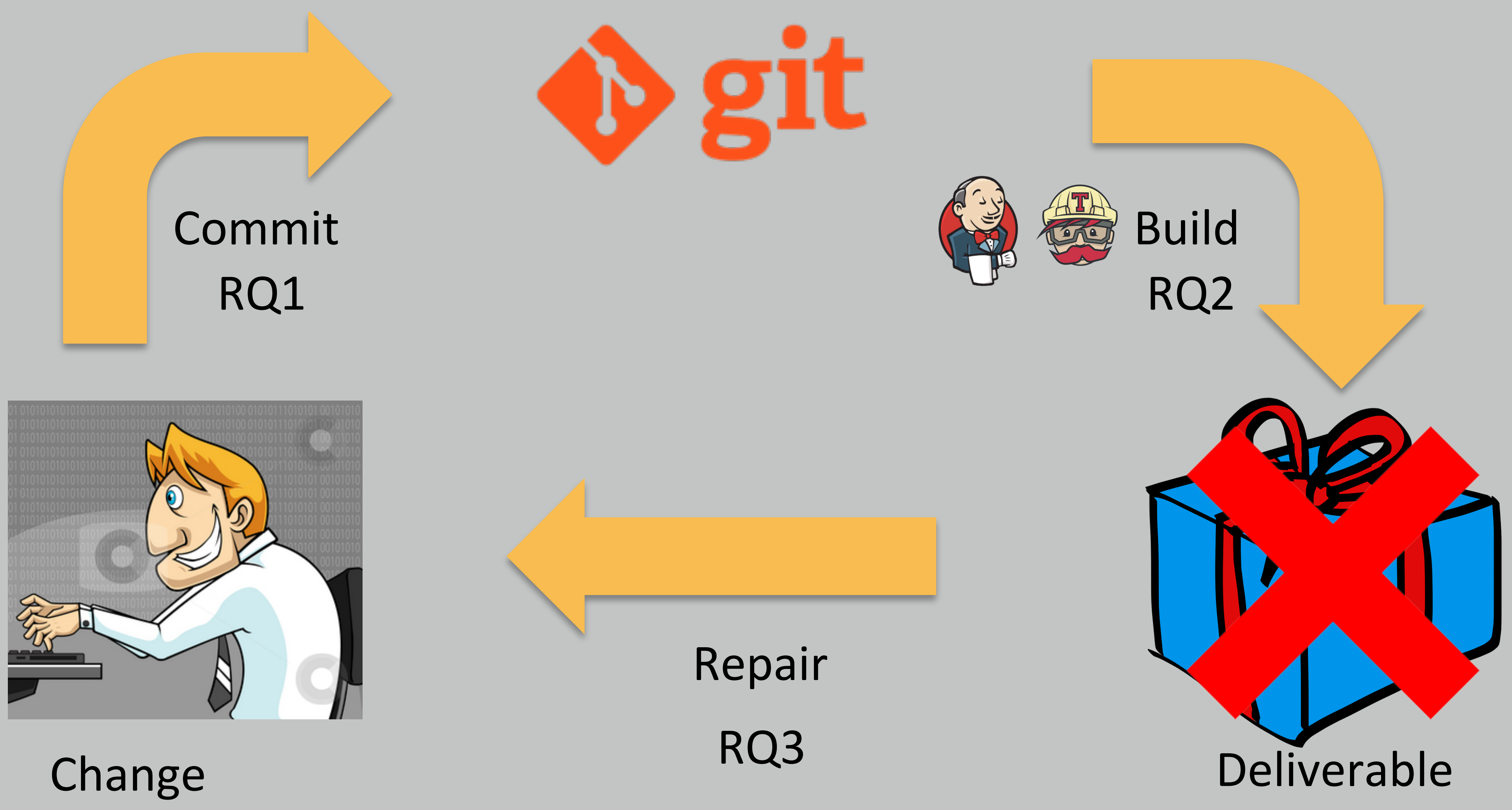
Preventing and Repairing Build Breakage

Christian Macho
Software Engineering Research Group, University of Klagenfurt

Problem Statement

- Builds cause a (hidden) overhead for teams [2] due to additional maintenance effort
- Neglected build maintenance is the main reason for build breakage [5]
- Build breakage prevents teams from continuing development and is expensive for companies [1]



(RQ1) What are the reasons for and characteristics of build breakage and fixes?
(RQ2) To what extent can we predict build breakage?
(RQ3) To what extent can we automatically refactor breakage-prone build configurations and repair broken builds?

Reasons and Characteristics of Build Breakage (RQ1)

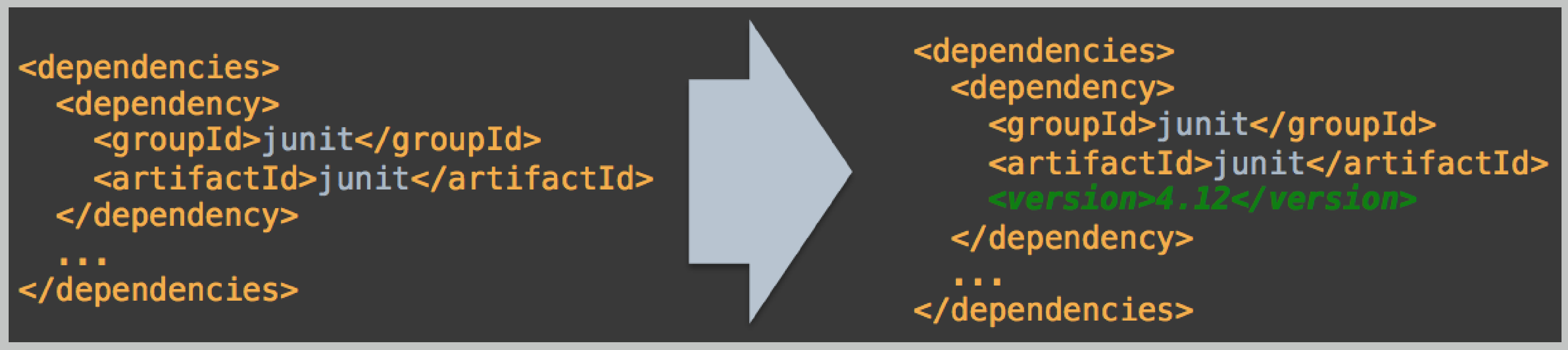
- Study *why* builds break based on past changes
- BuildDiff - Tool to extract fine-grained changes from Maven build configuration files [3]
- Investigation of changes and change patterns to understand the evolution of build configurations and the impact of changes on the build result
- Derive quality metrics for Maven build configuration files
- Evaluation through empirical analysis of (open source) repositories

Build Prediction (RQ2)

- Retrieving a build result or the need of a build configuration change in a revision, usually needs build execution (time consuming)
- Prediction models might help to estimate the build result (and save time)
- Two approaches
  - Build co-change prediction [4]
  - Build result prediction
- Evaluation of models on (open source) projects

Build Refactor and Repair (RQ3)

- Use knowledge gained in RQ1 and RQ2 to provide approaches to improve build configurations
- Refactoring
  - Focus on successful builds that can be improved
  - Reduce error-proneness of build configuration
  - Identify configuration smells
  - Provide best practice solutions
- Repair
  - Focus on failing builds
  - Derive repair strategies from successful repairs
- Evaluation by comparing repairs from our approach with repairs that developers performed



```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
  </dependency>
  ...
</dependencies>
```

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
  </dependency>
  ...
</dependencies>
```

Expected Contributions

- **Datasets** containing extracted build changes and build results of the investigated projects.
- **Rules** retrieved by empirical evidence for bad and best practices for build configurations.
- **Models** to predict build co-changing work items and to predict build results for commits and work items.
- An **approach** to automatically refactor breakage-prone builds and repair broken builds.

References

[1] N. Kerzazi, F. Khomh, and B. Adams.
Why do automated builds break? an empirical study.

[2] G. Kumfert and T. Epperly.
Software in the doe: The hidden overhead of the build.

[3] C. Macho, S. McIntosh, and M. Pinzger.
Extracting Build Changes with BuildDiff.

[4] C. Macho, S. McIntosh, and M. Pinzger.
Predicting Build Co-Changes with Source Code Change and Commit Categories.

[5] H. Seo, C. Sadowski, S. Elbaum, E. Aftandilian, and R. Bowdidge.
Programmers' build errors: a case study (at google).

Contact: http://serg.aau.at/bin/view/ChristianMacho/WebHome christian.macho@aau.at