Visualizing the Workflow of Developers

Roberto Minelli and Michele Lanza
REVEAL @ Faculty of Informatics — University of Lugano, Switzerland

***Abstract*—Developers use the Integrated Development Environment (IDE) to develop a system at hand, by reading, understanding, and writing its source code. They do so by exploiting the tools and facilities provided by the IDE. This also allows them to build a mental model of the system to perform informed changes. It is however not clear how and when developers use which facility and tool, and to what extent the current services offered by the IDE appropriately support the navigation.**

We present an approach to visualize the activities of developers within the IDE, implemented in a tool: DFlow. DFlow records all IDE interactions that occur during a development session and visualizes them through a web-based visualization platform.

I. Introduction

Developers spend much time interacting with Integrated Development Environments (*i.e.,* IDEs), such as Eclipse[1]. In addition to writing new code, they use IDEs to comprehend existing code by building a mental model of the system. Kersten & Murphy argued that *"programmers spend more time navigating the code than working with it"* [1], which throws up the question whether IDEs appropriately support the navigation. For example, Lee *et al.* claimed that the current support for refactoring is unintuitive and inefficient [2].

We want to investigate how, when, and why developers use the IDE to navigate the software space and to what extent IDEs support the navigation. We present an approach to visualize the events that happen while a developer is working on a system. We implemented the approach in DFlow, which seamlessly records all the IDE interactions while a developer is working (*e.g.,* addition of a class, inspection of senders/implementors of a method, steps in a debugger). DFlow enables retrospective analyses to understand and characterize the development session through web-based interactive visualizations.

Researchers proposed a number of approaches and tools to enhance IDEs and to better support the issues related to navigation. Röthlisberger *et al.* developed Autumn Leaves, a tool to cure the window plague [3]. While navigating the system, developers are often forced to spawn several windows or open tabs, many of which quickly became useless. Their tool automatically suggests which windows or tabs can be closed, without impacting the current development session, to reduce the amount of noise present in the IDE. Singer *et al.* developed Navtracks, a tool that records the navigation history of developers and better support browsing through software [4]. Kersten *et al.* developed Mylar [1], which monitors the programmer's activity and extracts a degree-of-interest (DOI) model associated to each program element. They later use this model to infer the task context of program elements spread across a codebase. Yoon *et al.* proposed Fluorite, a low-level event logging for the Eclipse IDE [5]. The tool records interactions in the code editor and it is aimed at evaluating existing tools. Development sessions are a valuable asset for program comprehension. Robbes and Lanza proposed an approach to record semantic changes in real time, implemented in a tool called SpyWare [6]. The authors used the collected data to understand and characterize the development sessions to enhance program comprehension. DFlow differs from the related work, by focusing on the GUI-level events and by proposing novel ways to visualize this type of information. With SpyWare we share the goal of wanting to understand and characterize development sessions. Fluorite, Navtracks, and Mylar record different kinds of IDE interactions and provide links to related entities but, for example, they do not provide means to visualize or comprehend a development session.

We describe DFlow, detail the custom visualization we devised, and illustrate them through scenarios.

II. DFlow

DFlow is composed of DFlow-Pharo, an extension to the Pharo[2] Smalltalk IDE, and DFlow-Web, a web-based visualization platform. Figure 1 illustrates DFlow-Pharo (next to standard Pharo windows (1)), featuring a session manager (2) and a session browser (3).

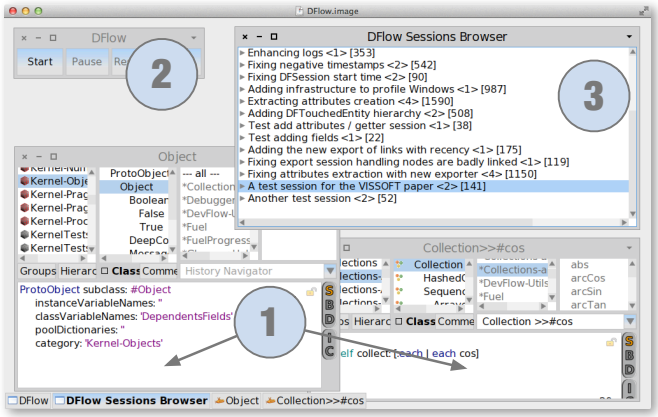

Fig. 1: The main window of the Pharo IDE with DFlow.

The manager allows the user to start, pause, resume, and stop development sessions. The browser provides means to inspect the recorded sessions and to perform a number of operations on them (*e.g.,* export, rename, delete, merge).

[1]See `http://www.eclipse.org`

[2]See `http://www.pharo-project.org/`

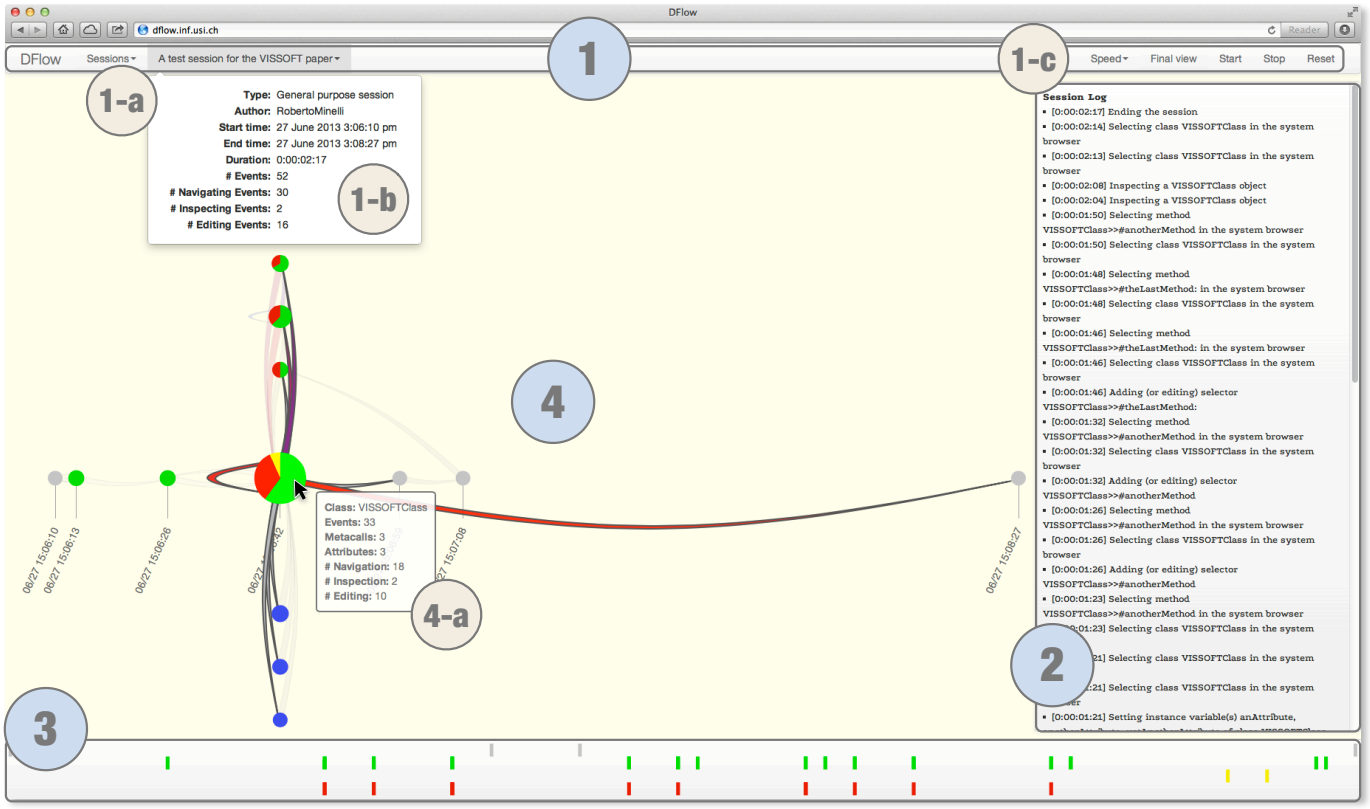

Fig. 2: DFlow-Web, composed of (1) a Navigation Bar, (2) a Session Log, (3) a Timeline, and (4) the Visualization Canvas.

Figure 2 shows DFlow-Web, the front-end of DFlow, implemented as a web application. It is composed of:

1) **Navigation Bar** to configure the visualization and browse information.
   a) *Select Session Menu* to let the user to select the session she wants to analyze.
   b) *Session Information Panel* to provide additional pieces of information about the session.
   c) *Replay Menu* to step into the session (*i.e.,* start & stop) and to configure the speed of the animations.
2) **Session Log** to show a time-ordered textual description of each event that happened during a session.
3) **Timeline** to represent the events happening in a session divided according to their category: handling, navigating, inspecting, and editing.
4) **Visualization Canvas** for the interactive visualizations.
   a) *Entity Information Panel* to reveal additional pieces of information about the hovered entity.

A. Visualizing a Development Session

We devised a custom visualization to depict a development session. Figure 3 shows the principles behind the view and the color mappings we use for the entities. We depict a development session using a *directed graph* in which nodes are the entities involved in the current session (classes, methods, attributes). The directed links (*i.e.,* source destination) depict "navigation-paths" and not structural relations.

Principles and proportions. Nodes are either classes, methods, attributes, or session handling events (*e.g.,* start, pause, stop, resume). The radius of a node is proportional to the number of events on that entity in the sessions (*i.e.,* how many times the user interacted, directly or indirectly, with this entity). We divided events in three categories: navigating (*i.e.,* green), inspecting (*i.e.,* yellow), and editing (*i.e.,* red). Navigation events are the less intrusive events (they do not modify the entity) while editing events represent the "real" editing activities (*e.g.,* adding/modifying a method/class). Inspection events do not modify the entity, but represent deeper forms of navigation (*e.g.,* inspecting the internals of an object). Class and method nodes are depicted as pie charts presenting the event distribution of that entity at a glance. Handling events are depicted in grey. Links depict "navigation paths" between entities, *e.g.,* if the developers creates `Class A` and right after browses the Method `foo` of Class `B` (*i.e.,* `B#>>foo`) DFlow-Web draws a directed link from `Class A` to `B#>>foo`.

The width of the link is proportional to the number of occurrences of that navigation path in the session. We use the color to present information about the age of the links. We divided links in quartiles, according to their age. For the first three quartiles (*i.e.,* old links) we used three tones of gray with increasing saturation, while for the last quartile (*i.e.,* the most recent links) we use a gradient from blue to red.

Class nodes and session handling nodes are positioned on a horizontal line, whose x-coordinate represents the temporal

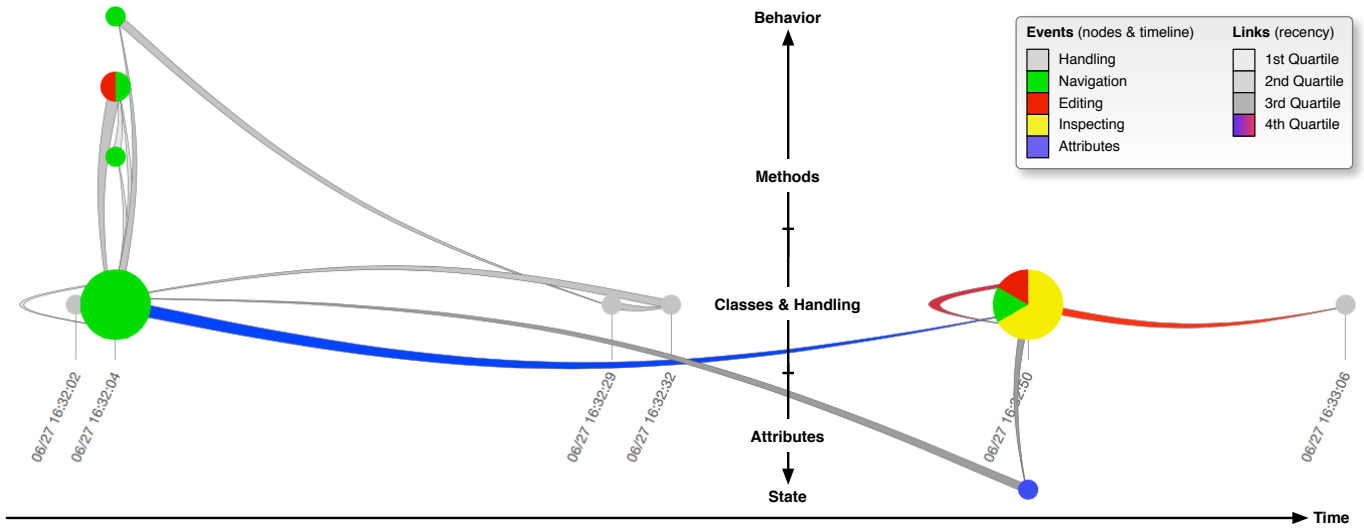

Fig. 3: Visualizing development sessions. The figure shows the principles behind the view and the color mappings used.

dimension of a session. Method nodes and attributes are not mapped to that time scale, but they take the x-position of their owner node (*i.e.,* a class node), unless the owner node is not part of the visualization. This visual cue helps one to perceive to what extent a class has been touched during a session.

The bottom timeline presents an outline of the session, where each rectangle is an event. The color of events follows the same color scale of the pie-charts. The y-coordinate of each rectangle represents one of the four categories of events (*i.e.,* handling, navigation, inspection, and editing). The x-coordinate of each rectangle represents the timestamp of the event it depicts.

B. *Interacting with* DFlow-Web

DFlow-Web is interactive. The user can pan (*i.e.,* drag & drop) and zoom (*i.e.,* mouse wheel) the view inside the canvas of DFlow-Web (Figure 2.4). The zoom performs an x-axis rescale and restricts the time interval being displayed. This helps to better understand the visualization at time steps where events have a high density. The user can also drag & drop single nodes to better understand links between the nodes. The "Replay menu" (Figure 2.1-c) offers additional means to interact with the view. DFlow records all the time steps of a session, and DFlow-Web is able to produce an animation of the session where the view evolves together with the session. Figure 4 shows three time steps from the evolution of a session.

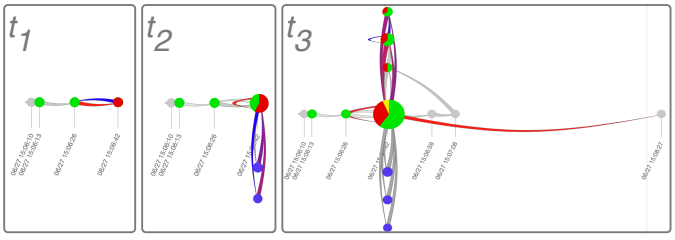

Fig. 4: Three time steps of the same session.

C. *Under the Hood*

DFlow-Pharo contains a suite of profilers which seamlessly observes and records all the IDE interactions of the developers. The profilers produce a large amount of raw data which, in addition to some meta data, compose a "development session". Another component, the session analyzer, receives the data, mines them, and produces a `JSON` representation of each session. The front-end, DFlow-Web, receives such `JSON` files, constructs the visualizations, and lets the user interact with them.

D. *The Corpus*

DFlow features 20 recorded development sessions, summarized in Table I. Most of the sessions were recorded by the first author of this paper while developing DFlow itself, while sessions 16 and 18 are courtesy of Mr. Andrei Chiş, a Ph.D. student from the University of Bern.

No.	Type	Duration (d:hh:mm:ss)	Events *Navigation*	*Inspection*	*Editing*	*Total*
1	General	0:00:01:04	10	4	3	21
2	General	0:00:02:17	30	2	16	52
3	Bug-fixing	0:00:02:52	22	1	0	27
4	Enhancement	0:00:05:13	42	0	2	46
5	Bug-fixing	0:00:05:19	20	0	1	23
6	Enhancement	0:00:06:46	47	2	5	56
7	Enhancement	0:00:09:57	58	5	15	82
8	Enhancement	0:00:17:23	131	4	26	165
9	Enhancement	0:00:21:35	223	0	32	257
10	Enhancement	0:00:27:39	101	14	24	141
11	Refactoring	0:00:35:15	211	1	92	308
12	Enhancement	0:03:26:21	457	126	54	651
13	Bug-fixing	0:06:34:43	319	25	34	386
14	Enhancement	0:06:40:55	385	88	63	544
15	Refactoring	0:20:59:00	896	26	183	1115
16	General	0:23:30:48	1200	135	113	1454
17	Bug-fixing	1:00:41:59	160	9	14	187
18	General	0:01:48:42	466	92	52	612
19	Enhancement	2:18:32:36	58	0	10	70
20	Enhancement	3:16:39:23	823	80	142	1081

TABLE I: The current corpus available to DFlow.

III. Telling Development Stories with DFlow

We used DFlow-Web to analyze the corpus sessions presented in Section II-D. We describe some insights and reflections by means of two scenarios.

A. Scenario 1: High Navigation Stacks & Back-Links.

Figure 5 shows part of an "enhancement session" recorded during the development of DFlow itself. There are 3 main stacks of events highlighted in the Figure. Stacks A and B refer to user-defined classes, `DFSessionAnalyzer` and `DFJSONTouchedEntityNode`. Stack C is a chain of navigation events involving the `String` class and its methods. High navigation stacks denote that the developer is browsing the API of some class to find a specific piece of functionality.

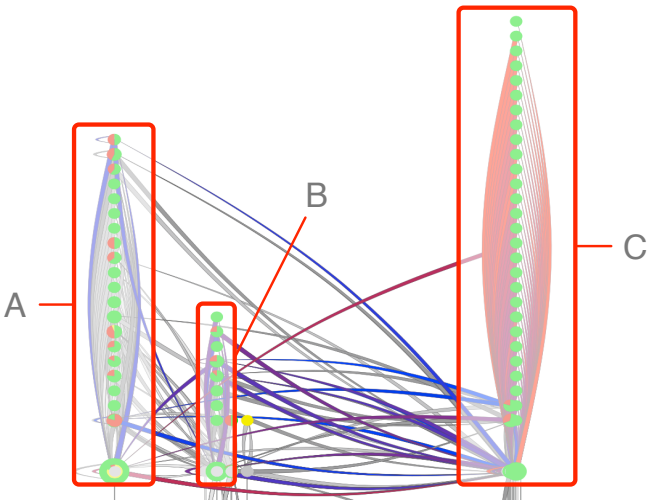

Fig. 5: DFlow depicting a fraction of the session No. 20.

Figure 6 shows a later fraction of the same session.

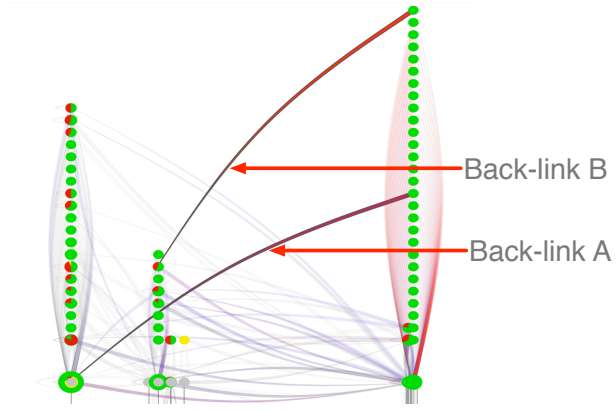

Fig. 6: DFlow depicting time step t_2 of the session No. 20.

In the figure we marked two special links, A and B. These are "back-links", where the developer, after repeatedly browsing some other class, has gathered enough knowledge to "go back" to another entity and finally perform an informed modification.

B. Scenario 2: Different Type, Different Shape.

Figure 7 shows a "bug-fixing session".

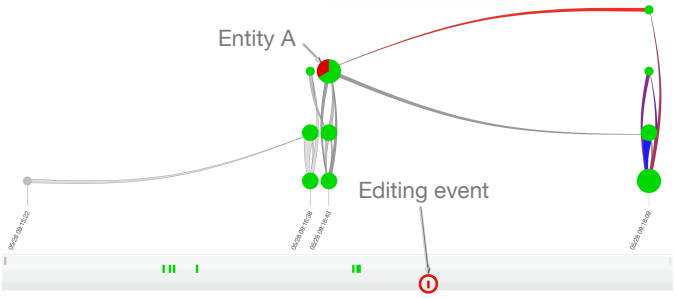

Fig. 7: DFlow depicting the session No. 5.

From both the bottom timeline, and the visualization, we see that this session features a series of navigation events (*i.e.,* green) and one single editing (*i.e.,* red). This denotes the fact that the user has been navigating code to gather information about the system to fix one specific entity, marked as A.

IV. Conclusion

We presented a novel approach to visualize the workflow of developers, implemented in DFlow. DFlow seamlessly records all the IDE interactions, and enables retrospective analyses through interactive web-based visualizations. This is still work at an early stage. In the future we plan to have developers use our tool to enlarge our data set, with the goal of collecting further insights and refining what seems to be an emergent pattern language of session types. Further plans include reflecting on which navigation events are performed in which sequence, and whether we could use that information to suggest to developers other ways of using the functionalities offered by the IDE.

Acknowledgements. We gratefully acknowledge the Swiss National Science foundation's support for the project "HI-SEA" (SNF Project No. 146734).

References

[1] M. Kersten and G. C. Murphy, "Mylar: a degree-of-interest model for IDEs," in *Proceedings of the 4th Int'l Conference on Aspect-Oriented Software Development (AOSD)*, 2005, pp. 159–168.

[2] Y. Y. Lee, N. Chen, and R. E. Johnson, "Drag-and-drop refactoring: intuitive and efficient program transformation," in *Proceedings of the 35th Int'l Conference on Software Engineering (ICSE)*, 2013, pp. 23–32.

[3] D. Roethlisberger, O. Nierstrasz, and S. Ducasse, "Autumn leaves: Curing the window plague in IDEs," in *Proceedings of the 16th Working Conference on Reverse Engineering (WCRE)*, 2009, pp. 237–246.

[4] J. Singer, R. Elves, and M. Storey, "Navtracks: supporting navigation in software maintenance," in *Proceedings of the 21st IEEE Int'l Conference on Software Maintenance (ICSM)*, 2005, pp. 325–334.

[5] Y. Yoon and B. A. Myers, "Capturing and analyzing low-level events from the code editor," in *Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU)*, 2011, pp. 25–30.

[6] R. Robbes and M. Lanza, "Characterizing and understanding development sessions," in *Proceedings of the 15th IEEE Int'l Conference on Program Comprehension (ICPC)*, 2007, pp. 155–166.