MosaiCode: Visualizing Large Scale Software

A Tool Demonstration

Jonathan I. Maletic[1], Daniel J. Mosora[1], Christian D. Newman[1],
Michael L. Collard[2], Andrew Sutton[3], Brian P. Robinson[4]

[1]Department of Computer Science
Kent State University
Kent, Ohio, USA
{jmaletic, dmosora, cnewman}@kent.edu

[2]Department of Computer Science
The University of Akron
Akron, Ohio, USA
collard@uakron.edu

[3]Dept. of Computer Science & Engineering
Texas A&M
College Station, Texas
asutton@cse.tamu.edu

[4]ABB Corporate Research
Raleigh, NC, USA
brian.p.robinson@us.abb.com

***Abstract*—A software visualization tool called MosaiCode is introduced and described. MosaiCode uses a Seesoft metaphor to support the visualization and understanding of various characteristics for large scale software systems. A usage scenario is given to demonstrate the tool.**

Keywords: software visualization, seesoft, maintenance

I. INTRODUCTION

Project managers and system architects often need to examine and explore a software system for evidence of code churn and hotspots. These are typically good indicators of a need to refactor a part of the code or, minimally, determine what is causing the behavior. Managers and architects also need to understand the impact of a proposed change or new feature in order to assess the risk and cost of such enhancements. Both of these tasks are difficult to undertake because of the large amount of data (source code) that needs to be visualized and explored.

Here we present a tool, MosaiCode, which supports multiple coordinated views of large software systems. It leverages one of the most successful and well-known software visualization metaphors, namely that of SeeSoft [1], 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 programmers, 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 this presentation, we introduce the tool and many of its features. A description of usage scenarios is given in the context of problems we have addressed at ABB Inc. (A large international engineering firm). The demonstration involves using software data from ABB projects to address the types of maintenance tasks mentioned previously.

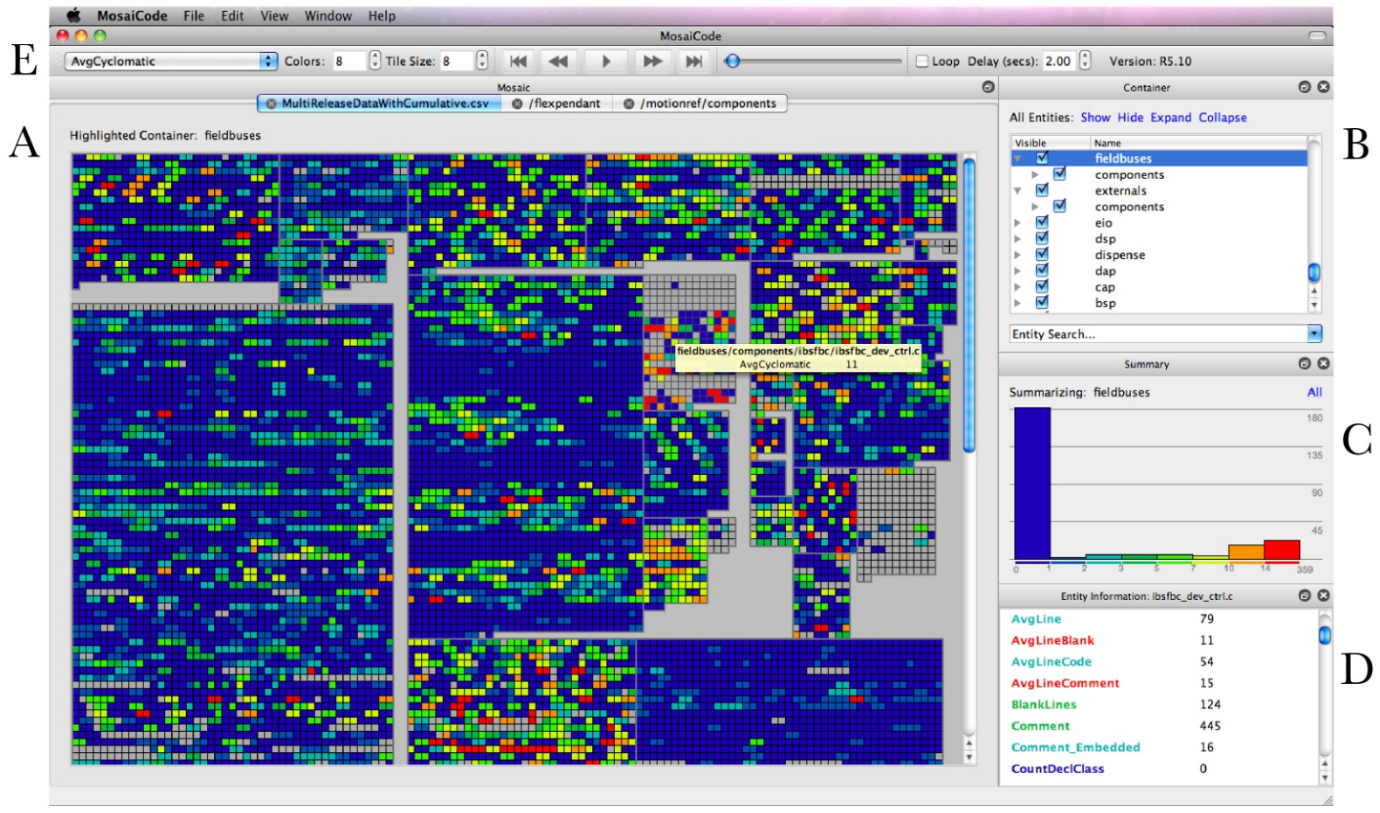
II. MOSAICODE

MosaiCode consists of multiple coordinated views with controls to allow users to easily traverse large amounts of data. A screenshot of the tool is given in Figure 1. The views include the Mosaic, Container, and Summary views. The mosaic view, Figure 1 (A), displays the system as a whole with the primary SeeSoft [1] metaphor visualization. The entities are displayed as tiles and arranged into containers. These tile boxes are arranged as rectangles with a golden ratio of approximately 0.62. The container view, Figure 1 (B), 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, Figure 1 (C), is 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, and in the Entity View, Figure 1 (D), which displays the rest of the attributes and their values.

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 is based on the data format for the sv3d tool [2] and 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, which 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 this attribute.

A. Architecture & Implementation

Visualization with MosaiCode follows a four-step process before control is passed to the user. This involves loading the XML or CSV input, parsing the data, analyzing and



AvgCyclomatic Colors: 8 Tile Size: 8 Delay (secs): 2.00 Version: R5.10

MultiReleaseDataWithCumulative.csv /flexpendant /motionref/components

Highlighted Container: fieldbuses

All Entities:

fieldbuses/components/ibsfbc/ibsfbc_dev_ctrl.c
AvgCyclomatic 11

Summarizing: fieldbuses

Entity Information: ibsfbc_dev_ctrl.c

AvgLine	79
AvgLineBlank	11
AvgLineCode	54
AvgLineComment	15
BlankLines	124
Comment	445
Comment_Embedded	16
CountDeclClass	0

Figure 1 The MosaiCode tool consists of the primary Mosaic visualization (A) with multiple tabs for additional views of the same data set, the Item Selection (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. For more specific data on an entity, the tooltip shows the full path and the value of the currently selected attribute, while additional attributes are shown in the Entity View (D). From left to right, the toolbar (E) allows for choice of displayed attribute, display characteristics, and control for the display of multi-version data.

summarizing the data, and building the UI components for exploration. Parsing occurs on the input data and the raw attribute values are recorded. In the analysis and summary steps, the tool determines a few common statistics of each attribute and establishes a threshold for clipping extreme outliers in the data set. Once the analysis is complete, the exploration of the data using the visualization can begin. We will now describe each view of the visualization in more detail.

B. Multiple coordinated views

The Mosaic View, Figure 1 (A), shows the entity nodes (visualized as tiles in the mosaic) logically grouped by top-level containers in the hierarchy and arranged and drawn as rectangular boxes. Selection of what numeric attribute is being displayed is controlled in the toolbar. The number of divisions of data (i.e., the number of colors) and the tile size is also controlled by the toolbar. The full path of an attribute, along with the numeric value of the current attribute, is available for each entity via a tooltip. Traversal of the hierarchy is made available visually through the use of a drill down/roll up functionality between hierarchical levels. The user may double-click on a container for drill down, and double-click on blank areas to roll up. In addition, during drill down the container can be opened for display in another tab. Individual containers can also be selected, which ties into other views. Additionally, it is possible to remove unwanted clutter in the mosaic view without drilling down by disabling and enabling entities from within the tree view.

The Container View, Figure 1 (B), shows the hierarchical structure of the data that comes from the *name* field. It allows for quick searching over the entire data set. If the user is looking for one specific file, then the hierarchy view will allow them to find the file by name. It will then allow the user to enable or disable the entity or container that the file is associated with or enable/disable other entities and containers in order to view specifically the entity they originally searched for. Furthermore, when an item is selected in the hierarchical tree view, it may be highlighted temporarily in the mosaic by holding down the Shift key. This is to aid in locating a certain container or entity among the large amount of data.

C. Summary Views

The Summary View, Figure 1 (C), shows common summary statistics calculated for each attribute. Values calculated during the analysis phase help determine the upper threshold of clipping outlier data points and the appearance of the summary histogram, and include the minimum, maximum, mean, median, range, and standard deviation. The tool compresses the data set using a natural logarithm and then uses the standard deviation of each attribute to detect extreme outlier data points and set a visualization threshold. This threshold is valued at three times the standard deviation from the median for that specific attribute and version.

D. Entity Information

The Entity Information View, Figure 1 (D), provides a fine-grained look into the data that determines the color that an entity assumes on the mosaic. It lists the attributes and their values for the currently selected entity in the mosaic. It also colors these attribute names in order to make it easy for the user to understand the state of the attribute. When a user clicks on an attribute within the Entity Information View, the mosaic automatically switches to that attribute.

E. Support for Historical Data

The tool supports multiple versions of each attribute and provides media-player style control buttons to switch between different versions. This allows the user to understand how the attributes have changed over the development of a software package. Furthermore, the impact of changes can be interpreted and communicated more easily when alternating the visualization between two versions. An example of viewing two different versions of data is given in Figure 2.

For the XML input, the historical values for each attribute are stored as bar ('|') separated values. The result is that the data for multiple versions has the same number of XML containers, entities, and attributes as a single-data version. The root XML element also contains an attribute *version* that uses the same attribute format to store descriptive names for each version, e.g., a timestamp or revision number. For CSV data, an additional field is used to store the name for each version, and each entity is repeated for each version. This allowed a simpler conversion process for CSV data. With multiple versions, the specific set of entities in each version of the input data may not be the same. Thus, it is able to reflect the addition or deletion of entities.

Once in MosaiCode, the particular version of interest can be selected using the controls in the toolbar, Figure 1 (E). The name of each version is displayed for context. Multi-version data can be animated to visually indicate growth of a particular point of interest, playback can be customized to loop and adjust the delay between version switching. As versions are displayed in the Mosaic View, the Summary View is also updated to the particular version.

F. Import/Export of Data

As mentioned previously, MosaiCode is able to import both XML and CSV file formats. It may also export to either format. This allows the user to directly take the data from their tool as CSV, import it into MosaiCode choosing which field to use for the name, and exporting the file to XML for future uses or conversely, to import from XML and export as CSV for use by additional analysis tools. Importing uses a SAX parser for XML because of its speed and generally low memory requirements. For exporting data, MosaiCode allows the user the option of either exporting the entire dataset, or a subset of the data. This allows for saving the particular area of interest for future visualizations.

G. Implementation

The tool was developed in C++ using Qt (qt.nokia.com). Qt provides the graphical and XML parsing tools needed. The tool is available for Windows, Linux, and OS X.

III. Demonstrating MosaiCode

The original motivation and support for the construction of MosaiCode was to support software maintenance tasks by ABB Inc. We use what was learned from the application of MosaiCode at ABB for the tool demonstration examples. ABB has used the tool to support two different software engineering activities.

First, project managers and software architects have used the tool to visualize data on code hot spots. That is, what are areas of the code with high code churn or high defect counts. These parts of the code are then prioritized for improvement in future releases. Before using the tool, managers could not readily use the hot spot information, as it was just a large .csv file. After deployment, management can now actively look at the results and are using it to visualize weekly churn data in ongoing development projects.

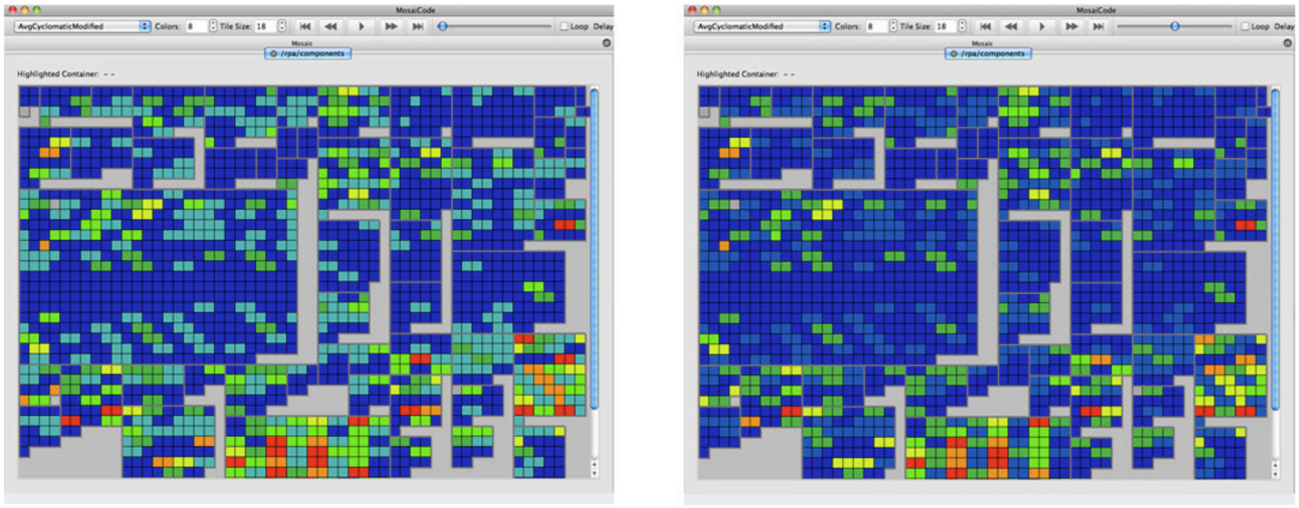


Figure 2 The MosaiCode tool supports multiple versions of the data. On the left is version 0.3 and on the right is version 1.0. The user can move between versions, or play them as part of an animation.

For this type of activity, managers/architects start with the output of their analysis tools in CSV format. With MosaiCode, they input the data choosing the field to use for the entity name. Once loaded, the entire mosaic is shown with the corresponding summary. The particular attribute of interest is then chosen. Additional tabs can be created to show more detailed views of specific directories. The context of specific value ranges is viewed in the summary. As managers/architects are exploring, particular entities are selected and viewed with the tooltip and with the entity information. In addition, the entity information can be used to turn the display to another attribute. For example, for the attribute complexity of a specific entity they may observe that the number of changes attribute is high and use that to turn the entire visualization. If the data has multiple versions, then they can quickly move through the data to see where the attributes values start to change. Once finished exploring, the original CSV data can be exported to XML for later presentation. This can be all of the original data or a subset consisting of only the particular subsystem and versions of interest.

To properly address this usage scenario, the visualization of large amounts of data is necessary. In some cases for ABB, up to 130,000 entities in over 100,000 containers are used. For data with lower amounts of entities, up to 61 attributes were included. While much of this is metric data, there is also quite a bit of textual data needed for context. For this data, the files size can often be over 40MB in the XML format, and 13 MB in the equivalent CSV format.

Developers and release managers also use the tool to visualize the impact that a change has on other parts of the software. This impact is calculated by a tool called Imp [3], which outputs raw results in a CSV file. When we first showed the raw results to developers, they were not using the data due to the complexity and size of the output file. Once the data was visualized with the tool, they started to use it to determine the magnitude of the risk of including late bug fixes into releases, as well as determining which areas to regression test due to code changes.

IV. Related Work

The design of MosaiCode is largely motivated by the Seesoft application [1]. 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 [2] in the tool sv3D.

The Seesoft idioms are used to visualize source code information to support a diverse set of engineering tasks. AspectBrowser uses the line view to display aspect dispersion [5]. Tarantula uses the same technique to visualize fault information from test cases for each line of code in a file [6]. An adapted version of the pixel view supports the detection of duplicated code by line using Duploc [7] and by token using CCFinder [8]. The idiom has also been exploited to visualize metrics supporting parallelization using SUIF's CodeViewer [9] and distributed collaboration using Augur [10].

V. Conclusions & Future Work

The experience of using MosaiCode with industrial data from ABB greatly influenced the design of the tool. The direct use of the output data from the various analysis tools used by ABB led to a need for more flexibility in the input format. This is what led to the CSV import feature, the associated interaction with the user, and the export features. As can be seen with the ABB examples, scalability was, and continues to be, a major concern with the tool. All parts of the tool, from the input of the data to the graphics, were tuned to support both the large amount of data to store, and the speed and interactivity of the display. We continue to look at ways of improving both of these aspects.

We plan to add and improve filtering options to assist in exploring the visualization. Also, we are working on improved layouts and groups of the mosaic in coordination with a hierarchical view.

Acknowledgment

This work was supported in part by funding from ABB Inc. Additionally, it was supported in part by a grant from the U.S. National Science Foundation under NSF grant CCF 08-11-21.

References

[1] 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.

[2] 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.

[3] M. Acharya and B. Robinson, "Practical change impact analysis based on static program slicing for industrial software systems," in 33rd International Conference on Software Engineering, Waikiki, Honolulu , HI, USA, 2011, pp. 746-755.

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

[5] W. G. Griswold, K. Y. Yoshikiyo, and J. J. Yuan, "AspectBrowser: Tool Support for Managing Dispersed Aspects," presented at the 1st Workshop on Multi-Dimensional Separation of Concerns in Object-Oriented Systems, Denver, Colorado, 1999.

[6] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of Test Information to Assist Fault Localization," presented at the 24th International Conference on Software Engineering, Orlando, Florida, 2002.

[7] S. Ducasse, M. Rieger, and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code," presented at the 15th International Conference on Software Engineering (ICSM'99), Oxford, England, 1999.

[8] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code," IEEE Transactions on Software Engineering, vol. 28, pp. 654-670, Jul, 2002 2002.

[9] S.-W. Liao, A. Diwan, R. P. J. Bosch, A. Ghuloum, and M. S. Lam, "SUIF Explorer: An Interactive and Interprocedural Parallelizer," presented at the 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Atlanta, Georgia, 1999.

[10] J. Froehlich and P. Dourish, "Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams," presented at the 26th International Conference on Software Engineering (ICSE'26), Edinburgh, United Kingdom, 2006.