**Question 2 :**

**PR Review** <https://github.com/canonical/jenkins-job-linter/tree/feature/linter-short-codes-pr-test>

*AI Acknowledgement : I used ChatGPT to beautify, structure my answers and also fix grammatical errors and order/bullet them. I also used GPT to generate some sentences for the XPath part because I felt GPT English was far more accurate and my English although decent, wouldn’t perfectly convey the essence.*

**Informal Comments**

**Firstly, these would be the things I would say to my fellow developer informally. Based on what I think and what I understood from the PR, and what I feel…can be improved. I feel it is important to communicate informally first, because there might be instances where we cannot frame solutions formally, but we still have the idea.**

So, overall, I think the changes look good and add some solid functionality to the Jenkins job linter. You’ve done a great job with adding the ability to filter linters through the --select and --ignore options. It’s definitely a useful feature and makes the tool more flexible for users who want more control. However, one thing I’d suggest is adding some validation for the short codes provided through these options. If someone passes in an invalid short code, we don’t really give them any feedback, so it could be good to warn them or show an error message instead of just silently ignoring it.

For example, if someone enters a short code that doesn’t exist, maybe we could print a warning message, like:  
Warning: {linter\_short\_code} is not a valid short code. This will make it clearer to the user what went wrong.

Another thing to consider is handling the configuration file (--conf). It’s great that we can load a config file if one is provided, but it would be a good idea to check if the file exists before trying to read it. If the file doesn’t exist or is invalid, an error message could be displayed. That way, users aren’t left wondering why things aren’t working when they pass in an incorrect file path.

When it comes to the linter classes, I think the subclasses like EnsureTimestamps and EnsureWorkspaceCleanup are a great addition, but the logic in those classes could be more DRY (Don’t Repeat Yourself). Since they all have similar structures, maybe we could create a base class with shared functionality, like handling the XPath checking, and then each subclass could just focus on its specific check. This would make the code more maintainable in the long run.

On the topic of short\_code, I think it’s a good system to use, but some of the codes like L006 could be a bit more descriptive. For example, L006 could be something like LINT\_SHELL\_BUILDER\_EMPTY to make it easier to understand what it’s doing at a glance.

I noticed that the docstrings are in place, which is awesome. But they could use a bit more detail in some places. For instance, in the linter classes, it could be helpful to mention the specific behavvior or edge cases the linter is designed to handle. This would make it easier for someone else (or even you in the future) to understand the purpose of the linter at a glance.

Also, just a heads up: I didn’t see any test cases related to the new features. It’s super important to add unit tests for these new linters, especially since they’ll be dealing with different job configurations. I’d suggest writing tests that simulate different job scenarios to ensure that each linter is working as expected. It doesn’t have to be super complex, just enough to check that the linters are doing their job.

I think the error handling in general could use some attention. If the user provides bad input or something goes wrong during execution, it would be great to show them some kind of error message instead of failing silently. This will make the tool more user-friendly and easier to troubleshoot.

As for the CLI interface, I love the idea of having the list-linters command to show available linters and their descriptions. But, if there are a lot of linters in the future, maybe we could improve the formatting of the output. For instance, using a table format (like with the tabulate library) or adding some color might make it more readable.

Overall, I think you’re on the right track. The features you’ve added are really useful and make the tool much more flexible. There are just a few things, like validation, error handling, and test coverage, that could make the experience even smoother. Keep up the great work, and let me know if you need help with anything specific!

**Formal Comments on the PR**

**These would be the comments I would make on the CodeControl & Version Control Location**

**1. Configuration File Handling (main() function)**

* **Line 82-91**: The addition of --select and --ignore options in the main() function is a useful enhancement. However, there’s no validation for the short codes passed through these options. It would be beneficial to validate that each short code exists in LINTER\_SHORT\_CODES before adding them to the configuration. If any invalid short codes are provided, we should return a warning message to the user rather than silently ignoring them.
  + **Suggested Improvement**: Add a check to ensure that each short code provided in --select and --ignore is valid. If not, display a warning like:  
    Warning: {linter\_short\_code} is not a valid short code.

**2. Configuration File Path (--conf Option)**

* **Line 92-101**: The logic for loading the configuration file works well when the --conf option is provided. However, we should check if the file actually exists before trying to read it. If the file is missing or invalid, it would be helpful to provide a clear error message to the user instead of silently failing.
  + **Suggested Improvement**: Before loading the configuration, check if the file exists. If not, print an error message:  
    Error: The provided configuration file does not exist.

**3. DRY Principle in Linter Classes**

* **Line 41-91**: The linter classes such as EnsureTimestamps, EnsureWorkspaceCleanup, and others all follow a similar pattern for checking certain elements in the XML configuration. This leads to some code duplication. To improve maintainability, we could create a base class for common functionality (e.g., XPath checking), which each specific linter can inherit from.
  + **Suggested Improvement**: Refactor the common logic into a base class and allow specific linters to implement only their unique checks.

**4. Clarity in Short Codes**

* **Line 87-96**: The use of short codes like L001, L002, etc., is fine but could be more descriptive. Instead of using generic codes like L001, L002, we could use more meaningful short codes that describe the check more clearly, such as LINT\_TIMESTAMP\_CHECK, LINT\_WORKSPACE\_CLEANUP, etc.
  + **Suggested Improvement**: Consider using more descriptive short codes to make the code more readable and easier to understand at a glance.

**5. Docstrings in Linter Classes**

* **Line 41-183**: While the docstrings in the linter classes are a good start, they could be more detailed. For example, in the EnsureTimestamps and EnsureWorkspaceCleanup classes, it would be helpful to explain the specific behavior of each linter and the edge cases it’s designed to handle. This will make it easier for future contributors to understand the purpose of each class.
  + **Suggested Improvement**: Expand docstrings to include detailed information about the behavior, edge cases, and any configuration options for each linter.

**6. Error Handling**

* **Line 92-101**: There is currently no robust error handling in the new features, particularly when invalid input is provided by the user (e.g., invalid short codes, missing configuration files). We should improve error handling by displaying clear error messages instead of allowing the program to fail silently.
  + **Suggested Improvement**: Implement error handling for invalid inputs, such as:
    - Invalid short codes
    - Missing configuration files
    - Invalid directory paths

**7. Test Coverage**

* The new linters like EnsureTimestamps and EnsureWorkspaceCleanup need adequate test coverage to ensure they work as expected. There are no test cases provided for these new features.
  + **Suggested Improvement**: Write unit tests for each linter to verify their behavior. This should include testing edge cases and ensuring that the linters behave correctly with various Jenkins job configurations.

**8. CLI Output Formatting (list-linters)**

* **Line 111-114**: The list-linters command works well to list available linters, but the output can be hard to read, especially when there are many linters in the future. Consider using a more readable format like a table or adding color for better visibility.
  + **Suggested Improvement**: Use a table format (e.g., with the tabulate library) to display the linters, or add some color to the output for better readability.

**9. Additional Suggestions**

* **Line 40-91**: It would be helpful to check if the --select and --ignore options are mutually exclusive with the --conf option, as currently, there is no clear message if both are provided at the same time.
  + **Suggested Improvement**: If both --conf and --select/--ignore are provided, display an error message saying these options cannot be used together.

**GPT Queries I made for your reference, to structure the data**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Estimated Time :**

It took me 15-20 minutes to understand the PR, and 35-40 minutes to translate my improvements into words. An additional 5 minutes were spent fixing the grammatical issues using ChatGPT.

So overall, it took around 1 hour. It could’ve been done faster; I faced problems organizing the data in the Word document due to OCR(Google Docx & Word) issues. I could complete the same task within 30-35 minutes if my OCR were more accurate