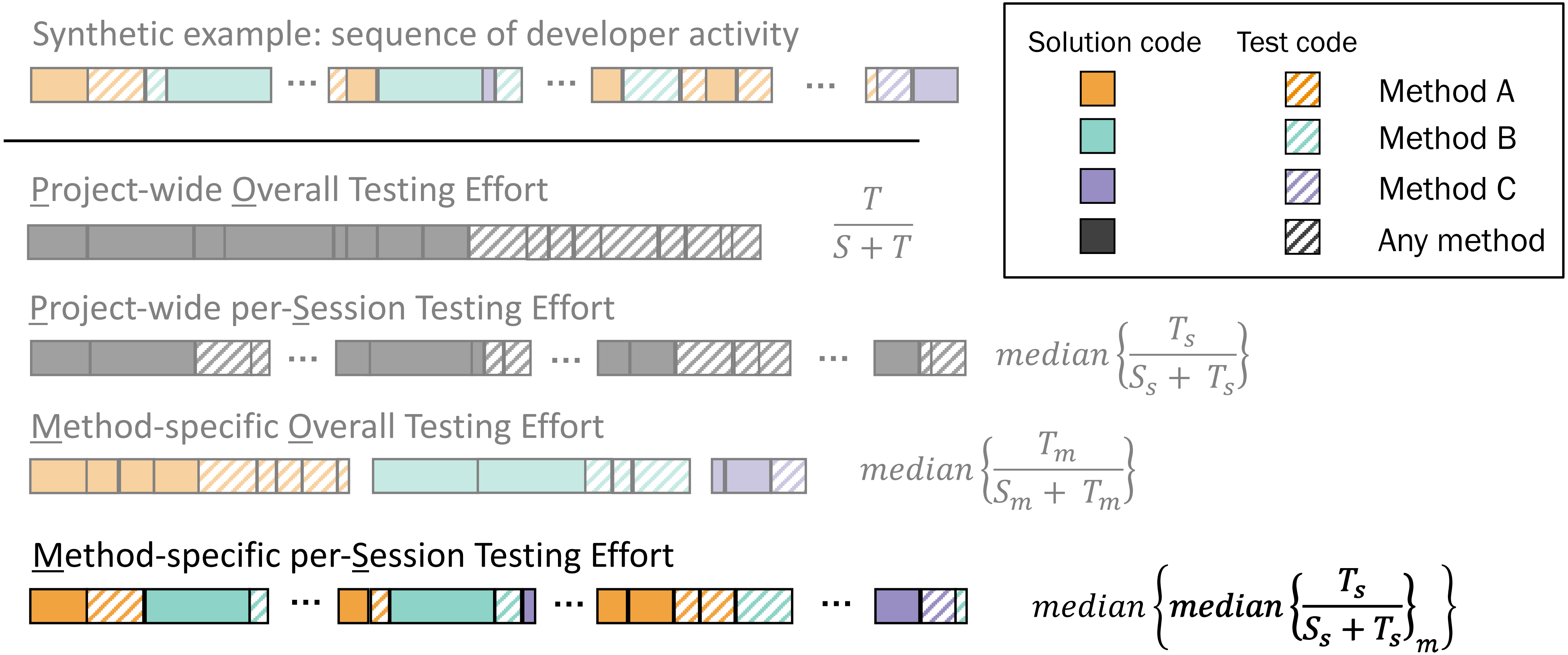
$$\frac{T}{S+T}$$

$$median\left\{\frac{T_s}{S_s + T_s}\right\}$$

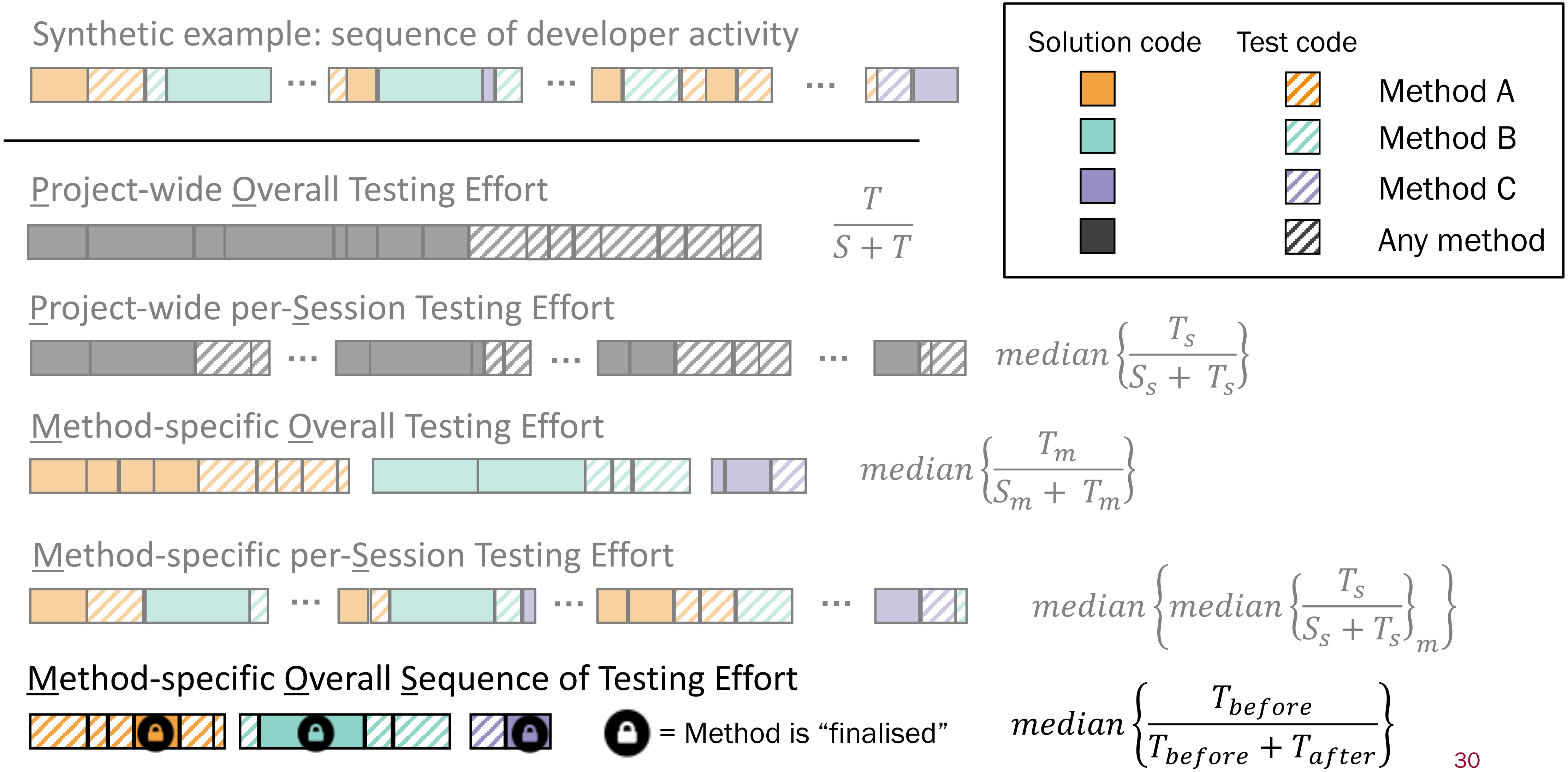
Proposed Metrics of Testing Effort



Proposed Metrics of Testing Effort

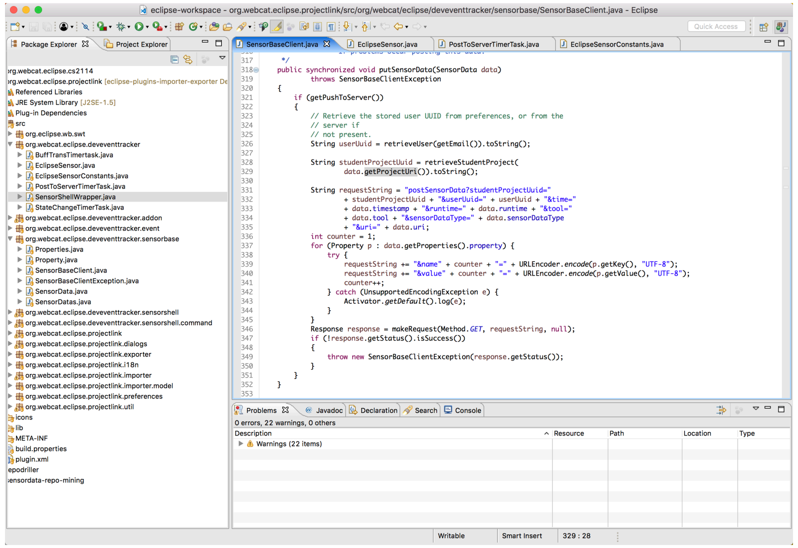


Proposed Metrics of Testing Effort



Data Collection – Automatically collected Git snapshots

- 400+ project implementations

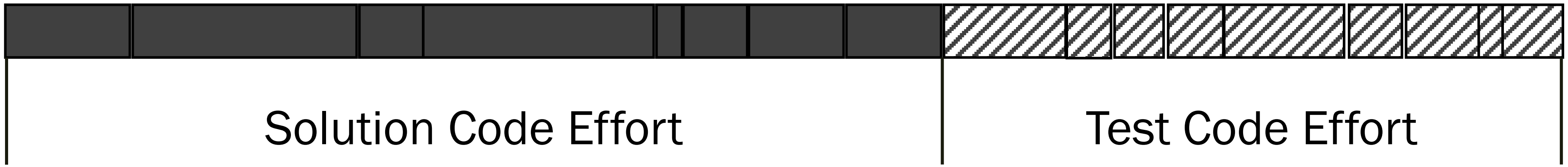


Edit Event

Type:	Edit
Time:	1477672862
Snapshot Id:	23479b3

Type	Size	Time
Change in method **`insertFront`**	+5	12:41:02
Change in method **`getSize`**	+1	12:41:02
Change in test for **`insertFront`**	+3	12:41:02

Overall Testing Effort

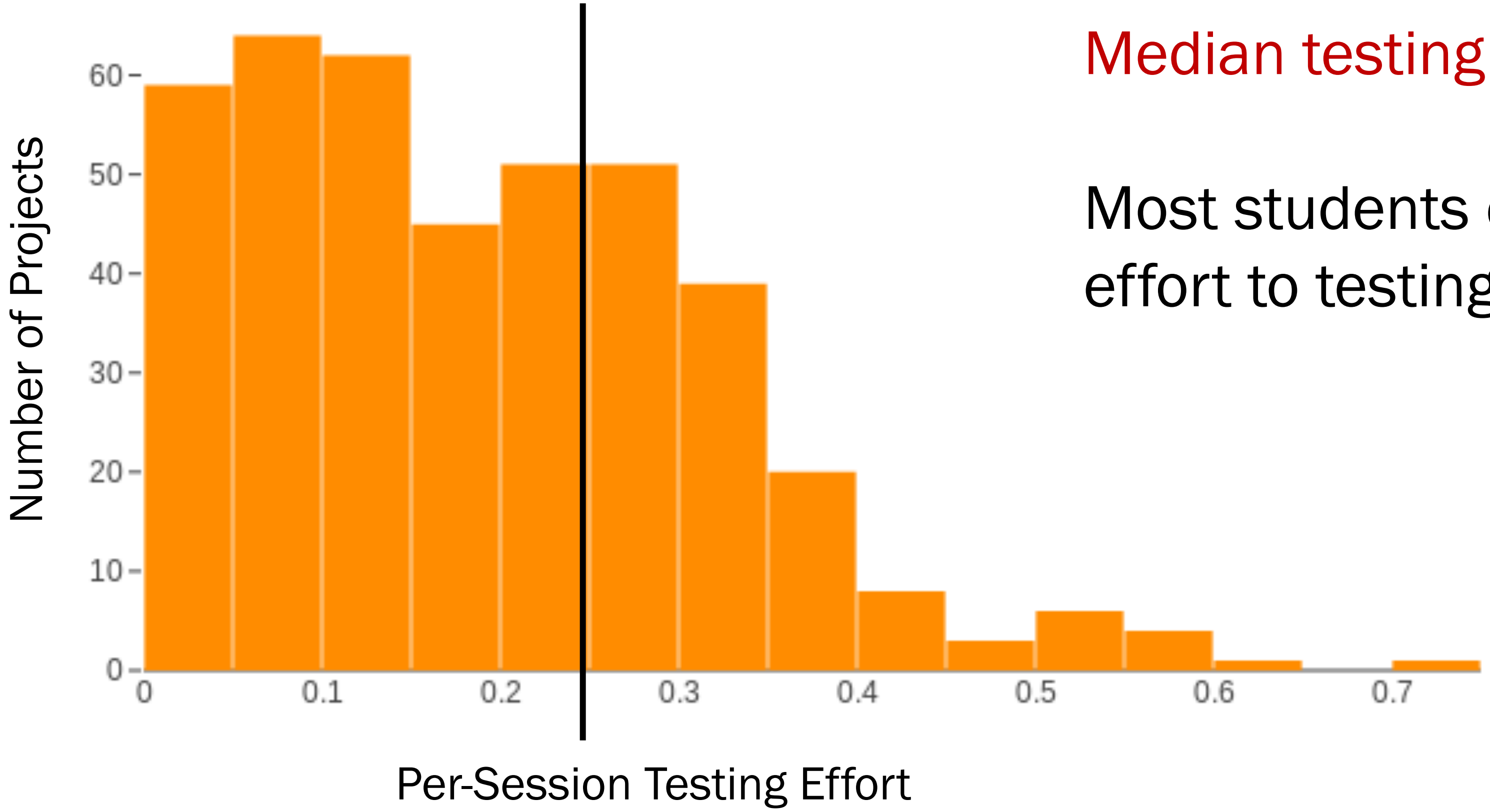


Relationship with project outcomes

When students put in more overall testing effort, they produced

- Programs with higher correctness
- Test suites with higher condition coverage

Per-Session Testing Effort



Median testing effort across work sessions

Most students devote less than 20–25% of effort to testing in most of their work sessions

Per-Session Testing Effort

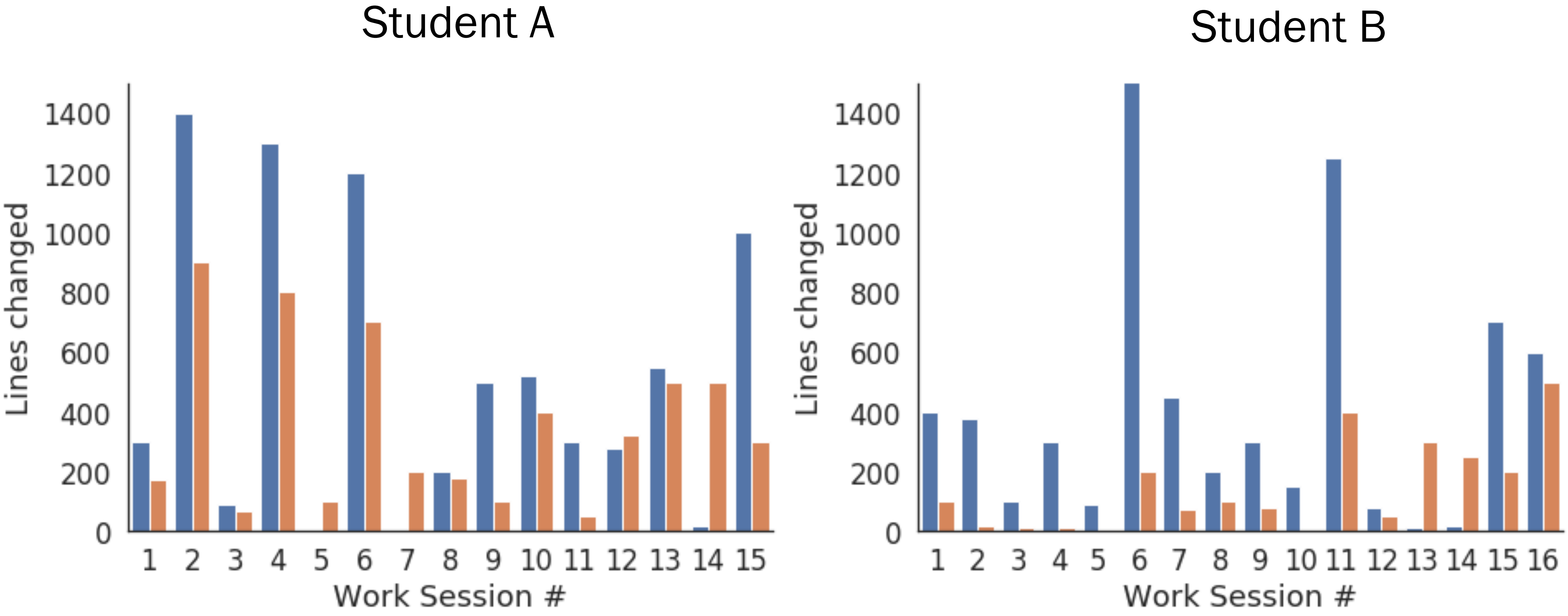
Relationship with project outcomes

When students put more testing effort in each session, they produced

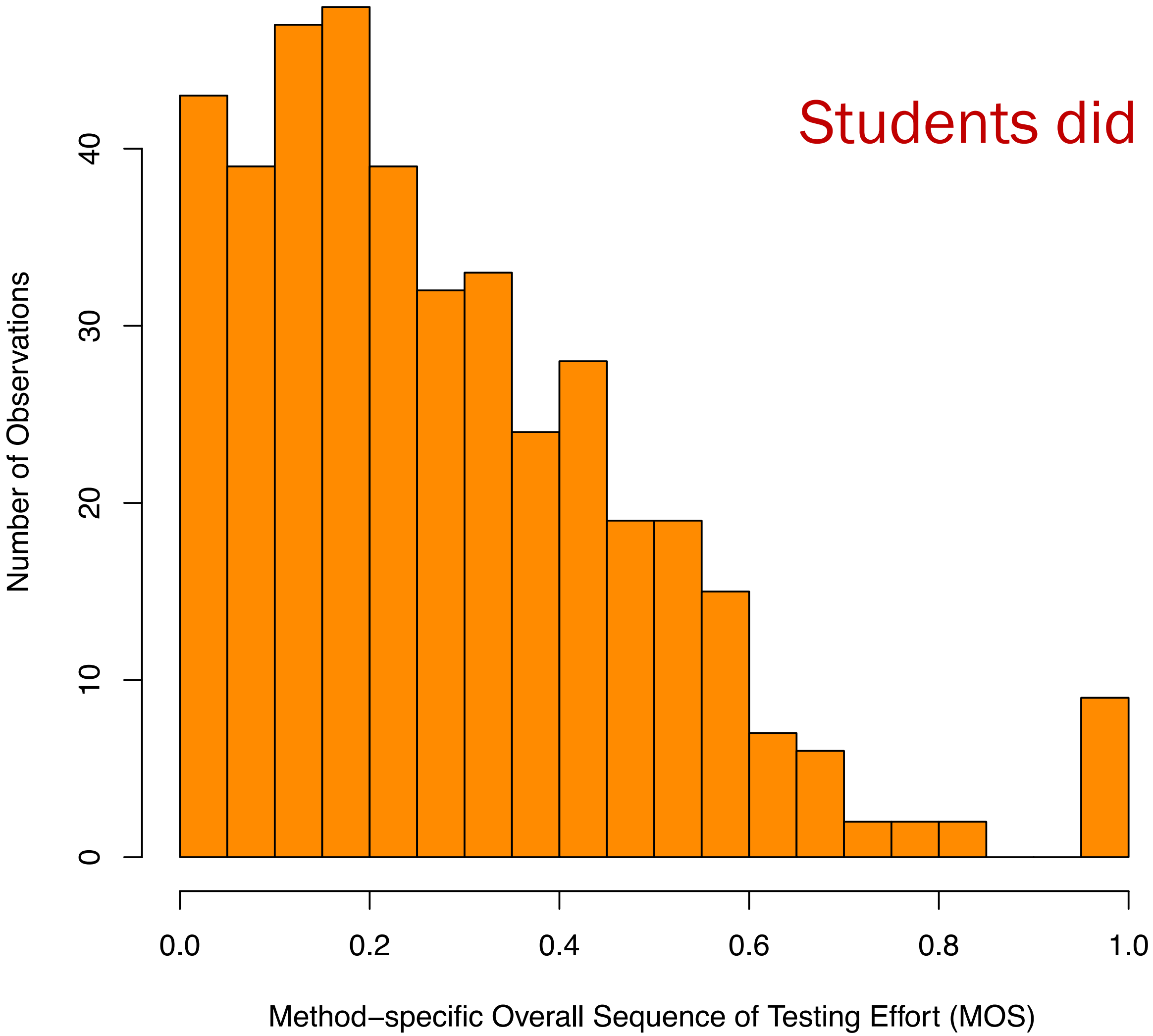
- Programs with higher correctness
- Test suites with higher code coverage

Motivating Example (Reprise)

Solution Code
Test Code



Method-specific Sequence of Testing Effort



Students did not tend to practice test-first development.

In 85% of projects, student did most of their testing after the relevant code under test was finalised.

Method-specific Sequence of Testing Effort

Relationship with project outcomes

When students did more testing before the relevant code was finalised

? Programs with no change in correctness

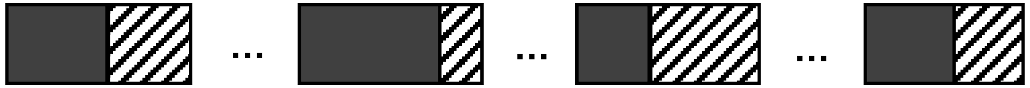
Test suites with *lower* code coverage

Summary: Incremental Testing on Software Projects

Overall testing effort

- + Correctness
- + Code Coverage

Per-session testing effort



- + Correctness
- + Code Coverage

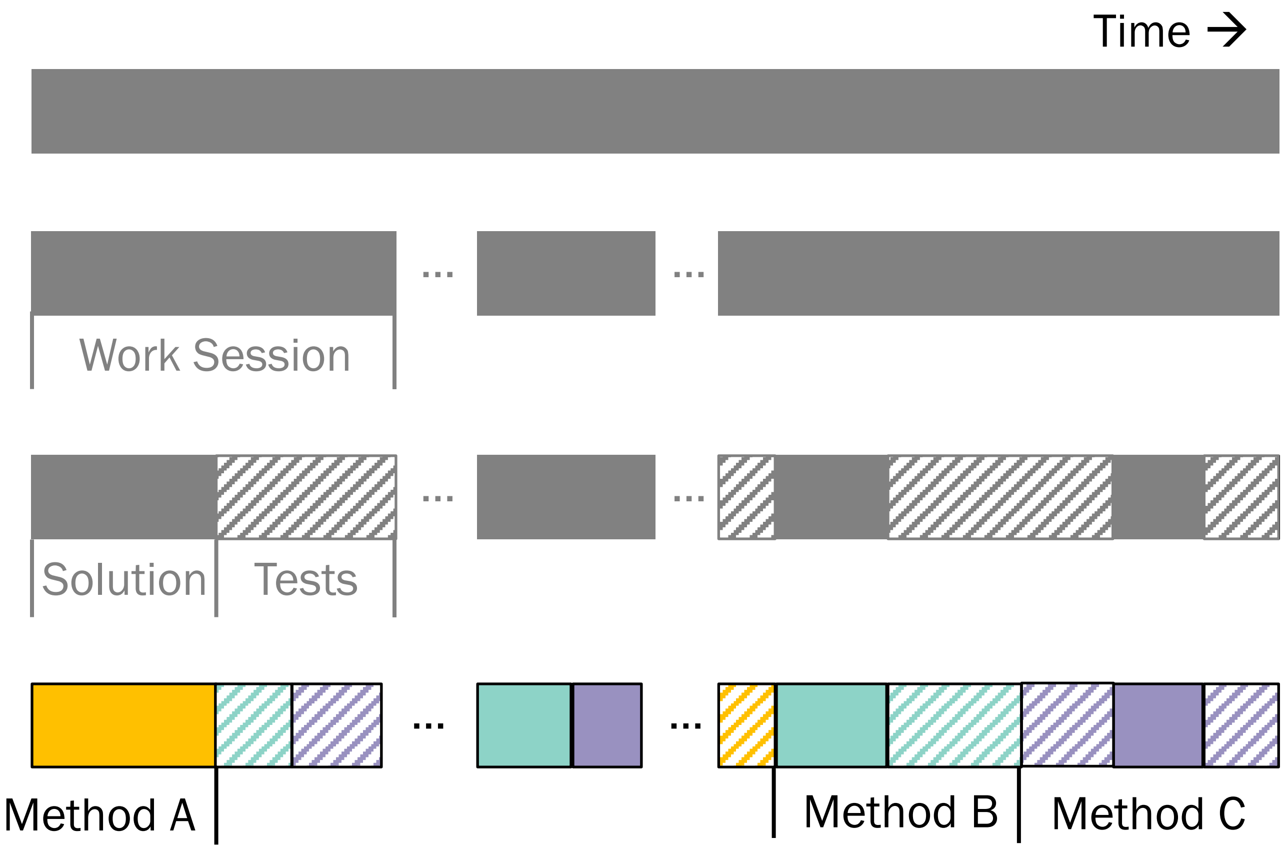
Tendency to "test first"

- ? Correctness
- − Code Coverage

Software Test Quality

Better Feedback on Software Development

Programming effort



Feedback

Correctness: 100%
Code coverage: 89%

Procrastination

Balance/sequence of testing

Thoroughness of testing

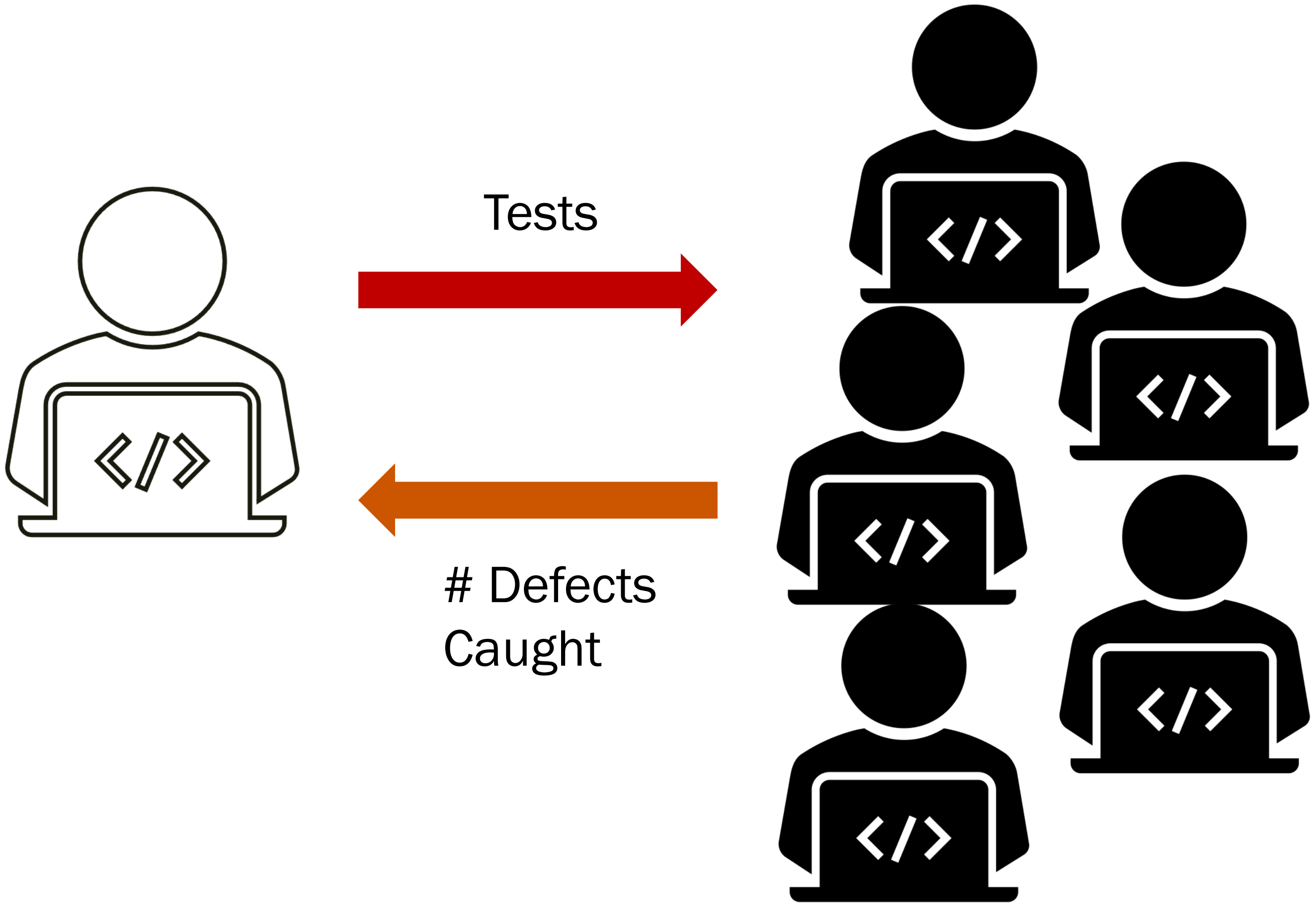
Code coverage

```
public static String compare(int a, int b) {
    if (a > b) {
        return "Greater than";
    } else if (a < b) {
        return "Less than";
    } else {
        return "Equal";
    }
}
```

```
@Test
public void testCompare() {
    compare(2, 3);
    assertTrue(
      compare(3, 2).length() > 0
    );
}
```

❌ Weak test adequacy criterion

Executing each student’s tests against every other student’s code



- ✓ Strong test adequacy criterion
- ✗ Precludes incremental feedback
- ✗ Assumes compatible API designs

Mutation Testing

Original Program

```
...
if (num1 >= num2) {
    return "GEQ";
} else {
    return "L";
}
```

Mutant Programs

```
...
if (num1 < num2) {
} el
}
```

```
...
if (num1 >= 0) {
} el
}
```

```
...
if (num1 > num2) {
    return null;
} el
}
```

```
...
if (true) {
    ...
} else {
    ...
}
```

Mutation Testing

- ✓ Strong test adequacy criterion
- ✓ Allows incremental feedback
- ✗ Prohibitively high computational cost

Context

Web-CAT
Automatic grading using student-written tests

1 submission / 5.5 seconds

Peak: 1 submission / 1.5 seconds

pitest.org
Coles 2016

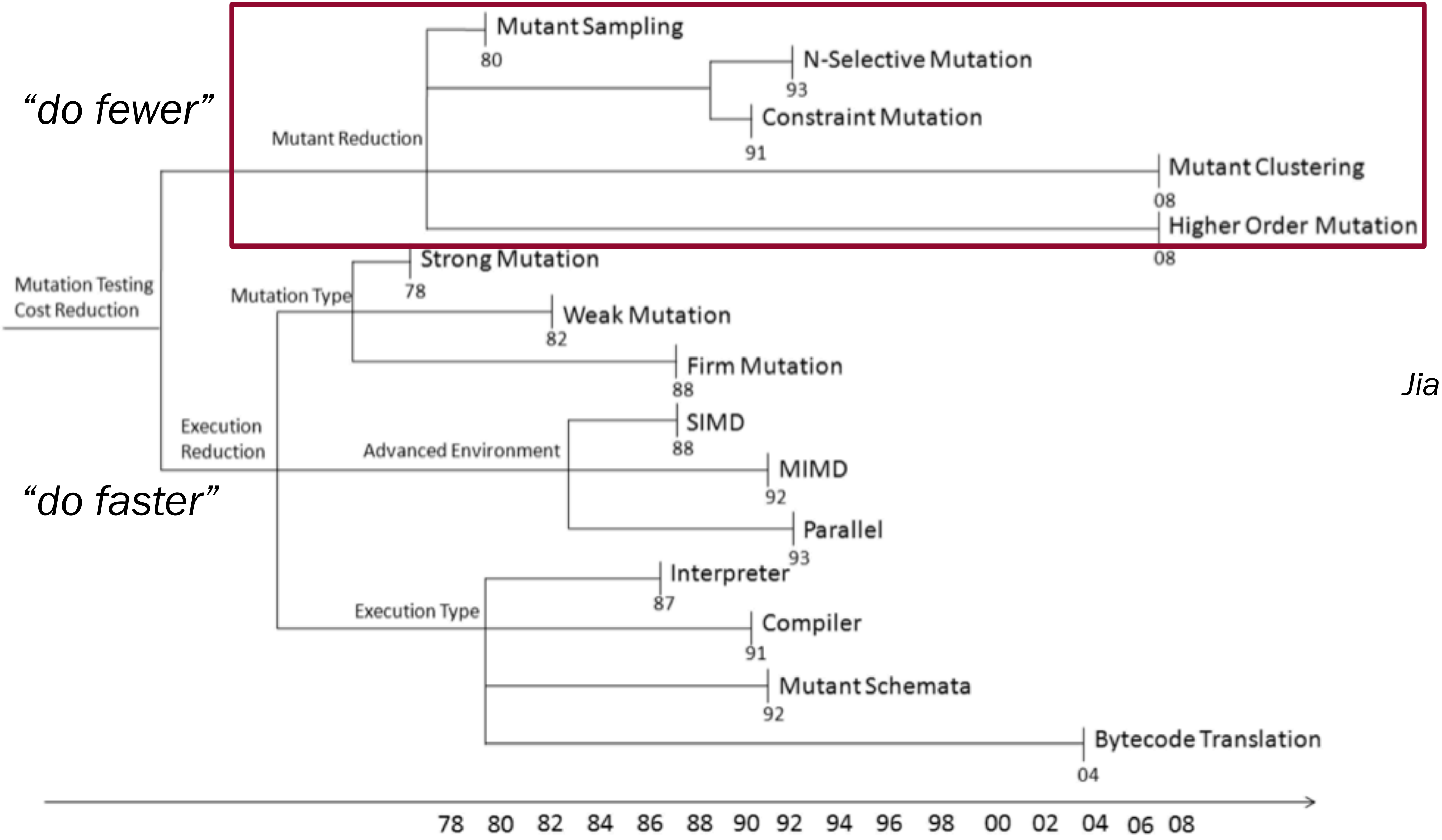
	# submissions	Mutation Testing
CS 2	1,019	30 seconds
CS 3	370	5.04 minutes

!

How can we reduce the cost of mutation analysis?

“do fewer”

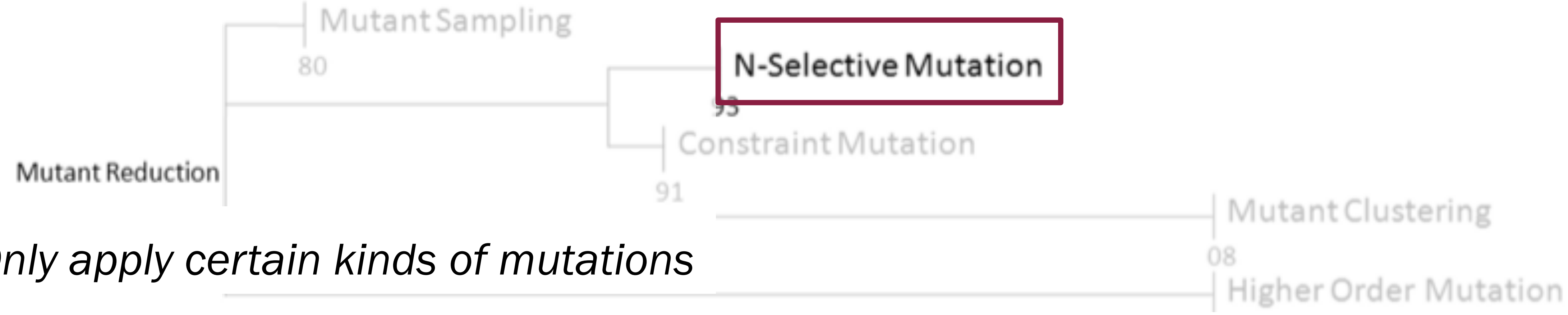
“do faster”



Jia & Harman 2010

How can we reduce the cost of mutation analysis?

Jia & Harman 2010



Only apply certain kinds of mutations

Replace conditionals with Boolean literals

`a > b → true, a > b → false`

Replace arithmetic expressions with its operands

`a + b → a, a + b → b`

Can we do this fast enough for incremental feedback?

Mutation by Deletion

(Offutt et al. 2014)

pitest.org

Mutator	Example
Delete conditional expressions	`a > b → true`
Delete arithmetic operators	`a + b → a`
Delete non-void method calls	`getString() → null` `getInt() → 0`
Delete void method calls	`performAction() →`
Delete assignments to member variables	`this.age = 25 → this.age = 0`
Delete constructor calls	`new String() → null`

Context

Web-CAT

Automatic grading using student-written tests

1 submission / 5.5 seconds

Peak: 1 submission / 1.5 seconds

pitest.org

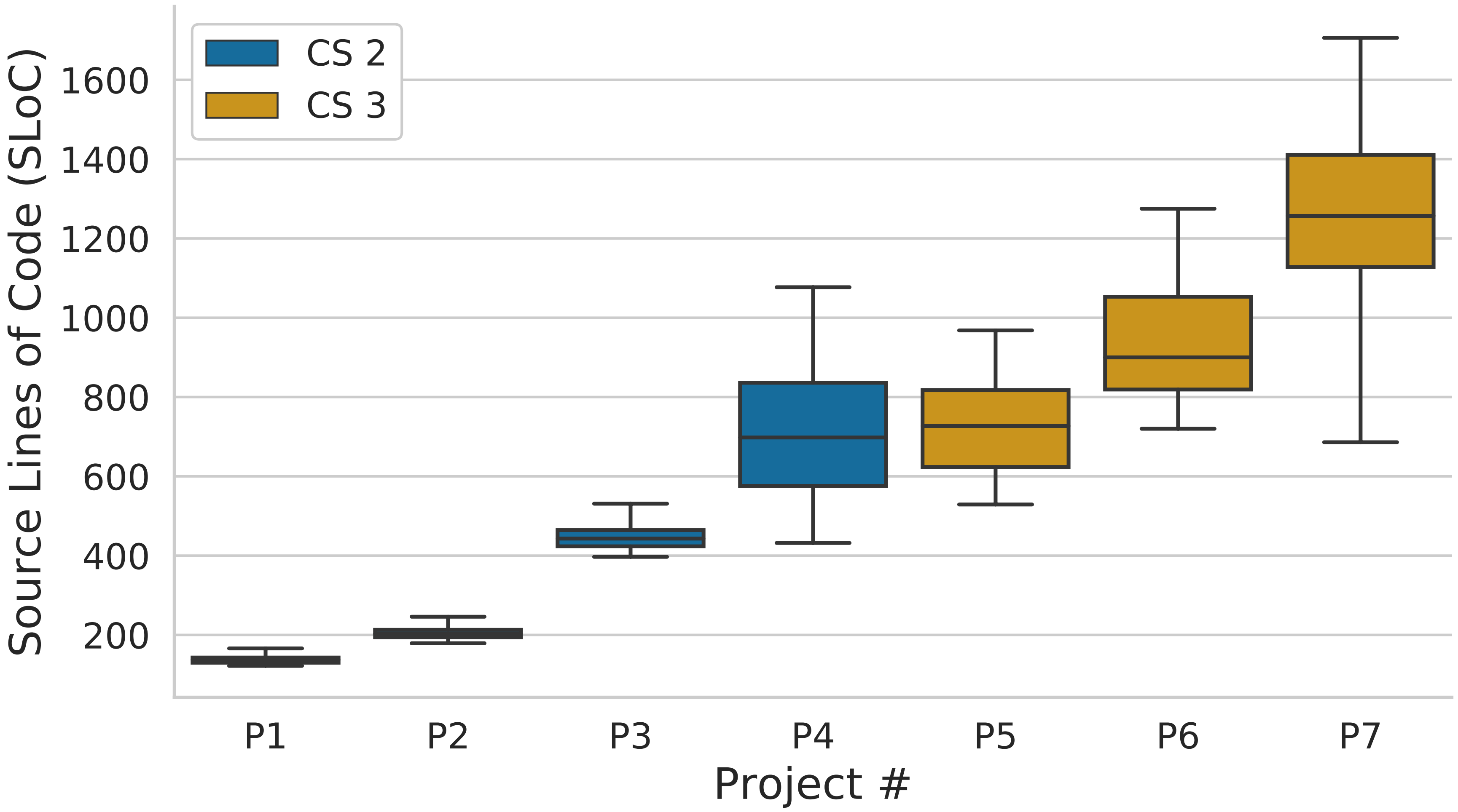
	# submissions	All ~30 mutators	6 deletion mutators	Our approach
CS 2	1,019	30 seconds	4.75 seconds	10% of the cost
CS 3	370	5.04 minutes	1.11 minutes	90% of the effectiveness

Incremental subsets of mutation operators

Forward selection. Which Deletion mutators best predict the full mutation score?

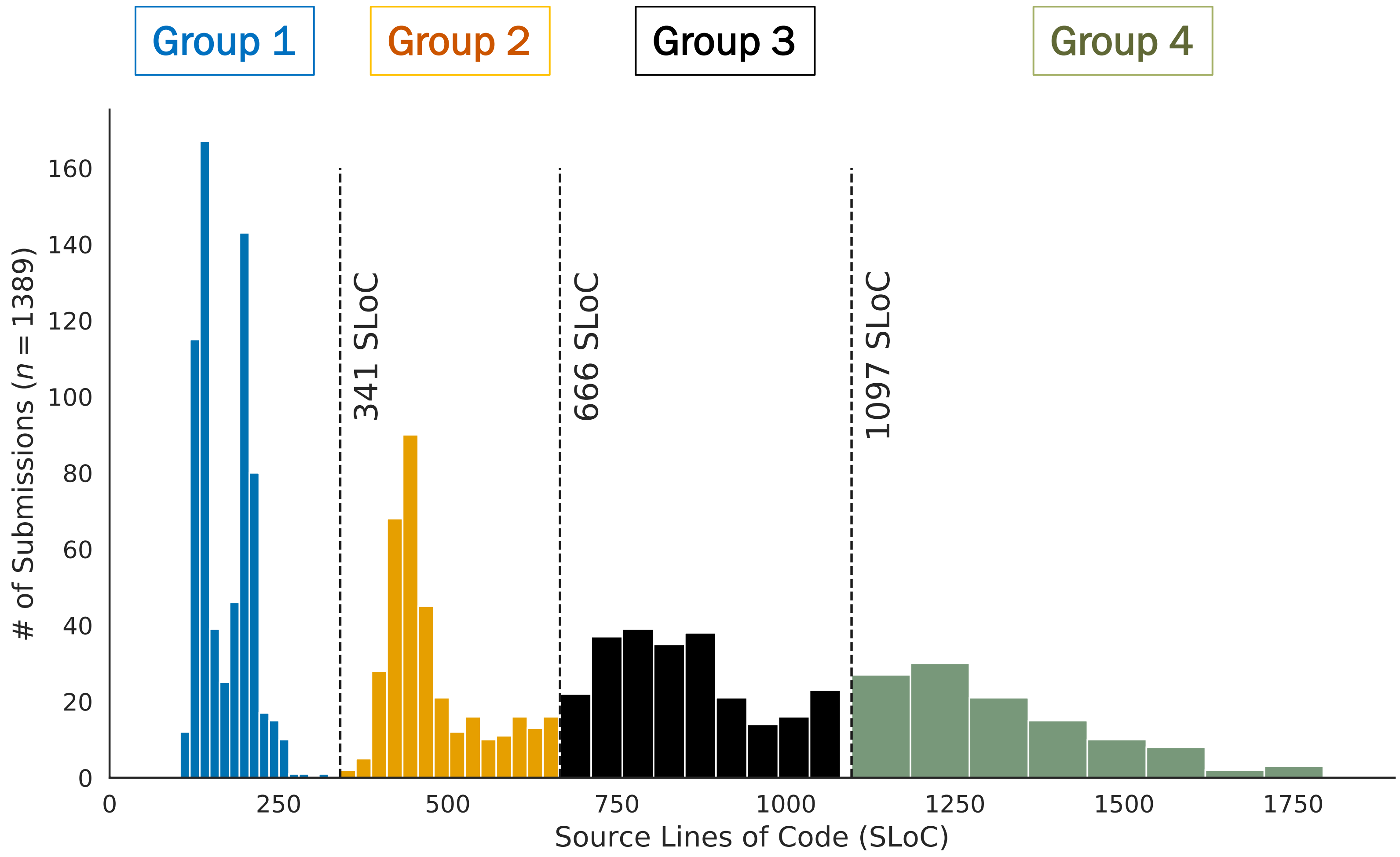
Mutators Added	# of Mutants Produced (per KSLoC)		Effectiveness
	Median	% of All Mutants	
RemoveConditionals	102	7.04%	78%
ArithmeticOperatorDeletion	140 (+38)	9.67%	88%
NonVoidMethodCalls	236 (+96)	16.30%	91%
VoidMethodCalls	[illegible]	[illegible]	[illegible]
MemberVariables	[illegible]	[illegible]	[illegible]
ConstructorCalls	[illegible]	[illegible]	[illegible]

How does the size of the program relate to the chosen operators?



How does the size of the program relate to the chosen operators?

Group projects based on size



How does the size of the program relate to the chosen operators?

Group projects based on size

Grow the subset by choosing the next Deletion operator

Operator Added	# of Mutants Produced		Adjusted R^2
	Median	% of All Mutants	
RemoveConditionals	102	7.04%	0.78
ArithmeticOperatorDeletion	140	9.67%	0.88
NonVoidMethodCalls	236	16.30%	0.91
VoidMethodCalls	240	16.57%	0.92
MemberVariables	271	18.72%	0.92
ConstructorCalls	283	19.54%	0.92

1-operator subset

2-operator subset

3-operator subset

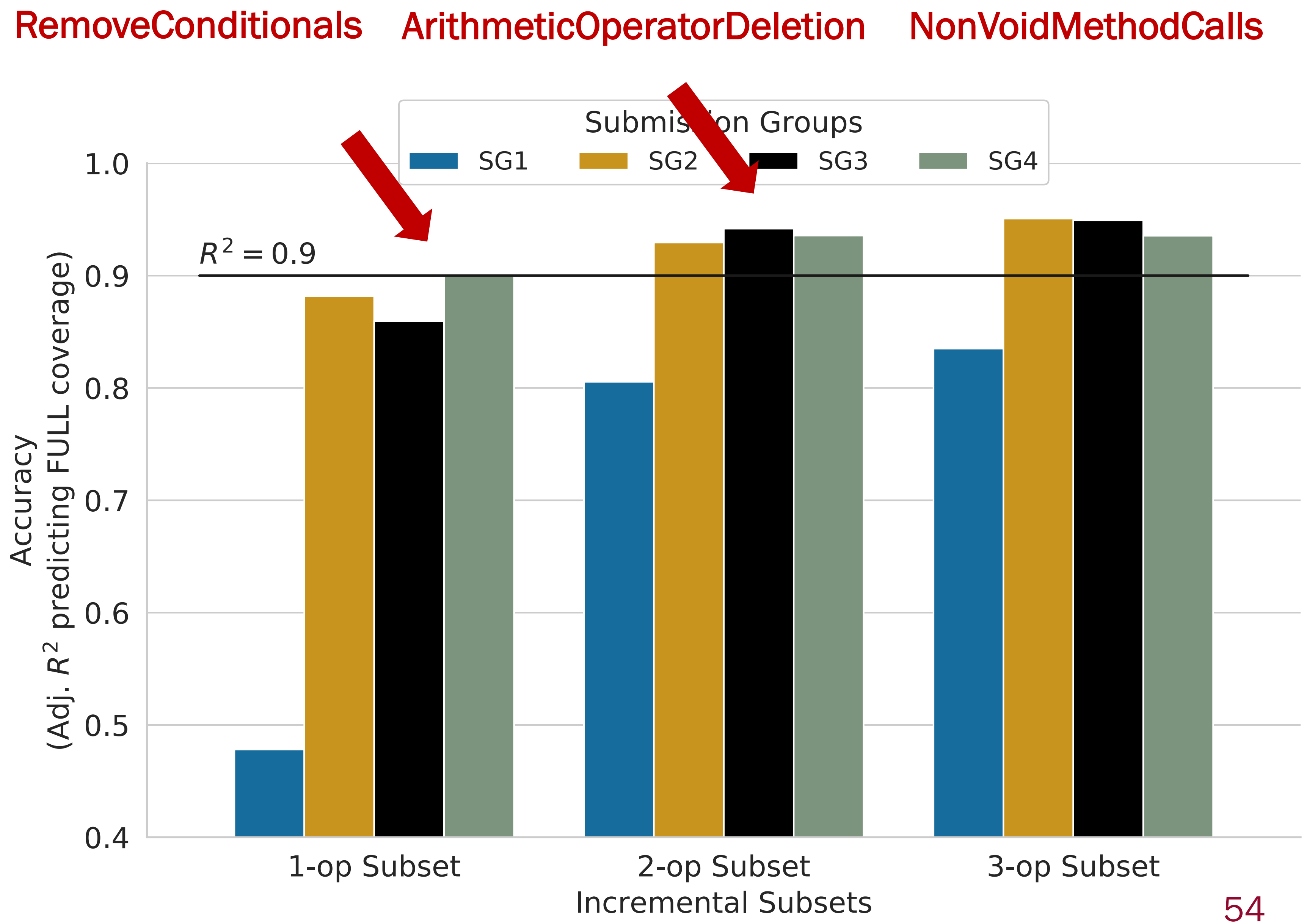
6-operator subset

How does the size of the program relate to the chosen operators?

Group projects based on size

Grow the subset by choosing the next Deletion operator

Evaluate each incremental subset on each group



Summary: Mutation Analysis

Using ALL mutators is too expensive

Using DELETION mutators is also too expensive (for larger projects)

Only deleting Conditionals and Arithmetic Operators

- 10% of the work (~30 seconds for CS 3 projects)
- 90% of the effectiveness

Can reduce further based on project size

- Large: Conditionals (~20 seconds)
- Medium: Conditionals + Arithmetic Operators (~30 seconds)
- Small: ALL mutators? (~16 seconds)

Closing Remarks

Summary

Time management

- Students are spending 30–40 hours on projects mostly in the last 10 days!
- Working early and often can lead to more constructive time spent on projects.
- Might lead to increased correctness and earlier finish times

Incremental Testing

- There is some evidence of incremental testing, but it can be improved
- We can identify it with lead time before the deadline
- Might lead to increased correctness and stronger test suites

Mutation Testing

- Much better method of evaluating test suites, hindered by computational cost
- Simple approaches can maintain effectiveness while drastically reducing cost
- Recommended approaches differ based on project under test

Future Work

Designing and deploying feedback based on software process measurements.

Why are students not self-regulating their development habits?

Mutation operator selection based on pedagogical value AND program characteristics.

Can this work be applied to industry or open-source projects?

What is good process for end-user software developers?

Longitudinal studies.

Thanks!

Committee members

Cliff Shaffer

Steve Edwards

Francisco Servant

Dennis Kafura

Jaime Spacco

National Science Foundation

Instructors and students of CS 3114 at VT

COLLEGE OF ENGINEERING
COMPUTER SCIENCE
VIRGINIA TECH

Summary

Time management

- Students are spending 30–40 hours on projects mostly in the last 10 days!
- Working early and often can lead to more constructive time spent on projects.
- Might lead to increased correctness and earlier finish times

Incremental Testing

- There is some evidence of incremental testing, but it can be improved
- We can identify it with lead time before the deadline
- Might lead to increased correctness and stronger test suites

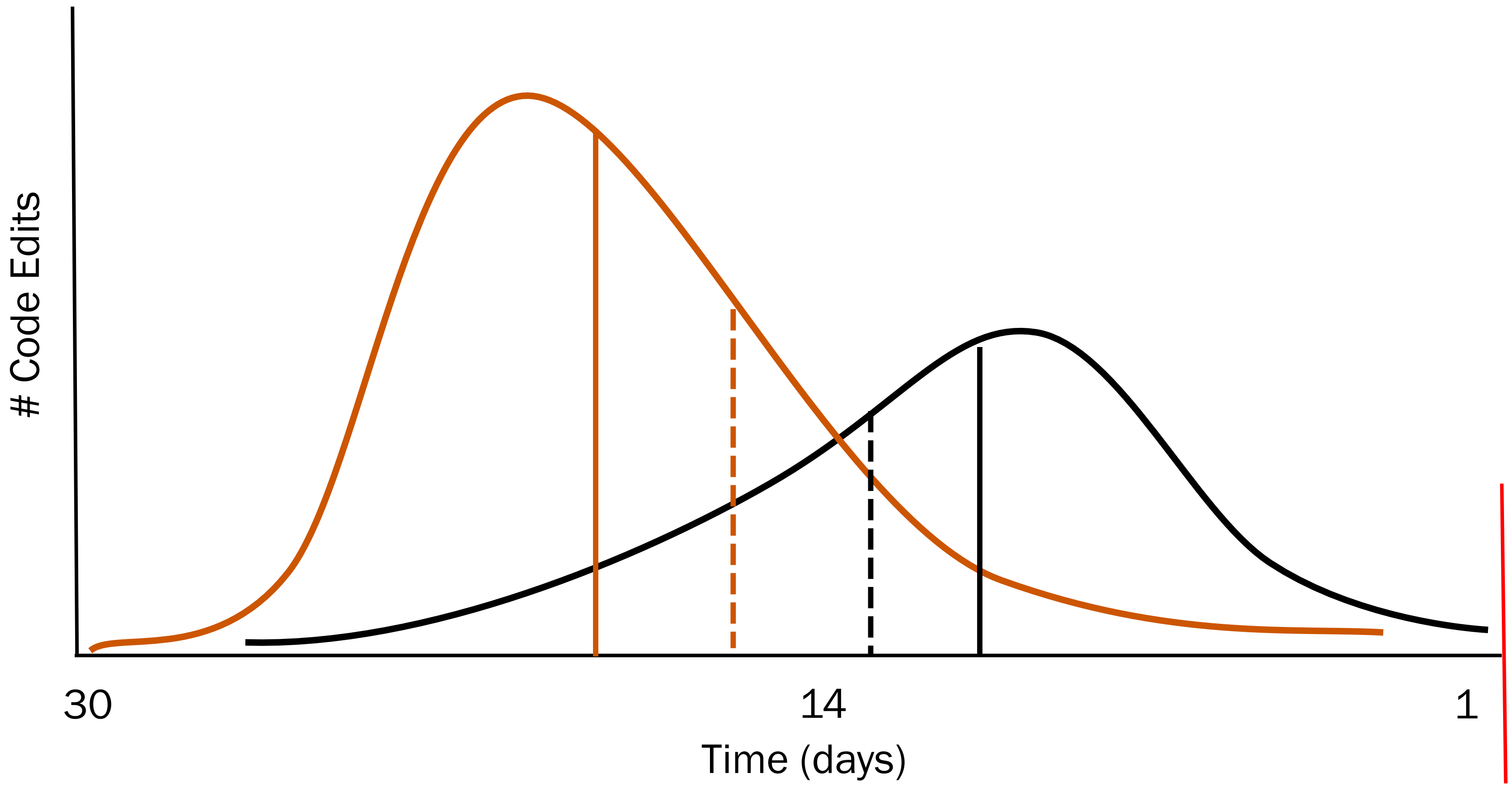
Mutation Testing

- Much better method of evaluating test suites
- Simple approaches can maintain effectiveness while drastically reducing cost
- Recommended approaches differ based on project under test

Bonus Slides

Other measures of central tendency

Median edit time in terms of days before the deadline

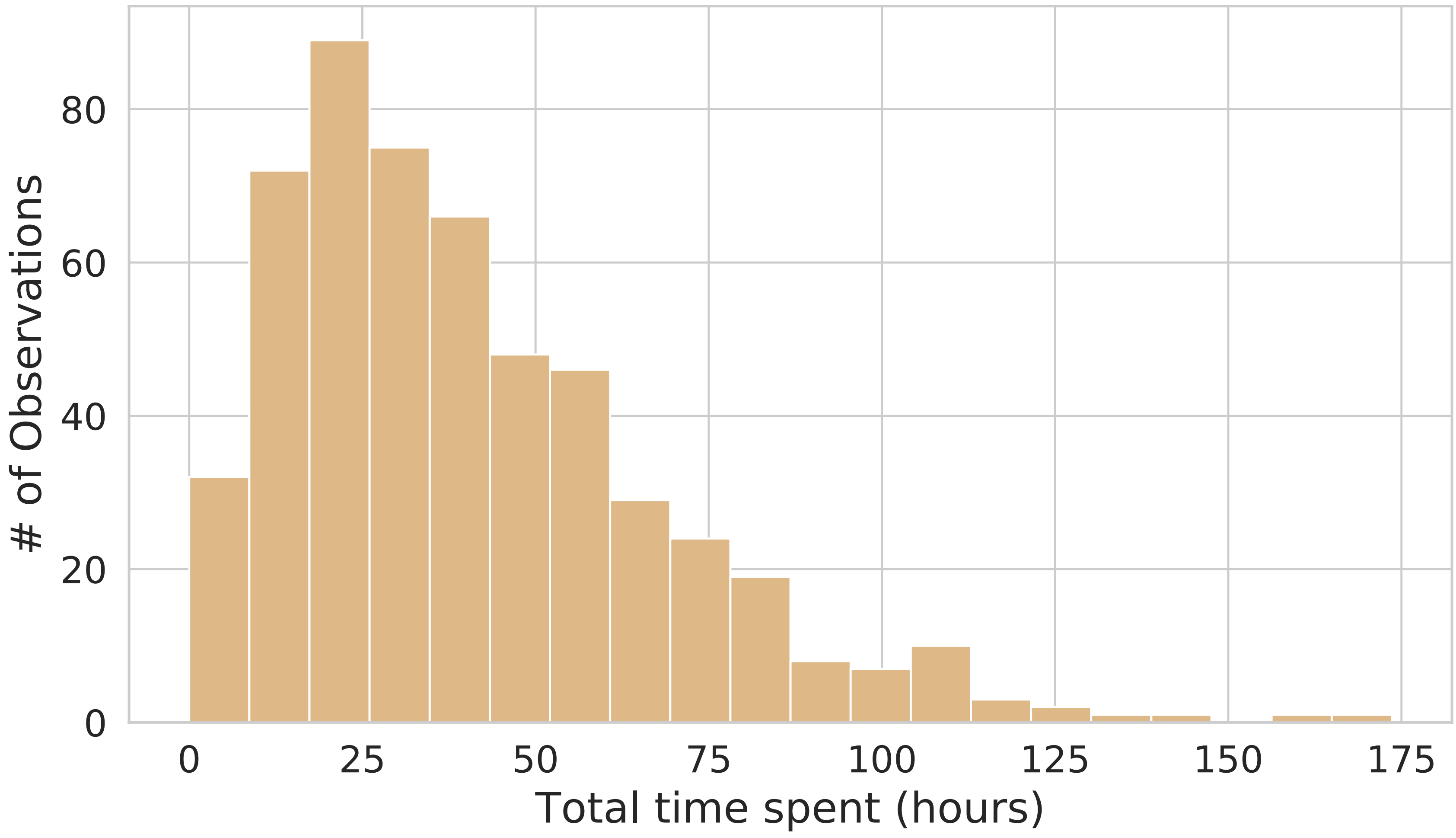


work was done farther before the deadline

work was done closer to the deadline

Total time spent on the project

Measured by adding up the lengths of individual work sessions



No significant relationship between total time spent and solution edit mean time

Earlier test edit median times were associated with more time spent on projects

Did students get better at programming over the semester?

Pairwise differences in project scores by Assignment

Assignment Pair (Left – Right)		Difference
Project 1	Project 2	0.14
Project 3	Project 2	0.19
Project 3	Project 4	0.11

There are significant differences in score means, but scores did not monotonically increase from Project 1–Project 4.

Incremental Testing—Process-Based Measurements

Metric	Correctness		Code Coverage	
	Regression estimate	*p*	Regression estimate	*p*
Testing per-Session	0.30	0.005 *	0.12	0.008 *
Testing per-Session per-Method	--	0.10	0.09	0.002 *
Sequence of testing	--	0.62	–0.06	0.02 *

Incremental Testing—All Measurements

Metric	Correctness		Code Coverage	
	Regression estimate	*p*	Regression estimate	*p*
Testing	0.30	< 0.001 *	0.23	< 0.001 *
Testing per-Method	--	0.12	--	0.41
Testing per-Session	--	0.83	--	0.97 *
Testing per-Session, per-Method	--	0.97	0.08	0.01 *
Sequence of testing	--	0.74	-0.06	0.03 *