Automatic Building of Java Projects in Software Repositories: A Study on Feasibility and Challenges

Foyzul Hassan*, Shaikh Mostafa, Edmund S. L. Lam, Xiaoyin Wang
Why Automatic Building?

Necessity of large corpus of built software
1. Large Scale Program Analysis.
   I. Points-to analysis.
   II. Call-graph generation.
   III. Dependency Analysis. etc.
2. Mining of software artifacts.

Single Project Build Requires to follow lot of steps. If we want to build 1000 projects, Bang!
Study Subjects

Top 200 Java Projects Based on Popularity

Build Systems We Considered
How We Can Build Automatically?

Running Default Build Command

- **Ant**
  - ant build

- **Maven**
  - mvn compile

- **Gradle**
  - gradle build
Build With Default Build Command

99 of 200 top Java projects cannot be built successfully with default build commands.
Other Study Vs Our Study

$ gradle build -x test --daemon
:compileJava

FAILURE: Build failed with an exception.

* What went wrong:
Could not resolve all dependencies for configuration ':compile'.
> Could not find mysql-connector-java-bin.jar (mysql:mysql-connector-java:5.1.16).
  > Could not find mysql-connector-java-bin.jar in central (https://repo.maven.apache.org/maven2)

Missing Dependency

- Missing JAR
- Environment Issue
- Wrong Build Option
- ....
Our Study

Our Study "Automatic Building of Java Projects in Software Repositories: A Study on Feasibility and Challenges" focuses on

**Build Failure Analysis**

I. Performed detailed manual analysis and building to find out and confirm the root causes of the build failures.

II. Build Failure Hierarchy based on manual analysis.

**Build Failure Fix Analysis**

I. Manual analysis on how build failures fix information can be extracted.

II. Feasibility of Automatic Building.
Overview of Our Study

- Running Default Build Command
- Manual Build Failure Analysis
- Failure Root Cause Information Extraction
- Feasibility of Automatic Build Failure Resolution
Study Design

**RQ1:** What proportion of top Java projects can be successfully built with default build commands of popular build tools?

**RQ2:** What are the major root causes of the observed build failures?

**RQ3:** How easily can root causes of build failures be identified from readme files and build failure logs?

**RQ4:** What proportion of build failures can be (or have the potential to be) automatically resolved?
**RQ1**: Build Status With Default Build Command

![Build Status Chart]

- **Maven**
  - Projects Built Successful: 40
  - Projects Failed: 25
  - Other/No Source Code: 13

- **Gradle**
  - Projects Built Successful: 53
  - Projects Failed: 56
  - Other/No Source Code: 13

- **Ant**
  - Projects Built Successful: 8
  - Projects Failed: 5
  - Other/No Source Code: 2

- **Other Build System**
  - Projects Built Successful: 11
  - Projects Failed: 11
  - Other/No Source Code: 2

- **No Source Code**
  - Projects Built Successful: 2
  - Projects Failed: 2
  - Other/No Source Code: 2

- **All**
  - Projects Built Successful: 101
  - Projects Failed: 86
  - Other/No Source Code: 13
RQ2: Major root causes of the build failures?

We classify the build failures to 3 general categories: environment issues, process issues, and project issues.

• Environment Issues
  Environment issues are build failures caused by the change of building environment.

• Process Issues
  Process Issues are build failures caused by the requirement of additional steps in the building process.

• Project Issues
  Project issues are build failures caused by defects in the project itself.
Environment Issues Build Failures

**Platform Version Issue**

(elasticSearch/elasticSearch:42a7a55)

A problem occurred evaluating root project ‘buildSrc’.
> Gradle 2.13 is required to build elasticsearch

**External Tools Issue**

(gocd/gocd: a3f77f9)

Execution failed for task ’:installers: agentPackageDeb’.
> A problem occurred starting process ‘command ’fpm”

**Removed Dependency Issue**

(Yalantis/Phoenix:188f2ec)

Searched in the following locations:
https://repo1.maven.org/maven2/com/android/tools/build/gradle/2.0.0-alpha1/gradle-2.0.0-alpha1.pom
Build Failure Hierarchy

Build Failures: 91

Environment Issues: 31
- Platform Version: 19
- Removed Dependency
  - Java: 9
  - 6
  - 2
  - 2
  - Maven
  - Android
  - Gradle
  - External Tools
- Prop: 11
  - 2
  - KeyStore
  - Options

Process Issues: 46
- Require File Setup: 13
- Non-default build command: 33
  - Config Version Conflict
    - Hard Code Path: 1
    - Incomplete Upload: 1
  - Target: 27
    - Compilation Error
  - Options

Project Issues: 14
- 9
- 3
Process Issues

Non-default Build Command.
Expecting
mvn clean install -P 'guice'

Exception in thread "pool-1-thread-1" java.lang.NoClassDefFoundError:
org/eclipse/aether/spi/connector/Transfer$State
at org.eclipse.aether.connector.wagon.WagonRepositoryConnector$GetTask.run(WagonRepositoryConnector.java:608)

Require File Setup Issue

A problem was found with the configuration of task ':android:packageDebug'.
> File '/home/~/google_iosched/android/debug.keystore' specified for property 'signingConfig.storeFile' does not exist.
Failed to apply plugin [id ‘com.android.application’]
Gradle version 2.2 is required. Current version is 2.1. If using the gradle wrapper, try editing the distributionUrl in ~/gradle/wrapper/gradle-wrapper.properties to gradle-2.2-all.zip

Compilation failed; see the compiler error output for details.
A problem occurred configuring project ‘doicommon’.
> The SDK directory '/home/~/153-singwhatiwanna_dynamic-load-apk/
DynamicLoadApk/D:\adt-bundle-windowsx86_64-20130219\sdk' does not exist.

internal.ExecException: Process 'command '/home/~/android-sdk-linux/build-tools/21.1.2/aapt'' finished with non-zero exit value 1
RQ3: Identifying Build Failure Cause

- Readme File
- Build Log
Most of the Non-Default Command mentioned in Readme

Dependency Missing Failures can be identified from Build Log
RQ4: Automatic Resolution of Build Failures

- Build Command Extraction and Prediction
- Version Reverting
- Dummy File Generation
# Building

To build this project, first time you try to build you need to run this (requires Apache Ant 1.8 or higher and JDK 1.6):

```
ant -f updat_dependencies.xml
```

which will setup the dependencies on intellij-core: is a part of command line compiler and contains only necessary APIs. idea-full: is a full blown IntelliJ IDEA Community Edition to be used in former plugin module. Then, you need to run

```
ant -f build.xml
```
Build Command Prediction

• *Readme Files are not there!!!*

• We can find which target is more like the correct building target by calculating the similarity between the target name and all the extracted commands in our training set of 857.

<table>
<thead>
<tr>
<th>gradle tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build tasks</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>assemble - Assembles all variants of all applications and secondary packages.</td>
</tr>
<tr>
<td>assembleAndroidTest - Assembles all the Test applications.</td>
</tr>
<tr>
<td>assembleDebug - Assembles all Debug builds.</td>
</tr>
<tr>
<td>assembleRelease - Assembles all Release builds.</td>
</tr>
<tr>
<td>....</td>
</tr>
</tbody>
</table>
Resolved Build Failures By Extraction and Estimation

![Bar chart showing resolved build failures by extraction and estimation. The chart compares different build systems including Maven, Gradle, and Ant. Different colors represent NER, Target Estimation, All fixed, and Not Fixed. The chart highlights Target Estimation for Gradle.]
Version Reverting

• Many build failure happens due to incompatible SDK and build tools.

• Straightforward way to resolve SDK and build tools dependency is to revert the versions of SDK and build tools from the latest version.
### Resolved Build Failures By Version Reverting

<table>
<thead>
<tr>
<th>Version Range</th>
<th>Java</th>
<th>Maven</th>
<th>Gradle</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revert 1 version</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Revert 2-5 versions</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Revert 6-10 versions</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Revert 11+ versions</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Max # reverted versions</td>
<td>10</td>
<td>9</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Avg # reverted versions</td>
<td>7.3</td>
<td>6.5</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

- Java
- Maven
- Gradle
- Android
Dummy File Generation

• In many projects, a sample local file (e.g., local.property.example) is provided, and users can refer to it for what to be put into the local file.

• We find that, simply generating an empty local file will resolve 7 of the 13 require file setup build failures, and renaming the sample local file back will resolve 1 additional build failures.
Build Failures Not Yet Analyzed

• Dependency Failure: Potential solution is to search for references to the Jar file in other projects’ configuration files.

• Config Version Conflict failures: We need to perform in-depth analysis of config files and their dependencies.
Lesson Learned

• It is a necessity.
  • Half of the top Java projects cannot be straightforwardly built with default build commands.

• It is feasible.
  • Among the 86 projects with build failures, 52 projects can be built successfully with different approach such as build command extraction and estimation, version reverting etc.

• The challenges.
  • Our study has also identified several build failure categories whose automatic resolution can be difficult.
Thank You