Gowri Prashanth Kanagaraj

Code Blueprint: Visualizing the Structure of Classes

Abstract - The project aims to develop an interactive tool designed to visually represent the intricate internal structure of classes within object-oriented programming paradigms. Drawing inspiration from the pioneering concept of Class Blueprints [1], elucidated in research contributions by Michele Lanza and Stephane Ducasse [1] [2], the project aims to provide a novel approach to understanding class compositions, encompassing methods, attributes, dependencies, and other critical elements. Utilizing the robust capabilities of D3.js, a versatile JavaScript library renowned for crafting bespoke data visualizations, the system embarks on transforming abstract code constructs into intuitive graphical representations. Leveraging the Abstract Syntax Tree method, the project extracts pertinent datasets from existing codebases, facilitating the creation of comprehensive class blueprints that afford developers deeper insights into software architectures and facilitate more informed decision-making processes.

1 Introduction

In programming world, knowing a class well can help us understand the whole system better. This is important for people who develop and maintain the system, so they can make changes without causing any disruptions within the modules. The Code Blueprint project facilitates this process by depicting the visual representation of the class's structure. This will allow users to identify patterns and anomalies within the class structures more easily. In this paper, the architecture, functionality, and utilization of Code Blueprint are delved into, showcasing how developers are empowered to navigate the complexities of software systems effortlessly and extract valuable insights with confidence. Through a detailed exploration of its features and capabilities, the transformative potential of Code Blueprint in revolutionizing software visualization and analysis is highlighted.

2 Code Blueprint In A Nutshell

In a nutshell, Code Blueprint is a web-based application that leverages Circular Packing Charts (https://observablehq.com/@d3/zoomable-circle-packing) as shown in Figure 1, to offer an enriched understanding of object-oriented codebases. It incorporates the evolutionary aspect of software development into its visualization [1].

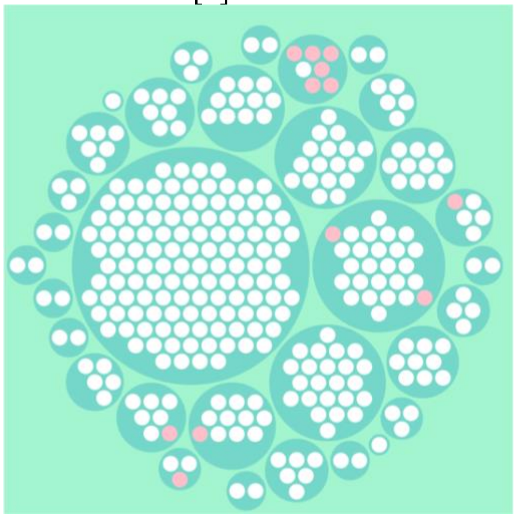


Figure 1: Circular Packing Chart of Code Blueprint

Code Blueprint dynamically captures the evolution of software by traversing through the commits of a Git repository. As show in Figure 2, each circle within the visualization encapsulates the intricacies of classes, methods, and attributes, with classes depicted in a distinct hue of HSL (332°, 80%, 85%), methods in white, and attributes in #FFC0CB.

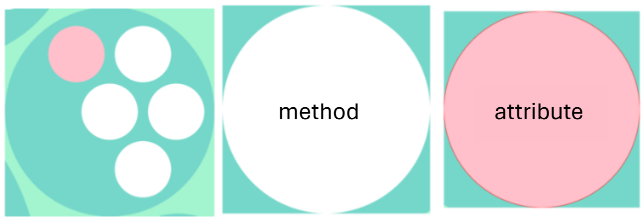
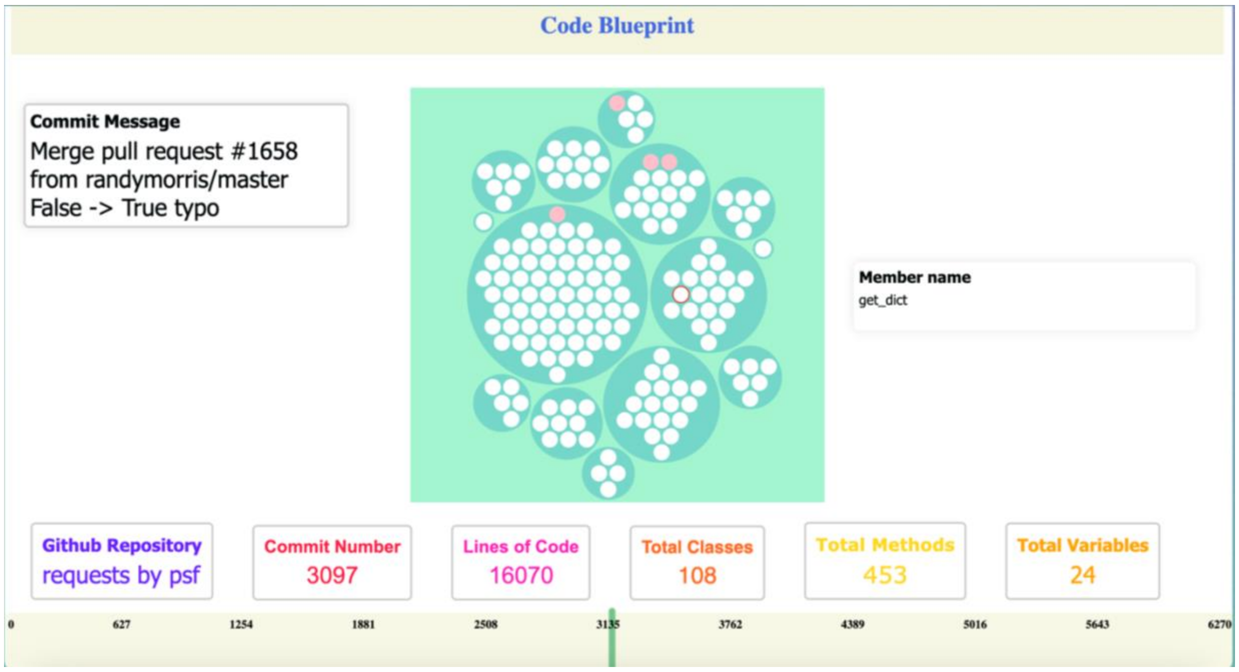


Figure 2: Circular Packing Chart Circles

This adaptive approach enables developers to track the progression of the software system and identify patterns or trends in its development lifecycle. It offers a powerful toolkit for them to gain insights into Python codebases, facilitating code comprehension, maintenance, and decision-making processes.

2.1 User Interface

Figure 3 showcases the primary user interface of Code Blueprint, designed to provide an immersive and intuitive experience for exploring object-oriented Python codebases. At the heart of the interface lies the Circular Packing chart, where classes, methods, and attributes are visualized as circles, each representing a distinct entity within the codebase. Users can navigate through different commits using a slider input located prominently within the interface, facilitating dynamic updates to the visualization and dashboard components based on the selected commit.


Code Blueprint

Commit Message
Merge pull request #1658 from randymorris/master
False -> True typo

Member name
get_dict

Github Repository: requests by psf
Commit Number: 3097
Lines of Code: 16070
Total Classes: 108
Total Methods: 453
Total Variables: 24

Figure 3: Code Blueprint UI

Code Blueprint offers a zoom view for class members within the Circular Packing chart, allowing users to delve deeper into the internal structure of classes and gain a more granular understanding of their dependencies and relationships. This zoom functionality enhances the user experience, enabling closer inspection of individual entities and their interactions within the codebase.

Additionally, it features multiple dashboards, each serving a specific purpose in providing insights into different aspects of the software system. These dashboards offer a comprehensive overview of metrics such as lines of code, total classes, total methods, total variables, and more, empowering users to analyze and comprehend various dimensions of the codebase's complexity and evolution.

Furthermore, Code Blueprint displays the name of each member within the Circular Packing chart, ensuring clarity and context for users as they navigate through the visualization. This feature enables users to easily identify and understand the significance of each entity within the codebase, facilitating informed decision-making and analysis.

Overall, the website’s user interface is designed with a focus on usability, interactivity, and accessibility, providing users with powerful tools and visualizations to explore and understand object-oriented Python codebases effectively. Through intuitive controls, dynamic updates, and comprehensive dashboards, taking reference from the code metaphor [7], Code Blueprint empowers users to gain insights into the structure, evolution, and complexity of software systems with ease.

2.2 Data Overview and Model Representation

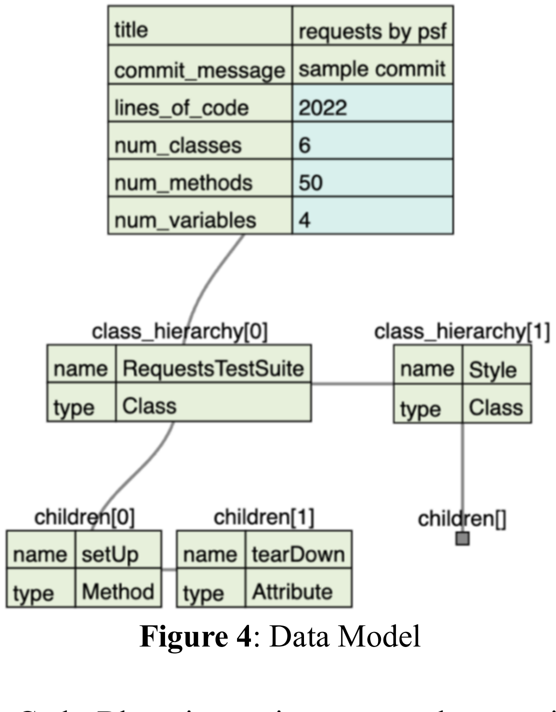


Figure 4: Data Model

The data provided for the Code Blueprint project encapsulates various metrics and attributes essential for comprehensively analyzing the software system. It includes key information such as the project title, commit message, lines of code, number of classes, methods, and variables as shown in Figure 4. Additionally, the data model incorporates a hierarchical structure representing the class hierarchy within the software system. Each class node contains information about its methods and attributes, allowing for a detailed examination of the internal structure of the software. This structured representation facilitates a deeper understanding of the project's evolution and internal composition, enabling stakeholders to make informed decisions regarding software maintenance and development strategies. Moreover, visualization techniques that use multiple stakeholder-specific views [8], can gain further insights into the architectural aspects of the software system, enhancing their ability to comprehend its complexities and make strategic decisions accordingly.

3 Architecture

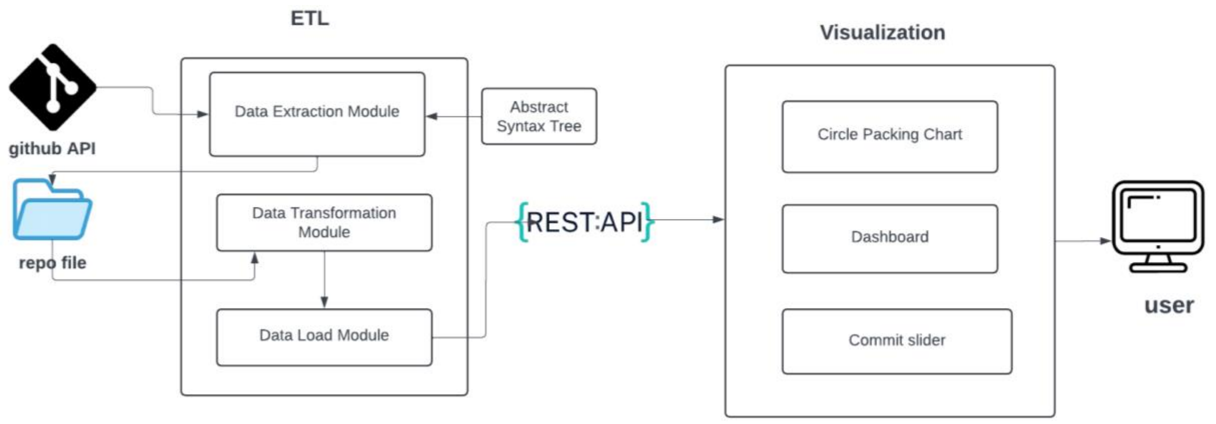


Figure 5: Project Architecture

Code Blueprint follows a modular architecture as shown in Figure 5 to facilitate the comprehensive analysis of software projects. The frontend, developed using HTML, CSS, and JavaScript, integrates D3.js for dynamic visualizations. On the backend, Flask, a Python web framework, is employed to handle data extraction and transformation tasks. The system leverages the GitHub API and Abstract Syntax Tree (AST) to retrieve codebase information, extracting essential details like lines of code, class counts, method quantities, and variable numbers. Subsequently, the Data Transformation Module processes the raw data into a structured format conducive to visualization, organizing class hierarchies, methods, and attributes into a hierarchical data model. The Visualization Module utilizes D3.js to generate circular packing charts, enabling users to interactively explore the internal structure of the software system. Complementing the visualizations, the Dashboard Module offers key metrics and insights, presenting summary statistics such as lines of code and class counts. Ultimately, the User Interface serves as the frontend, providing users with intuitive features like sliders, zoom functionality, and multiple dashboards for comprehensive analysis.

3.1 Abstract Syntax Tree

In the Code Blueprint project, Abstract Syntax Tree (AST) plays a pivotal role in understanding and representing the source code. AST provides a structured representation of the syntactic elements of the code, enabling deeper analysis and manipulation. Leveraging AST, Code Blueprint extracts essential code features such as class hierarchies, methods, and attributes, which are instrumental in creating a hierarchical data model for visualization. Through AST parsing, preprocessing, and encoding stages, the project transforms raw code into a structured format conducive to visualization, facilitating subsequent code-related tasks. AST-based code representation serves as a cornerstone for comprehensive analysis and exploration of software systems within the Code Blueprint framework, enhancing understanding and enabling informed decision-making.

AST parsing involves the conversion of raw code into a tree-like structure, where each node represents a syntactic element such as a class, function, or variable declaration. This process captures the hierarchical relationship between different elements of the code, providing a foundation for further analysis [5]. Subsequently, AST preprocessing involves refining the parsed AST to remove irrelevant information and standardize the representation, ensuring consistency and accuracy in subsequent stages [5].

3.2 Circle Packing Chart

Code Blueprint introduces a ground-breaking approach to visualizing software systems with Circular Packing charts, where each element—be it a class, method, or attribute—is distinguished by vibrant colors for easy identification. The incorporation of a zoom feature within the visualization enables users to delve deeper into class structures, facilitating a more nuanced understanding of their dependencies and relationships. By prominently displaying entity names, Code Blueprint ensures swift identification and analysis, empowering users to make informed decisions.

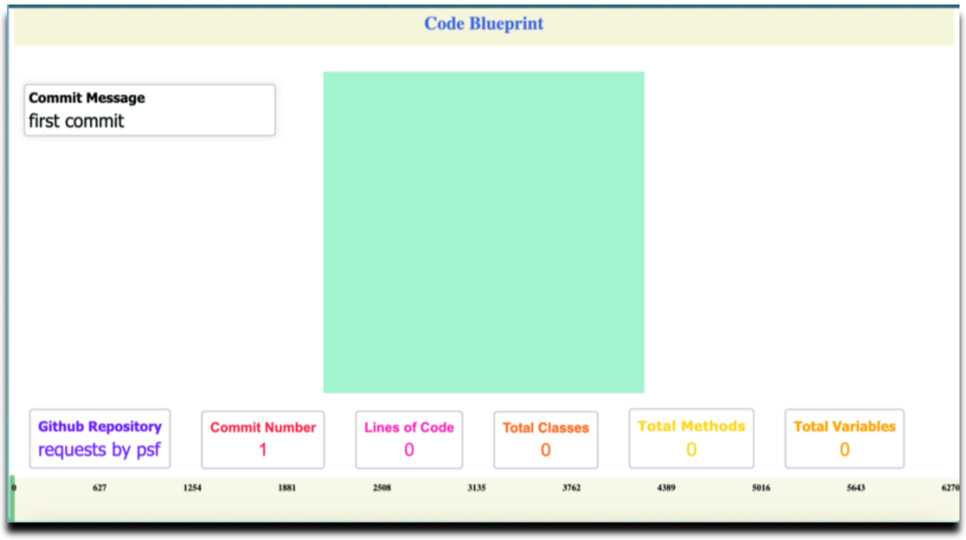
Moreover, the integration of circle packing and dot plot techniques offers a comprehensive view of hierarchical data structures, allowing for seamless pattern recognition and exploration of relationships [4]. Interactive features enhance the user experience by enabling exploration of cluster analysis results, shedding light on data distribution and outliers [4]. In sum, Code Blueprint provides an intuitive and insightful platform for navigating the complexities of software systems, empowering users to extract valuable insights effortlessly and make informed decisions with confidence.

4 Usage

To begin visualizing a software project with Code Blueprint, users initiate the process by selecting a specific commit number associated with the repository they are interested in. Utilizing the GitHub API, the system fetches the repository data, leveraging an Extract, Transform, Load (ETL) process to extract relevant information. This includes retrieving the internal structure of classes, which is accomplished through the use of Abstract Syntax Tree (AST) parsing techniques. The visualization process is then facilitated by the Visualization Module, which employs D3.js to represent classes based on their internal structure, providing users with an interactive visualization experience [3]. Additionally, the system incorporates reverse engineering methodologies to gain insights into the software architecture [3]. By implementing a dashboard layout, users can access key metrics and insights about the codebase, enhancing their understanding of its structure and evolution over time. Overall, the Visualization Module plays a central role in providing users with comprehensive insights into the software project's internal structure and evolution.

5 Request Python Repository

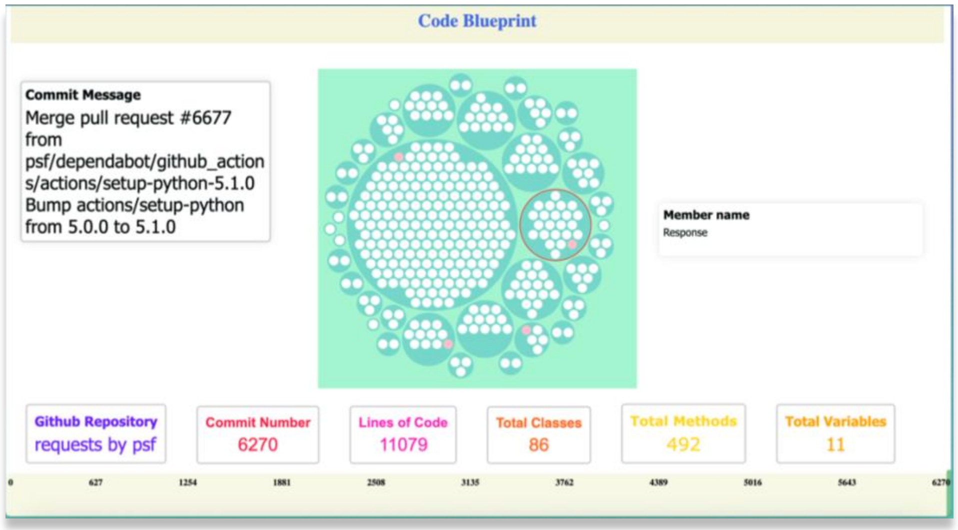
The Requests repository, a prominent Python library for sending HTTP/1.1 requests, boasts a substantial history of development and widespread usage. With a staggering 6270 commits contributed by a diverse community of 642 contributors, Requests has established itself as a cornerstone of modern Python programming. Since its inception on February 13, 2011 with first commit as show in Figure 6, Requests has undergone continuous evolution, with the latest commit made on April 1, 2024 as depicted in Figure 7. This library simplifies the process of making HTTP requests by eliminating the need for manual query string manipulation and data encoding, providing a convenient json method for handling data. Renowned for its reliability and ease of use, Requests has become one of the most downloaded Python packages, with approximately 30 million downloads per week and dependencies in over 1,000,000 repositories on GitHub. Developers can confidently rely on Requests for their HTTP request needs, making it a trusted and essential tool in the Python ecosystem. Github link : https://github.com/psf/requests


Code Blueprint

Commit Message
first commit

Github Repository: requests by psf
Commit Number: 1
Lines of Code: 0
Total Classes: 0
Total Methods: 0
Total Variables: 0

Figure 6: Initial Commit


Code Blueprint

Commit Message
Merge pull request #6677 from psf/dependabot/github_actions/actions/setup-python-5.1.0 Bump actions/setup-python from 5.0.0 to 5.1.0

Member name
Response

Github Repository: requests by psf
Commit Number: 6270
Lines of Code: 11079
Total Classes: 86
Total Methods: 492
Total Variables: 11

Figure 7: Final Commit

6 Related Work

Drawing inspiration from M3TRICITY [2], an interactive web application aimed at visualizing object-oriented software systems and their evolution, Code Blueprint endeavors to provide a similar platform for program comprehension and evolution analysis. Additionally, the concept of Class Blueprint serves as a foundational idea for Code Blueprint [1] [3], proposing a novel visualization method for classes that enhances the understanding of their internal structure. Class Blueprint focuses on representing class internals in terms of fields, their accesses, and method call flow, employing colors to convey additional information [6].

7 Future Works

In the future, the approach can be extended in the following ways:

- Implementing an approach based on a forest metaphor to enhance the comprehension of evolving object-oriented software systems [9]. This approach leverages familiar concepts such as forests of trees, trunks, branches, and leaves, with each software release represented as a forest for users to navigate and interact with, facilitating a deeper understanding of the software's evolution over time [9].
- Exploring the integration of various architecture-level information into a unified visualization technique, allowing stakeholders to quickly grasp the development, quality, and costs of a software system [10]. This approach aims to streamline the presentation of diverse architectural aspects within a single view, potentially reducing visual complexity and supporting the visualization of large-scale programs [10].
- conduct empirical evaluations to assess the effectiveness and usability of the implemented approaches, gathering user feedback to determine their utility in enhancing software architecture understanding and identifying areas for improvement.

8 Conclusion

In conclusion, a notable progress has been achieved in the visualization and comprehension of object-oriented software systems through the development of Code Blueprint. Inspired by existing works such as M3TRICITY and Class Blueprint, innovative techniques have been leveraged to create a web-based application that offers users a comprehensive understanding of software architecture evolution and internal structures. Through the integration of familiar metaphors like forests and cities, along with advanced visualization tools like Circular Packing charts and AST parsing, developers are empowered to navigate complex codebases with ease and extract valuable insights for informed decision-making. Moving forward, further enhancements in usability and effectiveness will be pursued, thereby advancing the field of software visualization and architectural understanding.

References

[1] S. Ducasse and M. Lanza, "The class blueprint: Visually supporting the understanding of classes," IEEE Transactions on Software Engineering, vol. 31, no. 1, pp. 75–90, Jan. 2005. doi:10.1109/tse.2005.14

[2] M. Lanza, S. Ardigo, C. Nagy, R. Minelli, "M3tricity: Visualizing Evolving Software & Data cities," 2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), May 2022. doi:10.1109/icse-companion55297.2022.9793749

[3] S. Ducasse and M. Lanza. A Categorization of Classes based on the Visualization of their Internal Structure: The Class Blueprint. In Proceedings of the 16th ACM SIGPLAN conference, volume 36, pages 300-311, Nov. 2001.

[4] E. Inman, R. West, and O. Wright, Kustomizing Your SAS® Viya Engine Using SAS® Studio Custom Tasks and D3.js, pp. 12–13, 2020. doi:https://support.sas.com/resources/papers/proceedings19/3434-2019.pdf [Accessed: 06-Apr-2024]

[5] H. Qian, W. Liu, Z. Ding, W. Sun, and C. Fang, "Abstract syntax tree for method name prediction: How far are we?," 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS), Oct. 2023. doi:10.1109/qrs60937.2023.00052

[6] N. J. Agouf, S. Ducasse, A. Etien, and M. Lanza, "A new generation of class blueprint," 2022 Working Conference on Software Visualization (VISSOFT), Oct. 2022. doi:10.1109/vissoft55257.2022.00012

[7] C. L. Jeffery, "The city metaphor in software visualization," Computer Science Research Notes, 2019. doi:10.24132/csrn.2019.2901.1.18

[8] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, and R. Vuduc, "Communicating software architecture using a unified single-view visualization," 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007), 2007. doi:10.1109/iceccs.2007.20

[9] U. Erra, G. Scanniello, and N. Capece, "Visualizing the evolution of software systems using the forest metaphor," 2012 16th International Conference on Information Visualisation, Jul. 2012. doi:10.1109/iv.2012.25

[10] N. J. Agouf, S. Ducasse, A. Etien, and M. Lanza, "A new generation of class blueprint," 2022 Working Conference on Software Visualization (VISSOFT), Oct. 2022. doi:10.1109/vissoft55257.2022.00012