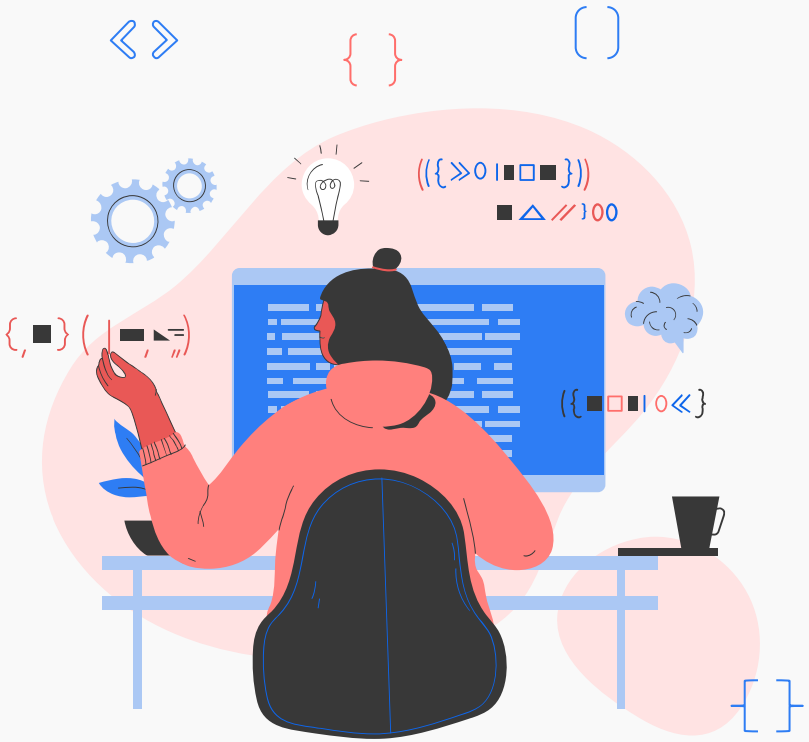
Software Engineering Testing

Group 7

Jay Sharma, Aanchal Sapkota, Vrushank Vaghani, Sanchit Duggal, Shamar Pryce, Kamsi Idimogu



Project Overview

01

Introduction

02

Selection Process

03

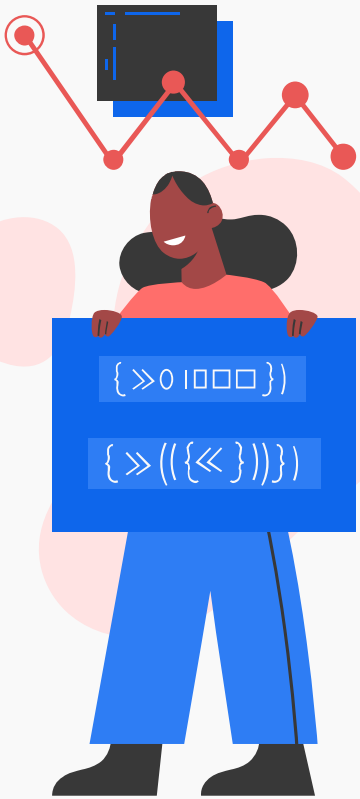
Testing Tools

04

Demo

05

Conclusions & Suggestions



01

Introduction

Overview & Evaluation

- Bug detection project focussed on **health** based projects, due to shared personal connection during COVID-19 lockdowns
- **Bug detection** is key to software development and **saving costs** in the maintenance part of the **software cycle**
- **Objective**: To successfully apply, understand and resolve bugs detected in **5 projects** different health based open source projects with more than 100 stars in GitHub using **3 tools**: Findbugs, Checkstyle, and Randoop
- **Structure**: Each team member conducted tests individually and most **impactful** and **resolvable** ones reported, all in **Java** language only

Evaluation Process & Projects Used

	Health Plus	Stop Coding	IBM Openshift	Health Care Service	Kardio Master
Description	21,488 LOC 54 classes	3,571 LOC 53 classes	1,665 LOC 15 classes	348 LOC 14 classes	1,393 LOC 21 classes
Relevance	Hospitals management system	Patient Data privacy	General Health safety	Medical Backend	Frontend D2C Health App
Diversity (Size & Scope)	Mid-sized company Large reviews	Open sourced project	Large multinational	Startup Member graph	Open sourced project 183 contributors
Popularity & Future	169 stars 112 forks More diagnostic	155 stars Submitting bug reports	134 forks Alter protocol	242 forks Submitting bug reports	212 stars Increase interoperability

02

Selection Process

Tool Selection Process

What we considered

- Specific to Java
  - It was crucial to select tools compatible with the Java
- Diverse in Function
  - Static analysis
  - Test generation
  - AI code analysis
- Proven and Reliable:
  - Preference was given to tools with a strong reputation and proven track record in the developer community.

Tool Selection Process

FindBugs

Pattern based Analysis
Graphical User Interface

CheckStyle

Static Analysis Tool
Eclipse Plug-in

Randoop

Automated Test
Generation Tool
Command Line

03

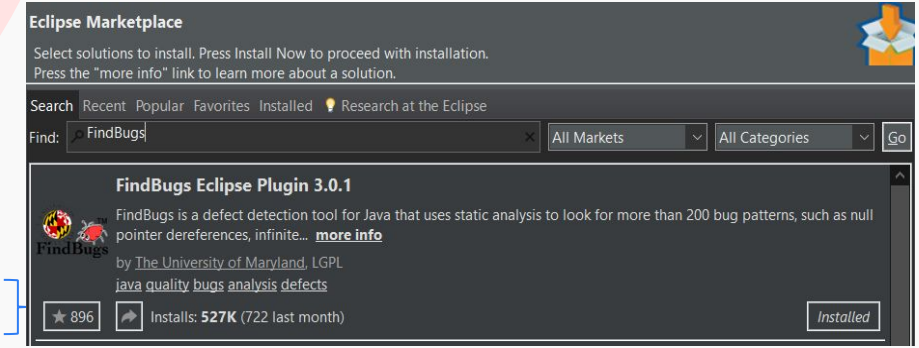
Testing Tools

FindBugs

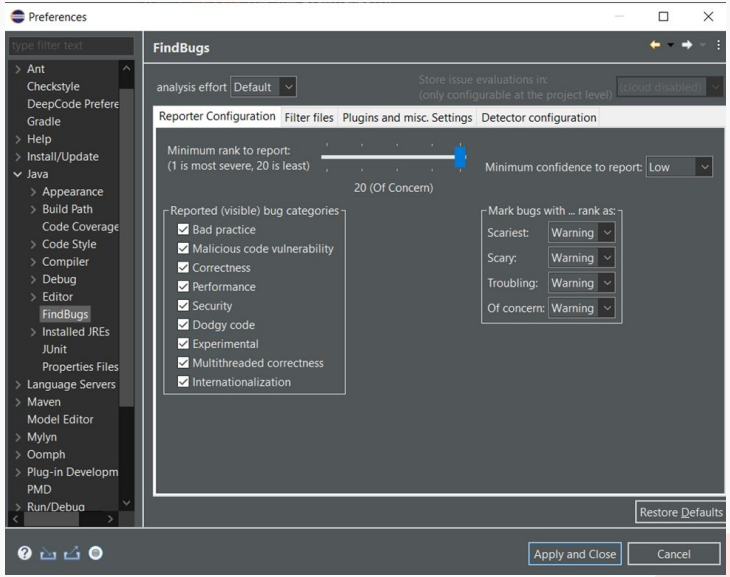
- Eclipse plugin
- Patterned based Static Analysis
- Detects
  - Runtime issues,
  - Memory leaks
  - Misuse of coding practices

Applying FindBugs

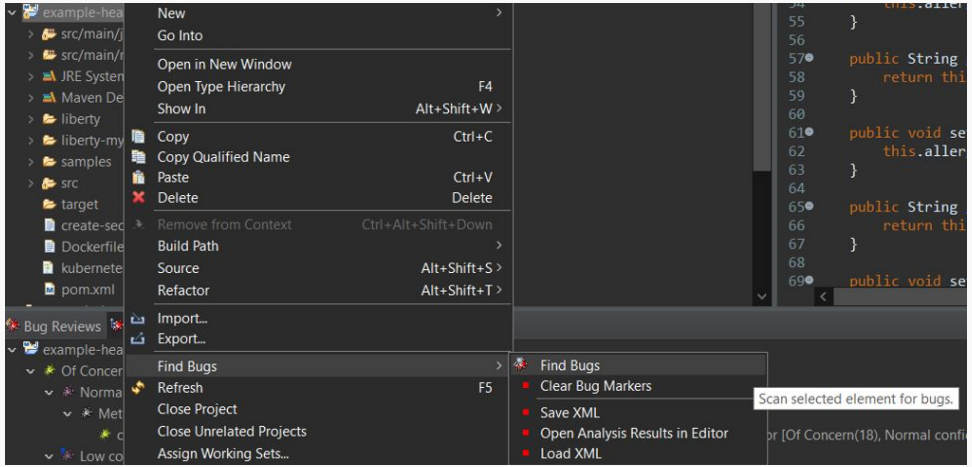
01



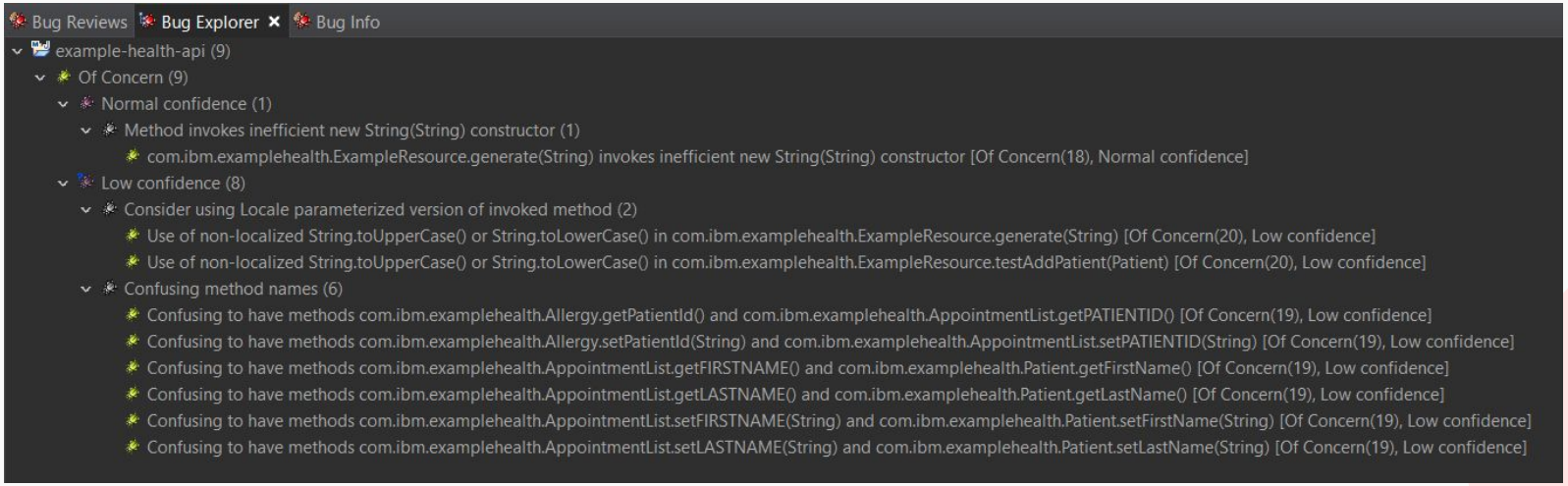
02



03


New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Find Bugs
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Find Bugs
Clear Bug Markers
Save XML
Open Analysis Results in Editor
Load XML
Scan selected element for bugs.
Bug Reviews
example-hea
src/main/
src/main/
JRE System
Maven De
liberty
liberty-my
samples
src
target
create-sec
Dockerfile
kubernete
pom.xml
Of Concer
Norma
Met
Low co
public String
return thi
public void se
this.aller
public String
return thi
public void se

04


Bug Reviews
Bug Explorer
Bug Info
example-health-api (9)
Of Concern (9)
Normal confidence (1)
Method invokes inefficient new String(String) constructor (1)
com.ibm.examplehealth.ExampleResource.generate(String) invokes inefficient new String(String) constructor [Of Concern(18), Normal confidence]
Low confidence (8)
Consider using Locale parameterized version of invoked method (2)
Use of non-localized String.toUpperCase() or String.toLowerCase() in com.ibm.examplehealth.ExampleResource.generate(String) [Of Concern(20), Low confidence]
Use of non-localized String.toUpperCase() or String.toLowerCase() in com.ibm.examplehealth.ExampleResource.testAddPatient(Patient) [Of Concern(20), Low confidence]
Confusing method names (6)
Confusing to have methods com.ibm.examplehealth.Allergy.getPatientId() and com.ibm.examplehealth.AppointmentList.getPATIENTID() [Of Concern(19), Low confidence]
Confusing to have methods com.ibm.examplehealth.Allergy.setPatientId(String) and com.ibm.examplehealth.AppointmentList.setPATIENTID(String) [Of Concern(19), Low confidence]
Confusing to have methods com.ibm.examplehealth.AppointmentList.getFIRSTNAME() and com.ibm.examplehealth.Patient.getFirstName() [Of Concern(19), Low confidence]
Confusing to have methods com.ibm.examplehealth.AppointmentList.getLASTNAME() and com.ibm.examplehealth.Patient.getLastName() [Of Concern(19), Low confidence]
Confusing to have methods com.ibm.examplehealth.AppointmentList.setFIRSTNAME(String) and com.ibm.examplehealth.Patient.setFirstName(String) [Of Concern(19), Low confidence]
Confusing to have methods com.ibm.examplehealth.AppointmentList.setLASTNAME(String) and com.ibm.examplehealth.Patient.setLastName(String) [Of Concern(19), Low confidence]

Testing Results: FindBugs

Project	HealthPlus	Healthcare-service	IBM-Openshift	StopCoding	Kardio
Number of bugs detected	42	4	9	12	10

Discussion of Detected Bugs: FindBugs

False Positive Rate

- Depending on the filter for the sensitivity and level of confidence used to display the found bugs, the false positive rate changes.
  - Setting the filter to show **all** bugs will generate more false positives.

Pros

- Easy to use and has a comprehensive filter
- Simplifies detecting bugs early on in development

Cons

- Manually checking false positives

Challenges Encountered

- For complex projects, manually determining false positives

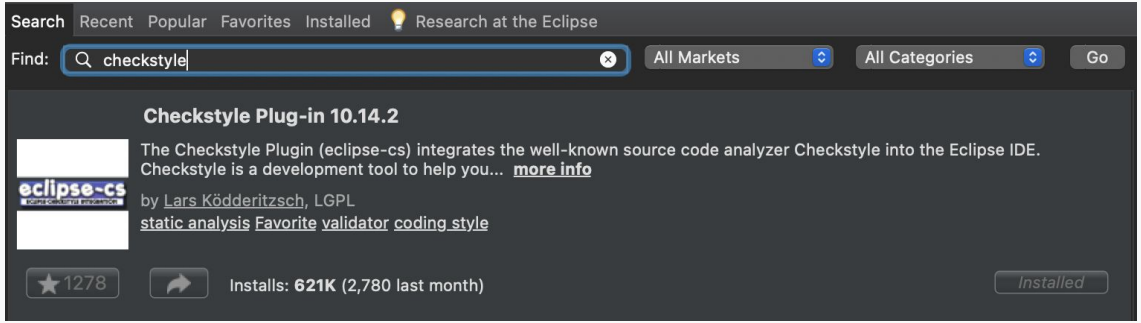
Checkstyle

- A static code analysis tool
- Checks Java code against the coding standards
- Provides feedback on - coding conventions, naming conventions, code style, and potential errors
- An open-source tool distributed under the Lesser General Public License

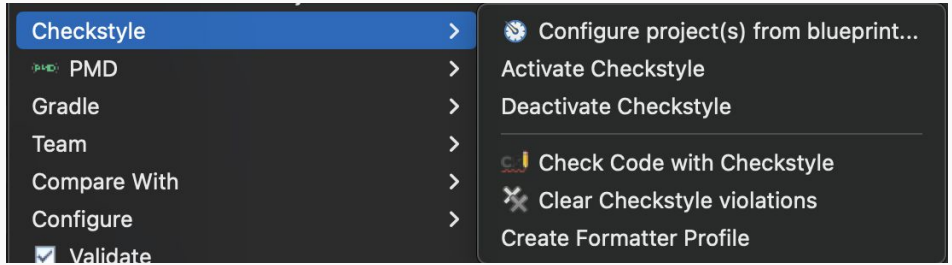
Applying CheckStyle

Supports various coding standards such as Sun Code Conventions, Google Java Style, and more.

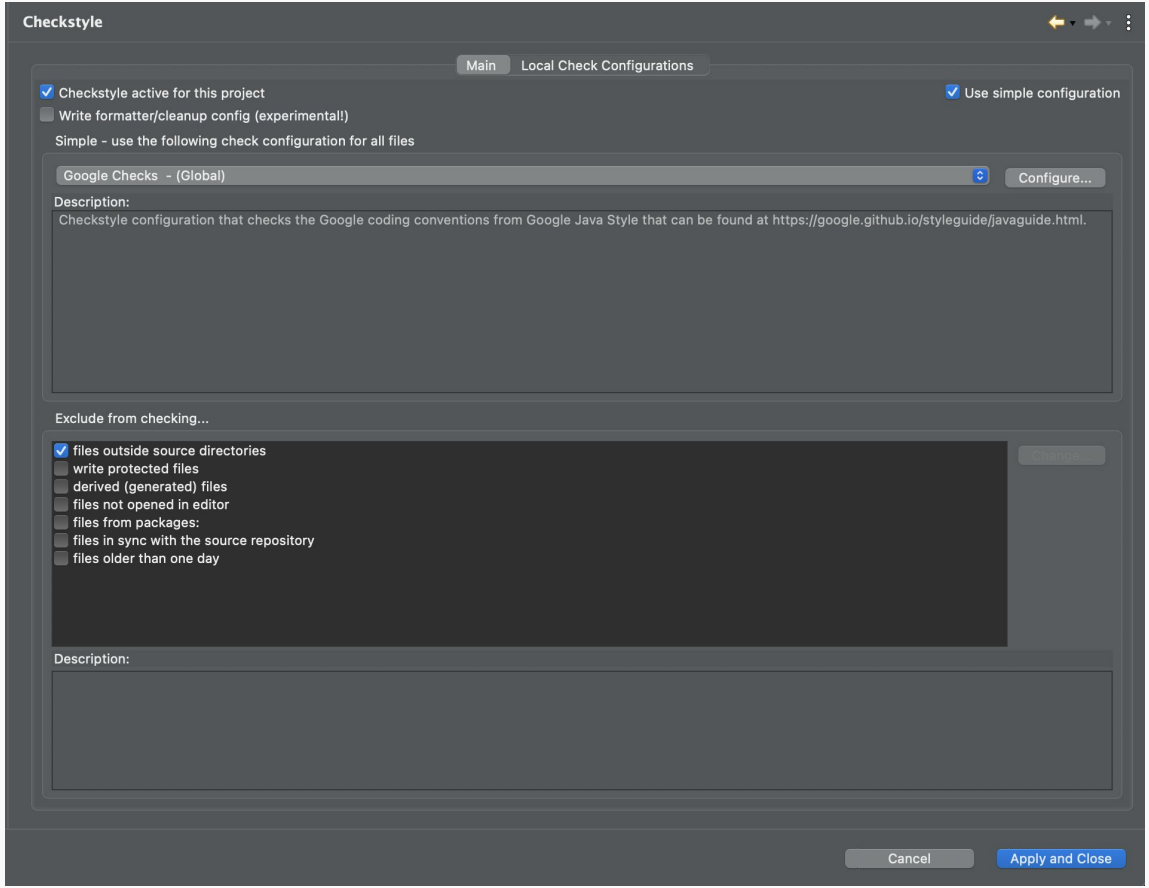
01



02



03


Checkstyle
Main
Local Check Configurations
Checkstyle active for this project
Use simple configuration
Write formatter/cleanup config (experimental!)
Simple - use the following check configuration for all files
Google Checks - (Global)
Configure...
Description:
Checkstyle configuration that checks the Google coding conventions from Google Java Style that can be found at https://google.github.io/styleguide/javaguide.html.
Exclude from checking...
files outside source directories
write protected files
derived (generated) files
files not opened in editor
files from packages:
files in sync with the source repository
files older than one day
Change...
Description:
Cancel
Apply and Close

Overview of Checkstyle violations - 256 markers in 10 categories (filter matched 256 of 256 items)

Checkstyle violation type	Occurrences
'X' has incorrect indentation level X, expected level shoul...	128
'X' is not followed by whitespace.	1
Using the '.*' form of import should be avoided - X.	2
Extra separation in import group before 'X'	6
Line is longer than X characters (found X).	6
'X' should be on a new line.	18
Only one statement per line allowed.	1
Missing a Javadoc comment.	13
'X' child has incorrect indentation level X, expected level s...	75
'X' construct must use '{}'s.	6

Testing Results: Checkstyle

Project	HealthPlus	Healthcare-service	IBM-Openshift	StopCoding	Kardio
Checkstyle Violations	27480	256	16124	46	43698

Discussion of Detected Bugs: Checkstyle

Pros of Checkstyle:

- Enforces consistent coding guidelines and standards
- Provides flexibility through customizable rule sets

Cons of Checkstyle:

- Requires configuration to align with project-specific coding standards, may not cover all types of bugs or issues

False Positive Rate:

- Due to project specific requirements, the possibility of generating false positives is low to moderate
- Fine-tuning rule sets, prioritizing high-impact checks can help reduce false positives

Challenges Encountered:

- Requires fine tuning and careful consideration to balance between false positives and false negatives
- Encourages programmers to use strict coding standards and practices which requires time to get used to

Randoop

- Automates Java unit test generation for bug detection.
- Uses random testing to explore program behavior comprehensively.
- Seamlessly integrates with JUnit for Java unit test generation.

Applying Randoop

To use Randoop, simply provide the Java class or classes you want to test as input, and Randoop will automatically generate a suite of JUnit tests based on the behavior it observes during execution exploration. Execute the generated tests within your Java development environment or build system to uncover potential bugs and ensure robustness in your codebase.

For example, for healthcare-service:

01

Problems Javadoc Declaration Terminal Bug Explorer Checkstyle violations

C:\WINDOWS\system32\cmd.exe

```
Microsoft Windows [Version 10.0.22631.3374]
(c) Microsoft Corporation. All rights reserved.

D:\UNIVERSITY\EECS4313\Project\Github Open Projects\healthcare-service>java -classpath ./target/classes/;./randoop-lib/randoop-all-4.2.1.jar;./randoop-lib/j
ackson-core-2.11.1.jar;./randoop-lib/jackson-databind-2.11.1.jar;./randoop-lib/jackson-annotations-2.11.1.jar randoop.main.Main gentests --classlist=myclass
list.txt --junit-output-dir=./randoop-tests --time-limit=100
```

02

myclasslist.txt

```
ru.netology.patient.Main
ru.netology.patient.entity.BloodPressure
ru.netology.patient.entity.HealthInfo
ru.netology.patient.entity.PatientInfo
ru.netology.patient.repository.PatientInfoFileRepository
ru.netology.patient.repository.PatientInfoRepository
ru.netology.patient.service.alert.SendAlertService
ru.netology.patient.service.alert.SendAlertServiceImpl
ru.netology.patient.service.medical.MedicalService
ru.netology.patient.service.medical.MedicalServiceImpl
```

03

04

Demo

Testing Results: Randoop

Project	No. of classes tested	Regression tests generated	Error-revealing tests generated
healthcare-service	10	2997	0
HealthPlus	14	1974	223
IBM-Openshift	15	2904	0
pg-index-health	1	39	0
healthcare-data-harmonization	2	21	0

Discussion of Detected Bugs: Randoop

Pros of Randoop:

- Saves time by automatically generating Java unit tests.
- Integrates smoothly with JUnit for easy adoption.

Cons of Randoop:

- Inability to control values supplied as arguments to tests (which are randomized), making it impossible to guarantee equivalence partition coverage on inputs.
- May yield false positives when dealing with complex program behaviors.

False Positive Rate:

- Randoop struggled with false positives in HealthPlus due to complex interactions with external libraries, highlighting the importance of considering dependencies in testing.

Challenges Encountered:

- Complex Project Detection: Projects with extensive external library usage, like HealthPlus, posed difficulties in bug detection, requiring careful consideration of dependencies and interactions.

05

Conclusion & Suggestions

Conclusion & Suggestions

- We identified common bugs present in our project and demonstrated the effectiveness of bug detection tools.
- Bug detection software helps us identify issues early, saving time and resources while ensuring software reliability and stability.
- While selecting bug detection tools, we must prioritize compatibility with our project, seamless integration into existing workflows and customization as needed to meet specific project requirements.

Conclusion & Suggestions

- Tools such as Randoop should prioritize enhanced dependency handling and adapt to diverse project environments.
- Tools must have regular updates to combat emerging issues.
- While using different tools modify the settings according to the project.
- Future development should focus on refining bug detection methodologies to address challenges like false positives and complex project structures.

References

- https://randoop.github.io/randoop/
- https://github.com/heshanera/HealthPlus/
- https://checkstyle.sourceforge.io/
- https://github.com/neee/healthcare-service
- https://findbugs.sourceforge.net/
- https://github.com/IBM/example-health-jee-openshift
- https://github.com/jogeen/StopCoding
- https://github.com/tmobile/kardio
- https://onlinelibrary.wiley.com/doi/full/10.1002/spe.3181
- https://www.baeldung.com/intro-to-findbugs
- https://www.baeldung.com/checkstyle-java

Questions?

Thank You