Logistics, Affordances, and Evaluation of Build Programming

A Code Reading Instructional Strategy

Amanpreet Kapoor, Tianwei Xie, Leon Kwan, Dr. Christina Gardner-McCune
University of Florida

Background

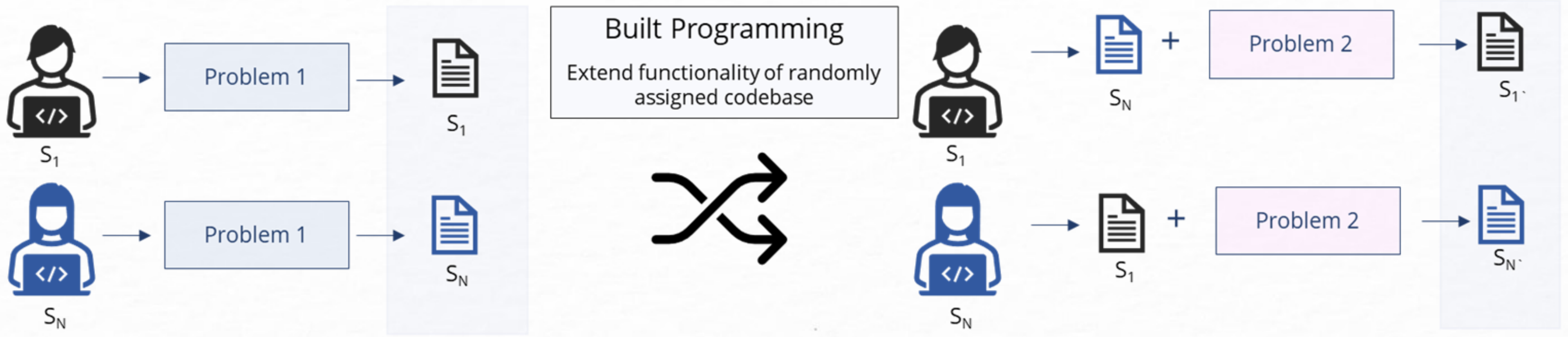
- Difficult to transition from university to industry for students
  - high expectations
    - unfamiliar, large codebases
    - code reviews
    - extend functionality
  - lack of experiences
    - build on top of other’s program
- 60% of time spent on code reading in the industry (Mistrik et al., 2021)

Prior work on Code Reading

- **Code deconstruction** (Griffin, 2016)
  - encompasses the process of reading, tracing, and debugging code
- **Explain in plain english activities** (Corney et al., 2014)
- **Reading aloud** (Swidan & Hermans, 2019)
  - learner reads out loud their code
- **Pair programming** (Ciolkowski & Schlemmer, 2002 and Kuppuswami & Vivekanandan, 2004)
  - focused on collaboration rather than code reading
  - our focus: individual contributions and code reading
- **Pair programming + pair trading** (DeClue, 2003)
  - work motivated by the collaborative aspects of pair programming
  - our work promotes code reading by using context as a scaffold
- **Remix approaches** (Richardson et al., 2011 and Sanchez, 2017)
  - used to introduce students to coding and reduces student anxiety
  - students inspect and edit existing projects to see underlying structures of code and reverse engineer solution
  - designed for scaffolding learning of CS concepts rather than code reading

What is Build Programming

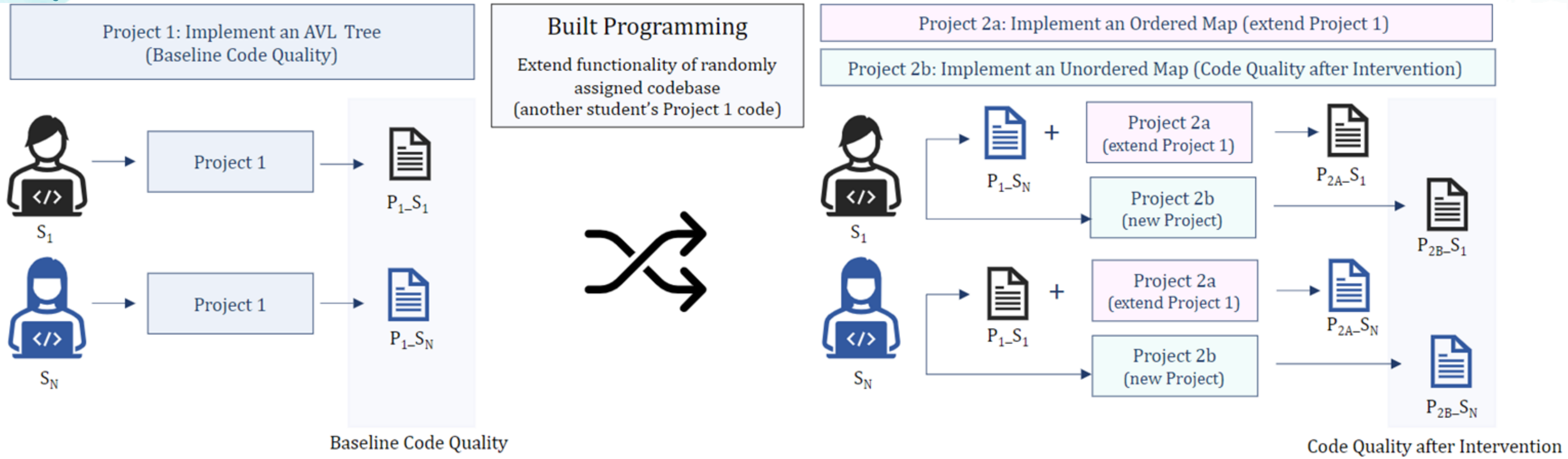
- Instructional strategy to promote code reading and extension.



Context

- undergraduate Data Structures and Algorithms course in Fall 2021
- large public university in the United States
- language of instruction is C++
- utilized Build Programming in the two independent projects
- Project 1:
    - a non-templated self-balancing binary tree data structure called AVL tree in C++
- Project 2:
    - Implement the tree-based or ordered map reusing a peer's tree implementation
    - Implement the hash-table based or unordered map
    - Compare the performance of ordered and unordered map
    - Review the code quality of the assigned codebase
    - Approval from the author of the assigned codebase

Context



Project 1: Implement an AVL Tree (Baseline Code Quality)
Built Programming
Extend functionality of randomly assigned codebase (another student's Project 1 code)
Project 2a: Implement an Ordered Map (extend Project 1)
Project 2b: Implement an Unordered Map (Code Quality after Intervention)
S1
Project 1
P1_S1
SN
Project 1
P1_SN
Baseline Code Quality
S1
P1_SN
+
Project 2a (extend Project 1)
P2A_S1
Project 2b (new Project)
P2B_S1
SN
P1_S1
+
Project 2a (extend Project 1)
P2A_SN
Project 2b (new Project)
P2B_SN
Code Quality after Intervention

Build Programming

Participants and Assignment Statistics

- 206 undergraduate students consented to share data from projects and filled out a post-survey (Class size: 387)
- Project 1: AVL Tree
  - Average Lines of code: 586 lines (Min = 207, Max = 2182, SD = 194)
  - Average Grade: 81/100 (Min = 16, Max = 100, SD = 14)
- Project 2b: Implement unordered map
  - Average Lines of code: 452 lines (Min = 114, Max = 1502, SD = 179)
  - Average Grade (Project 2): 92/100 (Min = 0, Max = 100, SD = 15)

Analysis

- RQ.1. What are the student perceptions of the affordances of *Build Programming* instructional strategy? How did they receive the activity?

- Student reflections from two open ended survey questions were coded using inductive content analysis following a constant comparison technique.
  - What did you learn from this activity?
  - Should this project be continued in the future? Any other comments?

Build Programming

Findings: Affordances

- Promoting and scaffolding code reading (40% of 104 responses, n = 42)
  - S214: "*the part on working with someone else's code is simple enough that the focus is entirely on understanding the code without the stress of figuring out the implementation, which I think is a really good way to introduce it*"

- Learning computing concepts (32% of 104 responses, n = 33)
  - S88: "*I learned that you could do* ***post/pre/in order traversals with stacks****. This is much more efficient than the way I employed*"
  - S107: "*the peer code provided to me [for the assignment] taught me a lot of new concepts that I had not touched on, such as* ***shared pointers***"

Findings: Affordances

- Importance of code quality (25% of 104 responses, n = 26)
    - S125: "*I learned that **documenting your code is extremely useful** to better understand it and work with it faster. In addition, I learned that documenting your code is really important so that **others can understand it to be able to work with it***"
    - S125, "***I will document and add more comments** everywhere as if I am explaining it to another person. I will add lines of comments in between my functions **for better readability***"

Findings: Affordances

- Alternate ways to solve a problem (9% of 104 responses, n = 9)
  - S104: "*I* ***got to see another way of going about the code*** *that I had already done. It was kind of interesting comparing her code to mine, and seeing how I had it more optimized in places and vise versa*"

- Authentic assessments (5% of 104 responses, n = 5)
  - S153: "*I* ***learned what it will be like in the real industry****. I am only used to working on my own coding projects in which I know and am familiar with. A lot of my time in the future will be spent reading code to understand its functionality rather than just writing it*"

Findings: Reception

- **91%** of the 119 student responses suggested: continue *Build Programming* as-is or with the logistical change of assigning working codebases

  - S1: *"I think so [should we continue Build Programming?]. A lot of my friends from industry in CS (FAANG., AMEX, BOA) have told me* ***its good practice to learn how to use different codebases to adapt to your own solution****. A lot of code is already written, sometimes its up to you to understand it, identify problems, and fix it"*

  - S49: *"I believe the project should absolutely be continued in the future. Writing data structures from a small bit of skeleton code is very good practice. [...] A critical point though, I do not believe it is a good idea to randomly assign codebases to people. [...]. The instruction team may wish to take a few known-to-be-functional codebases [...] and only pull from that pool in the future"*

Findings: Reception

- **9%** of the 119 student responses suggested that we discontinued *Build Programming* due to an unfavorable experience
    - These students were frustrated because they received imperfect codebases that did not pass all tests (>80% to < 100% tests on functionality)

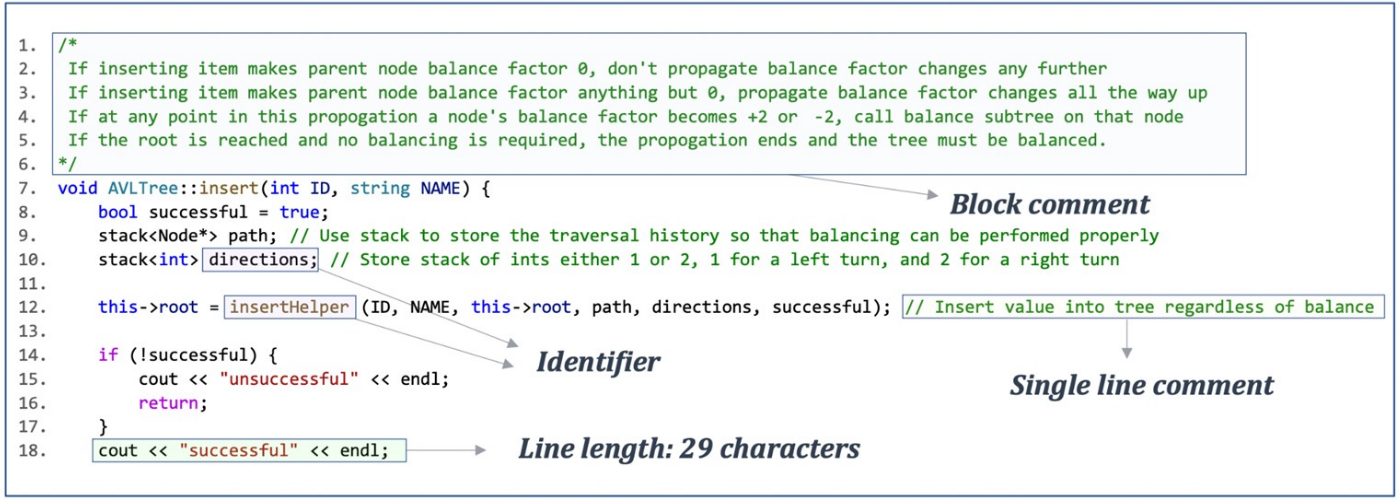
    - **S38:** *"I do not believe this project should be continued in the future. It is too dependent on the competence of others"*

Research Questions

- RQ.2. How effective is the *Build Programming* instructional strategy in improving a student's code quality as measured through readability metrics such as comment ratio, average line lengths, and average identifiers per line?

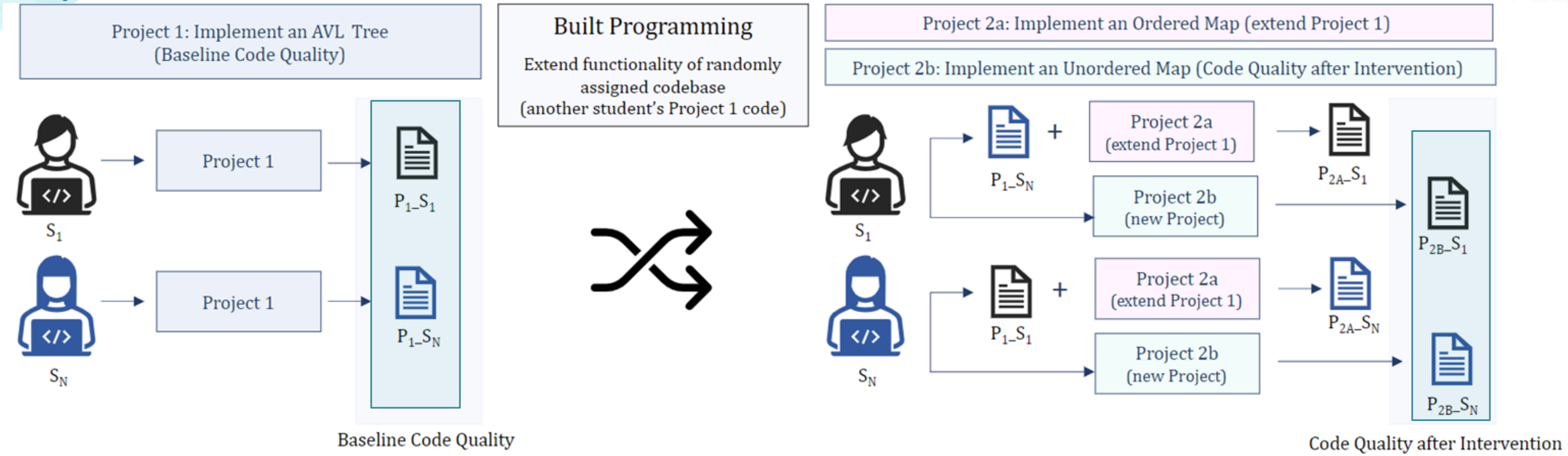
Literature on Code Quality

- Based on Buse and Weimer (2010), the code metrics correlated with readability
  - Negative correlation: Identifiers per line, average line length
  - Positive correlation: ratio of comments to lines of code



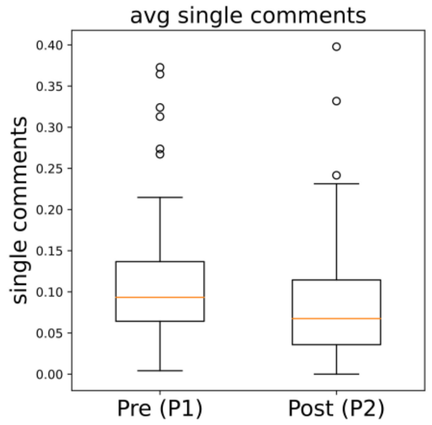
```
/*
 If inserting item makes parent node balance factor 0, don't propagate balance factor changes any further
 If inserting item makes parent node balance factor anything but 0, propagate balance factor changes all the way up
 If at any point in this propogation a node's balance factor becomes +2 or  -2, call balance subtree on that node
 If the root is reached and no balancing is required, the propogation ends and the tree must be balanced.
*/
void AVLTree::insert(int ID, string NAME) {
    bool successful = true;
    stack<Node*> path; // Use stack to store the traversal history so that balancing can be performed properly
    stack<int> directions; // Store stack of ints either 1 or 2, 1 for a left turn, and 2 for a right turn

    this->root = insertHelper(ID, NAME, this->root, path, directions, successful); // Insert value into tree regardless of balance

    if (!successful) {
        cout << "unsuccessful" << endl;
        return;
    }
    cout << "successful" << endl;
```

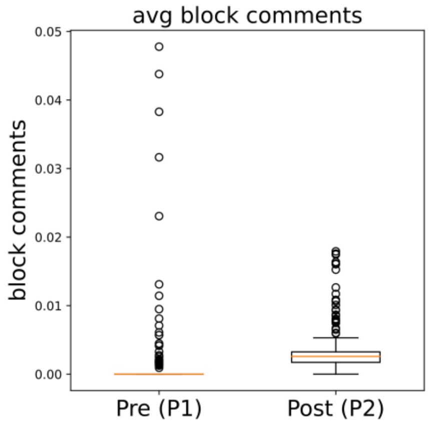
Context

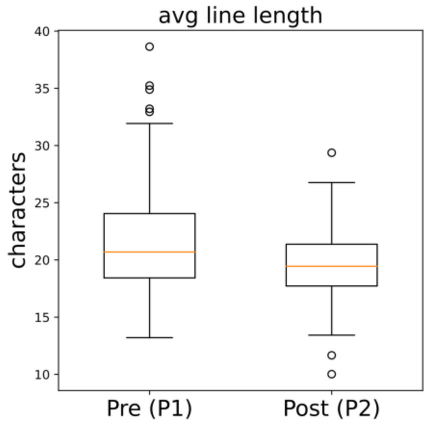


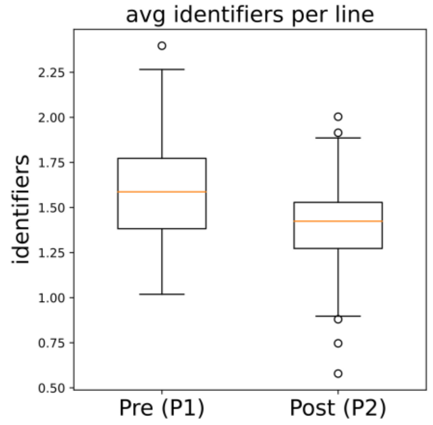
Project 1: Implement an AVL Tree (Baseline Code Quality)
Built Programming
Extend functionality of randomly assigned codebase (another student's Project 1 code)
Project 2a: Implement an Ordered Map (extend Project 1)
Project 2b: Implement an Unordered Map (Code Quality after Intervention)
S_1
S_N
Project 1
Project 1
$P_1\_S_1$
$P_1\_S_N$
Baseline Code Quality
$P_1\_S_N$
+
Project 2a (extend Project 1)
Project 2b (new Project)
$P_{2A}\_S_1$
$P_{2B}\_S_1$
$P_1\_S_1$
+
Project 2a (extend Project 1)
Project 2b (new Project)
$P_{2A}\_S_N$
$P_{2B}\_S_N$
Code Quality after Intervention

Findings









Wilcoxon Signed Rank Test: *p*<0.001

24% decrease

- Unexpected, possibly due to difference in rubric requirements for comments

Wilcoxon Signed Rank Test: *p*<0.001

50% increase in block comments

Wilcoxon Signed Rank Test: *p*<0.001

7% decrease in average line length

Paired two-tailed t-test: *p*<0.001

12.5% decrease in identifiers per line

Discussion and Conclusion

- Build Programming:
  - promoted and scaffolded code reading
  - ↑ exposure to alternate ways to solving problems
  - ↑ knowledge of computing constructs
  - ↑ awareness of the importance of code quality through an authentic experiential ("show") learning approach rather than a "tell" approach

- 91% students endorsed build programming

- Some assigned codebases had unresolved bugs
  - Future improvement by assigning functional codebases

- Recommend other instructors to incorporate Build Programming in their courses

Questions

kapooramanpreet@ufl.edu

Build Programming