Visual augmentation of source code editors: A systematic mapping study

Matúš Sulír*, Michaela Bačíková, Sergej Chodarev, Jaroslav Porubän

Technical University of Košice, Letná 9, 042 00 Košice, Slovakia

Abstract

Source code written in textual programming languages is typically edited in integrated development environments or specialized code editors. These tools often display various visual items, such as icons, color highlights or more advanced graphical overlays directly in the main editable source code view. We call such visualizations source code editor augmentation.

In this paper, we present a first systematic mapping study of source code editor augmentation tools and approaches. We manually reviewed the metadata of 5,553 articles published during the last twenty years in two phases – keyword search and references search. The result is a list of 103 relevant articles and a taxonomy of source code editor augmentation tools with seven dimensions, which we used to categorize the resulting list of the surveyed articles.

We also provide the definition of the term source code editor augmentation, along with a brief overview of historical development and augmentations available in current industrial IDEs.

Keywords: source code editor augmentation, integrated development environment (IDE), in situ visualization, systematic review, survey

1. Introduction

Despite decades of research effort, visual programming languages have not obtained a widespread industrial adoption. The majority of programmers write their programs using traditional, textual programming languages [1]. They use a standalone text editor or an integrated development environment (IDE) whose most important component is a textual source code editor. However, upon a closer look at these "plain text" editors, we can discover many visual features: from simple syntax highlighting, through underlining of the code violating code conventions, to information about last version control commits in the left margin. We call such visualizations *source code editor augmentation.*

The main purpose of visual source code editor augmentation is to enable spatial immediacy [2], i.e. to lower the on-screen distance between the related objects or events. For example, instead of reading the line number of the syntax error and manually navigating to it, the IDE notifies us about its occurrence by underlining the erroneous code directly in the editor.

*Corresponding author

Email addresses: `matus.sulir@tuke.sk` (Matúš Sulír), `michaela.bacikova@tuke.sk` (Michaela Bačíková), `sergej.chodarev@tuke.sk` (Sergej Chodarev), `jaroslav.poruban@tuke.sk` (Jaroslav Porubän)

There exist surveys about software visualization in general [3] and about its various subfields, such as architecture [4], algorithm [5] or awareness visualization [6]. Maletic et al. [7] presented a task-oriented taxonomy of software visualization. Sutherland et al. [8] review ink annotations of digital documents, including program code. Recent systematic mapping studies of other fields related to computer languages include domain-specific languages [9] and template-based code generation [10]. In our previous work, we surveyed the assignment of metadata to different parts of source code [11] and included a discussion about the presentation of source code annotations (inside or outside the code). Nevertheless, according to our knowledge, there is no survey available specifically about visual augmentation of source code editors. In this paper, we present a *first systematic mapping study of source code editor augmentation approaches and tools*, analyzing research papers published during the last twenty years.

In section 2, we will define the term source code editor augmentation, provide a brief overview of historical development in this area and mention several augmentations available in current industrial IDEs. We will describe the method used for the systematic mapping study in section 3. The result of our survey is a taxonomy with seven dimensions (section 4) and a categorized list of the surveyed articles (section 5).

2. Source code editor augmentation

In the research area of virtual reality, an augmented reality system "supplements the real world with virtual objects" [12]. We would like to apply this terminology to the area of source code editing. In our case, the "real world" is represented by the textual source code stored in a file and displayed in a text editor as-is. The "virtual objects" are various line decorations, icons, coloring, images, additional textual labels and other visual overlays.

Therefore, we define the term *visual source code augmentation* as an approach which displays additional graphical or textual information directly in plain-text source code. If the code is editable, we can call this also *source code editor augmentation.*

Let us make the definition more precise. First, note that the raw source code displayed in the editor must be the same as the text stored in a file. Projectional editors [13] that use a different editable (visual) and storage representation are, therefore, out of the scope of this article. Furthermore, the augmentation must not remove any part of the displayed source code – this is analogous to the term "diminished reality" in the area of virtual reality[1].

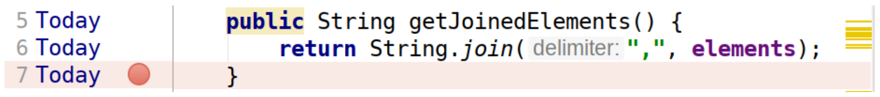
Second, by the expression "directly in the code editor", we mean also the left and right margin of the editor. Note that many IDEs offer many additional views, such as a package explorer or a class navigator. These views are clearly outside the scope of source code editor augmentation.

Alternative terms to "source code editor augmentation" are *in situ* software visualization [14] and source code *annotation* [15]. The latter can be easily confused with attribute-oriented programming (e.g., Java annotations [16]). We decided to use the term *augmentation*, particularly for its correspondence with an existing terminology in the field of computer graphics.

2.1. Historical view

Probably the most rudimentary source code editor augmentation feature is *syntax highlighting* (coloring), where each lexical unit is highlighted with a specific color and/or font weight according

[1]While some researchers consider diminished reality a subset of augmented reality [12], we decided to separate these two cases.



```
5 Today        public String getJoinedElements() {
6 Today            return String.join( delimiter: ",", elements);
7 Today ●      }
```

Figure 1: A preview of selected augmentation features in an industrial IDE, IntelliJ IDEA.

to its type. One of the first editors supporting real-time source code highlighting was the LEXX editor [17], developed in the eighties.

Another useful augmentation feature is *immediate* syntax error *feedback*, pioneered by the Magpie system [18], which incrementally compiled the program being written. Erroneous code fragments were highlighted directly in the editor.

One of the examples from the 90's is ZStep95 [2]. The expression being currently evaluated was *highlighted* with a border directly in the editor. This augmentation was interactive: a graphical output produced by this expression was displayed in a floating window located next to such a border.

During the last two decades, the advances in computer performance and incremental analysis algorithms enabled the integration of a multitude of augmentation features into mainstream IDEs.

2.2. Augmentation in industrial IDEs

In modern industrial IDEs, visual augmentations are almost omnipresent. For an illustration, Figure 1 displays a small selection of them. First, we can see a textual description of the last commit date of each line in the left margin. There is also an icon representing a breakpoint on line 7, which is highlighted also by the red background color. Next, the word "public" is highlighted because of a code inspection warning. The parameter name label "delimiter:" is only a visual augmentation drawn by the IDE in addition to the code itself. Finally, in the right margin, there is a mini-map of warnings (yellow) in the whole file.

In this section, we will provide an overview of selected visual augmentation possibilities available in current industrial IDEs. We chose the following representatives for the analysis: IntelliJ IDEA 2017.3.5, JetBrains WebStorm 181.4203.9, Visual Studio 2017, Eclipse Oxygen.2 Release (4.7.2) and NetBeans 8.2. All of them except WebStorm are listed in the top 10 used IDEs worldwide[2] and each is regularly used by at least one of the authors of this paper.

A common augmentation in industrial IDEs is the *highlighting* of the current places of interest: a currently evaluated language construct, currently edited line, a paired bracket, all occurrences of the currently refactored language construct in code, etc. Usually, text background color highlighting is used for such features. In the currently edited area, *code indentation* is often marked by vertical lines visually connecting the indented regions in code.

Another large category that has become almost as standard as the previous ones are *hints and warnings*. A light-gray font or text strike-through is applied on unused code, grammar typos or deprecated code. Errors and warnings are marked also with a standard interactive icon in the left editor margin. Hints standardly use orange or yellow and warnings use red color, both for text decoration and for the icons. This kind of augmentation was already used in Eclipse 1.0 in 2001. *Language-specific items*, references and dependencies or another kind of meta-information such as

[2]https://pypl.github.io/IDE.html (accessed in April 2018)

bookmarks, overriding, implementing an interface or recursive calls leverage from the same notation mechanism but usually with a specific icon in the left margin.

All occurrences of different kinds of information, including the aforementioned ones, are usually also marked by clickable, thin color markers in the *right editor margin*, so the programmer is able to quickly navigate to the specific location in a long source code file. Some IDEs also display a document analysis results icon on top of the right margin.

Apart from the details of visual representation, *debugging* augmentation remained pretty much the same historically. One more recent advance is, for example, the displaying of values currently stored in variables directly in code.

Industrial programming today is all about teamwork, which does not come easy without versioning support – another standardized feature in common IDEs. Changes in the current version are usually marked by icons or color regions in the left editor margin, e.g. *JetBrains* products use line marking of three types: 1.) a green-colored region for added lines, 2.) a blue-colored region for edited lines and 3.) grey triangle icons for deleted code sections. Eclipse uses a similar but visually less notable visualization. Visual Studio, however, uses a different, more recent approach: interactive team activity info markers *on top of code lines.*

Many recent advances in source code editor augmentation emerge mainly in connection with a specific programming language or technology. Considering the priority of important features such as syntax highlighting, they usually include simpler and usually less notable visualizations such as: underlined text for *HTTP links* used in code (Visual Studio); a *run icon* in the left editor margin, usually placed next to a main class, main method, or a runnable script (JetBrains products); color coding of HTML tags background according to their *level of nesting*; or colored rectangles in left margin next to *color codes* used in CSS-based languages. Recently, IntelliJ IDEA introduced smaller annotations such as displaying *method parameter names* in method calls, marked by a light-grey text before a parameter value (Figure 1, line 6).

3. Method

Now we will describe the method used for a systematic mapping study we conducted. A systematic mapping study is a form of a systematic review with more general research questions, aiming to provide an overview of the given research field.

3.1. Research questions

We were interested in the following two research questions:

- **RQ1:** What source code editor augmentation tools are described in the literature?
- **RQ2:** How can they be categorized?

3.2. Selection criteria

Inclusion and exclusion criteria are an important part of every systematic review. In our study, all included articles must fulfill the following criteria:

- It is a journal or conference article published between years 1998–2017 (inclusive).
- It presents a new tool (or significantly extends an existing one) – e.g., a desktop application, web application or an IDE/editor plugin.

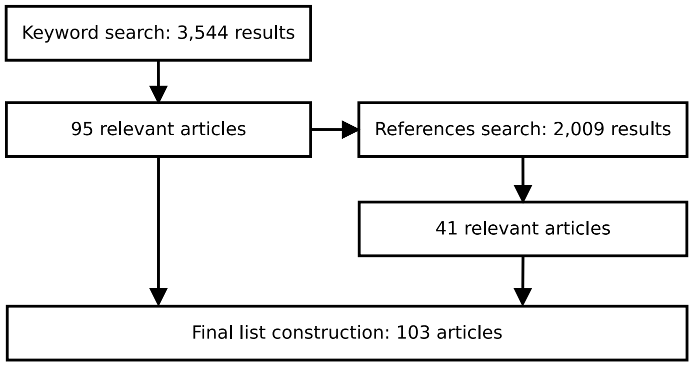

Keyword search: 3,544 results
95 relevant articles
References search: 2,009 results
41 relevant articles
Final list construction: 103 articles

Figure 2: A systematic survey search process and article selection overview

- The tool visually augments the source code editor.
- The article contains a clear textual and graphical description (picture) of the augmentation.

At the same time, articles meeting any of the following criteria were excluded:

- Secondary and tertiary studies, items such as keynotes and editorials, articles for which we could not access a full text.
- Articles describing graphical and semi-graphical languages.
- Projectional editing, hiding and replacing existing code with other information.
- Tools displaying only temporary pop-up windows above code (code completion, tooltips).
- Standard augmentation present in almost all IDEs (syntax highlighting, standard debugging support, syntax error reporting).

3.3. Search strategy

To obtain a list of potentially relevant articles, we used a combination of keyword search in digital libraries, forward and backward references search. For an overview of the article search and selection process, see Figure 2.

3.3.1. Keyword search

First, we performed a keyword search in digital libraries to produce a list of potentially relevant articles. The search query was constructed according to the principles suggested by Brereton et al. [19]: For each element of the study topic (software maintenance, IDE, source code editor, visualization, augmentation), we found synonyms and related words, connected them by the logical "OR" operator and then connected the subqueries by the "AND" operator. The resulting query is:

```
(software OR program) AND (maintenance OR comprehension OR understanding)
AND (tool OR "development environment")
AND ("source code" OR "source file") AND editor
AND (visualize OR visualization OR display)
AND (augment OR augmentation OR annotate OR annotation)
```

This query was entered into advanced search boxes of four digital libraries: IEEE Xplore[3], ACM Digital Library (DL)[4], ScienceDirect[5] and Springer Link[6]. These four libraries were chosen because of their popularity in software engineering research [20]. In the case of ACM DL, we had to accommodate the query to use a different syntax (without changing its semantics). The search was performed on metadata and full texts. We limited the search to the field of computer science, using the possibilities available in individual libraries, e.g., in IEEE Xplore, we excluded articles not contained in the Computer Society DL. We also filtered the entry types to journal/conference articles with a full text available and the publication year to 1998–2017, written in English.

For all found entries we exported metadata such as titles and abstracts. If the given library did not offer the export of abstracts, we downloaded them from Scopus[7]. After removing duplicate items, this phase produced a list of 3,544 articles.

Then, three of the authors each were given a subset of the articles to evaluate. The researchers manually decided which of these articles fulfill the selection criteria – first based on the title and abstract review and then, if the article seemed potentially relevant, by skimming the full text. The result was a list of 95 selected articles.

3.3.2. References search

Next, we tried to find relevant articles by examining the references of the 95 articles identified in the previous phase (keyword search). We were interested both in backward references (literature cited in these articles) and forward references (newer papers citing these articles). To export the lists of backward and forward references along with all necessary metadata, we used Scopus. Similarly to the previous phase, we limited the search to English computer science journal/conference articles published between 1998 and 2017, whenever these options were available.

Two of the articles were not indexed in Scopus and for these, we extracted the backward references manually from the full texts; forward references were exported from the digital libraries of the articles' origin (ACM DL and ScienceDirect).

We merged duplicate entries between forward and backward reference lists. Next, we removed papers already present in the keyword search list. This resulted in 2,009 unique articles.

This set of articles was again manually evaluated by the authors, similarly to the previous phase – first according to the metadata and then a subset of them considering the full texts. The result was a list of 41 more papers. The search was non-recursive (i.e., not snowballing) since we did not extract the reference lists of these new articles again.

3.3.3. Final list construction

By combining the results from keyword search and references extraction, we obtained 136 relevant articles. However, 31 of them were describing the same tools as the other ones, only in a different research stage (idea papers, evaluations, etc.). Two articles were classified as "not relevant" after a more careful inspection.

Therefore, the final set consists of 103 relevant articles.

[3]`http://ieeexplore.ieee.org`
[4]`http://dl.acm.org`
[5]`http://www.sciencedirect.com`
[6]`http://link.springer.com`
[7]`http://www.scopus.com`

3.4. Data extraction

For each article, we tried to find supplementary material on the web, such as manuals, screenshots, videos and executable tool downloads. We also extracted a tool name from every article; if the tool was unnamed, we devised a suitable pseudonym based on the keywords often used in the paper. In rare cases when one article presented more tools, we selected the most relevant one.

Three of the authors were each given a portion of the tools to classify. Using the full texts of the articles and available supplementary information, they tried to find similar and distinguishing characteristics of the tools. In this first round, each researcher gradually formed his/her own taxonomy (which could be freely inspired by others). A taxonomy consisted of a number of dimensions (e.g., "visualization type") with their attributes (e.g., "icon", "text"). One tool pertained to one or more attributes of every dimension.

Next, two of the classifying authors and the fourth author merged the taxonomies on a personal meeting. The three researchers then re-classified their tools accordingly. Finally, after a discussion, we renamed, split and merged some of the attributes.

The final result is: 1.) a taxonomy with 7 dimensions, each consisting of 2–6 attributes, and 2.) a categorization of all analyzed tools according to this taxonomy.

3.5. Threats to validity

Now we will discuss the validity threats of individual systematic mapping study parts.

Since the article selection process was performed by three authors, the inclusion and exclusion criteria could be comprehended differently by individual researchers. However, we intensively discussed the criteria in the team, along with the examples of individual tools matching and not matching the criteria to lower the variability to a minimum.

The terminology in the area (e.g., the expression "source code editor augmentation") is not yet standardized, so we could miss multiple relevant articles. An example of such an article is the vsInk tool paper [21], describing ink annotations for editable source code.

Although the keyword search results were not compared to any quasi-gold standard, we constructed it also by considering the results of our previous survey [11] that included multiple articles present in the current review.

During the backward references search in Scopus, we did not include so-called secondary documents, i.e. articles not indexed by Scopus but found in the reference lists. The reason is that this option resulted in many entries with incomplete bibliographical information, which would complicate the selection process.

Both the article selection and data extraction were performed by three researchers, each evaluating a non-overlapping portion of articles. Nevertheless, questionable items were discussed until a consensus was reached. Furthermore, after each part, one of the authors checked the list of all relevant articles to ensure the overall quality and to exclude inappropriate items.

Tools are often a neglected aspect of research, so some papers might not describe all available features of the given tool. Note that we do not draw any precise quantitative conclusions about the analyzed articles, such as exact percentages of tools in each category. The main purpose of this paper is to offer an overview of the field, where each of the analyzed tools acts as an example of particular augmentations and a reference for further information.

4. Taxonomy

To answer **RQ2**, we present the resulting taxonomy of source code editor augmentation in Table 1. In this section, we will describe the individual dimensions and their attributes, while mentioning representative examples of tools.

4.1. Source

The **source** dimension denotes where the data representing the augmentation were originally available before they were visually assigned to a part of source code. This dimension is similar to the one presented in our previous work [11].

Tools categorized as *code* analyze the source code of a system without executing it, i.e., using static analysis. This can range from simple markers about the presence of an MPI (Message Passing Interface) function call on the given line [22] to a sophisticated calculation of potential deployment costs using static analysis [23]. A very common topic is clone detection (e.g., CloneTracker [24], SimEclipse [25], SourcererCC-I [26]) and warnings about bad smells (JDeodorant [27], Stench Blossom [28]).

Data useful for augmentation can be collected by an execution of the program, using some form of dynamic analysis. These tools are marked as *runtime*.

The majority of *runtime*-sourced tools collect the data during an execution and display them afterward, when the program has stopped. For instance, IDE sparklines [29] are small graphics displaying the progress of numeric variable values over time. A common application is performance profiling (e.g., In situ profiler [30]). An important problem of this kind of tools is data invalidation: if the source code is modified after the data were collected, the runtime information may not be valid anymore and the program must be run again. This issue is rarely discussed in the reviewed articles. A notable exception is Senseo [31], where the authors explicitly state the data pertaining to the modified method and its dependencies are always invalidated.

Another type of tools utilizing the *runtime* source, live programming environments such as Impromptu HUD [15] and Gibber [32], display the augmentation in real time as the program is executing.

For tools utilizing the *human* source, a programmer must purposefully enter the data with a sole intention that they will be used by the augmentation. Typical examples are social bookmarks in Pollicino [33], tags in TagSEA [34] or manual concern-to-code mappings in Spotlight [35].

On the other hand, *interaction* data are collected automatically, possibly without the developer even knowing it (although that would be unethical). A typical example is local code change tracking for the purpose of clone detection (CnP [36], CSeR [37]). Some tools also collect data from external applications, for example, HyperSource [38] tracks the user's web browsing activity while editing a code part and then augments that part of the code with potentially relevant browsing history.

As software engineering is not an individual activity, collaboration data are formed naturally as the team communicates and cooperates. These data are utilized by *collaboration* tools. Some of these tools offer an analysis of existing artifacts. For instance, Rationalizer [39] displays last VCS (version control system) commit times, authors, commit messages and issues related to each source code. The second common category is represented by real-time collaborative source code editors, such as Saros [40], Collabode [41] and IDEOL [42].

Many tools use a combination of multiple methods. For example, Historef [43] automatically collects *interaction* data – local source code edits – in the background and then allows the developer to manually merge, split or reorder these edits (the *human* source) while highlighting the changes with different colors.

Table 1: The taxonomy of source code editor augmentation

Dimension *and its attributes*	**Description**
Source	Where does the data representing the augmentation come from?
code	Results of static source code analysis.
runtime	Results of the program execution; dynamic program analysis.
human	Manually entered information, previously present only in the human mind.
interaction	Interaction patterns of a single developer in the IDE or a similar tool.
collaboration	Behavior of multiple developers/users and their artifacts (like VCS commits).
Type	Data of what type does the augmentation directly represent?
boolean	The augmentation can be only present or non-present.
fixed enumeration	One of a set of possible categorical values, the categories are pre-defined by the tool itself.
variable enumeration	One of a set of possible categorical values, the number of categories is not fixed.
number	Numeric values (numbers, time, etc.).
string	A text string.
object	Another data type (an image, a complex structure, etc.).
Visualization	What does the augmentation look like?
color	A simple area filled with a background color, a color bar or text foreground color.
decoration	Decoration of the code, such as underline, overline or border.
icon	A small rectangular graphical object.
graphics	More complicated graphical representations – charts, arrows, diagrams, photos, etc.
text	An inserted text string (including a number written as a text).
Location	Where is the visual augmentation displayed?
left	To the left of the source code - usually in the left margin.
in code	Directly in the code.
right	It is right-aligned.
Target	To what is the augmentation visually assigned?
line	One line.
line range	Multiple lines, but without distinguishing characters in individual lines.
character range	A number of characters (on one line or multiple lines).
file	The whole file.
Interaction	How can we interact with the augmentation?
popover	A tooltip or a popup with additional information can appear at the place where the augmentation is displayed.
navigate	Performing an action directly on the augmentation navigates us to another code location, window, website, etc.
change	We can edit the displayed information directly by manipulating the augmentation (drag&drop, write, expand, collapse, highlight).
none/unknown	The augmentation does not offer interaction possibilities or they were not mentioned by the authors.
IDE	For what IDE/editor is the augmentation implemented?
existing	An existing IDE/editor is extended.
custom	A tool created by the authors specifically for the purposes described in the article (or prior works).

4.2. Type

The type of data represented directly by the visual augmentation is denoted by the **type** dimension. This can be regarded as an extension of the classical information visualization theory [44], which recognizes nominal, ordered and quantitative data.

The *boolean* type means that the augmentation is either present or not, without distinguishing any visual variations. Two most common boolean augmentations are one-type marker icons and one-color code highlights. For example, Remail [45] displays a marker icon in the left margin on each line which has a related e-mail available. The iXj plugin [46] highlights all code matching the given transformation pattern with a green background. Note that the boolean type is often insufficient to display all necessary information, so it is commonly combined with interaction possibilities (e.g., tooltips) or displayed only for a limited amount of time – after a specific action.

Some tools use a list of predefined values for displaying different augmentation notations. They can either be firmly stored as a finite list in the tool's data bank, i.e. it's a *fixed enumeration*; or the number of categories can be changed any time by the user or by the tool itself, i.e. *variable enumeration*.

An example of *fixed enumeration* augmentation can be seen in Syde [47] where a red left-margin highlight means a severe collaboration conflict and the yellow color represents moderate conflicts.

A very common application of *variable enumeration* augmentation is an assignment of a different color for every concern or feature in the source code. This technique is used with variations in at least seven tools: Spotlight [35], CIDE [48], ArchEvol [49], IVCon [50], FeatureIDE [51], FLOrIDA [52] and xLineMapper [53]. Another common approach is to assign a specific color for each developer working on a piece of code, used e.g., in ATCoPE [54]. However, there is a problem with such assignments: The color has only a limited number of easily distinguishable levels [55]. Without additional visualizations and interactions, the utility of such approaches would be questionable if the number of elements in the enumeration (features, people, etc.) raised beyond a reasonable limit.

If the augmentation is represented by a numeric value, e.g. a number of function calls or time, then we use the type *number*. For example, Clepsydra [56] calculates and displays worst-case execution times for individual code segments and Theseus [57] displays the number of function calls in a JavaScript file while the application is running in a browser.

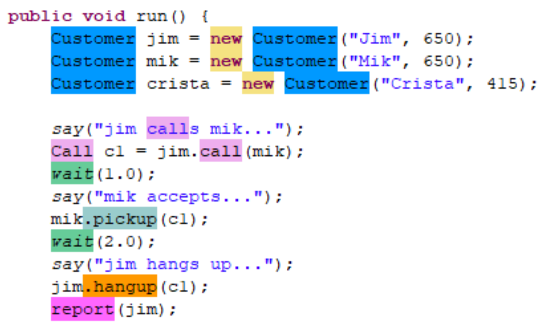
The *string* type represents a text string which is not enumerated, i.e., we cannot name all its possible values. For instance, the Live coding IDE [58] shows live values of variables in the source code editor.

Any other displayed information, such as an image or a complex data structure belongs to the *object* type. One such example is SE-Editor [59], which embeds web pages and images directly into the editor.

4.3. Visualization

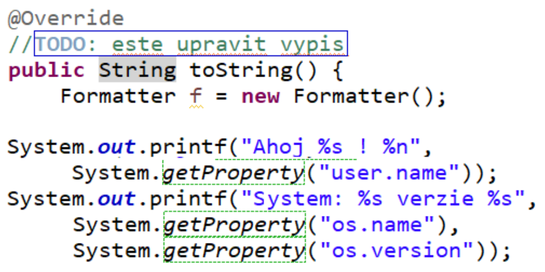
The **visualization** dimension relates to the visual appearance of the augmentation. For an illustration of various visualization kinds, see Figure 3.

Color is a very perceivable visual sign and at the same time the most simple one, applicable to both left or right editor bar and the code text itself. Color bars in the margins, text background and foreground color in the editor belong to the *color* category. A tool can utilize one color (Traces view [63]), multiple distinct colors (Jigsaw [64]) or a color spectrum. When utilizing the color spectrum, tools can change the hue (vLens [65]), saturation (CodeMend [66]) or brightness (CnP [36]) according to a numeric value. Note that the code foreground color is often reserved for syntax



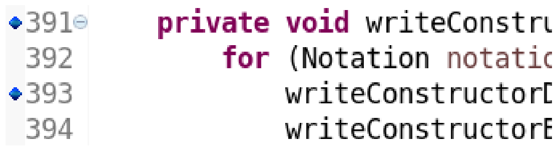
```
public void run() {
    Customer jim = new Customer("Jim", 650);
    Customer mik = new Customer("Mik", 650);
    Customer crista = new Customer("Crista", 415);

    say("jim calls mik...");
    Call c1 = jim.call(mik);
    wait(1.0);
    say("mik accepts...");
    mik.pickup(c1);
    wait(2.0);
    say("jim hangs up...");
    jim.hangup(c1);
    report(jim);
```

(a) *color*



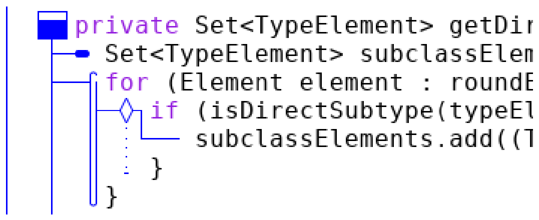
```
@Override
//TODO: este upravit vypis
public String toString() {
    Formatter f = new Formatter();

System.out.printf("Ahoj %s ! %n",
     System.getProperty("user.name"));
System.out.printf("System: %s verzie %s",
     System.getProperty("os.name"),
     System.getProperty("os.version"));
```

(b) *decoration*



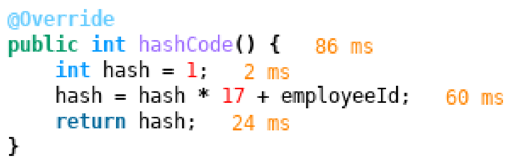
```
        private void writeConstru
            for (Notation notatic
                writeConstructorD
                writeConstructorE
```

(c) *icons*



```
private Set<TypeElement> getDi
   Set<TypeElement> subclassEle
   for (Element element : roundE
      if (isDirectSubtype(typeEl
         subclassElements.add((
      }
   }
```

(d) *graphics*



```
@Override
public int hashCode() {   86 ms
    int hash = 1;   2 ms
    hash = hash * 17 + employeeId;   60 ms
    return hash;   24 ms
}
```

(e) *text*

Figure 3: Tools with various visualization kinds: (a) Aspect Browser [60] uses text background *colors* to identify different aspects of code according to user-defined patterns; (b) eMoose [61] displays a solid border around TODO comments and adds dashed border *decorations* to code representing associated usage directives or reverse TODO references; (c) Pollicino [33] uses *icons* to represent bookmarks; (d) jGRASP [62] displays control structure diagram *graphics* directly in code; and (e) Clepsydra [56] calculates worst-case execution times and shows them as *textual labels* next to individual code lines.

highlighting. Therefore its use is very limited – we encountered only one such tool, IVCon [50], which does not use syntax highlighting at all.

A less notable highlight is usually represented by a text underline, overline, or border, belonging to the *decoration* category. A dashed border surrounding a code part is used in multiple tools, for example, in eMoose [61] it represents code with associated usage directives, such as threading limitations or reverse TODO references. The XSS marker plugin [67] utilizes a standard Eclipse visualization form, red squiggle underline, but with a different meaning – instead of denoting compilation errors, it warns about cross-site scripting vulnerabilities.

Icons are usually used in editor left bars as small, mostly rectangular and simple graphics. They can visually represent the metaphor of the given tool, e.g., in CodeBasket [68], an egg icon in the left margin represents a "code basket egg", i.e. a part of the programmers mental model. In other cases they are only attention-catching symbols, e.g., in DSketch [69], red square icons represent lines containing matches of dependency analysis. Occasionally, icons are located directly in the code editor (instead of the left margin) as in the ALVIS Live! tool [70].

More complicated graphical representations such as graphs, charts, arrows, diagrams, photos, etc. belong to the *graphics* group. For example, FluidEdt [71] displays heap graphs in the left margin. I3 [72] offers search similarity and change history views in form of small charts directly in the editor. The Fractional ownership tool [73] displays code authorship history as multi-colored stripes for each code line in the right editor part. A prototype "Error notifications" IDE [74] shows visual descriptions of compiler errors – e.g., in case of a name clash, two relevant code elements are visually connected with a line. DrScheme [75] connects individual uses of a selected language construct in the code by arrows. Finally, the Cares plugin [76] displays photos of code-related people in the editor.

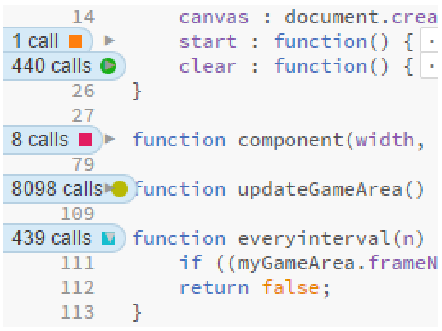
Any piece text added to the editor is considered to be a *text* augmentation. It can be subtle, as in CeDAR [77], where a clone region is labeled with a number in the corner: e.g., "Clone 1". Another example is the "ghost comments" technique of Moonstone [78] non-editable comment-like labels at the end of each line, or the Clepsydra tool's [56] line labeling with execution times. On the other hand, Code portals [79] embed significant portions of complementary texts into the code editor, such as relevant source code parts or visualizations of line differences.

4.4. Location

The **location** dimension denotes the position of the visual augmentation in the editor, which can either be to the *left* of the source code, displayed directly *in code*, or placed in the *right* editor part, aligned to the right editor margin. See Figure 4 for an illustration.

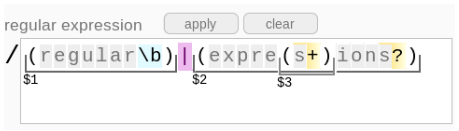
The *left* margin, alternatively called gutter or ruler, traditionally displays line numbers and simple icons (e.g., iMaus [81]). However, it can show also thin color bars (Diver [82]), thicker color fills (HyperSource [38]), "pills" containing text (Theseus [57]) or even a list of callers (Stacksplorer [83]).

Among tools displaying augmentation *in code*, we can distinguish multiple cases. The jGRASP tool [62] displays a control structure diagram in the indentation area of source code containing only tabs or spaces so it does not visually overlap with the text. DrScheme [75] displays arrows directly above code, covering small parts of it; however, this augmentation is only temporary displayed on mouse hover, which makes it acceptable. ARCC [84] draws augmentation behind the code in the form of a background color. SketchLink [85] displays icons directly in the code, but only at the end of lines. Finally, RegViz [80] slightly reflows the text so that the visual annotations do not overlap it.



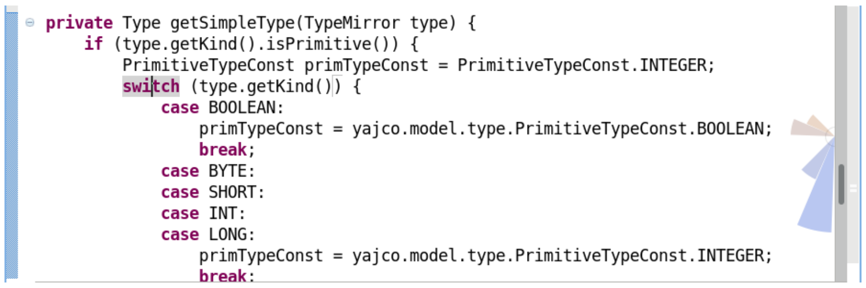
```
            canvas : document.crea
1 call        start : function() {
440 calls     clear : function() {
        }

8 calls     function component(width,
8098 calls  function updateGameArea()
439 calls   function everyinterval(n)
                if ((myGameArea.frameN
                return false;
            }
```

(a) *left*



```
/(regular\b)|(expre(s+)ions?)
```

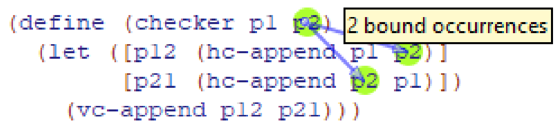
(b) *in code*



```
private Type getSimpleType(TypeMirror type) {
    if (type.getKind().isPrimitive()) {
        PrimitiveTypeConst primTypeConst = PrimitiveTypeConst.INTEGER;
        switch (type.getKind()) {
            case BOOLEAN:
                primTypeConst = yajco.model.type.PrimitiveTypeConst.BOOLEAN;
                break;
            case BYTE:
            case SHORT:
            case INT:
            case LONG:
                primTypeConst = yajco.model.type.PrimitiveTypeConst.INTEGER;
                break;
```

(c) *right*

Figure 4: Tools with various location kinds: (a) Theseus [57] displays the number of function calls in a JavaScript code in the *left* editor part; (b) RegViz [80] highlights regular expressions structure directly *in code*; and (c) Stench Blossom's [28] ambient smell detector that lives on the *right* edge of the program editor.



```
(define (checker p1 p2)
  (let ([p12 (hc-append p1 p2)]
        [p21 (hc-append p2 p1)])
    (vc-append p12 p21)))
```

Figure 5: DrScheme [75], currently named DrRacket – a tool utilizing the *popover* interaction in form of a tooltip denoting the number of bound occurrences of particular constructs in code.

Right augmentation components are pinned to the editor edges, which means that if the IDE window resizes (changes its width), the components move along with its edges. "Color chips" in CoderChrome [14], displayed in the right part of the editor, are an example of such a visualization. Their meaning can be configured, e.g., to mark starts and ends of code blocks. Because the right margin is usually not the central programmer's focus point, it is an ideal place to implement ambient, non-disturbing augmentations – such as the Stench Blossom [28] code smell visualization. Since this location is near the scrollbar, it is also useful for general overviews: heatmap-colored icons in Senseo [31] referring to available runtime information in the whole file.

4.5. Target

Target denotes the code construct to which the augmentation is visually assigned.

Line augmentations are bound to a single line of code, *line range* to multiple consecutive lines. Augmentations displayed in the left margin are practically always assigned to a *line* or a *line range*. For instance, each left-margin crash analysis icon in CSIclipse [86] is related to a particular *line* (e.g., "line may have been executed"). *Line range* augmentations are typically denoted as color bars in the left margin (Featureous [87]) or code background color spanning whole lines, up to the right margin (EnergyDebugger [88]).

On the other hand, a *character range* augmentation can mark an arbitrary selection of individual characters. The augmentation can either stay on one line (e.g., a border decoration in ChangeCommander [89]) or potentially span multiple lines (background color highlighting in CodeGraffiti [90]).

Line and *character range* augmentations are often used in tandem: A left margin icon marks the line of interest while the decoration (TexMo [91]) or background color (Gilligan [92]) highlights a more specific source code range in the given line.

If the augmentation relates to the whole code in the currently opened file, then it belongs to the *file* category. Such a visualization can spatially correspond to source code parts (jGRASP [62]), or its placement can be solely a matter of style (Gibber [32]).

4.6. Interaction

An icon, text or any other graphics or representation is usually not sufficient to give the user enough information about the augmentation. Tools usually provide some way of displaying more data about the particular case, which is usually initiated by the user interacting with a graphical component representing the augmentation. The **interaction** dimension describes the ways of how it is possible to interact with the displayed augmentation.

The most common interaction (if there is any) is a case when a *popover* (a tooltip or a popup) with additional information displays after a mouse click or a mouse cursor hovering over the augmentation. This can be a simple textual tooltip, as depicted in Figure 5 (DrScheme [75]), or a multi-line formatted text with a picture and multiple clickable actions (Cares [76]).

If there is a lot of additional data to display for the augmentation and they would not fit into a popup, clicking or double-clicking on the augmentation component usually fires a *navigation* event, which selects an item in another IDE view/window (traceability links in Morpheus [93]) or displays the information in some external program, e.g. a web browser (a bug report page in the case of Rationalizer [39]). If the augmentation is somehow related to other parts of the code, the user is navigated directly to those code parts in the same or another file – e.g., control-flow and data-flow hyperlinks in Flower [94].

Some tools enable direct manipulation of the augmentation itself using mouse or keyboard interaction, which *changes* the displayed information or its appearance. Code portals [79] allow the developer to edit the displayed inline texts. The FixBugs tool [95] offers drag&drop refactoring capabilities directly in the source code editor. In Fluid views [96], the interaction happens in two steps: First, an unobtrusive visual cue changes to an underline decoration on a mouse hover. After clicking it, the augmentation fully expands and the related code is displayed inside the editor.

If the augmentation component is not interactive in any way or the authors of the tool did not mention any possibilities of interaction in their paper or any supplementary materials, it is classified as *none/unknown*.

4.7. IDE

The last dimension denotes the **IDE** or editor, for which the augmentation is implemented. The solutions are either implemented as plugins into an *existing* IDE or code editor, or they are completely new *custom* environments created by the authors specifically for the purposes described in their article.

Most of the tools we identified were implemented as Eclipse plugins, which is obvious given the popularity and open architecture of this IDE. Some of them were created for other more or less popular IDEs such as Visual Studio (GhostFactor [97]), IntelliJ IDEA (wIDE [98]) or Brackets IDE (Theseus [57]). Examples of custom tools created from scratch are Omnicode [99], Shared-code editor [100], Code portals [79] and Code Bubbles[101].

5. Tools

As an answer to **RQ1**, we provide a complete list of the surveyed tools in Tables 2 and 3. The rows are individual tools, sorted alphabetically by the tool name. Each column represents an attribute of the corresponding dimension.

If the tool contains an augmentation fulfilling the given attribute, this is denoted by a filled square (■). Otherwise, an empty square (□) is displayed.

Note that the "IDE" dimension has only one attribute displayed since it has two mutually exclusive attributes. Also note that one tool can contain multiple related or unrelated augmentations (e.g. an icon and color highlighting) or one augmentation fulfilling multiple attributes (e.g., an icon offering both popover and navigation possibilities).

In Figure 6, there is a chart displaying the numbers of reviewed tools fulfilling each attribute of a particular dimension. In each dimension, the attributes are ordered by frequency.

It can be seen from the chart that the most popular source of data for augmentation is the code itself. This is understandable because source code augmentation was traditionally used by static analysis tools. As can be expected, simple visualization techniques like color or icon and simple data (boolean, fixed enumeration) were used more often than complex visualizations of complex data that are much harder to design and implement. Graphical visualizations are also rarely used

Table 2: Augmentation tools

Tool	Year	Source					Type						Visualization					Location			Target				Interaction				IDE
		code	runtime	human	interaction	collaboration	boolean	fixed enum.	variable enum.	number	string	object	color	decoration	icon	graphics	text	left	in code	right	line	line range	character range	file	popover	navigate	change	none/unknown	existing
ALVIS Live! [70]	2007	■	□	□	□	□	■	□	□	□	□	□	□	□	■	□	□	□	■	□	■	□	□	□	□	■	□	□	□
ARCC [84]	2017	■	□	□	□	□	■	□	■	□	□	□	■	■	□	□	□	■	■	□	■	□	■	□	■	□	□	□	■
ArchEvol [49]	2009	□	□	■	□	□	□	□	■	□	□	□	■	■	□	□	□	■	■	□	□	■	□	□	■	□	□	□	■
ASIDE [102]	2011	■	□	■	□	□	■	■	□	□	□	□	■	■	■	□	□	■	■	□	■	□	■	□	■	□	■	□	■
Aspect Browser [60]	2001	■	□	■	□	□	□	□	■	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	□	■	□	□	■
ATCoPE [54]	2012	□	□	□	□	■	□	□	■	□	□	□	■	□	□	□	□	■	■	□	□	■	□	□	□	□	□	■	■
BeneFactor [103]	2012	■	□	□	■	□	■	□	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	■	□	■	□	■
Cares [76]	2012	□	□	□	□	■	□	■	□	□	□	■	□	□	□	■	□	□	□	■	□	□	□	■	■	■	□	□	■
CeDAR [77]	2012	■	□	□	■	□	■	□	□	□	■	□	■	□	□	□	■	□	■	□	□	■	□	□	□	■	□	□	■
ChangeCommander [89]	2008	■	□	□	□	■	■	□	□	□	□	□	□	■	■	□	□	■	■	□	□	□	■	□	■	□	□	□	■
Cheetah [104]	2017	■	□	□	□	□	□	■	□	□	□	□	■	□	■	□	□	■	■	□	■	□	□	□	■	□	□	□	■
CIDE [48]	2008	□	□	■	□	□	□	□	■	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	■	□	□	□	■
Clepsydra [56]	2007	■	□	□	□	□	□	□	□	■	□	□	□	□	□	□	■	□	■	□	■	□	□	□	□	□	□	■	■
CloneTracker [24]	2007	■	□	□	□	□	■	□	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	■	□	□	□	■
CnP [36]	2010	□	□	□	■	□	■	□	■	■	□	□	■	□	■	□	□	■	□	□	■	■	□	□	■	□	□	□	■
Code Bubbles [101]	2010	□	□	□	■	□	■	□	□	□	□	□	■	□	□	■	□	■	■	■	■	□	□	□	□	□	■	□	□
Code Orb [105]	2011	□	□	□	□	■	□	□	□	■	□	□	■	□	□	□	□	■	□	□	■	□	□	□	□	■	□	□	■
Code portals [79]	2016	■	■	□	□	■	□	□	■	□	■	□	■	■	□	□	■	□	■	□	■	□	■	□	□	□	■	□	□
CodeBasket [68]	2015	□	□	■	□	□	■	□	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	□	■	□	□	■
CodeGraffiti [90]	2014	□	□	■	□	□	□	□	■	□	□	□	■	□	□	□	□	■	■	□	■	□	■	□	□	■	□	□	■
Codelink [106]	2004	■	□	■	■	□	■	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	□	□	■	□	■
CodeMend [66]	2016	■	□	□	□	■	□	■	□	□	□	□	■	□	□	□	□	□	■	□	■	□	■	□	□	■	□	□	□
CoderChrome [14]	2010	■	□	□	□	■	■	■	■	■	□	□	■	■	■	□	□	■	■	■	■	□	■	□	■	■	□	□	■
CodeTalk [107]	2010	□	□	■	□	□	□	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	□	□	□	□	□	□	■
Collabode [41]	2011	□	□	□	□	■	□	■	□	□	□	□	■	□	■	□	□	■	■	□	□	□	■	□	□	□	□	■	□
CoRED [108]	2012	□	□	□	■	■	■	□	■	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	■	□	■	□	■
CostHat [23]	2016	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	■	□	□	□	■
CSeR [37]	2010	□	□	□	■	□	□	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	■	□	□	□	■
CSIclipse [86]	2015	■	■	□	□	□	□	■	□	□	□	□	■	□	■	□	□	■	■	□	■	■	□	□	■	■	□	□	■
CssCoco [109]	2016	■	□	□	□	□	■	□	□	□	□	□	□	■	■	□	□	■	■	□	■	□	□	□	□	□	□	■	■
Debugger Canvas [110]	2012	□	■	□	□	□	□	□	■	□	□	■	■	■	■	■	□	■	□	■	■	□	□	□	■	□	□	□	■
Diver [82]	2010	□	■	□	□	□	■	□	□	□	□	□	■	□	□	□	□	■	□	□	□	■	□	□	□	□	□	■	■
DrScheme [75]	2002	■	□	□	□	□	■	□	□	□	□	■	□	□	□	■	□	□	■	□	□	□	■	□	■	□	■	□	□
DSketch [69]	2010	■	□	□	□	□	□	□	■	□	□	□	■	□	■	□	□	■	■	□	■	□	■	□	□	■	□	□	■
Dynamic text [111]	2012	□	□	■	■	□	□	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	□	□	□	■	■
Eclipse PTP [22]	2006	■	□	□	□	□	■	□	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	□	□	□	■	■
EG [112]	2004	□	■	□	□	□	□	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	■	□	□	□	■
eMoose [61]	2009	□	□	■	□	□	■	■	□	□	□	□	□	■	■	□	□	■	■	□	■	□	■	□	■	□	□	□	■
EnergyDebugger [88]	2016	□	■	□	□	■	■	□	□	□	□	□	■	□	□	□	□	□	■	□	□	■	□	□	□	□	□	■	■
Error notifications [74]	2014	■	□	□	□	□	□	□	□	□	□	■	□	□	□	■	□	□	■	□	□	□	■	□	□	□	□	■	□
FeatureIDE [51]	2016	■	□	□	□	□	□	□	■	□	□	□	■	□	□	□	□	■	■	■	□	■	□	□	□	□	□	■	■
Featureous [87]	2012	□	■	■	□	□	□	□	□	■	□	□	■	□	□	□	□	■	□	□	□	■	□	□	■	□	□	□	■
FixBugs [95]	2016	■	□	□	□	□	□	■	■	□	□	□	■	■	■	□	□	■	■	□	□	□	■	□	□	□	■	□	■
FLOrIDA [52]	2017	■	□	□	□	□	□	□	■	□	□	□	■	□	□	□	□	□	■	□	■	□	□	□	□	□	□	■	□
Flower [94]	2017	■	□	□	□	□	■	□	□	□	□	□	□	■	□	□	□	□	■	□	□	□	■	□	□	■	□	□	■
Fluid views [96]	2006	■	□	□	□	□	■	□	□	□	■	□	■	■	□	□	■	□	■	□	□	□	■	□	□	□	■	□	■
FluidEdt [71]	2015	□	■	□	□	□	□	□	□	□	□	■	□	□	□	■	□	■	□	□	■	□	□	□	□	□	■	□	□
Fractional owner. [73]	2015	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	□	□	■	■	□	□	□	■	□	□	□	■
GhostFactor [97]	2014	■	□	□	■	□	■	□	□	□	□	□	□	■	□	□	□	□	■	□	□	□	■	□	■	□	□	□	■
Gibber [32]	2014	□	■	□	□	□	□	□	□	□	□	■	□	□	□	■	□	□	■	□	□	□	□	■	□	□	□	■	□
Gilligan [92]	2007	■	□	■	□	□	□	■	□	□	□	□	■	□	■	□	□	■	■	□	■	□	■	□	□	□	□	■	■
Hermion [113]	2008	□	■	□	□	□	■	■	□	□	□	□	□	□	■	□	□	□	■	□	■	□	■	□	■	■	□	□	■

Table 3: Augmentation tools (continued)

Tool	Year	Source					Type						Visualization					Location			Target				Interaction				IDE
		code	runtime	human	interaction	collaboration	boolean	fixed enum.	variable enum.	number	string	object	color	decoration	icon	graphics	text	left	in code	right	line	line range	character range	file	popover	navigate	change	none/unknown	existing
Historef [43]	2015	□	□	■	■	□	□	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	□	□	□	■	■
HyperSource [38]	2011	■	□	□	■	□	□	■	□	□	□	□	□	■	■	□	□	■	□	□	■	□	□	□	■	□	□	□	■
I3 [72]	2015	■	□	□	□	■	□	■	□	□	□	■	■	□	□	■	□	□	■	□	□	□	■	□	■	□	□	□	■
IDE sparklines [29]	2013	□	■	□	□	□	□	□	□	■	□	□	□	□	□	■	□	□	■	□	■	□	□	□	■	□	□	□	■
IDEOL [42]	2014	□	□	□	□	■	□	□	■	□	□	□	■	□	□	□	■	□	■	■	□	□	■	□	□	□	□	■	□
iMaus [81]	2009	■	□	□	□	□	■	□	□	□	□	□	□	■	■	□	□	■	■	□	■	□	■	□	□	■	□	□	■
Impromptu HUD [15]	2013	□	■	□	□	□	□	□	□	■	□	□	□	□	□	■	■	□	■	□	■	□	□	■	□	□	□	■	■
In situ profiler [30]	2013	□	■	□	□	□	□	■	■	■	□	□	□	□	□	■	□	□	■	□	■	□	□	□	■	□	□	□	■
inCode [114]	2017	■	□	□	□	□	■	□	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	■	□	□	□	■
IVCon [50]	2009	□	□	■	□	□	□	□	■	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	■	□	□	□	□
iXj [46]	2007	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	□	■	■	□	□	□	■	□	□	■	□	□	■
Jazz Band [115]	2004	□	□	□	□	■	□	■	□	□	□	□	■	□	■	□	□	■	□	□	□	■	□	□	■	■	□	□	■
JDeodorant [27]	2008	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	□	□	□	■	■
jGRASP [62]	2011	■	□	□	□	□	□	□	□	□	□	■	□	□	□	■	□	□	■	□	□	□	□	■	□	□	■	□	□
Jigsaw [64]	2008	■	□	□	■	□	□	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	□	□	□	■	■
JScoper [116]	2005	■	□	□	□	□	■	□	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	□	■	□	□	■
Live coding IDE [58]	2014	□	■	□	□	□	□	□	□	■	■	□	□	□	□	□	■	□	□	■	■	□	□	□	□	□	■	□	■
Moonstone [78]	2017	■	□	□	□	□	■	■	□	□	■	□	■	■	■	□	■	■	■	□	■	■	■	□	■	□	□	□	■
Morpheus [93]	2017	■	□	□	■	□	□	■	□	□	■	□	■	■	■	□	■	■	■	□	■	■	■	□	■	■	■	□	■
NLP Eclipse [117]	2011	■	□	□	□	□	■	□	□	□	□	□	□	□	■	□	□	■	□	□	□	□	■	□	■	□	□	□	■
Omnicode [99]	2017	■	□	■	■	□	■	□	□	□	□	□	■	□	□	□	□	■	■	□	■	□	■	□	■	■	□	□	□
PerformanceHat [118]	2015	□	■	□	□	□	■	□	□	□	□	□	■	□	■	□	□	■	■	□	□	□	■	□	■	□	□	□	■
Pollicino [33]	2011	□	□	■	□	□	■	□	□	□	□	□	■	□	■	□	□	■	■	□	■	□	□	□	□	■	□	□	■
PyLighter [119]	2009	□	■	□	□	□	□	□	□	■	□	□	■	□	□	□	□	□	■	□	■	□	□	□	□	□	□	■	□
Rationalizer [39]	2011	□	□	□	□	■	□	■	■	■	■	□	■	□	■	□	■	□	□	■	■	□	□	□	■	■	□	□	■
Refactoring Ann. [120]	2008	■	□	□	□	□	□	□	□	□	□	■	□	□	□	■	□	□	■	□	□	□	■	□	□	□	□	■	■
RegViz [80]	2014	■	□	□	□	□	□	■	□	■	□	□	■	■	□	□	■	□	■	□	□	□	■	□	□	□	■	□	□
Remail [45]	2011	□	□	□	□	■	■	□	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	■	□	□	□	■
Saros [40]	2010	□	□	□	□	■	□	□	■	□	□	□	■	□	□	□	□	■	■	□	□	■	■	□	□	■	□	□	■
SE-Editor [59]	2009	□	□	■	□	□	□	□	□	□	□	■	□	□	□	■	□	□	■	□	□	■	□	□	□	□	□	■	■
Senseo [31]	2012	□	■	□	□	□	□	■	□	□	□	□	■	□	□	■	□	■	□	■	■	□	□	□	□	□	□	■	■
Shared-code editor [100]	2017	■	□	■	■	■	■	□	□	□	■	□	□	■	□	□	■	□	□	□	□	■	□	□	□	□	□	■	□
SimEclipse [25]	2015	■	□	□	□	□	■	□	□	□	□	□	■	□	■	□	□	■	■	□	■	■	□	□	■	■	■	□	■
SketchLink [85]	2014	■	□	□	□	□	■	□	□	□	□	□	□	□	■	□	□	□	■	□	■	□	□	□	■	■	■	□	■
Soot-Eclipse [121]	2004	■	□	□	□	□	■	□	□	□	□	□	■	□	■	□	□	■	■	□	■	□	■	□	■	■	□	□	■
Soul [122]	2011	■	□	□	□	□	■	□	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	■	□	□	□	■
SourcererCC-I [26]	2016	■	□	□	□	□	□	■	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	□	□	□	■	■
Spotlight [35]	2005	□	□	■	□	□	□	□	■	□	□	□	■	■	□	□	□	■	■	□	□	■	□	□	□	□	□	■	■
Stacksplorer [83]	2011	■	□	□	□	□	■	■	■	□	■	□	■	■	■	□	■	■	■	■	□	■	■	□	□	■	■	□	■
Stench Blossom [28]	2010	■	□	□	□	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	□	□	■	■	□	□	□	■
Syde [47]	2010	□	□	□	□	■	□	■	□	□	□	□	■	□	□	□	□	■	□	□	□	■	□	□	■	□	□	□	■
TagSEA [34]	2009	□	□	■	□	□	■	□	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	□	■	□	□	■
TexMo [91]	2012	■	□	□	□	□	□	■	□	□	□	□	■	□	■	□	□	■	■	□	■	□	■	□	■	□	□	□	■
Theseus [57]	2014	□	■	□	□	□	□	■	□	■	■	□	□	□	■	□	■	■	□	□	■	□	■	□	□	■	■	□	■
Traces view [63]	2012	□	■	□	□	□	■	□	□	□	□	□	■	□	□	□	□	□	■	□	■	□	□	□	□	□	□	■	■
vLens [65]	2013	■	■	□	□	□	□	□	□	■	□	□	■	□	□	□	□	□	□	□	□	■	□	□	□	□	□	■	■
wIDE [98]	2017	■	□	□	□	□	□	■	□	□	□	□	■	□	□	□	□	□	■	□	□	□	■	□	□	□	□	■	■
xLineMapper [53]	2017	□	□	■	□	□	■	□	■	□	□	□	■	□	■	□	□	■	■	□	■	□	■	□	□	□	□	■	■
XSS marker [67]	2011	■	□	□	□	□	■	□	□	□	□	□	□	■	■	□	□	■	■	□	■	□	■	□	■	□	□	□	■
YinYang [123]	2013	□	■	□	□	□	□	□	□	□	■	□	□	□	□	□	■	□	■	□	■	□	□	□	□	■	□	□	□
Zelda [124]	2009	■	□	■	□	□	■	□	□	□	□	□	□	□	■	□	□	■	□	□	■	□	□	□	□	□	□	■	■

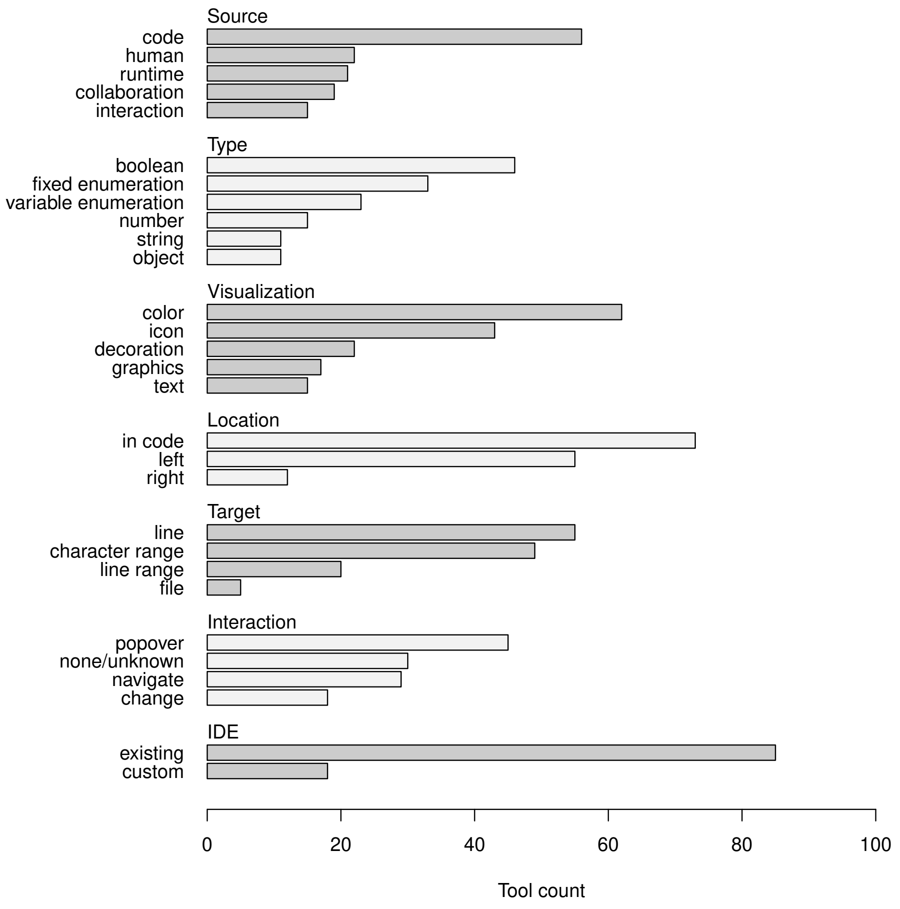


Figure 6: The numbers of tools fulfilling the given attributes of the dimensions

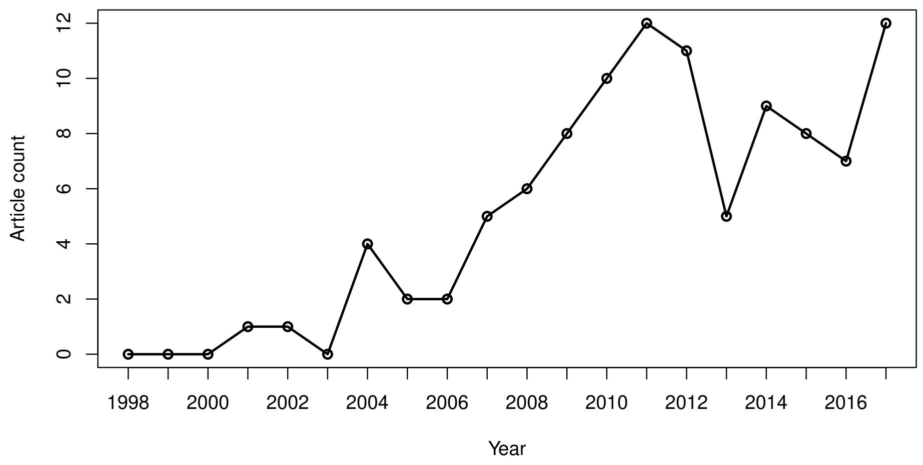


Figure 7: A distribution of the publications over years

by the industrial IDEs discussed in Section 2.2, although they actually implement augmentations from all attributes of our taxonomy.

Most of the visualizations were displayed directly in code or in the left margin. The right margin was used by only a few tools. This may be related to the fact that majority of the tools were based on existing industrial IDEs and these environments provide direct support for highlighting parts of code and displaying markers in the left margin. Some of these IDEs can also automatically display an overview of all left-margin icons in the right margin, but since this is their built-in feature, it is not shown in the tables.

The least represented attribute in our review is the *file* target. Integration of some information directly in the code editor is usually most useful if it is related to some specific parts of the code.

In Figure 7, we can see a distribution of the 103 reviewed publications over years 1998–2017. After a slow start, the number of articles per year tends to rise from 2005 to 2011 and then begins to oscillate. The sudden growth starting in the mid-2000s can be possibly attributed to the ascending popularity of the IDEs, particularly Eclipse. Note, however, that the numbers are only indicative since some articles might have been missed during the systematic review process.

We have also analyzed venues where the selected papers were published. Most of the papers (94) were published at conferences, while only 9 were published in journals. List of the top venues is provided in Table 4. It contains all venues with more than one publication in our study. Clearly, the most popular venue is the International Conference on Software Engineering (ICSE) with 18 publications.

6. Conclusion

We presented a systematic mapping study about visual augmentation of source code editors. Using keyword and references search combined with manual filtering, we constructed a list containing 103 relevant tools described in research articles. A taxonomy representing distinct and similar characteristics of these tools was constructed. It contains seven dimensions: source, type, visualization, location, target, interaction and IDE. Each tool was characterized according to this taxonomy,

Table 4: Top venues with count of publications used in the study

Venue	**Count**
International Conference on Software Engineering (ICSE)	18
International Conference on Program Comprehension (ICPC)	7
Conference on Human Factors in Computing Systems (CHI)	6
International symposium on Foundations of software engineering (FSE)	5
Symposium on Visual Languages and Human-Centric Computing (VL/HCC)	5
Symposium on User Interface Software and Technology (UIST)	4
Conference on Computer Supported Cooperative Work (CSCW)	2
Eclipse Technology eXchange (ETX)	2
European Conference on Software Maintenance and Reengineering (CSMR)	2
International Conference on Automated Software Engineering (ASE)	2
International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)	2
IEEE Transactions on Software Engineering	2
Science of Computer Programming	2

producing a table with article references. The assignment of the features (dimensions' attributes) to the tools provides a brief overview of their properties and can be used for comparison.

Our main contributions are:

- the definition of the term "source code editor augmentation",
- a taxonomy of source code editor augmentation features and
- a categorized list of augmentation tools with references.

This article can be a useful resource for researchers aiming to gain an overview of this area or finding a particular example of given augmentations. Furthermore, we provide multiple directions for future work, which we will now describe.

Since we found more than 100 tools augmenting the source code editor area, there is a need to think about filtering possibilities. Even though some augmentations have only a limited lifetime, the IDE may easily become visually cluttered. An interesting question is also how to resolve conflicts when displaying visually overlapping augmentations.

The underlying data or source code often change after the augmentation is initially displayed. Invalidation and possible recalculation of augmentations is another important issue, seldom discussed in the reviewed articles.

There are many papers mentioning source code editor augmentation; nevertheless, it is often only a marginal topic. Although in some articles, augmentation is a central topic (such as in our recent paper about variable values in the code [125]), they often miss empirical evaluation. On the other hand, while many articles include empirical evaluation, the visual code augmentation itself is rarely the main object of the studies. One of the notable exceptions is a recent runtime state visualization approach by Hoffswell et al. [126], accompanied by a thorough empirical study.

Further comparisons of the usability of separate views, in-code augmentations and their variations are definitely welcome.

Acknowledgment

This work was supported by project KEGA 047TUKE-4/2016 Integrating software processes into the teaching of programming.

References

References

[1] D. Asenov, P. Muller, Envision: A fast and flexible visual code editor with fluid interactions (overview), in: 2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2014, pp. 9–12. `doi:10.1109/VLHCC.2014.6883014`.

[2] D. Ungar, H. Lieberman, C. Fry, Debugging and the experience of immediacy, Communications of the ACM 40 (4) (1997) 38–43. `doi:10.1145/248448.248457`.

[3] D. Gračanin, K. Matković, M. Eltoweissy, Software visualization, Innovations in Systems and Software Engineering 1 (2) (2005) 221–230. `doi:10.1007/s11334-005-0019-8`.

[4] M. Shahin, P. Liang, M. A. Babar, A systematic review of software architecture visualization techniques, Journal of Systems and Software 94 (Supplement C) (2014) 161–185. `doi:10.1016/j.jss.2014.03.071`.

[5] S. Šimoňák, Algorithm visualization using the VizAlgo platform, Acta Electrotechnica et Informatica 13 (2) (2013) 54–64.

[6] M.-A. D. Storey, D. Čubranić, D. M. German, On the use of visualization to support awareness of human activities in software development: A survey and a framework, in: Proceedings of the 2005 ACM Symposium on Software Visualization, SoftVis '05, ACM, New York, NY, USA, 2005, pp. 193–202. `doi:10.1145/1056018.1056045`.

[7] J. I. Maletic, A. Marcus, M. L. Collard, A task oriented view of software visualization, in: Proceedings First International Workshop on Visualizing Software for Understanding and Analysis, 2002, pp. 32–40. `doi:10.1109/VISSOF.2002.1019792`.

[8] C. J. Sutherland, A. Luxton-Reilly, B. Plimmer, Freeform digital ink annotations in electronic documents: A systematic mapping study, Computers & Graphics 55 (2016) 1–20. `doi:10.1016/j.cag.2015.10.014`.

[9] T. Kosar, S. Bohra, M. Mernik, Domain-specific languages: A systematic mapping study, Information and Software Technology 71 (2016) 77–91. `doi:10.1016/J.INFSOF.2015.11.001`.

[10] E. Syriani, L. Luhunu, H. Sahraoui, Systematic mapping study of template-based code generation, Computer Languages, Systems & Structures 52 (2018) 43–62. `doi:10.1016/J.CL.2017.11.003`.

[11] M. Sulír, J. Porubän, Labeling source code with metadata: A survey and taxonomy, in: 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), 2017, pp. 721–729. doi:10.15439/2017F229.

[12] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, B. MacIntyre, Recent advances in augmented reality, IEEE Comput. Graph. Appl. 21 (6) (2001) 34–47. doi:10.1109/38.963459.

[13] M. Voelter, J. Siegmund, T. Berger, B. Kolb, Towards user-friendly projectional editors, in: Software Language Engineering, Springer International Publishing, 2014, pp. 41–61. doi:10.1007/978-3-319-11245-9_3.

[14] M. Harward, W. Irwin, N. Churcher, In situ software visualisation, in: Proceedings of the Australian Software Engineering Conference, ASWEC, 2010, pp. 171–180. doi:10.1109/ASWEC.2010.18.

[15] B. Swift, A. Sorensen, H. Gardner, J. Hosking, Visual code annotations for cyberphysical programming, in: 2013 1st International Workshop on Live Programming, LIVE 2013 - Proceedings, IEEE Computer Society, 2013, pp. 27–30. doi:10.1109/LIVE.2013.6617345.

[16] M. Sulír, J. Porubän, Exposing runtime information through source code annotations, Acta Electrotechnica et Informatica 17 (1) (2017) 3–9.

[17] M. F. Cowlishaw, LEXX—a programmable structured editor, IBM J. Res. Dev. 31 (1) (1987) 73–80. doi:10.1147/rd.311.0073.

[18] M. D. Schwartz, N. M. Delisle, V. S. Begwani, Incremental compilation in magpie, SIGPLAN Not. 19 (6) (1984) 122–131. doi:10.1145/502949.502887.

[19] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, Journal of Systems and Software 80 (4) (2007) 571–583. doi:10.1016/j.jss.2006.07.009.

[20] H. Zhang, M. A. Babar, P. Tell, Identifying relevant studies in software engineering, Information and Software Technology 53 (6) (2011) 625–637. doi:10.1016/j.infsof.2010.12.010.

[21] C. J. Sutherland, B. Plimmer, vsInk: Integrating digital ink with program code in Visual Studio, in: Proceedings of the Fourteenth Australasian User Interface Conference - Volume 139, AUIC '13, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 2013, pp. 13–22.

[22] G. Watson, N. DeBardeleben, Developing scientific applications using eclipse, Computing in Science and Engineering 8 (4) (2006) 50–61. doi:10.1109/MCSE.2006.64.

[23] P. Leitner, J. Cito, E. Stöckli, Modelling and managing deployment costs of microservice-based cloud applications, in: Proceedings - 9th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2016, ACM, 2016, pp. 165–174. doi:10.1145/2996890.2996901.

[24] E. Duala-Ekoko, M. Robillard, Tracking code clones in evolving software, in: Proceedings - International Conference on Software Engineering, 2007, pp. 158–167. doi:10.1109/ICSE.2007.90.

[25] M. S. Uddin, C. K. Roy, K. A. Schneider, Towards convenient management of software clone codes in practice: An integrated approach, in: Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering, IBM Corp., 2015, pp. 211–220.

[26] V. Saini, H. Sajnani, J. Kim, C. Lopes, SourcererCC and SourcererCC-I: Tools to detect clones in batch mode and during software development, in: Proceedings - International Conference on Software Engineering, IEEE Computer Society, 2016, pp. 597–600. `doi:10.1145/2889160.2889165`.

[27] N. Tsantalis, T. Chaikalis, A. Chatzigeorgiou, JDeodorant: Identification and removal of type-checking bad smells, in: Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2008, pp. 329–331. `doi:10.1109/CSMR.2008.4493342`.

[28] E. Murphy-Hill, A. Black, An interactive ambient visualization for code smells, in: Proceedings of the ACM Conference on Computer and Communications Security, 2010, pp. 5–14. `doi:10.1145/1879211.1879216`.

[29] F. Beck, F. Hollerich, S. Diehl, D. Weiskopf, Visual monitoring of numeric variables embedded in source code, in: 2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013, 2013. `doi:10.1109/VISSOFT.2013.6650545`.

[30] F. Beck, O. Moseler, S. Diehl, G. Rey, In situ understanding of performance bottlenecks through visually augmented code, in: IEEE International Conference on Program Comprehension, 2013, pp. 63–72. `doi:10.1109/ICPC.2013.6613834`.

[31] D. Röthlisberger, M. Härry, W. Binder, P. Moret, D. Ansaloni, A. Villazón, O. Nierstrasz, Exploiting dynamic information in IDEs improves speed and correctness of software maintenance tasks, IEEE Transactions on Software Engineering 38 (3) (2012) 579–591. `doi:10.1109/TSE.2011.42`.

[32] C. Roberts, M. Wright, J. Kuchera-Morin, T. Höllerer, Gibber: Abstractions for creative multimedia programming, in: MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia, ACM, 2014, pp. 67–76. `doi:10.1145/2647868.2654949`.

[33] A. Guzzi, L. Hattori, M. Lanza, M. Pinzger, A. Deursen, Collective code bookmarks for program comprehension, in: IEEE International Conference on Program Comprehension, 2011, pp. 101–110. `doi:10.1109/ICPC.2011.19`.

[34] M.-A. Storey, J. Ryall, J. Singer, D. Myers, L.-T. Cheng, M. Muller, How software developers use tagging to support reminding and refinding, IEEE Transactions on Software Engineering 35 (4) (2009) 470–483. `doi:10.1109/TSE.2009.15`.

[35] M. Revelle, T. Broadbent, D. Coppit, Understanding concerns in software: Insights gained from two case studies, in: Proceedings - IEEE Workshop on Program Comprehension, 2005, pp. 23–32. `doi:10.1109/WPC.2005.43`.

[36] P. Jablonski, D. Hou, Aiding software maintenance with copy-and-paste clone-awareness, in: IEEE International Conference on Program Comprehension, 2010, pp. 170–179. `doi:10.1109/ICPC.2010.22`.

[37] F. Jacob, D. Hou, P. Jablonski, Actively comparing clones inside the code editor, in: Proceedings - International Conference on Software Engineering, 2010, pp. 9–16. doi:10.1145/1808901.1808903.

[38] B. Hartmann, M. Dhillon, M. Chan, HyperSource: Bridging the gap between source and code-related web sites, in: Conference on Human Factors in Computing Systems - Proceedings, 2011, pp. 2207–2210. doi:10.1145/1978942.1979263.

[39] A. Bradley, G. Murphy, Supporting software history exploration, in: Proceedings - International Conference on Software Engineering, 2011, pp. 193–202. doi:10.1145/1985441.1985469.

[40] S. Salinger, C. Oezbek, K. Beecher, J. Schenk, Saros: An Eclipse plug-in for distributed party programming, in: Proceedings - International Conference on Software Engineering, 2010, pp. 48–55. doi:10.1145/1833310.1833319.

[41] M. Goldman, G. Little, R. Miller, Real-time collaborative coding in a web IDE, in: UIST'11 - Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, 2011, pp. 155–164. doi:10.1145/2047196.2047215.

[42] H. Dang, V. Nguyen, K. Do, T. Tran, EduCo: An integrated social environment for teaching and learning software engineering courses, in: Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services, Vol. 04-06-December-2014, ACM, 2014, pp. 17–26. doi:10.1145/2684200.2684280.

[43] S. Hayashi, D. Hoshino, J. Matsuda, M. Saeki, T. Omori, K. Maruyama, Historef: A tool for edit history refactoring, in: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015 - Proceedings, IEEE, 2015, pp. 469–473. doi:10.1109/SANER.2015.7081858.

[44] S. K. Card, J. Mackinlay, The structure of the information visualization design space, in: Information Visualization, 1997. Proceedings., IEEE Symposium on, 1997, pp. 92–99. doi:10.1109/INFVIS.1997.636792.

[45] A. Bacchelli, M. Lanza, M. D'Ambros, Miler: A toolset for exploring email data, in: Proceedings - International Conference on Software Engineering, 2011, pp. 1025–1027. doi:10.1145/1985793.1985984.

[46] M. Boshernitsan, S. Graham, M. Hearst, Aligning development tools with the way programmers think about code changes, in: Conference on Human Factors in Computing Systems - Proceedings, 2007, pp. 567–576. doi:10.1145/1240624.1240715.

[47] L. Hattori, M. Lanza, Syde: A tool for collaborative software development, in: Proceedings - International Conference on Software Engineering, Vol. 2, 2010, pp. 235–238. doi:10.1145/1810295.1810339.

[48] C. Kästner, S. Apel, M. Kuhlemann, Granularity in software product lines, in: Proceedings - International Conference on Software Engineering, 2008, pp. 311–320. doi:10.1145/1368088.1368131.

[49] E. Nistor, A. Van Der Hoek, Explicit concern-driven development with ArchEvol, in: ASE2009 - 24th IEEE/ACM International Conference on Automated Software Engineering, 2009, pp. 185–196. `doi:10.1109/ASE.2009.70`.

[50] N. Saigal, J. Ligatti, Inline visualization of concerns, in: Proceedings - 7th ACIS International Conference on Software Engineering Research, Management and Applications, SERA09, 2009, pp. 95–102. `doi:10.1109/SERA.2009.13`.

[51] J. Meinicke, T. Thüm, R. Schröter, S. Krieter, F. Benduhn, G. Saake, T. Leich, FeatureIDE: Taming the preprocessor wilderness, in: Proceedings - International Conference on Software Engineering, IEEE Computer Society, 2016, pp. 629–632. `doi:10.1145/2889160.2889175`.

[52] B. Andam, A. Burger, T. Berger, M. Chaudron, FLOrIDA: Feature LOcatIon DAshboard for extracting and visualizing feature traces, in: Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems, ACM, 2017, pp. 100–107. `doi:10.1145/3023956.3023967`.

[53] Y. Zheng, C. Cu, H. Asuncion, Mapping features to source code through product line architecture: Traceability and conformance, in: Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017, IEEE, 2017, pp. 225–234. `doi:10.1109/ICSA.2017.13`.

[54] H. Fan, C. Sun, H. Shen, ATCoPE: Any-time collaborative programming environment for seamless integration of real-time and non-real-time teamwork in software development, in: GROUP'12 - Proceedings of the ACM 2012 International Conference on Support Group Work, 2012, pp. 107–116. `doi:10.1145/2389176.2389194`.

[55] D. Moody, The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering, IEEE Trans. Softw. Eng. 35 (6) (2009) 756–779. `doi:10.1109/TSE.2009.67`.

[56] T. Harmon, R. Klefstad, Interactive back-annotation of worst-case execution time analysis for Java microprocessors, in: Proceedings - 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2007, 2007, pp. 209–216. `doi:10.1109/RTCSA.2007.44`.

[57] T. Lieber, J. Brandt, R. Miller, Addressing misconceptions about code with always-on programming visualizations, in: Conference on Human Factors in Computing Systems - Proceedings, ACM, 2014, pp. 2481–2490. `doi:10.1145/2556288.2557409`.

[58] J.-P. Kramer, J. Kurz, T. Karrer, J. Borchers, How live coding affects developers' coding behavior, in: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, IEEE Computer Society, 2014, pp. 5–8. `doi:10.1109/VLHCC.2014.6883013`.

[59] P. Schugerl, J. Rilling, P. Charland, Beyond generated software documentation - a Web 2.0 perspective, in: IEEE International Conference on Software Maintenance, ICSM, 2009, pp. 547–550. `doi:10.1109/ICSM.2009.5306385`.

[60] W. Griswold, J. Yuan, Y. Kato, Exploiting the map metaphor in a tool for software evolution, in: Proceedings - International Conference on Software Engineering, 2001, pp. 265–274. `doi:10.1109/ICSE.2001.919100`.

[61] U. Dekel, J. Herbsleb, Improving API documentation usability with knowledge pushing, in: Proceedings - International Conference on Software Engineering, 2009, pp. 320–330. doi:10.1109/ICSE.2009.5070532.

[62] J. Cross II, T. Hendrix, L. Barowski, Combining dynamic program viewing and testing in early computing courses, in: Proceedings - International Computer Software and Applications Conference, 2011, pp. 184–192. doi:10.1109/COMPSAC.2011.31.

[63] B. Alsallakh, P. Bodesinsky, A. Gruber, S. Miksch, Visual tracing for the Eclipse Java debugger, in: Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2012, pp. 545–548. doi:10.1109/CSMR.2012.72.

[64] R. Cottrell, R. Walker, J. Denzinger, Semi-automating small-scale source code reuse via structural correspondence, in: Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2008, pp. 214–225. doi:10.1145/1453101.1453130.

[65] D. Li, S. Hao, W. Halfond, R. Govindan, Calculating source line level energy information for Android applications, in: 2013 International Symposium on Software Testing and Analysis, ISSTA 2013 - Proceedings, 2013, pp. 78–89. doi:10.1145/2483760.2483780.

[66] X. Rong, S. Yan, S. Oney, M. Dontcheva, E. Adar, CodeMend: Assisting interactive programming with bimodal embedding, in: UIST 2016 - Proceedings of the 29th Annual Symposium on User Interface Software and Technology, ACM, 2016, pp. 247–258. doi:10.1145/2984511.2984544.

[67] P. Bathia, B. Beerelli, M.-A. Laverdière, Assisting programmers resolving vulnerabilities in Java web applications, in: Communications in Computer and Information Science, Vol. 133 CCIS, 2011, pp. 268–279. doi:10.1007/978-3-642-17881-8_26.

[68] B. Biegel, S. Baltes, I. Scarpellini, S. Diehl, CodeBasket: Making developers' mental model visible and explorable, in: Proceedings - 2nd International Workshop on Context for Software Development, CSD 2015, IEEE, 2015, pp. 20–24. doi:10.1109/CSD.2015.12.

[69] B. Cossette, R. Walker, DSketch: Lightweight, adaptable dependency analysis, in: Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2010, pp. 297–306. doi:10.1145/1882291.1882335.

[70] C. Hundhausen, J. Brown, What You See Is What You Code: A "live" algorithm development and visualization environment for novice learners, Journal of Visual Languages and Computing 18 (1) (2007) 22–47. doi:10.1016/j.jvlc.2006.03.002.

[71] J. Ou, M. Vechev, O. Hilliges, An interactive system for data structure development, in: Conference on Human Factors in Computing Systems - Proceedings, Vol. 2015-April, ACM, 2015, pp. 3053–3062. doi:10.1145/2702123.2702319.

[72] F. Beck, B. Dit, J. Velasco-Madden, D. Weiskopf, D. Poshyvanyk, Rethinking user interfaces for feature location, in: IEEE International Conference on Program Comprehension, Vol. 2015-August, IEEE Computer Society, 2015, pp. 151–162. doi:10.1109/ICPC.2015.24.

[73] C. Müller, G. Reina, T. Ertl, In-situ visualisation of fractional code ownership over time, in: Proceedings of the 8th International Symposium on Visual Information Communication and Interaction, ACM, 2015, pp. 13–20. doi:10.1145/2801040.2801055.

[74] T. Barik, J. Witschey, B. Johnson, E. Murphy-Hill, Compiler error notifications revisited: An interaction-first approach for helping developers more effectively comprehend and resolve error notifications, in: 36th International Conference on Software Engineering, ICSE Companion 2014 - Proceedings, ACM, 2014, pp. 536–539. doi:10.1145/2591062.2591124.

[75] R. Findler, J. Clements, C. Flanagan, M. Flatt, S. Krishnamurthi, P. Steckler, M. Felleisen, DrScheme: A programming environment for Scheme, Journal of Functional Programming 12 (2) (2002) 159–182. doi:10.1017/S0956796801004208.

[76] A. Guzzi, A. Begel, Facilitating communication between engineers with CARES, in: Proceedings - International Conference on Software Engineering, 2012, pp. 1367–1370. doi:10.1109/ICSE.2012.6227246.

[77] R. Tairas, J. Gray, Increasing clone maintenance support by unifying clone detection and refactoring activities, Information and Software Technology 54 (12) (2012) 1297–1307. doi:10.1016/j.infsof.2012.06.011.

[78] F. Kistner, M. Beth Kery, M. Puskas, S. Moore, B. Myers, Moonstone: Support for understanding and writing exception handling code, in: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, Vol. 2017-October, IEEE Computer Society, 2017, pp. 63–71. doi:10.1109/VLHCC.2017.8103451.

[79] A. Breckel, M. Tichy, Embedding programming context into source code, in: IEEE International Conference on Program Comprehension, Vol. 2016-July, IEEE Computer Society, 2016. doi:10.1109/ICPC.2016.7503732.

[80] F. Beck, S. Gulan, B. Biegel, S. Baltes, D. Weiskopf, RegViz: Visual debugging of regular expressions, in: 36th International Conference on Software Engineering, ICSE Companion 2014 - Proceedings, ACM, 2014, pp. 504–507. doi:10.1145/2591062.2591111.

[81] D. Kawrykow, M. Robillard, Improving API usage through automatic detection of redundant code, in: ASE2009 - 24th IEEE/ACM International Conference on Automated Software Engineering, 2009, pp. 111–122. doi:10.1109/ASE.2009.62.

[82] D. Myers, M.-A. Storey, Using dynamic analysis to create trace-focused user interfaces for IDEs, in: Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2010, pp. 367–368. doi:10.1145/1882291.1882351.

[83] T. Karrer, J.-P. Krämer, J. Diehl, B. Hartmann, J. Borchers, Stacksplorer: Call graph navigation helps increasing code maintenance efficiency, in: UIST'11 - Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, 2011, pp. 217–224. doi:10.1145/2047196.2047225.

[84] W. Nunez, V. Marin, C. Rivero, ARCC: Assistant for repetitive code comprehension, in: Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Vol. Part F130154, ACM, 2017, pp. 999–1003. doi:10.1145/3106237.3122824.

[85] S. Baltes, P. Schmitz, S. Diehl, Linking sketches and diagrams to source code artifacts, in: Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Vol. 16-21-November-2014, ACM, 2014, pp. 743–746. doi:10.1145/2635868.2661672.

[86] P. Ohmann, B. Liblit, CSIclipse: Presenting crash analysis data to developers, in: ETX 2015 - Proceedings of the Eclipse Technology eXchange, ACM, 2015, pp. 7–12. doi:10.1145/2846650.2846651.

[87] A. Olszak, B. N. Jørgensen, Remodularizing Java programs for improved locality of feature implementations in source code, Science of Computer Programming 77 (3) (2012) 131–151. doi:10.1016/j.scico.2010.10.007.

[88] A. Banerjee, H.-F. Guo, A. Roychoudhury, Debugging energy-efficiency related field failures in mobile apps, in: Proceedings - International Conference on Mobile Software Engineering and Systems, MOBILESoft 2016, ACM, 2016, pp. 127–138. doi:10.1145/2897073.2897085.

[89] B. Fluri, J. Zuberbühler, H. Gall, Recommending method invocation context changes, in: Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2008, pp. 1–5. doi:10.1145/1454247.1454249.

[90] L. Lichtschlag, L. Spychalski, J. Bochers, CodeGraffiti: Using hand-drawn sketches connected to code bases in navigation tasks, in: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, IEEE Computer Society, 2014, pp. 65–68. doi:10.1109/VLHCC.2014.6883024.

[91] R.-H. Pfeiffer, A. Wąsowski, TexMo: A multi-language development environment, in: Lecture Notes in Computer Science, Vol. 7349 LNCS, 2012, pp. 178–193. doi:10.1007/978-3-642-31491-9_15.

[92] R. Holmes, R. Walker, Task-specific source code dependency investigation, in: VISSOFT 2007 - Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007, pp. 100–107. doi:10.1109/VISSOF.2007.4290707.

[93] M. Eyl, C. Reichmann, K. Müller-Glaser, Traceability in a fine grained software configuration management system, in: Lecture Notes in Business Information Processing, Vol. 269, Springer Verlag, 2017, pp. 15–29. doi:10.1007/978-3-319-49421-0_2.

[94] J. Smith, C. Brown, E. Murphy-Hill, Flower: Navigating program flow in the IDE, in: Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC, Vol. 2017-October, IEEE Computer Society, 2017, pp. 19–23. doi:10.1109/VLHCC.2017.8103445.

[95] T. Barik, Y. Song, B. Johnson, E. Murphy-Hill, From quick fixes to slow fixes: Reimagining static analysis resolutions to enable design space exploration, in: Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, IEEE, 2016, pp. 211–221. doi:10.1109/ICSME.2016.63.

[96] M. Desmond, M.-A. Storey, C. Exton, Fluid source code views, in: IEEE International Conference on Program Comprehension, Vol. 2006, 2006, pp. 260–263. doi:10.1109/ICPC.2006.24.

[97] X. Ge, E. Murphy-Hill, Manual refactoring changes with automated refactoring validation, in: Proceedings - International Conference on Software Engineering, no. 1, IEEE Computer Society, 2014, pp. 1095–1105. doi:10.1145/2568225.2568280.

[98] A. Murolo, F. Stutz, M. Husmann, M. Norrie, Improved developer support for the detection of cross-browser incompatibilities, in: Lecture Notes in Computer Science, Vol. 10360 LNCS, Springer Verlag, 2017, pp. 264–281. doi:10.1007/978-3-319-60131-1_15.

[99] H. Kang, P. Guo, Omnicode: A novice-oriented live programming environment with always-on run-time value visualizations, in: UIST 2017 - Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, ACM, 2017, pp. 737–745. doi:10.1145/3126594.3126632.

[100] S. Lee, Y. Chen, N. Klugman, S. Gouravajhala, A. Chen, W. Lasecki, Exploring coordination models for ad hoc programming teams, in: Conference on Human Factors in Computing Systems - Proceedings, Vol. Part F127655, ACM, 2017, pp. 2738–2745. doi:10.1145/3027063.3053268.

[101] A. Bragdon, R. Zeleznik, S. Reiss, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, J. Laviola Jr., Code Bubbles: A working set-based interface for code understanding and maintenance, in: Conference on Human Factors in Computing Systems - Proceedings, Vol. 4, 2010, pp. 2503–2512. doi:10.1145/1753326.1753706.

[102] J. Xie, B. Chu, H. Lipford, J. Melton, ASIDE: IDE support for web application security, in: Proceedings of the 27th Annual Computer Security Applications Conference, 2011, pp. 267–276. doi:10.1145/2076732.2076770.

[103] X. Ge, Q. DuBose, E. Murphy-Hill, Reconciling manual and automatic refactoring, in: Proceedings - International Conference on Software Engineering, 2012, pp. 211–221. doi:10.1109/ICSE.2012.6227192.

[104] L. Do, K. Ali, B. Livshits, E. Bodden, J. Smith, E. Murphy-Hill, Cheetah: Just-in-time taint analysis for Android apps, in: Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering Companion, ICSE-C 2017, IEEE, 2017, pp. 39–42. doi:10.1109/ICSE-C.2017.20.

[105] N. Lopez, A. Van Der Hoek, The Code Orb - supporting contextualized coding via at-a-glance views (NIER track), in: Proceedings - International Conference on Software Engineering, 2011, pp. 824–827. doi:10.1145/1985793.1985914.

[106] M. Toomim, A. Begel, S. Graham, Managing duplicated code with linked editing, in: Proceedings - 2004 IEEE Symposium on Visual Languages and Human Centric Computing, 2004, pp. 173–180. doi:10.1109/VLHCC.2004.35.

[107] B. Steinert, M. Taeumel, J. Lincke, T. Pape, R. Hirschfeld, CodeTalk - conversations about code, in: 8th International Conference on Creating, Connecting and Collaborating through Computing, C5 2010, 2010, pp. 11–18. doi:10.1109/C5.2010.11.

[108] J. Lautamäki, A. Nieminen, J. Koskinen, T. Aho, T. Mikkonen, M. Englund, CoRED: Browser-based collaborative real-time editor for Java web applications, in: Proceedings of

the ACM Conference on Computer Supported Cooperative Work, CSCW, 2012, pp. 1307–1316. doi:10.1145/2145204.2145399.

[109] B. Goncharenko, V. Zaytsev, Language design and implementation for the domain of coding conventions, in: SLE 2016 - Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering, co-located with SPLASH 2016, ACM, 2016, pp. 90–104. doi:10.1145/2997364.2997386.

[110] R. DeLine, A. Bragdon, K. Rowan, J. Jacobsen, S. Reiss, Debugger Canvas: Industrial experience with the Code Bubbles paradigm, in: Proceedings - International Conference on Software Engineering, 2012, pp. 1064–1073. doi:10.1109/ICSE.2012.6227113.

[111] S. Chiba, M. Horie, K. Kanazawa, F. Takeyama, Y. Teramoto, Do we really need to extend syntax for advanced modularity?, in: AOSD'12 - Proceedings of the 11th Annual International Conference on Aspect Oriented Software Development, 2012, pp. 95–106. doi:10.1145/2162049.2162061.

[112] J. Edwards, Example centric programming, ACM SIGPLAN Notices 39 (12) (2004) 84–91. doi:10.1145/1052883.1052894.

[113] D. Röthlisberger, O. Greevy, O. Nierstrasz, Exploiting runtime information in the ide, in: IEEE International Conference on Program Comprehension, 2008, pp. 63–72. doi:10.1109/ICPC.2008.32.

[114] G. Ganea, I. Verebi, R. Marinescu, Continuous quality assessment with inCode, Science of Computer Programming 134 (2017) 19–36. doi:10.1016/j.scico.2015.02.007.

[115] S. Hupfer, L.-T. Cheng, S. Ross, J. Patterson, Introducing collaboration into an application development environment, in: Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW, 2004, pp. 21–24. doi:10.1145/1031607.1031611.

[116] A. Ferrari, D. Garbervetsky, V. Braberman, P. Listingart, S. Yovine, JScoper: Eclipse support for research on scoping and instrumentation for real time Java applications, in: Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange, eclipse'05, 2005, pp. 50–54. doi:10.1145/1117696.1117707.

[117] R. Witte, B. Sateli, N. Khamis, J. Rilling, Intelligent software development environments: Integrating natural language processing with the Eclipse platform, in: Lecture Notes in Computer Science, Vol. 6657 LNAI, 2011, pp. 408–419. doi:10.1007/978-3-642-21043-3_49.

[118] J. Cito, P. Leitner, H. Gall, A. Dadashi, A. Keller, A. Roth, Runtime metric meets developer: Building better cloud applications using feedback, in: Onward! 2015 - Proceedings of the 2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Part of SPLASH 2015, ACM, 2015, pp. 14–27. doi:10.1145/2814228.2814232.

[119] M. Boland, C. Clifton, Introducing PyLighter: Dynamic code highlighter, SIGCSE Bulletin 41 (1) (2009) 489–493. doi:10.1145/1539024.1509037.

[120] E. Murphy-Hill, A. Black, Breaking the barriers to successful refactoring: Observations and tools for extract method, in: Proceedings - International Conference on Software Engineering, 2008, pp. 421–430. doi:10.1145/1368088.1368146.

[121] J. Lhoták, O. Lhoták, L. Hendren, Integrating the soot compiler infrastructure into an IDE, in: Lecture Notes in Computer Science, Vol. 2985, 2004, pp. 281–297. doi:10.1007/978-3-540-24723-4_19.

[122] C. De Roover, C. Noguera, A. Kellens, V. Jonckers, The SOUL tool suite for querying programs in symbiosis with Eclipse, in: Proceedings of the 9th International Conference on the Principles and Practice of Programming in Java, PPPJ 2011, 2011, pp. 71–80. doi:10.1145/2093157.2093168.

[123] S. McDirmid, Usable live programming, in: SPLASH Indianapolis 2013: Onward! 2013 - Proceedings of the 2013 International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, ACM, 2013, pp. 53–61. doi:10.1145/2509578.2509585.

[124] S. Ratanotayanon, S. Sim, R. Gallardo-Valencia, Supporting program comprehension in agile with links to user stories, in: Proceedings - 2009 Agile Conference, AGILE 2009, 2009, pp. 26–32. doi:10.1109/AGILE.2009.66.

[125] M. Sulír, J. Porubän, Augmenting source code lines with sample variable values, in: Proceedings of the 2018 26th IEEE/ACM International Conference on Program Comprehension (ICPC), 2018. doi:10.1145/3196321.3196364.

[126] J. Hoffswell, A. Satyanarayan, J. Heer, Augmenting code with in situ visualizations to aid program understanding, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18, ACM, New York, NY, USA, 2018, pp. 532:1–532:12. doi:10.1145/3173574.3174106.