Dominoes: An Interactive Exploratory Data Analysis tool for Software Relationships

Jose Ricardo da S. Junior, Daniel Prett Campagna, Esteban Clua, Anita Sarma, and Leonardo Murta

Abstract—Project comprehension questions, such as "which modified artifacts can affect my work?" and "how can I identify the developers who should be assigned to a given task?" are difficult to answer, require an analysis of the project and its data, are context specific, and cannot always be pre-defined. Current research approaches are restricted to post hoc analyses over software repositories. Very few interactive exploratory tools exist because the large amount of data that need to be analyzed prohibits its exploration at interactive rates. Moreover, such analyses typically require the user to create complex scripts or queries to extract the desired information from data. Here we present Dominoes, a tool for interactive data exploration aimed at end users (i.e., project managers or developers). Dominoes allows users to interact with different types and units of data to investigate project relationships and view intermediate results as charts, tables, and graphs. Additionally, it allows users to save the derived data as well as their exploration paths for later use. In a scenario-based evaluation study, participants achieved a success rate of 86% in their explorations, with a mean time of 7.25 minutes for answering a set of (project) exploration questions.

Index Terms—Design Tools and Techniques, Interactive data exploration and discovery, Evaluation/methodology.

1 Introduction

Software development leaves traces of activities – history of changes and who made those changes, list of bugs and issues related to the software, which changes fixed which issues, which files were changed together, and so on. This data, when analyzed, can help project managers and developers understand work dependencies in the team, development patterns, artifact coupling, location of frequent bugs among other relationships [1], [2], [3], [4], [5], [6]. Software teams are now increasingly analyzing historical data to inform their engineering and business decisions [7].

For example, Alice is a developer who wants to find someone who can help her in her bug-fix task. To reach this goal, she may create smaller subgoals, finding and making sense of the information from each subgoal, which then guides her next set of explorations. For example, Alice might first want to explore the files that she has changed for her (incorrectly resolved) bug-fix task. She may then want to find who has expertise in those files. To do so, she may find developers in the recent past who modified those files. Or she may go one step further to find the dependent files on her bug-fix changes, and then find the developers who were involved with those dependent files. The key point is that in exploratory data analysis (similar to exploratory programming) users have a goal in mind that requires further experimentation and creative problem solving to reach that goal [8], [9].

- *Jose Ricardo da Silva Junior is with Computer Science of Instituto Federal do Rio de Janeiro.*
  E-mail: jose.junior@ifrj.edu.br
- *Daniel Prett, Esteban Clua, and Leonardo Murta are with the Instituto de Computaćão of Universidade Federal Fluminense, Brazil.*
  E-mail: danielcampagna@id.uff.br,{esteban,leomurta}@ic.uff.br.
- *Anita Sarma is with Electrical Engineering and Computer Science of Oregon State University.*
  E-mail: anita.sarma@oregonstate.edu.

Manuscript received April 19, 2005; revised September 17, 2014.

However, very few interactive tools exist that allow end users to seamlessly explore their project history at interactive rates, and across different granularities. This happens because of the following challenges:

- Data is currently fragmented across different repositories, and has different formats. For example, version control systems store source code and its change history; issue tracking systems store bugs and discussion comments; mailing lists and chat forums contain communication records;
- Data scales according to the complexity/duration of the project and the granularity of analysis. For example, typical software projects contain thousands of artifacts, developed by hundreds of developers, over many years. If we are to consider the changes at fine granularities – that is, at the level of lines of code or methods – the scale of analysis makes interactive explorations and data visualization infeasible; and
- Data is produced continually over time. For example, active projects may have tens to hundreds of new issues, commits, and e-mail messages per day. Preprocessing this data in advance and using these static datasets does not reflect the current state of the project, as they may become outdated as the project (and its data) evolves.

While some tools allow project explorations, they typically focus on a small subset of the data and the questions that they can answer (e.g., EEL [10] focuses on expertise identification, and scopes the amount of data that can be analyzed). Others allow explorations over project characteristics and relationships that are defined a priori (e.g., Tesseract [11] allows exploration across three interlinked panels displaying file, developer, and bug dependencies), or require the end user to write queries [12], [13].

Information Fragments [3] is the only tool that allows

users to "compose" information by combining data fragments about specific work items, files, and their authors. It allows answering one question at a time (e.g., who is working on what), but is not suited for re-composition or backtracking, and therefore, unsuited for exploratory data analysis. Moreover, Information Fragments, like the other tools, operates at a predefined (coarse) granularity level (i.e., files). Providing interactive explorations at coarse-grain is itself computationally expensive and becomes prohibitively expensive if one has to allow navigation from fine-grain (changed line of code) to coarse-grain (expertise in a project) levels and vice versa —a key need in exploratory analysis.

Here we present Dominoes, an interactive, project oriented exploration tool, where end users can explore different relationships across the different software project elements (e.g., which developer has changed a method that I have changed in the past). Our approach organizes data extracted from software repositories into matrices that are then visually represented as domino tiles reflecting relationship between two project elements (e.g., [commit|method]). It allows users to interact with these tiles, such that tiles can be interconnected based on a set of matrix operations to derive additional domino tiles. These derived domino tiles in turn represent specific project entity relationships (e.g., number of commits in which two methods co-occurred), and can be used for further explorations or visualizations.

Dominoes has been specifically designed to allow data exploration by: (1) representing all the data as matrices and allowing transformation of these matrices by leveraging software project relationships, (2) allowing users to traverse between coarse- and fine-grained project explorations, (3) providing a domino game metaphor for intuitive and interactive explorations of software relationships, (4) showing the intermediate and final results through graphics, and (5) transferring heavy data computation processing to Graphical Processing Units Parallelism (GPU).

In previous papers [14], [15], we presented the underlying approach behind Dominoes, how it extracts data from different software repositories, the types of operations, and the GPU processing and data modeling. Besides that, in another paper [16] we presented results from empirical studies about how our approach can be used to identify developers expertise at a fine-grained level (method), how it varies across time, and the speedup of running the explorations in GPUs. Thus far, we presented the data analysis by using a command line interface.

In this paper, in addition to describing the approach, we introduce a novel interactive Graphical User Interface (GUI) for Dominoes, to enable end-user interactions. We present a scenario-based user study with nine participants, where seven were software professionals. This evaluation not only helped us assess the usefulness of Dominoes, but also sheds light on how developers explore software project relationships. We stopped at nine participants, since we reached saturation, that is, we were seeing the same patterns in which participants were combining the Dominoes tiles to answer the scenario questions. The user study focused on answering the following research questions:

RQ1: How useful is Dominoes in facilitating exploration of project repositories in terms of effectiveness and efficiency?

RQ2: What types of explorations do participants perform when using Dominoes?

Our results show that participants in the study achieved a success rate of 86% in their explorations, taking an average of 7.25 minutes for answering (project) exploration questions. Participants used intermediate data visualizations as check-points to guide and correct their exploration paths. While there was a learning curve, participants became adept in completing the study tasks after their initial interactions.

We then performed two additional evaluations. First, we interviewed five software professionals to determine the kinds of questions that they have to answer in their everyday work and how Dominoes could help in answering these questions. Through these interviews, we identified different real-world exploratory questions that Dominoes can help answer. Then, as some participants of the previous evaluation suggested adding automation features to Dominoes, we implemented a recommendation support where developers provide the expected endpoints of an answer and Dominoes shows all answers that respect such endpoints. We could observe that such feature is promising, showing the correct answer in the top two to five recommendations.

The remainder of this paper is organized as follows: Section 2 presents the Dominoes approach, summarizing its architecture, presenting the basic tiles extracted from software repositories, and showing operations that allow creation of new tiles. In the same section we also present the design rationale of Dominoes GUI. Section 3 presents the scenario-based user study used to assess the usefulness of Dominoes. Section 4 presents the feedback collected from industry professionals regarding the applicability of Dominoes in their everyday work. Section 5 presents how a recommendation feature would help Dominoes to answer the scenario questions introduced in Section 3. Section 6 presents some related work while Section 7 concludes the paper and discusses future work.

2 DOMINOES

In this section we first present an overview of the Dominoes infrastructure and its tiles. We then discuss the design rationale behind Dominoes. Finally, we present the GUI of Dominoes, using a scenario to explain how users can interact with its different features.

2.1 Dominoes Infrastructure

Dominoes is designed such that it extracts data from software project repositories and cross links the associated information. It is comprised of a set of modules that extract and process the data, as shown in Fig. 1.

The **Extractor** module (Fig. 1, blue box) accesses the repositories to extract data that is then used for building relationships. For example, commits, issues, discussions about a commit, or pull request can be collected from GitHub. Currently, we mine Git (for version management) by cloning and accessing the remote project repository. Dominoes then pre-processes the data from the repository by analyzing which files, packages, classes, and methods were modified to create a tree of all modifications. We use the Eclipse

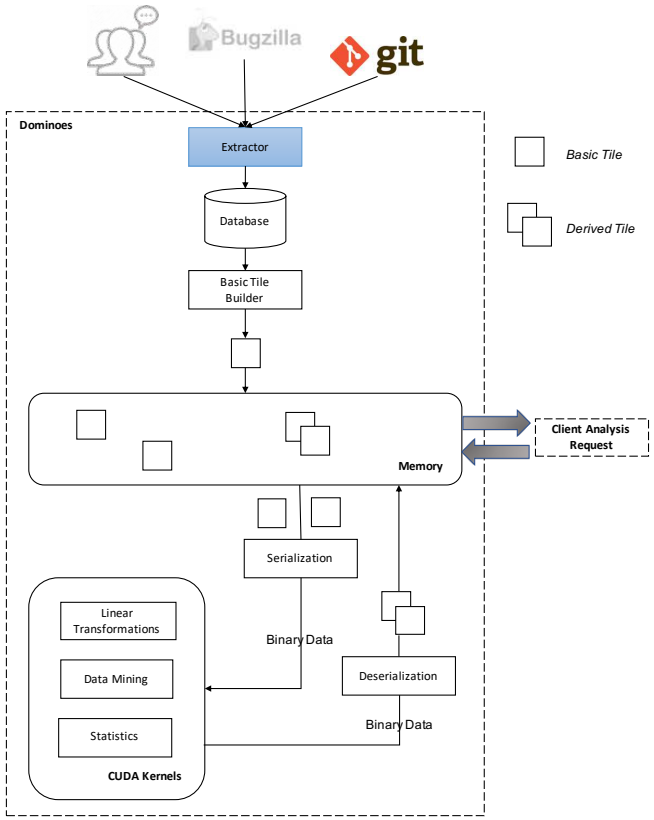


Fig. 1. Dominoes' architecture.

ASTParser (suitable for Java-based projects) to obtain a fine-grained view of the modifications for each commit. For example, even if we represent changes at the package level (for a coarse-grained analysis), we know exactly which classes, as well as which methods were modified. This information is then stored in a relational database. After the initial data collection, the database can be updated incrementally to accommodate subsequent project activities.

After the pre-processing stage, the **Basic Tile Builder** module constructs the *basic building tiles*. The basic building tiles are the elements used by Dominoes to organize and represent data relationships. They consist of two-dimensional matrices of elements from the database. These tiles then become available to the users through the UI, allowing them to manipulate the tiles according to their needs. Basic building tiles can be combined or manipulated to create *derived building tiles*, which can be further combined with other basic or derived tiles.

Dominoes provides several operations over the data, which are performed by the following modules: (1) **Linear Transformations** for multiplication, transposition, aggregation, and sorting of tiles, (2) **Data Mining** to obtain support, confidence, and lift of tiles, and (3) **Statistics** for calculating the mean or median of tiles. In Section 2.2 we discuss the basic and derived tiles together with the linear transformations provided by Dominoes. The data mining and statistics operations are not discussed here as they were not used in the analyses presented in this paper, but are detailed in [15].

Performance becomes an issue when we compute relationships at the fine-grain level. Therefore, in order to allow efficient computation at interactive speeds, we model the above operations into a Single Instruction Multiple Data (SIMD) architecture, making it possible to execute the intensive matrix operations on a GPU device. When a matrix manipulation is required, Dominoes forks its execution by triggering the respective asynchronous GPU code (called *kernel*) based on the desired operation.

While designing Dominoes, we considered different approaches for modeling our data in a way that would make it easy and intuitive for end users to manipulate the data. One approach was the MapReduce method [17]. MapReduce relies on two important operations: mapping data to a key and reducing this data to a smaller set. However, among all the operations available in Dominoes, only one involves dataset reduction ("aggregate). Additionally, MapReduce involves constantI/O operations, leading to a constant data transfer between CPU and GPU memory. It is important to note that data transfer between memories is a bottleneck for GPU applications [18]. Modeling the data structure as matrices allows optimal parallelization, especially in the case of operations that have only local data dependencies and avoid code divergence, as it is in our case.

Except for the CUDA kernel operations, Dominoes is developed in JAVA. Performing operations over these tiles, therefore, requires communicating the data with the kernels in CUDA. Consequently, Dominoes implements a Java Native Interface (JNI) that is responsible for the **Serialization** and **Deserialization** of building tiles to and from CUDA.

2.2 Dominoes Tiles

Dominoes consists of a set of *basic* building tiles that can be combined to produce *derived* building tiles. Here is a list of the basic building tiles available in Dominoes:

- **[class|method]** ([Cl|M]): relationship between a class and its constituent methods, where cell [i,j] has a value of 1 when class i contains method j.
- **[file|class]** ([F|Cl]): relationship between a file and its constituent classes, where cell [i,j] has a value of 1 when a file i contains class j.
- **[commit|file]** ([C|F]): relationship between commits and files, where cell [i,j] has a value of 1 when commit i adds or modifies file j. Note that the index i does not denote the commit id.
- **[commit|method]** ([C|M]): relationship between commits and methods, where cell [i,j] has a value of 1 when commit i adds or modifies method j.
- **[developer|commit]** ([D|C]): relationship between developers and their commits, where cell [i,j] has a value of 1 when developer i is the author of commit j.
- **[package|file]** ([P|F]): relationship between a package and its constituent files, where cell [i,j] has a value of 1 when a package i contains file j.
- **[issue|commit]** ([I|C]): relationship between commits and issues, where cell [i,j] has a value of 1 when commit j implements/fixes issue i.

Dominoes allows different operations to be applied over these tiles. The multiplication operation ($\times$) multiplies two dominoes tiles generating a derived tile. In fact, it behaves in the same way of matrix multiplication. The transposition (^{T}) operation transposes rows and columns in a matrix that represents a Dominoes tile. Aggregate ($\sum$) is an operation

that sums up either all the rows or columns of a matrix, producing a column or row matrix (i.e., a vector). As an example, applying the "aggregate by row" over the [C|F] tile ([$\sum$C|F]) produces the count of changes over each file, whereas aggregating by column on the same tile ([C|$\sum$F]) produces the count of files changed by each commit. Finally, (↑) and (↓) represent increasing and decreasing sorting orders, respectively. In this case, [C$^{\uparrow}$|F] performs an increasing sort over commit values.

The basic tiles can be combined in many different ways. Here we list a small set of derived tiles that are computed by using only multiplication and transposition:

- **[method|method]** ([M|M] = [C|M]^T × [C|M]): represents method dependencies, where [i,j] denotes the strength of dependencies between methods *i* and *j*. Additionally, the main diagonal of this matrix represents how many times a method has been modified (also achieved through [$\sum$C|M]). The rationale behind this matrix is based on logical coupling, as elements that are co-committed share some programming logic. We can also create an [M|M] matrix through program analysis– in this case it would be a basic building tile. Such [M|M] matrices have been explored by Steward [19] in creating Design Structure Matrices.
- **[class|class]** ([Cl|Cl] = [Cl|M] × [M|M] × [Cl|M]^T): represents class dependencies, where [i,j] denotes the strength of the dependency between class *i* and class *j*. In Dominoes, results at a higher level of abstraction can be easily obtained by combining tiles from lower level (e.g., composing this [Cl|Cl] tile with [F|Cl] or [P|F] × [F|Cl] would raise the abstraction to the file or package level, respectively).
- **[issue|method]** ([I|M] = [I|C] × [C|M]): represents the methods that were changed regarding an issue. Applying an aggregation operation over the (bugfix) issues allows identifying the methods that are "buggy", as these methods tend to have a high number of issues related to them.
- **[developer|method]** ([D|M] = [D|C] × [C|M]): represents the methods that a developer has changed and can be used to identify experts on a particular method.
- **[developer|class]** ([D|Cl] = [D|M] × [Cl|M]^T): represents classes that a developer has changed. [D|Cl] uses the composition operation to provide expertise information at the class level, which is typically used during bug triaging [3].
- **[developer|developer]** ([D|D] = [D|M] × [M|M] × [D|M]^T): represents the social dependency among developers due to the underlying technical dependencies in their work. This derived building tile uses other derived building tile ([M|M] and [D|M]) in its definition.
- **[commit|commit]** ([C|C] = [C|M] × [C|M]^T): represents similarity among commits by considering common methods that were changed by them.

2.3 Design Guidelines

Dominoes' graphical user interface was designed to enable end users, who in our case can be project managers or developers, to perform exploratory data analysis over their projects. The Dominoes design leverages the following set of guidelines that, when blended together, provide a highly interactive and powerful tool for exploratory analysis of software engineering data.

Domino tile metaphor: A primary goal of Dominoes is to allow a user to reason about the data relationships among project elements. Additionally, we want users to be able to explore different project relationships. We, therefore, use graphical elements that resemble domino tiles to allow users to explore their project by combining these tiles. Users can try different (data) tile combinations and operations, such that they can explore different relationships among the project elements.

We allow direct manipulation of graphical pieces, instead of a query-based approach [12], [20] because: (1) the tiles allow users to more easily visualize the different project elements, and how they can be combined together, (2) users can incrementally compose their final query by exploring the different project relationships, and (3) users do not need know or learn a specific query language, or formally express ahead of time how the data should be integrated; creation of appropriate queries is often a barrier for end users [3].

Data, operations, and visualizations are first-class elements: Performing exploratory data analysis, by its definition, involves exploring and evaluating different aspects of the (project) data. We leverage visualizations to enable users to check the results of their exploration, and refine their exploration as needed. Further, we explicitly treat *data* (project elements), *operations* that can be performed on the data, and *visualizations* as first-class entities, such that users can seamlessly explore different data elements by operating over the tiles, and checking the results of the exploration through visualizations.

Seamless transition across granularities: Different types of questions can be answered at different levels of granularity. For example, a class may need to be refactored if there are multiple developers working on the same class, which might lead to merge conflicts. Similarly, a package may need to be refactored to match the structure of a distributed team to improve coordination (Conways Law [21]). Sometimes, the same question can be asked at different levels of granularity. For example, developers' past edits to a class can be used to determine the developer who is an expert on a class [1]. However, edits at the method level can reflect the expertise coverage of a developer on a class [16]. We, therefore, allow seamless transition among granularities (low to high) by leveraging different types of composition matrices ([class|method], [file|class], [package|file], etc.) and operations over these matrices.

Exploration at interactive speed. As noted, a key requirement of Dominoes is to support project exploration across different project elements and granularity. For this to be successful, the tool needs to have high computation performance. Large software projects have hundreds of developers, thousands of files, which in turn may contain many methods, and thousands of commits. This quickly becomes a big-data problem. We therefore, allow incremental data updates, use GPU for big data transformations (matrix transformations), and adopt thresholds in visualizations and in segmenting the data to allow quick data explorations. Our GPU implementation achieved up to three orders of magnitude of performance increment when compared with our

efficient matrix-processing algorithms running in CPU [16].

Exploration history: A key aspect of exploratory data analysis is investigating different ideas, and then identifying a (good) solution. Such an exploration requires backtracking of ideas or jumping off of a previous (partial) analysis. We allow users to perform such exploration of project data through: (1) undo-redo of operations, (2) derived tiles that represent partial analysis steps that users can reuse, and (3) exploration paths that are archived for each derived tile so that users can understand how they arrived at a solution (derived tiles), or reuse the exploration path partially or in its entirety. The exploration paths can be viewed in a tree format (along with backtracks).

Extensibility: A primary goal of our work is to allow Dominoes to be easily extended to include different types of data, operations, or visualizations. We, therefore, decouple the data collection from consumption. New types of data can be easily collected by creating wrappers for different types of repositories (e.g., a choice of version control systems such as SVN, Git, Mercurial), or different types of data (e.g., version histories, issues, email). Currently, we collect data from version histories (Git) and extract issue information from Bugzilla. Additional wrappers for different repositories or types of data can be "plugged" into the system. Similarly, we currently have visualizations such as, network graph, matrix view, tree structure, and bar chart. Other visualizations, such as heat-maps or radial charts, can be easily added. This extensibility of data collection and visualization is possible because Dominoes uses matrices, a simple and general data structure.

2.4 Dominoes GUI

Dominoes interface comprises four panels as shown in Fig. 2. The top pane (Fig. 2 (a)) allows users to select a project to work with, as well as the time frame for analysis. Besides that, it presents a timeline view of the project activities regarding the number of commits and the number of new issues. Users can use the "Project" button to import a new project or update an existing project with new data. Additionally, the library panel (Fig. 2 (b)) holds all the dominoes tiles (the basic building tiles and the derived ones). The editor canvas (Fig. 2 (c)) is where users can compose the tiles or operate over them. Finally, the visualization canvas (Fig. 2 (d)) is used for presenting different types of data visualization allowed by Dominoes.

We explain the features of Dominoes through a hypothetical scenario. Let us consider Alice as our persona. She needs to identify remote team members who can collaborate with her. In the rest of the section, we describe how Alice uses Dominoes to get this information. Note, we use the (real) data from the Apache Derby project when explaining this (intentionally simple) scenario. In real life, developers can explore more complex software relationships.

Alice starts Dominoes and accesses its main window (Fig. 2). Alice decides to use the last thirteen months for her analysis (Jan 2013 to Jan 2014) and, after selecting the period in the project panel, she clicks on the "Set" button to start the analysis. Dominoes then generates a collection of basic building tiles representing relationships in the project (Fig. 2 (b)), as discussed in Section 2.1.

Alice decides to start her explorations by first looking at each tile. She decides to pick the [commit|method] ([C|M]) and [class|method] ([Cl|M]) tiles. To perform this action, she double clicks these tiles in the library pane (Fig. 2 (b)) to copy them to the editor canvas pane. The editor canvas (Fig. 2 (c)) with the tiles is shown in Fig. 3.

Alice is interested in knowing which commits were related to which classes, she therefore tries to combine the [C|M] and [Cl|M] tiles. However, note that to combine (multiply) the matrices they need to have an equal number of column and row for the first and second matrix, respectively, which translate to the same edge relationship. Since [C|M] and [Cl|M] do not share the same dimension, Dominoes does not allow Alice to connect these two tiles (red colored compartments in Fig. 4).

Alice realizes that the relationship types to be combined (method-method) needs to be aligned between the two tiles. So, she transposes the [Cl|M] matrix by double clicking the tile, which graphically swivels the tile in the editor canvas, as shown in Fig. 5. As an alternative, she could have right clicked on the tile and selected "transpose" operation.

Once the [Cl|M] matrix is transposed, Dominoes allows both tiles to be connected by presenting green colored compartments. This leads to the derived tile [C|Cl], containing information about the commits involved with the classes. Fig. 6 illustrates this operation and the derived tile.

In order to avoid redoing these operations, she saves the tile by clicking over the piece and selecting the "save" option. Once she performs this operation, the derived tile become available in the library panel, as can be seen in Fig. 2 (b) (the last piece in the column). In order to ensure that she remembers the logic behind the derived tile, since there can be other ways to achieve the same relationship, each saved tile shows its history (small gray letters) below its name (Fig. 6). Hovering over a tile also provides the information about that tile in a tool tip, as well as the number of rows and columns in the matrix.

Alice then continues her exploration in Dominoes. She decides to create a [commit|commit] matrix by combining the [C|Cl] tile with its transpose to generate the [C|C] tile. She then multiples the following tiles: [D|C] ([developer|commit]), [C|C], and [D|C] transposed to generate the [developer|developer] matrix. This matrix identifies developers that committed over the same classes, which presents relevant information for identifying which developers are able to replace others because their commits involved a common set of classes. Alice decides to save the [D|D] tile for future use.

Now that Alice has created the underlying data, she wants to identify which developers are interconnected with her. Dominoes allows four different types of visualizations to be used: network graph, matrix view, tree structure, and bar chart, as shown in Fig. 2 (d) and Fig. 7.

As Alice wants to visualize the interconnection among developers, she selects the "Graph" view to visualize the [D|D] matrix. This visualization (Fig. 2 (d)) shows developers (blue nodes) who are interconnected because of a common set of classes that they committed together. The view allows Alice to set a threshold on the edge weight; that is, she can filter out developers whose edges (number of connections) are below a certain value. When she searches

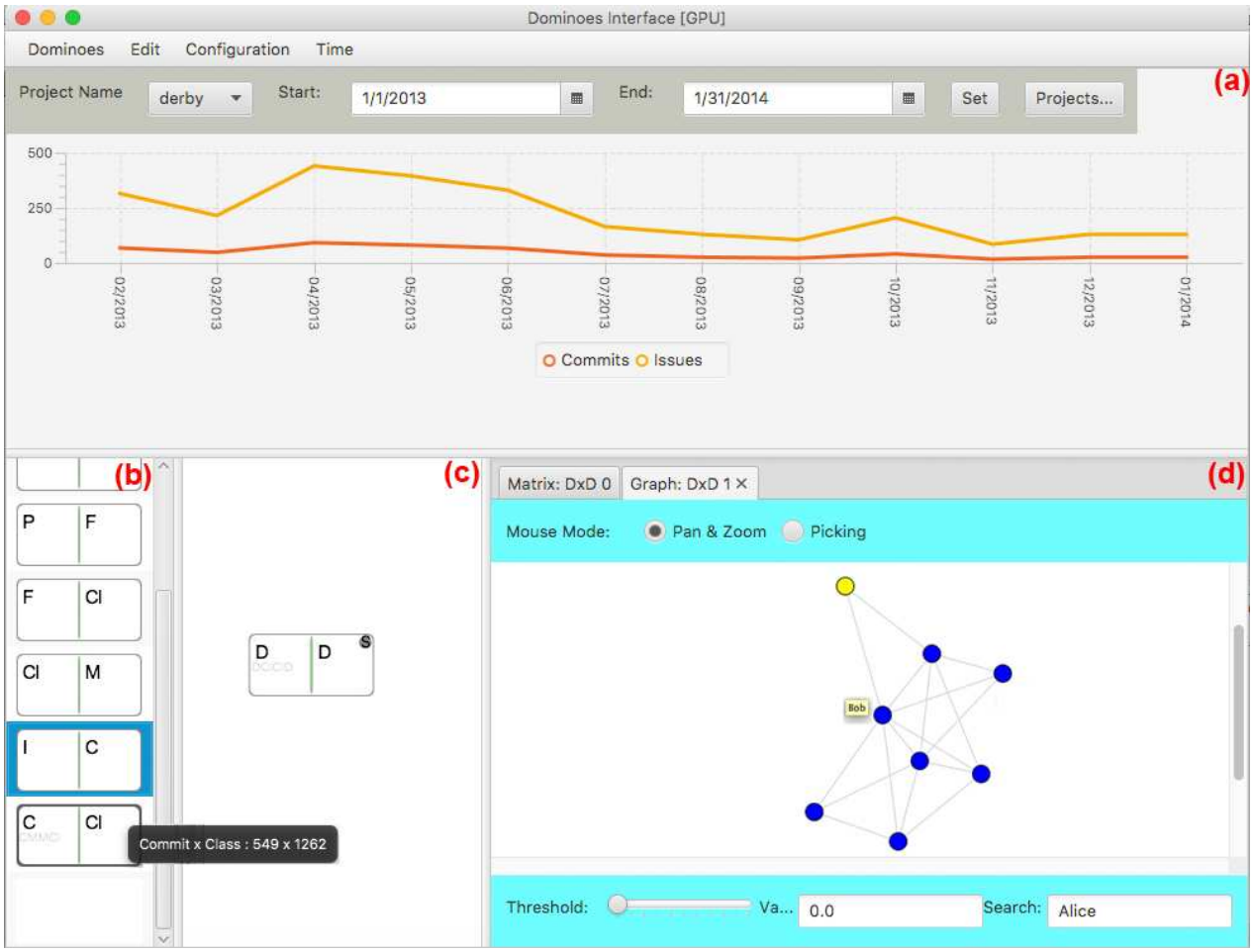


Fig. 2. Dominoes' main interface. It is composed of four panels: (a) project selection, (b) library of tiles, (c) editor canvas, and (d) visualizations.

Fig. 3. Editor canvas after adding [C|M] and [Cl|M] tiles.

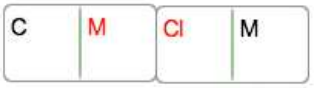


Fig. 4. Wrong combination feedback provided by Dominoes.

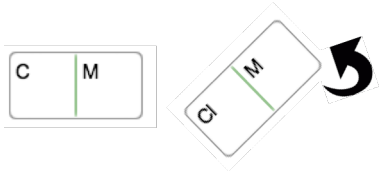


Fig. 5. Dominoes' piece transposition.

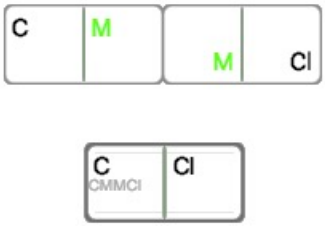


Fig. 6. Dominoes' piece connection, producing the derived tile [C|Cl].

for her name (lower right corner of the UI in Fig. 2 (d)), Dominoes highlights in yellow the node representing her. Then she can follow the edges from her node to identify the two other developers who are connected to her. Hovering over a node shows the name of the developer (here, Bob) connected to her. Besides that, as seen in the top left of Fig. 2 (d), Dominoes allows multiple visualizations to be open on separate tabs allowing different perspectives. All charts stay active as long as the tile that was used to create the visualization remains in the editor canvas.

3 User Study

Our evaluation goal is to understand how developers would explore their project relationships and assess how Dominoes can facilitate such explorations. Through a scenario-based user study we assess Dominoes in terms of its effectiveness, efficiency, and usability in performing project explorations.

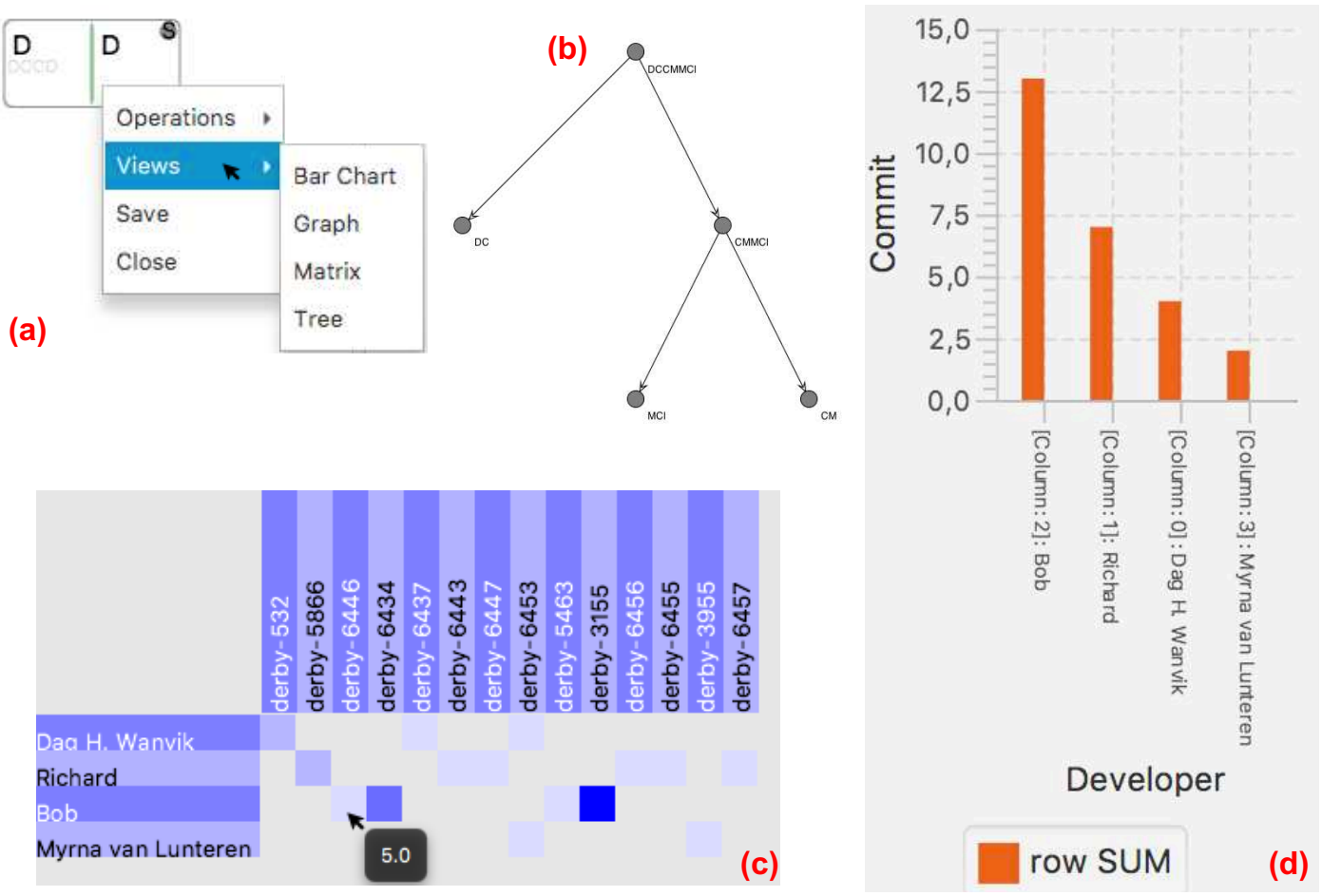


Fig. 7. Different types of visualizations allowed in Dominoes: (a) the contextual menu for selecting the type of visualization, (b) tree structure, (c) matrix, and (d) bar chart visualization.

We also investigated the different types of exploration, backtracking, and corrections participants performed in their exploratory data analysis, resulting in two research questions:

RQ1: How useful is Dominoes in facilitating exploration of project repositories in terms of effectiveness and efficiency?

RQ2: What types of explorations do participants perform when using Dominoes?

3.1 Methodology

We used a scenario-based, think-aloud study, where we presented participants with situations where they had to investigate a project's history using Dominoes to answer a set of questions. Think-aloud studies help us better understand participant behavior–which aspects of Dominoes they considered using, where they faced problem, how they problem-solved. A subset of the Apache Derby project history has been used to formulate the following four scenarios (i.e., tasks). The project history comprised data from Jan 2013 to Jan 2014. The selected period included **602 commits, 1,316 changes to classes, 7,792 changes to methods**, and **264 issues**. We chose Apache Derby as our test project because it has been stable, long-living, and active since its beginning.

3.1.1 Scenarios

Participants were presented with four scenarios. Each scenario was guided by the list of questions that developers ask, as presented by Fritz and Murphy [3]. They identified 78 questions grouped into eight domains. Our scenarios build on these questions, but map to more than one question. We selected those questions (from [3]) that require investigation across at least two software dependencies and include a historical component. We chose compound questions so that we could: (1) investigate the process followed by participants as they explored different aspects of their project data and (2) assess Dominoes' usefulness in aiding this process. Of course, end users can explore simpler or more complex software relationships. Each consecutive scenario was slightly more complex, with respect to the complexity of operations and granularity of data.

We had all participants follow the same order of tasks as we wanted participants to first learn to use Dominoes in less complex situations. We recognize that such a task sequencing leads to learning effects, but this is not a problem as all participants had the same training opportunity. In the following, we discuss each scenario, as well as the mapping of these scenarios to the questions (referred to as Q#) in Fritz and Murphy work [3].

Scenario 1: *"Richard is planning on performing a major refactoring over the code he has worked on in the last 3 months. He wants to analyze the commit history of his modifications to identify which developers might be affected by his refactoring. How can he do so?"* This scenario maps to Q13 (*"Who is working on the same classes as I am and for which work item?"*) or Q26 (*"How do recently delivered changes affect changes that I am working on?"*). One way a user can answer this scenario is by identifying all the developers whose changes are related to Richard's changes (Richard is an actual Derby developer). One possible answer when considering changes at the method-level is [D|D] = [D|C] $\times$ [C|M] $\times$ ([D|C] $\times$ [C|M])^T, where similar commits (i.e., changing the same methods) are used to identify the dependency among developers. This scenario represents the basic usage of Dominoes, where tiles can be combined based on their underlying relationships by using the multiplication and the transposition operators.

Scenario 2: *"Knut has been a core developer in Derby, but lately has too many issues to resolve and is not able to fix them quickly enough. Therefore, his manager has decided to give him a team of developers. Knut would like to include in his team people he has worked with before, in the context of fixing issues in the past 6 months. How can Knut identify the developers he should include in his team?"* This scenario maps to Q11 (*"What classes has my team been working on?"*), Q43 (*"Who has made changes to [a] defect?"*), or Q1 (*"Who is working on what?"*). An ideal solution is identifying the issues that were related to commits that Knut made, and then identifying other developers who also committed to these issues: [D|D] = [D|C] $\times$ [I|C]^T $\times$ [I|C] $\times$ [D|C]^T. This scenario is more open-ended than the previous scenario, incorporates additional data source (issues), and spans a longer time period.

Scenario 3: *"A senior developer, Susan, wants to identify the appropriate developer to be assigned to a new task that requires significant modifications to the class java.drda.org.apache.derby.drda.NetworkServerControl. She wants to do so using the development history of the class in the last 4 months. How can Susan identify the best developer for this task?"* This scenario maps to Q5 (*"Who to assign a code review to? / Who has the knowledge to do the code review?"*), Q6 (*"What have people been working on?"*), or Q8 (*"What is the evolution of the code?"*). An ideal solution involves the following operations: [D|Cl] = [D|C] $\times$ [C|M] $\times$ [Cl|M]^T. This operation is more complex than the previous ones, as there is no piece that directly involves changes (commits) to a class. Participants have to navigate from fine-grain (method) to coarser grain (classes). They also need to use the appropriate visualization (graph) to find the right information from that visualization.

Scenario 4: *"The Derby team has realized that they have not refactored their code base in a while and functionalities have been added in an ad-hoc manner, so they need to refactor their code base. However, they have limited time for doing this, so they want to first identify the classes that are the most brittle – that is, classes that have undergone a lot of changes in the last six months of development. How can the team do this refactoring?"* This scenario can be mapped to Q23 (*"Which class has been changed most?"*), Q8 (*"What is the evolution of the code?"*), or Q27 (*"What code is related to a change?"*). This scenario is the most open-ended, and there are several different ways to answer it. One possible answer is: [C|Cl] = [C|M] $\times$ [Cl|M]^T; [$\sum$C|Cl$^{\downarrow}$], where a user sums the number of times that a classs method has changed, followed by a (decreasing) sorting operation on the classes. A user may also analyze how much of a class has changed (the number of methods in the class that has been edited).

TABLE 1
Participants' characteristics backgrounds.

P#	Gender	CM Experience		Degree	Repo Analysis Experience
		Industry	Academic		
P1	F	-	4 years	Ph.D.	No
P2	M	-	2 years	MS	No
P3	M	1 year	4 years	MS	No
P4	M	4 years	3 years	BS	No
P5	M	0.5 years	5 years	MS	Yes
P6	M	8 years	2 years	BS	No
P7	M	4 years	1 year	BS	No
P8	M	8 years	4 years	BS	Yes
P9	F	7 years	2 years	MS	Yes

3.1.2 Participants

We were interested in how managers and developers would use Dominoes to explore project relationships. Since novice developers as well as non-programming managers will need to understand their projects, we selected developers with different backgrounds and experiences. Table 1 shows the general demographics.

All participants had experience in software development and version control systems. Seven out of nine (P3-P9) had experience working in an industrial setting: two with less than 2 years of experience, whereas five had 4 or more years of experience. Of the two participants with only academic project experience, P2 was a novice (2 years) and the other (P1) had 4 years of experience. Three participants (P5, P8, P9) had experience exploring version control repositories as part of their job. They had used the command line options to analyze version histories to identify who made a change in the past or how a class had evolved. Participants had varied academic backgrounds – ranging from BS (four), MS (four), and Ph.D. (one) in computer science, stating that they had worked on programming as part of their academic studies. Seven participants were male and two were female.

3.1.3 Study Design and Analysis

Participants were instructed on the overall experiment setup to get their (informed) consent. We then explained the think-aloud method, followed by a video tutorial about Dominoes. Participants were also reminded about the concepts of matrix multiplication and transposition. These steps took around 25 minutes. After this, we gave participants five minutes to explore Dominoes, using a different database from the one used in the experiment.

Once participants had finished the training, they were presented with the four scenarios, one at a time. Each scenario was time-boxed to 15 minutes; participants were asked to move to the next task after this time. We time-boxed each scenario so that all participants could experience all the scenarios. We did not provide any help to participants during the session. At the end of each scenario, participants saved a screenshot of their solution. Participants talked-aloud when answering question for each scenario. We recorded audio of what the participant said and screen-capture video.

The first author, sitting off to the side, took notes about unusual actions or problems faced by the participant. At the end of the study, participants were given a short break, during when the author conferred his notes and the recordings to identify actions that needed clarifications. He then conducted a semi-structured, retrospective interview. For unusual actions or mistakes, he replayed the screen-capture video to participants and asked them about their action(s).

At the end of the interview, participant filled out an exit survey containing the following questions about their experience with Dominoes: (1) *Were Dominoes tiles easy to interact with?* (2) *Were Dominoes derived tiles easy to create and use?* (3) *Were Dominoes operations easy to use?* (4) *Were Dominoes visualizations useful in answering the questions?* and (5) *Did Dominoes help you to investigate the Apache Derby project?* These questions had a five-point Likert scale, ranging from Strongly Disagree (1) to Strongly Agree (5). Additionally, participants used the Microsoft Product Reaction Card [1] to best select their experiences during Dominoes usage.

We answer the RQ1 through a quantitative analysis of participant data and feedback (Section 3.2.1), and RQ2 through qualitative analysis, as detailed below.

We used a baseline code set inspired by previous research [22], [23] to analyze the exploration behavior of participants and the barriers they faced (Section 3.2.2). We created new codes to represent participant actions when using Dominoes (Table 4). We coded the video recording by annotating the video with the specific actions. Two researchers performed this action on a participant's data and performed negotiated agreement to reach consensus about the code categories and the rule set. A code was dropped if we could not reach consensus after three rounds of discussions. After this step, the first author coded the rest of the participant data.

3.2 Results

Here we present our evaluation results structured around our research questions.

3.2.1 How useful is Dominoes in facilitating exploration of project repositories?

We answer this question by analyzing Dominoes in terms of its effectiveness, efficiency, and user satisfaction.

Effectiveness: We calculate the correctness scores of participants for each scenario (see Table 2). On an average 3.44 (out of 4) answers were correct (median 4). Five out of nine participants got all answers correct; whereas, three participants got one answer incorrect, and one participant got two answers incorrect (P5). On further analysis regarding P5, we found that he did the least amount of exploration of the different tiles and ways to combine them (see Section 3.2.2). Moreover, he was convinced that his answers were correct for all the scenarios.

All participants answered scenario 4 correctly, despite it being an open-ended question. When considering P5, although he did not explore different options for this scenario, he did formulate the scenario correctly. Moreover, this allowed him to reuse the concepts (e.g., sorting the bar graph) from Scenario 3, which he had answered correctly.

1. Developed by ®. 2002 Microsoft Corporation. All rights reserved.

TABLE 2
Task Completion Times (in minutes) and correctness (✓) or failure (✗).

P#	Scenario				Average
	S1	S2	S3	S4	
P1	7.3 ✓	3.7 ✓	12.4 ✓	10.1 ✓	8.38
P2	3.9 ✓	6.1 ✓	8.9 ✓	7.2 ✓	6.53
P3	4.2 ✓	3.8 ✓	6.4 ✗	6.1 ✓	5.13
P4	7.5 ✓	5.8 ✓	14.2 ✓	5.9 ✓	8.35
P5	3.7 ✗	2.5 ✗	7.8 ✓	8.3 ✓	5.58
P6	8.1 ✓	3.4 ✓	10.4 ✓	10.1 ✓	8.00
P7	7.3 ✓	8.1 ✗	15.6 ✓	8.3 ✓	9.83
P8	3.1 ✓	4.2 ✓	8.7 ✓	4.2 ✓	5.05
P9	4.2 ✓	2.9 ✓	14.2 ✗	12.3 ✓	8.40
Average	5.48	4.50	10.96	8.06	7.25

Finding 1: Participants using Dominoes reached correct answers in 86% of the cases.

Learning: On average, participants took 7.25 min. to complete the tasks (see Table 2). Some participants (P3, P5, P8) finished the tasks relatively quickly (avg. = 5.25 min), whereas some (P1, P4, P6, P7, P9) took longer (avg. = 8.6 min). P8 took the least amount of time per task (avg. 5.05 min) and got all the questions correct. On the other hand, P5 spent less time on scenarios 1 and 2 and got both of them wrong. Apart from P5, there is no clear pattern between the time spent on a question and correctness. Note that both P5 and P8 performed repository analysis as part of their jobs, however, P8 had more experience.

Past studies have shown that after a substantial training over SQL (1.5 hours), users could write queries that join two domains in a mean time of 5.10 minutes [24]. The first two scenarios that required combining data from two domains (S1: [D|C] $\times$ [C|M], S2: [D|C] $\times$ [I|C]^T) were completed in an average of 5.48 and 4.50 minutes, respectively. This is very close to what can be accomplished using queries, but only after extensive training. In contrast, our participants had only 15 minutes of training in Dominoes.

Scenarios 3 and 4 were open-ended and needed more exploration, and hence more time (10.96 min and 8.06 min, respectively). Scenario 3 took the longest time. We found this was because participants had to sort on a specific column and use the bar chart, therefore, they had to figure out the right operation (sort), on the right data column/tile, and the correct way to order the bar chart.

Please notice that P7 spent 15.6 minutes (exceeding the time-box of 15 minutes). It was due to a short computer frezing during screen capture. However, the answer to the question has been formulated during the specified time.

Finding 2: Dominoes has a 15-minute learning curve. After 15 minutes of training, participants could use Dominoes (taking about 5 min. for answering each structured scenario).

Satisfaction: We infer the satisfaction of using Dominoes based on our survey and the terms chosen by participants

Fig. 8. Word cloud chose by participants using the Microsoft Reaction Card.

in the Microsoft Reaction Card[2]. All the questions were answered by the participants in a private room.

The exit interview contained Likert Scale questions about the ease of use of Dominoes (Table 3), where answers could range from strongly disagree (1) to strongly agree (5). The results show that all answers are positive – all above "agree". Of special note is Q5, which was about how useful was Dominoes in performing exploration of the Apache Derby project. It received the highest score (4.89). The lowest score (4.11) is for how easy it was to use the operations; two scored their answers as neutral (3). These participants were P5 and P7; both had performed explorations that were deviating from the results, and had incorrect answers. The difficulties in using the operations likely affected their perceptions (and their results). These responses reflect that participants were satisfied with using Dominoes. However, we recognize that participants could be influenced by the wording of our questions and the fact that they were responding to a tool evaluation. We discuss this further in Section 3.3.

We also asked the participants to rate each of the four scenarios on the ease of performing exploration. The average Likert scores for the scenarios are: 4.56 (S4), 4.44 (S2), 4.11 (S1), and 3.33 (S3). The median scores for all scenarios was 5, except for S3, which was 3. This was a scenario that two participants got wrong (P3 and P9), and took the longest time (avg. 10.56 min), which is a likely reason for the neutral answer from participants.

We present the results of the Microsoft Reaction Card through a word cloud depicting the frequency of terms selected by participants (see Fig. 8). The five most frequent words were: Efficient (6), Time-Saving (5), Fast (5), Innovative (5), and Useful (5). These results show that participants found Dominoes to be quick and efficient in allowing them to explore the project repository. We believe that the ability to pick a tile (project relationship), compose it with other tiles, and visually verify the answer helped participants quickly find answers. As P6 mentioned: *"I am impressed... Dominoes produces data very fast"*.

Participants also found Dominoes to be innovative (three additional participants chose "Novel" as a term), and wanted to learn and "play" with the tool. Participant P1 said: *"Congratulations. It [Dominoes] is an interesting tool. When it will be available for use?"*.

A few participants found the tool to be Complex (2) or Too-Technical (1). These were participants with the lowest experience with version control systems and had never analyzed data repositories before. This suggests that there is a learning curve associated with performing repository analysis using Dominoes, especially in framing the exploration as a composition of software relationships. A larger, longitudinal study is needed to understand the severity of the learning curve and its impact on project exploration.

TABLE 3
Participants' satisfaction.

Question	Avg. (med)
Q1. Easy to interact with Domino tiles	4.44 (5)
Q2. Easy to create and use derived tiles	4.56 (5)
Q2. Easy to use operations over tiles	4.11 (4)
Q4. Visualizations were useful to answer tasks	4.78 (5)
Q5. Dominoes helped in project exploration	4.89 (5)

Finding 3: Most participants addressed Dominoes as efficient, time-saving, fast, innovative, and useful tool in Microsoft Reaction Card.

3.2.2 What types of explorations do participants perform when using Dominoes?

To better understand the kinds of exploration behavior facilitated by Dominoes, we qualitatively analyzed participant actions. We frame our results by first analyzing the steps that participants took to perform their explorations. We then investigate the effects of the different navigation strategies employed by our participants.

Analytics Step

We categorized the different analytic steps taken by our participants into four main categories: *exploration, verification, adjustments, and organization*. Within each of these categories, we identify the specific exploration behavior. Table 4 details these steps further.

Based on recorded videos, each participant's actions were encoded using the negotiated agreement as described earlier. Fig. 9 presents a visual overview of the different analytic steps taken by participants per scenario (task). Each participant operation is represented as a cell (3 millimeters wide), with specific user actions (e.g., save, reuse, backtrack) annotated in the graph with an icon. The length of each bar represents the number of steps a participant performed during the study.

We color the segments of the graph based on their exploration type. When participants use tiles or operations that lead towards the correct solution (i.e., "Move Forward"), these operations are colored green. On the other hand, when participants use tiles or operations that do not lead to the right solution (i.e., "Deviate"), such operations are colored red. Finally, when participants are repeating previous actions, they are colored in yellow with horizontal lines in

2. Developed by and © 2002 Microsoft Corporation. All rights reserved.

TABLE 4
Categories used for classifying participants' steps.

Code		Description
Category	**Name**	
Exploration	Move Fwd.	Performs actions towards the correct solution
	Deviate	Performs actions that do not lead to a correct solution
	Repeat	Repeats past (wrong or right) actions
Verification	Checkpoint	Verifies if the actions thus far are correct
	Confirm	Checks another visualization to ensure the correctness of answer
Adjustment	Viz. Tweak	Adjusts some aspects of the visualization
	Tile Tweak	Adjusts some aspect of the tile
	Backtrack	Abandons current exploration path
Organization	Save	Saves derived tiles
	Reuse	Uses derived tiles

green or red, should these actions lead to the right solution or deviation, respectively.

For each scenario, we mark in the graph whether the participant got the correct answer (a tick mark) or failed to do so (a cross mark).

To contrast different types of user behavior let us consider participants P7 and P5. P7 had 4 years of industry and some experience in academic development (1 year). He had no experience in repository analysis. He performed the largest number of actions using the tool (283 actions and 48 tiles), and had many actions where he was deviating from the path (26 actions), as well as repeating right actions (9) and wrong actions (2). He frequently used the visualizations as check points to understand whether he was in the right path (or still on the wrong path). Based on these visualization checks, he backtracked his investigation (in both S1 and S4). He was relatively quick in the actions (see Table 2).

In contrast, P5 had 5 years of experience in academia, but 0.5 years in industry; and had experience in performing repository analysis. He got incorrect answers to S1 and S2. He also performed the fewest actions (92 steps and 25 tiles). He never recovered from the deviations (in both S1 and S2). He used the visual checkpoints for all tasks, but for S1 and S2 he did not realize that the data presented by the visualization was wrong.

Finally, we take the example of P2, the person with no experience in industry or repository analysis. We observe that P2 obtained correct answers for all the scenarios. He deviated in his explorations, but was able to recover by using checkpoints. We see that although P2 was a novice, he was able to easily grasp the working of Dominoes. In fact, he recommended additional functionality such as ability to order rows in the matrix visualization.

In summary, our observations and participant feedback indicate that participants investigated different exploration paths using the Dominoes interface, which was quick and easy to operate. In the following subsections, we discuss the participants exploration behaviors further.

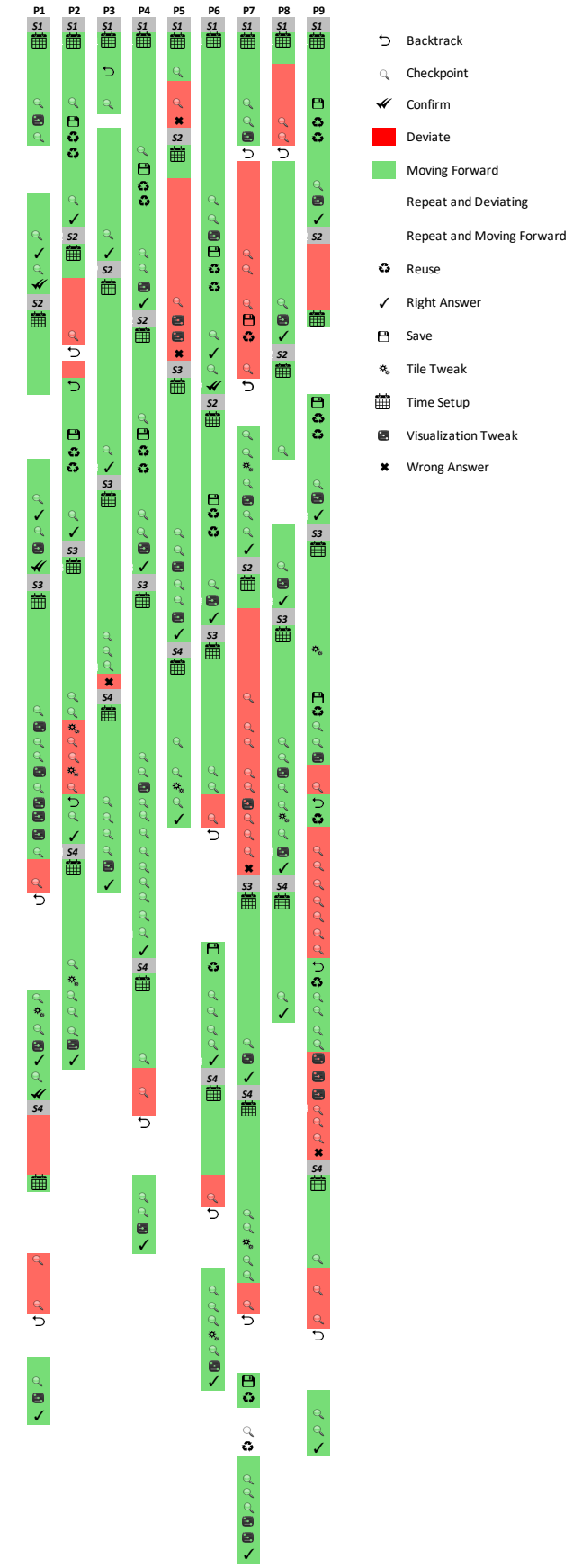


Fig. 9. Participants' action map.

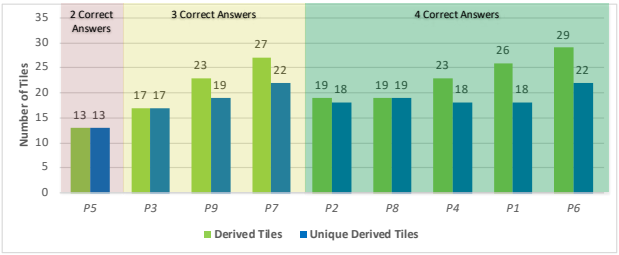


Fig. 10. Total of derived and unique tiles, and right answer.

Finding 4: Dominoes allowed participants with different backgrounds to perform different types of data exploration.

Alternative ideas

We found that trying multiple ways to analyze relationships was helpful in reaching the correct answer. This showcases the usefulness of allowing exploratory analysis, and multiple paths to solving a problem–key principles of Dominoes.

The number of unique and derived tiles used by participants and the correctness of their results indicate the usefulness of exploration. Fig. 10 shows the total number of unique derived tiles (blue) and derived tiles (green), sorted based on the total number of correct answers.

Unique tiles: Each unique derived tile represents different relationships among artifacts, and thus a different perspective of analysis. For instance, one can explore the knowledge of a developer by using the edited files, leading to tile [D|F] = [D|C] $\times$ [C|F] (coarse grain). The same insight can be obtained by inspecting the edited methods by a developer, leading to tile [D|M] = [D|C] $\times$ [C|M] (fine grain). Thus, if a participant was stuck when analyzing the [D|M] tile, she could derive the [D|F] tile and continue her analysis, as [D|M] can easily connect with $[Cl|M]^T \times [F|Cl]^T$.

Our participants performed such explorations reviewing data from alternative perspectives. In fact, some used exploration paths we had not envisioned. For instance, in S3 we expected participants to use the total amount of modifications per class ([$\sum$D|Cl]). However, P1 and P5 answered the scenario by using a [D|Cl] tile. They used a graph to visualize this tile and tweaked the threshold until just one edge remained. Similarly, P3 also used a graph to answer scenario 4. Participants using alternative ways of composing information, which in turn helped them arrive at the correct solution.

Finding 5: Dominoes allowed participants to generate alternative and unique solutions for the same problem.

Derived Tiles: The total number of derived tiles, which mirrors the number of unique tiles, also shows that when participants explored composing data in different ways they did better. At one end of the spectrum, we have P5 (leftmost participant in Fig. 10) with the minimum number of derived tiles (13) and just 2 right answers. On the other hand, P6 had the maximum number of derived tiles (29) and all four scenarios having correct answers.

When we unpack the exploration behavior of the participants in Fig. 10, we see that exploration (high number of derived tiles) helped participants (P4, P1, P6) get (all) correct answers. For example, P6, the participant with the most derived tiles, tinkered a lot. He combined tiles in different ways to visualize the relationship in the data from different perspectives. P6, said: *"What about if I combine these two pieces [tiles]? Maybe this path will lead to the answer"*.

Moreover, we find that exploration was helpful when users were struggling to find the right answer. For example, P5 barely performed any exploration when he was in the "wrong path" (red patches in Fig. 9) and ended up with incorrect solutions for scenarios 1 and 2. However, he spent more time in scenarios 3 and 4, performing checks and tweaking the visualization to arrive at the correct answers. On the other hand, P7 had the most number of deviating actions performed in the study (45 deviation steps in Fig. 11). However, these explorations allowed him to recover, and he was able to correctly answer three scenarios (S1, S3, S4). In scenario 2 (S2), where expertise was to be computed based on the number of issues fixed; P7 however used the number of file modifications. In the retrospective interview, we found that he misunderstood the scenario. He said: *"I did not notice it [that issue fix was required]"*.

P2 and P8 show a different pattern. Both have all correct answers with not much exploration. Our interviews revealed that both participants understood the concept of Dominoes very quickly. P8 had experience in repository analysis and was used to combining data based on the underlying relationships. P2 did not have experience in repository analysis, but had background in data provenance, allowing him to understand the concepts underling Dominoes quickly. He said: *"Dominoes pieces [tiles] are self-explanatory and it is possible to easily understand relationships"*.

Finding 6: Dominoes tends to fosters explorations; the majority of participants who explored more were likely to get correct answers.

Parallel Exploration: A key aspect of exploration is being able to compare and contrast alternatives [25]. Dominoes allows users to explore multiple data transformations at the same time. Users can simply keep multiple tiles in the canvas (where a tile itself can be the output of data transformations). They can also combine different transformation paths to get to a solution. Dominoes also allows parallel explorations of its visualizations.

We found multiple cases (P1 and P9) where participants performed parallel explorations during data manipulation. For example, P1 in S3 made a wrong operation (aggregation) on a tile, but she realized this after checking the visualization. She then resumed working on a previous exploration path (tile) that she had left on the canvas. She realized that her previous path could be re-used and said: *"I've made a mistake but I can continue the exploration from this path"*, and switched to that path to reach a correct solution.

TABLE 5
Checkpoints performed by participants in their explorations.

Part.	Keep moving fwd.	Keep deviating	Change from moving fwd. to deviating	Change from deviating to moving fwd.	Total
P1	14	1	1	2	18
P2	9	2	1	2	14
P3	9	0	1	0	10
P4	19	1	1	0	21
P5	7	2	1	0	10
P6	14	0	1	2	17
P7	14	11	1	3	29
P8	9	1	0	1	11
P9	9	10	2	3	24
Total	104 (67.53%)	28 (18.18%)	9 (5.84%)	13 (8.44%)	154

Finding 7: Dominoes supports parallel exploration, allowing users to compare and contrast alternatives.

Checking intermediary results

When exploring the different types of data transformations, participants often checked their intermediate results. These checkpoints were steps where participants visualized a data fragment (base or derived), annotated with a "magnifying glass" in Fig. 9. As an example, participant P6 first looked at the [D|C] tile to see the activities of a developer. He recognized that he was in the right path, but needed to see the activities at file-level, so he created the [D|F] tile and checked the data again.

We found that the ability to check intermediate steps was an important part of participants' exploratory data analysis. Participants used checkpoints to test if their "strategy" had worked. As an example, after being stuck for a while on S1, P6 created the right derived tile and checked the output using the visualization. He exclaimed: *"That's it! My rationale to answer the question is right and now I know how to proceed"*. On the other hand, P2 working on S2 performed a checkpoint and realized he was in the wrong direction, and said: *"It will not help me as it is a diagonal matrix. It does not make sense to be used"*.

Our study had about 17.11 checkpoints per participant. Table 5 presents the number of checkpoints performed by the participants, categorized based on the type of exploration path they were on. Columns 2 and 3 present instances where participants kept moving forward in the right path (shown as green in Fig. 9) or kept deviating (red in Fig. 9). Columns 4 and 5 represent checkpoints performed immediately before switching from deviation to moving forward (switch color from red to green or yellow in Fig. 11) and from moving forward to deviation (switch color from green or yellow to red in Fig. 9), respectively.

We see from Table 5 that of all the checkpoints performed, 67.53% (104 checkpoints) were done by participants to ensure that their explorations were on the right path. Participants were also performing checkpoints when they were stuck to see if what they had done was correct (18.18%, 28 checkpoints). In few cases (5.84%, 9 checkpoints), participants performed checkpoints and immediately changed from moving forward to deviate, misinterpreting the information presented to them.

Finally, in 8.44% (13 checkpoints) of the cases participants were able to get to the right path following a checkpoint. Fig. 9 shows that of the 16 instances where participants moved from deviating (red) to the right path (green or yellow), 13 of them were after a checkpoint. This means checkpoints helped participants know that they were using an incorrect strategy and were able to find a new strategy. In two of the remaining three instances, participants recovered by restarting their exploration.

Finding 8: Dominoes checkpoints were useful in preventing users from deviating (67.5% of the uses).

Backtracking

We define backtracking as an action by which participants change the direction of their exploration. This occurred when participants performed an undo action or started a new line of exploration by selecting a different set of tiles from the library. As an example, P1 when trying to find dependencies among developers first explored finding the methods changed by a developer (by [D|M] = [D|C] $\times$ [C|M]). But, she realized that this was not the right approach, so she backtracked, deleted the tiles on the editor canvas, and started exploring another path ([D|F] = [D|C] $\times$ [C|M] $\times$ [Cl|M]^T $\times$ [F|Cl]^T). She said: *"That is not what I want. It is wrong. I need another dominoes piece [tile]"*

Fig. 11 shows the number of backtracks performed by participants. We follow the same color coding as in Fig. 9, where green is moving forward, yellow is repeat, and red is deviating. We see that every participant, except P5, backtracked at least once. We also see that scenarios 3 and 4, which were open-ended, needed a lot more backtracking than the first two scenarios.

In 14 out of the 15 backtracking instances, participants reverted from an incorrect path to a correct one (red to green or yellow). These backtracks were triggered by checkpoints: 13 out of the 15 backtracks. For example, after a checkpoint in S3, participant P6 realized that he was in the wrong direction. He backtracked after saying: *"That is not what I want. It is wrong. I need another dominoes piece [tile]"*.

Finally, we find backtracking to be helpful in arriving at the right solution. When we evaluate the incorrect answers (5 cases), we find that in four out of the five cases participants did not backtrack (P3 (S3), P5 (S1 and S2), P7 (S2)).

Finding 9: Dominoes backtracking mechanisms allowed the participants to revert incorrect exploration paths in 93% of the times – 80% of the users that got incorrect answers did not backtrack.

3.3 Limitations

Here we discuss the limitations of our study design and of the tool itself.

Study design Limitations: This paper reports an exploratory study of how participants use Dominoes when

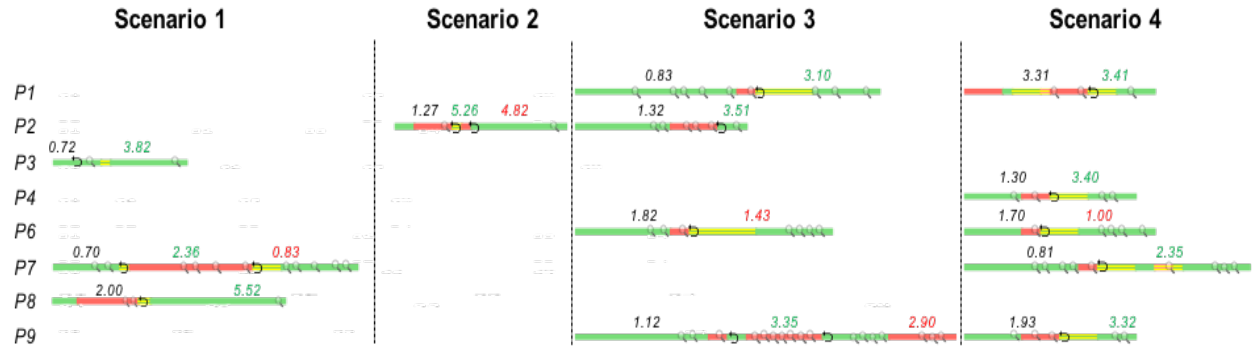


Fig. 11. Backtracking and checkpoint. The numbers indicates the rate of pieces / minutes used by the participant.

performing a set of data exploration tasks. Such an exploratory study was needed to first understand whether users understand the Dominoes tile metaphor, and how they perform explorations. Further studies are needed to evaluate Dominoes in comparison to other data exploration tools (e.g., Information Fragments [3]).

We stopped recruitment after 9 participants since we found saturation in participant behaviors, participants were repeating the exploration paths and the mistakes. Like in any participant feedback, our exit interviews may be subjective and include desirability bias, we aimed overcome these by basing the results mainly on the qualitative analysis of user actions (and not just their responses to the questionnaire) and maintained rigor in the analysis through negotiated agreement. The inherent nature of an observational think-aloud study might create additional cognitive load in participants needing to think aloud [26]. Such limitation can be removed in future controlled quantitative studies.

Finally, we did not counterbalance the four tasks (scenarios). This was a conscious choice. The tasks were designed to be of increasing complexity, and it was important that participants first start their exploration by answering simpler questions before attempting open-ended explorations.

Tool Limitations. Currently, Dominoes provides seven basic tiles that link project entities, such as commits and its changes (method and file level), developer, and related issues, as well as containment relationships of methods into classes, files, and packages. Other kinds of data, such as communication, team organization, and code review information, are not yet available and would be interesting (as noted in feedback from P6). The Dominoes architecture is designed to makes it easy to add tiles for such information, by creating a wrapper for the repository containing this data. Some interactions, such as "drag and drop", were different than what participants (P2, P9) expected. For example, double clicking a tile in the library brings it onto the canvas, and tiles are rearranged in the library window by dragging; P2 and P9 expected to drag a tile onto the canvas.

Participants suggested enhancements to the visualizations that would make it easier to understand or retrieve the data. Some of the suggestions included a search (P1, P6) and reordering (P2) functionality for the matrix visualization, which can become quite large for tiles (e.g., scenario 3 led to an 8×300 matrix). Another recommendation was to provide tool tips or to annotate the bars (in the bar chart) with the data, since it was difficult to accurately pin point the data in a large visualization (lots of bars).

Another feature that could have benefitted participants was if the visualizations automatically updated themselves reflecting changes to the derived tiles. Currently, Dominoes requires explicitly opening a visualization by right clicking on a tile. This allows multiple visualizations to exist that can be compared side-by-side, but sometimes, participants created too many visualization and got confused. Additionally, derived tiles do not retain their history when saved as a tile (in the library), which might be useful to participants for future explorations (P7).

Finally there are some questions that are ill-suited to the matrix approach employed by Dominoes. One example of such questions are the ones that involve navigating in the transitive closure of data. For instance, identifying all methods that could be directly or indirectly affected by a bug in a specific method cannot be answered using the current matrix manipulation logic of Dominoes.

4 Member Checking: Industry Professionals' Interview

As an additional investigation of how Dominoes can help professional software developers, we interviewed five software professionals. The goal of this study was to determine the kinds of questions that these software professionals had to answer in their every day work and how Dominoes could help answer these questions.

Participants: We recruited these software professionals by emailing eleven developers, who had collaborated with one of the co-authors in industrial research projects. We could schedule five out of the eleven participants in a 1-week period. Table 6 shows participants' details. Participants (numbered, P11 to P15) all had significant experience in software development (from 6 to 30 years), had different occupations (e.g., software development, software engineering, project leader, and project manager), and worked with different technologies. All participants had performed some form of exploratory analysis as part of their work.

Method: The interviews were about one hour long. We started the session by providing some explanation about our research and obtaining their consent to record the interview. We then explained the purpose of Dominoes and the concepts behind matrix operations; these explanations

TABLE 6
Participants' profile.

Part.	Experience	Occupation	Exploratory Analysis	Technologies Used
P11	10 years	Project leader	Identify developers that changed problematic code and identify expert developers in the code	Git, Subversion, Track
P12	30 years	Soft. eng.	Identify students' profile for merchandising	Oracle
P13	13 years	Soft. eng.	Identify releases and release related information in a project	Git, Java, Python
P14	21 years	Project manager	Identify how developers are organized among repositories	Java, .Net, Clearcase, Subversion, Git
P15	6 years	Soft. dev.	Evaluate students' profile	Subversion, Git

were part of a script that the first author read out to ensure consistency across participants. Next, we gave a hands on demonstration of Dominoes by walking the participants through scenarios 3 and 4, described in Section 3.1.1. During this time, participants were encouraged to ask any questions they might have had. After this demonstration, we allowed participants to "play" with Dominoes if they desired, otherwise, we asked them to provide us with examples of how they envisioned using Dominoes in their own projects along with concrete examples of such usage.

Analysis: The interviews helped uncover 16 distinct questions that participants said they would use Dominoes to answer. We categorized these questions into six broad categories based on the intent of the question. The first author analyzed the interview transcripts to create these categories. All the authors then met to discuss these categories and the placement of the questions in these categories. Through one round of negotiated agreement (two categories were similar and were combined into one and another was dropped), we arrived at the following set of four categories:

- **Locate who:** identify developer(s) with expertise on different pieces of the code (at different levels of granularity).
- **Locate what:** identify details about particular changes or issues in the project.
- **Productivity:** identify the amount of work performed by individuals.
- **Team organization:** identify optimum work responsibilities.

Results: Table 7 presents the questions that our participants said they would ask Dominoes when exploring their project. Four out of five participants (P11-P14) mentioned they would appreciate using Dominoes to seamlessly explore different types of data. A common theme of questions was to identify which parts of the code were involved with issues (Q7, Q9). Others included questions about what was changed in the project (Q6-Q9); sometimes to find out which parts of the code was likely to be buggy (Q7, Q9) or for help in refactoring (Q8). P15 mentions how Dominoes can be helpful by recalling an example: "*...to identify ancient methods to perform refactoring. A file can be changed a lot of times but it does not guarantee all methods were changed... I worked in a three-year-long project. In a few months before delivering it, I asked all developers to check all the methods they created but never revised... it was not possible to do so... we know some were missed*".

Participants wanted to know who had changed what to figure out task assignments (Q1-Q4) or from whom to seek help (Q15). Some participants wanted to use Dominoes to help with their team management: decide which type of tasks a developer should have (Q16), facilitate communication in the team (Q15), answer productivity questions of the team (Q10-Q13), or organize the team to ensure there is redundancy in expertise on complex parts of the codebase (Q14): P12 said: "*...in case just one developer is working on a complex system, I would assign someone to work with him to mitigate the risk*".

The ability to investigate data at a finer grained level was appreciated by participants (Q1, Q6). P11 said: "*allows fine granularity... it becomes much easier to identify the exact methods that have been changed by a developer*".

Additionally, participants mentioned that Dominoes would be helpful in wrangling large amounts of data (P11, P13). They appreciated the fact that by using Dominoes they would not have to export the data into multiple tools or data format (P12, P14). Participants felt Dominoes made project exploration much less complex than what they would need to do in Git (P12, P15), and Dominoes made data exploration possible for both novice and experts (P14). Two participants (P11, P15) felt that automation could help further (P15): "*it would be nice to have a feature where users inform the relationship they desire and Dominoes produces all possible combinations to reach this target relationship*". In Section 5, we explore how such an automation feature could work in Dominoes.

Participants liked the ability to explore relationships across different repositories (P11-P14). This ability to ask questions across several repositories allowed them to chain questions as they understood more about the project (P13): "*...Explore potential developer who creates a bug... combining commit, issues, and classes tiles I can identify the classes that have a high amount of modifications for solving a bug. From this, I can then check the developers that changed this class in the recent past... then I can infer that the one that most changed it introduced the bug... I can combine pieces to explore developers that are adding more bugs in the code. Using the information of commits, I can produce a metric of effectiveness about the changes performed by a developer*".

Additionally, participants appreciated the ability to explore the project data via visualizations (P11-P14). P12 said: "*biggest problem when using tools for selecting info in a database is related to visualization. All the time, I needed to export the resulting data in order to visualize the information*". All participants (P11-P15) agreed that using Git to explore their project is difficult, requiring them to know what information they want to process, export/extract the information, and run SQL queries. P14 said: "*...difficult as the data produced by Git should be imported into another tool in order to understand how these information connect to each other. Besides that, a query language would be necessary, which increases even more the complexity of the analysis*". In fact, P13 had built a visualization

TABLE 7
Participants' questions.

Category	Question #	Question Type	Participant P11	P12	P13	P14	P15
Locate Who	1	Who has changed what at a finer grained lens (method, complex files)?	✓				✓
	2	Who has (historical) expertise on parts of code?				✓	
	3	Who should edit a (given) code part?		✓			
	4	Who should be assigned to an issue?				✓	
	5	Who introduced a bug?			✓	✓	
Locate What	6	What has changed at a finer grained (method, parts of code base, complex files, across repo)?	✓		✓	✓	✓
	7	Which files has had a lot of change (help find buggy places)?			✓		
	8	Which files have become stale (help refactor)?					✓
	9	Which bug is related to which parts of code?	✓		✓	✓	
Productivity	10	Who is active now?		✓			
	11	Who has done what (amount of commits, complex code)?	✓	✓			
	12	How long it takes someone to finish task?	✓				
	13	Who creates buggy code?			✓		
Team organization	14	How well is expertise balanced to mitigate risk of lost expertise because of turnover?	✓	✓			
	15	Who has expertise to help someone/ facilitate communication?		✓			
	16	Who is right to fix bugs vs. make new feature implementation?			✓		

TABLE 8
Dominoes answers for the questions in Table 7.

Question #	Answer
#1	$[D\|M] = [D\|C] \times [C\|M]$
#2	Usage of 3D Dominoes pieces [16]
#3	$[D\|Cl] = [D\|C] \times [C\|M] \times [Cl\|M]^T$
#4	Same as #3
#5	$[D\|F] = [D\|C] \times [C\|F]$ over a time frame
#6	$[C\|F]$ over a time frame
#7	$[\sum C\|F^{\downarrow}]$
#8	$[\sum C\|F^{\uparrow}]$
#9	Not yet implemented
#10	$[\sum D\|C^{\downarrow}]$ over a time frame
#11	Same as #10
#12	$[D\|I] = [D\|C] \times [I\|C]^T$
#13	Same as #3
#14, #15	Same as #2
#16	Same as #1

to simplify his project exploration. The tool showed which developers had changed which files and which files were changed together.

Finally, we investigated possible answers to these questions using Dominoes. Most of these questions are open-ended, allowing a multitude of solutions. Table 8 shows one possibility for answering each question in Dominoes.

In Table 7, question #1 asks for an overview of changes as well as who performed it. The answer in Table 8 is at a fine grain (method level). Questions #3 and #4 ask for advice regarding task distribution, which can be answered by determining the most appropriate developer, such as the one who most changed the specified part of the code. Question #2 is more specific, as it requires a historical analysis that can be done by Dominoes API [16] (not available in GUI). For question #5, one possible answer could be selecting the last five days, for instance, and then retrieving the person who edited the problematic artifact.

Question #6 is mostly like question #1 but viewed from the artifact perspective in order to identify changes in a selected time frame. On the other hand, questions #7 and #8 are expected to indicate artifacts with more and fewer modifications, respectively. It can be extracted by counting the number of commits and sorting the result in descending order, for the former, or ascending order, for the latter. Question #9 cannot be answered by Dominoes in its current version, but a transitive closure operation can handle these questions, as stated by Kim et al. [27]. We are working on a new version of Dominoes with such an operation.

Related to productivity, answering question #10 can be done by counting the number of commits performed by each developer and sorting the result. By scoping the time frame, it is possible to select the most recent active developers for this analysis. Question #11 follows the same answer of #10, but just counting commits may not be appropriate, as some commits are more complex than others. Question #12 can only be answered by counting the number of commits related to a task, while question #13 can use the number of commits performed on a target artifact over a time frame for identifying the developer that most changed it.

Questions #14 and #15 can be answered by using 3D Dominoes pieces [16] in order to evaluate the expertise variation along the time. Finally, question #16 requires some sort of interpretation. To fix a bug, we can check the developer that has performed most changes in the piece of code where the bug was found, for instance. For implementing new features, one possibility is to identify the developer with consistent expertise in this part of code by using 3D Dominoes pieces.

5 Exploration vs. Recommendation

Although we explicitly designed Dominoes to support exploratory analyses over software repositories, we believe that users could benefit from additional recommendation support during exploration (see Section 4). For instance,

Dominoes could ask the users for expected relationship endpoints and search for all tiles combinations that respect such endpoints. Let us suppose that a user wants to know all methods that were changed together with other methods. The answer clearly has method in both endpoints: [M|M]. A recommendation system could list possible tile combinations that would satisfy this query, such as: $[Cl|M]^T \times [Cl|M]$, $[C|M]^T \times [C|M]$, etc. Then, the user would need to choose the most appropriate recommendation, which is the second in this list. This kind of support could be especially useful for newcomers, to understand possible combinations of tiles and speed-up the exploration process.

To assess the feasibility and utility of such automated support, we implemented a Jupyter notebook[3] to simulate the execution of the operations over tiles. This Jupyter notebook receives the expected endpoints as input. Then, it combines the existing tiles to create derived tiles, using both transposition and multiplication, recursively. After that, it filters the results by the desired tile endpoints and sorts the results in ascending order of the number of operations used. We used this Jupyter notebook to simulate all four scenarios discussed in Section 3.1.1. For each scenario, we provided the expected endpoints and collected the possible recommendations. Table 9 shows the results.

The first column of Table 9 shows the scenario id. The second column shows the expected endpoints, extracted from the scenario description in Section 3.1.1. The third column shows the number of necessary operations to reach the correct answer. This number was derived from the correct answer shown in Section 3.1.1. For instance, the correct answer for Scenario 1 is [D|D] = $[D|C] \times [C|M] \times ([D|C] \times [C|M])^T$, which requires one multiplication for the first two tiles (X = $[D|C] \times [C|M]$), one multiplication for the last two tiles (Y = $[D|C] \times [C|M]$), one transposition ($Z = Y^T$), and one final multiplication ($X \times Z$). The fourth column shows the total number of possible answers after applying all combinations of operations listed in the third column. Please notice that not all theoretical combinations are possible in practice because, while transposition can always occur, multiplication can only occur when both tiles have endpoints on different sides in common. For instance, the two operations (multiplication and transposition) applied over the seven basic tiles would produce just 21 combinations: the seven original basic tiles, the seven tiles derived from the transposition of the basic tiles, and the 7 tiles derived from the multiplication of basic tiles ([F|Cl] × [Cl|M], [C|F] × [F|Cl], [P|F] × [F|Cl], [D|C] × [C|F], [I|C] × [C|F], [D|C] × [C|M], [I|C] × [C|M]). When applying the two operations in an iterative manner over the obtained tiles, we reach 53, 117, 231, and 450 combinations, respectively for 2, 3, 4, and 5 operations. Finally, the fifth column indicates the position in which the correct answer appears among the combinations listed in the fourth column after sorting all combinations with appropriated endpoints (second column) in ascending order of the number of operations. For instance, from the 231 possible combinations for Scenario 1, after filtering the ones with [D|D] endpoints and sorting the results in ascending order, the correct answer appears in the third position.

As shown in Section 3.1.1, the correct answer for Scenario 1 can be reached after applying 4 operations. The total number of possible tiles after applying 4 operations is 231. However, when we just consider tiles that start with 'D' and end with 'D', the correct answer can be found in the third position: $[D|C] \times [D|C]^T$, $[D|C] \times [C|F] \times ([D|C] \times [C|F])^T$, $\mathbf{[D|C]} \times \mathbf{[C|M]} \times \mathbf{([D|C] \times [C|M])}^T$. For Scenario 2, out of 450 tiles generated after 5 operations, the correct answer is listed in the fifth position, should the user provide a template starting with 'D' and ending with 'D': $[D|C] \times [D|C]^T$, $[D|C] \times [C|F] \times ([D|C] \times [C|F])^T$, $[D|C] \times [C|M] \times ([D|C] \times [C|M])^T$, $[D|C] \times [D|C]^T \times [D|C] \times [D|C]^T$, $\mathbf{[D|C]} \times \mathbf{[I|C]}^T \times \mathbf{[I|C]} \times \mathbf{[D|C]}^T$. For Scenarios 3 and 4, the answers, respectively with 3 and 2 operations, would appear in second out of 117 and 54 possible tiles, should the user inform the expected endpoints: $[D|C] \times [C|F] \times [F|Cl]$, $\mathbf{[D|C]} \times \mathbf{[C|M]} \times \mathbf{[Cl|M]}^T$ for Scenario 3 and $[C|F] \times [F|Cl]$, $\mathbf{[C|M]} \times \mathbf{[Cl|M]}^T$ for Scenario 4.

TABLE 9
Tiles recommendation statistics.

Scenario	Endpoints	# Operations	# Combinations	Rank
1	[D|D]	4	231	3
2	[D|D]	5	450	5
3	[D|Cl]	3	117	2
4	[C|Cl]	2	54	2

Although promising, a feature like this is still limited when composing these tiles. The user would still need to: (1) know the expected endpoints of the resulting tile, (2) choose the correct tile from the list of possible solutions, (3) choose the correct visualisation to interpret the results, and (4) search for the correct answer in the visualization. Depending on the results, the user may need to go back to previous steps and start over. For instance, Scenario 4 also demands an aggregation over a specific column. Choosing the correct tile suggested by such a recommendation feature would be just a starting point for the exploration. Consequently, we see such a recommendation feature as complementary to the exploration features that are already in-place in Dominoes.

6 Related Work

While working on a software project, developers tend to ask a variety of questions, such as "where is this method called?" [3] or "who modified this class the most?" [28]. Some of these questions are easy to answer, as they target individual information and have little or no ambiguity.

On the other hand, some questions such as "what artifacts being changed by my co-workers may affect my work?" or "how can I identify the developers who should be allocated to a given task?" require more effort to be mined and answered. In order to answer these questions, it is necessary to link together different pieces of information, potentially coming from different repositories [29]. In this case, it may be necessary to check an issue tracker in order to verify in which parts of the code someone is working on. Additionally, it may be necessary to check over communications from these coworkers.

Sillito et al. [28] conducted two qualitative studies about developers performing changes tasks from medium to large

3. http://bit.ly/2UYV2EF

projects. One of the studies involved newcomers working over these tasks while the other involves experienced programmers. The main focus of their work was to measure what information developers need while performing the task and how they achieve this information. The final result is the categorization of 44 kinds of different questions; the vast majority of which involves relationships with other entities in the project.

In the same way, Ko et al. [30] conducted a study with seventeen developers in order to analyze the information they sought, the data source used by them, and most importantly, the barriers that prevent these information to be acquired. Interestingly, most of the questions types are related to awareness about artifacts and coworkers. Besides that, the cost of testing hypothesis and the risk of a false hypothesis often prevent developers from finding their answer. In this case, tools that allow fast and easy data manipulation without requiring so many technical aspects can reduce these barriers. Dominos goes in this direction by providing mechanisms to compose and explore these relationships in a fast way by employing GPU.

Cataldo et al. [1] stands out as they use matrices to process dependencies among developers based on dependencies among artifacts. In their approach, both structural and logical dependencies become Task Dependency (T_D) matrices, and change requests, associating developers to artifacts, becoming Task Assignment (T_A) matrix. These matrices are used in an equation that indicates coordination requirements $T_A \times T_D \times T_D^T$. Our approach generalizes this idea by allowing different kinds of exploration over matrices. Our identification of relationships allows for combining support, confidence, and lift, to compose the dependency matrix depending on the required analysis. Moreover, we shift the perspective from a predefined and offline processing to an exploratory and online (interactive) processing using the Dominoes tile metaphor and GPU processing.

A majority of current Exploratory Data Analysis tools support exploration via visualizations and/or queries on a predetermined set of project relationships and predetermined type of data repositories. Voinea and Talea [31] present a framework capable of mining software repositories at coarse grain and presenting an analysis of this data through different types of visualizations. Most of these visualizations provide functionalities to sort a set of (predetermined) characteristics (such as developer's-id or file size) in order to infer information. In the same way, Metrics Viewer [32] provides visualization over repository, such as source code changes, without allowing manipulation of these data. On the other hand, tools such as Tukan [33], CollabVS [34], and Palantír [35] perform code analysis in order to identify dependencies, allowing either predefined questions or limiting the amount of artifacts that can be analyzed in a reasonable amount of time. In the case of Tesseract [11], for example, the available relationships are preprocessed and the matrices are fixed at a coarse grain level ([file|file], [file|developer], [file|bug], [bug|developer]), imposing a subset of the data that can be explored. Tableau [36] is a drag-and-drop application that generates queries from different pre-selected data sources, allowing visualization using different built-in chart types. However, to do this the user needs to use VizQL [37]–a structured query language–for selecting the data to be combined. Dominoes on the other hand does not require knowledge of any particular SQL query language.

Codebook [38] is a framework that mines relationships from software repositories, building a graph of developers and artifacts. For instance, such graph can encode the developers that are the authors of commits and the files changed by commits. Expert users can build applications that instantiate Codebook to run specific regular expressions over the graph. End users can use such applications to query Codebook through the predefined regular expressions according to specific parameters (e.g., the name of a developer or a file they want to search for). Dominoes distinguishes from Codebook in two primary ways. First, Codebook is a framework conceived to be instantiated into applications (e.g., Hoozizat [38], WhoseisThat [39], and Deep Intellisense [40]) that focus on providing predefined queries to end users. Each of these applications focus on specific types of project information. Dominoes, on the other hand, leverages an intuitive and a fast interface that allows end users to explore different project relationships on the fly. Second, although Codebook was designed to be scalable, its dependence on compilation of the regular expressions (queries) into state machines requires almost an hour–a high start-up cost for exploratory analysis. Dominoes multiplies matrices cells in parallel over GPU in few seconds for most cases [14].

Information Fragments [3] is a tool that supports querying information obtained from different sources (e.g., source code, team, etc.), and is the closest in functionality to Dominoes. Each information source, called information fragment, is represented as a graph of relationships among information from the respective domain (e.g., method calls, leadership, etc.). End users can compose new information fragments (e.g., code ownership) by connecting existing base fragments. These composed informations are in fact new edges connecting nodes of previously disconnected graphs. The composition operation relies on id matching to connect nodes from different graphs. Text matching was also proposed, but not implemented in the tool yet. Differently from Information Fragments, that requires composing queries from scratch, Dominoes allows saving and reusing composed tiles. This not only enhances the list of available tiles for other users but also speeds up the answer of new questions. Moreover, Information Fragments uses simple hierarchical views to show the answer of the queries, while Dominoes provide multiple views for presenting intermediate and final data. Finally, Dominoes was specifically designed for supporting exploratory analysis of large repositories by processing relationships as matrices in GPU. The Information Fragments paper does not provide performance evaluation; neither does it provide characterization of the size of the project used in the evaluation. We plan to contact the authors to get access to their tool to perform such a comparative evaluation in the future.

7 Conclusion and Future Work

In this paper we presented Dominoes, a tool that allows users to perform interactive exploratory data analysis by dragging, dropping, and connecting Dominoes tiles, which

represent project entities. Dominoes allows seamless hands-on data exploration of a software project without requiring the user to write scripts or queries, or be limited by predefined relationships. Users can explore different project relationships, backtrack, and save their exploration paths. The key design decisions behind Dominoes are: (1) the use of a high-level metaphor for abstracting project relationships (dominoes tiles), (2) an intuitive mechanism for deriving additional project relationships (composing dominoes tiles), and (3) a fast mechanism for performing the matrix transformations under the hood (via GPU).

We evaluated Dominoes' usability by having nine participants complete a set of four tasks, where they explored the Apache Derby project by creating new derived tiles, exploring different project relationships, and investigating new perspectives when visualizing these relationships. Participants were successful in 86.11% of their explorations, and were able to learn the tool quickly. While our study recruited participants with software engineering experience, we did not control for repository analysis experience; Only three of them had such experience (P5, P8, and P9). When comparing these three participants with the others they took less time on average (6.34 vs 7.70 minutes). However, they had more incorrect answers (25% vs 8%). Further analysis revealed, on average, they did less checkpoints (15.00 vs 18.16) and performed less backtracking (1.33 vs 2.00). So, while they had speed this did not convert to more accurate answers. Further studies with different types of target users and other types of software development projects are needed to generalize our results.

We also presented Dominoes to five professionals and collected real-world questions that Dominoes would be able to support answering. Finally, we experimented with a promising recommendation feature for Dominoes. A natural extension of these evaluations is to release Dominoes to the community in order to get feedback on tool enhancements and a deeper understanding about the exploration behavior of people exploring their own projects. Finally, the Dominoes approach can be applied to other domains by allowing users to extract and cross-link different data elements, which can then be used to create the Dominoes tiles (matrices).

ACKNOWLEDGMENTS

We thank the participants of our study. This work is partially funded by CAPES, CNPq, FAPERJ, and NSF grants: 1815486 and 1560526.

REFERENCES

[1] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley, "Identification of coordination requirements: implications for the Design of collaboration and awareness tools," ser. CSCW '06. New York, NY, USA: ACM, 2006, pp. 353–362.

[2] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse, "How Developers Drive Software Evolution," vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2005, pp. 113–122.

[3] T. Fritz and G. C. Murphy, "Using Information Fragments to Answer the Questions Developers Ask," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 175–184.

[4] J. D. Herbsleb and R. E. Grinter, "Splitting the Organization and Integrating the Code: Conway's Law Revisited," ser. ICSE '99. New York, NY, USA: ACM, 1999, pp. 85–95.

[5] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity," ser. ESEM '08. New York, NY, USA: ACM, 2008, pp. 2–11.

[6] S. Mcintosh, B. Adams, M. Nagappan, and A. E. Hassan, "Mining Co-change Information to Understand When Build Changes Are Necessary," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, sep 2014, pp. 241–250.

[7] M. Kim, T. Zimmermann, R. DeLine, and A. Begel, "Data scientists in software teams: State of the art and challenges," *IEEE Transactions on Software Engineering*, vol. 44, no. 11, pp. 1024–1038, Nov 2018.

[8] A. J. Ko and B. A. Myers, "Debugging reinvented: Asking and answering why and why not questions about program behavior," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 301–310.

[9] T. Green, "Programming languages as information structures," pp. 118–137, 1990.

[10] S. Minto and G. C. Murphy, "Recommending Emergent Teams," in *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*. IEEE, may 2007, pp. 5–5.

[11] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," ser. ICSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 23–33.

[12] C. Kiefer, A. Bernstein, and J. Tappolet, "Mining Software Repositories with iSPAROL and a Software Evolution Ontology," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 10—-.

[13] S. Paul and A. Prakash, "A Query Algebra for Program Databases," *IEEE Trans. Softw. Eng.*, vol. 22, no. 3, pp. 202–217, 1996.

[14] J. R. da Silva Junior, E. Clua, L. Murta, and A. Sarma, "Exploratory data analysis of software repositories via gpu processing," in *The International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2014, pp. 495–500.

[15] ——, "Multi-Perspective Exploratory Analysis of Software Development Data," *International Journal of Software Engineering and Knowledge Engineering*, vol. 25, no. 01, pp. 51–68, 2015.

[16] ——, "Niche vs. breadth: Calculating expertise over time through a fine-grained analysis," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 409–418.

[17] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10.

[18] S. Breß, M. Heimel, N. Siegmund, L. Bellatreche, and G. Saake, *GPU-Accelerated Database Systems: Survey and Open Challenges*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1–35.

[19] D. Steward, "The design structure system: A method for managing the design of complex systems," *IEEE Transactions on Engineering Management*, vol. EM-28, no. 3, pp. 71–74, 1981.

[20] P. Prabhu, T. B. Jablin, A. Raman, Y. Zhang, J. Huang, H. Kim, N. P. Johnson, F. Liu, S. Ghosh, S. Beard, and Others, "A survey of the practice of computational science." New York, NY, USA: ACM, 2011, p. 19.

[21] M. E. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[22] A. J. Ko, B. A. Myers, and H. H. Aung, "Six Learning Barriers in End-User Programming Systems," in *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, ser. VLHCC '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 199–206.

[23] S. K. Kuttal, A. Sarma, and G. Rothermel, "On the Benefits of Providing Versioning Support for End Users: An Empirical Study," *ACM Trans. Comput.-Hum. Interact.*, vol. 21, no. 2, pp. 9:1—-9:43, feb 2014.

[24] H. Chan, K. Siau, and K.-K. Wei, "The Effect of Data Model, System and Task Characteristics on User Query Performance: An Empirical Study," *SIGMIS Database*, vol. 29, no. 1, pp. 31–49, 1997.

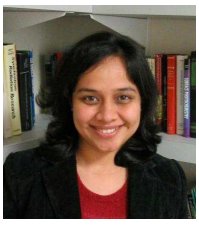
[25] S. Srinivasa Ragavan, S. K. Kuttal, C. Hill, A. Sarma, D. Piorkowski, and M. Burnett, "Foraging Among an Overabundance of Similar Variants," in *Proceedings of the 2016 CHI Conference on*

Human Factors in Computing Systems, ser. CHI '16. New York, NY, USA: ACM, 2016, pp. 3509–3521.
[26] S. Xu and V. Rajlich, "Dialog-based protocol: an empirical research method for cognitive activities in software engineering," in *2005 International Symposium on Empirical Software Engineering, 2005.*, Nov 2005.
[27] S. Kim, T. Zimmermann, K. Pan, and E. J. Jr. Whitehead, "Automatic identification of bug-introducing changes," in *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*, Sep. 2006, pp. 81–90.
[28] J. Sillito, G. C. Murphy, and K. De Volder, "Questions Programmers Ask During Software Evolution Tasks," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. SIGSOFT '06/FSE-14. New York, NY, USA: ACM, 2006, pp. 23–34.
[29] A. E. Hassan, "The road ahead for Mining Software Repositories," in *2008 Frontiers of Software Maintenance*. IEEE, sep 2008, pp. 48–57.
[30] A. J. Ko, R. DeLine, and G. Venolia, "Information Needs in Collocated Software Development Teams," in *Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 344–353.
[31] L. Voinea and A. Telea, "Visual querying and analysis of large software repositories," *Empirical Software Engineering*, vol. 14, no. 3, pp. 316–340, jun 2009.
[32] S. Yasutaka, S. Matsumoto, S. Saiki, and M. Nakamura, "Visualizing Software Metrics with Service-Oriented Mining Software Repository for Reviewing Personal Process," in *2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. IEEE, jul 2013, pp. 549–554.
[33] T. Schümmer and J. M. Haake, "Supporting distributed software development by modes of collaboration," in *Proceedings of the Seventh European Conference on Computer Supported Cooperative Work*, W. Prinz, M. Jarke, Y. Rogers, K. Schmidt, and V. Wulf, Eds. Dordrecht: Springer Netherlands, 2001, ch. Supporting, pp. 79–98.
[34] P. Dewan and R. Hegde, "Semi-synchronous conflict detection and resolution in asynchronous software development," *European Conference on Computer-Supported Cooperative Work (ECSCW)*, pp. 159–178, 2007.
[35] A. Sarma and A. van der Hoek, "Palantir: coordinating distributed workspaces," in *Proceedings 26th Annual International Computer Software and Applications*, Aug 2002, pp. 1093–1097.
[36] P. Terlecki, F. Xu, M. Shaw, V. Kim, and R. Wesley, "On improving user response times in tableau," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 1695–1706.
[37] K. Morton, R. Bunker, J. Mackinlay, R. Morton, and C. Stolte, "Dynamic workload driven data integration in tableau," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 807–816.
[38] A. Begel, Y. P. Khoo, and T. Zimmermann, "Codebook: Discovering and Exploiting Relationships in Software Repositories," ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 125–134.
[39] A. Begel, K. Y. Phang, and T. Zimmermann, "WhoselsThat: Finding Software Engineers with Codebook," ser. FSE '10. New York, NY, USA: ACM, 2010, pp. 381–382.
[40] R. Holmes and A. Begel, "Deep Intellisense: A Tool for Rehydrating Evaporated Information," in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, ser. MSR '08. New York, NY, USA: ACM, 2008, pp. 23–26.

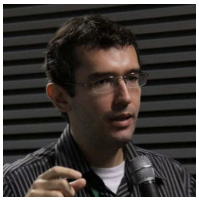
Jose Ricardo da Silva Junior is a professor at Instituto Federal do Rio de Janeiro. He received his Ph.D. in computer science from the Universidade Federal Fluminense (UFF) in 2015 and a B.S. in computer science from the Universidade Estacio de Sa in 2005. His research interest includes Digital Games, Virtual Reality, fluid simulation in real time, and repository data analysis. During his fellowship at the University of Nebraska, he started to develop Dominoes with Prof. Anita Sarma. Jose has experience in computer science with emphasis in GPGPUs, HPC, simulation, and optimization.

Daniel Prett Campagna is a computer science graduate student at Universidade Federal Fluminense (UFF). He holds a B.S. (2019) in computer science from UFF and an information technology technician degree (2013) from Instituto Federal do Esprito Santo (IFES). He started his research career in 2014, working with data analytics over configuration management repositories. Afterwards, he received a CNPq scientific initiation grant (2016) to work with data analytics over undergraduate student transcripts. His research interests area is in systems and information engineering.

Esteban W. Gonzalez Clua is professor at Universidade Federal Fluminense and coordinator of UFF Medialab. He was nominated Young Scientist of the State of Rio in 2009 and 2013 and in 2015 received the nomination of CUDA Fellow. His main research and development area are Digital Games, Virtual Reality, GPUs and Visualization. He is today the coordinator of the NVIDIA Center of Excellence, that is located at the CS Institute of UFF.



Anita Sarma is an Associate Professor in the School of Electrical Engineering and Computer Science, at Oregon State University. She holds a Ph.D. degree in Computer Science from the University of California, Irvine. Her research interests lie primarily in the intersection of software engineering and computer-supported cooperative work, focusing on understanding and supporting coordination as an interplay of people and technology. She has over 100 papers in journals and conferences. Her work has been recognized by an NSF CAREER award as well as several best paper awards.



Leonardo Gresta Paulino Murta is an Associate Professor at the Computing Institute of Universidade Federal Fluminense (UFF). He holds a Ph.D. (2006) and a M.S. (2002) degree in Systems Engineering and Computer Science from COPPE/UFRJ, and a B.S. (1999) degree in Informatics from IM/UFRJ. He has published over 150 papers in journals and conferences and received two ACM SIGSOFT Distinguished Paper Award at ASE 2006 and MSR 2019. His current research interests include configuration management, software evolution, and provenance.