Preventing and Repairing Build Breakage - Extended Abstract

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

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. Recent research, such as [9], started to investigate build systems as a component of the software development lifecycle. 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 [7]. 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 [10]. Kerzazi *et al.* found that suffering from build breakage blocks development teams from continuing their work and consequently costs the companies money [6]. Moreover, Seo *et al.* and Kerzazi *et al.* investigated the number of failing builds and obvserved a high ratio of build breakage (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 divide our research into three main research questions (RQ) and seven work packages (WP) described in the following. We will follow the idea of design science research [5]. To that extent, we will build prototypes and evaluate the approaches with the prototypes. The evaluation will be twofold, following a mixed methods approach [1]. First, we plan to use quantitative methods, such as random forest, 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. 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.

RQ 1 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 because prior studies using ChangeDistiller showed that the fine-grained representation of changes helped to improve studies [2]. However, we are the first to extract build code changes from build configurations. We believe that the fine-grained build changes will help to understand the evolution of build configurations and contribute to the knowledge of understanding build breakage. The evaluation of our approach will be twofold by providing an extensive tests suite to show that our approach can extract each change type properly, and by performing a manual investigation of the extracted changes of real world 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 retrieved by well known algorithms, such as the apriori algorithm. With this knowledge, we can investigate their impact on build breakage. Furthermore, we plan to build each revision of selected open source Java projects and classify the output of the build log into build results. We will 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 and common metrics, such as f-score and AUC. Furthermore, we plan to evaluate the impact of changes and change patterns on the build result by conducting a survey with experienced developers.

Bad and Best Practice Build Configuration (WP3). In WP3, we plan to investigate the impact of 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 change patterns and build configuration quality. We aim at classifying change patterns into bad and best practice for build configuration. The quality measures will be evaluated by experienced developers. We plan to compare the manually assigned values with the values retrieved by our approach and measure the agreement. The expected results of this work package are a set of quality measures for build configurations and a classification of change patterns in terms of their impact on these quality measures.

The goal of **RQ2** is to predict the build result before the actual build has been performed. Current research, such as [4], uses two different approaches for prediction. First, predicting potentially missing changes to the build configuration and second, predicting the build result.

Build Co-Change Prediction (WP4). We already performed a study on predicting build co-changes based on information of the source code changes and commit categories [8]. 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.

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 *e.g.*, using social network metrics. However, 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 will use change and general metrics, and 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.

RQ 3 aims at providing approaches to improve the quality of the build configuration. We split this research question into two work packages. First, we plan to improve the quality of build configurations by refactoring and second, we aim at repairing broken build configurations.

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 [3]. 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 to measure 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 smelly build configurations and recommends 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). Contrary to WP6, we do not aim at restructuring the build configuration to improve the quality of the build configuration but focus on repairing a failed build. Automatic repair is also an open issue in many other research areas *e.g.*, automatically fixing bugs [11]. 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.

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.

References

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

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

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

[4] Ahmed E Hassan and Ken Zhang. Using decision trees to predict the certification result of a build. In *ASE*, pages 189–198. IEEE, 2006.

[5] 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.

[6] 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.

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

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

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

[10] 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.

[11] 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.