Crafting Strong Identifier Names

Christian Newman & SCANL Lab

SCANL
Source Code Analysis And Natural Language Laboratory

SCANL Lab

We study the latent connection between source code behavior and the natural language elements used to describe that behavior

Lab Founders: Christian D. Newman, Reem S. Alsuhaibani, Michael J. Decker, and Emily Hill

Members: Anthony Peruma, Tejal Vishnoi, Satyajit Mohapatra, Shimon Johnson, and Dishant Kaushik

Collaborators: Mohamed Wiem Mkaouer, Marcos Zampieri, Timothy J. Sheldon

For more information on our tools, datasets, and research:

https://scanl.org/

Identifiers

```
boolean isArraysAreEquals()
void getVobs()
List<int> arru8NumberList

String to_string()
void sort()
long query_Timeout_In_Ms

String[] method_Name_Prefixes
bool is_First_frame
List<int> all_invocation_matchers
```

Are they high-quality?

- What makes an identifier high-quality?
  - Quality based on group naming conventions
  - Quality based on education
  - Quality based on **who is doing the comprehending**
    - Domain knowledge, programming knowledge, general experience
- Quality is at least partially subjective

- There are no objective quality metrics for identifiers

But we've all seen it

- "pclHldr doesn't make any sense, what is pcl??"
- "Why does this use str, and it's not a string?"
- "Ugh hungarian notation is the worst"
- "Why is this identifier named 'a'???"

- We know a bad identifier name when we see it.
  - Sometimes it's bad enough to approach being objectively bad

Metrics would be v.helpful

- Program comprehension is critically important
  - Identifiers **must be understood** before any other coding activity
- All developers rely on identifiers
  - Students and novices
  - Onboarding new developers
  - Experienced developers
  - Senior developers and architects
- Everyone **relies on program comprehension** to learn, do their job, and express code behavior

What makes this hard?

- We (developers) invented our own sub-language
  - Identifiers follow a **non-standard human language grammar**
- And to top it off? It's ~70% of the code
  - We have a sub-language that was not designed, but evolved

- We didn't design this language, but we **did create it**
  - And evolve it to a changing environment, just like software

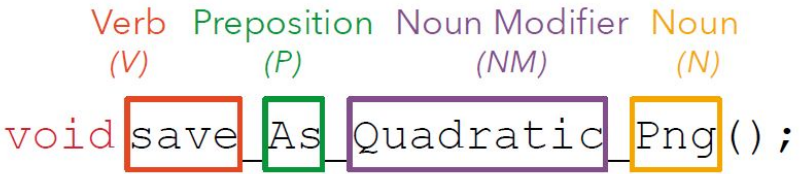
What needs to be done?

- Study the sub-language used by software developers
  - We need a full, written understanding of its structure and rules
- Formally **measure and describe** identifier quality
  - Combine measurements to create an **explainable** metric
- Use our understanding of both to support identifier name creation, appraisal, and maintenance

Grammar Patterns

- Split identifier into its constituent words, apply a **specialized** part of speech tagger to the words as if they were a sentence

NM NM N



V P NM N

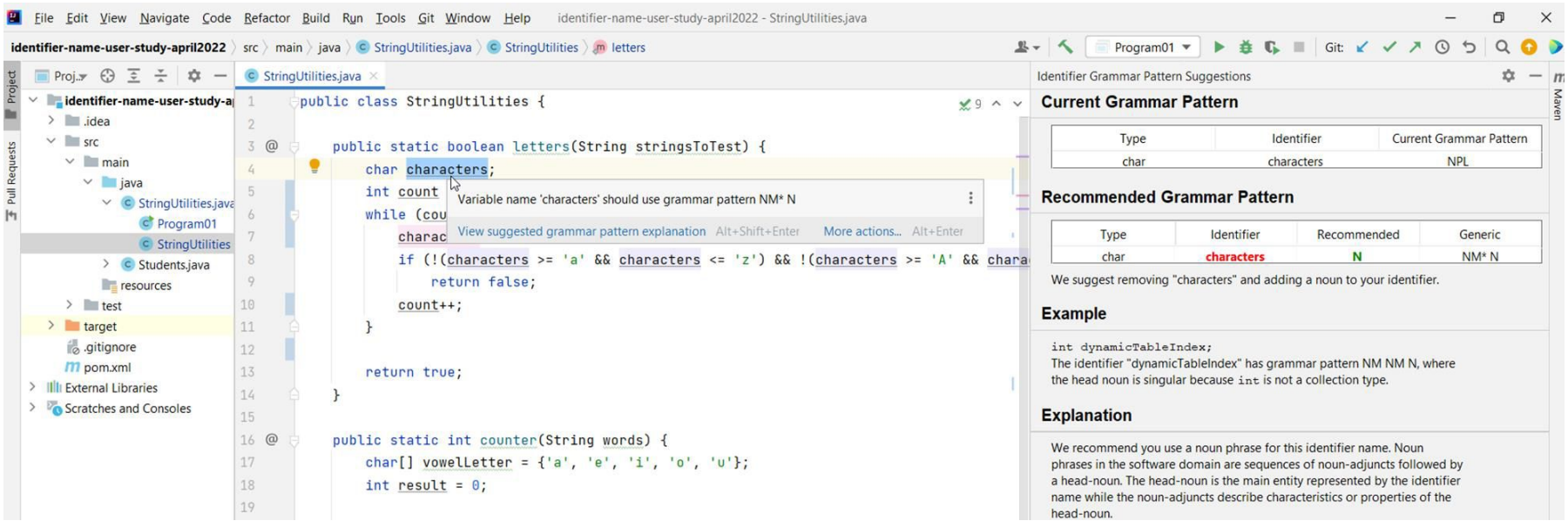
Name structure catalogue

- We created a catalogue of identifier naming structures:
  - https://github.com/SCANL/identifier_name_structure_catalogue

- This a glimpse of the sub-language we have been discussing

- *These are some of the basic natural language phrasal structures that we, as developers, have created*

A Tool for Developers



Future Work

- Goal 1: Fully explore the diversity of grammar patterns
  - And, thus, gain a formal understanding of the language of software
- Goal 2: Create data-driven naming guidelines
  - A set of **measurements** that are used to create a **naming metric**
  - Requires many, many **human subjects** trials
- Goal 3: Create a framework for optimizing names
  - Prioritize ease of comprehension **for the reader**
  - Prioritize **explainability**
  - Approach **optimal comprehensibility**
- Goal 4: Educating developers at all levels

Questions?

- You can find more at our webpage!
  - https://www.scanl.org/
- The catalog is on GitHub and is public data
  - https://github.com/SCANL/identifier_name_structure_catalogue
- IntelliJ prototype is also open
  - https://github.com/SCANL/IDEAL
- Contact:
  - cdnvse@rit.edu

Our work so far

- One of our more recent contributions is in the idea of ***grammar patterns***
  - A way to ***measure, express, and comprehend*** identifier meaning and characteristics using part-of-speech tag sequences

- Allow us to study high-level identifier naming patterns and semantics

- Provide *insight* into the language of software devs

Most importantly

We need a formal understanding of how identifier characteristics influence comprehension.

This has to take into account varying levels of experience

Doing so will allow us to provably support comprehension for anyone (even machines) despite experience level.

How does this help?

- Grammar patterns highlight the semantics identifiers
  - We can tie these semantics to a coding context, such as event-driven code, or code that computes type conversions
- Grammar patterns allow us to explain
  - We know what the pattern means, and can explain how it is typically used by developers.
  - When we recommend a pattern, we recommend it ***with a reason***; this reasoning can be used to approve, or reject, a recommendation
- Grammar patterns are a measurable structure
  - We can measure the effectiveness of different patterns for comprehension tasks

Why are they important?

- Identifiers must be understood before **any other** coding activity
  - Fixing bugs, adding features, cleaning, refactoring, patching
- Difficult to measure the influence of identifiers on comprehension
  - For humans OR for automated tools (e.g., AI)
- Many techniques use identifier information
  - Identifiers are a **threat to validity**
    - Can we trust techniques/research that leverage identifiers if we can't reason about (measure) the quality of identifiers they will be using?
- Problem can't be solved using AI
  - Developers may just assume that predictions are always correct
    - Exacerbated because the AI can't explain itself and we cannot measure identifier quality

How does this help?

- We need a way to deal with the semantics (i.e., meaning) of the identifier within its coding context
- Grammar patterns improve the semantic quality
  - We can use them to understand **both meaning and structure**
  - We can explain **why** we are recommending a grammar pattern
  - We can measure comprehension improvements using grammar patterns, which is currently very hard to do without them
  - They help us understand **correctness** of the name
- Linters only deal with lexical information
- Purely AI approaches are not explainable

Identifier Naming

- Identifier names
  - Function, class, function-local, parameter, attribute, global variables
  - On average, 70% of the characters in code are in identifier names
- Identifier naming is a largely unresolved problem
  - No way to objectively measure quality, very little understanding of how identifiers and their characteristics influence comprehension
  - “Bad” names slow comprehension, cause developers to introduce bugs or other problems into the code due to misunderstanding
- Identifiers have a unique phrasal structure
  - They don’t follow the same rules as human language
  - Off-the-shelf NLP techniques underperform on identifiers