TRABAJO FIN DE MÁSTER

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA.

MÁSTER UNIVERSITARIO EN INGENIERÍA DEL SOFTWARE E INTELIGENCIA ARTIFICIAL

E.T.S. INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE MÁLAGA

PREDICCIÓN AUTOMÁTICA DEL RESULTADO DE INTEGRACIÓN CONTINUA EN EL DESARROLLO DE SOFTWARE MODERNO.

AUTOMATIC PREDICTION OF CONTINUOUS INTEGRATION OUTCOME IN MODERN SOFTWARE DEVELOPMENT.

Realizado por
Joaquín Alejandro España Sánchez

Tutorizado por
Gabriel Jesús Luque Polo
Francisco Javier Servant Cortés

UNIVERSIDAD DE MÁLAGA

Escuela Técnica Superior de Ingeniería Informática

—

Universidad de Málaga,
Málaga, Septiembre de 2024

Índice general

Resumen – En el contexto del desarrollo de software moderno, la Integración Continua (*CI*) es una práctica ampliamente adoptada que busca automatizar el proceso de integración de cambios de código en un proyecto. A pesar de ofrecer numerosas ventajas, implementarla conlleva una serie de costos significativos que deben ser abordados para la garantizar la eficiencia a largo plazo. La fase de Integración Continua puede resultar costosa tanto en términos de recursos computacionales como económicos, llevando a grandes empresas como Google y Mozilla a invertir millones de dólares en sus sistemas de *CI* [1]. Han surgido numerosos enfoques para reducir el costo asociado a la carga computacional evitando ejecutar construcciones que se espera que sean exitosas [2]. Sin embargo, estos enfoques no son exactos, llegando a hacer predicciones erróneas que omiten ejecutar construcciones que realmente fallan. Además de los costos asociados con la carga computacional y económica de la *CI*, otro problema al que se enfrentan los equipos de desarrollo de software es el tiempo que deben esperar para obtener *feedback* del resultado del proceso de *CI* [3]. Este tiempo de espera en ocasiones puede ser significativo y puede afectar negativamente a la productividad y eficiencia del equipo, así como a la capacidad de respuesta ante problemas y ajustes rápidos en el desarrollo. Así, en este trabajo nuestro objetivo es reducir el costo computacional en CI, al mismo tiempo que maximizamos la observación de construcciones fallidas. Para ello, se ha realizado un estudio empírico sobre técnicas existentes [2,4,5,6,7,8], y se ha propuesto una implementación, *JAES24*, que busca contribuir a las mismas. Posteriormente, se han realizado una serie de experimentos para verificar y validar la efectividad de *JAES24* en comparación con otras técnicas existentes. Finalmente, se desarrollarán unas conclusiones sobre los resultados obtenidos y se propondrán posibles líneas de trabajo futuro.

Palabras clave: Integración Continua, Predicción de Builds, Costo de Mantenimiento, Aprendizaje Automático, Fallas de Builds, Predicción de Fallas, Ahorro de Costos, Predicción de Resultados de Builds, Mantenimiento de Software, Características de Builds

Abstract – In the context of modern software development, Continuous Integration (CI) is a widely adopted practice that aims to automate the process of integrating code changes in a project. Despite offering numerous advantages, implementing CI involves significant costs that need to be addressed to ensure long-term efficiency. The Continuous Integration phase can be costly in terms of computational and economic resources, leading large companies like Google and Mozilla to invest millions of dollars in their CI systems [1]. Several approaches have emerged to reduce the cost associated with computational load by avoiding running builds that are expected to be successful [2]. However, these approaches are not accurate, often making erroneous predictions that skip running builds that actually fail. In addition to the costs associated with computational and economic load of CI, another problem faced by software development teams is the time they have to wait to get feedback on the CI process outcome [3]. This waiting time can sometimes be significant and can negatively impact team productivity and efficiency, as well as the ability to respond to issues and make quick adjustments in development. Therefore, the objective of this work is to reduce the computational cost in CI while maximizing the observation of failed builds. To achieve this, an empirical study on existing techniques has been conducted [2,4,5,6,7,8], and an implementation, *JAES24*, has been proposed to contribute to these techniques. Subsequently, a series of experiments have been conducted to verify and validate the effectiveness of *JAES24* compared to other existing techniques. Finally, conclusions will be drawn on the obtained results and possible future lines of work will be proposed.

Keywords: Continuous Integration, Build Prediction, Maintenance Cost, Machine Learning, Build Failures, Failure Prediction, Cost Saving, Build Outcome Prediction, Software Maintenance, Build Features

1. Introducción

La Integración Continua (*Continuous Integration, CI*) es una práctica de desarrollo de *software* que busca automatizar el proceso de fusión de cambios de código en un proyecto, donde cada integración es verificada mediante la ejecución automática de pruebas. Este proceso busca la detección temprana de errores y mejorar la calidad del software, permitiendo una integración más frecuente y rápida del trabajo de todos los desarrolladores. Las buenas prácticas de *CI* [9] permiten una rápida detección de errores y su resolución, un *feedback* rápido, la reducción de errores que provienen de tareas manuales, unas tasas de *commits* y *pull requests* más altas, una calidad del *software* mayor, reconocer errores en producción temprano antes del despliegue, etc. Numerosos son sus ámbitos de aplicación: *software* empresarial, desarrollo de aplicaciones web, proyectos de código abierto, aplicaciones móviles, etc. Todo ello, haciendo uso de las distintas herramientas que existen en el mercado [10], como *GitHub Actions*, *Jenkins*, *Travis CI*, *CircleCI*, *Azure DevOps*, entre otras.

El ciclo de vida de la Integración Continua, a pesar de ofrecer numerosas ventajas, conlleva grandes costos asociados debido a los recursos computacionales [11] necesarios para ejecutar las construcciones, comúnmente denominadas *builds*. A lo largo de este trabajo, nos referiremos como costo computacional al hecho de ejecutar una *build*, es decir, el proceso de construir el *software* y ejecutar todas las pruebas cuando la *CI* es lanzada. Este costo asociado se acentúa en empresas de gran tamaño, donde el número de *builds* que se ejecutan diariamente es muy elevado [12,13]. Además, hay que sumar la larga duración que pueden tene la ejecución de las *builds* en este tipo de empresas.

En los últimos años, han surgido numerosos enfoques centrados en reducir el costo computacional asociado a la ejecución de *CI* [2,4,5,6,7,8]. La idea principal de estos enfoques es reducir el número de *builds* que se ejecutan, prediciendo el resultado antes de su ejecución y, por lo tanto ahorrándose ese costo computacional. Las *builds* predichas como construcciones exitosas (*build pass*) no se ejecutan, mientras que las predichas como construcciones fallidas (*build failure*) sí se ejecutan. De esta forma, se mantiene el valor conceptual de la *CI*, que es la detección temprana de errores, pero reduciendo el costo computacional asociado en el proceso. Este estudio toma como punto de partida el algoritmo de *machine learning SmartBuildSkip* [2]. La idea principal es realizar una contribución a este algoritmo, usando features más significativas para predecir, o bien realizando una variante propia del mismo que mejore los resultados obtenidos. Por lo tanto, este estudio se enmarca en el desarrollo de software moderno, específicamente en el ámbito de la Integración Continua y la predicción automática del resultado de dicha integración.

La memoria queda organizada de la siguiente forma: en primer lugar, se estableceran los objetivos y preguntas de investigación que pretende reponder este trabajo. Posteriormente, se realizará un estudio del estado del arte que sitúe los antecedentes previos a la Integración Continua y la predicción automática de resultados de *builds*. A continuación, se describirá la solución propuesta, describiendo los detalles de la implementación, las tecnologías usadas. Después se presentarán las pruebas y resultados obtenidos, comparando la solución con otras existentes, a modo de validar y verificar la aportación. Seguidamente, se comentarán las amenazas a la validez, una parte esencial en cualquier trabajo de investigación. Este apartado nos permite identificar y discutir posibles limitaciones que podrían afectar a la validez de los resultados y a las conclusiones. Por último, se darán unas conclusiones sobre los resultados obtenidos y se propondrán posibles líneas de trabajo futuro.

2. Objetivos y Preguntas de Investigación

Continua...

Referencias

1. Xianhao Jin and Francisco Servant. 2023. HybridCISave: A Combined Build and Test Selection Approach in Continuous Integration. ACM Trans. Softw. Eng. Methodol. 32, 4, Article 93 (July 2023), 39 pages. https://doi.org/10.1145/3576038
2. Xianhao Jin and Francisco Servant. 2020. A cost-efficient approach to building in continuous integration. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20). Association for Computing Machinery, New York, NY, USA, 13–25. https://doi.org/10.1145/3377811.3380437
3. Xianhao Jin and Francisco Servant. 2021. CIBench: a dataset and collection of techniques for build and test selection and prioritization in continuous integration. In Proceedings of the 43rd International Conference on Software Engineering: Companion Proceedings (ICSE '21). IEEE Press, 166–167. https://doi.org/10.1109/ICSE-Companion52605.2021.00070
4. Xianhao Jin and Francisco Servant. 2022. Which builds are really safe to skip? Maximizing failure observation for build selection in continuous integration. J. Syst. Softw. 188, C (Jun 2022). https://doi.org/10.1016/j.jss.2022.111292
5. Islem Saidani, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. 2020. Predicting continuous integration build failures using evolutionary search. Inf. Softw. Technol. 128, C (Dec 2020). https://doi.org/10.1016/j.infsof.2020.106392
6. Xianhao Jin and Francisco Servant. 2023. HybridCISave: A Combined Build and Test Selection Approach in Continuous Integration. ACM Trans. Softw. Eng. Methodol. 32, 4, Article 93 (July 2023), 39 pages. https://doi.org/10.1145/3576038
7. Bihuan Chen, Linlin Chen, Chen Zhang, and Xin Peng. 2021. BuildFast: history-aware build outcome prediction for fast feedback and reduced cost in continuous integration. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20). Association for Computing Machinery, New York, NY, USA, 42–53. https://doi.org/10.1145/3324884.3416616
8. Foyzul Hassan and Xiaoyin Wang. 2017. Change-aware build prediction model for stall avoidance in continuous integration. In Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '17). IEEE Press, 157–162. https://doi.org/10.1109/ESEM.2017.23
9. O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. A. Ernst and M. -A. Storey, Üncovering the Benefits and Challenges of Continuous Integration Practices,ïn IEEE Transactions on Software Engineering, vol. 48, no. 7, pp. 2570-2583, 1 July 2022, doi: 10.1109/TSE.2021.3064953.
10. Rostami Mazrae P., Mens T., Golzadeh M., Decan A. On the usage, co-usage and migration of CI/CD tools: A qualitative analysis (2023) Empirical Software Engineering, 28 (2), art. no. 52, Cited 15 times. DOI: 10.1007/s10664-022-10285-5
11. M. Hilton, T. Tunnell, K. Huang, D. Marinov and D. Dig, Üsage, costs, and benefits of continuous integration in open-source projects,"2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), Singapore, 2016, pp. 426-437.
12. Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE '16). Association for Computing Machinery, New York, NY, USA, 426–437. https://doi.org/10.1145/2970276.2970358
13. Kim Herzig, Michaela Greiler, Jacek Czerwonka, and Brendan Murphy. 2015. The art of testing less without sacrificing quality. In Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15). IEEE Press, 483–493.