Artifacts Available CBSoft 2025

Artifacts Functional CBSoft 2025

Highlight Test Code: Visualizing the Co-Evolution of Test and Production Code in Software Repositories

Charles Miranda
Federal University of Piauí
Teresina, Piauí, Brazil
charlesmiranda@ufpi.edu.br

Guilherme Avelino
Federal University of Piauí
Teresina, Piauí, Brazil
gaa@ufpi.edu.br

Pedro Santos Neto
Federal University of Piauí
Teresina, Piauí, Brazil
pasn@ufpi.edu.br

ABSTRACT

The asynchronous evolution of test and production code can compromise software quality and maintainability. However, identifying and analyzing such co-evolution dynamics remains a complex task, often hindered by the lack of scalable tools and comprehensive visualizations. This paper presents *Highlight Test Code*, a web-based platform for analyzing the co-evolution of test and production code in open-source repositories. The platform implements a multi-stage pipeline encompassing repository mining, time series generation, clustering, and statistical correlation. It is built upon a curated dataset of 526 GitHub repositories across six programming languages. *Highlight Test Code* offers visual analytics and AI-generated insights that assist users in identifying test evolution patterns, evaluating co-evolution levels, and understanding their relationship with maintenance activities and project characteristics. The tool enables researchers and practitioners to assess testing practices at scale.

Video link: https://youtu.be/U29eEg_gXXM.
Software License: General Public License (GPL).

KEYWORDS

Test code co-evolution, software testing, software repository mining

1 Introduction

Continuous maintenance and adaptation are essential for software systems to remain relevant over time [10, 11]. As projects evolve to meet new requirements and resolve bugs, test code must co-evolve with production code to preserve quality and maintainability [13, 15]. Given the critical role of software in modern society, ensuring its correct functioning is essential [17, 23]. In this context, software testing plays a fundamental, yet complex, role [5]. However, managing the co-evolution of production and test code remains a significant challenge in software development.

Several studies have proposed visualizations to represent the co-evolution of code and tests, offering insights into development dynamics and supporting informed decision-making [12, 13, 26, 28]. These visual approaches help identify how changes in production and test code relate over time, enabling project managers to recognize patterns, evaluate testing strategies, and allocate resources more effectively. Nevertheless, most prior work has focused on limited samples or single-language contexts, restricting broader generalizations.

To address these limitations, this work introduces *Highlight Test Code*, a platform designed to support large-scale, interactive exploration of co-evolution behaviors in software projects. The tool integrates repository mining, time series analysis, clustering algorithms, and statistical correlation to identify co-evolution patterns between test and production code. In addition to these core features, it offers AI-generated commentary as a complementary resource to help users interpret the results and gain deeper insights. The platform provides a user-friendly and extensible environment for both research and practice. The platform targets researchers and software developers alike. Researchers can perform large-scale test evolution studies, while developers can use its visual analytics for evaluating testing, monitoring maintenance, and guiding technical decisions. Highlight Test Code offers a user-friendly, extensible environment for both academic and practical software engineering purposes.

Highlight Test Code is built upon a dataset of 526 open-source GitHub repositories spanning six programming languages. The platform allows users to explore both personal and community-analyzed projects, examine test evolution clusters, and investigate how test co-evolution correlates with team characteristics, development activity, and maintenance behavior. Furthermore, it supports the registration and processing of new repositories through an automated pipeline that extracts metrics and generates visual analytics. The continuous extraction of new data not only expands the dataset but also contributes to the refinement of the underlying metrics used in the research, enabling the generation of more accurate highlights and analytical insights over time.

This paper is organized as follows. Section 2 presents background concepts and related work on test and production code co-evolution and compares our approach with existing tools and studies. Section 3 introduces the *Highlight Test Code*, outlining its main features and architecture. Section 4 illustrates a usage scenario of the tool. Section 5 discusses the tool's current limitations. Section 6 concludes the paper and outlines directions for future work.

2 Related work

This section provides an overview of related work, focusing on the literature concerning the co-evolution of test and production code. These studies highlight the diversity of co-evolution scenarios and underscore the need for deeper, large-scale exploration of test-related co-evolution—an aspect that serves as the primary motivation for our study.

2.1 Test Co-evolution

Several studies have explored test and production code co-evolution using association rule mining. Marsavina et al. [13] identified co-evolution patterns in five Java projects, noting that well-tested systems tend to exhibit more positive patterns. Vidács et al. [26] extended this work by analyzing a single project with added dimensions such as project properties and commit structures, finding both synchronized and independent evolution between production and test code.

Other researchers have adopted different perspectives to study co-evolution. Zaidman et al. [28] examined co-evolution through change history, growth history, and test coverage evolution, concluding that concurrent modifications to production and test code are the most common. Levin et al. [12] focused on semantic aspects of co-evolution and showed that production code changes do not always trigger corresponding updates in tests. Their predictive models suggest that the nature of the maintenance activity influences test maintenance behavior.

Recent work has also emphasized how co-evolution insights can improve test automation and maintenance. Studies by Gonzalez et al. [7], Hu et al. [8], and Shimmi et al. [21, 22] propose tools and models to identify outdated tests and recommend updates. Yalçın et al. [27] contributed with a visualization approach to support understanding the interplay between code and tests. Sun et al. [24] emphasized the need for advanced tools to detect co-evolution patterns. These works collectively inform our approach, particularly Gonzalez et al.'s catalog for identifying test files and Zaidman et al.'s multi-perspective framework for analyzing co-evolution dynamics.

2.2 Related tools

Several tools have been proposed to address the co-evolution of production and test code. We compare *Highlight Test Code* with three representative tools, highlighting its unique contributions.

ChronoTwigger [6] stands out as a visual analytics tool designed to explore the co-evolution of source and test files using both 2D and 3D visualizations. It leverages the concept of *co-change*—files that are frequently modified together—to group and visualize source and test artifacts across the project's timeline. While ChronoTwigger excels at spatial and temporal visualization, especially through immersive 3D exploration, it is primarily focused on a limited set of repositories and requires specialized infrastructure for full interactivity.

Another tool, METEOR [25], focuses on monitoring the behavioral preservation of test code refactorings. It uses static and dynamic analysis to ensure that changes in test code do not alter their intended verification purposes. While powerful in the context of regression safety and refactoring validation, METEOR does not provide insights into the long-term evolution of production and test code together.

More recently, AutomTest 3.0 [20] proposed the use of large language models (LLMs) to automate test generation from user stories. While it offers innovation in automated test case creation, its primary focus is on generating new tests rather than understanding or analyzing the evolution of test-production relationships over time.

Unlike these tools, *Highlight Test Code* offers a unique combination of large-scale repository mining, time series clustering, co-evolution metric analysis, and AI-generated insights within an accessible web-based interface. Specifically, it enables researchers and practitioners to identify distinct test evolution patterns, explore correlations with project characteristics, and gain actionable insights into testing practices across a diverse range of programming languages and project sizes. Therefore, *Highlight Test Code* addresses the need for a scalable, multi-language, and interpretable platform for analyzing test co-evolution.

3 Highlight test code

In this section, we present the construction of the dataset that powers the tool and describe the system's architectural design. We detail the project selection process, filtering criteria, and characteristics of the final dataset, followed by an overview of the backend and frontend components that support analysis and visualization. Together, these elements establish *Highlight Test Code* as a practical and extensible platform for studying test co-evolution in the open source ecosystem.

3.1 Dataset construction

The *Highlight Test Code* platform relies on a curated dataset comprising 526 open-source software projects, selected to support the analysis of test and production code co-evolution in real-world settings. This dataset was constructed as part of a prior empirical study conducted by the authors [16]. Beyond enabling large-scale and comparative analyses of test evolution patterns across diverse development contexts, the dataset plays a critical role in establishing reference values and behavioral baselines for the metrics and analyses provided by the tool.

The projects were collected from GitHub, a popular platform for collaborative software development, using the GitHub REST API. We targeted repositories from six widely-used programming languages—JavaScript, TypeScript, Java, Python, PHP, and C#—to ensure language diversity and representation of varied development ecosystems. The initial selection process involved retrieving the 500 most starred repositories for each of the six target languages, totaling 3,000 projects. The number of GitHub stars serves as a proxy for community relevance and adoption, as commonly employed in empirical software engineering studies [2–4]. By focusing on highly visible projects, we aim to capture mainstream development practices, including those related to automated testing.

To narrow the scope and focus on projects with meaningful testing activity, we applied a filtering process based on key metrics: percentage of test files, percentage of test-related lines of code (LOC), project age, and number of contributors. Projects below the 25th percentile in any of these metrics were excluded to ensure the remaining dataset contained repositories with sufficient investment in testing. This process resulted in a refined collection of 526 projects, distributed across the six languages, and representing a wide range of sizes, ages, and levels of community engagement.

The resulting dataset exhibits considerable diversity. The median project age is 7 years, with interquartile ranges highlighting substantial variation in a number of files (27 to 459), commits (181 to 1,884), and team sizes (11 to 101 developers). The median number of forks (441) and open issues (43) also indicate moderate community involvement. These characteristics reflect the maturity and relevance of the selected projects and provide a robust foundation for studying test co-evolution patterns at scale.

As new project submissions are shared with the tool community, we continually integrate additional projects into our dataset. This continued expansion increases the representativeness and longevity of our dataset, allowing us to refine and replicate our findings across the broader open source software development landscape.

3.2 Test co-evolution extraction

To analyze co-evolution between production and test code across diverse software projects, the study employs two complementary time series formats: fixed-length and variable-length. This dual approach allows for both consistent comparison across projects and alignment with common practices in software evolution research.

The fixed-length group consists of time series with 75 commits per project, enabling uniformity for clustering analysis of test evolution patterns. The number 75 was chosen as it represents the maximum number of commit intervals that could be consistently extracted across all 526 repositories, ensuring both granularity and computational feasibility.

The variable-length group aggregates code changes by month and year, allowing the analysis of long-term co-evolution trends. This temporal aggregation captures meaningful development patterns and is supported by prior empirical studies, which often use monthly intervals to study software evolution over time.

For both groups, the study uses the `cloc` tool to extract test and production lines of code at each interval. From these measurements, a standardized metri, called testRatio, is calculated to represent the percentage of test code in relation to the total source code. This approach enables the identification of co-evolution dynamics within individual projects and supports comparative analysis across the dataset.

3.3 Test evolution clusterization

To identify test evolution patterns, we adopted a two-stage clustering approach grounded in empirical evaluation and inspired by best practices in time series clustering. A preliminary analysis was conducted using five algorithms: K-Shape, DTW K-Means, Hierarchical Clustering, SOM, and Time Series Forest (TSF). Among them, K-Shape Clustering [18] proved to generate the most cohesive and structurally meaningful groups, justifying its use in the initial refinement stage.

In the first stage, we applied K-Shape Clustering with $k = 5$ to the entire dataset and computed the Shape-Based Distance (SBD) [18] between each project's time series and its cluster centroid. Repositories falling within the fourth quartile of the SBD distribution—those with the greatest dissimilarity—were removed as outliers. This filtering step enhanced the overall consistency of the dataset for the final analysis.

The second stage involved reapplying all five clustering algorithms to the refined dataset. This multi-algorithm strategy followed recommendations from the time series clustering literature, emphasizing the importance of exploring different modeling perspectives to reduce methodological bias. For algorithms requiring a predefined number of clusters, we employed the β_{CV} heuristic [14], which selects the smallest k after the stabilization of the intra- and inter-cluster variance ratio.

Each algorithm was executed ten times to account for randomness, and the median SBD of each clustering result was used as a quality metric. After comparing outcomes, the algorithm yielding the lowest median SBD was chosen for final analysis. This ensured that the selected model produced the most cohesive and interpretable test evolution clusters across the dataset.

3.4 Co-evolution level categorization

After standardizing test evolution time series and identifying evolution patterns, the next step is to quantify and categorize the level of test co-evolution within each project. Using the time series data generated in Subsection 3.2, we calculate the proportion of test code across equal segments of each project's development history.

In this work, co-evolution is defined as the extent to which test and production code evolve in a temporally synchronized manner, meaning that changes occur within 30-day time intervals, a granularity commonly used in the literature. We operationalize this concept by analyzing variations in Lines of Code (LOC) for both test and production code over time.

To measure the degree of co-evolution, we compute the Pearson Correlation Coefficient for each project. This measures the strength and direction of the relationship between test LOC and production LOC, indicating how closely the evolution of test code mirrors that of production code. It is important to emphasize that correlation does not imply causation. While correlation can reveal patterns of synchronized evolution, it does not prove that changes in production code cause changes in test code, or vice versa. This limitation should be taken into account when interpreting the results.

Based on the distribution of Pearson coefficients across all projects, we classify them into three co-evolution levels: *High Co-evolution* (top quartile), indicating strong synchronization; *Moderate Co-evolution* (middle quartiles), indicating a less strict but still present relationship; and *Low Co-evolution* (bottom quartile), indicating a weak relationship between test and production code evolution.

This categorization allows us to analyze the characteristics of each group and investigate how different levels of test co-evolution correlate with project attributes and development practices. It also facilitates the comparison of co-evolution behaviors across the dataset, providing insights into the impact of testing alignment on software evolution.

3.5 Project and community metrics extraction

The tool extracts four key metrics to assess both project activity and community engagement: number of commits (development effort), contributors (team size), forks (popularity), and open issues (community interaction). These indicators provide a comprehensive view of a project's internal dynamics and its external visibility.

3.6 Maintenance activities extraction

Understanding the correlation between test co-evolution and maintenance activities offers valuable insights into software quality and development practices. The intersection of these dimensions can provide rich data for assessing the health of a software project and the quality of the evolving system. While co-evolution can indicate synchronized testing practices, analyzing the nature of maintenance—particularly corrective actions—helps contextualize quality issues within the codebase. Projects with a high volume of corrective maintenance often reflect lower code quality, as such changes are typically driven by defect discovery or user-reported issues [9, 19].

To categorize maintenance activity types, we employed an automated classification technique developed by Amit et al. [1], which uses commit message analysis to distinguish between corrective, adaptive, and perfective changes. This method parses all commit messages and applies pattern-matching rules and regular expressions to detect keywords and phrases that reflect the nature of each maintenance activity.

The classification process enables the identification of maintenance profiles across projects, which are then compared to their corresponding test co-evolution levels. In particular, we focus on the prevalence of corrective maintenance, aiming to explore whether projects with higher co-evolution tend to exhibit fewer bug-fix-related commits, suggesting better preventive testing practices.

By combining co-evolution metrics with maintenance activity classification, this analysis contributes to a deeper understanding of how testing alignment may influence software stability. It also supports the broader hypothesis that higher test co-evolution can serve as a potential indicator of reduced corrective maintenance effort and, by extension, improved code quality.

3.7 System design

The *Highlight Test Code* is an integrated software solution developed to support the analysis of test and production code co-evolution in software repositories. The system is architected as a web-based platform, composed of two main components: a frontend interface and a backend infrastructure. This separation of concerns ensures modularity, scalability, and a smooth user experience for interacting with repository data and insights.

The frontend is implemented using TypeScript and React, offering a modern and interactive interface for end users. Its main functions include user authentication, repository navigation, data visualization, and detailed inspection of repository pipelines. Visual analytics, such as synchronized evolution charts and commit timelines, provide users with actionable insights into how their test code evolves alongside production code. The system requires users to authenticate to access features, enabling personalized views and protecting sensitive data through session-based access.

The backend is built using Python, leveraging the FastAPI framework for high-performance web service creation and PostgreSQL as the database engine. This architecture is responsible for orchestrating data collection and processing tasks, including cloning repositories, extracting commit metadata, and generating structured time series data. By integrating these components, the system ensures that all metrics and historical data are persistently stored, efficiently queried, and reliably managed.

On the analytical side, the backend performs several key operations. It computes co-evolution metrics such as Pearson correlation between test and production code time series, and derives maintenance activity indicators like corrective, adaptive, and perfective changes. These insights are made available to the frontend through API endpoints, allowing users to explore test co-evolution patterns in both personal and community-shared repositories. The system's architecture lays the groundwork for extensibility, supporting future integration of new metrics, repository sources, and visualization capabilities.

4 Usage scenario

This section illustrates a typical usage scenario of the *Highlight Test Code*, highlighting its functionalities through the user interface steps. The scenario begins with a user accessing the tool to analyze the test co-evolution behavior of one or more software repositories (see Figure 1).

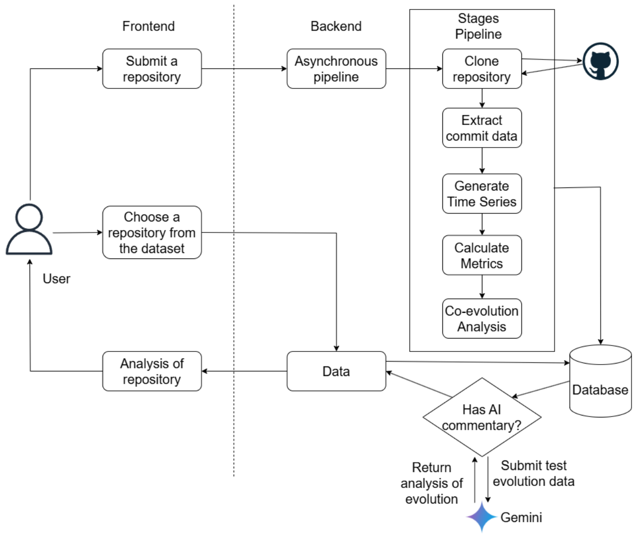
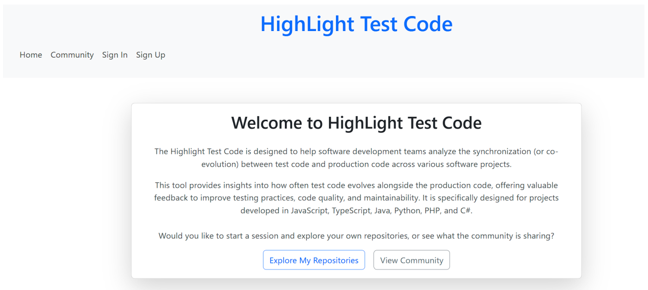


Figure 1: *Highlight Test Code* operation diagram

Upon entering the platform (Figure 2), users are greeted with a brief overview of the tool's purpose and features. From the homepage, they can choose to explore their own repositories or view results shared by the community. Selecting the *View Community* option redirects the user to a list of public repositories (Figure 3), where each entry displays the repository URL, default branch, registration date, and allows users to inspect details through the *View Details* button.



Welcome to HighLight Test Code

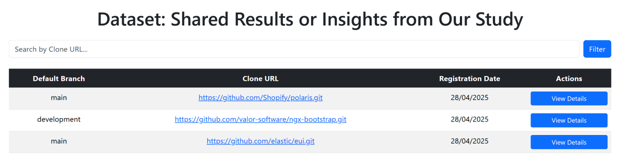
The Highlight Test Code is designed to help software development teams analyze the synchronization (or co-evolution) between test code and production code across various software projects.

This tool provides insights into how often test code evolves alongside the production code, offering valuable feedback to improve testing practices, code quality, and maintainability. It is specifically designed for projects developed in JavaScript, TypeScript, Java, Python, PHP, and C#.

Would you like to start a session and explore your own repositories, or see what the community is sharing?

Figure 2: Welcome page of Highlight Test Code

Upon repository selection, the system checks for an active user session, prompting for login if necessary. Users can sign in, authenticate via GitHub, or create an account. The resulting dashboard (Figures 4 and 5) displays repository statistics (commits, contributors, forks, open issues), co-evolution analysis (correlation coefficient, co-evolution level), code growth, and test ratio evolution over



Dataset: Shared Results or Insights from Our Study

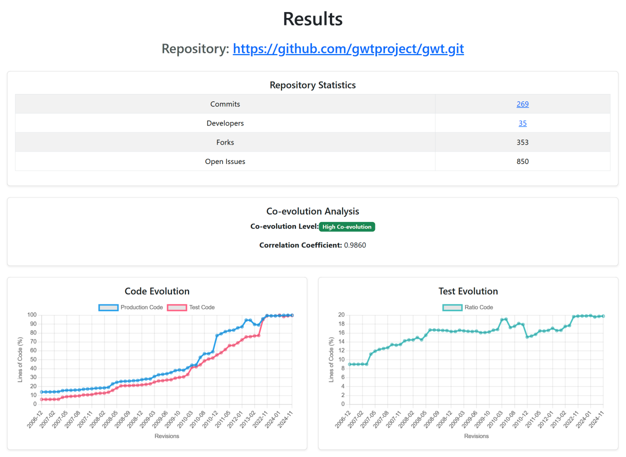
Search by Clone URL... Filter

Default Branch	Clone URL	Registration Date	Actions
main	https://github.com/Shopify/polaris.git	28/04/2025	View Details
development	https://github.com/valor-software/ngx-bootstrap.git	28/04/2025	View Details
main	https://github.com/elastic/eui.git	28/04/2025	View Details

Figure 3: Community page showing shared results from public repositories

time. It also includes a bar chart comparing test and production code and a bar chart showing maintenance activity types.

In the lower section, the tool presents the detected test evolution pattern (e.g., Growing Tests) and provides AI-generated commentary (using Gemini) based on the project's time series, cluster characteristics, and centroid behavior. This commentary analyzes similarities and deviations between the repository and its assigned cluster, offering insights into the repository's testing practices.



Results

Repository: https://github.com/gwtproject/gwt.git

Repository Statistics

Commits	269
Developers	35
Forks	353
Open Issues	850

Co-evolution Analysis

Co-evolution Level: High Co-evolution

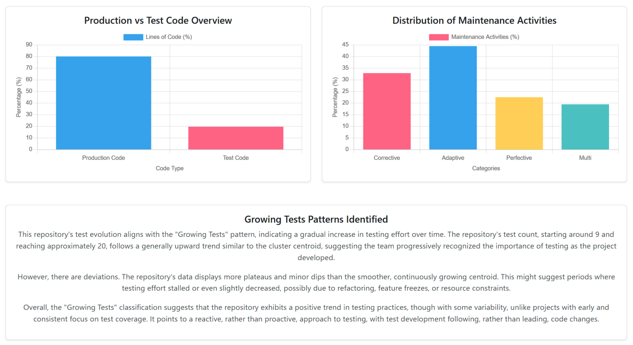
Correlation Coefficient: 0.9860

Figure 4: Top section of the results page for the selected repository (e.g., GWT), with repository statistics and co-evolution charts

For more detailed analysis, users can explore co-evolution detection per revision. The tool showcases scatter and table(6) views for navigating co-evolved file pairs. The table provides mappings between production and test files, the co-evolution detection result (Yes/No), the revision date, enabling traceability of test-production relationships[1]. Currently, this file-level visualization is limited to Java projects. This limitation stems from the specific parsing and matching logic required to detect links between production and test files, which currently relies on language-specific conventions and test framework patterns. Support for additional languages would require tailored adaptations of these detection heuristics.

The platform also supports the analysis of new repositories. Users may register repositories manually by filling out the form or by selecting one from their authenticated GitHub account. Only

[1]Note: Detailed file-level information in the dataset analysis may not be available at the time of access, as it may still be processing.



Growing Tests Patterns Identified

This repository's test evolution aligns with the "Growing Tests" pattern, indicating a gradual increase in testing effort over time. The repository's test count, starting around 9 and reaching approximately 20, follows a generally upward trend similar to the cluster centroid, suggesting the team progressively recognized the importance of testing as the project developed.

However, there are deviations. The repository's data displays more plateaus and minor dips than the smoother, continuously growing centroid. This might suggest periods where testing effort stalled or even slightly decreased, possibly due to refactoring, feature freezes, or resource constraints.

Overall, the "Growing Tests" classification suggests that the repository exhibits a positive trend in testing practices, though with some variability, unlike projects with early and consistent focus on test coverage. It points to a reactive, rather than proactive, approach to testing, with test development following, rather than leading, code changes.

Figure 5: Bottom section of the results page with test evolution summary, co-evolution level, correlation coefficient, and maintenance distribution

public GitHub repositories are currently supported. Once a repository is added, an asynchronous pipeline is triggered, consisting of stages such as cloning, commit extraction, time series generation, metric calculation, and co-evolution analysis (Figure 7). Depending on the size of the repository, these stages may take some time to complete. The system provides real-time status updates for each stage, enabling users to monitor progress. This comprehensive and interactive process allows for both individual and community-level reflection on test strategies, fostering improved testing practices and long-term maintainability.

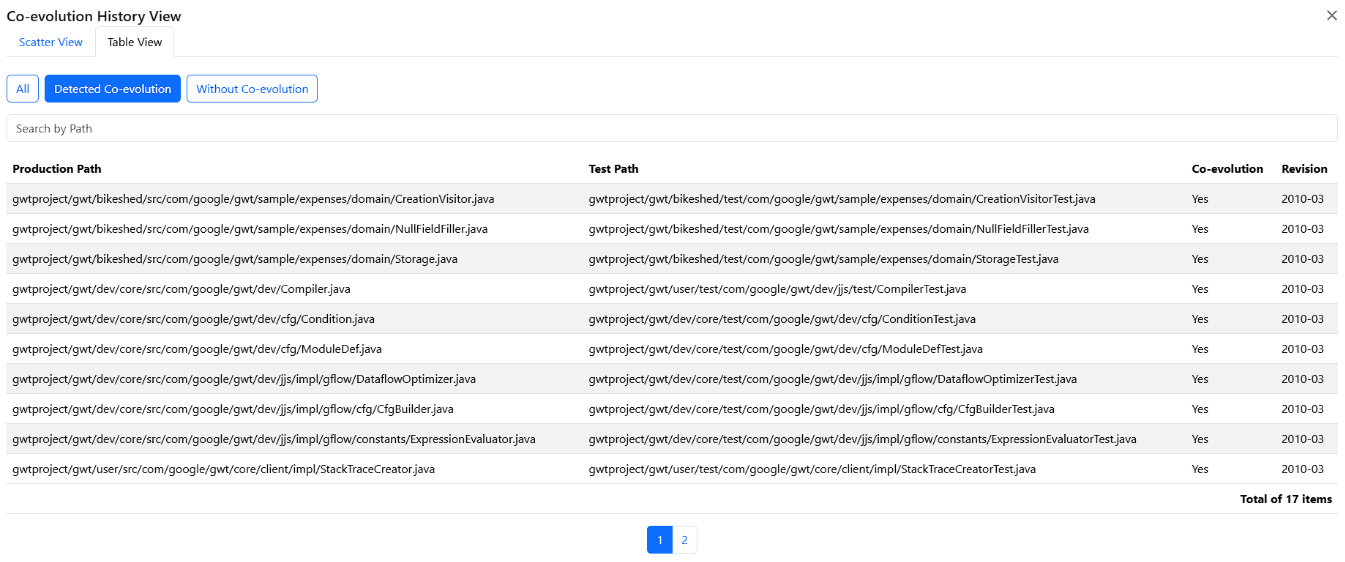
5 Limitations

Highlight Test Code provides a comprehensive environment for analyzing test and production code co-evolution, but the current version has limitations that could affect its applicability and generalizability.

First, the tool currently supports only public repositories hosted on GitHub. This restriction excludes private projects and repositories from other platforms (e.g., GitLab, Bitbucket), limiting the analysis scope to publicly available data. Expanding compatibility to support private and cross-platform repositories would enhance adoption in industry and private research contexts.

Second, the visualization of co-evolution at the file level is, at this stage, implemented only for Java projects. This limitation arises from the reliance on filename and path conventions to infer links between production and test files—a common but risky heuristic. While such naming conventions are frequently adopted in Java projects, they are not universally consistent and may lead to incorrect associations, especially in projects with non-standard structures or weak adherence to testing conventions.

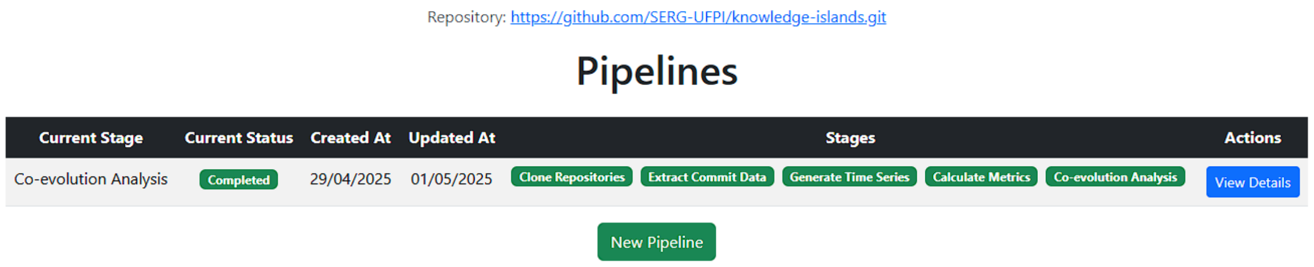
Finally, the data extraction and analysis pipeline operates asynchronously and may take a considerable amount of time depending on repository size and complexity. Although progress tracking is available to users, large repositories with extensive histories can lead to longer processing times. Optimization of pipeline stages and the implementation of processing parallelism are areas identified for future improvement.


Co-evolution History View

Production Path	Test Path	Co-evolution	Revision
gwtproject/gwt/bikeshed/src/com/google/gwt/sample/expenses/domain/CreationVisitor.java	gwtproject/gwt/bikeshed/test/com/google/gwt/sample/expenses/domain/CreationVisitorTest.java	Yes	2010-03
gwtproject/gwt/bikeshed/src/com/google/gwt/sample/expenses/domain/NullFieldFiller.java	gwtproject/gwt/bikeshed/test/com/google/gwt/sample/expenses/domain/NullFieldFillerTest.java	Yes	2010-03
gwtproject/gwt/bikeshed/src/com/google/gwt/sample/expenses/domain/Storage.java	gwtproject/gwt/bikeshed/test/com/google/gwt/sample/expenses/domain/StorageTest.java	Yes	2010-03
gwtproject/gwt/dev/core/src/com/google/gwt/dev/Compiler.java	gwtproject/gwt/user/test/com/google/gwt/dev/jjs/test/CompilerTest.java	Yes	2010-03
gwtproject/gwt/dev/core/src/com/google/gwt/dev/cfg/Condition.java	gwtproject/gwt/dev/core/test/com/google/gwt/dev/cfg/ConditionTest.java	Yes	2010-03
gwtproject/gwt/dev/core/src/com/google/gwt/dev/cfg/ModuleDef.java	gwtproject/gwt/dev/core/test/com/google/gwt/dev/cfg/ModuleDefTest.java	Yes	2010-03
gwtproject/gwt/dev/core/src/com/google/gwt/dev/jjs/impl/gflow/DataflowOptimizer.java	gwtproject/gwt/dev/core/test/com/google/gwt/dev/jjs/impl/gflow/DataflowOptimizerTest.java	Yes	2010-03
gwtproject/gwt/dev/core/src/com/google/gwt/dev/jjs/impl/gflow/cfg/CfgBuilder.java	gwtproject/gwt/dev/core/test/com/google/gwt/dev/jjs/impl/gflow/cfg/CfgBuilderTest.java	Yes	2010-03
gwtproject/gwt/dev/core/src/com/google/gwt/dev/jjs/impl/gflow/constants/ExpressionEvaluator.java	gwtproject/gwt/dev/core/test/com/google/gwt/dev/jjs/impl/gflow/constants/ExpressionEvaluatorTest.java	Yes	2010-03
gwtproject/gwt/user/src/com/google/gwt/core/client/impl/StackTraceCreator.java	gwtproject/gwt/user/test/com/google/gwt/core/client/impl/StackTraceCreatorTest.java	Yes	2010-03

Total of 17 items

Figure 6: Table view displaying file-level co-evolution details for a specific revision


Repository: https://github.com/SERG-UFPI/knowledge-islands.git

Pipelines

Current Stage	Current Status	Created At	Updated At	Stages	Actions
Co-evolution Analysis	Completed	29/04/2025	01/05/2025	Clone Repositories; Extract Commit Data; Generate Time Series; Calculate Metrics; Co-evolution Analysis	View Details

Figure 7: Pipeline progress view displaying current stage, status, and execution dates for repository processing

6 Conclusion and future work

This paper presented *Highlight Test Code*, a tool designed to support the visualization and analysis of co-evolution between test and production code in open-source software repositories. The *Highlight Test Code* provides an interactive web interface and a multi-stage analysis pipeline (including commit extraction, time series generation, clustering, and co-evolution metric computation), enabling researchers to identify distinct test evolution patterns and explore the correlation between co-evolution and project characteristics. The platform is backed by a large-scale dataset of 526 repositories across six programming languages, allowing for in-depth investigation of test evolution and its relationship with project attributes.

For future work, several directions can be explored to enhance the platform's applicability and impact. These include: expanding file-level co-evolution analysis to support more programming languages; optimizing the processing pipeline for performance, especially with large repositories, and establishing performance benchmarks; enabling analysis of private and cross-platform repositories to facilitate enterprise adoption; distinguishing between structural and semantic changes in test code to improve co-evolution analysis precision; empirically evaluating the tool's usefulness and usability with real users; and gathering feedback from practitioners and researchers to refine features and improve usability.

ARTIFACT AVAILABILITY

The authors declare that the research artifacts supporting the findings of this study are available at https://zenodo.org/records/16756417. A demonstration video of the platform is also publicly accessible[2] [3].

REFERENCES

[1] Idan Amit and Dror G. Feitelson. 2020. The Corrective Commit Probability Code Quality Metric. arXiv:2007.10912 [cs.SE]

[2] Hudson Silva Borges. 2018. Characterizing and predicting the popularity of github projects. (2018).

[3] Otávio Cury, Guilherme Avelino, Pedro Santos Neto, Marco Túlio Valente, and Ricardo Britto. 2024. Source code expert identification: Models and application. *Information and Software Technology* (2024), 107445.

[4] Otávio Cury, Guilherme Avelino, Pedro Santos Neto, Ricardo Britto, and Marco Túlio Valente. 2022. Identifying source code file experts. In *Proceedings of the*

[2]https://doi.org/10.5281/zenodo.15492742
[3]https://youtu.be/U29eEg_gXXM

16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. 125–136.

[5] Marcio Delamaro, Mario Jino, and Jose Maldonado. 2013. *Introdução ao teste de software.* Elsevier Brasil.

[6] Barrett Ens, Daniel Rea, Roiy Shpaner, Hadi Hemmati, James E Young, and Pourang Irani. 2014. Chronotwigger: A visual analytics tool for understanding source and test co-evolution. In *2014 Second IEEE Working Conference on Software Visualization.* IEEE, 117–126.

[7] Danielle Gonzalez, Joanna CS Santos, Andrew Popovich, Mehdi Mirakhorli, and Mei Nagappan. 2017. A large-scale study on the usage of testing patterns that address maintainability attributes: patterns for ease of modification, diagnoses, and comprehension. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR).* IEEE, 391–401.

[8] Xing Hu, Zhuang Liu, Xin Xia, Zhongxin Liu, Tongtong Xu, and Xiaohu Yang. 2023. Identify and Update Test Cases When Production Code Changes: A Transformer-Based Approach. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 1111–1122.

[9] Sunghun Kim, Thomas Zimmermann, E James Whitehead Jr, and Andreas Zeller. 2007. Predicting faults from cached history. In *29th International Conference on Software Engineering (ICSE'07).* IEEE, 489–498.

[10] MM Lehman and FN Parr. 1976. Program evolution and its impact on software engineering. In *Proceedings of the 2nd international conference on Software engineering.* 350–357.

[11] Meir M Lehman. 1979. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software* 1 (1979), 213–221.

[12] Stanislav Levin and Amiram Yehudai. 2017. The co-evolution of test maintenance and code maintenance through the lens of fine-grained semantic changes. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 35–46.

[13] Cosmin Marsavina, Daniele Romano, and Andy Zaidman. 2014. Studying fine-grained co-evolution patterns of production and test code. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation.* IEEE, 195–204.

[14] Daniel A Menasce and Virgilio Almeida. 2001. *Capacity Planning for Web Services: metrics, models, and methods.* Prentice Hall PTR.

[15] Tom Mens, Michel Wermelinger, Stéphane Ducasse, Serge Demeyer, Robert Hirschfeld, and Mehdi Jazayeri. 2005. Challenges in software evolution. In *Eighth International Workshop on Principles of Software Evolution (IWPSE'05).* IEEE, 13–22.

[16] Charles Miranda, Guilherme Avelino, and Pedro Santos Neto. 2025. Test Co-Evolution in Software Projects: A Large-Scale Empirical Study. *Journal of Software: Evolution and Process* 37, 7 (2025), e70035.

[17] Glenford J Myers, Tom Badgett, Todd M Thomas, and Corey Sandler. 2004. *The art of software testing.* Vol. 2. Wiley Online Library.

[18] John Paparrizos and Luis Gravano. 2015. k-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data.* 1855–1870.

[19] Foyzur Rahman, Daryl Posnett, Abram Hindle, Earl Barr, and Premkumar Devanbu. 2011. BugCache for inspections: hit or miss?. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering.* 322–331.

[20] Joanderson Gonçalves Santos and Rita Suzana Pitangueira Maciel. 2024. AutomTest 3.0: An automated test-case generation tool from User Story processing powered with LLMs. In *Simpósio Brasileiro de Engenharia de Software (SBES).* SBC, 769–774.

[21] Samiha Shimmi and Mona Rahimi. 2022. Leveraging code-test co-evolution patterns for automated test case recommendation. In *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test.* 65–76.

[22] Samiha Shimmi and Mona Rahimi. 2022. Patterns of Code-to-Test Co-evolution for Automated Test Suite Maintenance. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST).* IEEE, 116–127.

[23] Ian Sommerville. 2011. Software engineering 9th Edition. *ISBN-10* 137035152 (2011), 18.

[24] Weifeng Sun, Meng Yan, Zhongxin Liu, Xin Xia, Yan Lei, and David Lo. 2023. Revisiting the Identification of the Co-evolution of Production and Test Code. *ACM Transactions on Software Engineering and Methodology* 32, 6 (2023), 1–37.

[25] Tiago Samuel Rodrigues Teixeira, Fábio Fagundes Silveira, and Eduardo Martins Guerra. 2024. METEOR: A Tool for Monitoring Behavior Preservation in Test Code Refactorings. In *Simpósio Brasileiro de Engenharia de Software (SBES).* SBC, 755–761.

[26] László Vidács and Martin Pinzger. 2018. Co-evolution analysis of production and test code by learning association rules of changes. In *2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE).* IEEE, 31–36.

[27] Ali Görkem Yalçın and Tugkan Tuglular. [n. d.]. Visualization of Source Code and Acceptance Test Co-evolution. ([n. d.]).

[28] Andy Zaidman, Bart Van Rompaey, Arie van Deursen, and Serge Demeyer. 2011. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empirical Software Engineering* 16, 3 (2011), 325–364.