S.E.A.L. Article

Example

1 Methodology

1.1 Bug prediction

The study aims to analyse the performance accuracy of cloud specific bugs using traditional bug prediction pipeline. Several steps are included in machine learning pipeline. Few iterations were carried out to improve the accuracy of the bug prediction model.

Project Sample

Since we used the tool CK [2] to compute some of our metrics which specifically targets JAVA projects, we only included Java cloud projects in our sample. Furthermore, we selected projects with at least 100 releases.

Data Preperation

Metrics Selection Important metrics types to feed in to bug prediction pipeline were chosen based on earlier researches on similar context. (add a small description) loc cbo dit wmc lcom

Data Ingestion JGit is a java implementation of Git version control system. Git parser was implemented as the initial step of the bug prediction pipeline. All the commit messages between 2 releases were retrieved through the parser and then bug related commits were filtered using specific keywords to filter out the class names with the bugs. Keywords to filter bugs. • ERROR • FIX • FAILURE • CRASH • UNEXPECTED • WRONG CK library [1] was used in for all the classes at the time of the release in order to retrieve the certain features to generate the metrics. CK java library provides class level and metric level code metrics without compiling the code. Above mentioned loc, cbo, dit, wmc, lcom were calculated written into the excel sheet with an extra column mentioning whether each of the class has a bug or not by going through the list of class names with bugs.

Data Cleaning Data duplication couldn't be handled at the level of the java code due to the limitations of the used libraries. Before feeding in to the machine learning pipeline quality and reliability of data was assured through various pre-processing steps. In this stage irrelevant and redundant data is removed.

Model Building and Training

Logistic regression machine learning technique was chosen to build the model based on hyper parameter properties. RWeka library which is R interface of a collection of machine learning algorithms for data mining tasks written in java.

1.2 ASAT adoption in Cloud System

To answer RQ2, we searched for cloud applications and collected information about ASATs of the Go language.

Projects

We created a sample of 10 cloud projects written in GO that use ASAT and proceeded with the collection as follows: We used GITHUB's search functionality[1] using query words such as *cloud*, *PaaS* or *SaaS* to find suitable projects. We filtered the projects by the programming language (GO) and the number of stars (>500). We chose Go because it is considered a major language for cloud computing [10]. We then manually checked whether a given project really represented a cloud system by reading the repository's description, e.g. we found projects that just provide a CLI to a cloud service or tools for the development of cloud projects, but are not cloud applications themselves. Finally, we removed projects that do not use ASAT using a script that scans all projects files for ASAT command usage (more details in the next section).

Model building and training

ASATs

There are more than 30 ASATs available for the Go language. We gathered these tools from a GitHub repository [1] that compiles a list of ASATs for many programming languages. We excluded some of these tools, namely those that were archived or deprecated (e.g. [7]), or exploratory in nature (e.g. [6]) as we were interested in ASATs highlighting issues in the code. We stored all ASATs in a sqlite database.

The majority of ASATs specialize on specific problems in the code such as detecting deadlocks [3] or unused arguments in function declarations [8]. Two ASATs in our database are linter aggregators , i.e. they run multiple ASATs [4, 5]. Finally, some ASATs cover a variety of code issues (e.g [9]).

Many ASATs can be configured in terms of which warnings are enabled, what files they are run on or at which thresholds a warning should be issued. These configuration settings may be passed via a file or command line arguments.

Commandline configuration

References

[1] awesome static analysis. `https://github.com/mre/awesome-static-analysis`.

[2] ck tool. `https://github.com/mauricioaniche/ck`.

[3] dingo-hunter tool. `https://github.com/nickng/dingo-hunter`.

[4] golangci-lint tool. `https://github.com/golangci/golangci-lint`.

[5] goreporter tool. `https://github.com/360EntSecGroup-Skylar/goreporter`.

[1] https://github.com/search

[6] goroutine tool. https://github.com/linuxerwang/goroutine-inspect.
[7] interfacer tool. https://github.com/mvdan/interfacer.
[8] nargs tool. https://github.com/alexkohler/nargs.
[9] staticcheck tool. https://staticcheck.io/docs/.
[10] thenewstack article. https://thenewstack.io/go-the-programming-language-of-the-cloud/.