Preventing and Repairing Build Breakage

Christian Macho
Software Engineering Research Group
University of Klagenfurt
Klagenfurt, Austria
Email: christian.macho@aau.at

***Abstract*—Build systems play a crucial role in modern software engineering. Recent studies have shown that many builds fail, mostly due to neglected maintenance. This blocks teams from continuing the development and costs time and resources to fix. The target of the thesis is to reduce build breakage by investigating changes that lead to failing builds, identifying bad and best practices for build configuration, and providing an approach to automatically repair broken builds. As a first step, we conduct empirical studies to determine changes and change patterns that lead to build breakage and reveal the reasons for build breakage. Based on these findings, we develop an approach to automatically refactor build configurations that are likely to fail and an approach to repair broken builds. We plan to evaluate our approaches first, quantitatively by measuring the performance of our approaches in open source projects and second, qualitatively by asking developers to use our approaches and give feedback on their applicability and usefulness.**

***Keywords*-build systems; maintenance; automated repair;**

I. Problem Statement

Build systems are an essential component in modern software engineering. They are used to automate tasks of the software development lifecycle, such as compiling code, assembling artifacts, generating documentation, testing artifacts, and deploying artifacts. Although they play an important role in software development, research has ignored this field for a long time [17]. Recent research, such as [18], started to investigate build systems as a component of the software development lifecycle. Moreover, relations of the build system to other components of the lifecycle as well as problems in the usage and evolution of a build system have been revealed.

Kumpfert *et al.* observed that using a build system decreases the workload of developers but its usage also adds additional overhead for maintaining the build configuration [13]. Seo *et al.* addressed this issue and found that neglected maintenance of the build configuration is one of the most frequent reasons for breaking a build [21]. Suffering from build breakage blocks development teams from continuing their work and consequently costs the companies money as Kerzazi *et al.* [12] found. Moreover, Seo *et al.* and Kerzazi *et al.* investigated the number of failing builds and showed that the ratio of build breakage is high (28.5% and 17.7%, respectively).

The reasons for build breakage and solutions to avoid and fix build breakage are still open issues. Hence, the objective of the thesis is to understand build breakage and to reduce the number of failing builds. First, we plan to investigate changes to the build configuration to reveal the reasons of build breakage. Second, we will build prediction models to explain and predict build breakage. Finally, we will investigate an approach that avoids build breakage and an approach that fixes failing builds. Our research is based on the following main hypothesis:

> *With a better understanding of how and why builds break, we can improve the quality of build configurations to reduce the number of failing builds.*

We plan to address this main hypothesis by dividing the work into three main research questions:

(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?

II. Research Approach

We split each research question into several work packages (WP). To answer our research questions, we will follow the idea of design science research [11]. To that extent, we will build prototypes and evaluate the approaches with the prototypes. The evaluation will be twofold, following a mixed methods approach [2]. First, we plan to use quantitative methods, such as random forest, support vector machine, or frequent item set mining, and second, qualitative methods, such as card sorting, or surveys. We plan to use open source Java projects to evaluate our prototypes because they provide a large corpus of data and are freely available. We restrict our studies to Apache Maven, a state of the art build system that is widely used, and leave it to future work to study other build systems. We plan to give our tools to developers to get feedback on the applicability and usefulness of our tools. In the following, we describe each research question in detail.

A. Reasons and Characteristics of Build Breakage (RQ1)

The first research question consists of three work packages and focuses on the properties and characteristics of build breakage. We plan to study why build breakage happens and which changes or change patterns are often involved in build breakage. To that extent, we will mine the version history of selected projects to reveal the relation between changes and build breakage.

Build Changes (WP1). The goal of this work package is to investigate which changes are applied to the build configuration to understand its evolution. We plan to develop an approach to extract fine-grained changes from build configurations in analogy to ChangeDistiller for Java [6]. Prior studies using ChangeDistiller showed that the fine-grained representation of changes helped to improve studies *e.g.*, on co-evolution [5] or code change prediction [8]. However, we are the first to extract build code changes from build configurations. We plan to define a taxonomy of fine-grained build changes. We believe that the fine-grained build changes will help to understand the evolution of build configurations and furthermore, contribute to the knowledge of understanding build breakage.

The evaluation of our approach will be twofold. First, we will provide an extensive tests suite to show that our approach can extract each change type properly. These tests will be created with artificially constructed data to ensure that no side effects bias our extraction. Second, we will perform a manual investigation of the extracted changes. We plan to use build configuration files of open source Java programs to show that the approach can also extract changes of real-world configurations properly. We will validate the extracted changes of our approach by comparing them with the changes retrieved by experienced developers. The expected results of this work package are an approach to extract fine-grained build changes, a preliminary evaluation of its performance, and an application of the approach to understand the evolution of build configurations.

Change Patterns (WP2). The goal of this work package is to find changes that often appear together. We plan to extend WP1 by investigating frequent patterns of build changes. With the knowledge about frequent change patterns, we can investigate their impact on build breakage. We suppose that some changes and change patterns are more likely to be involved in or cause build breakage than others. We plan to obtain frequent change patterns by using well known methods, such as the apriori algorithm, the eclat algorithm, and the fp-growth algorithm. Furthermore, we plan to build each revision of selected open source Java projects and classify the output of the build log into build results. First, we will propose a taxonomy of build results that refines the failing build category. We argue that the well known categorization of build results into successful or failing builds is too coarse-grained to study the impact of changes or change patterns on the build result. Second, we plan to propose an approach to automatically classify the output of build logs into build results of our taxonomy. We then link the changes and change patterns to build results and investigate their impact.

We plan to evaluate the taxonomy and the classification with card sorting. We will ask experienced developers to classify the output of the build log to the categories of our taxonomy and compare the results with our classification. We then plan to calculate metrics, such as f-score and AUC, to measure the performance of the classification and the interrater agreement to show the agreement of the classification with the manual investigations. Furthermore, we plan to evaluate the impact of changes and change patterns on the build result by conducting a survey with experienced developers. Besides other questions, we will ask the developers if they agree with our findings and if they experienced similar observations in their work. The expected findings are changes and change patterns and their impact on build results.

Bad and Best Practice Build Configuration (WP3). In WP3, we plan to investigate the impact of changes and frequent change patterns on the quality of the build configuration. We first plan to propose a set of quality measures for build configurations. We then investigate which properties of a build configuration affect its quality in terms of maintainability or likelihood to fail a build. Furthermore, we plan to use the set of quality measures to examine whether we can find a relation between particular changes or change patterns on the quality of the build configuration. We aim at classifying changes and change patterns into bad and best practice for build configuration.

The quality measures will be evaluated using selected open source Java projects. We plan to randomly select a sample of build configurations from the version history of the projects. We will give the sample to experienced developers and ask them to estimate the proposed quality measures for each of the samples. We then plan to compare the manually assigned values with the values retrieved by our approach. To evaluate the classification, we plan to ask developers and measure their agreement with our classification. The expected results of this work package are a set of quality measures for build configurations and a classification of changes and change patterns in terms of their impact on these quality measures.

B. Build Prediction (RQ2)

The goal of this research question is to predict the build result before the actual build has been performed to warn developers that their change set is likely to break the build. Current research, such as [10, 19], uses two different approaches for prediction. First, predicting potentially missing changes to the build configuration (build co-change) and second, predicting the build result itself. We divide this research question into two work packages.

Build Co-Change Prediction (WP4). We already performed a study on predicting build co-changes based on information on the source code changes that have been performed and commit categories [16]. This work is an extension to the work of McIntosh *et al.* [19] and Xia *et al.* [24]. We extracted change information and commit message information of 10 open source Java projects and built a random forest classifier to predict build co-changes. We evaluated our models using a repeated random sampling approach and well known metrics, such as f-score and AUC. Our results show that we could improve the state of the art classifier for intra- and cross-project prediction by 11.54% and 9.46%, respectively. Furthermore, we observed that advanced resampling methods strongly support the improvement.

Build Result Prediction (WP5). The goal of this work package is to gain knowledge of the factors that impact the

build result and to build a model that predicts the build result. This has already been investigated from other points of view. For example, Kwan *et al.* [14] used socio-technical congruence, and Wolf *et al.* [23] used coordination structures of development teams to predict the build result. In contrast to these studies, we plan to base the prediction models mainly on changes.

We believe that notifying developers if their current change set might break the build will improve the quality of the build configuration and hence, decrease the number of breaking builds. We plan to use general metrics of projects and commits and detailed change information on source code and build configuration to build the prediction model. Furthermore, we will use classic machine learning approaches, such as random forest or support vector machines, to build the prediction model.

We plan to evaluate the resulting models with selected open source Java projects and measure common performance measures, such as f-score and AUC. Furthermore, we will use cross fold validation to show the stability of the model. The impact of the attributes of our model on the prediction result will be evaluated by providing code snippets from open source Java projects as a rationale. As a result, we expect a model that predicts the build result based on our investigated metrics. Furthermore, the model is expected to explain relations and reasons for build breakage.

C. Build Repair (RQ3)

Our third research question aims at providing approaches to improve the quality of the build configuration. We split this research question into two work packages. WP6 focuses on situations where the build configuration is likely to break a build. We plan to propose an approach that restructures the build configuration to improve its quality and hence, decrease the likelihood of build breakage. WP7 deals with situations where the build is already broken. We aim at providing an approach that detects the reason for the breakage and recommends a solution to repair the build.

Build Configuration Refactoring (WP6). Refactoring is a well known action to transform code into semantically identical code to improve the structure and reduce error proneness [7]. In this work package, we plan to use the knowledge of the previous research questions to derive refactorings that improve the quality of the build configuration. The refactorings will be based on the findings of WP1, WP2, and WP3 regarding the investigation of error-prone changes and change patterns. Furthermore, we plan to use the findings of WP4 and WP5 to identify build configurations that are likely to fail a build.

The planned evaluation of our approach is twofold. First, we plan to evaluate our approach with the metrics which we defined in WP3. We will list the target metrics which will be improved by the respective refactoring and measure them before and after the refactoring process to evaluate the improvement. Second, we plan to ask developers if they agree with the refactorings. The expected result of this work package is an approach that identifies build configurations that are likely to fail the build and recommends a set of refactorings to improve the quality of the build configuration. We expect that a build configuration with higher quality measure will reduce the number of failing builds.

Build Configuration Repair (WP7). WP7 focuses on already broken builds. Similar to the intention of WP6, we plan to learn refactorings from the previous work packages WP1, WP2, and WP3. Contrary to WP6, we do not aim at restructuring the build configuration to improve the quality of the build configuration. Instead, we focus on repairing a failed build. Automatic repair is also an open issue in many other research areas *e.g.*, automatically fixing bugs [22]. We aim at learning actions by mining the version history of open source Java projects to modify the build configuration in a way that the build is successful again.

We will evaluate the approach first by checking how many failing builds can be repaired and second, by comparing the repair actions of our approach with repair actions that have been performed by developers for the same failing build. The expected result of this research question is an approach that automatically repairs broken builds.

III. Expected Contributions and Implications

In this section, we list the expected contributions and state the expected implications on software development as well as on research. The expected contributions are:

- **Datasets** containing extracted build changes of the investigated projects (WP1) and build results of the investigated projects (WP3).
- **Rules** retrieved by empirical evidence for bad and best practices for build configurations (WP2, WP3).
- **Models** to predict build co-changing work items (WP4) and to predict build results for commits and work items (WP5).
- An **approach** to automatically refactor breakage-prone builds (WP6) and repair broken builds (WP7).

We already performed an empirical study to improve the prediction of build co-changes (WP4) which has been published at SANER'16 [16]. We expect several implications of our planned findings on software development and research.

For **developers**, we will provide approaches and tools that will be integrated into the IDE of developers to support software development and maintenance. We expect that our contributions enable software developers to avoid build breakage in their projects and to automatically repair failing builds. Furthermore, we plan to derive rules from our investigations for bad and best practices of build configurations which can be used by developers to increase the quality of their build configuration. This allows software development teams to focus on their core business because they will not be blocked by broken builds.

Concerning **research**, we will provide insights into the evolution and maintenance of build configurations. Furthermore, we will make our datasets and prototype tools publicly available so that researchers can use them and extend our studies.

IV. Related Work

Build breakage has already been studied in previous research. Seo *et al.* [21] and Kerzazi *et al.* [12] studied builds at large companies and found that 30% and 18%, respectively, of the builds fail. Other studies aim at predicting the build result. Wolf *et al.* [23] and Kwan *et al.* [14] built models to predict the outcome of a build. They used coordination structures of development teams of an IBM project and socio-technical congruence, respectively, to predict the build result.

Build Maintenance and the co-evolution of build configuration with other artifacts has also been a topic of recent research. Adams *et al.* developed MAKAO [1], a re(verse)-engineering framework for build configurations. They found that maintenance is the main driver for evolution and observed that the complexity of a build system increases over time. Java build systems have also been studied. For example McIntosh *et al.* investigated the co-evolution of production and test code with build configuration code [18]. They also found a relationship between build technology and maintenance effort in a follow-up work [20].

Previous research already developed tools to **extract changes** from two versions of a file. Hashimoto and Mori [9] developed Diff/TS. Their approach uses the raw Abstract Syntax Tree (AST) to extract the changes. Another approach that uses the AST is ChangeDistiller of Fluri *et al.* [6]. This approach extracts differences from two consecutive versions of a Java file and maps the differences to 48 different change types [4]. ChangeDistiller was improved by the GumTree approach of Falleri *et al.* [3] that scans the AST in two directions to generate the differences.

Automatic repair has been studied for programs by Weimer *et al.* [22]. They proposed GenProg, a tool to repair programs. Le Goues *et al.* [15] studied the number of bugs that can be fixed automatically with GenProg and the costs of the fixes.

References

[1] Bram Adams, Herman Tromp, Kris De Schutter, and Wolfgang De Meuter. Design recovery and maintenance of build systems. In *Proc. of Intl. Conf. on Software Maintenance*, pages 114–123. IEEE, 2007.

[2] John W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. SAGE publications, 2013.

[3] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. In *Proc. of Intl. Conf. on Automated Software Engineering*, pages 313–324. ACM, 2014.

[4] Beat Fluri and Harald C. Gall. Classifying change types for qualifying change couplings. In *Proc. of Intl. Conf. on Program Comprehension*, pages 35–45. IEEE, 2006.

[5] Beat Fluri, Michael Würsch, Emanuel Giger, and Harald C. Gall. Analyzing the co-evolution of comments and source code. *Software Quality Journal*, 17(4):367–394, 2009.

[6] Beat Fluri, Michael Würsch, Martin Pinzger, and Harald C. Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *Transactions on Software Engineering*, 33(11):725–743, 2007.

[7] Martin Fowler, Kent Beck, J Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the design of existing programs*. Addison-Wesley Reading, 1999.

[8] E. Giger, M. Pinzger, and H. C. Gall. Can we predict types of code changes? an empirical analysis. In *Proc. of Work. Conf. of Mining Software Repositories*, pages 217–226. IEEE, 2012.

[9] Masatomo Hashimoto and Akira Mori. Diff/TS: a tool for fine-grained structural change analysis. In *Proc. of Work. Conf. on Reverse Engineering*, pages 279–288. IEEE, 2008.

[10] Ahmed E Hassan and Ken Zhang. Using decision trees to predict the certification result of a build. In *Proc. of Intl. Conf. on Automated Software Engineering*, pages 189–198. IEEE, 2006.

[11] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Q.*, 28(1):75–105, March 2004.

[12] Noureddine Kerzazi, Foutse Khomh, and Bram Adams. Why do automated builds break? an empirical study. In *Proc. of Intl. Conf. on Software Maintenance and Evolution*, pages 41–50. IEEE, 2014.

[13] Gary Kumfert and Tom Epperly. Software in the doe: The hidden overhead of the build. *Lawrence Livermore National Laboratory, Technical Report*, 2002.

[14] Irwin Kwan, Adrian Schroter, and Daniela Damian. Does socio-technical congruence have an effect on software build success? a study of coordination in a software project. *IEEE Transactions on Software Engineering*, 37(3):307–324, 2011.

[15] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In *Proc. of Intl. Conf. on Software Engineering*, pages 3–13. IEEE, 2012.

[16] Christian Macho, Shane McIntosh, and Martin Pinzger. Predicting Build Co-Changes with Source Code Change and Commit Categories. In *Proc. of Intl. Conf. on Software Analysis, Evolution, and Reengineering*, pages 541–551. IEEE, 2016.

[17] Shane McIntosh, Bram Adams, and Ahmed E. Hassan. The evolution of ANT build systems. In *Proc. of Int. Work. Conf. on Mining Software Repositories*, pages 42–51. IEEE, 2010.

[18] Shane McIntosh, Bram Adams, and Ahmed E. Hassan. The evolution of java build systems. *Empirical Software Engineering*, 17(4-5):578–608, 2012.

[19] Shane McIntosh, Bram Adams, Meiyappan Nagappan, and Ahmed E. Hassan. Mining co-change information to understand when build changes are necessary. In *Proc. of Intl. Conf. on Software Maintenance and Evolution*, pages 241–250. IEEE, 2014.

[20] Shane McIntosh, Meiyappan Nagappan, Bram Adams, Audris Mockus, and Ahmed E. Hassan. A large-scale empirical study of the relationship between build technology and build maintenance. *Empirical Software Engineering*, 20(6):1587–1633, 2015.

[21] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge. Programmers' build errors: a case study (at google). In *Proc. of Intl. Conf. on Software Engineering*, pages 724–734. ACM, 2014.

[22] Westley Weimer, Stephanie Forrest, Claire Le Goues, and ThanhVu Nguyen. Automatic program repair with evolutionary computation. *Communications of the ACM*, 53(5):109–116, 2010.

[23] Timo Wolf, Adrian Schroter, Daniela Damian, and Thanh Nguyen. Predicting build failures using social network analysis on developer communication. In *Proc. of Intl. Conf. on Software Engineering*, pages 1–11. IEEE, 2009.

[24] Xin Xia, David Lo, Shane McIntosh, Emad Shihab, and Ahmed E. Hassan. Cross-project build co-change prediction. In *Proc. of Intl. Conf. on Software Analysis, Evolution, and Reengineering*, pages 311–320. IEEE, 2015.