An IDE Plugin for Clone Management

Ahmad Al Shihabi
Ruhr-University Bochum
Germany, Bochum

Jan Sollmann
Ruhr-University Bochum
Germany, Bochum

Johan Martinson
Ruhr-University Bochum
Germany, Bochum

Wardah Mahmood
Chalmers | University of Gothenburg
Sweden, Gothenburg

Thorsten Berger
Ruhr-University Bochum and
Chalmers | University of Gothenburg
Germany, Bochum

ABSTRACT

Development and maintenance in variant-rich systems often involves the replication of specific software code, known as *software cloning*. This process allows for code reuse but presents challenges in managing independently evolving variants. This paper discusses the necessity of effective clone management tools to maintain code quality and efficiency. We present an extension[1] to the HAnS IDE plugin. This extension enhances the plugin by supporting basic clone management, and by facilitating the tracking and synchronization of cloned assets and features through a well-designed, lightweight trace database. The plugin is evaluated through unit and integration testing, as well as user experiments, demonstrating its effectiveness in addressing the challenges associated with software cloning. The evaluation results indicate that 80% of participants rated the trace database as intuitive, and 100% rated the notification system as both intuitive and user-friendly.

CCS CONCEPTS

• **Software and its engineering** → **Software product lines**; **Software maintenance tools**.

KEYWORDS

feature-oriented software evolution, embedded feature annotations, tool support, feature asset management, Software Evolution, IDE

ACM Reference Format:
Ahmad Al Shihabi, Jan Sollmann, Johan Martinson, Wardah Mahmood, and Thorsten Berger. 2024. An IDE Plugin for Clone Management. In *28th ACM International Systems and Software Product Line Conference (SPLC '24), September 02–06, 2024, Dommeldange, Luxembourg*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3646548.3678298

1 INTRODUCTION

In the context of developing variant-rich systems, companies typically choose between two strategies: software configuration with an integrated platform or the *clone&own* approach.

Clone&Own involves replicating code segments, configurations, or entire components, enabling code reuse in different contexts. It provides an immediate, cost-effective solution but faces the risk of bug propagation as well as scalability challenges. It leads to independently evolving variants that are difficult to manage and synchronize. Developers often lose track of which variant was cloned from which, increasing maintenance costs due to the need for consistently applying changes across all clones [14, 17]. This process typically involves clone detection techniques to identify and manually apply changes to all related variants.

Alternatively, software configuration involves building a configurable platform from scratch. A configurable platform is a comprehensive system or framework that supports the development and deployment of software applications. It offers a set of predefined functionalities and components that can be customized and extended to meet specific needs. While this approach provides ultimate flexibility and scalability, it is time-consuming, expensive, and requires substantial domain-specific expertise [9]. Given the rapid pace of technological advancement, this method is not always feasible.

This is where clone management tools come into play, providing a good middle ground. Clone management involves tracing cloned assets and features to maintain synchronization and manage changes across variants. By leveraging traceability, clone management tools help identify relationships between clones, ensuring that changes such as bug fixes are consistently applied across all variants, thereby reducing maintenance costs and minimizing the risk of bug propagation [5, 7, 12, 14, 19].

Clone management requires a representation of clone tracing and concrete developer support that is closely integrated with mainstream development tooling. This integration is essential to provide developers with the necessary context and tools to efficiently manage and synchronize cloned assets. By embedding clone management functionalities within familiar development environments, we can streamline the process, reduce overhead, and enhance the overall effectiveness of clone management [7, 11, 12, 19].

This paper presents a lightweight clone management tool, demonstrates its effectiveness through empirical validation, and discusses its impact on software development processes. In that regard, we present an extension of the Helping Annotate Software (HAnS) IDE plugin [10], which supports the recording of feature locations [6, 16] in source code. Our extension enhances HAnS by incorporating clone management capabilities, utilizing a well designed trace database that is lightweight and comprehensible, as confirmed by our evaluation. This database facilitates easier tracking and synchronization of cloned assets to their features. This aligns with our

[1]Demo video: https://youtu.be/cCG_-NBlVgo

SPLC '24, September 02–06, 2024, Dommeldange, Luxembourg

ACM ISBN 979-8-4007-0593-9/24/09
https://doi.org/10.1145/3646548.3678298

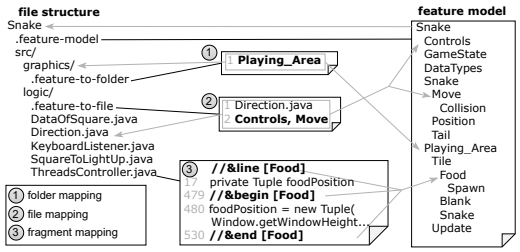

Fig. 1: Illustration of feature models and feature locations in a sample file structure [10, 13, 18].

vision for robust feature tracking in clone management, ensuring consistent and efficient management across software variants.

2 BACKGROUND & CHALLENGES

Effective feature tracking enhances code comprehensibility and clone management in software development. Embedding feature tracking in code helps developers identify and manage features in clones, ensuring better understanding and maintenance. To further illustrate how HAnS supports feature tracking, it records feature locations in source code as shown in Fig. 1. It can be used to create mappings of folders, files and code fragments to features defined in a feature model using a lightweight embedded feature annotations system. Files and folders are mapped with textual files while code fragments are in-lined as comments within code. The annotations can be used to browse features and their usages in the assets, as well as to facilitate refactoring [10].

Specifically, our plugin aims to overcome the following challenges associated with proposing a lightweight, scalable and practicable tool for real-world software projects:

Challenge 1: Granularity of the clone traces. One of the principal concerns is the identification of the elements that should be traced by the clone management tool in order to ensure its suitability for real-world practical applications. It is essential that only relevant and significant data is shown in traces to make our traces purposeful and clear for users. Furthermore, for a feature-oriented tool, it is necessary to trace the features implemented in assets.

Challenge 2: Synchronization of traces. In a real-world environment, the synchronization of assets should ensure that changes made to the original assets are immediately reflected in the clones. This necessitates tooling that is both lightweight and efficient in terms of monitoring for changes occurring in assets.

Challenge 3: Integration with mainstream developer tools. Ensuring the usability of the clone management tool is critical to its adoption in real-world projects. A key aspect of usability is the inclusion of clone records that allows users to review the clone history within the current application. This feature provides users with vital data about the existing clones in their project, which will enable them to better manage and understand their code base.

In addition, the tool maintains the consistency of the feature model, even when dealing with clones that contain features that are not currently mapped to the existing feature model. It is essential for accurate feature tracking and management to ensure that all implemented and named features are present in the feature model. Another key option is the ability to synchronize assets. This feature is particularly valuable as it ensures that users do not miss any maintenance changes and keeps their assets consistent and up to date.

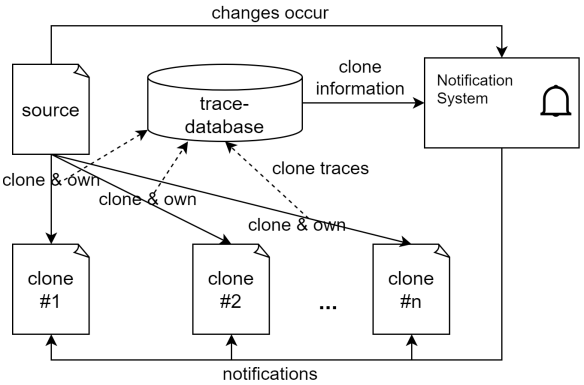

Fig. 2: Clone management system. Traces are stored in the trace database on any clone action and used by the notification system to create banners in the clone files given the source file changed

(1) `src/pojo/Tuple.java;src/graphics/TupleClone.java; 20240602192648;#Gretel`

(2) `Snake/src/pojo;src/pojo;20240602195455;#Gretel Snake::GameState;SnakeTest::GameState`

(3) `Snake::Size;SnakeTest::UNASSIGNED::Size`

Fig. 3: Traces generated by the plugin illustrating the structure of the trace database. Each clone action is recorded with timestamps and feature annotations if existent.

3 THE PLUGIN

Implementation. The Clone Management Plugin uses a robust 3-layer architecture for effective cloned asset tracking and management. The User Interaction Layer detects copy, paste, and file open actions within the IDE through a CopyPastePreProcessor, using before()/after() methods to capture source and resulting clone paths for accurate tracking. The Middleware Layer enables communication between tool components and identifies features assigned to folders or files. The Clone Management and Notification Layer employs PSI language injection to locate feature implementations and displays banners to notify users of relevant changes. This design ensures the plugin is efficient and user-friendly.

To illustrate the functionality of the Clone Management Plugin, consider a scenario where a company is developing a system with features that must be extracted from an existing system. The assets representing the implementation of these features are cloned. By cloning these assets, the clone management tool creates a text file called ".trace-db" in the project root of the current system. This file acts as a database for all clone traces, recording and managing each instance of a clone action independently. Additionally, the necessary feature model maintenance is organized. The tool informs developers of the cloned file assets that exist in the project and the source changes that need to be checked. This process is illustrated in Fig. 2, which shows how the tool tracks and manages clone actions.

3.1 Trace Database

To address the stated challenges we need some sort of database so that users can view the clone actions within the current application. To achieve this, the tool is accompanied by a textual trace database which is inspired by a traceability mechanism (asset referencing

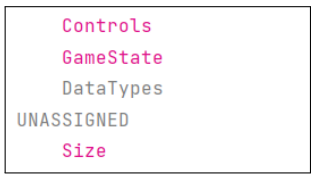

Fig. 4: Feature model with unassigned feature "Size"

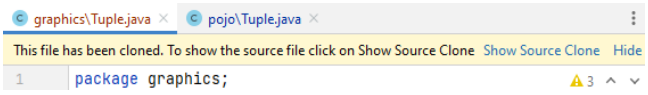

Fig. 5: Editor panel as clone notification and action selection

scheme) from the Virtual Platform [8, 9], which stores the relative paths of several clones of asset types. The traces comprise the source asset path, target asset path and the version number, which indicates the point in time at which the clone was created as well as the Git username of the individual responsible for the clone (see. Fig. 3 Item 1). While relative paths are stored, absolute paths are mapped to them using a persistent Hash Map. This ensure readability and portability as well as platform in-dependency. In the case of clones that originate from external instances, the trace starts with the instance name. Each asset clone is identified through the use of feature annotations, which are mapped to it by the HAnS tool. These annotations consist of the source instance name, target instance name, and feature name, all separated by colons (see. Fig. 3 Item 2). Any features that cannot be mapped to the target feature model are indicated by an *UNASSIGNED* prefix before the actual feature name (see. Fig. 3 Item 3). This indicates that the feature in question could be found under the lazy-initialized *UNASSIGNED* feature in the current feature model as shown in Fig. 4.

3.2 Cloning of Code Assets

Accurate tracking of cloning actions is crucial. Regardless of whether the clone action involves a file, folder or a textual clone (i.e., a method, class, or block of code representing features within or outside a method), the tool records this data in a trace database, providing significant information about the clones. The tool provides the relevant asset paths, beginning with the project name and ending with the asset name as it is represented in the abstract syntax tree (AST). The second item in the list Item 2, presents an example of a folder clone. The tool listens to these clone actions due to the implementation of lightweight services, which then perform the appropriate techniques dependent on the asset type.

The trace is stored in the trace database at the time the asset is pasted into the project. The variant number, which indicates the time of the clone, is expressed in a format that includes the year, month, day, hour, minutes, and seconds.

3.3 Synchronization of Cloned Code Assets

Notifications, as shown in Fig. 5, represent a fundamental component of the clone management system in HAnS. For each file that is cloned and traced in our trace database, a panel is displayed at the top of the file, informing the user that the file has been cloned. Clicking on the "Show Source File" option will redirect the user to the original file from which the clone was created. The user also has the option to hide this notification by clicking on the "Hide"

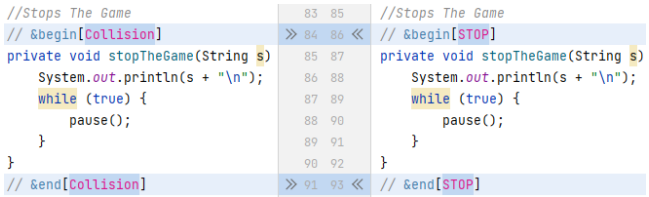

Fig. 6: Merge window for displaying differences and editing

button. Upon modification of a source file, a cloned file will receive a notification of the changes. This notification banner can be viewed again at the top of the editor, informing the user that the source file has been modified. The user has two options to choose from: one for merging the changes that occurred in the source file, and the other for hiding the banner. Clicking on the "Merge" button will open a merge window in the editor, which displays the differences between the two files. As with any merge window in the JetBrains platform, the user can select which code changes to replicate in the currently cloned file, as shown in Fig. 6. These modifications will be found directly in the cloned file when it is opened again.

4 EVALUATION

To evaluate the clone management tooling, we performed unit and integration tests[2] for assessing the technical performance, and conducted a user study to determine the practical usability of the tool.

User Experience. User experience feedback was gathered through an experiment, conducted with a doctoral supervisor at a software company and four student developers. Three of whom had a bachelor's degree and one of whom was an undergraduate. Most of the users were familiar with the HAnS plugin. Users experimented with the tool and created several code clones. They started by cloning folders and files within the same project and into other open projects. During cloning, they were prompted to check the trace database and verify the results. They then performed several editor clones, such as cloning methods with features mapped by block annotation and by line annotation (see Fig. 1). The participants were instructed to clone assets from disparate projects that were mapped to features that did not exist in the target feature model. This was done to facilitate experimentation with the tracking of unmapped features. Some of the cloned source files were then modified to test the notification and merging process. This entailed switching to the source files by triggering the notification button and merging the changes into the cloned files by using the editor notification option again.

After performing these tasks, the users were asked to judge the clone management features regarding their usability and intuitiveness. As observable in Fig. 7, the majority of participants found the design of the trace database intuitive, and even 100% of participants found the notification system intuitive. Just one respondents would not use the tool on a daily basis. They were particularly impressed by the various editor notifications provided with different actions in a real-time environment. One main improvement suggestion was to extend the trace for a second-level clone and beyond, i.e. clones cloned from other clones should have the paths up to the source asset in their trace. This would ensure that some clones can still be found even if the series of clones is interrupted somewhere in the trace e.g. by deleting the first clone, we can still find the source asset.

[2]GitHub: https://github.com/isselab/HAnS/tree/feature/vpIntegration-cloning

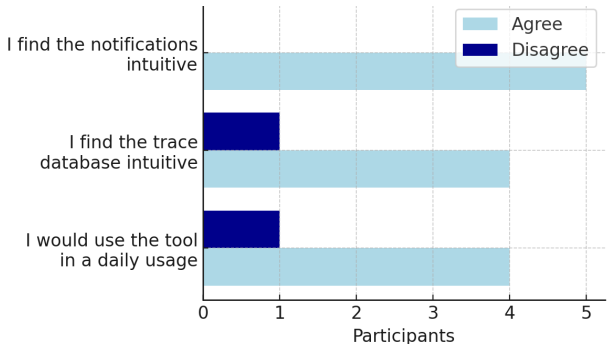

Fig. 7: Evaluation results of user feedback

In conclusion, the evaluation indicates that the IDE plugin for clone management is a robust and user-friendly tool that effectively aids in the management of code clones.

Future Work. Future work will address user feedback on keeping traces of cloned clones. While renaming and changes within clones are already managed by the file system and version control, adding consistency checks could enhance reliability. Additional functionalities like change filtering and a UI sidebar could improve user experience. Although the trace database is lightweight and scales well, using index structures or hashing could boost performance in large projects. The current solution, which involves the addition of developers' Git names to each trace, should be replaced by an automatic synchronization strategy to avoid merging conflicts.

5 RELATED WORK

Clone management in software engineering has been studied before [15]. An exploratory study on cloning in industrial software product lines [2] has shown that cloning is still seen as a beneficial and natural reuse approach by the majority of surveyed practitioners.

Ji et al. [4] show that manually tracking and updating traceability information for cloning and synchronization of software variants is costly and error-prone. Barbosa et al. [1] found that maintaining cloned software parts takes on average 136% more time. Heisig et al. [3] introduced a traceability metamodel to establish uniform traceability workflows in a variability-aware, model-based environment.

Pfofe et al. [14] and Kehrer et al. [5] described VariantSync, a tool to automate synchronization of software variants, reducing the gap between clone&own and product lines. Schmorleiz et al. [17] described managing the similarity of cloned-and-owned variants over time using annotations and automatic change propagation. Montalvillo's studies [11, 12] proposed enhancements to version control systems to support synchronization paths for software product lines and introduced *peering bars* for better awareness of feature upgrades. Lillack et al. [7] developed a tool to aid in integrating cloned variants into a configurable platform using domain-specific integration intentions and visualizations. Zibran et al. [19] provided a roadmap for clone management, highlighting the importance of tools for detecting, managing, and refactoring clones.

Mahmood et al. [8, 9] introduced the virtual platform, an approach to support the incremental development of variant-rich systems. It also relies on clone traces and features. Our tool is inpsired by concepts described by the virtual platform. While the latter offers a clone-management framework, it does not provide a user interface beyond a command-line interface and suggestions for tool integration. Our plugin integrates with mainstream IDEs, implementing virtual platform concepts.

Unlike previous tools that focus on either clone&own or virtual platforms, our plugin bridges these two approaches through a lightweight IDE plugin, reducing the cost for developers and improving overall efficiency in clone management.

6 CONCLUSION

We outlined the need for a clone management system and emphasized its importance in maintaining code quality and efficiency in software development. We demonstrated the effectiveness of our plugin rigor testing and user experiments. Our tool is lightweight, and the trace database can improve the development and maintenance and practices in clone&own development. Future work aims to refine and extend the current validation to confirm these anticipated benefits, as well as extend tool functionalities as discussed.

REFERENCES

[1] Jefferson Barbosa, Rossana Andrade, João Filho, C. I. M. Bezerra, Isaac Barreto, and Rafael Capilla. 2018. Cloning in Customization Classes: A Case of a Worldwide Software Product Line. In *SBCARS*.
[2] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An Exploratory Study of Cloning in Industrial Software Product Lines. In *CSMR*.
[3] Philipp Heisig, Jan-Philipp Steghöfer, Christopher Brink, and Sabine Sachweh. 2019. A generic traceability metamodel for enabling unified end-to-end traceability in software product lines. In *SAC*.
[4] Wenbin Ji, Thorsten Berger, Michal Antkiewicz, and Krzysztof Czarnecki. 2015. Maintaining feature traceability with embedded annotations. In *SPLC*.
[5] Timo Kehrer, Thomas Thüm, Alexander Schultheiß, and Paul Maximilian Bittner. 2021. Bridging the gap between clone-and-own and software product lines. In *ICSE, NIER*.
[6] Jacob Krüger, Thorsten Berger, and Thomas Leich. 2018. *Features and How to Find Them: A Survey of Manual Feature Location.* Taylor & Francis Group, LLC/CRC Press.
[7] Max Lillack, Stefan Stanciulescu, Wilhelm Hedman, Thorsten Berger, and Andrzej Wąsowski. 2019. Intention-Based Integration of Software Variants. In *ICSE*.
[8] Wardah Mahmood, Gül Calikli, Daniel Strüber, Ralf Lämmel, Mukelabai Mukelabai, and Thorsten Berger. 2024. Virtual Platform: Effective and Seamless Variability Management for Software Systems. *IEEE Transactions in Software Engineering (TSE)* (2024).
[9] Wardah Mahmood, Daniel Strueber, Thorsten Berger, Ralf Lämmel, and Mukelabai Mukelabai. 2021. Seamless variability management with the virtual platform. In *ICSE*.
[10] Johan Martinson, Herman Jansson, Mukelabai Mukelabai, Thorsten Berger, Alexandre Bergel, and Truong Ho-Quang. 2021. HAnS: IDE-Based Editing Support for Embedded Feature Annotations. In *SPLC, Tools and Demonstrations*.
[11] Leticia Montalvillo and Oscar Díaz. 2015. Tuning GitHub for SPL development: branching models & repository operations for product engineers. In *SPLC*.
[12] Leticia Montalvillo, Oscar Díaz, and Thomas Fogdal. 2018. Reducing coordination overhead in SPLs: peering in on peers. In *SPLC*.
[13] Mukelabai Mukelabai, Kevin Hermann, Thorsten Berger, and Jan-Philipp Steghöfer. 2023. FeatRacer: Locating Features Through Assisted Traceability. *IEEE Transactions on Software Engineering* 49, 12, 5060–5083.
[14] Tristan Pfofe, Thomas Thüm, Sandro Schulze, Wolfram Fenske, and Ina Schaefer. 2016. Synchronizing software variants with variantsync. In *SPLC, Tools and Demonstrations*.
[15] Chanchal K. Roy, Minhaz F. Zibran, and Rainer Koschke. 2014. The vision of software clone management: Past, present, and future (Keynote paper). In *CSMR-WCRE*.
[16] Julia Rubin and Marsha Chechik. 2013. *A Survey of Feature Location Techniques.* Springer, Berlin, Heidelberg, Germany, 29–58. https://doi.org/10.1007/978-3-642-36654-3_2
[17] Thomas Schmorleiz and Ralf Lämmel. 2016. Similarity management of 'cloned and owned' variants. In *SAC*.
[18] Tobias Schwarz, Wardah Mahmood, and Thorsten Berger. 2020. A Common Notation and Tool Support for Embedded Feature Annotations. In *SPLC, Demonstrations and Tools*.
[19] Minhaz F Zibran and Chanchal K Roy. 2012. The road to software clone management: A survey. *Dept. Comput. Sci., Univ. of Saskatchewan, Saskatoon, SK, Tech. Rep* 3 (2012).