Visual Clone Analysis with SolidSDD

Lucian Voinea
SolidSource BV, Eindhoven, the Netherlands
Email: lucian.voinea@gmail.com

Alexandru Telea
University of Groningen, Groningen, the Netherlands
Email: a.c.telea@rug.nl

***Abstract*—We present SolidSDD, an integrated tool for the extraction and visual analysis of code clones. SolidSDD aims to simplify and speed up the entire process of clone extraction from code bases written in C, C++, Java, and C#, and visual analysis of the extracted results. To this end, we combine several scalable visualization techniques such as hierarchical edge bundles, table lenses, annotated text views, and linked views. We demonstrate SolidSDD for both fine-grained clone analysis and aggregated report production tasks on several large-scale code bases.**

***Keywords*-clone analysis, clone visualization, bundled graphs**

I. Introduction

Code duplication (clone) detection is a key tool in software maintenance. Many clone detectors exist, *e.g.* text-based [1], [2], [3], token-based [4], [5], syntax-tree-based [6], [7], metric-based [8], [9], and hybrid [10], [11]. Such tools produce a large amount of clone pairs annotated with line, file, clone type, clone metric, and program structure facts. To understand such data, visualization techniques are crucial [12]. Examples hereof are flat lists [13], scatterplots [14], Hasse diagrams [15], node-link views [11], hyperlinked text and metric graphs [4], and polymetric views [2]. This advocates the construction of tools that integrate fast-and-easy clone detection with a rich-and-intuitive multilevel navigation of the detected clones.

In line with the above, we developed SolidSDD (Software Duplication Detector). SolidSDD finds structural clones from C, C++, C# and Java code by a token-based method similar to [4]. Detection is configurable by clone length (in statements), identifier renaming (allowed or not), gap size (inserted or deleted code fragments in a clone), and whitespace and comment filtering. Detection creates a *compound duplication graph*, stored in an SQLite database. Nodes are cloned code fragments. Edges are clone relations. Structure is added either from the code directory data (default) or from a syntax-based code hierarchy [16]. Metrics are computed on nodes (code location) and edges (cloned code percentage, number of distinct clones, and if a clone uses identifier renaming). For analysis, we use several scalable information visualization techniques: Annotated text shows clones in their file context and allows navigating between all pairs of a clone. Bundled graphs show clones *vs* system structure. Table lenses show clone and file metrics. Views are linked, so we can navigate between text, clones, and system-structure. Our tool supports the following questions: (a) How are clones distributed *vs* system structure? (b) Which subsystems have high clone percentages? (c) What kind of clones does a given file contain? (d) Which files are affected by a given clone? These are explained next.

II. Clone Visualization

We illustrate SolidSDD (Fig. 1) by clones extracted from the well-known Visualization Toolkit (VTK), a class library for data visualization (3163 C++ files, 1113 C files, 3193 headers, 2.9 MLOC [17]). On VTK version 5.8, SolidSDD found 1124 clones in 220 seconds on a 2.66 GHz PC with 4 GB RAM, using the default tool settings.

Structure view: This view (Fig. 1 a,b) uses hierarchical edge bundling (HEB [18]) to show clones atop of system structure. Its design is similar with the ClonEvol tool [16]. The radial rings show the system hierarchy (folders and files). Node colors show the percentage of cloned code in a subsystem on a green-to-red colormap (green=smallest cloned-code amount; red=maximal cloned-code amount; gray=files with no clones). Edges show aggregated clone relations between files (two files are linked when they share at least one clone), routed along the system structure, so high-level clone relations show up as bundles. Edge colors show the percentage of cloned code with respect to the smaller size of the file-pair they connect on a green-to-red colormap, *i.e.*, red edges show fully-cloned files. Subsystems can be opened or collapsed by double-clicking their nodes; and nodes and edges can be selected, to focus analysis on a specific set of files or clones. Nodes (files) and clone relations (edges) can be filtered based on metrics, *e.g.* to eliminate clones with few instances or files with small cloned-code amounts.

The structure view helps finding subsystems (strongly) linked by clones and/or having high cloned-code percentages. Fig. 1 a shows that our VTK code base has many intra-system clones (edges linking files in the same folder) but also some inter-system clones (edge linking files in different folders). Three subsystems have high clone percentages (red in Fig. 1 a): *examples* (S_1), *bin* (S_2) and *Filtering* (S_3). Browsing these, we found that clones in S_1 and S_2 are in tutorial code and test drivers, likely created by copy-paste. Such clones are not very interesting for *e.g.* refactoring as this would arguably make sample code harder to read and learn from. Clones in S_3, a core VTK subsystem, are more interesting. In Fig. 1 b, we zoom on S_3 and select the file vtkGenericDataSetAlgorithm (f, marked black) having over 50% cloned code. This is the baseclass of all VTK algorithms. Selecting f highlights the files f_c sharing clones with f, *i.e.* the *clone pairs* of f. We find five such clone pairs in the same S_3 subsystem: vtkStructuredGridAlgorithm, vtkDataObjectAlgorithm, vtkUnstructuredGridAlgorithm, vtkHyperOctreeAlgorithm, vtkPolyDataAlgorithm (Fig. 1 b). Less expectedly, we find one extra clone pair in the *Rendering* subsystem (vtkLabelHierarchyAlgorithm, marked g in Fig. 1 b). Each such file contains a separate class, as standard in VTK. When writing these subclasses, developers likely copy-pasted code between the baseclass and subclasses and/or between sibling subclasses. Given VTK's coding guidelines to maximize code reuse *and* keep subsystems independent, clone g is a good refactoring candidate, *e.g.* by moving the common algorithm part to a superclass.[1]

Detail views: SolidSDD offers four levels-of-detail to inspect clones (Fig. 1). These implement the "overview first, zoom and

[1]This clone was removed in a recent VTK release, whose commit log describes precisely the above design violation and proposed refactoring.

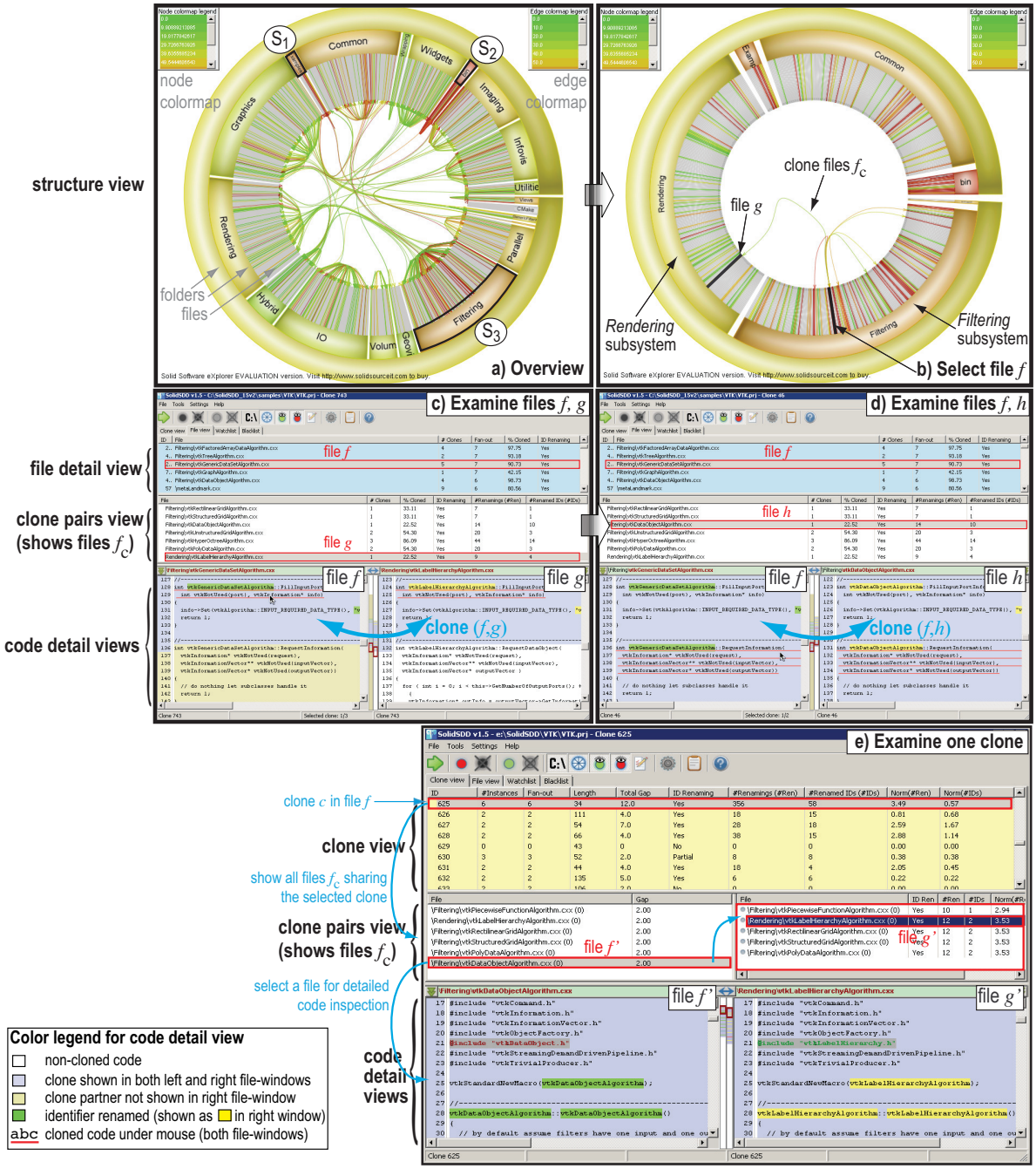


Figure 1. SolidSDD clone visualization with the structure view (a,b) and detail views (c-e). Arrows show interactive workflow with the views (see Sec. II).

filter, then details-on-demand" visualization mantra of Shneiderman [19], thus making SolidSDD more versatile than structure-view-only tools using HEB, such as *e.g.* [16], [20]. The *file detail* view shows all files with cloned-code percentage, number of clones, and presence of identifier renaming. Sorting this table helps *e.g.* finding files with most clones or highest cloned code percentage. This view is linked with the structure view: selecting file f in the structure view (Fig. 1 b, black) highlights it in this table (Figs. 1 c,d, red marker). The *clone-pair* view shows the clone-pair files f_c of f. Here we find our file g which shares clones with f but is in another subsystem. We select g and use the two *code detail* views (Fig. 1 b, bottom panels) to study all clones between f and g. The left code-detail view shows code in file f; the right view shows code in file g. Scrolling these views is synchronized to easily compare matching code fragments. Text is color-coded: non-cloned code (white), code in f cloned in g (light blue), renamed identifier pairs (green in left view, yellow in right view), and cloned code in f whose clones are in some other file $h \neq g$ (beige). The last color helps us to navigate from f to other clone-pair files h: Control-clicking on a beige code-fragment in f (Fig. 1 c) replaces file g in the right view by file h, and also selects h in the clone-pairs view (Fig. 1 d). Repeating this allows us to cycle, for a given clone instance in a given file f, through all its clone-pair files h.

The previous three views offer a file-centric clone exploration. The *clone view* offers a clone-centric perspective, *i.e.*, inspect a clone over *all* files where it occurs (Fig. 1 e). The top list in this view shows all clones in the entire code base. Selecting a clone c, either in this list or by clicking in the code view, shows all files f_c where the clone occurs in the

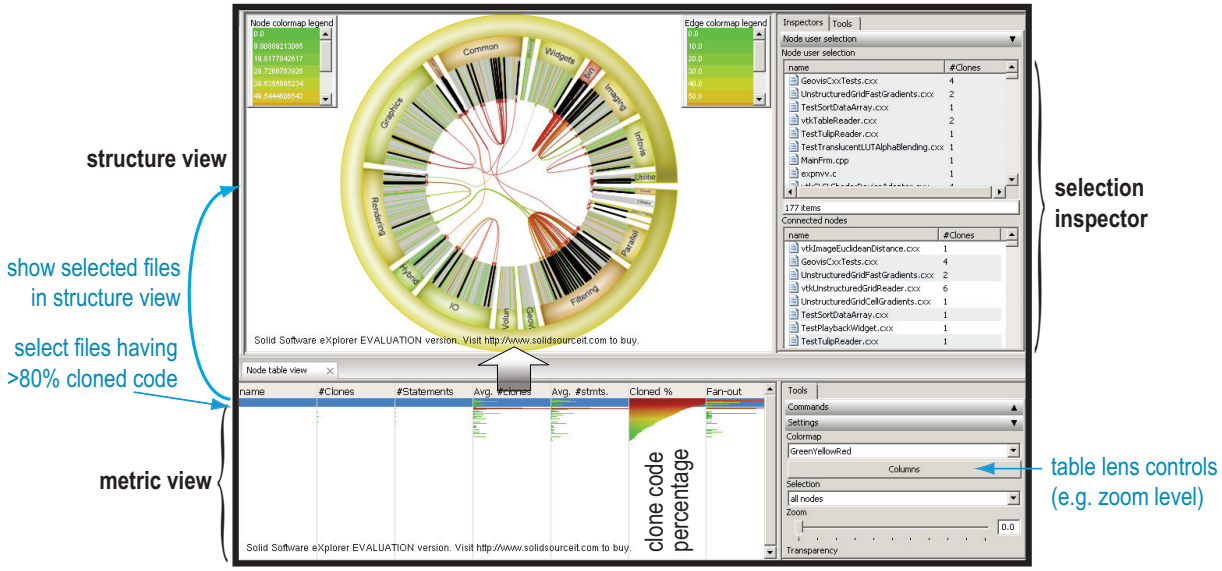


Figure 2. SolidSDD metric view (bottom) and structure view (top) with high-clone percentage files highlighted (see Sec. II).

clone-view middle panel. In Fig. 1 e we selected a clone in the vtkGenericDataSetAlgorithm file. We now select a file $f' \in f_c$ in the middle panel which contains a *subclass* of vtkGenericDataSetAlgorithm. The middle panel right view shows now all files f'_c sharing clones with f'. We see here sibling subclasses of vtkGenericDataSetAlgorithm. Hence, *both* our hypotheses are true: Clones exist both between the superclass and subclasses and between subclasses themselves.

Metric view: This view (Fig. 2 bottom) uses a table where rows are code files, and columns are file attributes: name, size, clone code percentage, and clone fan-in (number of intra-file clones) and fan-out (number of inter-file clones). Tables can be sorted by clicking on their column headers. A zoom slider allows changing the level of detail, using the table lens technique [21]: When zooming out, table cells are shrunk; if their size goes below a few pixels, cell text is replaced by bars colored and scaled to show metric values. When zoomed out and sorted, the table shows the spread of a given metric on the entire system. Selecting row-ranges allows focusing analysis on a subset of interest. In Fig. 2, we sort files by cloned-code percentage and zoomed out the table to see a distribution of this metric (sixth column from right). We see that around 30% of all files contain a large amount of cloning. Selecting the top 10% files in the metric view highlights all files with >80% cloned code in the structure view in black (Fig. 2 top). We see that every single VTK subsystem contains such files. Details on the selected files are shown in the selection inspector (Fig. 2 right). This supports refactoring planning by giving insight on where recoding work would need to be done to *e.g.* remove the largest clones.

III. Aggregated Clone Inspection

SolidSDD is written in Python (clone extraction) and C++ with OpenGL (visualization) and SQLite (clone storage). The HEB is implemented following [18]. Architectural details are given in [22]. An installer with manual and sample datasets is freely available for researchers [23]. SolidSDD can be customized in various ways: (a) The structure-view hierarchy (Sec. II) can be specified as an XML file; (b) Custom line-based filters to exclude parts of the scanned code can be written as Python scripts; (c) The tool can be run in batch-mode (without the visual front-end) and its analysis results can be exported in various formats, *e.g.* CSV and XML. (d) Users can mark certain clones as uninteresting (black-listed) or of interest (watchlist). When running SolidSDD on newer versions of the same code, the tool hides clones on the blacklist and highlights clones on the watchlist. This allows monitoring the evolution of *interesting* clone relations in a code base in a simpler way than *e.g.* the animation-based approach proposed by the ClonEvol tool [16].

Besides fine-grained clone inspection (Sec. II), the IT industry also needs creating aggregated clone *reports*. To test SolidSDD's scalability, robustness, and ease-of-use for this task, we selected a set of open-source applications within six classes: office, enterprise/financial, databases, communication, networking/embedded, and development tools. In each class, we selected three popular samples, based on their download count on SourceForge.net and Google rank. The samples are written in C, C++, C#, and Java (Tab. I), and have between 30K and 1.8M LOC (Fig. 3 a). For each sample, SolidSDD was used to measure code-duplication amount and potential size-decrease by refactoring the top 5% largest clones. These are typical measurements often present in aggregated clone reports [4]. Other clone-related metrics can be considered equally easily. Cloning was measured for high ($>$ 35 statements), medium ($>$ 25 statements) and low clone sizes ($>$ 15 statements). These metrics, exported from SolidSDD to Excel (Fig. 3 b), show that the cloning amount is low in four of the six considered classes: communication, development, office, and databases (under 5% at medium..high clone sizes). Two classes show larger deviations: networking/embedded ($>$ 10% cloning) and enterprise/financial ($>$ 25% cloning). Figure 3 c shows the potential code reduction upon refactoring the top 5% largest clones, *i.e.*, removal of all but one copy of each clone. For most of our samples, the top 5% largest clones account for more than 25% of all cloning (reaching 40% for networking/embedded). This study took under 5 hours, including code download, installing and running SolidSDD from scratch, and making the reports. This shows that SolidSDD is fast and easy to use to create clone reports for a wide range of code bases.

IV. Conclusions

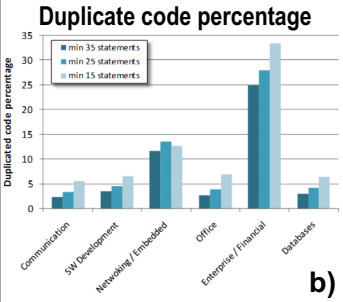
We have presented SolidSDD, an integrated tool for the detection and exploration of code clones. SolidSDD combines

Application class	Class description and selected applications for clone inspection
Communication	File-transfer and P2P applications (Apache httpd, FileZilla, Vuze)
Office	Productivity applications (KOffice, AbiWord, Scribus)
Entreprise/financial	Company resource planning / CRM (Jasper Reports, Compiere, ADempiere)
Databases	Database engines (MySQL, SQLite, Hibernate)
Networking/embedded	Network management/monitoring (Freesco, Whireshark, net-snmp)
Development tools	Software development and maintenance (TortoiseSVN, Notepad++, WinMerge)

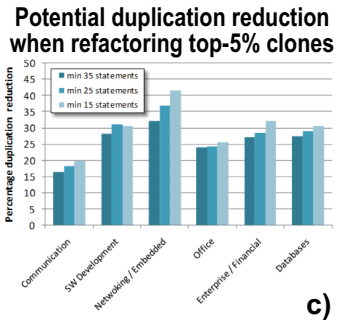
Table I
OPEN-SOURCE APPLICATIONS SELECTED FOR SOLIDSDD TOOL ASSESSMENT (SEC. III).

a)

	min 35 statements		min 25 statements		min 15 statements		Size	
Project	Clone %	Gain %	Clone %	Gain %	Clone %	Gain %	#files	LOC
Communication								
Apache httpd	4.25	13.1	5.7	16.03	6.84	20.32	1040	360156
Filezilla	0.53	18.69	0.8	18.53	2.19	17.22	355	110726
Vuze	2.12	17.48	3.63	20.51	7.61	21.9	3226	472622
Average	2.30	16.42	3.38	18.36	5.55	19.81		
SW Development								
Notepad++	2.56	18.4	3.74	19.96	6.5	23.51	334	111042
TortoiseSVN	2.36	47.92	3.27	40.58	5.16	35.47	2623	538726
WinMerge	5.46	17.89	6.34	32.74	8.12	32.54	799	160631
Average	3.46	28.07	4.45	31.09	6.59	30.51		
Netwoking / Embedded								
Net-snmp	8.71	13.9	10.65	18.56	13.79	25.05	1007	221196
Freesco	8.51	40.48	9.48	50.18	11.34	57.77	2404	697021
Wireshark	17.59	42.28	20.45	41.55			2392	1890721
Average	11.60	32.22	13.53	36.76	12.57	41.41		
Office								
Abiword	2.03	25.86	3.17	24.39	6.14	24.91	1970	458369
Koffice	2.5	29.31	3.69	28.68	7.59	27.51	5777	643115
Scribus	3.43	16.59	4.56	19.49	7	24.32	1152	264986
Average	2.65	23.92	3.81	24.19	6.91	25.58		
Enterprise / Financial								
JasperReports	17.24	17.52	20.42	19.41	27.54	34.01	1454	175433
Compiere	26.88	41.63	30.29	43.02	35.73	39.51	2481	398107
Adempiere	30.35	21.9	32.84	23.1	36.89	22.93	4047	617680
Average	24.82	27.02	27.85	28.51	33.39	32.15		
Databases								
MySql	2.01	40.13	2.48	40.51	3.66	36.89	2114	961661
SQLLite	4.57	25.9	4.7	31.05	5.64	37.9	170	99185
Hibernate	2.56	15.84	5.47	15.28	9.92	16.98	500	30567
Average	3.05	27.29	4.22	28.95	6.41	30.59		



b)



c)

Figure 3. Producing aggregate clone-amount and clone-reduction potential reports using SolidSDD (Sec. III).

several techniques for clone detection [4] and data visualization (table lenses, HEB views, linked views, and annotated text) to make detection, examination, and quantification of clones faster and easier than when using several separate analysis and visualization tools. We illustrate SolidSDD with both a fine-grained clone analysis (Sec. II) and coarse-grained aggregated analysis of several large code bases from the open-source arena.

REFERENCES

[1] B. Baker, "A program for identifying duplicated code," *Computing Science and Statistics*, vol. 24, pp. 49–57, 1993.

[2] S. Ducasse, M. Rieger, and S. Demeyer, "A language independent approach for detecting duplicated code," in *Proc. ICSM*, 1999, pp. 109–118.

[3] A. Marcus and J. Maletic, "Identification of high-level concept clones in source code," in *Proc. ASE*, 2001, pp. 107–112.

[4] T. Kamiya, "CCfinderX official site," 2014, www.ccfinder.net.

[5] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: A tool for finding copy-paste and related bugs in operating system code," in *Proc. OSDI*, 2004, pp. 289–302.

[6] I. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *Proc. ICSM*, 1998, pp. 368–377.

[7] W. Evans, C. Fraser, and F. Ma, "Clone detection via structural abstraction," in *Proc. WCRE*, 2007, pp. 150–159.

[8] K. Kontogiannis, R. Demori, E. Merlo, M. Gallery, and M. Bernstein, "Pattern matching for clone and concept detection," in *Proc. ASE*, 1996, pp. 77–108.

[9] F. Calefato, F. Lanubile, and T. Mallardo, "Function clone detection in web applications: a semiautomated approach," *J. Web Eng.*, vol. 3, no. 1, pp. 3–21, 2004.

[10] R. Koschke, R. Falke, and P. Frenzel, "Clone detection using abstract syntax suffix trees," in *Proc. WCRE*, 2006, pp. 253–262.

[11] L. Jiang, G. Miserghi, Z. Su, and S. Glondu, "Deckard: Scalable and accurate tree-based detection of code clones," in *Proc. ICSE*, 2007, pp. 137–145.

[12] C. Roy, J. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci Comp Program*, vol. 74, no. 7, pp. 470–495, 2009.

[13] E. Juergens, F. Deissenboeck, and B. Hummel, "CloneDetective: a workbench for clone detection research," in *Proc. ICSE*, 2010, pp. 98–107.

[14] Y. Higo and S. Kusumoto, "Code clone detection on specialized PDGs with heuristics," in *Proc. CSMR*, 2011, pp. 75–84.

[15] J. Johnson, "Visualizing textual redundancy in legacy source," in *Proc. CASCON*, 1994, pp. 32–38.

[16] A. Hanjalic, "ClonEvol: Visualizing software evolution with code clones," in *Proc. Vissoft*, 2013, pp. 1–4.

[17] Kitware, "VTK home page," 2013, www.kitware.com/vtk.

[18] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE TVCG*, vol. 12, no. 5, pp. 741–748, 2006.

[19] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Proc. ACM VL*, 1996, pp. 336–343.

[20] B. Hauptmann, V. Bauer, and M. Junker, "Using edge bundle views for clone visualization," in *Proc. IWSC*, 2012, pp. 86–87.

[21] A. Telea, "Combining extended table lens and treemap techniques for visualizing tabular data," in *Proc. EuroVis*, 2006, pp. 51–58.

[22] D. Reniers, L. Voinea, O. Ersoy, and A. Telea, "The Solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product," *Science of Computer Programming*, vol. 79, no. 1, pp. 224–240, 2014.

[23] SolidSource BV, "SolidSDD clone detector," 2014, www.solidsourceit.com/products.