MosaiCode: Visualizing the Use and Evolution of Different Programming Languages in the Linux Kernel

Michael A. Weyandt
Department of Computer Science
The University of Akron
Akron, Ohio, USA
maw120@uakron.edu

Michael L. Collard
Department of Computer Science
The University of Akron
Akron, Ohio, USA
collard@uakron.edu

Jonathan I. Maletic
Department of Computer Science
Kent State University
Kent, Ohio, USA
jmaletic@kent.edu

***Abstract*—MosaiCode, a software visualization tool that renders metrics from a code base as an interactive mosaic, is presented in a new release after its debut at VISSOFT'11. MosaiCode extends the Seesoft metaphor to support the visualization and understanding of various characteristics for large scale software systems as a tile mosaic. The tool is demonstrated by applying it to visualize a dynamic set of metrics generated from all the change data in the Linux kernel from release v6 to release v7. The evolution of the Rust language in the kernel is chosen as the example use case. MosaiCode supports multiple coordinated views of the software that allows users to traverse large amounts of data. Data generation and visualization are completely decoupled for maximum flexibility. Video URL: https://www.youtube.com/watch?v=nMR-_8CcBmY**

Keywords—software visualization, seesoft, software evolution, Rust

I. Introduction

Software developers often need to examine and explore a software system for evidence of code churn, recent changes, and long-term evolution. These are typically good indicators of a need to refactor a part of the code or longer term trends in the development of a system. However, these tasks are difficult to undertake because of the large amount of data (source code) that needs to be visualized and explored.

Here we use MosaiCode [1], a software visualization tool which supports multiple coordinated views of large software systems to understand change data in the Linux kernel. It leverages one of the most successful and well-known software visualization metaphors, namely that of SeeSoft [2], which was proposed by Eick in the early 90's. The metaphor is successful mainly due to its direct mapping from the visual metaphor back to the source code (or data). This leads to natural navigation of the representation. It makes the visualization easy to understand for a variety of stakeholders including developers, managers, and architects. Color and pixel maps are used to represent physical software concepts such as lines of code, functions, files, and subsystems. The metaphor is 2D and is fairly scalable.

We take advantage of this metaphor and incorporate other coordinated views to assist in navigation and understanding of the visualization. Additionally, drill down and abstraction mechanisms are utilized for this same end.

In the work presented here we examine the evolution and new use of the Rust programming language in the Linux kernel. Specifically, we examine the time frame of release v6 to release v7 of the kernel. The use of Rust in the Linux kernel is relatively new (2022) and deviates from the historical use of mainly C. The goal is to demonstrate that MosaiCode is able to present visualizations that can help developers better understand the use and evolution of Rust within the system.

II. MosaiCode

MosaiCode [1] consists of multiple coordinated views with controls to allow users to easily traverse large amounts of data. The design of MosaiCode is largely motivated by the Seesoft application [2]. They both address the same requirement: visualizing a large volume of source code data in a single view. This can be done using Seesoft by minimizing a line of code to either a line of pixels or a single pixel [4]. The idiom has also been extended to 3 dimensions to support even greater dimensionality [3] in the tool sv3D.

A screenshot of the tool is given in fig. 1. The views include the Mosaic, Container, and Summary views. The mosaic view displays the system as a whole with the primary SeeSoft [2] metaphor visualization. The entities are displayed as tiles and arranged into containers. The container view shows the hierarchical structure in the form of a file system tree layout and is based on a path attribute in the data set. The summary view a histogram showing the amount of each colored tile (characteristic) present in the current version of the selected container. This also indicates the range covered by the different colors. More detailed information about a selected attribute is given in a tooltip, with the full path and the current numeric value of this attribute.

In MosaiCode, the visualization and data generation are separate for maximum flexibility. Data for visualization is created separately by analysis tools and then converted into a XML or CSV format for input to the tool. The data formats are generic to allow for display of different types of data, not just that of source-code files. The XML format primarily consists of containers and entities, e.g., directories and files. Containers

form a list and are not arranged hierarchically in the input data for maximum flexibility of display in the tool. For each entity, a set of attributes is given with a value. The attributes may have numeric or textual values. Numeric-valued attributes are displayed visually and textual attributes are only used for context in the Entity View. Each container and entity has an attribute *name* used to form the hierarchical arrangement of the entities. The CSV format only requires a header line for the names of the attribute fields, with each row corresponding to an entity. Since the attribute *name* may not exist, the user is given the choice of which field to use for the name.

III. Architecture & Current Implementation

The original MosaiCode (circa 2007) was a native desktop application written in C++ and Qt. To provide more accessible and modern distribution, we convert it to a web application running in the browser. Thus making the application available for the work presented here. This new release reduces the overall memory footprint well below 4GB to enable the ability to run the tool in a browser.

In our example dataset of the Linux kernel, we track a total of 70,011 entities, with 12 metrics each, across 21 releases (fig. 2). In the original release, this amount of data took up roughly 2.5GB of memory. By repacking the underlying data model and reducing duplicate data, the new release brings that total footprint down to roughly 0.6GB of memory.

Memory footprint during data ingestion is also addressed. In the original release, data is sorted and stored in memory while a second pass built the in-memory model. This led to a high watermark of memory usage of over 2x the original data size. We enhanced this to use less data while storing the initially processed data, and to free no longer needed data during the creation of the in-memory model. The combination of these two reduces the memory usage during ingestion by ~4.5x.

Leveraging both the changes to in-memory structure and ingestion, MosaiCode now uses ~4x less memory in total. For our example data set, usage dropped from a little over 4GB to roughly 0.8GB. Since the memory usage is a factor of the number of entities * the number of attributes * the number of releases, these changes enable visualizations at a new scale within MosaiCode. Anywhere between 2 metrics across 100 releases of the Linux kernel, to 500 metrics across 2 releases of the Linux kernel, are now able to be explored within MosaiCode. In addition to these architectural changes, many smaller enhancements were made. A few highlights include, adding support for ingesting compressed files to support the larger datasets now possible, multiple calls are made asynchronous to improve the feel of the application, and a preconfigured one-click demo dataset.

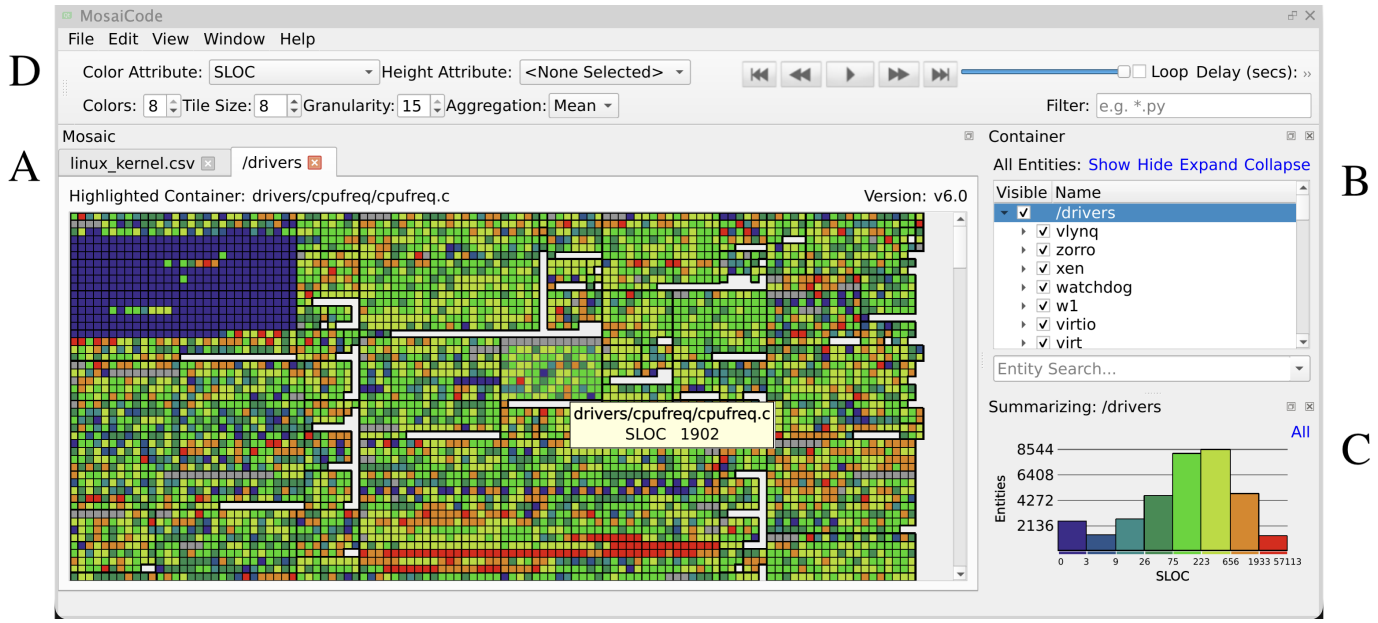

Fig. 1 The MosaiCode tool consists of the primary Mosaic visualization (A) with multiple tabs for additional views of the same data set, the Container View (B) which selects what entities are shown in the Mosaic, and a Summary View (C) showing a histogram of the data currently shown in the Mosaic. From left to right, the toolbar (D) allows for choice of displayed attributes, display characteristics, control for the display of multi-version data, and the ability to filter based on entity name.

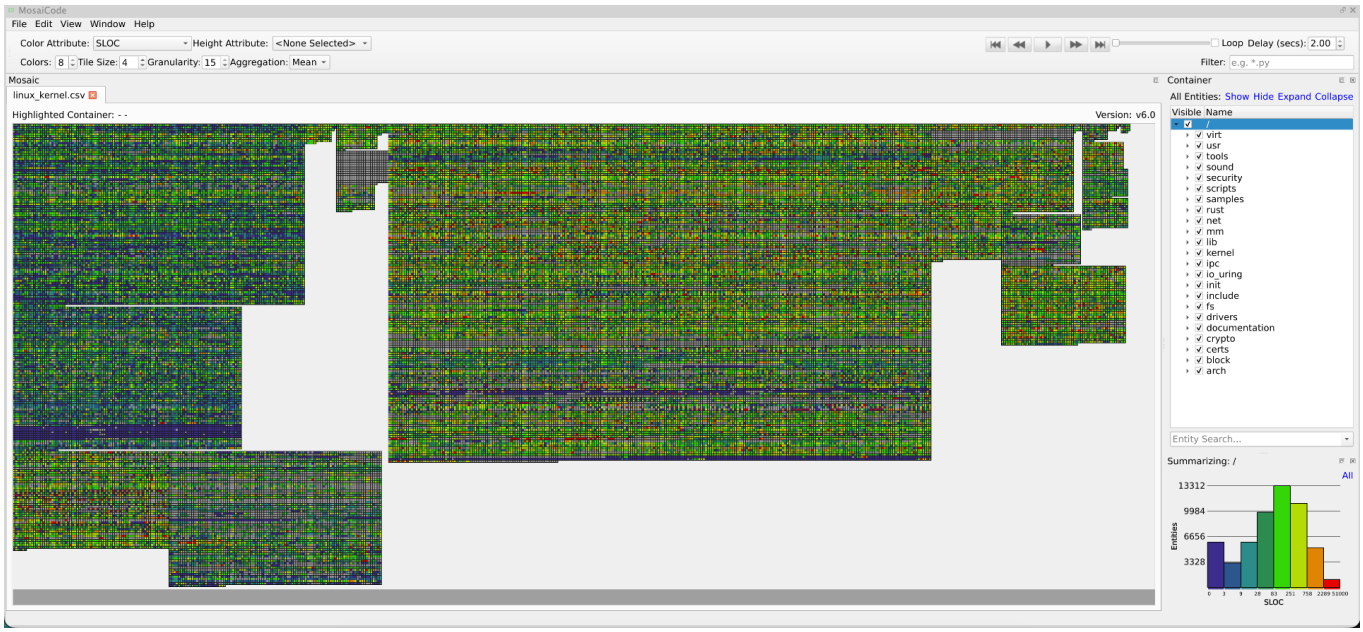

Fig. 2. The entire Linux kernel for all the metrics computed for the data set. While the mosaic view is not particularly insightful here, it demonstrates the scalability of the visualization. From this top-level view, one can navigate down into subsystems and identify interesting aspects of the system.

IV. Evolution of Rust in the Linux Kernel

We now examine the data collected from the releases of the Linux kernel. Rust initially landed in the Linux kernel in 2022 marked as experimental. Since that time, its usage within the kernel has grown, and Rust is now promoted as an officially supported language within the kernel as of December 2025 [5].

To better understand how Rust evolved within the kernel, we created a simple dataset of the Linux kernel containing core metrics across multiple releases. The dataset includes computable metrics such as number of lines (LOC), number of comments, source lines of code (SLOC), and line churn for each file. We first examine SLOC, to visualize the adoption of the Rust language across the numerous releases between v6.0 and v7.0 of the Linux kernel. At v6.0 the kernel's core languages are C and assembly, Rust is absent. At v6.1 we have the first Rust files, and see tiny, isolated islands of a few files in addition to an in-tree copy of Rusts standard alloc crate, adapted for the kernel. We drill down into the Rust directory to get a better view (fig. 3). All dark grey tiles in this view are non-Rust files, filtered via *.rs. All light grey tiles here represent Rust files that are not present in the currently selected version, either having been deleted in a prior version or not yet created.

Stepping forward through the releases to v6.10, we see the removal of the Rust in-tree alloc crate and the addition of some of its replacement under the rust/kernel/alloc directory (fig. 4). One of the most visually dramatic transitions lands at v6.19 where Rust roughly doubles from 36,000 lines of code to a little under 90,000 lines of code (fig. 5). This spike turns out to be a one-time inclusion of the in-tree Rust crates syn, quote, and procmacro2, which we can see better by including TotalChurn as the height attribute (fig. 6). By v7.0, the Linux kernel reaches 90,672 lines of code across 349 files.

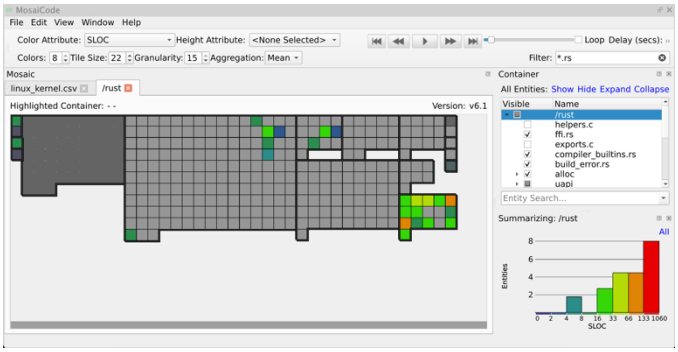

Fig. 3. Rust usage in the v6.1 kernel release, single files scattered throughout.

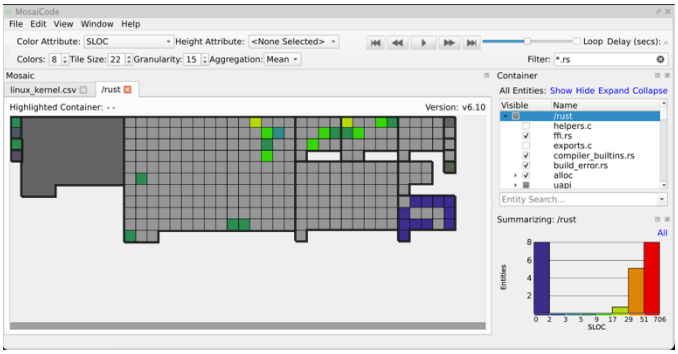

Fig. 4. Deletion of in-tree alloc crate and addition of some replacement files under rust/kernel/alloc, dark blue is the deletion.

V. Insights and Discussion

There are three findings that are visible in MosaiCode that get covered up in a raw table of percentages. First, the tool answers not just how much Rust exists but where Rust exists within the Linux kernel. Second, the tool allows easy identification of broad adoption versus small hot spots of Rust, the example here being the Rust crates copied into the Linux kernel, which can easily be mistaken for broader adoption if only looking at the file or line count. Third, animation over multiple releases of the Linux kernel visually highlights the rate and direction of adoption.

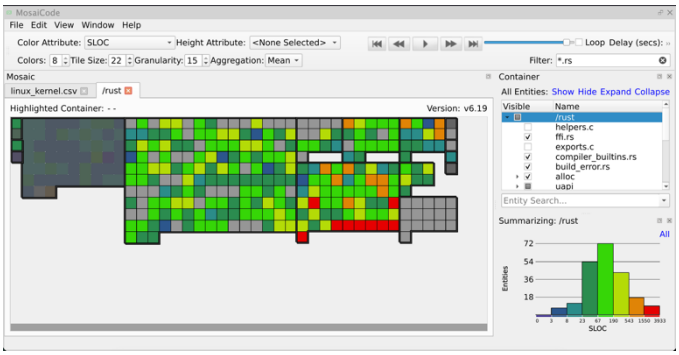

Fig. 5. Rust usage in the v6.19 kernel release, inclusion of in-tree Rust crates.

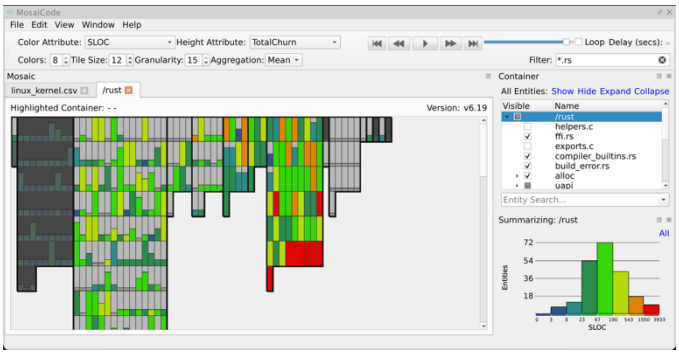

Fig. 6. Rust usage in v6.19 kernel release, drilled down into rust directory, with the height attribute set to TotalChurn.

VI. Conclusion

A new release of MosaiCode is presented and used to visualize the adoption of the Rust language across multiple Linux kernel versions. MosaiCode is demonstrated to be highly scalable and able to work with large software systems such as the Linux kernel. The visualizations are also shown to highlight the adoption of Rust over time in the kernel. We also see per file the additions, modifications, and removals of Rust code between releases. A key aspect is that the distribution of the code across files is easily seen. This can prove quite useful versus a raw percentage (or change) of Rust code per file or across the entire system.

We feel the fact that MosaiCode is not coupled to any analysis tool as an advantage. One can leverage any analysis tool to generate data which can then be easily viewed by the tool. This allows for experimenting with different types of metrics that reflect the features of interest.

The current new version of MosaiCode is available on GitHub at https://github.com/KSU-SDML/MosaiCode, with a demo available at:

https://ksu-sdml.github.io/MosaiCode/app/MosaiCode.html.

VII. References

[1] Maletic, J.I., Mosora, D.J., Newman, C.D., Collard, M.L., Sutton, A., Robinson, B.P., (2011), "MosaiCode: Visualizing Large Scale Software: A Tool Demonstration", in the Proceedings of the IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'11), Williamsburg, VA, USA, Sept 31 – Oct 1, pp. 96-101.

[2] S. G. Eick, J. L. Steffen, and E. E. J. Sumner, "Seesoft--A Tool for Visualizing Line Oriented Software Statistics," IEEE Transactions on Software Engineering, vol. 18, pp. 957-968, Nov 1992.

[3] A. Marcus, L. Feng, and J. I. Maletic, "3D Representations for Software Visualization," presented at the 2003 ACM Symposium on Software Visualization (SoftVis'03), San Diego, California, 2003.

[4] T. Ball and S. G. Eick, "Software Visualization in the Large," Computer, vol. 29, pp. 33-43, Apr 1996.

[5] https://lwn.net/Articles/1049831/