The Plague Doctor: A Promising Cure for the Window Plague

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

***Abstract*—Modern Integrated Development Environments (IDEs) are often affected by the "*window plague*", an overly crowded workspace with many open windows and tabs. The main cause is the lack of navigation support in IDEs, also due to the many—and not always obvious—complex relationships that exist between program entities.**

Researchers have shown that it is possible to mitigate the window plague by exploiting the data obtained by monitoring how developers interact with the user interface of the IDE. However, despite initial results the approach was never fully integrated in an IDE.

In our previous work, we implemented DFLOW, an automatic interaction profiler that monitors all the fine-grained interactions of the developer with the IDE. Here we present a first prototype of the PLAGUE DOCTOR, a tool that seamlessly detects the windows that are less likely to be used in the future and automatically closes them. We discuss our long term vision on how to fully exploit the interaction data recorded by DFLOW to provide a more effective cure for the window plague.

I. INTRODUCTION

Modern language paradigms, such as object oriented programming, introduced a number of benefits in terms of how software systems are developed, structured, and organized: Better separation of concerns, modularity, and reusability are mere examples. However, this comes at a cost: Program entities are organized in hierarchies, stored over complex repositories, and thus there can be complex, hidden, and transitive relationships among them [1], [2]. This significantly hampers program comprehension that, among other objectives, aims to explore and understand such complex relationships.

Integrated Development Environments (IDEs) are the primary means developers use to manipulate source code. To explore source code artifacts, IDEs provide two main user interface (UI) paradigms: *window-based*, like in the PHARO IDE[1], or *tab-based*, like in the ECLIPSE IDE[2]. Neither of the two paradigms effectively supports the navigation of the complex software space [3]. In fact, both paradigms force developers to open one tab (or window) per program entity, leading to what researchers called "*window plague*" [4]. The window plague is the tendency of IDEs to quickly become overcrowded by unused windows (or tabs). Figure 1 shows an IDE after only a few minutes of development manifesting the window plague. Moreover, IDEs do not keep track of relationships among windows and provide little or no support to automatically maintain a low level of entropy inside the IDE, *e.g.,* by closing unused windows.

[1]See http://pharo.org
[2]See http://eclipse.org

Researchers have shown that it is possible to mitigate the window plague by monitoring how developers interact with the UI of the IDE and exploiting such data [4]. Röthlisberger *et al.* developed a preliminary cure for the window plague in *Autumn Leaves*. *Autumn Leaves* is an extension for the PHARO IDE that detects windows that are unlikely needed for further use and closes them. It also adds visual clues to the more important ones to provide cognitive feedback to the IDE. The authors conducted a benchmark evaluation with promising results. Unfortunately, *Autumn Leaves* remained a prototype, it was never integrated in the IDE and no one could take advantage of its potential benefits. We believe that one of the reasons for this was overly coarse grained data leveraged by *Autumn Leaves*, and it remains an open issue to quantify how much the window plague hinders development.

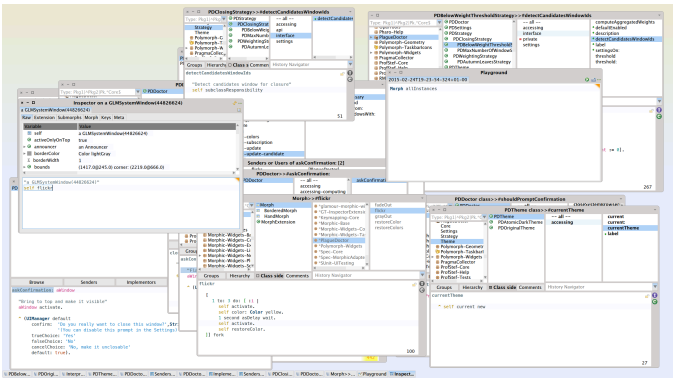

Fig. 1. A screenshot of the PHARO IDE manifesting the window plague.

In our previous work, we implemented DFLOW, a silent interaction profiler that automatically monitors all the fine-grained interactions of the developer with the IDE [5]. Util now we only used the recorded interactions *a posteriori* to visualize development sessions [6] or to study the behavior of developers, *e.g.,* measuring how developers spend their time in the IDE [5]. Our long-term goal is to leverage interaction data to better support the workflow of developers [7]. The current work is our first step towards this vision. Since the window plague is a relevant problem for the PHARO community, we devised and implemented a doctor for this plague. In this paper we present the PLAGUE DOCTOR, a tool inspired by *Autumn Leaves*, that exploits the data collected by DFLOW to mitigate the window plague. We also discuss how to fully exploit fine-grained interaction data to provide a more effective cure for the window plague.

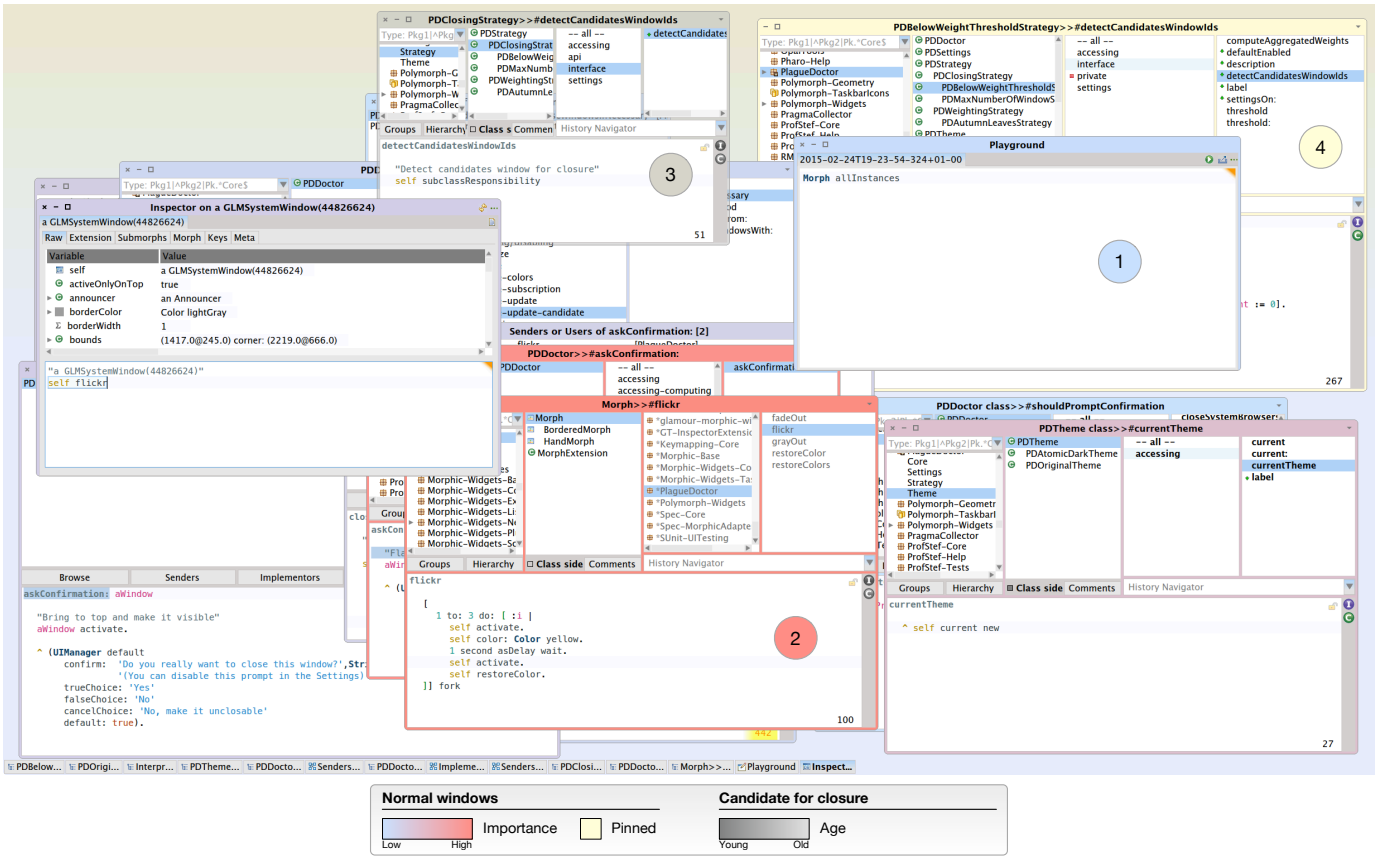

Fig. 2. A screenshot of the PHARO IDE with the PLAGUE DOCTOR installed depicting the same environment as in Figure 1.

II. THE PLAGUE DOCTOR

Figure 1 shows the PHARO IDE affected by the window plague after a few minutes of development. The UI presents the following problems:

- It is overcrowded by a significant number of apparently similar, overlapping windows;
- A developer cannot easily identify which windows are most relevant for the current development context and for her next task;
- Some of the opened windows—probably—deserve to be closed, but the IDE does not provide means to automatically lower its level of entropy, *e.g.,* closing windows.

Figure 2 shows the same environment depicted in Figure 1 with the PLAGUE DOCTOR enabled. The tool uses interaction data to compute the importance of windows, and thus the likelihood that they will be used again in the future. For example, a window becomes more important when a user types on it, or when she uses its UI components to perform a task.

The PLAGUE DOCTOR decorates the windows to keep the developer aware of their computed importance. A heat-scale from light blue (*i.e.,* less important, see Figure 2.1) to bright red (*i.e.,* more important, see Figure 2.2) reduces the cognitive load of a developer that faces this IDE: It is likely that she can concentrate her focus on the most warmer (important) windows and ignore the colder windows.

In addition to the color scale for the important windows, the PLAGUE DOCTOR lets the user identify the windows that are less likely to be used in the future. When less important windows become candidates for closure, the PLAGUE DOCTOR uses a gray scale from dark to bright gray to indicate their age, *i.e.,* in terms of how many user actions happened after the window became a candidate (see Figure 2.3). By default, the PLAGUE DOCTOR has a grace period of 5 user actions before closing a window that has been marked as a candidate. When the grace period expires, the doctor automatically closes the window, asking for user confirmation if configured to do so.

For some tasks, the developer may become aware that an unused window is still important for a future task, and thus she might require to avoid its closure in an explicit manner. The PLAGUE DOCTOR provides the ability to *pin* a window, that is, exclude it from the potential candidates for closure. Pinned windows are colored in light yellow (see Figure 2.4).

The PLAGUE DOCTOR is just the tip of the iceberg. Below the water silently lies DFLOW, our non-intrusive interaction data profiler [5]. While the developer programs, it observes all the user interactions, from UI events such as moving a window, to meta events such as the creation of a new class, down to the granularity of mouse and keyboard events. DFLOW then generates events that other tools, such as the PLAGUE DOCTOR, can intercept, process, and exploit.

A. Models and Strategies

The PLAGUE DOCTOR defines the *"importance"* (or weight) of a window in the current development context. To compute it, it maintains two weight models: the *window interaction model* and the *program entity model*. The global weight of a window, is computed by combining its weight from the window interaction model and the weight of the program entity displayed in the window itself (if any). To update these models the doctor uses a weighting strategy. Closing strategies, instead, determine which windows are candidates for closure. The user selects one weighting strategy and one or more closing strategies, *i.e.,* we call them *active* strategies. After every interaction the active strategies are applied: Models are updated, windows are decorated and closed, if needed.

The **Program Entity Model** associates a weight to each program entity (*i.e.,* class or method) observed during a development session. Every time the developer interacts with an entity (*e.g.,* observe, modify) its weight gets updated, according to the defined weighting strategy. The weight of a program entity is persisted even if all the windows that display that entity get closed. This allows the doctor to keep track of the entities that are relevant in the current development session.

Similarly, the **Window Interaction Model** associates a weight to each open window during a development session. The weight gets updated at each interaction with the specific window (*e.g.,* on window focus, minimization, movement), and depends on the active weighting strategy. When a window is closed, its weight is removed from the model.

A **Weighting Strategy** determines how weights are updated. In the original *Autumn Leaves* strategy, every user interaction brings a particular, fixed, weight update. The doctor implements this strategy, and uses the original parameters and weight updates suggested by Röthlisberger *et al.* [4]. However, we will investigate the effectiveness of the original parameters. The original strategy prescribes that 50% of the weight updates of program entities is propagated following structural source code relationships (*i.e.,* method propagates to its defining class, class to its direct superclasses and subclasses). Currently, only one weighting strategy at the time could be active.

A **Closing Strategy** is responsible to determine which windows are candidate for closure. For example, a strategy that involves the weight models can define a threshold (*i.e.,* sensitivity) on the weight of windows. As in the case of *Autumn Leaves*, we implemented a strategy that closes all the windows whose weight is below a customizable threshold. The default approach uses a percentage of the average weight of all the windows. An innovation with respect to *Autumn Leaves* is that the user can activate more than one closing strategy at the time. A strategy could consider the weight models or ignore them, *e.g.,* one might want to have a maximum number of open windows per window type. In an IDE, often, there are different *kinds* of windows: workspaces, code browsers, test runners, *etc.* We also implemented a strategy that closes the windows with lowest weight of a given type, when the IDE reaches the maximum amount of open windows for that type.

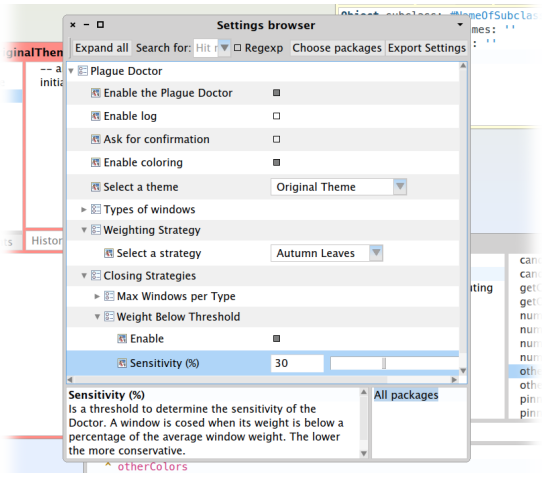

Fig. 3. The settings of the PLAGUE DOCTOR.

B. Advocatus Diaboli

In this section we play the devil's advocate and address some criticisms that can be raised against our approach.

It only works for window-based IDEs. Even though our prototype has been implemented in PHARO, a window-based IDE, the approach is not specific to such environments. The default weighting and closing strategies can be applied directly to tab-based IDEs such as ECLIPSE.

The Plague Doctor does not differ from Autumn Leaves. Our tool is indeed inspired by *Autumn Leaves*, and mimics all its functionalities, but it has a number of advantages:

- ▷ It exploits **more** and **better** interaction data. The doctor can leverage all the fine-grained data recorded by DFLOW. For example, using the code browser to navigate source code might increase the weight of a window and all the visited entities. Debugging events can be leveraged to increase the weight of the entities being investigated.
- ▷ It is **extensible**. The PLAGUE DOCTOR is designed to be extensible. For example, adding a new weighting (or closing) strategy requires minimal effort, *i.e.,* a new class copied from a template and a method that implements the strategy per se. The new strategy will immediately appear in the settings of the PLAGUE DOCTOR (depicted in Figure 3) and can replace the current one right away.
- ▷ It is **customizable**. The PLAGUE DOCTOR has a number of settings, depicted in Figure 3, that the developer can use to customize it. For example, all the colors are contained in a theme class that can be duplicated and changed to have a novel and more appealing color scheme.
- ▷ It is **real**. Differently from *Autumn Leaves*, the PLAGUE DOCTOR is currently available[3] and can be installed in the PHARO IDE. Currently, we are working with the community and gathering qualitative feedback towards the integration of the PLAGUE DOCTOR in the PHARO image.

[3]See http://goo.gl/wUtd7O

III. Long-Term Vision

The prototype of the PLAGUE DOCTOR described in Section II is only the first step towards fully exploiting the data collected by DFLOW while the developer is programming. This section discusses our future plans to provide a more effective cure for the window plague leveraging DFLOW data.

Fine-tuning the existing strategies. Weighting and closing strategies are parametrizable. The value of the weight update after a particular user interaction, for example, is a parameter of the weighting strategy. The usefulness of the PLAGUE DOCTOR strictly depends on how good the strategies are. Until now, we reused the values proposed by the authors of *Autumn Leaves* [4]. The authors used a benchmark evaluation to devise such values. We are in contact with the core developers of the PHARO community, and we plan to conduct a detailed user evaluation with them to fine-tune these parameters backing them up with evidence from interaction data.

Novel Strategies. Our initial PLAGUE DOCTOR prototype makes it easy to add new strategies to weight or close windows in different ways. Our plan is to devise a number of different strategies and test them in real settings scenarios, *i.e.,* involving real developers. A qualitative study where we can get feedback from developers about our approach could also trigger new ideas for novel strategies. Since the PLAGUE DOCTOR allows multiple closing strategies to be enabled, we should also investigate which combinations of strategies perform better.

Self-Adaptation. Our long term vision focuses on the *self-adaptability* of IDEs [7]. We believe that tools should also be subject to self-adaptation. In this context, for example, strategies could be self-adaptable. Consider the closing strategy that uses a threshold to decide which windows to close. Suppose that, when the developer realizes that the PLAGUE DOCTOR wants to close a window, she *pins* that window to force the doctor to leave it open. The doctor must lose confidence in itself and relax its sensitivity. In the opposite case, if the doctor closes windows without the user playing against, it should slightly increase its confidence and increase its sensitivity.

Exploiting more interactions. The PLAGUE DOCTOR currently exploits only a few more user interactions with respect to *Autumn Leaves*. Potentially, it can leverage all the fine-grained interactions collected by DFLOW [5]. Mouse events, for example, can be factored in the weighting strategies. In fact, developers might use the mouse as a *reading device*, *i.e.,* by following the source code that they are reading with the mouse cursor. Another source of information collected by DFLOW are debugging events. If a developer spends time in debugging a piece of code, it is likely that the program entities contained in this code snippet are relevant for the current development session, thus their weights should increase.

Evaluation Plan. With DFLOW we recorded more than 1,000 development sessions from more than 20 developers. To validate the PLAGUE DOCTOR, we plan to do a benchmark evaluation and feed the recorded sessions to the tool. The idea is to define the level of entropy of the IDE (*i.e.,* how many unused windows are left open) and measure if and how it varies with the support of the PLAGUE DOCTOR. We also aim to obtain further evidence of the importance of the window plague in practice. Our expectation is that the tool is effective to reduce the level of entropy while being as precise as possible. By precise we mean that the PLAGUE DOCTOR should only close windows that the developer would not reuse in the future. There is a trade-off between precision and effectiveness that remains to be investigated and optimized. Another study could focus on the time spent by developers in program understanding tasks. From a previous analysis of recorded interaction data, we found that developers spent a considerable amount of time (ca. 15%) in fiddling with the UI of the IDE (*e.g.,* by rearranging windows that create confusion in the IDE). Since the window plague is one possible reason behind this, we should investigate if approaches such as *Autumn Leaves* or the PLAGUE DOCTOR reduce the time wasted by developers in fiddling with the UI of the IDE. Last but not least, we also plan to conduct a qualitative evaluation to gather direct feedback from developers.

IV. Conclusions

Developers construct and evolve software systems using IDEs, which offer little support to navigate the complex and implicit relationships among program entities. IDEs force developers to open one window (or tab) per program entity, leading to what researchers called "*window plague*", an overly crowded workspaces with many open windows (or tabs).

To mitigate this plague, we implemented the PLAGUE DOCTOR, a tool that leverages fine-grained interaction data collected by DFLOW. It computes the importance of windows, decorates them to reduce the cognitive load of a developer facing the IDE, and closes the windows that are less likely to be used again in the future. We also discussed our future plans towards a more effective cure for the window plague.

Acknowledgements

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

References

[1] A. Dunsmore, M. Roper, and M. Wood, "Object-oriented inspection in the face of delocalisation," in *Proceedings of ICSE 2000 (22^{nd} International Conference on Software Engineering)*, 2000, pp. 467–476.

[2] N. Wilde and R. Huitt, "Maintenance support for object-oriented programs," *IEEE Transactions on Software Engineering*, vol. 18, no. 12, pp. 1038–1044, 1992.

[3] J. Singer, R. Elves, and M.-A. Storey, "Navtracks: Supporting navigation in software," in *Proceedings of IWPC 2005 (13^{th} International Workshop on Program Comprehension)*, 2005, pp. 173–175.

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

[5] R. Minelli, A. Mocci, M. Lanza, and T. Kobayashi, "Quantifying program comprehension with interaction data," in *Proceedings of QSIC 2014 (14^{th} International Conference on Quality Software)*, 2014, pp. 276–285.

[6] R. Minelli, A. Mocci, M. Lanza, and L. Baracchi, "Visualizing developer interactions," in *Proceedings of VISSOFT 2014 (2^{nd} IEEE Working Conference on Software Visualization)*, 2014, pp. 147–156.

[7] R. Minelli, "Towards Self-Adaptive IDEs," in *Proceedings of ICSME 2014 (30^{th} International Conference on Software Maintenance and Evolution), Doctoral Symposium*, 2014, p. 666.