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”
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.
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

Feedback

<table>
<thead>
<tr>
<th>Time</th>
<th>Correctness: 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code coverage: 89%</td>
</tr>
</tbody>
</table>
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

| Work Session | ... | ... | Time → |

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

work was done farther before the deadline

work was done closer to the deadline
Mean edit time is September 8
(6 days before the deadline)
Validating the Early/Often Index

No readily available oracle to help measure accuracy.

<table>
<thead>
<tr>
<th>Interviews with students</th>
<th>$n = 7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual inspection of Git histories</td>
<td>$n = 12$</td>
</tr>
</tbody>
</table>

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
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

Time →

Feedback

Correctness: 100%
Code coverage: 89%

Procrastination

Balance/sequence of testing
Motivating Example from Fall 2016

Student A

Student B

Lines changed

Work Session #

Lines changed

Work Session #
Proposed Metrics of Testing Effort

Synthetic example: sequence of developer activity

<table>
<thead>
<tr>
<th>Solution code</th>
<th>Test code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Method A</td>
<td>Method A</td>
</tr>
<tr>
<td>Method B</td>
<td>Method B</td>
</tr>
<tr>
<td>Method C</td>
<td>Method C</td>
</tr>
<tr>
<td>Any method</td>
<td>Any method</td>
</tr>
</tbody>
</table>
Proposed Metrics of Testing Effort

Synthetic example: sequence of developer activity

<table>
<thead>
<tr>
<th>Solution code</th>
<th>Test code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method A</td>
<td>Method C</td>
</tr>
<tr>
<td>Method B</td>
<td>Method B</td>
</tr>
<tr>
<td>Method C</td>
<td>Method B</td>
</tr>
<tr>
<td>Any method</td>
<td>Any method</td>
</tr>
</tbody>
</table>

Project-wide Overall Testing Effort

\[
\frac{T}{S + T}
\]
Proposed Metrics of Testing Effort

Synthetic example: sequence of developer activity

Project-wide Overall Testing Effort

Project-wide per-Session Testing Effort

Solution code
- Orange: Method A
- Green: Method B
- Blue: Method C
- Black: Any method

Test code
- Orange: Method A
- Green: Method B
- Blue: Method C
- Black: Any method

\[
\text{median} \left\{ \frac{T_s}{S_s + T_s} \right\}
\]
Proposed Metrics of Testing Effort

Synthetic example: sequence of developer activity

Project-wide Overall Testing Effort

\[ \frac{T}{S + T} \]

Project-wide per-Session Testing Effort

\[ \text{median} \left\{ \frac{T_s}{S_s + T_s} \right\} \]

Method-specific Overall Testing Effort

\[ \text{median} \left\{ \frac{T_m}{S_m + T_m} \right\} \]
Proposed Metrics of Testing Effort

Synthetic example: sequence of developer activity

Project-wide Overall Testing Effort
\[ \frac{T}{S + T} \]

Project-wide per-Session Testing Effort
\[ \text{median} \left\{ \frac{T_s}{S_s + T_s} \right\} \]

Method-specific Overall Testing Effort
\[ \text{median} \left\{ \frac{T_m}{S_m + T_m} \right\} \]

Method-specific per-Session Testing Effort
\[ \text{median} \left\{ \text{median} \left\{ \frac{T_s}{S_s + T_s} \right\} \right\} \]
Proposed Metrics of Testing Effort

Synthetic example: sequence of developer activity

Project-wide Overall Testing Effort
\[ T \]
\[ \frac{S + T}{\text{median}} \]

Project-wide per-Session Testing Effort
\[ \text{median} \left\{ \frac{T_s}{S_s + T_s} \right\} \]

Method-specific Overall Testing Effort
\[ \text{median} \left\{ \frac{T_m}{S_m + T_m} \right\} \]

Method-specific per-Session Testing Effort
\[ \text{median} \left\{ \text{median} \left\{ \frac{T_s}{S_s + T_s} \right\}_m \right\} \]

Method-specific Overall Sequence of Testing Effort
\[ \text{median} \left\{ \frac{T_{\text{before}}}{T_{\text{before}} + T_{\text{after}}} \right\} \]

<table>
<thead>
<tr>
<th>Solution code</th>
<th>Test code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method A</td>
<td>Method A</td>
</tr>
<tr>
<td>Method B</td>
<td>Method B</td>
</tr>
<tr>
<td>Method C</td>
<td>Method C</td>
</tr>
<tr>
<td>Any method</td>
<td>Any method</td>
</tr>
</tbody>
</table>

- Method is “finalised”
Data Collection – Automatically collected Git snapshots

- 400+ project implementations

**Edit Event**
- Type: Edit
- Time: 1477672862
- Snapshot Id: 23479b3

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change in method <strong>insertFront</strong></td>
<td>+5</td>
<td>12:41:02</td>
</tr>
<tr>
<td>Change in method <strong>getSize</strong></td>
<td>+1</td>
<td>12:41:02</td>
</tr>
<tr>
<td>Change in test for <strong>insertFront</strong></td>
<td>+3</td>
<td>12:41:02</td>
</tr>
</tbody>
</table>
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

Work Session

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)

Student A

Student B

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

<table>
<thead>
<tr>
<th>Overall testing effort</th>
<th>Per-session testing effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>Correctness</td>
</tr>
<tr>
<td>Code Coverage</td>
<td>Code Coverage</td>
</tr>
</tbody>
</table>

### 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

Time →

Work Session

Solution

Tests

Method A

Tests

Feedback

Correctness: 100%

Code coverage: 89%

Procrastination

Balance/sequence of testing

Thoroughness of testing

Tests

Method B

Method C
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
class SomeClass {
  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

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

Mutant Programs

```java
... if (num1 < num2) {
    ...
} else {
    ...
}
```

```java
... if (num1 >= 0) {
    ...
} else {
    ...
}
```

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

```java
... if (true) {
    ...
} else {
    ...
}
```
Mutation Testing

- Strong test adequacy criterion
- Allows incremental feedback
- Prohibitively high computational cost
# Context

<table>
<thead>
<tr>
<th>Course</th>
<th># Submissions</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS 2</td>
<td>1,019</td>
<td>30 s</td>
</tr>
<tr>
<td>CS 3</td>
<td>370</td>
<td>5.04 min</td>
</tr>
</tbody>
</table>

1 submission / 5.5 seconds
Peak: 1 submission / 1.5 seconds
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?

Only apply certain kinds of mutations

Replace conditionals with Boolean literals

\[ a > b \rightarrow true, \ a > b \rightarrow false \]

Replace arithmetic expressions with its operands

\[ a + b \rightarrow a, \ a + b \rightarrow b \]

Can we do this fast enough for incremental feedback?
### Mutation by Deletion

*(Offutt et al. 2014)*

<table>
<thead>
<tr>
<th>Mutator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete conditional expressions</td>
<td>$a &gt; b \rightarrow \text{true}$</td>
</tr>
<tr>
<td>Delete arithmetic operators</td>
<td>$a + b \rightarrow a$</td>
</tr>
<tr>
<td>Delete non-void method calls</td>
<td>$\text{getString()} \rightarrow \text{null}$</td>
</tr>
<tr>
<td></td>
<td>$\text{getInt()} \rightarrow 0$</td>
</tr>
<tr>
<td>Delete void method calls</td>
<td>$\text{performAction()} \rightarrow$</td>
</tr>
<tr>
<td>Delete assignments to member variables</td>
<td>$\text{this.age} = 25 \rightarrow \text{this.age} = 0$</td>
</tr>
<tr>
<td>Delete constructor calls</td>
<td>$\text{new String()} \rightarrow \text{null}$</td>
</tr>
</tbody>
</table>
### Context

<table>
<thead>
<tr>
<th>Course</th>
<th># Submissions</th>
<th>All ~30 Mutators</th>
<th>6 Deletion Mutators</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS 2</td>
<td>1,019</td>
<td>30 seconds</td>
<td>4.75 seconds</td>
<td></td>
</tr>
<tr>
<td>CS 3</td>
<td>370</td>
<td>5.04 minutes</td>
<td>1.11 minutes</td>
<td></td>
</tr>
</tbody>
</table>

1 submission / 5.5 seconds

Peak: 1 submission / 1.5 seconds

10% of the cost

90% of the effectiveness
## Incremental subsets of mutation operators

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

<table>
<thead>
<tr>
<th>Mutators Added</th>
<th># of Mutants Produced (per KSLoC)</th>
<th>Median</th>
<th>% of All Mutants</th>
<th>Effectiveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoveConditionals</td>
<td>102</td>
<td></td>
<td>7.04%</td>
<td>78%</td>
</tr>
<tr>
<td>ArithmeticOperatorDeletion</td>
<td>140 (+38)</td>
<td>102</td>
<td>9.67%</td>
<td>88%</td>
</tr>
<tr>
<td>NonVoidMethodCalls</td>
<td>236 (+96)</td>
<td>140</td>
<td>16.30%</td>
<td>91%</td>
</tr>
<tr>
<td>VoidMethodCalls</td>
<td>240 (+4)</td>
<td>236</td>
<td>18.57%</td>
<td>91%</td>
</tr>
<tr>
<td>MemberVariables</td>
<td>271 (+91)</td>
<td>240</td>
<td>18.72%</td>
<td>91%</td>
</tr>
<tr>
<td>ConstructorCalls</td>
<td>283 (+13)</td>
<td>271</td>
<td>19.54%</td>
<td>91%</td>
</tr>
</tbody>
</table>
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

- Group 1
- Group 2
- Group 3
- Group 4
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

<table>
<thead>
<tr>
<th>Operator Added</th>
<th># of Mutants Produced</th>
<th>% of All Mutants</th>
<th>Adjusted $R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoveConditionals</td>
<td>102</td>
<td>7.04%</td>
<td>0.78</td>
</tr>
<tr>
<td>ArithmeticOperatorDeletion</td>
<td>140</td>
<td>9.67%</td>
<td>0.88</td>
</tr>
<tr>
<td>NonVoidMethodCalls</td>
<td>236</td>
<td>16.30%</td>
<td>0.91</td>
</tr>
<tr>
<td>VoidMethodCalls</td>
<td>240</td>
<td>16.57%</td>
<td>0.92</td>
</tr>
<tr>
<td>MemberVariables</td>
<td>271</td>
<td>18.72%</td>
<td>0.92</td>
</tr>
<tr>
<td>ConstructorCalls</td>
<td>283</td>
<td>19.54%</td>
<td>0.92</td>
</tr>
</tbody>
</table>
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

RemoveConditionals, ArithmeticOperatorDeletion, NonVoidMethodCalls

Submission Groups
- SG1
- SG2
- SG3
- SG4

$R^2 = 0.9$

Accuracy (Adj. $R^2$ predicting FULL coverage)

1-op Subset

2-op Subset

3-op Subset
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
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.
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?

<table>
<thead>
<tr>
<th>Assignment Pair (Left – Right)</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project 1 - Project 2</td>
<td>0.14</td>
</tr>
<tr>
<td>Project 3 - Project 2</td>
<td>0.19</td>
</tr>
<tr>
<td>Project 3 - Project 4</td>
<td>0.11</td>
</tr>
</tbody>
</table>

There are significant differences in score means, but scores did not monotonically increase from Project 1–Project 4.
## Incremental Testing—Process-Based Measurements

<table>
<thead>
<tr>
<th>Metric</th>
<th>Correctness</th>
<th></th>
<th>Code Coverage</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Regression estimate</td>
<td>$p$</td>
<td>Regression estimate</td>
<td>$p$</td>
</tr>
<tr>
<td>Testing per-Session</td>
<td>0.30</td>
<td>0.005 *</td>
<td>0.12</td>
<td>0.008 *</td>
</tr>
<tr>
<td>Testing per-Session per-Method</td>
<td>--</td>
<td>0.10</td>
<td>0.09</td>
<td>0.002 *</td>
</tr>
<tr>
<td>Sequence of testing</td>
<td>--</td>
<td>0.62</td>
<td>–0.06</td>
<td>0.02 *</td>
</tr>
</tbody>
</table>
## Incremental Testing—All Measurements

<table>
<thead>
<tr>
<th>Metric</th>
<th>Correctness</th>
<th>Code Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Regression estimate</td>
<td>$p$</td>
</tr>
<tr>
<td>Testing</td>
<td>0.30</td>
<td>&lt; 0.001 *</td>
</tr>
<tr>
<td>Testing per-Method</td>
<td>--</td>
<td>0.12</td>
</tr>
<tr>
<td>Testing per-Session</td>
<td>--</td>
<td>0.83</td>
</tr>
<tr>
<td>Testing per-Session, per-Method</td>
<td>--</td>
<td>0.97</td>
</tr>
<tr>
<td>Sequence of testing</td>
<td>--</td>
<td>0.74</td>
</tr>
</tbody>
</table>