Robolint: automated analysis of liquid handler method code to flag programming errors and align to team practices

Andrew Hurder[1]*, Mark R Southern[1], Niket Karode[1], Sura H Hadi[1], Eli J Fine[1‡]

[1]Department of Automation & Process Engineering, National Resilience, San Diego CA
*presenting author
‡ corresponding author: eli.fine@resilience.com

Overview

- Writing method code to control laboratory robotics is often a manual and error-prone activity
- Lab Automation Engineering lags Software Engineering in using automated tools to streamline programming

- We created a code linting tool to analyze method code to automatically detect issues while programming
- Robolint was developed as a plugin to the Python Pylint framework and is run a pre-commit hook by Git
- Robolint currently supports analyzing MethodManager4 files (Dynamic Devices), but can be extended to process any text-readable method code

- Robolint detects issues with inconsistent style and syntax thereby making it easier to understand the code
- Robolint detects potential logical errors where the code may be directing the liquid handler to perform an action inconsistent with the programmer's intent
- Automated linting of method code reduces time programmers and reviewers spend checking code manually and reduces the chance of mistakes being deployed in the lab

Introduction

As lab automation engineers, our role is to automate the work of scientists, but the way we go about our work is often frustratingly manual. The software engineering industry has developed various tools and practices that automate daily programming tasks. One essential tool is a linter, which automatically analyzes code as it is being programmed before it is merged with others' work and before deployment. While method simulators are provided within most liquid handler control software, their scope is limited to detecting issues that are not physically possible for the robot to perform. The role of a linter is to help detect logical errors within the code (accidentally programming the robot to do something other than what was intended) and to enhance the readability of the code.

Mistakes still slip through despite time-consuming, detailed code examination by skilled automation engineers. Humans are fallible, which is why we automate many lab activities, and we see this when reviewing liquid handler code. Avoiding costly mistakes in the lab by utilizing automated linting to increase code quality is long overdue in our industry.

Methods

- Implemented to analyze files generated by MethodManager4 v1.4.8425, using Python v3.9.13 PyLint v2.15.10, pre-commit v2.21.0, Git v2.30.2
- Tested on Windows 11, Windows Server 2019, Linux (Debian 10)
- Currently hosted on internal company servers but working on transitioning to hosting as an open-source repository to accept community contributions for new linting rules and method languages to support. https://github.com/resilience-bio

Configuring as a Git hook

```yaml
minimum_pre_commit_version: 2.21.0
default_install_hook_types: [pre-commit]
repos:
-   repo: local
    hooks:
    -   id: robolint
        name: robolint
        entry: python -m hooks.robolint.run
        language: system
        stages: [commit]
        require_serial: true
        files: '.*\.met$'
        exclude: '.*/Methods/Test.*'
        verbose: true
        args:
        - --rcfile=robolintrc
```

Figure 1. Pre-commit hook configuration. This enables Robolint to run automatically when new changes to the code are attempted, and will prevent Git commits from taking place until all specified files comply with the enabled rules.

Robolint Configuration

```
loop-start-index=0

# Maximum number of lines in a method
max-module-lines=300

# Regular expression matching correct variable names
variable-rgx=^[A-Z][a-z]+(?:[A-Z][a-z]+)*(\_\d+)?$

# Regular expression matching correct labware names
labware-rgx=^(1|2|3|4|6|12|24|48|96|384|1536)( ([[A-Z][A-Z

# Regular expression matching correct method file names
method-file-rgx=(([a-z_][a-z0-9_]*)|([A-Z][a-zA-Z0-9]+))$
```

Figure 2. Example rule configuration. Various settings can be configured in the `robolintrc` file based on team preferences.

Rule Implementation

```python
class LoopIndexChecker(StepChecker):
    """Checks loop steps for not being indexed at the specified value."""

    name = "invalid_loop_start_index"
    msgs = {
        "C9001": (
            "The loop start index was %s, please replace with %s.",
            "invalid-loop-start-index",
            "Identifies when a loop start index doesn't match convention.",
        ),
    }
    options = (
        (
            "loop-start-index",
            {
                "default": 0,
                "type": "int",
                "metavar": "<int>",
                "help": "Value that loop counters should begin with. Typically 0 or 1.",
            },
        ),
    )

    step_type_ids = {BEGIN_LOOP_STEP_ID}

    @override
    def _visit_step(self, step: HH4Step) -> None:
        """Check if step has invalid loop start index."""
        if step.get_parameter("VariableValuevariable") != "":
            # if the loop start index is bound to a variable, don't check further
            return
        actual_loop_start = int(step.get_parameter("VariableValue"))
        expected_loop_start = self.linter.config.loop_start_index
        if actual_loop_start != expected_loop_start:
            self.add_message("invalid-loop-start-index", args=(actual_loop_start, expected_loop_start))
```

Figure 3. Example implementation. By extending the Pylint framework, a new configurable rule can be implemented with very little additional code. The complexities of running the linter, collating and displaying messages, and processing all the files are all managed by leveraging the existing functionality of Pylint.

Results

Overriding a Rule

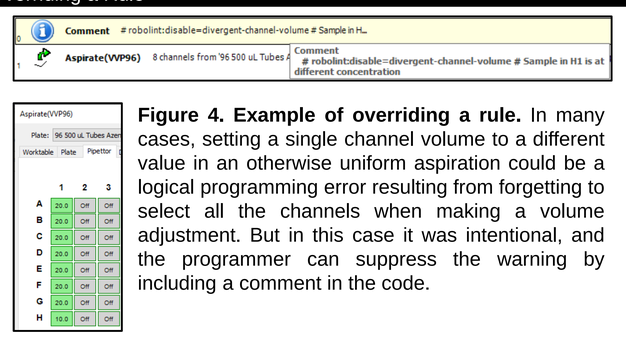

Figure 4. Example of overriding a rule. In many cases, setting a single channel volume to a different value in an otherwise uniform aspiration could be a logical programming error resulting from forgetting to select all the channels when making a volume adjustment. But in this case it was intentional, and the programmer can suppress the warning by including a comment in the code.

Robolint Output

```
************* Module VgTiter_CapsidLysisSetup_Module_1
LM748/Methods/VgTiter_CapsidLysisSetup_Module_1.met:14:0: C9001: The loop start index was 1, please replace with 0. (invalid-loop-start-index)
************* Module VgTiter_PreDilutionDNAseAdd_Module_0
LM748/Methods/VgTiter_PreDilutionDNAseAdd_Module_0.met:35:0: C9001: The loop start index was 1, please replace with 0. (invalid-loop-start-index)
```

Figure 5. Example output. When run, Robolint scans the files, flags violations and provides information on the line number and suggested corrections.

Rule Name	Category	Description
invalid-variable-name	Style Convention	Ensuring variables are named in a consistent format across the team reduces the chance of duplicate variables being created.
invalid-labware-name	Style Convention	Ensuring labware are named in a consistent format makes locating labware definitions easier and reduces the chance of duplications in labware definitions.
invalid-method-name	Style Convention	Ensuring method files are named in a consistent format reduces confusion for operators looking for files to start a run, as well as for programmers needing to edit a file originally created by someone else.
too-many-lines	Style Convention	Used when a single file has too many lines, reducing its readability. Long workflows should be broken into submodules and repeated actions refactored unto reusable subfunctions.
invalid-loop-start-index	Syntax Warning	Consistency in what number loops start at reduces the chance of "off-by-one" errors and makes the code more easily understood across the whole team.
invalid-boolean-value	Syntax Warning	Consistency in how to represent boolean values (1 / 0, y / n, true / false, True / False)
divergent-channel-volume	Logical Error	In systems with independent channels, many workflows involves the channels all moving a different volume of liquid. But many steps simply need to treat all wells the same. However sometimes mistakes in programming can lead to a single channel being left at an old volume when the rest of the channels are changed to a new uniform volume. This circumstance is flagged as a possible error.
missing-initialization-steps	Logical Error	Many teams have a standard set of commands they want each method to begin with (e.g. re-initializing the instrument, confirming that the `production` branch of code rather than `development` is checked out when an Operator is running a method…). This rule automatically checks and enforces that those are included.
hardcoded-aspirate-volume	Logical Error	Many teams prefer that all volumes in a step be bound to variables---rather than hardcoded---so that it's less error-prone to make future adjustments to the method. E.g. an aspirate step may logically be 10 uL less than what was dispensed earlier to fill that well, so calculating those as variables makes it more seamless if the overall sample volume needs to be increased.
no-full-z-retraction-before-travel	Logical Error	When optimizing method speed, a programmer may often disable retracting all the way to Z-max because they've deemed it safe in the current deck layout. However, when updating a method, the deck may change and this movement can get overlooked. By forcing an explicit comment in the code describing reasoning for the original safe motion path, it is easier to spot and take into account when modifying the code to avoid potential crashes.
excess-z-retraction	Logical Error	By default, many liquid handler software suites defaults to retracting to maximum Z height after every command. However, when performing multiple pipetting actions within the same plate the programmer likely only intends the robot to retract to the plate height for more efficient movements.

Table 1. Example Rules

Conclusion

- Static code analysis (linting) can now be easily incorporated into the workflow for programming laboratory automation equipment.
- By building on the Pylint framework, Robolint leverages its extensive existing documentation and rich feature set to facilitate creating new rules and configurations easy for people with limited software engineering backgrounds.
- Robolint can detect a variety of issues while code is being created that can be fixed to avoid potential problems in the lab or during future revisions to the code.
- Robolint is easily configurable to comply with different conventions and best practices adopted by different organizations. Rules can be completely disabled in the configuration file, and any naming rules (e.g., labware, variables) can be customized (using regular expressions).

Resilience.com

References

1. Pylint, https://pylint.readthedocs.io/
2. Pre-commit, https://pre-commit.com/