HireBuild: An Automatic Approach to History-Driven Repair of Build Scripts

Authors: Foyzul Hassan, Xiaoyin Wang

Presented by Xianhao Jin

Outline

1. Problem Statement
2. Background Knowledge
3. Approach
4. Evaluation
5. Related work
6. Conclusion
7. Discussion

Problem Statement

1. Build tools such as Maven and Gradle are popular.
2. They need maintenance as frequently as source code.
3. Existing work focus on repairing source code.
4. Reparing build scripts has unique challenges:
    a. involve open knowledge that do not exist in the current project
    b. no test suite
    c. the semantics of build scripts is very different from normal programs

Problem Statement - Example

Example 2 A Gradle Build Failure and Patch *(puniverse/quasar: Build Failure Version:017fa18, Build Fix Version:509cd40)*

```
Could not resolve all dependencies for
    configuration ':quasar-galaxy:compile'.
> A conflict was found between the following
    modules:
  - org.slf4j:slf4j-api:1.7.10
  - org.slf4j:slf4j-api:1.7.7
```

```
compile ("co.paralleluniverse:galaxy:1.4") {
    exclude group: 'com.lmax', module: 'disruptor'
    exclude group: 'de.javakaffee', module: 'kryo-
        serializers'
    exclude group: 'com.google.guava', module: '
        guava'
+ exclude group: "org.slf4j", module: '*'
      }
```

Difference:

1. Possible to find from existing scripts or past fixes that we need to perform an *exclude* operation, however, “org.slf4j” is hard to generate.
2. We are able to, and need to consider build-specific operations.
3. The build log information is very important and helpful.

Background Knowledge - Gradle

Gradle is an open-source build automation system that builds upon the concepts of Apache Ant and Apache Maven and introduces a Groovy-based domain-specific language (DSL) instead of the XML form used by Apache Maven for declaring the project configuration.
Gradle uses a directed acyclic graph ("DAG") to determine the order in which tasks can be run.
Gradle was designed for multi-project builds, which can grow to be quite large. It supports incremental builds by intelligently determining which parts of the build tree are up to date; any task dependent only on those parts does not need to be re-executed.

Approach - three steps

1. Log Similarity Calculation to Find Similar Fixes
2. Generation of Build-Fix Patterns
3. Generation and Validation of Concrete Patches

Approach - Step 1

1. Build Log Parsing (Error-and-exception part).
2. Text Processing.
3. Similarity Calculation.

```
* What went wrong:
A problem occurred evaluating project ':android-
    rest'.
>
Gradle version 1.9 is required. Current version is
     1.8. If using the gradle wrapper, try editing
     the distributionUrl in /home/travis/build/47
    deg/appsly-android-rest/gradle/wrapper/gradle-
    wrapper.properties to gradle-1.9-all.zip
```

Approach - Step 2

1. **Build-Script Differencing**
2. Hierarchical Build-Fix Patterns
3. Merging of Build-Fix Patterns
4. Ranking of Build-Fix Patterns

Example 3 Build Script Differencing Output *(BuildCraft/BuildCraft: 98f7196)*

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <patch>
3  <lineno id="30"><exp id="0">
4     <operation>Update</operation>
5     <nodetype>ConstantExpression</nodetype>
6     <nodeexp>1.7.2-10.12.1.1079</nodeexp>
7     <nodeparenttype>BinaryExpression</nodeparenttype>
8     <nodeparentexp>(version = 1.7.2-10.12.1.1079)
9     </nodeparentexp>
10    <nodeblockname>minecraft</nodeblockname>
11    <nodetaskname> </nodetaskname></exp>
12   </lineno>
13 </patch>
```

Approach - Step 2

1. Build-Script Differencing
2. **Hierarchical Build-Fix Patterns**
3. Merging of Build-Fix Patterns
4. Ranking of Build-Fix Patterns

Example 5 Gradle Build Fix *(BuildCraft/BuildCraft: 98f7196)*

```
- version = "1.7.2-10.12.1.1079"
+ version = "1.7.2-10.12.2.1121"
```

Example 6 Gradle Build Fix *(ForgeEssentials/ForgeEssentialsMain:fcbb468)*

```
-version = "1.4.0-beta7"
+version = "1.4.0-beta8"
```

Approach - Step 2

1. Build-Script Differencing
2. Hierarchical Build-Fix Patterns
3. **Merging of Build-Fix Patterns**
4. Ranking of Build-Fix Patterns

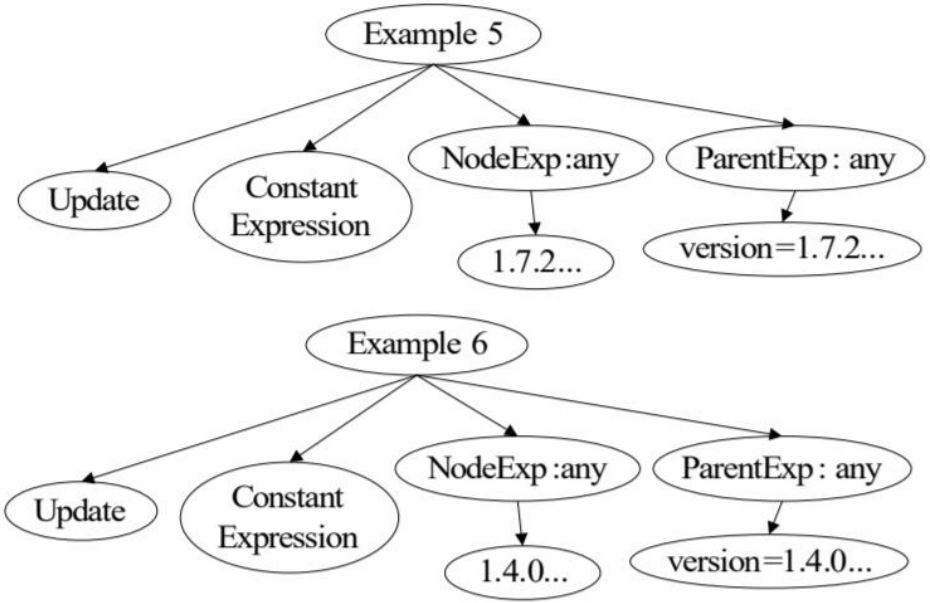

Figure 1: Hierarchies of Build-Fix Patterns

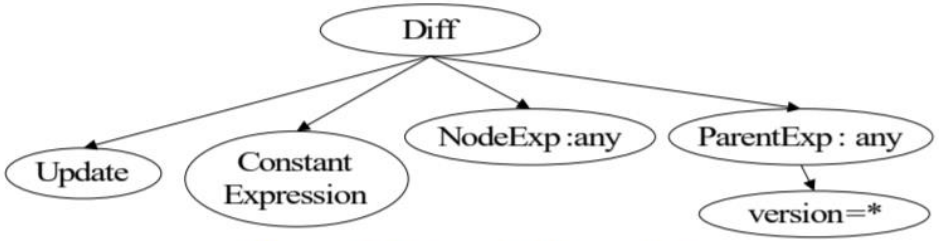

Figure 2: Merged Hierarchies

Approach - Step 2

1. Build-Script Differencing
2. Hierarchical Build-Fix Patterns
3. Merging of Build-Fix Patterns
4. **Ranking of Build-Fix Patterns**

$$P_\alpha = \frac{n_\alpha^t}{N}$$

Approach - Step 3

1. **Which file to apply**
2. Where in the file to apply
3. Determine the possible values of the abstract nodes
4. Ranking of generated patches
5. Patch application

Approach - Step 3

1. Which file to apply
2. **Where in the file to apply**
3. Determine the possible values of the abstract nodes
4. Ranking of generated patches
5. Patch application

Approach - Step 3

1. Which file to apply
2. Where in the file to apply
3. **Determine the possible values of the abstract nodes**
4. Ranking of generated patches
5. Patch application

Approach - Step 3

1. Which file to apply
2. Where in the file to apply
3. Determine the possible values of the abstract nodes
4. **Ranking of generated patches**
5. Patch application

Approach - Step 3

1. Which file to apply
2. Where in the file to apply
3. Determine the possible values of the abstract nodes
4. Ranking of generated patches
5. **Patch application**

Evaluation

1. Data Set : training set 135 + test set 40 (24 reproduced)
2. Research Questions:
    a. How many reproducible build failures in the evaluation set can HireBuild fix?
    b. How many patches HireBuild generated and tried during the build-failure fixing?
    c. What are the amount of time HireBuild spends to fix a build failure?
    d. What are the sizes of build fixes that can be successfully fixed and that can not be fixed?
    e. What are the reasons behind unsuccessful build-script repair?

Evaluation - Result

RQ1: Number of successfully fixed build failures.

Table 2: Project-wise Build Failure / Fix List

Project Name	#Failures	#Correctly Fixed
aol/micro-server	2	1
BuildCraft/BuildCraft	2	0
exteso/alf.io	1	1
facebook/rebound	1	1
griffon/griffon	1	0
/btrace	1	1
jMonkeyEngine/jmonkeyengine	2	0
jphp-compiler/jphp	1	0
Netflix/Hystrix	2	0
puniverse/quasar	6	2
RS485/LogisticsPipes	5	5
Total	24	11

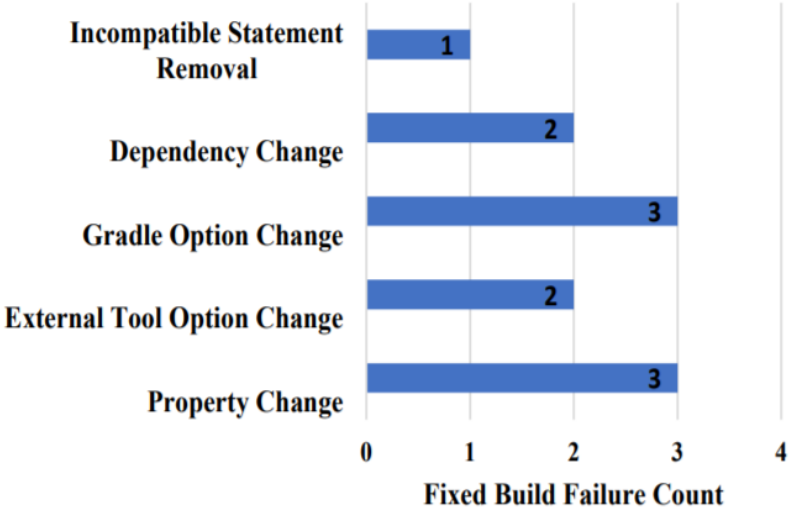

Figure 3: Breakdown of Build Fixes

Evaluation - Result

RQ2: Patch list size.

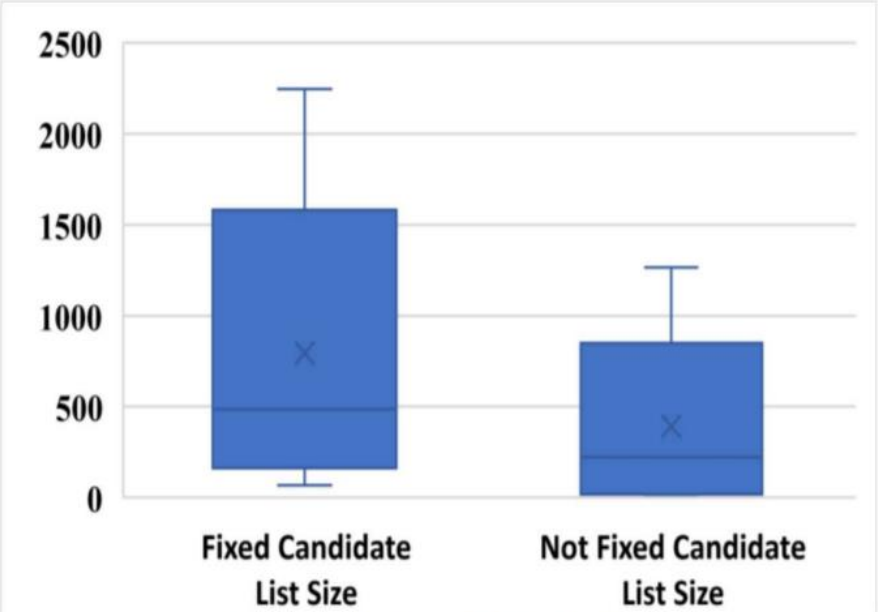

Figure 4: Patch List Sizes

Evaluation - Result

RQ3: Time Spent on Fixes.

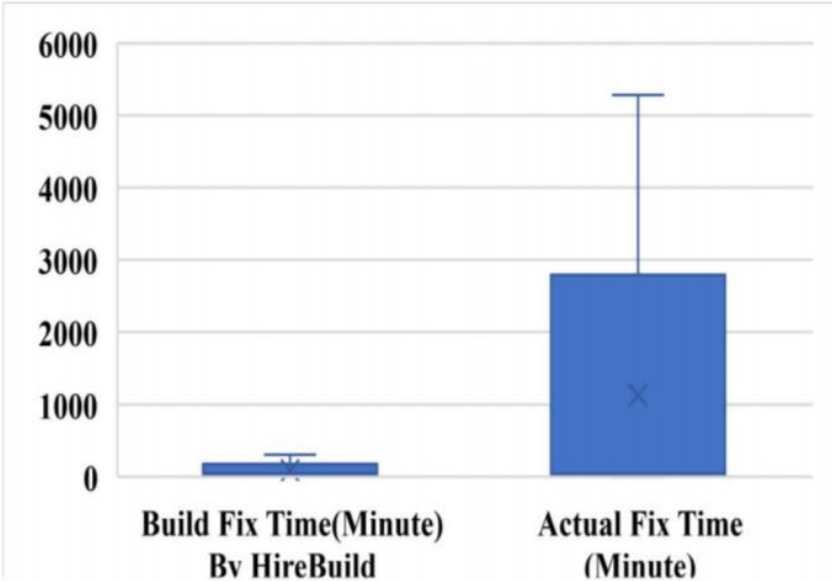

Figure 5: Amount of Time Required for Build Script Fix

Evaluation - Result

RQ4: Actual Fix Size.

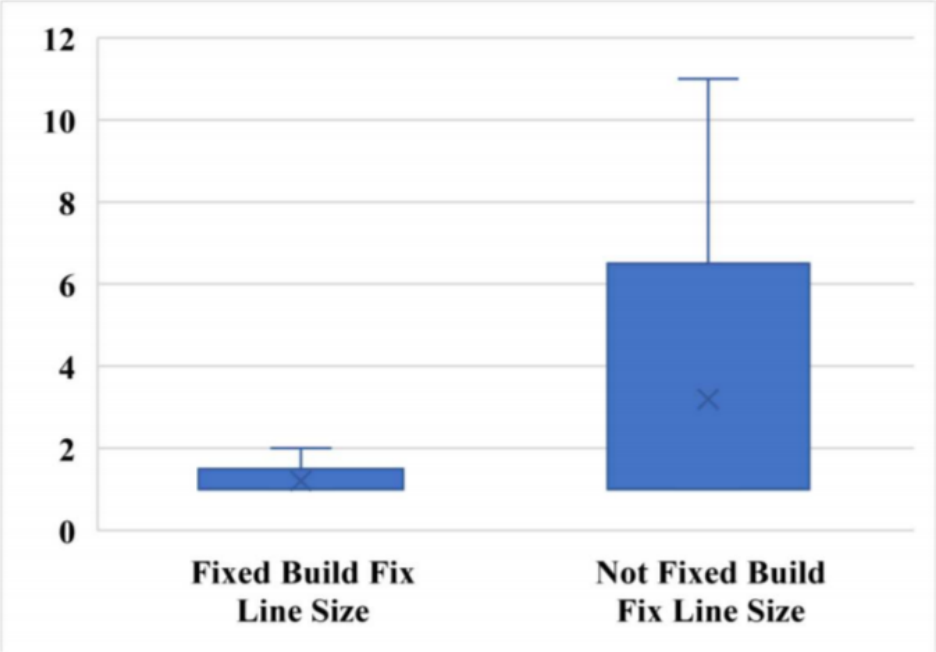

Figure 6: Actual Fix Sizes

Evaluation - Result

RQ5: Failing reasons for the rest 13 build failures.

Table 3: Cause of unsuccessful patch generation

Fix Type	#of Failures
Project specific change adaption	2(15%)
No matching patterns	6(46%)
Dependency resolution failures	3(23%)
Multi-location fixes	2(15%)

Related Work – Automatic Code Repair

2012 GenProg: A Generic Method for Automatic Software Repair.

2013 Automatic Patch Generation Learned from Human-written Patches.

2014 The Strength of Random Search on Automated Program Repair.

2015 Relifix: Automated Repair of Software Regressions.

2016 History Driven Program Repair.

2016 Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis.

Difference:

1. Applicable for build scripts.
2. Use build failure log similarity.
3. Fix candidate lists with reasonable size, with abstract fix template matching.

Related Work – Analysis of Build Files

2004 Dynamically Evolving Concurrent Information Systems Specification and Validation.

2007 Design recovery and maintenance of build systems.

2011 An empirical study of build maintenance effort.

2012 SYMake: A Build Code Analysis and Refactoring Tool for Makefiles.

2014 Fault Localization for Build Code Errors in Makefiles.

2015 GNU Autoconf - Creating Automatic Configuration Scripts.

Difference:

1. A different purpose (i.e., automatic software building).
2. Estimates run-time values of string variables with grammar-based string analysis.
3. Analyzes flows of files to identify the paths.

Conclusion

1. The first approach for automatic build fix candidate patch generation for Gradle build script.
2. Based on (1) build failure log similarity and historical build script fixes, (2) GradleDiff for AST level build script change identification, (3) a ranked list of patches.
3. Fix 11 out of 24 reproducible build failures

Conclusion - Contributions

1. A novel approach to automatic patch generation for repairing build scripts to resolve software build failures.
2. A dataset of 175 build fixes which can serve as the basis and a benchmark for future research.
3. An empirical evaluation of our approach on the dataset of 175 real-world build fixes.
4. An AST diff generation tool for Gradle build scripts.

Discussion

1. The popularity of the problem?
2. Where do selected patches come from? Within Project or Cross Project? How about the its ranking?
3. Fix time comparison?
4. How to deal with a build with multiple commits?
5. The threshold of 5? How about a ratio?
6. Complated failures?

Thank you!