Clone-World: A Visual Analytic System for Large Scale Software Clones

***Abstract*—With the era of big data approaching, the number of software systems, their dependencies, as well as the complexity of individual system are growing larger and more intricate. Understanding these evolving software systems is thus a primary challenge for cost-effective software management and maintenance. In this paper we perform a case study with evolving clones. We propose an interactive visualization system, *Clone-World*, that leverages the big data visualization frameworks to manage code clones in large software systems. We believe that *Clone-World* will not only ease the management and maintenance of clones, but also inspire future innovation to adapt visual analytics to manage big software systems.**

Visual investigation of how clone fragments are evolving together or in a group is important for clone refactoring, tracking, and clone related bug analysis. The programmers often need to manually analyze the co-evolution of clone fragments to decide about refactoring, tracking, and bug removal. However, manual analysis is time consuming, and nearly infeasible for large number of clones, e.g., with millions of similarity pairs, where clones are evolving over hundreds of software revisions. A few clone visualization techniques are already available in the literature, but they do not scale well with the number of software versions. In addition, a single visualization is not sufficient to capture the intangible and complex evolution of the clones. Our clone analytic system, *Clone-World*, gives an intuitive yet powerful solution to these problems. *Clone-World* combines multiple information-linked zoomable views, where users can explore and analyze clones through interactive exploration in real time. User studies and experts' reviews suggest that *Clone-World* may assist developers in many real-life software development and maintenance scenarios.

I. Introduction

Code cloning is a common yet controversial software engineering practice which is often employed by the programmers during software development and maintenance for repeating common functionalities. Cloning refers to the task of copying a code fragment from one place of a code-base and pasting it to some other places with or without modifications [45]. The original code fragment (i.e., from which the copies were made) and pasted code fragments become clones of one another. Two exactly or nearly similar code fragments form a clone pair. A group of similar code fragments forms a clone class.

Code clones are of great importance from the perspective of software maintenance and evolution. A rich body of studies [4], [6], [11], [15], [16], [20]–[22], [24], [27]–[30], [33], [34], [36], [37], [53] have already analyzed the impacts of clones on the evolution and maintenance of software systems. While some of these studies [4], [15], [16], [20], [24], [27]–[29] identifed positive impacts of code clones, a number of studies [6], [11], [21], [22], [30], [33], [34], [36], [37] have shown empirical evidences of strong negative impacts of code clones such as hidden bug propagation [30], late propagation [6], unintentional inconsistencies [6], and high instability [37]. Because of these negative impacts, code clones are considered among the most threatening aspects in a software system's code-base. Software researchers suggest us to manage code clones through refactoring and tracking.

While making decisions regarding clone management (such as clone refactoring or tracking) programmers often require to understand how the refactoring or tracking candidates evolved together in the past [38], [39]. More importantly, in order to analyze the origin of clone related bugs and inconsistencies, programmers need to visually analyze how the suspected clone fragments changed during the past commits. Due to the unavailability of appropriate tools or interfaces, the programmers are to manually check how each of the candidate clone fragments changed during the past commits. However, manual analysis of the co-evolution of multiple clone fragments residing in different source code files can be extremely time consuming. Evolution analysis of a single clone fragment involves locating each instance of that fragment in each of the previous revisions and analyzing the differences (i.e., changes) between the instances of the consecutive revisions. In such a situation, automatic support for visualizing the co-evolution of multiple clone fragments at a time can assist programmers analyze and manage clones in a time efficient way.

We develop a visualization tool, *Clone-World*, that can assist programmers to visualize clone classes or communities, their distributions over file systems, and co-evolution of multiple clone fragments from the same or different clone classes at a time. *Clone-World* supports seamless user interaction and multiple zoomable views that are information linked. *Clone-World* is inspired by the simplicity and usefulness of the geographic map navigation systems. *Clone-World* enables user to see the distribution of frequently changed clones in a software, analyze the impact of a clone change, understand the clone co-evolutionary relations for various clone refactoring tasks, and all these in an intuitive way.

The existing version control systems (such as SVN, GIT, CVS) help us visualize how a particular source code file evolved during the past commits. However, none of these systems can show us evolutionary history considering block level granularity. *Clone-World* can help us visualize the evolution history considering the block level granularity. In our research, the blocks are the clone fragments detected by a clone detector. *Clone-World* can not only show the evolution of a single clone fragment but also can show how two or more clone fragments from the same or different clone classes co-evolved in the past. In other words, *Clone-World* can help us visually analyze both within-boundary and cross-boundary evolutionary coupling among clone fragments. Such a visualization is important for analyzing and fixing clone related bugs, and taking clone refactoring and tracking decisions.

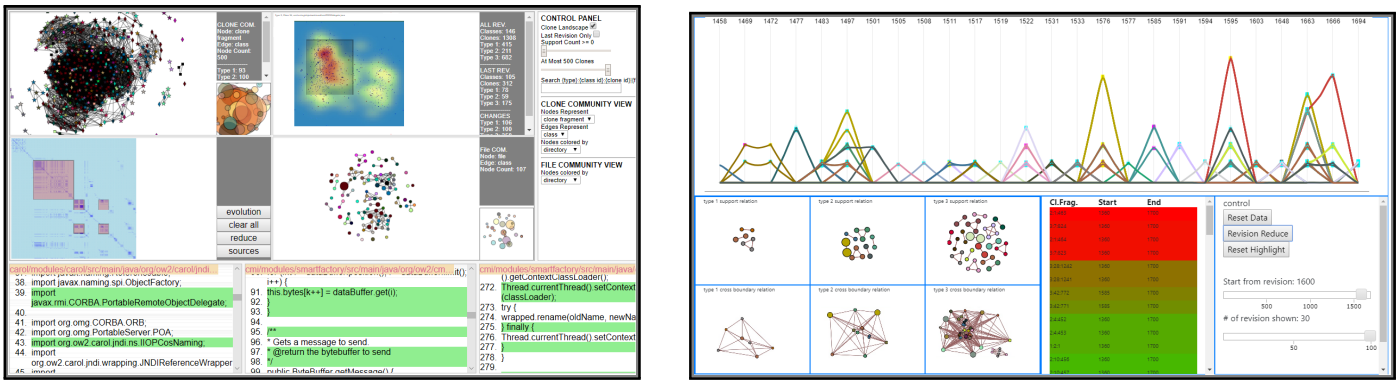

Fig. 1. Snapshots of *Clone-World*: (left) A clone-landscape, associated with clone and file communities, as well as with a SPCP-heatmap. (right) A clone-evolution view.

The rest of the paper is organized as follows. Section II presents some usage scenarios that motivated *Clone-World*. Section III, reviews existing big data visualization techniques and the current state of the art approaches in software and clone visualization. Sections IV–VII, describes the design of *Clone-World* and user experiments. Finally, Section VIII concludes the paper.

II. Usage Scenarios and Analytical Tasks

Assume that a programmer in a software company is working on a project. He receives a bug-report from the client about a particular project module. He fixes the bug, uploads the updated code to the production, and asks the client to check if the reported bug has got fixed. The client checks the module and reports that the problem is resolved. However, after a few days the client again reports the same bug in another module of the project. The project manager asks the programmer to investigate why they have got the same bug report even after fixing it. The programmer investigates and reports that the second module where the client found the same bug actually contains a similar piece of buggy code that the first module had. The manager then realizes that there can be some other modules in the project with the same bug and all these bugs should be fixed in the same way. He advises the programmer to find out all the modules that contain the similar buggy codes or functionalities, and fix them. After observing many such occurrences to repeat over time, the manager feels the necessity of identifying groups of similar code fragments in the project so that whenever a fragment in a group gets reported to contain a bug, the other fragments with the same bug can be easily spotted and fixed. Thus he asks the programmer to identify all groups (*clone classes*) of similar code fragments (*clones*) from the code-base of the project.

The programmer finds that a number of clone detectors are available on-line. He downloads one and applies it to the project's code-base in order to detect clone classes (i.e., groups of similar code fragments). He discovers that a huge number of code clones exist in the code-base. He also realizes that it would be beneficial if he could merge/refactor the fragments in a clone class, because in that case changing just one fragment would sometimes be enough to fix a bug. Existence of multiple fragments in a clone class can often require us to implement the same change in each of the fragments in the class individually and this is time consuming. Merging the fragments of a class into a single one can save a considerable amount of development time. However, it is practically impossible to apply merging/refactoring to all the detected clone classes. Thus, he needs to identify which clone classes are important for refactoring.

Finding important clone classes for refactoring from a huge number of clone classes is challenging. Analysis of evolutionary history of code clones is essential to mitigate this challenge. Code clones that mostly remain unchanged during evolution should be given less priorities compared to the change-prone ones. However, manually analyzing the evolutionary history of code clones for identifying the change-prone ones is much time consuming, because it requires retrieval of multiple instances of all clone fragments residing in multiple revisions of the project managed by the version control system. In such a situation, the programmer feels the necessity of a tool that can help him in analyzing the comparative change-proneness of the clone classes and fragments for refactoring.

The above scenario already presents us with some major questions: How can we analyze the clone distribution in a codebase? What are the uses of these clones? Do they correspond to the same functionalities? Which package contains the most clones? Which clones are important or relatively easy to refactor? The clone visualization framework we propose can help in all the above scenarios and beyond. Our framework applies a clone detector to all the revisions of a subject system from its SVN or GitHub repository, detect clone genealogies from these revisions, analyze their change-proneness, and finally integrate it into a visual interface for interactive visual investigation.

Our visualization framework can be used in many real-life scenarios, and is easy to adapt for the analysis of thousands of clones as well. For space reasons and a more focused analysis, we demonstrate only the following five usage scenarios.

S_1. **Identifying clone classes for refactoring.** *Clone-World* lets users to easily identify which clones or clone classes are important for refactoring on the basis of the changes they experienced in the past.

S_2. **Clone tracking.** While a programmer changes a particular clone fragment, *Clone-World* can help him determine other similar clone fragments from the same clone class. *Clone-World* also visualizes whether that particular clone fragment has coupling

with other clone fragments of different clone classes. Such cross-boundary couplings [38] are important for determining the impact set while making changes to a clone fragment.

S_3. **Visualizing changes that occurred in the past.** When fixing bugs or inconsistencies in the code-base, programmers might often need to analyze which changes that were made to a clone fragment in the past are buggy or inconsistent changes. *Clone-World* can help programmers in such a situation by instantly showing them which changes occurred to a clone fragment or to a group of clone fragments in the past commits.

S_4. **Analyzing clone distribution in a software system.** As cloning is considered harmful, project managers might want to investigate which parts or modules in a software system contain most of the clones so that those parts or modules can be considered for refactoring. *Clone-World* can help perform such investigations. It instantly clusters selected clone fragments in such a way that a user can easily understand whether they are making communities by residing in files or directories with close proximity.

S_5. **Clone usages & structural analysis.** This is a deeper investigation, where the developers may want to understand the uses of clones in a system. Why the clones are being created? How to they capture the functionalities of the system? *Clone-World* allows users to explore and get insight on such questions.

III. Background and Related Work

In this section we review existing approaches to visualize softwares, in particular, clones, and common big data visualization techniques. We begin our discussion by defining different types of code clones.

A. Different Types of Code Clones

According to the literature [49] Code clones are the exactly or nearly similar code fragments in the code-base of a software system. Two code fragments that are similar to each other form a clone-pair. A group of similar code fragments forms a clone class or a clone group. Code clones are of four types: Type 1, Type 2, Type 3, and Type 4. **Type 1** clones, also knows as identical clones, are exactly the same code fragments in a code-base. **Type 2** clones are syntactically similar code fragments. These are mainly created from Type 1 clones because of renaming variables or changing data types. **Type 3** clones can get created from Type 1 or Type 2 clones because of addition, deletion, or modification of statements. Type 3 clones are also known as gapped clones. Finally, code fragments that perform the same task but were implemented in different ways are known as **Type 4** clones. Type 4 clones are also knows as semantic clones. Our research in this paper incorporates the first three clone-types which are are the major clone-types in the literature.

B. Visualization of Softwares and Clones

A number of metaphor have been explored to visualize softwares such as city [54]–[56], park-like environment [25], or solar system [42]. Wettel and Lanza [56] proposed CodeCity, a 3D visualization that models softwares as cities: buildings in a district represent classes in software packages. The number of methods for a class, the number of attributes, etc. are mapped to different visual features of the building (e.g., width, height and color). CodeCity can process large scale software systems (e.g., systems with a million LOC). Vincúr et al. [54] designed 'VR City' that adapts city metaphor in an interactive 3D virtual reality environment. Khaloo et al. [25] modeled software codebase as a park where the user can engage with the system in an intuitive game-like environment.

Code-map, where the code is visualized in a scaled down representation of text (using colored pixed lines), is another popular technique to visualize a big-picture of the codebase. In a survey Bacher et al. [5] compiled several studies that use code-map approach. They found that most visualizations are targeted towards the maintenance activity, and aim to support multiple actives.

While the use of metaphor is common in visualizing a software system, visualization of software clones targets more on the clone analysis tasks. They have often been explored from a network visualization perspective, where the nodes correspond to clones and the links (edges) correspond to the pairwise relationships (such as similarity or co-evolution) [1], [13]. In a survey, Basit et al. [7] compiled 50 clone analysis tasks under 7 categories and examined which existing visualization technique (among about 22 approaches) could be appropriate to assist on those tasks.

Adar and Kim introduced SoftGUESS [1] to visualize clones. SoftGUESS has a 'genealogy browser' that arranges clones from left to right (every column represents a version). An edge between a pair of node reflects the predecessor and successor relationship during the evolution of the software. Another view 'encapsulation browser' shows how clones are distributed in different parts of a system, where the system itself is shown as a tree structure. Finally, a 'dependency browser' visualizes a network of package, class or method over different versions, where the edges represent how they evolved from one version to another version.

CYCLONE, proposed by Harder and Göde [18], can visualize clone evolution. Each software version in CYCLONE corresponds to a row, where each clone fragment is represented by a small circle. Each clone class, i.e., fragments that are clones of each other, is shown by a rectangle that groups the corresponding fragments. The ancestry of cloned fragments has been shown by vertical lines. They used colors to either illustrate the clone type or to show the type of changes. A similar visualization support is available in VisCad [3].

Although the existing clone visualization literature is rich, existing techniques attempts to visualize software revisions (rather than versions), and cannot visualize large datasets (e.g., clone networks with millions of clone pairs, and their evolution over thousands of revisions). In addition, the complexity of clone analysis tasks [7] requires different visualization views to be information linked and integrate seamless interaction. To overcome such challenge, one needs to adapt the innovation in big data visualization.

C. Interactive Visualization

The common approach to deal with the big data problem in visualization is to use various forms of data reduction, sampling and clustering techniques [2], [31], [32], [43]. Even

TABLE I. SUBJECT SYSTEMS

Systems	Lang.	Domains	LLR	Revs
Ctags	C	Code Definition Generator	33,270	774
Carol	Java	Game	25,091	1700
Freecol	Java	Game	91,626	1950
jEdit	Java	Text Editor	1,91,804	4000
JabRef	Java	Reference Management	45,515	1545

LLR = LOC in the Last Revision Revs = No. of Revisions

after such preprocessing, the size of the refined datasets remains large enough to challenge real-time user interactions. This inspired the idea of "overview first, zoom and filter, then details on demand" [51], and other forms of interactive visualizations. Hierarchical visualization is a very well-alight with this concept, where the datasets are visualized in several levels. The top level offer the overview of the datasets with sufficient hint to the important features that the user would be interested to explore further. Such visualizations often support zoom and filtering to so that the details can be reveled for a 'manageable' amount of information. An ideal example in this context is the geographic map, e.g., Google or Bing maps, which has been adapted to many visual analytic systems [41], [58]. Another common technique to visualize very high-dimensional data is by parallel coordinates, where dimensions are represented by parallel vertical lines, and each datapoint is drawn as a polyline that connects points on these vertical lines based on its corresponding values. With appropriate preprocessing parallel coordinate visualization can be constructed for millions of data points [19]. We refer interested readers to [9] for a survey on known approaches to process and visualize big data.

Only a few techniques have recently been proposed that support real-time spatial queries for large multidimensional spatio-temporal datasets [12], [32], e.g., twitter data. These approaches are very effective for computing Heatmaps or histograms, but not suitable for complex visualization. A number of systems for large network analytics exist, but they come at an expense of sophisticated algorithms and high system complexity [17], [44].

IV. SUBJECT SYSTEMS & DATASET PREPARATION

In this section we describe systems that we analyzed using our clone visualization system, and discuss the process flow for the overall data preparation for visual analysis of these systems.

A. Systems Under Test

We conduct our experiment on five open-source subject systems by downloading those from an on-line SVN repository called SourceForge.net [**?**]. Table I contains the details of these systems. To better explain the behavior of our visualization system, we choose systems that are diverse in terms of application domains, implementation languages, and revision history lengths. Considering each of our subject systems, we detect clones from all the revisions using the NiCad clone detector [10]. Then, we detect clone genealogies using the SPCP-Miner tool [40]. After detecting clone genealogies, we apply UNIX *diff* operation to automatically identify how a clone fragment corresponding to a clone genealogy changed over the evolution. Genealogies of different clone-types along with their change information are stored in a database. We will describe our database in Section IV-C. Clone detection results from any clone detector arranged in the format described in Section IV-C can be visualized using our visualization tool.

B. Settings for the Clone Detector

We used NiCad [10] for detecting code clones, which has recently been reported as a very effective clone detector in a study conducted by Svajlenko and Roy [52]. We used NiCad with a configuration that it would detect clones of a minimum of 10 LOC with 30% dissimilarity threshold, using a blind renaming of the identifiers. Svajlenko and Roy [52] used these settings for comparing NiCad with other existing alternatives and found NiCad to be a very promising tool in terms of both precision and recall in detecting all three major types of code clones (Type 1, Type 2, and Type 3).

C. A Comprehensive MySQL Database

SPCP-Miner creates an individual database for each of our subject systems. Our databases are available on-line []. A particular database contains the genealogies of three types of code clones (Type 1, Type 2, and Type 3) from a subject system in three tables: 'type1clones', 'type2clones', and 'type3clones'.

Each of the three tables 'type1clones', 'type2clones', and 'type3clones' contains code clones from each of the revisions of the subject system. Let us assume that we are now working on the table 'type1clones' in the database called 'ctags' (for subject system Ctags). The very first revision that contains code is revision 2. The last revision that we analyzed for Ctags is 774. From each of the revisions, 2 to 774, we detected Type 1 clones and stored those clones in table 'type1clones'. A particular row in this table contains information about a particular clone fragment in a particular revision. The information is distributed to the fields. Let us now discuss the fields. The value of the field called 'revision' in a particular row refers to the revision where the clone fragment denoted by that row resides. The field called 'filepath' contains the path to the file where the clone fragment remains. The 'startline' and 'endline' fields contain the starting and ending line numbers of the clone fragment in the file. The 'cloneclass' field contains the clone class id of the clone fragment. There can be multiple entries in the table with the same revision number and the same clone class id. These entries (i.e., these clone fragments) belong to the same clone class. Two entries with the same clone class id but with different revision numbers should never be considered belonging to the same clone class, because they reside in two different revisions. The field named 'changecount' contains the number of places where the clone fragment denoted by the row experienced changes before being forwarded to the next revision. Here are two example queries that we used to extract information for the clone visualization.

Identifying all clone classes in a particular revision. Suppose we want to identify all the clone classes in revision 10. We first need to retrieve all entries from the table (type1clones) corresponding to revision 10. We then need to group these entries on the basis of the clone class ids (i.e., 'cloneclass' field). Entries with the same clone class id shuold belong to the same group or the same clone class. We can also retrieve Type 2 and Type 3 clone classes in any revision from the tables 'type2clones' and 'type3clones' respectively.

Retrieving clone genealogies. If we look at the fields of table 'type1clones' we can see a field called 'globalcloneid'.

Let us consider the global clone id 10. From the entire table 'type1clones', we find all the entries with globalcloneid = 10. These entries are of different revisions. It is impossible that two entries with the same global clone id belong to the same revision. Now if we arrange the entries that we obtained with global clone id 10 in chronological order (i.e., ascending order) of revisions, we get the clone genealogy with global clone id = 10. The entries in this genealogy are the snapshots of the same clone fragment in different revisions. The number of distinct global clone ids in table 'type1clones' is the total number of Type 1 clone genealogies that were created during the whole period of evolution of the software system. We can retrieve Type 2 and Type 3 clone genealogies in a similar way from the tables, 'type2clones' and 'type3clones' respectively.

V. Visual Analytics of Clones

A. System Interface

Clone analytics tasks are complex and diverse, which requires interface that combines multiple views of the same data, as well as supports seamless user interaction. For example, a high-level visualization that depicts the spread of clones in a system may not be useful for the detailed analysis of pairwise changes between clones. Similarly, visualizations targeting clone refactoring tasks may not be suitable for users who attempts to understand how the clones are being used by the developers. To cope with such challenges, *Clone-World* includes an interactive system with multiple information-linked zoomable views for clone analytics. The interactions are synchronized over all views. The users can see multiple visual instances of the same information in a different context. They can navigate through various levels of details and can apply real-time filtering based on their need. Fig. 1 shows a snapshot of our clone analytics system. In the following we describe the details of various components of the system.

1) Clone-Landscape View: The clone-landscape view uses the landscape metaphor to capture the clone distribution across the system. The dots on the landscape corresponds to the clones. Each clone is associated with a change count, i.e., the number of times the clone changed over all the previous revisions. The landscape generated using a surface plot weighted by the clone change count. The color is generated from a diverging color scale that changes from blue to red depending on how frequently the clones in a region are changing. Fig. 2 represents the clone-landscapes of the subject systems.

Design Details: To project the file system structure onto the landscape, we employ a multidimensional scaling approach [26]. The distances between a pair of clones is computed based on a locality metric. clone pairs in the same file are considered to have one unit of distance between them. Clone clone pairs with the same parent (resp., grand parent) directory but in different files are assigned a distant of two (resp.,three) units. The usefulness of such distance matrices have widely been studied in the literature for clone locality analysis [23], [50]. The multidimensional scaling attempts to assign coordinates for the clones such that the locality metric is realized as the Euclidean metric as much as possible. Since the coordinates calculated may sometimes put two clone instances very close to each other, we apply a overlap-removal algorithm [57] to clear the plot further. Even if the clone density is high, the users can zoom in and pan to see further details (similar to map navigation systems).

Once the coordinates are calculated, then a natural approach to plot a surface is based on weighted point density. To give the users an idea of the clone set that are most change prone, we choose clone change count (over all the previous revisions) as the weight parameter. However, we observed that only a small fraction of clones changes frequently, while majority of them don't change much (see Fig. 3). Therefore, we computed the 25%, 50%, and 75% quantile of the change counts, and then assigned each clone a weight based on its position in the quantile ranges. The surface plot is then computed using a weighted density estimation.

Interaction: To support the real-life zoom interaction, we use pre-rendered tiles. We store the tiles in a tree data structure of depth 5. The top level corresponds to the overall overview of the landscape, which can already support a few millions of clones. The system can scale up to a higher number, but with larger numbers the first overview becomes heavily cluttered. To tackle this problem we can use a larger depth trees and maintain a quota per tile. Secondly, we can layout the clones based on their priorities (change count) respecting the tile quota, following a technique proposed in [35], [41]. The user can select a subset of clones by drawing a rectangular range in the landscape.

2) Clone-Community view: The clone-community view (Fig. 3) adapts the community concept in networks to capture how the clones form communities based on different distance metrics. The user can choose the network nodes in this view to be clones or clone classes. If the nodes are clones, then the edges may correspond to whether the corresponding clones changed together (SPCP score), belong to the same file or same clone class. If the nodes are clone classes, then an edge denotes that the corresponding clone classes hit a common file, i.e., they have clones that reside in a common file. The clone network is constructed in real time based on the range selected by the user in the landscape view.

Design Details: A community detection algorithm attempts to partition a the network nodes into subsets that maximizes the edge density within each subset and reduces the edge density across different subsets [14]. We use the Louvain method [8], a popular community detection approach based on iterative modularity maximization, where modularity is a function that measures the quality of the current communities.

Once the communities are formed, we use a force layout algorithm [] to render them on the canvas. The force layout algorithm apply charges to the edges so that the nodes repel each other but also try to achieve the desired edge length based on minimizing the energy of the layout. We use distinct symbols to represent different types of clones, i.e., clones of type 1, 2, 3 are represented using diamond, square, and star shapes respectively. The view is associated with a mini-map to give a quick overview of the sizes of different communities.

Interaction: The user can select the dependency (SPCP score, common file, or common clone class) denoted by the edges between a pair of clones as they need. We assign each clone class, file and community, a distinct color. Thus for each selection for the edge, users can color the nodes either by class, filepath, or community. This gives raise to 10 combinations in total, 9 when nodes are clones, and 1 when they are clone class.

Right clicking a clone brings up a context menu, which

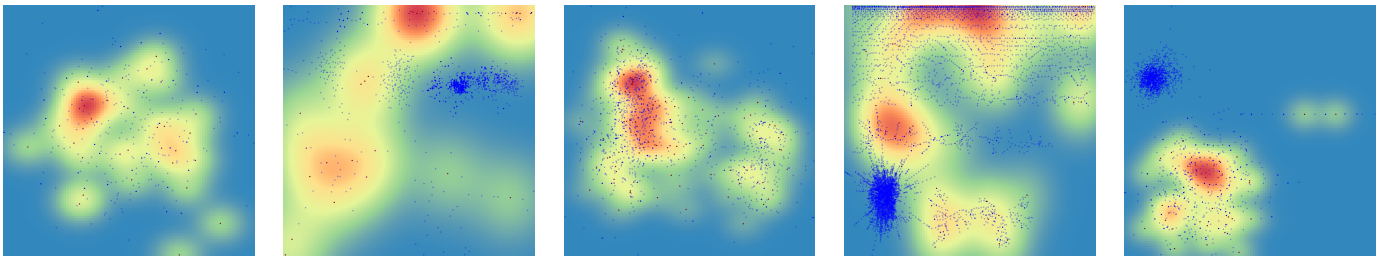

Fig. 2. Clone-Landscapes for the subject systems (from left to right) Ctags, Carol, Freecol, jEdit, JabRef.

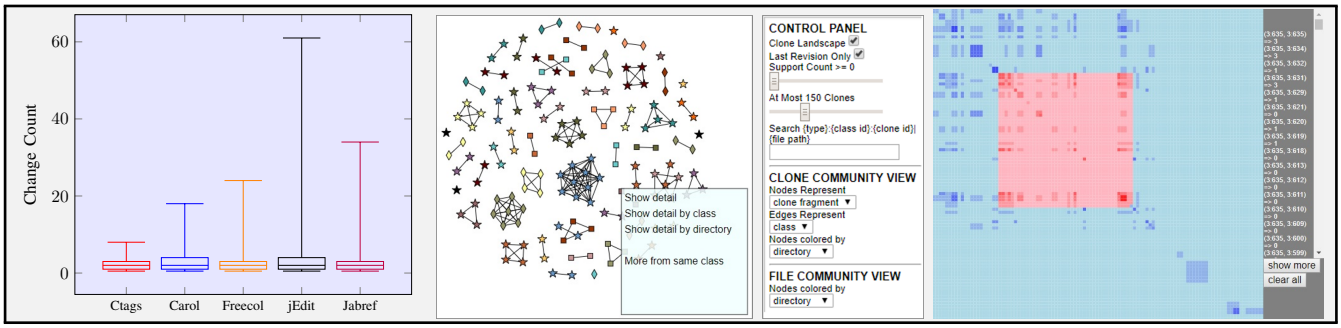

Fig. 3. (left) Change Count in Different Systems. (middle) A clone-community view of *Clone-World*. (right) A SPCP-Heatmap view of *Clone-World*.

gives the users options to navigate to the source files. The user may choose to open one file, where the lines corresponding to the clone are highlighted. They can also open all files (side-by-side in new tabs) that contain all clones of the same class, or other clones in the same file, add other clones into the view that belongs to the same class, and so on. The user can hover on the nodes to see quick information, and zoom and pan the view as needed.

3) SPCP-Heatmap view: This is a Heatmap (Fig. 3) that visualizes the SPCP-scores of the clones selected in the clone-community view. Although the user has an option to visualize the clones such that community is formed based on SPCP-scores, it is not possible for a force layout algorithm to realize all the desired edge lengths accurately. We thus introduced this SPCP-heatmap to give further control to the users so that they can interactively select a set of frequently changing clones.

Design & Interaction: The rows and columns of the Heatmap correspond to the clones, where they are sorted by their type from left to right. Each cell in the Heatmap corresponds to the SPCP-score between a pair of clones, and its color intensity corresponds to the score. The users can select/deselect an area of the Heatmap by right clicking and dragging the mouse. The ids of the selected clones are accumulated in a panel. The user can click the 'reduce' button to remove all the clones from the clone-community view that have not been selected. The user may see the details of the selected clones, i.e., their genealogy, the revisions where they changed together, types of changes, etc., by clicking the 'clone-evolution view' (described later). The user can also zoom and pan the view as needed.

4) File-Community view: This view is similar to the clone-community view except that the nodes correspond to the files corresponding to the user-selected clones, and user can connect a pair of files based on whether they share a common clone class, or lie in a common parent directory. The files are denoted by circles of varying sizes, based on the number of clones it contains. Users can color the nodes by the directory or community. They can the hover on the nodes to see the filepath, and zoom and pan the view as needed. In addition, the user can drag file nodes into the three code-view panels to see the source codes.

5) Clone-evolution view: The evolution view is activated from the Heatmap where the user selects a set of fragments and click the 'clone-evolution button'. The evolution view appears in a new tab. The clone-evolution view adapts a parallel-coordinate view to show the change count (Fig. 1(right)).

Design & Interaction: Each x-monotone line in the view corresponds to a clone chain, and the parallel vertical lines correspond to the revisions. The height of the chain at each revision corresponds to the number of changes (additions, deletions and modifications) the clone experienced from the previous revision. Since there may sometimes be thousands of revisions, we equip the evolution view with two range sliders, one that controls the starting revision, and the other controls how many revisions (vertical lines) will be visualized in a single view. However, the clone changes are often sparse, e.g., there may be many revisions where the clones of interest have not been changed. We thus add another 'revision reduce' control to render all revisions except for those that did not experience any change. In addition, the parallel coordinate view support brushing and filtering, which is a standard interaction where user can select a set of clones by drawing a rectangle on the revision axis for further exploration. We also maintain a dynamic table that show the selected clone ids, and their corresponding start and end revision information. The types of changes, e.g., addition, deletion, other changes, are shown as a colored square at the intersection point of the clone chain and the revision axis.

The evolution view also supports a detailed analysis of the SPCP-scores by partitioning the clone network in a 2×3 grid,

where an edge denotes that the corresponding clones changed together. The columns of the grid correspond to different clone types. The top row only contains edges connecting the clones from the same class, whereas the bottom row contains the rest of the edges, which are known as the cross boundary relations, i.e., they represent clones that changed together, but belong to different clone classes. The size of the nodes correspond to the change count of the clones. It is interesting to observe that the communities contains nodes of the same classes (color) and size, whereas the communities of the bottom row contains nodes of different classes and sizes. The users can navigate to the source files of a particular clone or clone community by right clicking the clone node and interacting with the associated context menu.

6) Further Interface Controls: In addition to the user controls described above, there are two additional range sliders in the control panel. One slider is to set an upper bound on the number of fragments that appear in the clone community view. The other slider is to set a lower bound on the SPCP-score. Thus an upper bound of u and a lower bound ℓ will select all the pair of clones that have SPCP-score at least ℓ, but chose at most u clones among them to render. Here the clones are selected based on their priorities, i.e., change count.

The control panel also includes a search box. The user can search for a particular clone, all clones of a clone type, or all clones of a clone class. For example, a search for '$a : b, x : y : z$' will add all clones of type a and class b, as well as the clone z with type x and class id y, in the clone community view. They search box also supports file search, e.g., searching for "|apk.java" adds all files with filepath ending with 'apk.java' in the file community view.

VI. Implementation of *Clone-World*

The back-end (data access and prepossessing) of *Clone-World*has been implemented using Java and R programming language. The front-end has been implemented using D3, which is a popular JavaScript library for producing dynamic, interactive data visualizations in web browsers. An independent front-end implementation of our system, integrated with a moderate size dataset, has been hosted here[1]. This front-end takes the clone and file networks in a JSON file format, does not have any data preprocessing overhead. Therefore, given the network data in the appropriate JSON format, *Clone-World* can readily visualize the dataset on its user interface.

VII. Case Studies and Experiments

In this section we first discuss how to interact with *Clone-World* for the usage scenarios S_1–S_5 (as discussed in Section II). We the summarize our observations of the case studies, and list tasks for a subsequent usability analysis.

Case Study S_1 (Identifying clone classes for refactoring): The developer selects a set of change-prone clones by drawing a rectangle that covers a red region in the clone landscape view (Fig. 4(a)). If many clones are selected, then he can use the range sliders to focus on a fraction of clones with high change count. He then interacts with the clone and file community views to select the desired clone classes.

The suitability of a class for refactoring may depend on many factors, e.g., refactoring may be easier if the class does not contain many clones, or if the clones belong to only a few files. The developer can choose the edges to represent 'common-class' relation, which groups the clones based on clone classes. This gives the developer an idea of the size of the classes. Subsequently, he can choose to color the clone nodes by the directory colors (Fig. 4(b)). If the clones of a class are colored by a few colors, then it may be a suitable candidate for refactoring.

The developer may further refine his choice based on the SPCP-scores (co-change information) of the clones. He can select cells with high SPCP-scores, and click on the 'reduce' button to keep only the selected clones. He can then analyze them further in the clone-evolution view by clicking on the 'clone-evolution' button.

Case Study S_2 (Clone tracking): The developer first selects a set of clone pairs with high SPCP-scores in the same way as in S_1, and then uses the clone-evolution view to see whether these clone fragments have cross boundary couplings (Fig. 5). These are clones that may be difficult to refactor (since they belong to different clone classes), but important for tracking, and thus for handling future clone change or bug fixes.

The developer can also search for one or more clone or clone classes in the search box, and visualize these clones in the clone-evolution view to analyze their cross boundary relationships.

Case Study S_3 (Visualizing changes that occurred in the past): This scenario is motivated by the bug fixing tasks. If the bug corresponds to a cloned code, then the developer searches for the corresponding clone id, uses the context menu to visualize evolution of the all clone fragments that changed together with the current clone.

In the clone-evolution view, he selects the 'reduce revision' button to get the parallel coordinate view of only the revisions that experienced clone change (Fig. 6 (top)). Then he uses the brushing and filtering interactions to further analyze the co-evolution of the clones.

Case Study S_4 (Analyzing clone distribution in a software system): Since clones are considered harmful, developer may want to investigate the files and directories that contains change-prone clones. In this case, developer first interacts with the landscape and Heatmap (as in S_1), but keep his focus on the file community view. Once the developer selects a desired clone set, he can color the file community view by parent-directory colors. The communities and node sizes give him a quick idea of the clone density in different directories.

The developer can also explore whether files from different directories are forming communities. If so, then this is often an indication of a poor design, i.e., the system may have high coupling among modules lying in different directories.

Case Study S_5 (clone usages & structural analysis): *Clone-World* is integrated with the software source files such that the user can constantly switch between the sources and analytics interfaces. The developer can right-click on a clone-fragment node in the community view, and from the context menu, he can choose to open source files for one or more clones that are related (i.e., belong to the same class, or change together). The files are opened in new tabs, where the cloned codes are highlighted for further investigation.

[1] http:\asdf

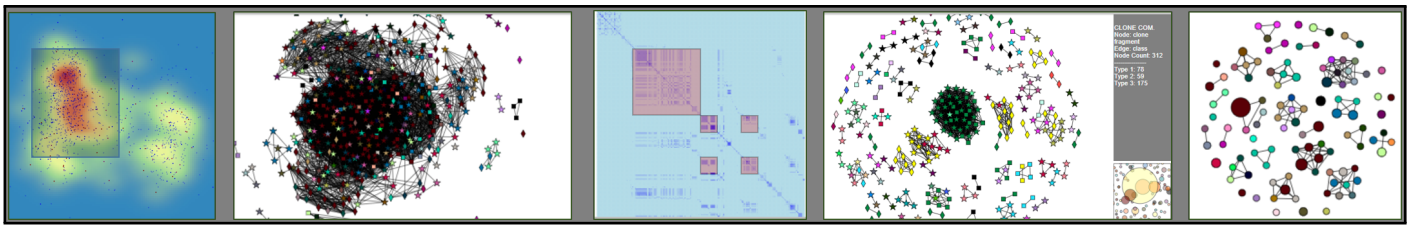

Fig. 4. Illustration for the case study S_1. (left) Selection on the landscape and the corresponding clone-community view, (middle) further refinement by region selection on Heatmap (right) refined clone community, and the corresponding file community.

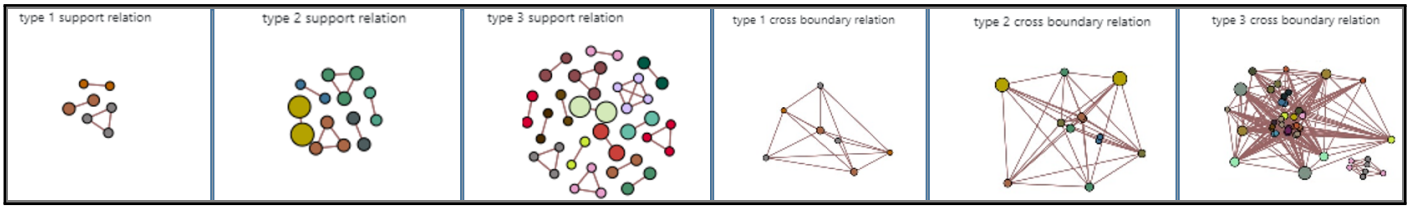

Fig. 5. Illustration for the case study S_2. The left three cells correspond to the within boundary relation, and the right three correspond to the cross boundary relation for different clone types.

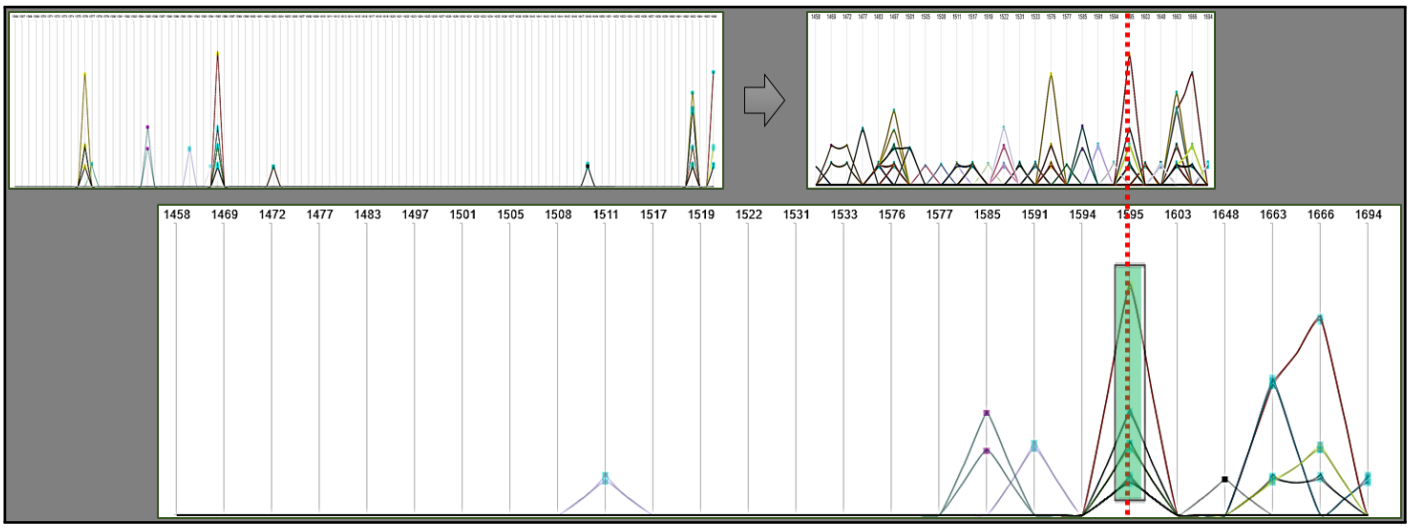

Fig. 6. Illustration for the case study S_1. (left) Selection on the landscape and the corresponding clone-community view, (middle) further refinement by region selection on Heatmap (right) refined clone community, and the corresponding file community.

A. *Case Study Results*

For each system, we analyzed the last revision available for the systems. Using the work-flow described in Fig. 4, we selected a set of highly changing clone fragments that are alive in the last revision. Table II shows how well the selected clones cover the number of total changes for all clones and co-changes for all pair of clones. Although the visual selection is subject to users' perception, the results show that *Clone-World* allowed us to choose for each system, a reasonably small set of clone fragments that covers most of the clone change events. We often found clone classes that contain between 10 to 20 clone fragments, but with less than three distinct colors (while colored by the file colors). This indicates that these clones are localized in a few files, and thus may be easier for refactoring. We then further explored their evolution information.

For the selected clones in S_1, we analyzed the within and cross-boundary relationships among the selected clones. We noticed that with a low SPCP-score threshold, each system depicts a large number of cross boundary relationships compared to the within boundary relationships. On the other hand, a high SPCP-score favours more within boundary relationships. For example, with an increase in the SPCP-score threshold from 1 to 5, the Carol system shows the following (within/cross) boundary relationship ratios: $\{0.13, 0.12, 0.16, 0.37, 2\}$.

TABLE II. COVERAGE OF THE SELECTED CLONE FRAGMENTS

Systems	Fragments Selected	Changes Covered	Co-changes covered
Ctags	35.64%	49.27%	40.05%
Carol	23.07%	17.42%	4.84%
Freecol	39.78%	47.73%	46.60%
jEdit	3.26%	33.09%	10.25%
JabRef	56.42%	16.33%	1.17%

The evolution view revealed insights on how the clones evolve, e.g., in Fig. 6 (bottom), we selected all the clone fragments (among the selected ones in S_1) of Carol that changed in

software revision 1595. We observed that 12 fragments were changed and the amount of change for some of them was similar (as they evolved in 4 groups). These fragments are perhaps important for refactoring or tracking since some of them changed several times in subsequent revisions. We found similar scenarios in all the other subject systems. Furthermore, there often exist small groups consisting of two to five clone fragments that changed closely over many software revisions. Thus *Clone-World* was helpful to easily select clones that evolved together and then to navigate to the source code to see further details, as described later in S_5.

For each system, we used *Clone-World* to select a set of highly changing clone fragments, and analyzed the large-size clone classes by investigating the source code files that covered those clone classes. Here we report a few scenarios where these clones have been used. For Carol, the analysis revealed functionalities such as encoding message (url/string) to a byte buffer, message decoding (which is analogous to encoding), constructing objects or threads with different function signatures, implementing conditional logics or exception handling, etc. For Ctags, the change-prone clone classes consist of functionalities that involve reading characters from files under various rules, initializing variables or formatting strings based on the character read from a file. For Freecol, we found the uses of clones in handling mouse events, and also in controlling different game modules. For jEdit, we noticed several functions with very similar functionalities to be cloned, where each of them has a different function signature. For JabRef, clones were used for processing strings, e.g., preparing a string in many different formats, for exception handling or implementing a chain of function calls, etc.

B. User Studies

To evaluate the usability of the system, we conduct a user study. We recruited 16 participants, each had a minimum of 2 years of experience either in industrial software development or in software engineering research. None of them had prior experiences in visual analytics of clones.

We first described the participants the visual encodings of *Clone-World*, and then asked to perform the following tasks by interacting with *Clone-World*. After each task, we asked users to score in a scale of 0(very-low) to 5(very-high) depending on how much effort it took to perform those tasks. Finally, we also asked for qualitative feedback on the system.

Task 1: Find 5 clone classes among a set of highly changing clones such that each class contains clone pairs with SPCP-score at least α. How many files correspond to these clone classes? [scenario: clone refactoring]

Task 2: Search for the clone classes a, b, c, d, and report all pairs of classes that correspond to cross boundary relationships. [scenario: clone tracking]

Task 3: Find the clone x, and visualize all the clones that changed with x in previous revisions. Report 5 clones that changed closely with x. [scenario: clone evolution analysis for a bug fix]

Task 4: Find a file x, and retrieve all the clones contained in it. Report the software revisions where these clone codes were modified. [scenario: the file needs to be deleted due to copyright issues]

Task 5: Find file communities that show high coupling, i.e., many files (nodes) of such a community would be colored in different colors. [scenario: software maintenance and structuring]

Results: Although Task 1 is highly dependent on the visual perception of individual participants, the result shows consistency among the participants' class selection task. Fig. 7(first), illustrates the top five clone classes that were frequently chosen by the participants. These 5 classes cover about 40% of the total participants' vote. About 70% of our participants reported that their chosen classes were in at most 5 files. For Task 2, we measured the accuracy of the participants while reporting the cross boundary relations (Fig. 7(fourth)). About 53% of our participant correctly reported the relations and 30% of the rest correctly identified all but 1 relation (mean = 2.30, s.d. = 0.94). Task 3 was also subject to individual concept of 'closeness' and 'coupling', however, *Clone-World* was again able to help participants' choose the relevant clone fragments consistently. As shown in Fig. 7(second), the top five frequently chosen clone fragments cover about 48% of the total participants' vote. Similarly, the top five frequently chosen software revisions (in Task 4) cover about 38% of the total participants' vote (Fig. 7(third)). The outcome for Task 5 (Fig. 7(fourth)), showed relatively more variability (mean = 8.61, s.d.=5.69), which perhaps attributes to the difference in individual perception about coupling.

Fig. 7(fifth) shows the participants' feedback on the difficulty of completing different tasks and their rating on a five point Likert Scale. A Friedman test showed that, the difficulty of task completion differed ($\chi^2_{(4,N=13)} = 15.08, p < 0.005$). An Wilcoxon Signed-rank test shows that there is a differences between Tasks 1 and 5 ($Z = 2.22, p < 0.05$), Tasks 2 and 4 ($Z = 2.35, p < 0.05$), Tasks 3 and 4 ($Z = 2.55, p < 0.05$), and Tasks 3 and 5 ($Z = 2.78, p < 0.01$).

On a question of how difficult was it to use *Clone-World* , only 1 participant ranked 5 (very difficult), 3 participants raked this as difficult, and the rest raked 3 or less. *Clone-World* was able to make positive impression among the participants. In a question where the participants were to rank their impression on *Clone-World* (1-very positive, 5-very negative), only 1 participant ranked 4, and the rest raked between 1 and 3 (mean = 2.38, s.d. = 0.86, median 2).

C. Experts' Review

We interviewed two software developers (Expert A, Expert B) in the area of code clone visualization. We first explained our system *Clone-World* in details, and then the case studies presented above. They both thought that *Clone-World* would be able to assist both software trainees and developers.

Expert A commented that "I was able to do different tasks such as identifying clone classes for refactoring, how frequently clone in same or different classes have changed over many revisions". She also suggested many thoughtful scope for extensions. For example, an user may select a region in the landscape, and subsequently may want to check whether a particular clone fragment belongs to the selected area by searching for that fragment in the search box. Similarly, one can search for a file and may want to see the clone fragments that it covers in the selected landscape. Since the search box is currently independent of the selection in the landscape, it would make sense to combine these two interactions to allow

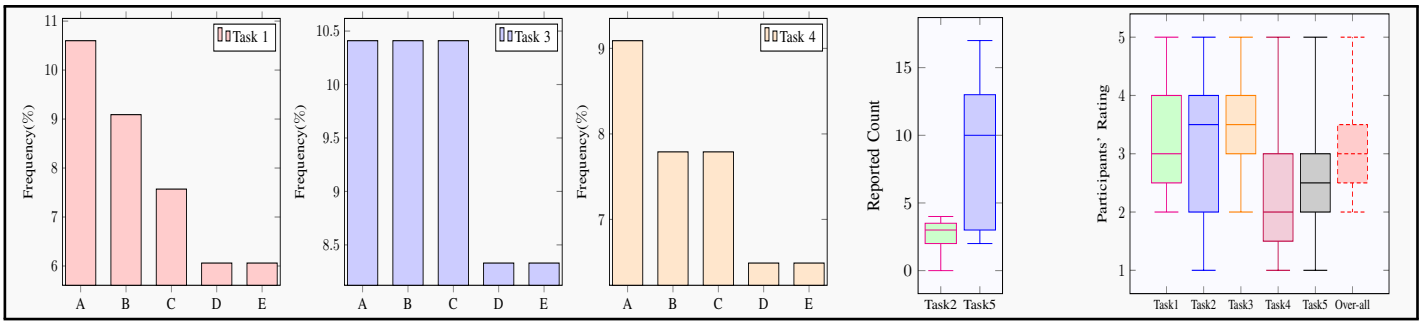

Fig. 7. A summary of the results obtained from the user study.

users to make more complex queries.

She also recommended to augment the Heatmap tool with more interactive options. For example, the user could apply reordering of the rows and columns based on the change count. They may also want to hover on the nodes in the clone community and expect to see the corresponding cells highlighted. Currently, the SPCP-Heatmap is used to refine the selection of the clone and file community views.

Expert B

VIII. Discussion

A. Threats to Validity

Our clone visualization tool, CloneWorld, works on top of the NiCad clone detector [10]. For different settings of NiCad, the clone detection results might be different, and thus, our findings from visual analysis might also be different. However, the settings that we have used for NiCad are considered standard [46]. NiCad has been reported to show high precision and recall with these settings [47], [48]. Moreover, according to a recent study [52], NiCad is a promising choice for detecting code clones among many other alternatives. Thus, we believe that our findings from visual analysis of clone data obtained by applying by NiCad are important.

Our user study consists of 16 participants. While a higher number of participants would be good for our study, each of our participants has an experience of working with code clones for at least two years. Most of these participants have expressed a positive impression regarding the applicability of CloneWorld tool. Thus, according to opinions even from these 16 participants, our tool can be considered an important contribution towards managing code clones.

B. Future Work and Conclusion

Write your conclusion here.

References

[1] Eytan Adar and Miryung Kim. Softguess: Visualization and exploration of code clones in context. In *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007*, pages 762–766. IEEE Computer Society, 2007.

[2] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In Zdenek Hanzálek, Hermann Härtig, Miguel Castro, and M. Frans Kaashoek, editors, *Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013*, pages 29–42. ACM, 2013.

[3] Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. Viscad: flexible code clone analysis support for nicad. In James R. Cordy, Katsuro Inoue, Stanislaw Jarzabek, and Rainer Koschke, editors, *Proceeding of the 5th ICSE International Workshop on Software Clones, IWSC 2011, Waikiki, Honolulu, HI, USA, May 23, 2011*, pages 77–78. ACM, 2011.

[4] L. Aversano, L. Cerulo, and M. D. Penta. How clones are maintained: An empirical study. In *CSMR*, pages 81 – 90, 2007.

[5] Ivan Bacher, Brian Mac Namee, and John D. Kelleher. The code-map metaphor - A review of its use within software visualisations. In Lars Linsen, Alexandru Telea, and José Braz, editors, *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017) - Volume 3: IVAPP, Porto, Portugal, February 27 - March 1, 2017.*, pages 17–28. SciTePress, 2017.

[6] L. Barbour, F. Khomh, and Y. Zou. Late propagation in software clones. In *ICSM*, pages 273 – 282, 2011.

[7] Hamid Abdul Basit, Muhammad Hammad, and Rainer Koschke. A survey on goal-oriented visualization of clone data. In *3rd IEEE Working Conference on Software Visualization, VISSOFT 2015, Bremen, Germany, September 27-28, 2015*, pages 46–55. IEEE Computer Society, 2015.

[8] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 10:P10008, 2008.

[9] Enrico Giacinto Caldarola and Antonio M. Rinaldi. Big data visualization tools: A survey - the new paradigms, methodologies and tools for large data sets visualization. In Jorge Bernardino, Christoph Quix, and Joaquim Filipe, editors, *Proceedings of the 6th International Conference on Data Science, Technology and Applications, DATA 2017, Madrid, Spain, July 24-26, 2017.*, pages 296–305. SciTePress, 2017.

[10] J. R. Cordy and C. K. Roy. The nicad clone detector. In *ICPC Tool Demo*, pages 219 – 220, 2011.

[11] N. Göde D. Steidl. Feature-based detection of bugs in clones. In *Proceedings of the 7th International Workshop on Software Clones (IWSC'13)*, pages 76 – 82, 2013.

[12] Cicero Augusto de Lara Pahins, Sean A. Stephens, Carlos Scheidegger, and João Luiz Dihl Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE TVCG*, 23(1):671–680, 2017.

[13] Christopher Forbes, Iman Keivanloo, and Juergen Rilling. Doppel-code: A clone visualization tool for prioritizing global and local clone impacts. In Xiaoying Bai, Fevzi Belli, Elisa Bertino, Carl K. Chang, Atilla Elçi, Cristina Cerschi Seceleanu, Haihua Xie, and Mohammad Zulkernine, editors, *36th Annual IEEE Computer Software and Applications Conference, COMPSAC 2012, Izmir, Turkey, July 16-20, 2012*, pages 366–367. IEEE Computer Society, 2012.

[14] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486:3–5, 2010.

[15] N. Göde and J. Harder. Clone stability. In *CSMR*, pages 65 – 74, 2011.

[16] N. Göde and Rainer Koschke. Frequency and risks of changes to clones. In *ICSE*, pages 311 – 320, 2011.

[17] Wook-Shin Han, Sangyeon Lee, Kyungyeol Park, Jeong-Hoon Lee, Min-Soo Kim, Jinha Kim, and Hwanjo Yu. Turbograph: a fast parallel

graph engine handling billion-scale graphs in a single PC. In *ACM SIGKDD (KDD)*, pages 77–85. ACM, 2013.

[18] Jan Harder and Nils Göde. Efficiently handling clone data: Rcf and cyclone. In *Proceedings of the 5th International Workshop on Software Clones*, pages 81–82. ACM, 2011.

[19] Julian Heinrich and Daniel Weiskopf. Continuous parallel coordinates. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1531–1538, 2009.

[20] K. Hotta, Y. Sano, Y. Higo, and S. Kusumoto. Is duplicate code more frequently modified than non-duplicate code in software evolution?: An empirical study on open source software. In *IWPSE*, pages 73 – 82, 2010.

[21] L. Jiang, Z. Su, and E. Chiu. Context-based detection of clone-related bugs. In *ESEC-FSE*, pages 55 – 64, 2007.

[22] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *ICSE*, pages 485 – 495, 2009.

[23] C. Kapser and M. Godfrey. Toward a taxonomy of clones in source code: A case study. In *ELISA*, pages 67–78, 2003.

[24] C. Kapser and M. W. Godfrey. "cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering*, 13(6): 645 – 692, 2008.

[25] Pooya Khaloo, Mehran Maghoumi, Eugene M. Taranta II, David Bettner, and Joseph J. LaViola. Code park: A new 3d code visualization tool. In *IEEE Working Conference on Software Visualization, VISSOFT 2017, Shanghai, China, September 18-19, 2017*, pages 43–53. IEEE, 2017.

[26] Mirza Klimenta and Ulrik Brandes. Graph drawing by classical multidimensional scaling: New perspectives. In Walter Didimo and Maurizio Patrignani, editors, *Graph Drawing - 20th International Symposium, GD 2012, Redmond, WA, USA, September 19-21, 2012, Revised Selected Papers*, volume 7704 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2013.

[27] J. Krinke. A study of consistent and inconsistent changes to code clones. In *WCRE*, pages 170 – 178, 2007.

[28] J. Krinke. Is cloned code more stable than non-cloned code? In *SCAM*, pages 57 – 66, 2008.

[29] J. Krinke. Is cloned code older than non-cloned code? In *IWSC*, pages 28 – 33, 2011.

[30] J. Li and M. D. Ernst. Cbcd: Cloned buggy code detector. In *ICSE*, pages 310 – 320, 2012.

[31] Lauro Didier Lins, James T. Klosowski, and Carlos Eduardo Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2456–2465, 2013.

[32] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. *imMens*: Real-time visual querying of big data. *Comput. Graph. Forum*, 32(3):421–430, 2013.

[33] A. Lozano and M. Wermelinger. Assessing the effect of clones on changeability. In *ICSM*, pages 227 – 236, 2008.

[34] A. Lozano and M. Wermelinger. Tracking clones' imprint. In *IWSC*, pages 65 – 72, 2010.

[35] Debajyoti Mondal and Lev Nachmanson. A new approach to graphmaps, a system browsing large graphs as interactive maps. In Alexandru Telea, Andreas Kerren, and José Braz, editors, *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2018) - Volume 3: IVAPP, Funchal, Madeira, Portugal, January 27-29, 2018.*, pages 108–119. SciTePress, 2018.

[36] M. Mondal, C. K. Roy, M. S. Rahman, R. K. Saha, J. Krinke, and K. A. Schneider. Comparative stability of cloned and non-cloned code: An empirical study. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC'12)*, pages 1227 – 1234, 2012.

[37] M. Mondal, C. K. Roy, and K. A. Schneider. An empirical study on clone stability. *ACM SIGAPP Applied Computing Review*, 12(3): 20 – 36, 2012.

[38] M. Mondal, C. K. Roy, and K. A. Schneider. Automatic identification of important clones for refactoring and tracking. In *Proceedings of the IEEE 14th International Working Conference on Source Code Analysis and Manipulation (SCAM'14)*, pages 11 – 20, 2014.

[39] M. Mondal, C. K. Roy, and K. A. Schneider. Automatic ranking of clones for refactoring through mining association rules. In *Proceedings of the IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE'14), Software Evolution Week*, pages 114 – 123, 2014.

[40] M. Mondal, C. K. Roy, and K. A. Schneider. SPCP-Miner: A tool for mining code clones that are important for refactoring or tracking. In *Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)*, pages 482 – 486, 2015.

[41] Lev Nachmanson, Roman Prutkin, Bongshin Lee, Nathalie Henry Riche, Alexander E. Holroyd, and Xiaoji Chen. Graphmaps: Browsing large graphs as interactive maps. In *GD*, volume 9411 of *LNCS*, pages 3–15. Springer, 2015.

[42] Roy Oberhauser. ViSiTR: 3D visualization for code visitation trail recommendations. *International Journal on Advances in Software*, 10(1&2):46–56, 2017.

[43] Yongjoo Park, Michael J. Cafarella, and Barzan Mozafari. Visualization-aware sampling for very large databases. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 755–766. IEEE Computer Society, 2016.

[44] Yonathan Perez, Rok Sosic, Arijit Banerjee, Rohan Puttagunta, Martin Raison, Pararth Shah, and Jure Leskovec. Ringo: Interactive graph analytics on big-memory machines. In *ACM SIGMOD*, pages 1105–1110. ACM, 2015.

[45] C. K. Roy. Detection and analysis of near-miss software clones. In *Proceedings of the Doctoral Symposium Track of the 25th IEEE International Conference on Software Maintenance (ICSM'09)*, pages 447 – 450, 2009.

[46] C. K. Roy and J. R. Cordy. NICAD: accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Proceedings of the 16th IEEE International Conference on Program Comprehension (ICPC'08)*, pages 172 – 181, 2008.

[47] C. K. Roy and J. R. Cordy. A mutation / injection-based automatic framework for evaluating code clone detection tools. In *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops (Mutation'09)*, pages 157 – 166, 2009.

[48] C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming Journal*, 74 (2009): 470 – 495, 2009.

[49] C. K. Roy, M. F. Zibran, and R. Koschke. The vision of software clone management: Past, present, and future (keynote paper). In *Proceedings of the IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE'14), Software Evolution Week*, pages 18 – 33, 2014.

[50] Chanchal K. Roy and James R. Cordy. Near-miss function clones in open source software: an empirical study. *Journal of Software Maintenance*, 22(3):165–189, 2010.

[51] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages, Boulder, Colorado, USA, September 3-6, 1996*, pages 336–343. IEEE Computer Society, 1996.

[52] J. Svajlenko and C. K. Roy. Evaluating modern clone detection tools. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME'14)*, pages 321 – 330, 2014.

[53] S. Thummalapenta, L. Cerulo, L. Aversano, and M. D. Penta. An empirical study on the maintenance of source code clones. *Empirical Software Engineering Journal*, 15(1): 1 – 34, 2009.

[54] Juraj Vincur, Pavol Návrat, and Ivan Polásek. VR city: Software analysis in virtual reality environment. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS-C 2017, Prague, Czech Republic, July 25-29, 2017*, pages 509–516. IEEE, 2017.

[55] Richard Wettel. Visual exploration of large-scale evolving software. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Companion Volume*, pages 391–394. IEEE, 2009.

[56] Richard Wettel and Michele Lanza. Codecity: 3d visualization of large-scale software. In Wilhelm Schäfer, Matthew B. Dwyer, and Volker Gruhn, editors, *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008, Companion Volume*, pages 921–922. ACM, 2008.

[57] Hsiang-Yun Wu, Shigeo Takahashi, Chun-Cheng Lin, and Hsu-Chun Yen. A zone-based approach for placing annotation labels on metro

maps. In Lutz Dickmann, Gerald Volkmann, Rainer Malaka, Susanne Boll, Antonio Krüger, and Patrick Olivier, editors, *Smart Graphics - 11th International Symposium, SG 2011, Bremen, Germany, July 18-20, 2011. Proceedings*, volume 6815 of *Lecture Notes in Computer Science*, pages 91–102. Springer, 2011.

[58] Michael Zinsmaier, Ulrik Brandes, Oliver Deussen, and Hendrik Strobelt. Interactive level-of-detail rendering of large graphs. *IEEE TVCG*, 18(12):2486–2495, 2012.