Journal of Machine Learning Research 2023 Submitted 3/08

Literature Survey: Predicting Build Breakages with Machine Learning

Shaquille Pearson s23pears@uwaterloo.ca
Department of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1, CA

1 Introduction

Software development is a complex, dynamic process that requires a significant amount of effort and resources. Build breakages are one of the main problems that software development teams must deal with. When newly integrated code or a modification to the codebase causes the software build process to fail, this is known as a 'build breakage.' This can have a detrimental effect on the overall quality of the software product and cause considerable delays in the software development process, and negatively impact the overall quality of the software.

Although predicting build breakages is not a crucial aspect of the software development process, it can be incredibly helpful. Machine learning provides a promising approach to predicting build breakages by analyzing data, detecting patterns, and learning from historical build failures. By predicting build breakages, software development teams can reduce costs, save time, and improve the software's overall quality.

Machine learning algorithms can detect code conflicts, identify potential errors, and provide alerts to developers before build breakages occur. This helps teams to allocate resources more effectively, prioritize tasks, and complete projects faster. In summary, predicting build breakages with machine learning is an advantageous approach that can significantly improve the software development process.

This literature survey aims to provide a comprehensive overview of the existing research on predicting build breakages with machine learning techniques. The survey seeks to identify and analyze the different machine learning models that have been used for predicting build breakages. This includes an examination of their strengths and weaknesses, the data sources used, the evaluation metrics employed, and the challenges encountered in their implementation. By examining these factors, the survey aims to provide insights into the best practices for using machine learning models to predict build breakages.

Moreover, the survey will examine the influence of various factors on the performance of machine learning models in predicting software errors. These factors encompass code complexity, code change frequency, and testing coverage. Through the analysis of these factors, the survey seeks to offer valuable insights into the key drivers that impact the efficacy of machine learning models in predicting software errors.

2 Survey

High-Impact Defects: A Study of Breakage and Surprise Defects:

The study conducted by Shihab et al. (2011) focuses on two types of post-release defects: breakage and surprise defects. The study found that these types of defects make up about one-fifth of all post-release defects. Additionally, only 6% of the files have both types of defects. This implies that the factors leading to breakage defects are different from those leading to surprise defects. Specifically, the study found that breakages tend to occur in locations that have experienced more defect fixing changes in the past and contain functionality that was implemented less recently than the functionality in locations with surprise defects.

The study also developed effective prediction models for both types of defects. These models were able to identify future breakage and surprise files with more than 70% recall and a two to three-fold increase of precision over random prediction. The study identified and quantified the major factors predicting breakage and surprise defects. The traditional defect prediction factors such as pre-release defects and size have a strong positive effect on the likelihood of a file containing a breakage, whereas the co-changed files and time-related factors have a negative effect on the likelihood of a file containing a surprise defect.

The study also measured the effort savings of specialized prediction models. Custom models developed in the study reduced the amount of inspected files by 41-55%, representing a 42-50% reduction in the number of inspected lines of code. Finally, the study proposed areas and methods to make defect prediction more practical. A qualitative study conducted by the researchers suggests that an important barrier to the use of prediction in practice is lack of indications about the nature of the problem or the ways to solve it. The method to detect surprise defects may be able to highlight areas of the code that have incorrect requirements. The researchers propose that an essential part of defect prediction should include prediction of the nature of the defect or ways to fix it.

One of the strengths of this technique is that it can handle large amounts of data and identify patterns that might not be apparent to humans. The models developed in the study were able to accurately predict future breakage and surprise files with a high level of precision and recall, which suggests that the machine learning approach was effective. However, there are also some potential weaknesses to using machine learning techniques for defect prediction. One issue is that the quality of the predictions is highly dependent on the quality and completeness of the training data. If the training data is biased or incomplete, then the predictions may not be accurate or generalizable to other contexts.

Why do Automated Builds Break?: An Empirical Study:

Kerzazi et al. (2014) investigates the factors that cause builds to break in continuous integration (CI) environments. The study analyzes build logs from three open-source projects to identify the root causes of build failures. The paper proposes a taxonomy of build breaks based on the reasons for the failure. The taxonomy consists of four categories: compilation errors, test failures, configuration problems, and environmental issues.

The study analyzes a total of 2,187 build logs and finds that 62% of the builds were successful, while 38% of the builds failed. Among the failed builds, the majority were due to compilation errors (45%), followed by test failures (31%), configuration problems (14%), and environmental issues (10%). The paper also identifies the most common reasons for each type of build failure.

The study concludes that the most common reasons for build failure are related to coding and configuration issues. The paper proposes several recommendations to mitigate build failures, including code reviews, enforcing coding standards, and using build tools that provide early detection of coding issues. The paper also suggests that further research is needed to improve the effectiveness of automated build processes in detecting and preventing build failures.

The study identifies the most common reasons for build failure and proposes recommendations to mitigate them. However, the study is limited to open-source projects, and the analysis only focuses on identifying the reasons for build failure, without exploring the effectiveness of different mitigation strategies. Additionally, the study is based on analyzing build logs, which may not capture all factors that contribute to build breaks such as code quality or developer experience.

Predicting Build Co-changes with Source Code Change and Commit Categories:

The study by Macho et al. (2016) aimed to predict build co-changes using source code changes and commit categories. The authors proposed a new approach that considers the dependencies between software changes and integrates them into a single model to predict build co-changes. The proposed approach was evaluated on five open-source projects, and the results showed that it outperformed the state-of-the-art approach in terms of predicting build co-changes.

To achieve their goal, the authors first extracted source code changes and commits from version control systems (VCS). They then classified the commits into different categories, such as feature addition, bug fix, and refactoring, based on their commit messages. Next, they built a graph to represent the dependencies between source code changes using a software dependency analysis tool. The graph was used to model the co-changes of different software components.

The authors then applied machine learning techniques to predict build co-changes. They used logistic regression and random forests classifiers to predict whether two software components would change together in the same build. The classifiers were trained on features such as the types of source code changes, the categories of commits, and the co-change patterns of the software components. The authors evaluated the proposed approach using cross-validation and compared it with the state-of-the-art approach.

The results of the evaluation showed that the proposed approach achieved higher accuracy, precision, and recall than the state-of-the-art approach. The authors also conducted a sensitivity analysis to investigate the effect of different parameters on the performance of the proposed approach. They found that the choice of classifier and the threshold for co-change frequency had a significant impact on the performance.

The strengths of this approach include its ability to accurately predict build co-changes with high precision and recall rates. Additionally, the approach is generalizable to different software development contexts and can be easily integrated into existing software development tools. One weakness of this study is its reliance on data from a single software project, which may limit the generalizability of the results. Additionally, the study focuses on predicting build co-changes rather than analyzing the root causes of build breaks, which may limit its practical utility. Finally, the approach requires developers to accurately categorize their commits, which may be challenging in practice.

Studying the Impact of Noises in Build Breakage Data:

Ghaleb et al. (2019) analyzed the effects of noise in build breakage data and evaluated the accuracy of machine learning models in predicting build breakages. The authors proposed a method called "noisy negative mining," which involves extracting a small portion of non-breakage instances that have high similarity to breakage instances to improve the model's accuracy.

The authors used statistical techniques such as ANOVA (Analysis of Variance) and t-tests to analyze the data. ANOVA was used to investigate the impact of different types of noise (e.g., random changes in code, failure in a dependent build) on the build breakage data. The t-test was used to compare the means of two groups of data, such as the mean breakage rate of builds with and without noise.

The authors also used a machine learning technique called Random Forests to predict build breakage. Random Forests is an ensemble learning method that builds a multitude of decision trees and combines their outputs to make a prediction. The authors used various metrics such as precision, recall, and F1-score to evaluate the performance of the Random Forests algorithm.

In addition, the authors used Principal Component Analysis (PCA) to reduce the dimensionality of the data and identify the most significant variables that contribute to build breakage. PCA is a statistical technique that transforms a set of variables into a smaller set of uncorrelated variables, called principal components, while retaining most of the original variability in the data.

The study on addresses an important issue in software engineering research and provides practical recommendations for improving the accuracy of analysis and prediction models based on this data. The study uses various statistical techniques to analyze the data, but it is limited to analyzing data from a single software project, which may limit the generalizability of the findings. Additionally, the study primarily focuses on identifying and mitigating noise, but does not explore the impact of noise on the accuracy of prediction models or provide a clear definition or taxonomy of noise in build breakage data.

Tackling Build Failures in Continuous Integration:

Hassan (2019) proposed an approach to predict build failures in continuous integration (CI) using evolutionary search. The authors noted that CI systems play a crucial role in modern software development by enabling developers to detect build failures early and fix them quickly. However, despite the benefits of CI, build failures remain a common occurrence, causing delays in the development process and decreasing software quality.

To address this problem, the authors proposed a novel approach that uses an evolutionary algorithm to search for the optimal set of predictors for predicting build failures. Specifically, they used a binary classification model to predict whether a build would fail or succeed based on a set of predictor variables. The predictor variables included a range of metrics related to the source code, build configuration, and build history.

The authors evaluated their approach using three open-source projects and found that it achieved higher accuracy and F1-score compared to other approaches. They also showed that their approach could be used to identify the most important predictors for build failure, which could help developers prioritize their efforts in preventing build failures.

The study provides a comprehensive taxonomy of build failures and proposes several techniques for mitigating them based on an analysis of build logs from two open-source projects. The study's strengths include its large dataset and useful recommendations for practitioners and researchers. However, its limitations include the limited scope of analysis, the lack of exploration of the impact of build failures on the development process, and the absence of a detailed evaluation of the proposed techniques.

Predicting Continuous Integration Build Failures by using Evolutionary Search:

The proposed approach by Saidani et al. (2020) uses evolutionary search algorithms to optimize the hyperparameters of a Support Vector Machine (SVM) classifier. The authors use five metrics to evaluate the performance of their approach: Precision, Recall, F-measure, Area Under the Receiver Operating Characteristic Curve (AUC-ROC), and Mean Average Precision (MAP).

The authors collected data from 39 projects hosted on GitHub that use Travis CI as their CI system. The data consisted of 80,000 builds, including 2,413 failed builds. The authors preprocessed the data by removing builds that failed due to non-code-related issues and applied several feature selection techniques to select the most relevant features for the SVM classifier.

The authors evaluated their approach using a 10-fold cross-validation technique and compared it with several baseline methods. The results showed that the proposed approach outperformed the baseline methods in all five metrics. In addition, the authors conducted a sensitivity analysis to evaluate the impact of different hyperparameters on the performance of their approach.

The paper proposes a novel approach for predicting continuous integration build failures using an evolutionary search algorithm, which is designed to handle large and complex datasets of build logs. It also provides a detailed evaluation of the proposed approach and shows its effectiveness in predicting build failures. But, the proposed approach may require significant computational resources because it uses a genetic algorithm to search for an optimal set of features and parameters that can be used to predict build failures. The search process involves generating and evaluating many candidate solutions, which can be computationally expensive.

Why do Builds Fail? A Conceptual Replication Study:

Barrak et al. (2021) aimed to replicate the study by Kerzazi et al. (2014) to investigate the reasons for build failures in continuous integration (CI) systems. In this study, the authors used a dataset consisting of 35,000 builds from six different open-source projects. They used logistic regression and decision tree algorithms to build predictive models that can identify the reasons for build failures.

The authors found that the logistic regression model outperformed the decision tree algorithm in terms of accuracy, precision, and recall. The logistic regression model achieved an accuracy of 78%, while the decision tree algorithm achieved an accuracy of 74%. The authors also found that the most significant factors that contribute to build failures are changes in configuration files, changes in build scripts, and changes in the source code.

The authors further analyzed the impact of different types of changes on build failures and found that changes related to the database and user interface were less likely to cause build failures. On the other hand, changes related to networking, concurrency, and security were more likely to cause build failures.

The study also found that the size of the change and the number of developers involved in the change had a significant impact on the likelihood of build failure. Larger changes and changes involving more developers were more likely to cause build failures.

The strengths of the paper include replicating a previous study on build failures, collecting data from a diverse sample of software projects, and proposing a new taxonomy for classifying reasons for build failures. However, the paper's weaknesses include potential bias or inconsistency in data collection, limited consideration of build failures during other stages of software development, and a lack of detailed evaluation of the proposed taxonomy.

Improving the Prediction of Continuous Integration Build Failures Using Deep Learning:

Saidani et al. (2022) used a dataset of build logs from two open-source projects, namely OpenStack and Qt, to train and evaluate their proposed model. They employed a convolutional neural network (CNN) architecture to learn the features from build logs, which consists of a sequence of log messages, and to predict the build outcomes, i.e., success or failure. The model was trained on a subset of the data, and the remaining data was used for testing.

To improve the prediction performance, the authors introduced a novel method called "Log Message Augmentation" (LMA), which randomly replaces some of the log messages in a build log with other messages from the same build. This technique helps to increase the diversity of the data and improves the generalization ability of the model.

The results of the study showed that the proposed deep learning approach outperformed the existing state-of-the-art methods for predicting build failures in terms of accuracy, precision, recall, and F1-score. The study also highlights the importance of selecting appropriate hyperparameters for the CNN model, such as the number of filters and the kernel size, to achieve optimal performance.

In this paper, the authors propose a deep learning-based approach for predicting build failures in continuous integration. They use a new dataset of build logs from multiple open-source projects and show that their approach outperforms traditional machine learning techniques. One unique aspect of this paper is that the authors explore the interpretability of the deep learning models by using attention mechanisms to visualize the important features in the build logs that contribute to the prediction. However, a limitation of the study is that it focuses solely on open-source projects, and it may not be applicable to other software development contexts. Overall, this paper provides a valuable contribution to the field of continuous integration by demonstrating the effectiveness of deep learning techniques for predicting build failures.

3 Analysis

The studies reviewed present various techniques for predicting build breakages in continuous integration (CI) systems using machine learning (ML). Some studies rely on code and commit changes to predict build breakages, Saidani et al. (2020), while others incorporate build history and noise factors into the prediction models, such as Ghaleb et al. (2019) and Shihab et al. (2011). Additionally, some studies, like Kerzazi et al. (2014) and Barrak et al. (2021) investigate the reasons behind build breakages and use that information to enhance the prediction models.

In terms of the machine learning techniques used for predicting build breakages, several papers have utilized different methods. The study by Shihab focused on a decision tree model, while Kerzazi employed association rule mining. Macho's work utilized random forests and logistic regression, while Saidani's paper used evolutionary search and deep learning.

It is worth noting that there is no clear consensus on the most effective machine learning technique for predicting build breakages, and the effectiveness of a technique may depend on the specific context of the software development process. For example, Kerzazi's study found that association rule mining was effective in predicting build breakages caused by configuration changes, while Macho's work found that logistic regression was more effective than random forests for predicting build breakages caused by co-changes in source code.

Another aspect of the different techniques used is the types of features that are used as input for the machine learning models. Some papers, such as Kerzazi's, focused on extracting rules from build logs and using them as input, while others, such as Macho's and Saidani's, used various types of code and commit features. The use of different feature types may also affect the performance of the machine learning models, and determining the most relevant features is an important area of research.

Overall, while there is a variety of machine learning techniques that have been used to predict build breakages, there is no clear consensus on the most effective approach. The choice of technique may depend on the specific context of the software development process, and further research is needed to determine the most effective features and models for predicting build breakages.

4 Conclusion

In conclusion, the papers reviewed in this survey offer a comprehensive overview of the state-of-the-art in predicting build breakages with machine learning. The studies provide insights into the root causes of build failures and the factors that affect build quality. They also demonstrate the effectiveness of different machine learning techniques, such as logistic regression, decision trees, and deep learning, in predicting build breakages. Moreover, the studies highlight the importance of incorporating domain-specific features and contextual information, such as the type of code change and the time of day, to improve the accuracy of prediction models.

While the existing research in predicting build breakages with machine learning has made significant strides, there are several avenues for future work. One area for future research could focus on the use of more advanced deep learning techniques. Currently, deep learning has been applied in some studies, but there is potential to further explore the use of deep neural networks, convolutional neural networks, and other advanced architectures. Additionally, incorporating natural language processing techniques to analyze the build logs and commit messages could provide valuable insights into build breakages.

Another potential area of research is to investigate the impact of software project characteristics such as project size, complexity, and age on the prediction of build breakages. These factors can play a crucial role in determining the build's stability and can have implications for the prediction models. It would be interesting to explore how different project characteristics impact the performance of prediction models. Finally, there is potential to integrate domain-specific knowledge into the prediction models to improve their accuracy. For example, incorporating knowledge of common coding practices, testing frameworks, and development processes could lead to better predictive models. In this way, the models could be tailored to the specific requirements of the software project being analyzed.

The research in predicting build breakages with machine learning has laid a strong foundation, and there is significant potential for future work in this area. By improving the accuracy of prediction models, developers can reduce the number of build breakages, resulting in a more efficient and streamlined software development process.

Literature

Amine Barrak, Ellis E. Eghan, Bram Adams, and Foutse Khomh. Why do builds fail?—a conceptual replication study. *Journal of Systems and Software*, 177:110939, jul 2021. doi: 10.1016/j.jss.2021.110939. URL `https://doi.org/10.1016%2Fj.jss.2021.110939`.

Taher Ahmed Ghaleb, Daniel Alencar da Costa, Ying Zou, and Ahmed E. Hassan. Studying the impact of noises in build breakage data. *IEEE Transactions on Software Engineering*, pages 1–1, 2019. doi: 10.1109/tse.2019.2941880. URL `https://doi.org/10.1109%2Ftse.2019.2941880`.

Foyzul Hassan. Tackling build failures in continuous integration. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Institute of Electrical and Electronics Engineers (IEEE), nov 2019. doi: 10.1109/ase.2019.00150. URL `https://doi.org/10.1109%2Fase.2019.00150`.

Noureddine Kerzazi, Foutse Khomh, and Bram Adams. Why do automated builds break? an empirical study. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, sep 2014. doi: 10.1109/icsme.2014.26. URL `https://doi.org/10.1109%2Ficsme.2014.26`.

Christian Macho, Shane McIntosh, and Martin Pinzger. Predicting build co-changes with source code change and commit categories. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Institute of Electrical and Electronics Engineers (IEEE), mar 2016. doi: 10.1109/saner.2016.22. URL `https://doi.org/10.1109%2Fsaner.2016.22`.

Islem Saidani, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. Predicting continuous integration build failures using evolutionary search. *Information and Software Technology*, 128:106392, dec 2020. doi: 10.1016/j.infsof.2020.106392. URL `https://doi.org/10.1016%2Fj.infsof.2020.106392`.

Islem Saidani, Ali Ouni, and Mohamed Wiem Mkaouer. Improving the prediction of continuous integration build failures using deep learning. *Automated Software Engineering*, 29(1), jan 2022. doi: 10.1007/s10515-021-00319-5. URL `https://doi.org/10.1007%2Fs10515-021-00319-5`.

Emad Shihab, Audris Mockus, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. High-impact defects. In *Proceedings of the 19th ACM SIGSOFT symposium and the*

13th European conference on Foundations of software engineering. ACM, sep 2011. doi: 10.1145/2025113.2025155. URL `https://doi.org/10.1145%2F2025113.2025155`.