Revolutionizing Software Development: Autonomous Software Evolution

D. Mlinarić*, J. Dončević†, M. Brčić‡, I. Botički§
Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
danijel.mlinaric@fer.hr* juraj.doncevic@fer.hr† mario.brcic@fer.hr‡ ivica.boticki@fer.hr§

Abstract—**This paper discusses a software development method focused on creating a self-adapting and evolving system using AI and ML techniques. The goal is to reduce the need for manual software updates, offering a solution that continuously adapts to changing requirements. To achieve this goal, an AI-based update model is presented, and a possible system is discussed. Use case example demonstrate the applicability of the update model in real-world scenarios. As a cloud-based solution, this method could ensure scalability and broad applicability across various industry sectors.**

***Keywords*—automatic software development, autonomous systems, software evolution, artificial intelligence**

I. INTRODUCTION

Given the rapid pace of technological development, where software products evolve quickly, the ability of software to adapt to rapidly changing requirements and evolve quickly is becoming a key competitive advantage. Therefore, there is a need for more agile and adaptive approaches to software development. Traditional methods of development and software updating are often slow and cannot effectively cope with this need for agility. Software companies face the challenge of maintaining and securing their products. Any failure to update or maintain can lead to serious security breaches and financial losses. This is particularly important in sectors such as finance and healthcare, where the security and reliability of software are critical. There is a clear opportunity and motivation to utilize advanced technologies such as AI and ML in addressing these challenges. These technologies offer opportunities for automation and intelligent decision-making that go beyond current capabilities. The presented approach represents the continuous integration of user feedback and semi-automated adaptation of software, which can lead to an improved end-user experience and higher product value.

The AI-based update model represents an important step in the development of automated software engineering. It focuses on the creation of systems that use artificial intelligence (AI), in particular generative models and machine learning (ML), to automate the process of software updating by extending the idea of dynamic software updating. This approach has two objectives. First, to enable the software to continuously adapt and evolve to changing end-user requirements and environmental conditions. Second, to reduce the need for constant human supervision and intervention in the software development process. In other words, to create an new model for software development - a model that is automated, adaptable, and future-oriented.

The idea of proposing a system based on this model stems from the current challenges in software engineering, where update processes are often slow, error-prone, and require significant resources. Existing solutions in the industry face challenges such as delays in implementing new functionalities, difficulties in maintenance, and security risks. One example is the 2011 incident with Amazon's Elastic Compute Cloud (EC2), where an update failure led to the disruption of online services hosted on Amazon EC2 [1], including popular websites and services. Another example is the case from 2022 when Rogers Communication experienced a service interruption due to a maintenance upgrade, affecting millions of users. The AI-based update model provides a solution to reduce the risk of such service interruptions through automation and intelligent update management. Furthermore, end users who are not enthusiastic about the current automatic update solutions would benefit from update systems that are more personalized and learn based on users' actions [2].

II. MOTIVATION

The inspiration for this approach comes from fundamental research of generative models, such as the work of Goodfellow et al. [3] on Generative Adversarial Networks (GANs) and the work of LeCun [4] and Bengio [5] on Deep Learning. These models offer a new paradigm for code generation and enable the creation of more efficient and reliable software solutions [6]. Furthermore, natural language processing (NLP) techniques, based on work such as that of Mikolov et al. [7], [8], enable the understanding and analysis of source code at a deeper semantic level such as in [9]–[11]. In the area of test automation and validation, this approach uses machine learning techniques to detect bugs and security vulnerabilities [12], based on the approaches described in the work of Kingma and Welling [13] on autoencoders.

Dynamic software updating facilitates the rapid integration of new code and ensures that the software remains relevant and effective. In the context of programming languages such as Java and C#, "hot-swapping" enables replacing code parts during program execution. The presented approach can use these mechanisms for code updates without interrupting or restarting the application. For

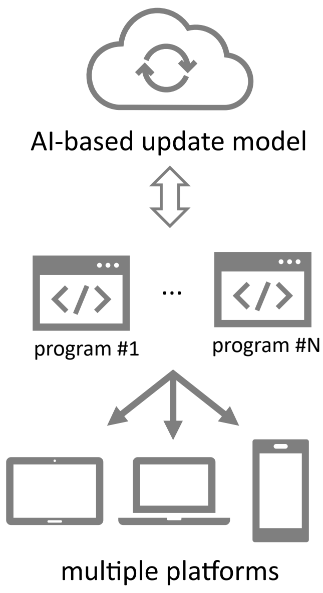

Fig. 1: Autonomous update system in cloud

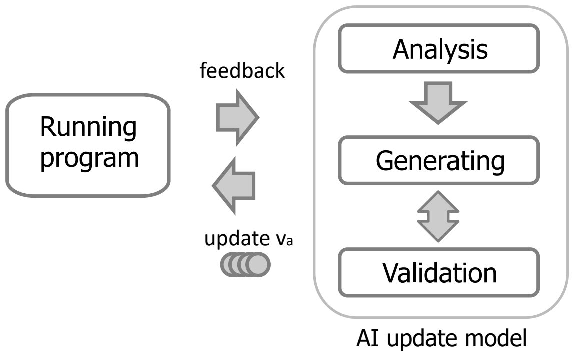

Fig. 2: Representation of the AI-based update model

example, the Java Platform Debugger Architecture (JPDA) [14] is a set of interfaces that enable development tools such as debuggers to communicate with Java applications. JPDA includes the Java Debug Interface (JDI), which allows the "hot-swapping" of code by replacing method code during execution [15]. Similarly, the "Hot reload" feature in .NET [16], for instance, for C#, allows developers to modify code while the program runs in debug mode. These mechanisms and their extensions [17] can be used to implement dynamic updating within the presented update model.

III. Available solutions

Currently, manual software updates as the norm in development often take too long due to the complexity of the code and the need for extensive manual testing. There are various software development and update automation tools and platforms that can be used to automate certain aspects of the development process, which can speed up and ease the process to some extent. Tools such as Jenkins [18], Travis CI [19], and GitLab CI/CD [20] enable continuous integration (CI) and continuous delivery (CD), automating the processes of testing and deploying software. This enables updates and new functions to be made available to end-users more quickly. On the other hand, automating code testing, e.g., with Selenium [21] or JUnit [22], helps to identify and fix problems in the software before it is released to production, ensuring a high-quality final product. Systems such as Nagios [23], Prometheus [24], and the ELK stack (Elasticsearch, Logstash, Kibana) enable the automation of application performance monitoring and logging, allowing problems to be quickly identified and resolved.

The approach presented in this paper sets itself apart from existing solutions through its ability to automate the software update process, relying on the latest research in the field of artificial intelligence and machine learning, and using sophisticated machine learning algorithms for real-time decision-making and adaptation. This represents a significant advance in the adaptability and efficiency of software solutions compared to current industry standards.

IV. Update Model

The update process in the model begins with a thorough analysis of the existing software. This analysis involves reviewing and understanding the software in the form of source code and configuration files using natural language processing (NLP) techniques. NLP is used to understand the structure and function of the code, taking into account existing research and techniques in the field. Methods such as tokenization and semantic analysis are used to understand the code and identify areas for improvement.

After the analysis, generative models take over. They use deep generative learning to develop and implement new parts of the code aiming to improve performance, correct errors, or add new functionalities. This approach finds parallels in industrial initiatives such as automated programming [25]–[27] and automatic software repair [28]–[30], but the presented update model goes one step further and integrates advanced AI techniques to enable automation, and improve autonomy and situational adaptability. A crucial capability to achieve is the continuous analysis of end-user feedback and external influences (e.g. changes in laws and regulations). This information adapts and optimizes the generated code to meet current user and environmental requirements. It also includes the development of advanced methods for automated testing and validation of new code using machine learning methods to detect errors and potential security vulnerabilities. This ensures that the updated code improves software functionality and a high level of security and reliability before the code is implemented into the existing software.

Dynamic updating, an essential part of the model, enables the updated code to be successfully integrated into the software in real-time so that generated changes can be performed quickly and without interrupting the end-user's workflow. With a focus on autonomy, adaptability, and continuous development, such an update model represents

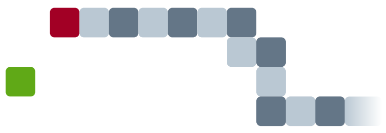
Fig. 3: Snake Game

a era of software development in which software continuously adapts and improves.

Based on the previous description, the key components of the update model are (Fig. 2):

1) Code analysis
2) Generating code adjustments
3) Testing and validation

These components are explained in more detail in the following subsections.

A. *Code analysis*

The update model could use Natural Language Processing (NLP) and Deep Learning (DL) techniques to analyze the existing code. Models such as BERT (Bidirectional Encoder Representations from Transformers) or GPT (Generative Pretrained Transformer) could be used to understand the context and structure of the code. Such analyses are focused on understanding the functionality of the code, identifying areas that need to be improved or optimized, and detecting potential errors or security vulnerabilities.

B. *Generating code adjustments*

The update model could use generative models such as GANs (Generative Adversarial Networks) or Variational Autoencoders (VAEs) to generate changes in the code. These models can be trained on large quantities of code data to learn how to generate functional and optimized code segments and effectively predict the necessary changes based on the analysis. Once analyzed, generative models can generate suggestions for updating or changing code automatically. This includes bug fixes, performance optimizations, and the introduction of new features that respond to new requirements or trends.

C. *Testing and validation*

In the testing and validation phase, techniques such as reinforcement learning (RL) can be used to simulate various application scenarios and test the robustness of the code. In addition, automated testing tools such as Selenium or JUnit can be integrated to check functionality and performance. This includes checking the correctness of the generated code, its compatibility with existing systems and security aspects. The system can also use advanced algorithms to evaluate the efficiency of the code and detect problems before updates are implemented.

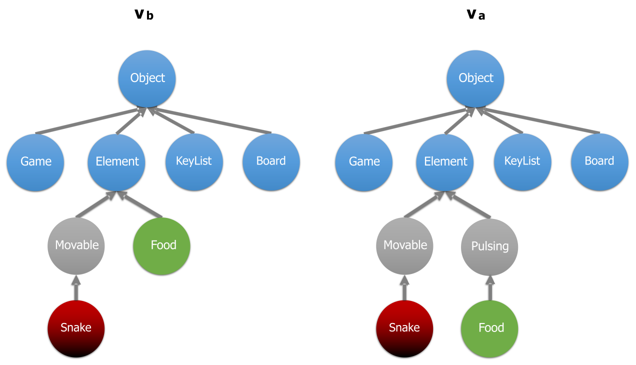

Fig. 4: Example of a class hierarchy update from basic version to generated version (v_b - basic, v_a - autonomously generated)

V. System features

A system developed according to the proposed model (Fig. 1) would enable the following features: continuous learning, multi-platform support, and security. The base version v_b represents the previously deployed version of the compiled code. The system performs adjustments on v_b to autonomously generate the v_a version (Fig. 4). This is the mode in which continuous updates would be carried out. In addition, cloud-based analysis would enable support for multiple platforms, regardless of the domain and programming language used. Security measures should be integrated by monitoring the latest findings on security protocols and issues so that the system can create patches to fix security vulnerabilities. Features are described in detail in the following subsections.

A. *Continuous learning and adaptation*

The system can use online learning and adaptive algorithms, such as online versions of machine learning algorithms, to continuously adapt and learn from new data. This allows the system to be updated and adapted in real-time without the need to regularly retrain the models. The system can continuously improve by collecting and analyzing data on performance and software usage, allowing it to better adapt to future requirements and trends in software development.

B. *Multiple platforms*

The system could be designed to support a wide range of programming languages and development environments, ensuring its applicability to different types of software projects. This flexibility allows system to be widely used in different software development contexts. The system can utilize technologies such as Docker and Kubernetes to support different programming languages and development environments, ensuring the system's flexibility and scalability and enabling its application to different types of software projects.

C. Security

Security is a critical component of any software system. Integrating advanced security protocols would ensure that all automated updates are secure and reliable, such as machine learning-based methods that can detect unusual patterns in the code that could indicate security vulnerabilities. Security integration, therefore, includes algorithms to detect and prevent security vulnerabilities and anomalies, as well as protocols to ensure data privacy and integrity.

D. System as a Service

An autonomous software evolution system could be implemented as a cloud-based solution that offers scalability and service diversity for system users, specifically targeting developers and companies focused on software development and maintenance rather than end-users directly. Limited functionality will suffice for small system users, while large system users will have access to more advanced system features. The limited functionality could take the form of less complex upgrades and a smaller number of upgrades, as envisioned by the concept. Functionality could range from user customization and adding new features based on external influences to security maintenance. In addition, system access could be based on subscription models, e.g., on a monthly basis or according to the number of upgrades with selected feature levels. The primary users of the system would be companies developing applications, offering end-users a unique experience and enhanced software security with continuous development potential. Potential system users could also include educational institutions for research and government agencies or organizations focused on security and data protection.

E. Potential limitations

Potential obstacles to the implementation and use of the system include the technical complexity of execution and the generalization of the derived models. In particular, models trained for specific business applications may not be efficient and adaptable to other domains, such as educational software or video games. This will be demonstrated by implementing the concept of autonomous software evolution in the form of limited functionality, as described in the following section. The limited functionality could be in the form of less complex upgrades and a smaller number of updates. Such an approach enables gradual testing and improvement. However, the generalization of the update model should be successfully demonstrated in a broader application.

On the other hand, no software can be delivered completely bug-free - due to potential bugs in the libraries used, scenarios overlooked by the developers, or unexpected user input. The key is to improve existing models to detect and fix logical errors without inadvertently introducing more severe ones. This emphasizes the need for advanced validation and sophisticated feature extraction methods to mitigate risk and highlights the need for further research in this area.

Possible challenges and limitations in validation include ensuring the system's robustness and reliability under different operating conditions and platforms. Compliance with ethical and legal standards, e.g., in industries where updates are critical, such as healthcare, transportation, and infrastructure, can limit the scope of automatic updates. In addition, maintaining transparency and explainability of the model's decisions, especially in the event of errors or unwanted updates, also needs to be considered.

VI. Autonomous Software Concept

The plan for the future is to develop an autonomous system as a proof of concept and to carry out simple updates. As a practical example to test the update model concept, the system's development and implementation for the popular 2D game *Snake* is planned. The goal is to enable the system to identify opportunities to improve the game, such as optimizing the end-user interface, enhancing AI opponents, or adding new levels and features. The system will analyze user interactions, feedback, and gameplay patterns, using machine learning techniques to identify areas for improvement.

The system can also be used to automatically balance difficulty levels to adjust game parameters to ensure an optimal gaming experience for players of different abilities. Another possible example is the development of new levels or challenges based on the popularity of existing game elements to automatically generate unique content that matches players' preferences and abilities.

Using deep learning and generative models, the system could not only passively monitor the game but also actively experiment with minor changes, learn from the results, and gradually introduce improvements. This method would provide valuable data about the end-user experience and interaction and enable dynamic, real-time software updates that ensure continuous development and improvement of the game. By gathering valuable data from the concept, insights can be gained into areas of future development.

A. Use case example

Let's take the example of the snake game: the player controls a moving snake that consumes food so that it grows and the player can collect points (Fig. 3). This game can be placed in a hypothetical environment that is monitored by an AI-based update system. A player provides feedback that the game is becoming repetitive and boring. Analyzing the player's feedback results in the requirement to change the game to keep the player engaged. Analyzing the existing code automatically provides the idea of generating the food so that it pulsates (appearing and disappearing) and can only be consumed at certain times. This is then used as a functional requirement to generate suitable code that is validated. Once successfully validated, the code update is applied to the running game - much to the player's surprise.

Fig. 4 shows the game versions in the form of a class hierarchy. On the left is the basic version v_b and the autonomously generated version v_a, on the right, introducing the pulsating behavior of the food.

VII. Conclusion

The AI-based update model is at the forefront of technological progress and focuses on the development of an autonomous system that uses artificial intelligence, in particular generative models, for dynamic software updates. This innovative approach combines the latest research in the field of machine learning and natural language processing and aims to achieve a level of autonomy and adaptability currently lacking in industrial and academic solutions. It will enable the software to adapt and update itself to changing requirements and conditions without the need for manual intervention.

The contribution of the presented AI-based update model lies in its ability to change the software development and updating process. To achieve this, an analysis of end-user feedback and the currently running software code is used. Based on the analysis results, decisions are made to make code adjustments that introduce new or changed features for users. Using the game Snake as an example, it is shown how the system could provide unique experiences for the end-user during the game based on the presented model.

The presented approach aims to streamline software updates, reduce costs, and enhance security. It also seeks to personalize end-user experience, making software more adaptable to individual needs. These improvements are expected to boost efficiency and safety, facilitating AI's broader application in sectors like healthcare, finance, and manufacturing, thereby revolutionizing software engineering towards autonomous, adaptable solutions.

References

[1] Amazon Web Services, "Summary of the amazon ec2 and amazon rds service disruption in the us east region," https://aws.amazon.com/message/65648/, 2011, accessed: 2024-01-22.

[2] A. Mathur and M. Chetty, "Impact of user characteristics on attitudes towards automatic mobile application updates," in *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, 2017, pp. 175–193.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, p. 139–144, oct 2020. [Online]. Available: https://doi.org/10.1145/3422622

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[5] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning*. MIT press Cambridge, MA, USA, 2017, vol. 1.

[6] H. Yu, B. Shen, D. Ran, J. Zhang, Q. Zhang, Y. Ma, G. Liang, Y. Li, Q. Wang, and T. Xie, "Codereval: A benchmark of pragmatic code generation with generative pre-trained models," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–12.

[7] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*. PMLR, 2014, pp. 1188–1196.

[8] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, 2013, pp. 746–751.

[9] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, 2019.

[10] S. Luan, D. Yang, C. Barnaby, K. Sen, and S. Chandra, "Aroma: Code recommendation via structural code search," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–28, 2019.

[11] K. Wang and Z. Su, "Blended, precise semantic program embeddings," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 121–134.

[12] C.-W. Tien, T.-Y. Huang, P.-C. Chen, and J.-H. Wang, "Using autoencoders for anomaly detection and transfer learning in iot," *Computers*, vol. 10, no. 7, p. 88, 2021.

[13] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[14] "JDPA Enhancements 1.4." [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/enhancements1.4.html#hotswap

[15] D. Mlinarić and V. Mornar, "Dynamic software updating in java: comparing concepts and resource demands," in *Companion Proceedings of the 1st International Conference on the Art, Science, and Engineering of Programming*, 2017, pp. 1–6.

[16] Microsoft, "Hot reload," 2023. [Online]. Available: https://learn.microsoft.com/en-us/visualstudio/debugger/hot-reload?view=vs-2022

[17] D. Mlinarić *et al.*, "Challenges in dynamic software updating," *TEM Journal*, vol. 9, no. 1, pp. 117–128, 2020.

[18] "Jenkins," https://www.jenkins.io/, accessed: 2024-01-22.

[19] "Travis ci - test and deploy with confidence," https://app.travis-ci.com/, accessed: 2024-01-22.

[20] "Get started with gitlab ci/cd," https://docs.gitlab.com/ee/ci/, accessed: 2024-01-22.

[21] "Selenium," https://www.selenium.dev/, accessed: 2024-01-22.

[22] "Junit," https://junit.org, accessed: 2024-01-22.

[23] "The standard in it infrastructure monitoring | nagios," https://www.nagios.com/, accessed: 2024-01-22.

[24] "Overview | prometheus," https://prometheus.io/, accessed: 2024-01-22.

[25] M. Nye, L. B. Hewitt, J. B. Tenenbaum, and A. Solar-Lezama, "Learning to infer program sketches," *ArXiv*, vol. abs/1902.06349, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:62841423

[26] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[27] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago *et al.*, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.

[28] M. Monperrus, "Automatic software repair: A bibliography," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–24, 2018.

[29] J. Bader, A. Scott, M. Pradel, and S. Chandra, "Getafix: Learning to fix bugs automatically," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–27, 2019.

[30] Z. Fan, X. Gao, M. Mirchev, A. Roychoudhury, and S. H. Tan, "Automated repair of programs from large language models. in 2023 ieee/acm 45th international conference on software engineering (icse)," *IEEE Computer Society, Los Alamitos, CA, USA*, pp. 1469–1481, 2023.