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.

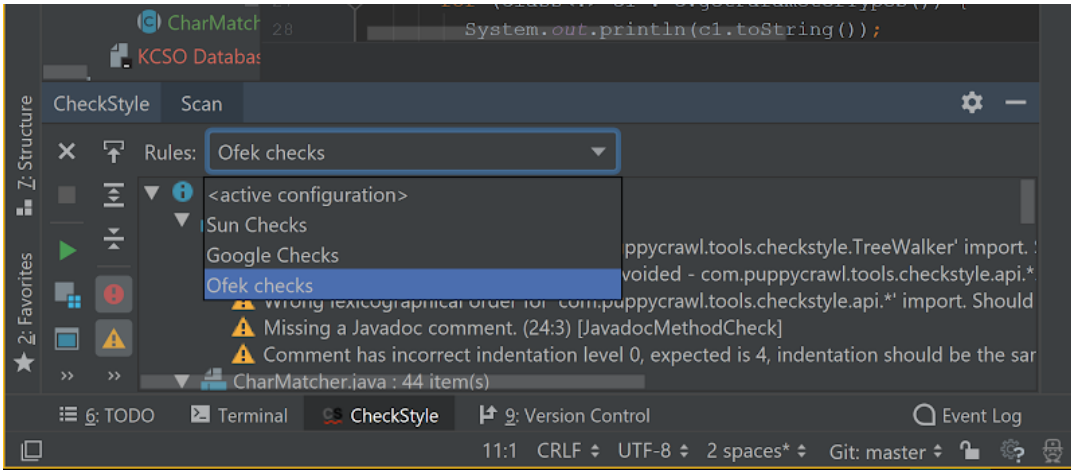
Problem

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.

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. The

GUI will allow the user to easily look through all of the rules Checkstyle offers (along with their documentation), add checks to their configuration, and export it to a properly-formatted XML file. We also plan to integrate this tool directly with Checkstyle-IDEA, so the custom configuration will get automatically added to the list the user can select from. This will mean that the user can install, run, and configure Checkstyle while only interacting with GUI's and never having to read or write a line of XML. This ensures the user need not worry about syntax or format and also allows the user to look up rules and add them to the configuration from the same window.



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.

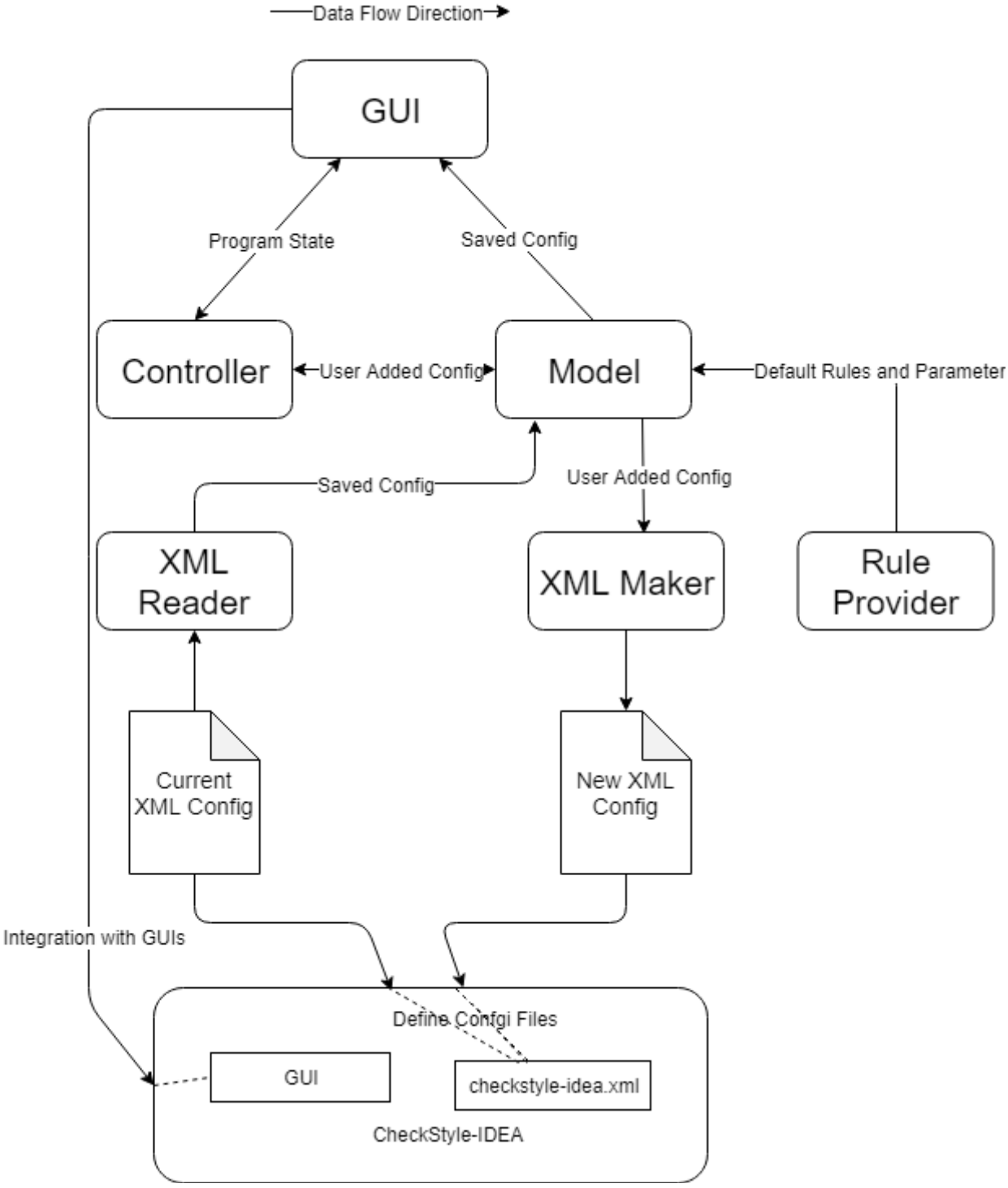
Another challenge we may face is to design an interface and workflow that we expose no use of XML to the user. The tool we are going to build should not require the user to have any prior knowledge about what XML is and how they work.

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.

User Testing

One crucial aspect of creating the GUI Configuration Editor for CheckStyle-IDEA will be conducting and reviewing useful and informative experiments to help develop and refine the tool. We will conduct User Testing after the product is fully built and integrated with CheckStyle-IDEA. User Testing will consist of asking Java programmers to use our GUI Configuration Editor to generate a style guide to use on one of their projects with the premade CheckStyle's rules, without having to have prior knowledge of XML syntax and using only the user manual and documentation that will be available to all new users. Optionally, for programmers who have prior knowledge of XML and CheckStyle, we will ask them to import Checkstyle configuration files with custom style rules for their projects and assess the use of our tool for someone who has used Checkstyle without it previously. Ideally, we want users to be able to use our tool with ease from the first use, so seeing how new users interact with the software will allow us to refine the features of the tool as well as the user manual. We will measure success on two fronts: 1) User feedback on how easy it was to figure out how to use the main features of the tool/interact with the GUI, and 2) User feedback on how helpful/useful the tool was, and how likely they would be to use it in the future. If these experiments succeed, we will have received useful input from real users on which to base our final product.

Architecture Design



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

GUI:

- 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.

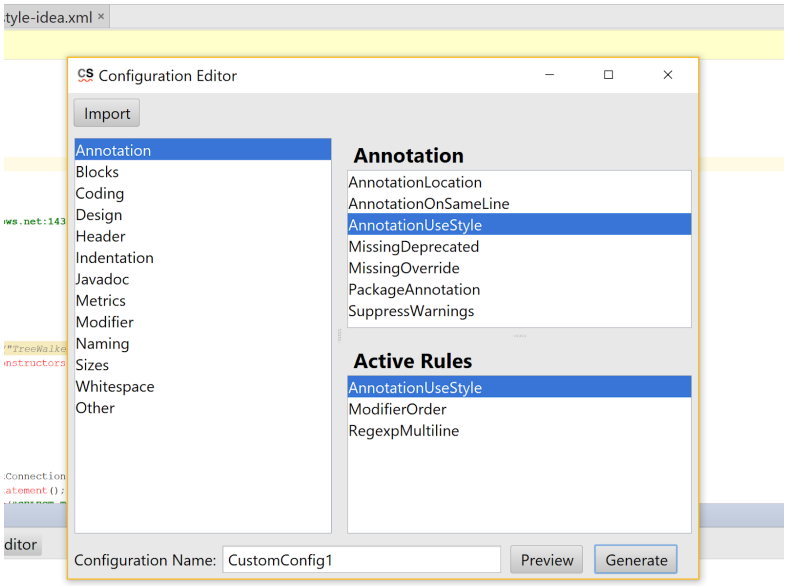
XML Maker:

- 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



Here we have a mockup of the GUI for our tool. When a user has our version of Checkstyle-IDEA installed on their IDE they will have an added button, along with the rest of Checkstyle-IDEA's controls, labeled *Generate Configuration.* Clicking on this button will display our GUI for generating XML configuration files. In the top left of the larger window there is an import button,

which when clicked will allow the user to import an existing XML configuration file. To the right of the import button we have a search bar that allows the user to search our rule database for a specific rule, which will be displayed in the section below the search bar. On the left side of the window there is a list of categories. Checkstyle has a large set of premade rules that are divided into categories and each category will be displayed on the left, including a *Custom Rule* category, which will contain any custom rules the user chooses to write. For a users custom rule to be included in our tool they will need to include a Java file or JAR file for the rule in a designated *CustomRules* directory in the project's *.idea* directory. Clicking on a category will display all the rules contained in that category in the top-right panel. Clicking on a rule in the top-right panel will display another small window containing the description and parameters for the selected rule (as well as text fields to provide values for them), a "cancel" button, and an "ok" button that, when clicked, will add the rule with the given parameter values to the config. This rule will then show up in the *Active Rules* panel below. At the bottom of the GUI window there is a field to define the name of this configuration (which will be used as the file name when it is exported). If a configuration file is imported all rules in that file will appear in the *Active Rules* section and the name field will be auto-filled with the imported file's filename. There is then a *Preview* button which will display a preview of what the XML file will look like given all the rules that have been added thus far. Finally we have the *Generate* button which will generate the new configuration file under the specified file name.

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.

The first stage of our implementation plan is to develop the XML Maker component. When the user clicks *Generate* in the GUI the selected rules will be passed through to the XML Maker, where it create a XML Checkstyle configuration file. The next stage will be to develop the GUI to start a dialog with the user when it is opened. Once we have our GUI we will integrate it with the XML Maker and Checkstyle-IDEA.

We foresee using several tools and technologies while completing this project. We will use **Java** to write the tool and **Gradle** to build the project, as it is the build system we are most familiar with. We will also be using **JUnit** to run our automated test suite. We will use **Checkstyle-IDEA** and the **IntelliJ Idea** plugin API to build our extension, and we will use the IDE to test our plugin.

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.

Initial Results

Our initial results include a configuration XML read/write tool, and the rough GUI prototype that shows up in IntelliJ IDEA.

The XML read/write contains the read and write part. The reader reads in a checkstyle configuration and converts it into a Checkstyle-specific rule representation. The writer takes the

Checkstyle-specific rule representation, and turns it into a checkstyle-specific configuration XML file. Our demo of the XML read/write is on branch **init-result** (https://github.com/s92025592025/checkstyle-idea/tree/init-result), with the instruction in folder **Notes/InitialResultInstruction.md (https://github.com/s92025592025/checkstyle-idea/blob/init-result/Notes/InitialResultInstruction.md).** What the demo is essentially doing is that it reads in the offcial google style checkstyle config in the repo root, convert to our self-defined ADT, then write it back to a newly regenerated checkstyle configuration file. After that users may see that there is no difference between the newly generate config and the original one when they run both on checkstyle-idea.

The instruction to view the initial result of the GUI prototype is as follows:

1. Clone the repo
2. git checkout 34f6f388cc5b9cd67d469bbf33f2e042138d0cd8
3. Run './gradlew.sh runIde' or '.\gradlew.bat runIde' from the root of the repository (On Windows, making sure your **PATH** or **JAVA_HOME** is pointing to the right jdk)
4. A sandbox IDEA window should open up
5. Open an IntelliJ IDEA project in the sandbox window
6. Wait for the editor to finish loading (this may take a few seconds)
7. Hover the mouse over the icon in the lower left-hand corner (as shown in the image below)
8. Click the 'CheckStyle' button

9.
10. Click "Configuration Editor"

Week-by-Week

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

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/