Elliott de Bruin
Ofek Inbar
Daniel Wang
CSE 403

Flint

Abstract

Checkstyle-IDEA is a widely-used IDE extension that, within its own GUI ecosystem, allows users to scan Java code for style violations according to a configuration (custom or default). It allows for the user to define an existing configuration to use, run the CheckStyle tool with the specified configuration over a file, module, or project, and view the output of the tool either in a list (akin to console output) or as highlights of the lines in question. However, it lacks the ability to create custom configurations within this GUI. We propose adding this functionality to Checkstyle-IDEA through a GUI Configuration Editor.

Background

When developing, style guides help maintain a clean codebase. Style guides contain rules for formatting and documenting code. Many software companies such as Google, Twitter, and Mozilla use style guides (that can span over 1000 lines) to motivate their Java style. Style linters such as Checkstyle are used to check for style violations in code. Checkstyle-IDEA, a plugin for IntelliJ IDEA, uses Checkstyle to allow users to check for style violations (as defined by default or custom configurations) in a Java file, module, or project, all in their IDE. It is incredibly easy to install and set up, but is noticeably lacking support for defining custom configurations within its functionality.

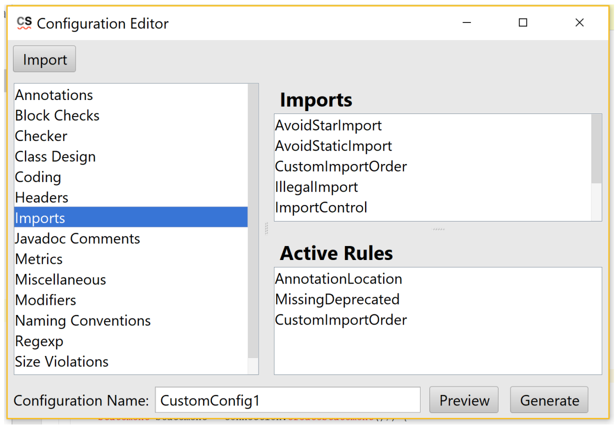
Problem

The two biggest inconveniences with the current way of defining custom configurations are that a) the user must write the configuration in XML, and must therefore have to worry about syntax and formatting errors while determining the rules to apply and b) the user must switch to a different window to view Checkstyle's Check documentation, thus breaking up the workflow and reducing productivity. All too often developers have to think about too many things at once and get lost in the chaos. Part of the reason for this is needing to be able to think about what they are trying to do as well as how they need to do it. Syntax is the last thing anybody should have to struggle with when configuring the style rules to run for their project. Switching between windows, otherwise known as context-switching or task-switching, to find rules and add them to the configuration can also be incredibly tedious. To quote Joel Spolsky, "the longer it takes to task switch, the bigger the penalty you pay for multitasking" (Spolsky). Having to separately think about looking up the rules you wish to apply and including them in the configuration slows developers down and reduces their productivity. These two reasons may be the primary deterrent for new users to lint their code.

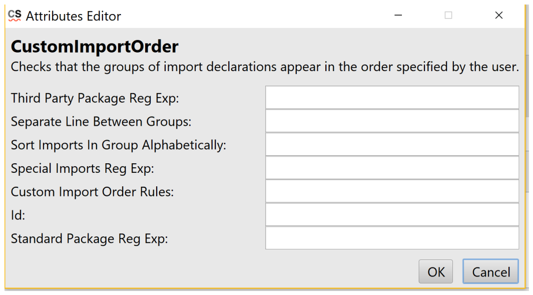
On the CheckStyle-IDEA repository's issue tracker, we located several instances of users of the tool facing errors that stemmed from having to manually write the XML file (here, here, and here). All of these issues were caused by an incorrect syntax for the XML configuration; more specifically, the configurations were written in a syntax that is older than the most recent version of CheckStyle.

Solution

The solution we propose is to transform the task of manually writing lines of XML configuration code into a point-and-click experience with a Configuration GUI integrating with Checkstyle-IDEA. Below is a screenshot of the tool.



The GUI allows the user to view all of the rules CheckStyle offers by default, filtered by category. When a rule in the top-right panel is clicked, a dialog such as the one below opens up.



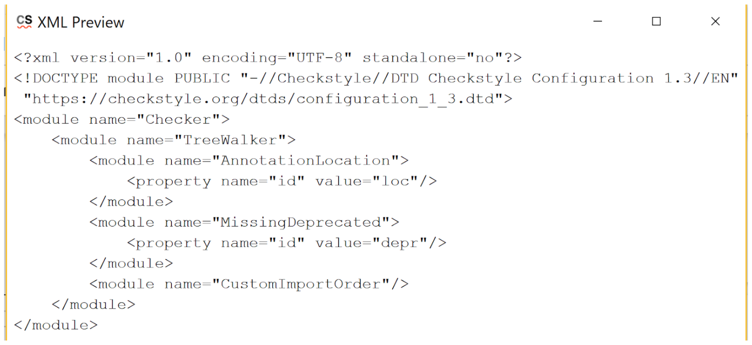
Attributes Editor

CustomImportOrder

Checks that the groups of import declarations appear in the order specified by the user.

Third Party Package Reg Exp:	
Separate Line Between Groups:	
Sort Imports In Group Alphabetically:	
Special Imports Reg Exp:	
Custom Import Order Rules:	
Id:	
Standard Package Reg Exp:	

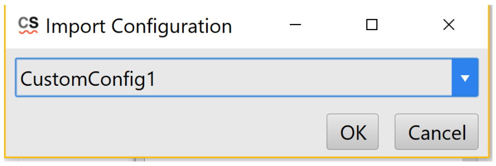
This dialog presents to the user all of the properties they can set for the rule they selected. Pressing the "OK" button adds the rule with the given parameters to the current configuration. The rules present in the current configuration are seen in the bottom-right panel of the window. Clicking the "Preview" button allows a user to open a window with a text preview of what the XML file will look like.

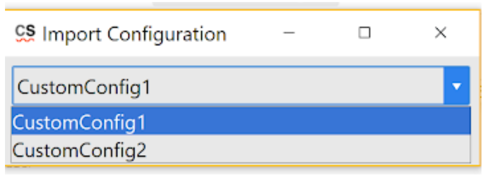


XML Preview

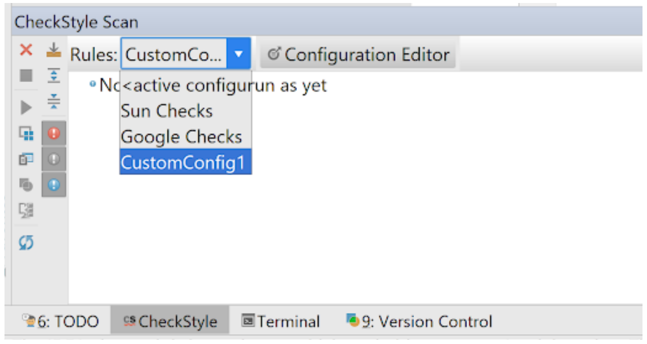
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE module PUBLIC "-//Checkstyle//DTD Checkstyle Configuration 1.3//EN"
 "https://checkstyle.org/dtds/configuration_1_3.dtd">
<module name="Checker">
    <module name="TreeWalker">
        <module name="AnnotationLocation">
            <property name="id" value="loc"/>
        </module>
        <module name="MissingDeprecated">
            <property name="id" value="depr"/>
        </module>
        <module name="CustomImportOrder"/>
    </module>
</module>
```

Clicking the "Generate" button will export the current configuration to a file with filename corresponding to the text provided in the "Configuration Name" text box. Clicking the "Import" button pops up a window that allows the user to select a configuration they have generated, and pressing "OK" with a config selected will import that configuration into the editor window.





In the near future we hope to have the tool integrate any generated configuration files with the original CheckStyle plugin so that the user can select to use the config to lint their files.



As a result of the fact the the configuration file is automatically generated, the user never has to worry about making a syntax error when creating or maintaining a configuration. Additionally, since all of the rules are displayed in the window along with their descriptions, the user doesn't need to switch back and forth between their configuration and CheckStyle's documentation, instead only having to use the GUI itself.

Related Work

There are several similar projects to note. JCSC is a style linter that has a GUI "Rule Editor" that allows the user to configure the rules to run on their file, and even has an extension for IntelliJ IDEA. However, JCSC has not been maintained since 2005 and is incompatible with Java past Java 5 (we are now on Java 11). The link to its repository is broken as well. Another notable project is Eclipse-CS. Eclipse-CS is a plugin for the Eclipse IDE that uses Checkstyle to check for style violations, and it offers a GUI to customize the configuration. The limitation is that Eclipse-CS is specific to the Eclipse IDE, and cannot be applied to IntelliJ IDEA, which we hope to do.

Challenges and Risks

The first challenge we may face is how to integrate our work into **Checkstyle-IDEA**, different from any personal or class projects that the scope is limited to a few people, this is a open source project that has their own contributing guidelines that we have to follow. We might need to familiarize ourselves with the guidelines and codebase first before we dig too deep into the project.

Another challenge we may face is to design an interface and workflow that we expose no use of XML to the user. The tool we are going to build should not require the user to have any prior knowledge about what XML is and how they work.

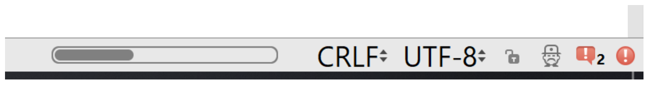
One other obvious big challenge we will face is time. Since this project is a remake from the previous one, we are essentially squeezing in the workload we have to do in a quarter into a half quarter schedule. However, we believe this can be solved by having previous experience working with checkstyle code base, and having a more efficient work distribution in the team based on the experience we have worked together so far.

Evaluation

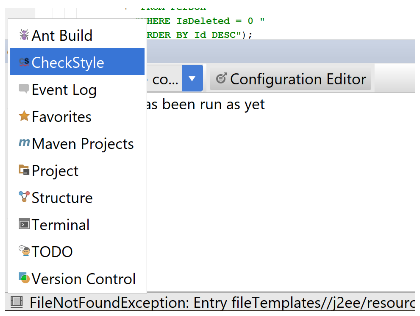
In order to evaluate the success of our tool we will utilize several different methods. The first way in which we will measure our tool's effectiveness is by identifying existing developers' struggles with Checkstyle-IDEA and verifying that our tool helps to solve them. We have found several GitHub issues opened against the repository expressing users' issues with creating XML files for CheckStyle-IDEA. To get concrete results on how our tool addresses the problems we are attempting to solve, we will evaluate the project through user testing. We plan to ask the Teaching Assistants to use the tool and subsequently answer some questions about its utility and ease-of-use. Some of the ways we will ask the Teaching Assistants to use are tool will put them in the same position as the users who created issues on the CheckStyle-IDEA repository, and the subsequent questions will be focused on verifying they did not encounter the same problem when using our Configuration Editor. This will help us to assess whether we are meeting our goals of reducing developer frustration due to syntax errors or frequent context switches. In order for users to be able to test the tool, we will have to release an initial launch to the IntelliJ marketplace. We also plan to open a Pull Request into the original Checkstyle-IDEA repository to receive feedback (via code review) from the original plugin's creators.

Initial Results

Our repository is located here, and the main development branch in on "**flint-development**". As we do not yet have data from user testing, we don't have a way to generate figures representing this data in our repository. Following is a set of instructions to run the tool in its current state (note that, as we are actively working to get a v1 working, many features may not behave as expected):

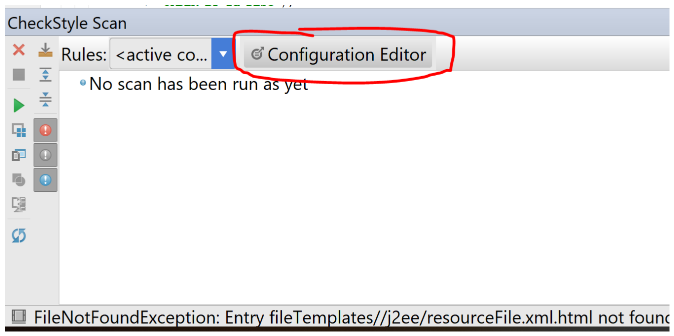
- Clone the repository
- From inside the root project directory ("checkstyle-idea"), checkout the branch "flint-development" (the most recent version)
- From the same directory, run the command ".\gradlew runIde" on Windows or "./gradlew runIde" on Mac/Linux
- This will (after some time) open an IntelliJ-IDEA window
- Open an IntelliJ project in the IDE
- Allow the project to finish loading (you can see its progress in the progress bar in the lower right-hand corner)
    - 

- Hover over the button in the lower-left corner and click the text that says "CheckStyle"

- ○ 

```
"HERE IsDeleted = 0 "
RDER BY Id DESC");
```

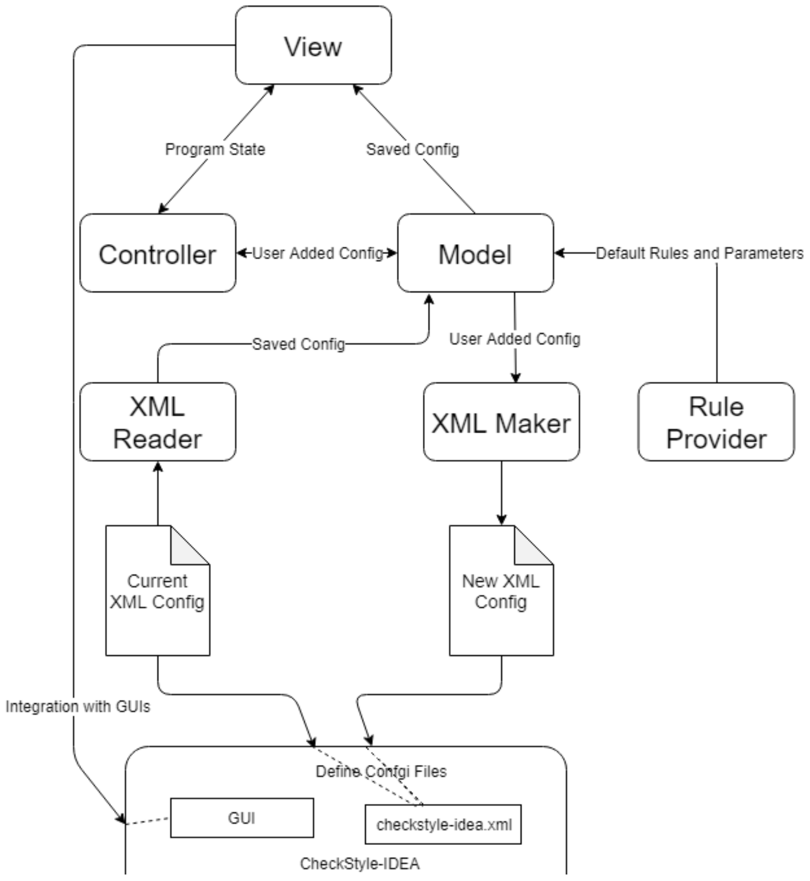
FileNotFoundException: Entry fileTemplates//j2ee/resourc

- Click on the “Configuration Editor” button to open our editor window
    - ○ 

No scan has been run as yet

FileNotFoundException: Entry fileTemplates//j2ee/resourceFile.xml.html not founc

Architecture Design



Since this involved a user interface, we have decided to use the MVC model as the main part of the architecture.

View:

- Responsible to show the rules contained in attached to the current config
- Show the available rule modules to users in a structured manner
- Allow users to add, remove, and modify (through attribute values) rules from the config
- Sends user behavior to Controller

Controller:

- Monitors what have the user clicked, and call for corresponding changes on the GUI or Model

Model:

- Provide the formatted XML data from existing XML configuration to Controller
- Pass the user made configuration to XML Maker.
- Provide available rules to Controller

XML Reader:

- Parse the existing XML configuration file into Java friendly format.

XML Maker:

- Export the user-defined rule modules into XML format
- Save user-defined rule modules as XML file

Rule Provider:

- Provides the set of available rules and possible internal settings.

Implementation Plan

The first stage of the development is to make sure we can read and write our XML based implementation to either display as a easy to understand abstraction or save the abstraction as actual configuration that a user has entered. We built the XML reader and writer along with a ADT that represents individual checkstyle configuration files. The evaluation of this part is to take an existing checkstyle configuration, convert it to the configuration ADT, and then write that ADT back to another configuration. Since we claim the XML reader and writer will correctly convert between actual configuration and easier to understand format, the two file should generate the exact same set of result on any given file. We did this evaluation by regenerating the google checkstyle configuration, ran both on a couple different java file, and proved our claim.

The second stage of the development was to build a GUI that users can easily go through a set of information they need, and add checker rule without actually writing the XML. In this stage, as we slowly built up the GUI, we also found more ways to actually make the tool easier to use for the user. For example, the GUI had to show description and possible properties for each type of check, which urged us to also built supporting providers to supplement all those user facing metadata. By the end of this stage, we had built out current running GUI, rule provider, property metadata provider, and input validator in order to enhance the functionality of the editor.

The last stage of the implementation is to connect all the rule provider, property metadata provider, input validator and XML reader/writer with the GUI. We manually tested this connection by ensuring that adding parameters for a check and pressing “OK” indeed caused the check to be displayed in the “Active Rules” panel. Additionally, we verified that the XML displayed in the “Preview” window accurately represented the state of the current configuration. Finally, we ensured that providing a Configuration Name and pressing “Generate” produced an XML file in the project (and that the “Import” dialog included that configuration in its combobox).

The XML file generated by our addition to Checkstyle-IDEA will be formatted and written specifically to be used with Checkstyle. Once the user generates a new configuration file they will be a able select their new rule configuration file to use with Checkstyle-IDEA.

We used several tools and technologies while completing this project. We used **Java** to write the tool and **Gradle** to build the project, as it is the build system **Checkstyle-IDEA**’s repository was already using. We are also using **JUnit** and **Mockito** to run our automated test suite. We are using the existing **Checkstyle-IDEA** codebase and the **IntelliJ Idea Plugin SDK** to build our plugin, and we use the IDE to test its GUI and integration.

Feedback

In order to address the previous feedback that we needed to provide better motivation for our work, we tried to find a more tangible problem developers may face when using Checkstyle, and decided that our previous solution was not necessarily a minimal solution to the problem. Thus, we have re-thought our approach and solution.

We later received feedback that our evaluation strategy was not present/detailed enough, and hence we have added more detail regarding how we will measure success for this project.

Lessons Learned

Throughout this entire quarter, our team has gone through a lot of major setbacks in our project development. Including struggling to give a proper motivation to the first project, restarting the entire project in the middle of the quarter, losing a teammate during our development phase, and finishing the project in a time frame much shorter than other teams. However, we did learn a lot about the important takeaway in this class.

We should do a lot more research on the project before we start implementing anything. In our first version of our project proposal, one of the things that had been pointed out was that our motivation lacks concrete example and proof to show our project show any currently existing issue. We were not able to address the issue properly until we decided to examine real-world reports and current solution in our problem space. After doing this research on current solutions and error report, we are finally able to come up with a project proposal that “has the potential for more impact” as Mike stated.

Another type of feedback we have received from our previous proposal is that we shouldn’t be reinventing the wheels. In terms of our previous project, we were basically trying to write another linter, that the only difference is the configuration style comparing to checkstyle. Considering the scope of this class, we will not be able to have a product that could perform more variety of checks than checkstyle does. Therefore, only having a configuration style that was subjectively considered “easier” will not justify the shortcomings of our product comparing to the currently matured tools. In order to solve the issue that linter configuration could be hard to pick up by users without compromising currently matured feature, we switched to forking the currently existing checkstyle-idea tool and do enhancements on it. Through this way, we only have a linter that already works really well, but also has a solution can be provided to the already large user base.

Right before we restart our project, we already have some implementation done to address the problem we are trying to choose. However, at that point, we hadn’t properly address how and why problem we are solving is important and worth solving. This eventually led to that fact that we had to give up tens of hours we have put into the project. From this, we learned that having a good problem to solve is actually more important than having an implementation or not. If a problem we are trying to solve is not going to make any positive changes, the implementation will eventually be worthless. With this in mind, in the restarted project, we made sure we had a good problem that worth solving and made sure our proposal is good enough before we start implementing our feature.

The last thing we have learned is that we should know our audience. In the initial result presentation, we got two kinds of distinct comment. One type of comment raised questions about what our new project has actually changed from the previous one, and the other type of comment understand what our new project is actually doing and raised questions about further details in the progress. This demonstrated that our presentation was only gear towards people who know what our latest changes were, without providing context to the audience that might not have such close interaction with our project. We should have put more emphasis on explaining what the entire project is about the first time we changed our direction, and then give a quick review for later presentation.

Bibliography

"Checkstyle – Checkstyle 8.17." *Checkstyle*, 27 Jan. 2019, checkstyle.sourceforge.net/.

Maria Christakis , Christian Bird, What developers want and need from program analysis: an empirical study, Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, September 03-07, 2016, Singapore, Singapore [doi>10.1145/2970276.2970347]

Jshiell, "checkstyle-idea". *Checkstyle-IDEA,* 18 Feb. 2019, https://github.com/jshiell/checkstyle-idea

Checkstyle, "eclipse-cs". *Eclipse-CS,* 18, Feb. 2019, https://github.com/checkstyle/eclipse-cs

Twitter, "twitter/commons". *Twitter Java Style Guide*, 18 Feb. 2019, https://github.com/twitter/commons/blob/master/src/java/com/twitter/common/styleguide.md

Mozilla, "Coding Style". *Java Practices,* 18 Feb. 2019, https://developer.mozilla.org/en-US/docs/Mozilla/Developer_guide/Coding_Style#Java_practices

Google, "Google Java Style Guide". *Google*, 18 Feb. 2019, https://google.github.io/styleguide/javaguide.html

Joel Spolsky, "Human Task Switches Considered Harmful", 12 Feb. 2001, https://www.joelonsoftware.com/2001/02/12/human-task-switches-considered-harmful/

Balazsmaria, "LeftCurly: Property 'maxLineLength' in module LeftCurly does not exist", 31 Oct 2017, https://github.com/jshiell/checkstyle-idea/issues/355

SButterfly, "The Checkstyle rules file could not be parsed.", 29 Jul 2017, https://github.com/jshiell/checkstyle-idea/issues/339

Marcelstoer, "SuppressionCommentFilter is not allowed as a child in Checker" , 8 Aug 2017, https://github.com/jshiell/checkstyle-idea/issues/341

Danilaml, "Error when importing custom checkstyle config", 25 Sep 2017, https://github.com/jshiell/checkstyle-idea/issues/348

Appendix:

Week-by-Week Schedule

Jan 28 - Feb 1	● ~~Research IDE plugin/extension development (decide on an IDE to develop for)~~ ● ~~Design architecture & implementation plan (produce deliverables)~~ ● ~~Write up specifications & user manual~~
Feb 4 - Feb 8	● ~~Write tests for Syntax Parser~~ ● ~~Integrate 3rd Party Syntax Parser package~~ ● ~~Test Driver~~ ● ~~Driver Fully Built~~ ● ~~Test CLI Adapter~~ ● ~~CLI Adapter Fully Built~~
Feb 11 - Feb 15	● ~~Test customization support~~ ● ~~Build customization support~~
Feb 18 - Feb 22	● ~~Integrate a button in the CheckStyle-IDEA plugin that pulls up a blank GUI window~~ ● ~~Build and test XML Reader component~~ ● ~~Build and test XML Maker component~~
Feb 25 - Mar 1	● ~~Finish building GUI~~ ● ~~Build and test Model~~ ● ~~Build and~~ test Controller
Mar 4 - Mar 8	● ~~Connect all MVC modules~~ ● ~~Test integration with CheckStyle-IDEA~~
Mar 11 - Mar 15	● User testing
Mar 18 - Mar 21	● Prepare presentation

Hours Spent
15 hrs