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

Foyzul Hassan and Xiaoyin Wang
Presented by: Md Mahir Asef Kabir

Problem Background

- Travis CI [*] is one of the most popular Continuous Integration platform for Open-Source Software (OSS) development
- TravisTorrent [1] is a freely available data set based on Travis CI and GitHub that provides easy access to hundreds of thousands of analyzed builds from more than 1,000 projects containing 2,640,825 builds
- According to TravisTorrent [1] dataset, 29% code commits encounters project build failure on the integration server (**22% code commits modify build scripts**)
- Build failures can be caused by - 1) Faulty Source Code and 2) Faulty Build Scripts

[*] https://travis-ci.com/

Problem Statement

- Faulty Source Code issues have been dealt by automatic software patch generation by repairing source code
- Faulty Build Script fix requires repairing of build scripts. It has following challenges -
  - Requires knowledge that are not available elsewhere in the project (e.g. - Updating dependency to a newly available version)
  - Does not have test suite to facilitate fault localization
  - Semantics of build scripts is very different from normal programs
- Need a technique to automatically repair build scripts for fixing build failures

Solution Background

- Build failures provide richer log information that normal test failures (See image)
- Many Build failures are not project specific, so cross-project solutions can be provided

```
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
```

A Gradle Build Failure

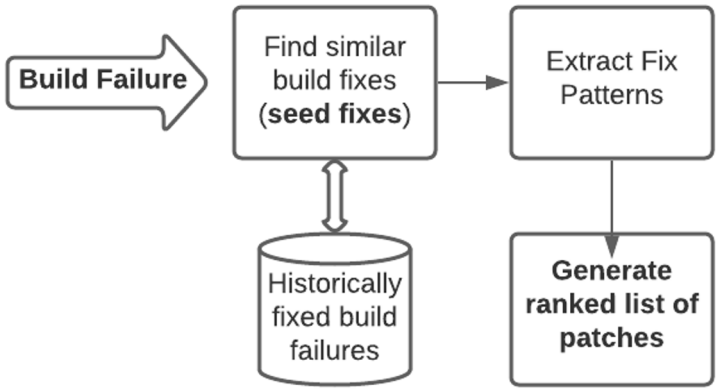
Solution Assumptions

- Gradle is the most promising build tool, because according to recent statistics, 50% of the top GitHub apps use Gradle [2]
- Projects using Gradle will produce similar build logs for similar build failures
- It is possible to locate similar build failure logs and their fixes from build-fix datasets
- Based on previous fixes of the build failures, new fixes can be generated for similar new build failures

Solution Sketch

Proposed a novel approach: **HireBuild**

- Input: Build Failure of current project
- Output: Ranked list of patch candidates
- Used dataset: TravisTorrent [1]
- Notes
  - Ignore CE and Unit-test failures
  - Only consider Build Failures for now



How Gradle works

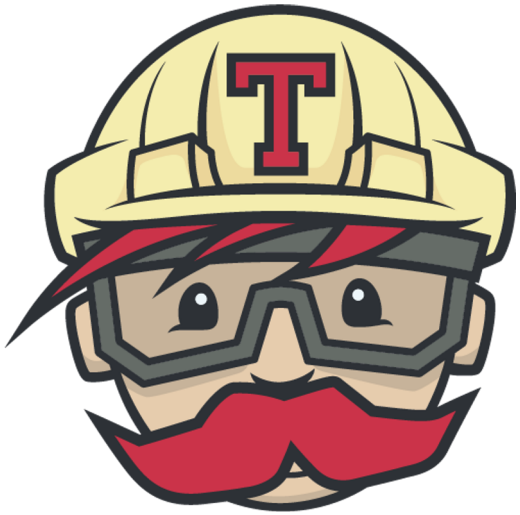
- Gradle is a build automation tool
- It supports automatic download and **configuration of dependencies**
- A Gradle build may consist of one or more build projects
- Each project **consists of some tasks**
- Gradle describes its build process in **build.gradle** file located typically in root

```
...

repositories {
  mavenCentral()
}

dependencies {
  implementation 'com.package.class:library:version'
}

task taskName {
    doLast {
        println 'Task output'
    }
}

...
```

Dataset

- From TravisTorrent [1] dataset, fetch commits having
  - Fail/Error status in immediate previous commit
  - Success status in the current commit
  - Changes **only in Gradle build scripts**

- Extracted 175 commits (comes from 54 projects)
- Used 135 commits for training & 40 for evaluation



[*] https://travis-ci.com/

Approach

Build Log Parsing

- Point of interest in **error-and-exception** part
- Extracts portion after "*** What went wrong:**"
- Only considers the last error (as that is the one halting build)
- Removes noisy stack trace

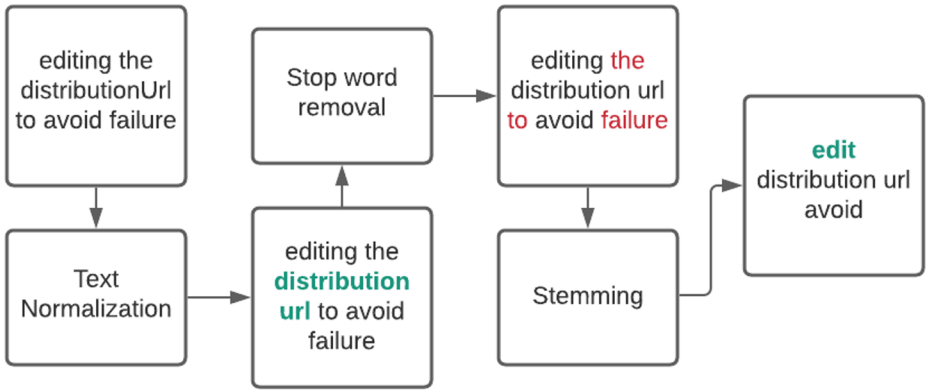
```
* What went wrong:
A problem occurred evaluating project ':android-
    rest'.
>
Gradle version 1.9 is required. Current version is
     1.8. If using the gradle wrapper, try editing
     the distributionUrl in /home/travis/build/47
    deg/appsly-android-rest/gradle/wrapper/gradle-
    wrapper.properties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

Sample Build Log

Text Processing

Input text: “editing the distributionUrl to avoid failure”

Output text: “edit distribution url avoid”



Similarity Calculation

- Term Frequency-Inverse Document Frequency (TF-IDF) [3] to weight all words
  - Given a document collection D, a word w, and an individual document d ϵ D

$$w_d = f_{w,\,d} * \log\left(|D|/f_{w,\,D}\right)$$

where fw, d equals the number of times w appears in d, |D| is the size of the corpus, and fw, D equals the number of documents in which w appears in D

Similarity Calculation (Contd.)

- Cosine Similarity to find most similar historical fixes
  - Cosine similarity is a measure of similarity that can be used to compare documents. Let x and y be two vectors for comparison. The formula is -

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||}$$

Where, x = (x1, x2, …, xn). Here **xi** is prepared from TL-IDF weight of word **wi**

Build-Script Differencing

Uses Modified GumTree [4] tree difference algorithm on AST of two commits

```
29      minecraft {
-           version = "1.7.2-10.12.1.1079"
30  +       version = "1.7.2-10.12.2.1121"
```

Where is the updated version in output?

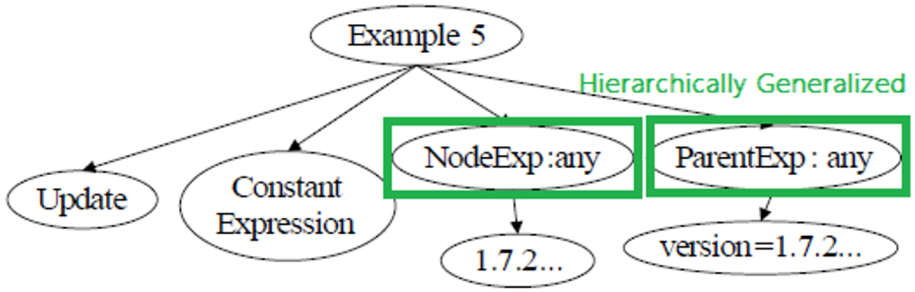
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<patch>
<lineno id="30"><exp id="0">
   <operation>Update</operation>
   <nodetype>ConstantExpression</nodetype>
   <nodeexp>1.7.2-10.12.1.1079</nodeexp>
   <nodeparenttype>BinaryExpression</nodeparenttype>
   <nodeparentexp>(version = 1.7.2-10.12.1.1079)
   </nodeparentexp>
   <nodeblockname>minecraft</nodeblockname>
   <nodetaskname> </nodetaskname></exp>
 </lineno>
</patch>
```

[*] https://github.com/BuildCraft/BuildCraft/commit/98f7196

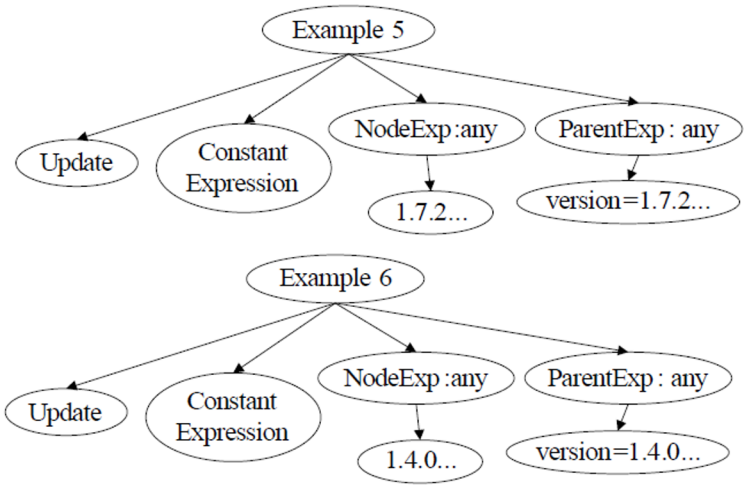
Hierarchical Build-Fix Patterns

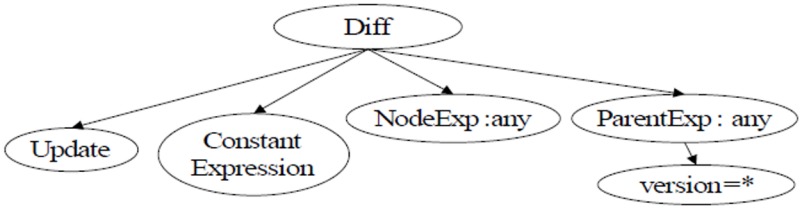
“ParentExp: any” means updating the value 1.7.2 without considering its parent

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



Merging of Build-Fix Patterns





Ranking of Build-Fix Patterns

- HireBuild ranks build-fix patterns according to the frequency
- For each build-fix pattern α, we counted n_α^t: α's frequency among seed fixes. Then probability of α is as follows.

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

- N is the total occurrences of build-fix patterns
- Rank based on probability (If P**a** == P**b**, then **b**>**a** if **a** is a generalized version of **b**)

Generation & Validation of Concrete Patches

- Which .gradle file to use for applying the fix?
  - If .gradle file names are mentioned in build log, take the first mentioned file
  - If no .gradle file is mentioned in log, take the build.gradle file of root folder

- Given Input: A Build-fix pattern to try, buggy Gradle build script
  HireBuild task:
  - Parse the buggy Gradle build script to AST
  - Find the location where the patch can be applied

Generation & Validation of Concrete Patches(Contd.)

- Main types of code differences are - Updates, Deletions, Insertions
- Updates & Deletions:
  - Pattern: “update constant expression with parent expression **version=*”**
  - Task: Match the pattern to a ConstantExpression type node in AST where parent expression node has a value matching **version=***
  - If pattern can be mapped to multiple AST Nodes, generate patch for all
  - If pattern can be mapped to multiple AST Nodes in same block, generate patch for the first mapped node

Generation & Validation of Concrete Patches(Contd.)

- Insertions: Insert the patch at the end of a matching task or block
- Patch Generation needs to figure out 3 main things
  - Which build-fix pattern to apply? (Done)
  - Where to apply the pattern? (Done)
  - What values to use for the abstract parts (**concrete fix**)? (Yet to do)
    - E.g. - What version to use in place of version=*?
  - The most commonly used values in build scripts are - Identifiers, Names of plug-ins/libraries/tools, File Paths, Version Numbers

Generating values for commonly used value types

- Identifiers
  - Considers identifiers in the concrete seed fixes and fix location
- Names of plug-ins/libraries/tools
  - Considers names appearing in concrete seeds, build log & buggy build script
- File Paths
  - Considers paths appearing in concrete seeds, build log & buggy build script
- Version Numbers
  - Searches Gradle central repository for all existing version numbers

Ranking of Generated Patches

Patches are ranked based on priority value. Priority values are set in the following way:

- Initial priority of a patch is the probability of its build-fix pattern
- If mapped location L has same task/block name as merged seed fixes A or B, then add 1.0 to priority value
- If patch involves a value which appears in build log, then add 1.0 to priority value
- Rank all patches with updated priorities

Patch Application

- Apply patch one by one until failure is fixed or timeout threshold is reached
- Consider failure to be fixed if
  - Build process returns 0 and build log shows success
  - All source files are compiled
- HireBuild generally focuses on one line fixes
- Considers multi-line patches after applying all single line patches

Evaluation

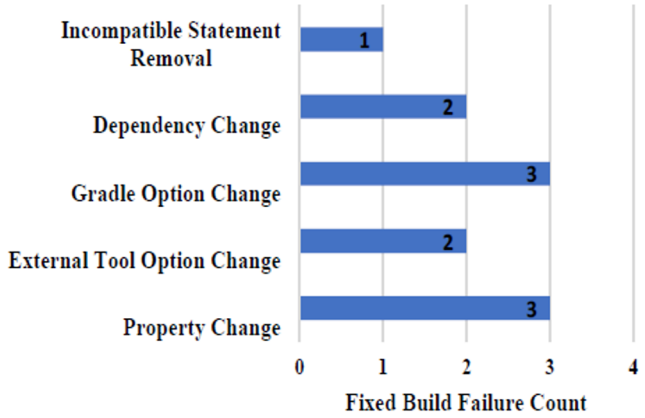
Research Questions (RQs)

- RQ1 How many reproducible build failures in the evaluation set can HireBuild fix?
- RQ2 How many patches HireBuild generated and tried during the build-failure fixing?
- RQ3 What are the amount of time HireBuild spends to fix a build failure?
- RQ4 What are the sizes of build fixes that can be successfully fixed and that cannot be fixed?
- RQ5 What are the reasons behind unsuccessful build-script repair?

Results of RQs - RQ1

How many reproducible build failures in the evaluation set can HireBuild fix?
Among 24 reproducible build failures, HireBuild can generate correct fix for 11 of them

The table shows breakdown of successful generation of build fixes according to the type of changes to make

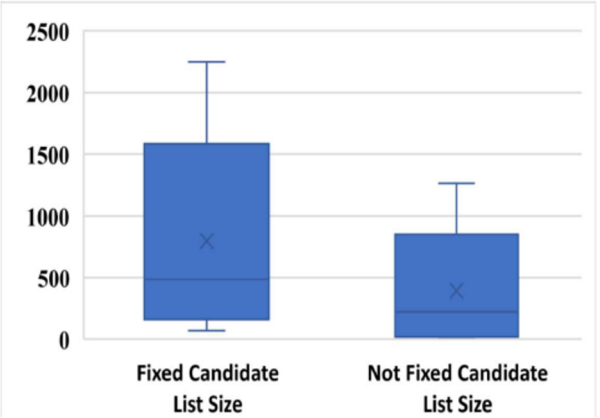


Results of RQs - RQ2

How many patches HireBuild generated and tried during the build-failure fixing?
Patch list size is the number of ranked patches generated by HireBuild.

Minimum, Median and Maximum Patch list size for fixed builds are 68, 486, 2245

For failed builds, the stats for Minimum, Median and Maximum are 8, 223, 1266

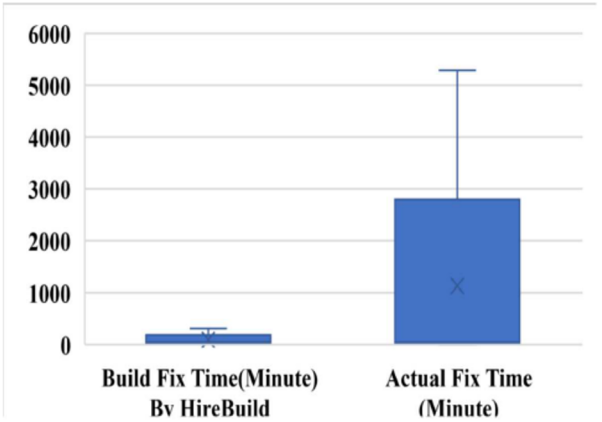


Results of RQs - RQ3

What are the amount of time HireBuild spends to fix a build failure?
Time spent is the manual time difference between failure commit and fix commit

Minimum, Median and Maximum time spent for fixed builds are 2, 44, 305 minutes

For failed builds, the stats for Minimum, Median and Maximum are 2, 42, 5281 minutes

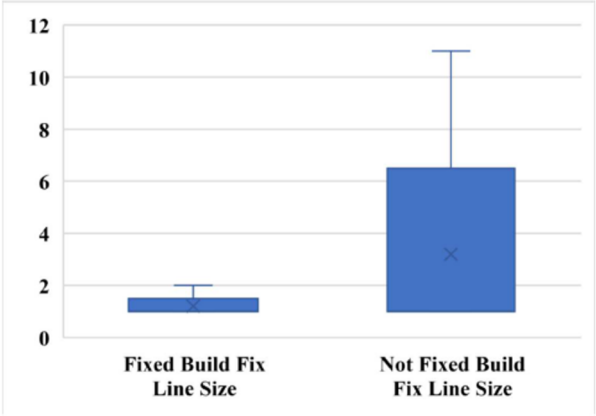


Results of RQs - RQ4

What are the sizes of build fixes that can be successfully fixed and that cannot be fixed?
Fix size is the lines of changes made to fix build

Minimum, Median and Maximum fix size
for fixed builds are 1, 1, 2

For failed builds, the stats for Minimum,
Median and Maximum are 1, 1, 11
So the approach works better with small changes



Results of RQs - RQ5

What are the reasons behind unsuccessful build-script repair?
Four major reasons behind **13** unsuccessful build-script repair -

1. Project specific change adaption: 2 (15%)
2. Non-matching patterns: 6 (46%)
3. Dependency resolution failures: 3 (23%)
4. Multi-Location Fixes: 2 (15%)

Threats to Validity

Internal Validity:

- There can be mistakes in the data processing and bugs in HireBuild
- Manual fixes used as ground truth can have flaws
- Fix time may not be accurate

External Validity:

- Limited number of reproducible fixes
- Only considers Gradle build tool and Gradle-only changes

Related Works

- Automatic Program Repair
  - Weimer et al. [5] introduced GenProg which uses genetic programming for automatic patch generation
  - Pattern-based Automatic Program Repair [6] uses manually generated templates from humans to generate patch
  - Relifix [7] takes advantage of version history information to repair fault

Novelty of this paper: 1) Applicable for build scripts 2) Generate fix templates using log similarity 3) Can generate fix candidate list with reasonable size

Related Works (Contd.)

- Analysis of Build Configuration File
  - Adams et al. [8] proposed a framework to extract a dependency graph for build configuration files and provide automatic tool
  - McIntosh et al. [9] carried out an empirical study on the efforts of the developers for building project configuration
  - Hyunmin et al. [10] conducted empirical study to categorize build errors in Google

Summary

- First approach to propose automatic build fix candidate patch generation for Gradle Build Script
- The solution works on automatic build fix template generation based on build failure log similarity and historical build script fixes
- Succeeded in fixing 11 out of 24 reproducible build failures among 40 build failures of evaluation dataset
- In future, the authors are planning to increase dataset and apply genetic programming for better ranking of generated patches

References

[1] Beller, M., Gousios, G., Zaidman, A.: Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In: Proceedings of the 14th working conference on mining software repositories (2017)
[2] Sulír, M., Porubän, J.: A quantitative study of java software buildability. In: Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools. pp. 17–25. PLATEAU 2016, ACM, New York, NY, USA (2016), http://doi.acm.org.libweb.lib.utsa.edu/10.1145/3001878.3001882
[3] Ramos, J., et al.: Using tf-idf to determine word relevance in document queries. In: Proceedings of the first instructional conference on machine learning (2003)

References (Contd.)

[4] Falleri, J.R., Morandat, F., Blanc, X., Martinez, M., Monperrus, M.: Fine-grained and accurate source code differencing. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. pp. 313–324. ASE ’14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2642937. 2642982
[5] Goues, C.L., Nguyen, T., Forrest, S., Weimer, W.: Genprog: A generic method for automatic software repair. IEEE Transactions on Software Engineering 38(1), 54–72 (Jan 2012)
[6] Kim, D., Nam, J., Song, J., Kim, S.: Automatic patch generation learned from human-written patches. In: Proceedings of the 2013 International Conference on Software Engineering. pp. 802–811. ICSE ’13, IEEE Press, Piscataway, NJ, USA (2013), http://dl.acm.org/citation.cfm?id=2486788.2486893

References (Contd.)

[7] Tan, S.H., Roychoudhury, A.: Relifix: Automated repair of software regressions. In: International Conference on Software Engineering (2015)
[8] Adams, B., Tromp, H., De Schutter, K., De Meuter, W.: Design recovery and maintenance of build systems. In: Software Maintenance, 2007. ICSM 2007. IEEE International Conference on. pp. 114–123 (Oct 2007)
[9] McIntosh, S., Adams, B., Nguyen, T., Kamei, Y., Hassan, A.: An empirical study of build maintenance effort. In: Software Engineering (ICSE), 2011 33rd International Conference on. pp. 141–150 (May 2011)
[10] Seo, H., Sadowski, C., Elbaum, S., Aftandilian, E., Bowdidge, R.: Programmers' build errors: A case study (at google). In: Proceedings of ICSE. pp. 724–734(2014)

Discussion

- How much comfortable would you feel using an automated tool that can modify your build script in production server?
- What needs to be done differently to create a HireBuild for other languages like JavaScript?
- Which unsuccessful patch generations can be fixed and how?

Thank you