This is the replication package of our FSE'23 paper:

Kobi Feldman, Martin Kellogg, and Oscar Chaparro, "Proceedings of the 31st ACM Joint Meeting on the Foundations of Software Engineering (ESEC/FSE'23)", (to appear), 2023

This document is formatted as a Markdown document. See the README.pdf for the rendered version of this document, or use your favorite editor/IDE to render this document.

Package Structure

- `complexity.pdf`: a copy of our (almost) camera-ready paper
- `data`:
  - All verification tool output. (i.e., the results of running the tools. You can do the correlation analysis from these raw tool output files, if you just want to repeat our analysis and not our data collection.)
  - All csv sheets created with the analysis tool output.
  - The file "correlation_analysis_timeout_zero.xlsx", which stores an overview of data collected including correlation results for when openjml timeouts are treated as zeros
  - The file "correlation_analysis_timeout_max.xlsx" which stores an overview of data collected including correlation results for when openjml timeouts are treated as the max # of warnings openjml found on any snippet per dataset.
- `tool-execution`:
  - All shell scripts for executing different steps of the pipeline. Note: ALL .sh scripts are designed to be run from the project root directory
- `simple-datasets`:
  - Contains the snippets for COG datasets 1, 2, and 3. Also contains the snippets for the fMRI study.
- `dataset6`:
  - Contains the snippets for COG dataset 6
- `dataset9`:
  - contains the snippets for COG dataset 9
- `meta-analysis`:
  - contains the R scripts used to perform the meta-analysis
- `scatter_plots (and related directories)`:
  - raw scatterplots generated by all of our R analysis
- `MetricList.xlsx`: data about the metrics from prior studies
- other folders/files that you can ignore (e.g., CogniCrypt, jpf, cognicryptoutput.txt, etc.): relics from tools that we tried to run on the datasets, but could not for various reasons. We leave them here for completeness; see section 2.3.2 of the paper.
- various `.py` files: the python scripts used for data analysis. Their use is described below.

Kick-the-tires Instructions

- follow the "setting up the project environment" instructions below after the full instructions (< 5 minutes).
- run `./tool-execution/run_checker_framework.sh` to re-collect the Checker Framework data (about 5 minutes: should print "Checker Framework successfuly ran. View output in data/checker_framework_output.txt" when it finishes successfully).
- run `./tool-execution/run_correlation_analysis.sh` to run the first part of the data analysis: the individual correlations between each metric and verifiability (< 5 minutes)
- run the `scatter_plots_and_cor_analysis.py` Python script to generate the results of our ablation experiments.
- open the file `meta-analysis/metafor-based-meta-analysis.r` in R Studio, modify the `base_path` to the current folder (the full path), and execute the script. Open the file `forest-plot/all_tools_positive_negated_agg.pdf`, which the R script should have re-generated; it should look very similar (hopefully identical!) to the plot in Figure 1 in the paper. (< 5 minutes)

Full Instructions

- same as the kick-the-tires instructions, except try to regenerate all of the data. See the instructions in the section of this document labeled "Collecting the Data".
    - except maybe the openJML data for datasets 3 and 6, which take a very long time (at least a week on our server) and require you to set timeouts based on your hardware. We recommend skipping these during artifact evaluation and using our pre-collected data, for your own sanity.
- when viewing the results of the meta-analysis, also look at the other forest plots in the same folder: they support RQs 3 and 4 (what each plot is should be obvious from the file names: the single-tool analyses are named after the tool; the ablation study results are named "no_toolX" for each leave-one-out fold; the "metric categories" have the name of the category in the plot.)
- this documents also contains sections with "checklists/steps" for adding a new verification tool, snippet dataset, or comprehensibility metric to the pipeline. These will hopefully enable others to reuse this artifact and build on our work (an undergrad has successfully followed them, so we hope they are sufficiently detailed).

Setting Up Project Environment

- Make sure that you have Python and Java installed on your machine; you need Java 8, 11, and 17 JDKs
    - I use Java versions 17.0.3, 11, and 8 and Python version 3.10.5 although other similar versions will probably work fine.
    - You will need to swap between the Java versions when using some of the different analysis tools. On Linux, this can be easily done with the following command: `sudo update-alternatives --config java` or by changing your JAVA_HOME environment variable

- Go to project root
  - `cd complexity_verification_project`
- Run the following 3 commands from project root to setup the python environment:
  - Create the virtual environment: `python3 -m venv complexity_verification_project_venv` Note: The command name might differ depending on your version of Python: Could be python, python3, python3.10, etc
- Activate the virtual environment with bash/zsh: `source complexity_verification_project_venv/bin/activate` Note: activation differs based on platform: https://docs.python.org/3/library/venv.html
- Install the required libraries to the virtual environment: `pip3 install -r requirements.txt` Note: you might have to give yourself permission to run the .sh scripts using chmod.
- Install the other software specified in the REQUIREMENTS file (R and RStudio, in particular)

Adding a New Verification Tool

Note: this is only needed when extending the work, not when replicating our results

- Create new .bat and .sh scripts to run the analysis tool + add any other dependencies, files, etc that are needed to run the analysis tool.
- Create a new function in warning_parser.py to parse the analysis tool's output and convert it to a .csv.
- Create a new sheet in "correlation_analysis.xlsx" named after the analysis tool and fill in its column titles as well as the first 2 columns worth of data (same as the other sheets).
- In `correlation.py`:
  - Update the 'if **name** == "**main**":' section at the bottom so that correlations can be done on the new analysis tool isolated from the others.
  - Update the readAnalysisToolOutput() function to read in the .csv file created for this analysis tool.

Adding a New Snippet Dataset or Comprehensibility Metrics

Note: this is only needed when extending the work, not when replicating our results

- Add the dataset to the project
- Add any relevant metric results from the previous study that used the dataset to the project to data folder in our repo
- Update the .bat and .sh scripts that are used to run the analysis tools on all the datasets to include the new dataset:

    - Some tools such as the Checker Framework automatically check all .java files within the project
    - Other tools such as the Typestate Checker and OpenJML need to explicitly be given single directories of .java files at a time, thus needing to be updated when adding a new dataset
- Update every sheet in "correlation_analysis.xlsx". Add a new row containing the name of the dataset and metric in the same format as the other rows. Also add to the #of snippets judged column.
    - Columns to manually add: metric, dataset, metric_type, expected_cor, num_snippets_judged
    - add info for each sheet on the excel file
- In correlation.py, update the setupCorrelationData() function to read in the relevant metrics that we will correlate our warning counts against.
    - Write function to read metric data from your study
        - Ex: "readCOGDataset1StudyMetrics()"
    - Write function to create dataframe that contains both the metric and warning count for the snippets in that dataset
        - Ex: "setCogDataset1Datapoints()"

Collecting the Tool Warnings

- Run the Checker Framework (Checker Framework version 3.21.3, Java version 17.0.3) on all the snippets. Collect all of the warnings generated.
    - How To: Run this command from the project root directory: `./tool-execution/run_checker_framework.sh`
    - Output: The Checker Framework outputs all of its warnings to "checker_framework_output.txt"
- Run the Infer tool (version 1.1.0, Java version 17.0.3) on all the snippets. Collect all the warnings generated. Note that some warning types such as null dereferences cause the tool to stop analyzing the current method when spotted and move on to the next one.
    - First install infer (system-wide): https://github.com/facebook/infer/blob/main/INSTALL.md
    - How To: Run this command from the project root directory: `./tool-execution/run_infer-no-filter.sh` (note that there is also a `run_infer.sh` script; we did not use it for the paper, since it doesn't include the `--no-filter` argument to infer)
    - Output: Infer outputs all of its warnings to "infer_output.txt"
- Manually swap to Java version 8.
- Run the TypeState Checker (Checker Framework version 3.14.0, Java version 8 as recommended on the typestate Github) on a single folder of snippets. Collect all of the warnings generated. This differs from the Checker Framework in that the Checker Framework checks all snippets simultaneously while the Typestate Checker only operates on a single folder at a time.

    - How To: Run the script "run_typestate_checker.sh" from the project root with this command. This will run a series of commands to run the Typestate Checker separately for each dataset. `./tool-execution/run_typestate_checker.sh`
    - Output: The Typestate Checker outputs all of its warnings for each dataset to separate files called: "typestate_checker_output_{dataset name}.txt"
- Manually swap to Java 11.
- Setup OpenJML. We only have scripts to do this on Ubuntu 20.04. If you're on that OS, run `./tool-execution/setup-openjml-linux.sh` and then follow the instructions it prints. If not, create a new version of that script for your operating system following the instructions in that script's comments.
- Run OpenJML by running the ./tool-execution/run_openjml.sh script. WARNING: this is unlikely to terminate, because no timeouts are set. To set a timeout on a particular dataset, add the –timeout option (which takes a number of seconds as input) to the appropriate openJML invocation in run_openjml.sh. For example, to run on DS6 with a 30 second timeout, add –timeout 30 to the line starting with $OJ that mentions dataset 6. If you expect this script to terminate in reasonable time, DS3 and DS6 both need timeouts. We used 3600-second (i.e., 1 hour) timeouts for both in the paper; the resulting run took over a week on a powerful server.
    - You might want to just run OpenJML on the simple datasets. We provide the script `tool-execution/run_openjml_small.sh` to run just the datasets that did not require timeouts. That script should terminate on most modern machines in under a few hours.
- At this stage, all warnings have now been collected across all analysis tools and all datasets.

Run Correlation of Individual Metrics

- How to compute the results in table 3 (the individual metric correlation results):
    - Run the file "run_correlation_analysis.sh" to complete ALL steps 1-7 automatically. Therest of this section lays out how they work in detail, too, to explain how the process works. You probably don't need to read anything after this bullet if you don't want to modify the data analysis. **Skip the rest of this section and just run `run_correlation_analysis.sh` unless you want to understand how that script works.**
        - Run this command from the project root directory: `./tool-execution/run_correlation_analysis.sh`
    - The two scripts, "warning_parser.py" and "correlation.py" can alternatively be run manually and independently of each other if needed.
        - "warning_parser.py" is for Step 1
        - "correlation.py" is for Steps 2-7.

1. Parse the warning output from each analysis tool separately. a. Read the text file output from each analysis tool one by one. i. Note: Some analysis tools such as the typestate checker only run on a single directory at a time so they output multiple text files. b. For each analysis tool, read the text file(s) line by line, looking for keywords that definitively indicate the start of a new warning (as per the structure of each analysis tool's output which does differ). c. For

each warning, get the name of the snippet (format: {dataset} - {snippet #}) as well as the "warning type", the warning's definition in the output. d. Store the 2 pieces of data collected for each warning in 2 lists, then create a pivot table that shows the number of each type of warning per snippet e. Output: Pivot table is stored in a csv file named after the analysis tool the warnings came from. Example: "checker_framework_data.csv". i. Note: If dealing with multiple text files from the same analysis tool, warnings are combined together resulting in ONE csv file output PER analysis tool.

2. Gather all of the csv files created in Step 1.
3. Determine the number of snippets that contain warnings within each dataset. a. Using column 1 of "correlation_analysis.xlsx", get a list of all datasets being used. b. Form a list of all unique snippets that contain warnings by name (across all datasets). c. Using the lists of datasets and unique snippets with warnings, sort those snippets by dataset. d. Output: Counts are stored in "correlation_analysis.xlsx".
4. Determine the number of warnings per snippet per dataset. a. Get a list of all datasets being used. b. Get a list of the total number of snippets in each dataset. c. Go through each analysis tool's output. i. For each row of data, sum the values in that row, and note the snippet name at the start of the row. This gets the number of warnings for each snippet produced by a specific analysis tool ii. Keep a running count for each snippet to get its total number of warnings across all analysis tools.
5. Using the data collected in Step 4, Determine the total number of warnings per dataset a. Since each snippet's individual count of warnings is already sorted by dataset, simply sum those warning counts for each dataset. b. Output: These counts are stored in "correlation_analysis.xslx".
6. Setup the data for conducting a correlation analysis. a. For each dataset/metric, form 2 lists of equal size. The first contains the complexity metric for each snippet used on the dataset in its previous study. The second contains the total number of warnings (across all analysis tools) produced on each snippet. This forms the (x,y) datapoints used for conducting the correlation analysis. b. Update raw_correlation_data.csv c. Note: Some studies used multiple different metrics on a single dataset. These metrics are correlated with our warnings separately. d. Output: The number of datapoints for each dataset are stored in "correlation_analysis.xlsx".
7. Conduct Correlation Analysis a. Kendall's Tau: i .For each dataset, take the data points collected in Step 6 where x = complexity metric, y = # of warnings. ii. Perform Kendall's Correlation, getting the correlation coefficient and p-value for each dataset. iii. Output: The correlation coefficients and p-values are stored in "correlation_analysis.xlsx". b. Spearman's Rho: i. Exactly the same process as Kendall's correlation.