Hourglass Smart Test Job Runner

Hourglass Software LLC

Contact: support@hourglass-software.com

About

Hourglass Smart Test Job Runner is a tool that enhances and optimizes your CI/CD Pipelines. It uses subcomponents and services for risk analysis and quality assessment tasks, also using Machine Learning, and determinations where the resulting data finds areas of risk in your software and executes CI/CD Pipeline jobs accordingly. This targets CI/CD Pipeline jobs that are test automation. The results from the risk assessment will prioritize which test automation jobs to execute in priority order and only those that have been assessed to have test coverage for software areas that are "risky" and have a higher potential to yield defects. Thus, it is an efficient means of managing your CI/CD pipeline test automation jobs.

Amazon AMI and CloudFormation

The system is delivered as an Amazon AMI on a Windows Environment using CloudFormation to create the infrastructure. Once the AMI is delivered to an EC2 instance and the CloudFormation template completes creating the stack (template file included), it is ready for usage after some configuration.

IMPORTANT: Do not move or delete any of the files and folders delivered in the AMI. See below for file and directory structure.

Systems Integration

The tool runs on Windows from command line (you can connect via Session Manager). It integrates (requires) the following systems:

1. ***Atlassian Jira*** (for Machine Learning processing Bug history within areas of software under test)
2. ***BitBucket*** or ***Github*** (for using statistical data analysis on commits and changes to the repository to yield another set of risk assessment)
3. ***Jenkins*** or ***TeamCity*** (or any other CI/CD system that can trigger job execution using Web Requests via REST)
4. Runs on Windows (Command line)

System Requirements

The tool requires to following on your Windows host. These are already included in the AMI.

1. Java 8 / JDK 8
2. .NET Core 3.1
3. NodeJS

You will need to use Amazon S3 for specific configuration files. An S3 bucket is US West 2 is required which is automatically created by the CloudFormation template (the bucket is called s3hourglasssmarttestjobrunner).

These Environmental Variables are set and required:

NODE_ENV - C:\Program Files\nodejs

JAVA_HOME - C:\Program Files\Amazon Corretto\jdk11.0.13_8

How it works

Once executed, the application performs the tasks every 2 hours (application must be running). It runs/uses a Java application using Machine Learning (Hourglass Bug Predictor) to predict bug counts for specified areas in your software under test and results with bug prediction counts (Jira is required). It then runs "Code Risk Analyzer" which is a NodeJS application that uses either Github or BitBucket to query statistical data on code commits and code changes, and results with which source code files have the highest "risk" of yielding defects. Then, based on the results of these both, the application will execute test automation jobs in CI/CD (ie: Jenkins or TeamCity) that are associated with the software areas that have the highest probability for defects to occur. The results of "Hourglass Bug Predictor" and "Code Risk Analyzer" are normalized and combined.

Thus, as an example, if there are 100 CI/CD jobs that are in place to run, every 2 hours this tool will automatically only run the ones (10 jobs, for example) that have the highest probability of finding defects. This is a "smart" and efficient means of triggering test automation jobs in pipelines.

(see below for how "Hourglass Bug Predictor" and "Code Risk Analyzer" work)

Setup

- Follow Amazon's process to use the CloudFormation template to create the system.

- DO NOT change the installed files and folders installed or any related directories and directory structure and location
- DO NOT change any file or directory names
- DO NOT change the environment variables
- (see execute instructions below)

Background

- The application requires following data:
    - An integer ID for each of the CI/CD test automation jobs
    - A mapping between source files or folders to the job ID
    - A mapping between Jira data to job ID
    - A mapping for job ID to the test automation trigger REST URL into your CI/CD jobs triggered remotely

Required Configuration files

The following files will need to be placed in your S3 bucket.

Samples are included and shown where they will be placed.

The file names and type MUST MATCH.

- bug-predictor-schema.xml
    - see sample file
    - see "Hourglass Bug Predictor" document for requirements, format, and contents.
- predictParamData.csv
    - see sample file
    - see "Hourglass Bug Predictor" document for requirements, format, and contents.
- job_to_file_mapping.csv (sample provided)
    - 2 values separated by comma on each line
    - First column is job ID number
    - Second column is source file name of folder name in source file path that is associated with the job ID (see sample)
    - Can have as many entries as you like
    - Job IDs can be repeated for more than 1 file (entry)
- Job_to_webhook_mapping.csv (sample provided, replace this)
    - Each line has 2 values separated by comma
    - First value is job ID
    - Second value is job triggering URL used in Web Request to trigger (see sample)
        - Can be a parameterized URL
    - Only 1 entry for job ID to URL allowed

Running the Application

To run the application on the EC2 instance, you will need to go to the command line on the Instance using **Session Manager**. (go to AWS console for EC2 and run session manager on the instance to connect).

Once you have installed the files and folders and have setup the configurations required, simply run the tool from command line with:

Go to folder with executable:

C:\Hourglass Software\Hourglass Smart Test Job Runner\Controller\HourglassSmartTestJobRunner

Execute:

```
.\HourglassSmartTestJobRunner.exe <jira url> <jira user> <jira auth
token or password> <github/bitbucket> <number of jobs to trigger. Ie:
top X> <CI/CD user- agent user>  <CI/CD user-agent token or password>
```

If using github, append command line parameters:

```
<github repo> <github user> <github token>
```

If using bitbucket:

```
<bitbucket url> <bitbucket project> <bitbucket repo> <bitbucket token>
```

S3 parameters:

```
<s3 bucket name> <region>
```

All command line parameters must be 100% valid.

Example for github:

```
.\HourglassSmartTestJobRunner.exe "https://hourglass-
software.atlassian.net" "faisal@hourglass-software.com" "xx112233"
"github" 2 "fizzq" "abababb" "Fizz-Q/code-risk-analyzer" "Fizz-Q"
"xyxyxyxy" "s3hourglasssmarttestjobrunner" "us-west-2"
```

Valid region parameter values are:

us-east-1

us-east-2

us-west-1

us-west-2

Once executed, let the application continuously run. It will trigger jobs every 2 hours.

Appendix: Services Included and used in execution

Hourglass Bug Predictor

User Guide

V2.0

Contact: support@hourglass-software.com

About

Hourglass Bug Predictor is a Command Line Tool that integrates with Jira and uses Machine Learning to predict the future quality of your products in development. It will predict how many bugs you should expect in the next seven days, from the time the software is run.

It will query your Jira tracking system for Bugs under a specified Jira “Project”. Based on the data it finds in your Jira Management System, it will train a Machine Learning Model and the predict the number of bugs you should expect to find in the following week of your development.

For example, it will tell you/predict that there will be 5 bugs in the next week or some other prediction value. The prediction takes into account the fields of your Jira Bugs both for training as well

as prediction. The prediction feature vector also takes historical timelines and grouping of your bug data and respective attributes (see details below).

This will help you gauge the quality of your systems.

Example Output

Bug Prediction Count for next 7 days for input parameters:0.5

Pre-requisites

1. Jira host/endpoint
2. Java 8
3. Windows, MacOS, or Linux
4. Jira API token and username
5. Jira Project & Bugs history
6. (Product under development as target)

Setup

1. DO NOT DELETE ANY FILES OR FOLDERS
    a. The Application binary classes will be in the \bin folder
    b. Runtime dependencies will be in the \dependencies folder (do not delete any)
2. The install includes a sample input schema file and a sample input parameter data file (see details below)
3. Make sure you have Java (8) bin in your PATH environment variables and JAVA_HOME

Usage

- See “Hourglass Smart Test Job Runner”

Schema XML File

The schema xml file (bug-predictor-schema.xml) is to tell the software how your Jira Bugs are structures and what fields they use (for querying Jira, parsing the responses, and creating training data).

The schema XML file root element is <bugpredictor>. And then has the following fields with additional definitions of the fields:

<project> - this tells the software which Jira Project to query

Example:

<project>HBP</project>

<components> - provide the components your Bugs use

Example:

<components>

<component>test1</component>

<component>test2</component>

<component>test3</component>

</components>

<priorities> - priorities of your bugs. Can be Jira default or custom, but should be defined here

Example:

<priorities>

<priority>Lowest</priority>

<priority>Low</priority>

<priority>Medium</priority>

<priority>High</priority>

<priority>Highest</priority>

</priorities>

<labels> - the labels you use for your bugs

Example:

<labels>

<label>label1</label>

<label>label2</label>

<label>label3</label>

</labels>

<customFieldName> - this is OPTIONAL (with children elements). If your Bugs use a custom field that you would like to include for prediction. Define it here. Only 1 custom field definition is allow (but with as many possible values)

Example:

<customfields>

```
            <customfield>cf1</customfield>
            <customfield>cf2</customfield>
            <customfield>cf3</customfield>
</customfields>
```

Full XML file example:

```
<?xml version="1.0" encoding="UTF-8"?>
<bugpredictor>
    <project>HBP</project>
    <components>
        <component>test1</component>
        <component>test2</component>
        <component>test3</component>
    </components>
    <priorities>
        <priority>Lowest</priority>
        <priority>Low</priority>
        <priority>Medium</priority>
        <priority>High</priority>
        <priority>Highest</priority>
    </priorities>
    <labels>
        <label>label1</label>
        <label>label2</label>
        <label>label3</label>
    </labels>
    <customFieldName>customfield</customFieldName>
    <customfields>
        <customfield>cf1</customfield>
```

```
        <customfield>cf2</customfield>
        <customfield>cf3</customfield>
    </customfields>
</bugpredictor>
```

You can modify this file for different projects before any execution of Hourglass Bug Predictor.

Input Parameter CSV File for Prediction Control

This file (predictParamData.csv) allows you to make predictions of the number of bugs to expect with certain field values. For example, you can predict High Priority Bugs with Component A and Label B. The prediction will tell you the number of expected Bugs to find with those attributes. You can modify these parameters for different executions to find different predictions based on field values. The different types of parameters are on separate lines of the file. For example, component to use in the first, labels to use in the second, priority to use in the third, and custom field value to use (if applicable) on the last. You can leave lines blank for not using custom fields or labels, but the newline must be there. The file must also end at last line with default end of file (ie: no extra newline). The test automation job ID is associated with the data specified. Job Ids can be repeated for different sets of data.

The format of the .csv file is (no spaces allow between labels values, only comma):

```
<Job ID>
<Component name to predict for>
<Comma separated list of label values to predict for (or just one value>)
<Priority of Bug to predict for>
<Custom field value to predict for (if used as is optional)>
```

For example:

```
1
test1
label1,label2
Medium
cf1
2
```

```
test2
label3
High
(this is an empty line due to not including custom field)
3
test2
label1
Low
cf2
2
test3
label1
Medium
(this is an empty line due to not including custom field)
```

Hourglass Code Risk Analyzer

V2.0

Contact: support@hourglass-software.com

About

Hourglass Code Risk Analyzer used in "Hourglass Smart Test Job Runner" analyzes your source files to determine which files have the highest risk of potential defects. It uses the past 300 commits of your GitHub or Bitbucket repository and gets source code statistics per commit, and then uses a proprietary formula to calculate the risk score per file. Data for each file per commit is used to calculate a risk score such as:

1. number of commits in past 7 days
2. number of lines of code changes in past 7 days
3. number of commits in past 30 days

4. number of lines of code changes in past 30 days

5. number of commits total

6. number of lines of code changes total

7. commit message contains words that are scored (ie: fix, bug, etc)

8. number of unique users that committed in past 7 days

9. number of unique users that committed in past 30 days

10. number of unique users that committed total

The higher the score, the riskier it can contain defects.

The calculation also normalizes values.

Example Output

analyzing...

Analysis Complete. Results and risk scores for source files from highest to lowest risk:

SOURCE FILE: src/com/hourglasssoftware/bugpredictor/JiraManager.java RISK SCORE:2.806338028169014

SOURCE FILE: src/com/hourglasssoftware/bugpredictor/runner/HourglassBugPredictor.java RISK SCORE:2.4836747759282973

SOURCE FILE: src/com/hourglasssoftware/bugpredictor/DataManager.java RISK SCORE:2.422215108834827

SOURCE FILE: src/com/hourglasssoftware/bugpredictor/BugPredictor.java RISK SCORE:2.2877720870678617

The results are written to file with the source code file information and the score, which then gets read by the “Hourglass Smart Test Job Runner” Controller.