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.

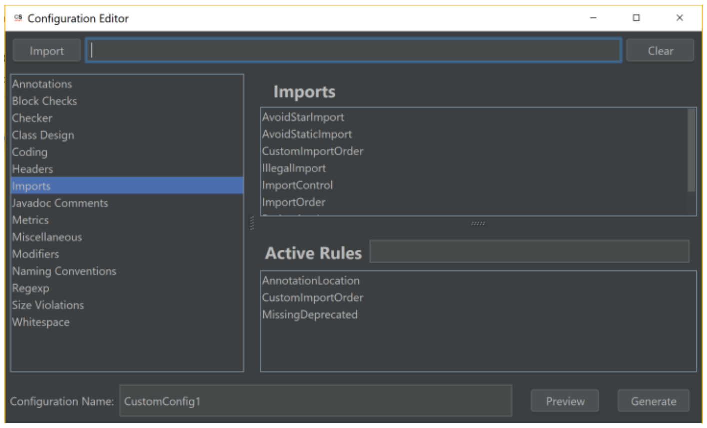
Problem

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. If marcelstoer, the author of the first issue linked, had built his configuration through a visual GUI that obscured the internal XML syntax, he would not have had this issue, as the GUI would have generated a syntactically correct configuration.

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.



Annotations
Block Checks
Checker
Class Design
Coding
Headers
Imports
Javadoc Comments
Metrics
Miscellaneous
Modifiers
Naming Conventions
Regexp
Size Violations
Whitespace

Imports

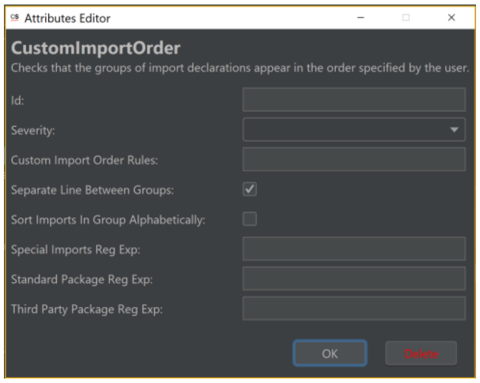
AvoidStarImport
AvoidStaticImport
CustomImportOrder
IllegalImport
ImportControl
ImportOrder

Active Rules

AnnotationLocation
CustomImportOrder
MissingDeprecated

Configuration Name: CustomConfig1

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.

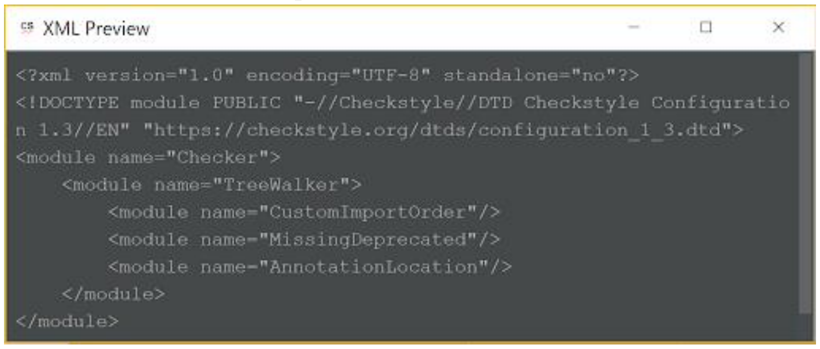


CustomImportOrder

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

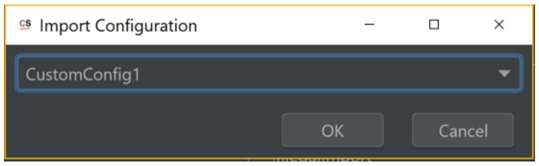
Id:
Severity:
Custom Import Order Rules:
Separate Line Between Groups: ✓
Sort Imports In Group Alphabetically:
Special Imports Reg Exp:
Standard Package Reg Exp:
Third Party Package Reg Exp:

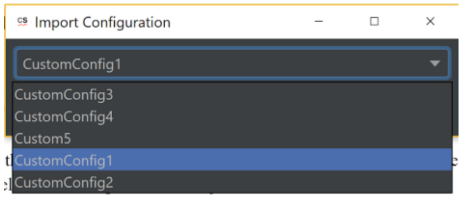
This dialog presents to the user all of the properties they can set for the rule they selected. Pressing the "OK" button adds the rule with the given parameters to the current configuration. The rules present in the current configuration are seen in the bottom-right panel of the window. Clicking the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



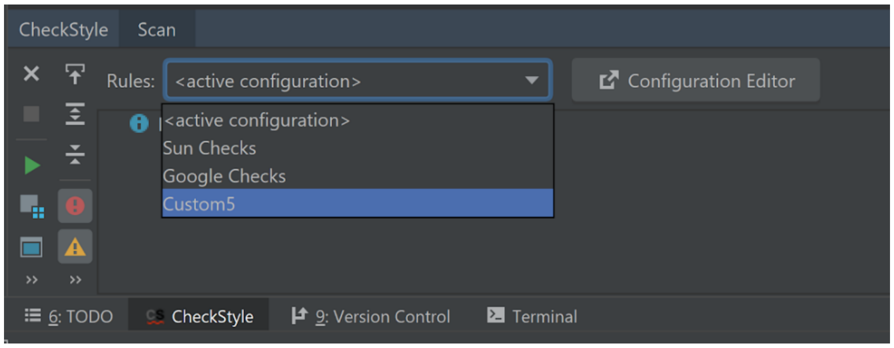
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE module PUBLIC "-//Checkstyle//DTD Checkstyle Configuratio
n 1.3//EN" "https://checkstyle.org/dtds/configuration_1_3.dtd">
<module name="Checker">
    <module name="TreeWalker">
        <module name="CustomImportOrder"/>
        <module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

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.

Related Work

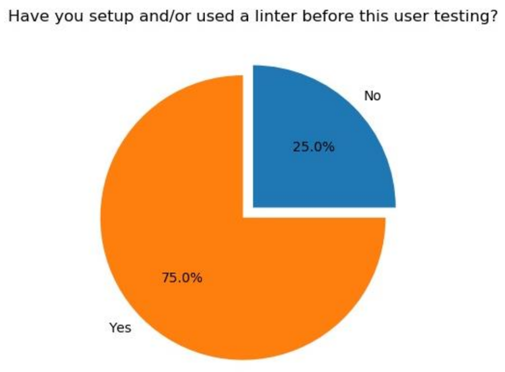
There are several similar projects to note. JCSC is a style linter that has a GUI "Rule Editor" that allows the user to configure the rules to run on their file, and even has an extension for IntelliJ IDEA. However, JCSC has not been maintained since 2005 and is incompatible with Java past Java 5 (we 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.

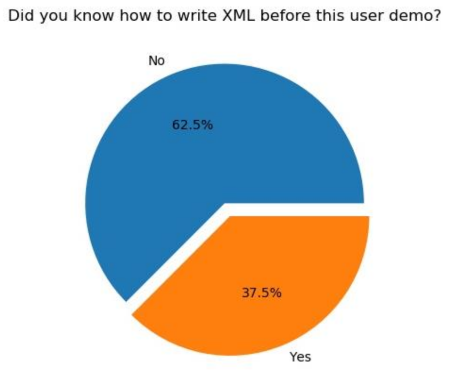
Evaluation

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 evaluated the project through user testing. We have also asked the Teaching Assistants to use the tool and subsequently answer some questions about its utility and ease-of-use. We asked the Teaching Assistants to use in the way that the Teaching Assistants will be put in the same position as the users who created issues on the CheckStyle-IDEA repository, and the subsequent survey focused on verifying that 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 released an initial launch to the IntelliJ marketplace. We also opened a Pull Request into the original Checkstyle-IDEA repository to receive feedback (via code review) from the original plugin's creators.

Initial Results

After doing our user testing with 8 participants, we have initial result to determine the effectiveness of our product. Among our participants, 75% of them have used a linter before, and 63% of them have never used XML before. We have users of all experience levels in terms of programming work, from novice level user who has only taken CSE 143, to software engineers who are currently working in the industry.

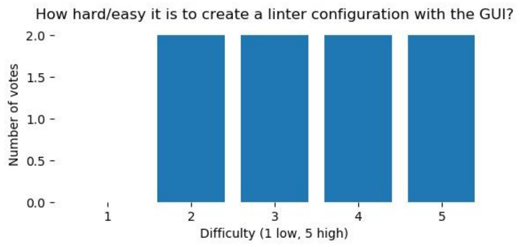


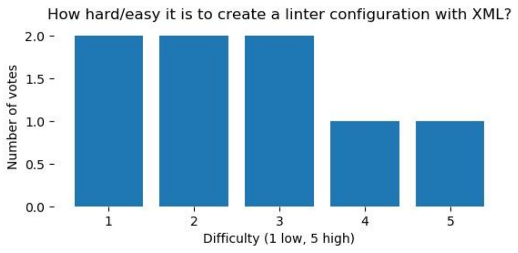


In this round of user testing, we have assigned three tasks to the participants. The first task is to ask users to make a configuration that checks two style rules, a line length limit of 80 characters and a warning against unused imports. This task is designed to be the easiest task for the participants to actually get familiar with how to find and add the rules they want. The two rules needed to be added in this task only required the user to enter numbers, and add the rule straight in. In addition, the corresponding rules' names in checkstyle, "LineLength" and "UnusedImports", are much easier to understand than others. The second task is that we ask users are to check for whether there is a space in between the method name and the parameter declaration, and to check for stand-alone semicolons (";") in the code. This task has rules that are not only having properties needing a specific set of options to fill in, but also unintuitive names. For example, to check for spaces in between the method name and parameter declaration, the corresponding rule is named "MethodParamPad", which did not directly address the whitespace part of the check. In the third task, we asked the participants to edit the checkstyle configuration that performs checks according to the Google Java Style Guide. The participants had to change the 100 character line limit to 80, change the indentation for the "throws" keyword to 2, and turn off "AvoidStarImport". This third task is to test whether users can find and change a large configuration more easily in a GUI rather than a XML file. We chose one participant who has only taken introductory programming and one other

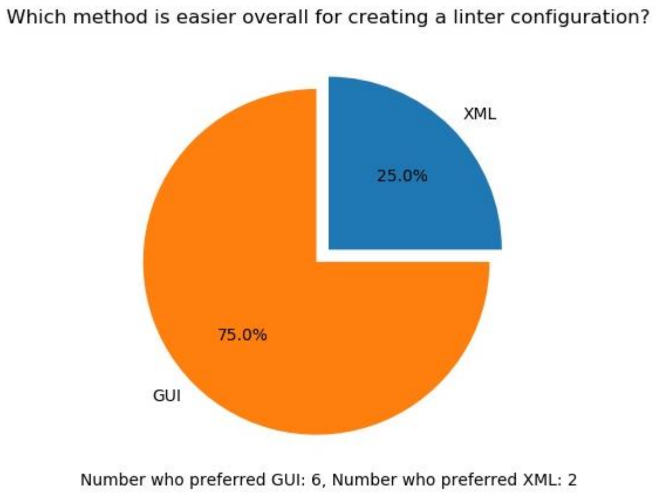
CS student to do all three tasks in the GUI editor. The rest of the participants were assigned one task to complete by directly writing XML, and the rest to complete with the GUI editor. After all the tasks were done, all users filled out a survey about their experience.

In terms of the ease-of-use score, we asked the participants to score from a 1 to 5 scale, 1 being very hard to use and 5 being very easy to use, for both methods of creating configurations (GUI editor or direct XML). The overall average of the GUI editor is 3.4, and the average for the XML config is 2.5. Although this is not a huge difference, this does show that the GUI editor is easier to use overall while comparing to the XML (according to our test users).

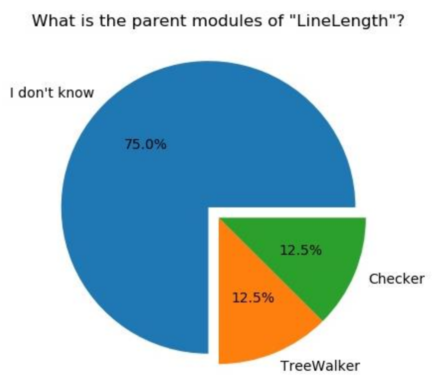


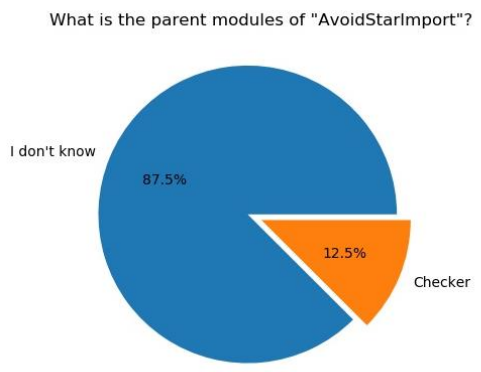


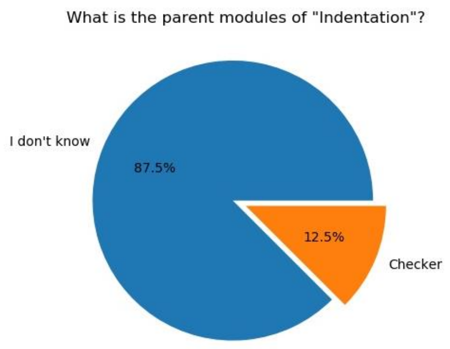
In addition, when we look at the scores from the individual participants, only two participants gave XML a higher (easier) score over the GUI editor. The rest of the participants gave a higher score for the GUI editor over the XML setup. Therefore, with this result we concluded that this GUI editor does provide a easier way for most of the users to make their checkstyle configuration.



One of the claims we make about this project is that we can allow a user build a working CheckStyle configuration without having to know anything about the XML syntax structure. To test this claim, we asked three questions regarding the checkstyle configuration structure after the users finished the three assigned tasks. We asked about what the parent modules of the rule “LineLength”, “AviodStarImport”, and “Indentation” are respectively.

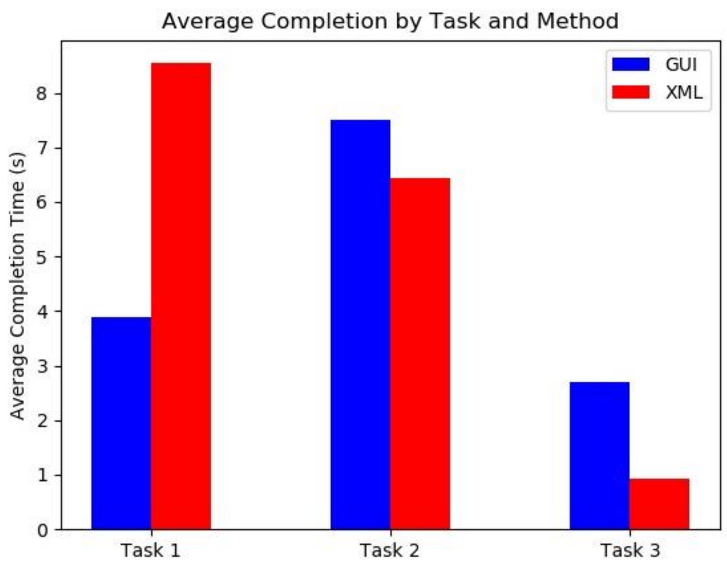






Only one participant answered any of the three questions correctly (and even then, it was only for "LineLength"), the rest of the participants either gave an incorrect answer or claimed not to know. This result occurred even after XML tasks was assigned due to the fact that most participants either could not figure out how to build an XML configuration from scratch within 10 minutes (and so they quit early), or they just edited a very specific location in the configuration file in task 3 without actually needing to understand the structure of the CheckStyle configuration setup. As a result, the main takeaway for this survey question revealed that even without any knowledge about the checkstyle configuration setup structure, users are able to build configurations from scratch as they wanted.

The last claim we try to justify is that our tool can effectively save users time (thus increasing their productivity). We have measured the time it took each participant to complete each task, categorized by whether the user did the task with XML or the GUI editor. The following chart is the average time spent in each task, categorized by completion method.



From the chart, we may see that the time participants spent in task 1 when they are using GUI is significantly less than those who did task 1 with XML. In this case, since the rules the participants need to find is relatively easy, the people using GUI had a clear advantage in terms of time needed. However, in task 2 and 3, the average time used for people did with GUI is longer than those who did in XML. Does this mean our tool failed? We believe there are other factors that contribute to this result. As mentioned before, task 2 is designed to let the users find out the rules that were named in a way what is not intuitive to understand. In addition, from the user feedback, one major difficulty users faced was to "understand the name of the function" (Benson, 2019), understanding the "Checkstyle naming convention" (Jessica, 2019), and "find[ing] the rules to change" (Thomas, 2019). From this feedback and the design of task 2, we would like to conclude that this slightly more time needed in GUI was contributed by users trying to understand what each checkstyle rule means. As for the situation in task 3, we believe this was caused by not having a search function for the active rules. In the version of the plugin that we used for testing, there was a search function over possible rules to add within a specific category, but there was no search function for the active rules (and also none that searched available rules over all categories). From what we observed, participants who did task 3 simply searched some key words in the XML file and changed the right rule really quickly. For the participants who used the GUI editor, they have to look at each active rule entry in order to find what they want. In addition, from the freeform feedback regarding the difficulty of using the GUI editor, the limited searching capabilities were brought up multiple times.

At the end, we are able to justify several of our original claims. Firstly, we are able to provide an, albeit subjectively, easier way of creating CheckStyle configurations. We are also able to abstract away the structure of the checkstyle XML configuration from the user. However, we still cannot confidently make the claim that reduce the time spent building CheckStyle configurations until we can factor out the influence of the sometimes obscure rule names and descriptions provided by CheckStyle. We discuss how we plan to address our user feedback in the "Future Plans" section later.

The other aspect of of our evaluation was to create a pull request with our Configuration tool on the original CheckStyle-IDEA repository and receive feedback on our tool from the creator of the plugin.

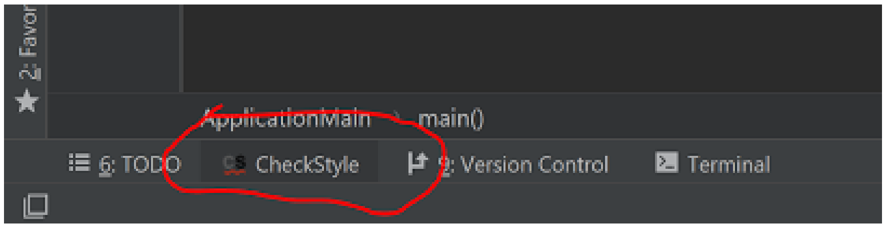
We created the pull request and the creator of CheckStyle-IDEA, jshiell, replied that he is currently traveling for work and will review the pull request when he returns home later this week.

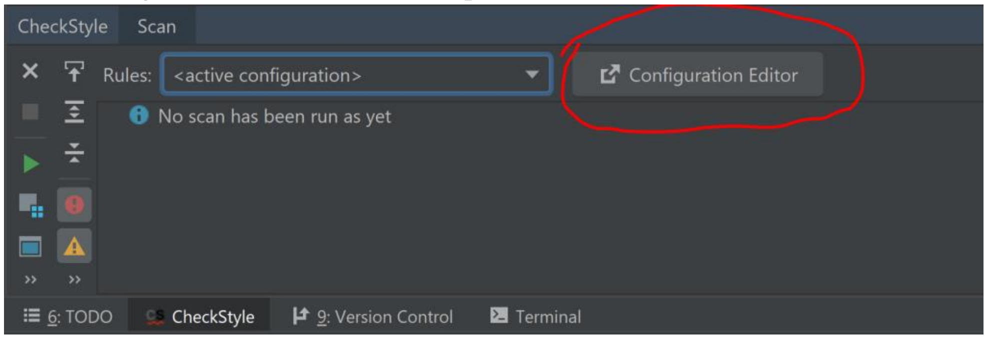
Reproducing our results

Our repository is located here, and the main development branch is on "**flint-development**". To generate the figures that follow, heed the following steps:

- Clone the repository
- Checkout the master branch (should be default)
- Run the command ".\generate_figures" on Windows CMD or "./generate_figures" on Mac/Linux Terminal from the root directory of the repo "checkstyle-idea"
    - Note you will need Python 3 installed
    - You will also need the packages "requests", "matplotlib", and "numpy" installed. As they do not come pre-packaged with Python, you may need to run the command "pip install <package-name>" to get them.
- The figures should be generated (up-to-date) in the directory "reports/week10/evaluation/figures/"

We have a pending update to the public plugin "CheckStyle-IDEA (Flint Distribution)". Once it is accepted, you will be able to run the tool by installing it in IntelliJ IDEA, but until then you can see the most recent version of the tool by heeding the following instructions:

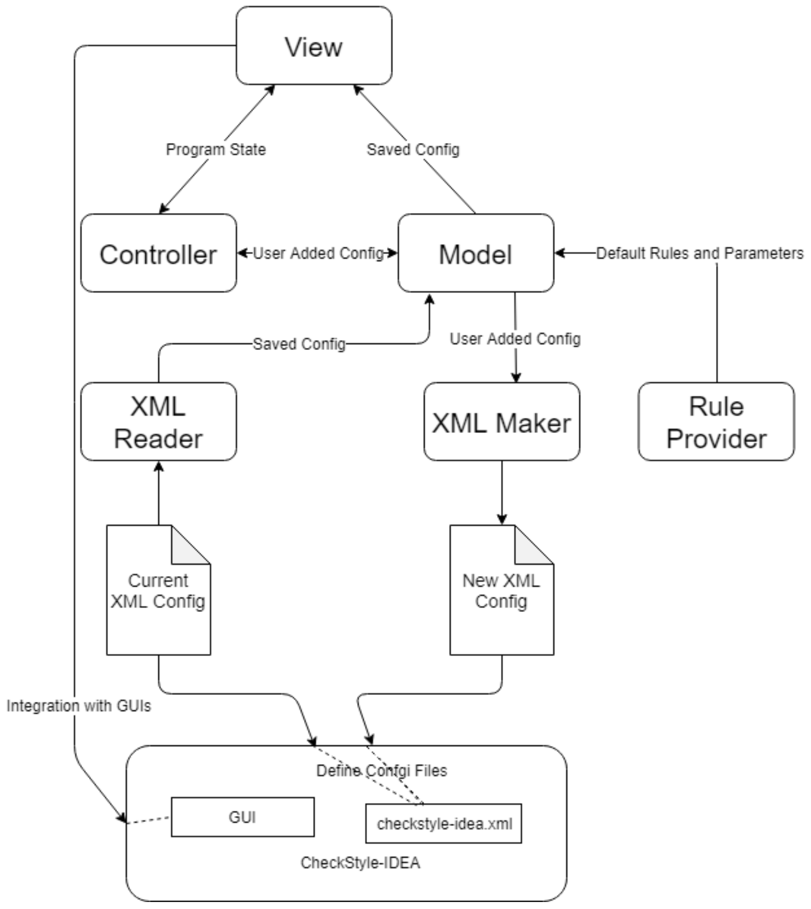
- Clone the repository
- From inside the root project directory ("checkstyle-idea"), checkout the branch "flint-development" (the most recent version)
- From the same directory, run the command ".\gradlew build" on Windows or "./gradlew build" on Mac/Linux
- Open IntelliJ IDEA (and open or create a project)
- Install the plugin from disk from the ".zip" file found in "checkstyle-idea/build/distributions/"
- You should see a button to open the "CheckStyle" tool window at the bottom of the window. Click it.
    - 

- Click on the "Configuration Editor" button to open our editor window
    - 

No scan has been run as yet

Architecture Design

Since this project was essentially a user interface, we have decided to use the MVC (Model-View-Controller) design pattern as the basis for 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.

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.

During our user testing phase we received a lot of feedback from our users on how we could improve the product from a user's perspective. There were three major concerns our users expressed during the testing. The first was the delete button on active rules was too similar to where a cancel button would otherwise be so a user accidentally deleted an active rule. To fix this we changed the color of the text on the button to red, thereby better distinguishing it. The second and most common concern was that it was difficult to search for rules both in the Visible Rules panel (in the top-right) and in the Active Rules section. To address this we changed the implementation for searching to search rules by name and description in all categories rather than just by name in the selected category. We also added a search bar for the Active Rules section so the user can easily search for rules by name, description, attribute name, and attribute value in their current configuration. Lasty, another concern was that the tool did not provide any feedback to the user when a configuration file is successfully generated. In response to this we added a pop up window that will inform the user when their file has been generated and where to find it.

Future Plans

In the future there are a few ways we can improve the Configuration Editor addition to CheckStyle-IDEA. The greater goal for our tool is to have our pull request accepted by CheckStyle-IDEA, integrating our work with the original plugin. Since our pull request has been acknowledged by jshiell, the creator of the plugin, the next step once we receive feedback will be to address it (all aspects) so that we can improve the tool such that it adds value to CheckStyle-IDEA.

Going forward there are two main features that we would like to implement, which we did not have time to do in the span of this project. The first feature is to automatically add all generated configuration files to CheckStyle-IDEA's configurations drop-down menu. At the moment to add a configuration to this menu, which allows them to run the configuration, they must add it through the plugin's preferences. Implementing this enhancement would require less of the user and improve their workflow. The second feature we would like to implement is the ability to search and add custom or 3rd-party checks through the Configuration Editor. This would allow the user to keep all their configuration work within our tool if they are using 3rd-party rules along with CheckStyle's rules.

We received a lot of good feedback for improving the Configuration Editor from our first round of user testing so after making improvements in the future we can conduct more user testing to gather feedback on a more developed version of our tool. For future user testing we would also like to have users that use CheckStyle and CheckStyle-IDEA on a regular basis as well as those who have never used either before. This would give us insight into how well the tool works for a wider range of customers and how we can improve the tool to fit the majority of users needs.

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.

Hours Spent

10

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