Staying Aware of Relevant Feeds in Context

Thomas Fritz
Department of Computer Science
University of British Columbia
fritz@cs.ubc.ca

ABSTRACT

To stay aware of relevant information and avoid productivity loss, a developer has to continuously read through new incoming information. Our approach supports the integration of dynamic and static information in a development environment that allows the developer to continuously monitor the relevant information in context of his work.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments

General Terms

Human Factors, Design

Keywords

Feeds, Human-centric software engineering

1. PROBLEM AND MOTIVATION

When building a software system, a developer has to stay aware of a lot of information, such as changes to the project's code, the work of his coworkers, new bugs or changes to the API of used libraries. Missing relevant information may result in extra work and time that has to be spent. For example, Damian and colleagues report on a case, in which a developer missed an important email due to the information overload he experienced, resulting in a broken build and productivity loss for the whole team [2].

A common solution to stay aware of relevant information is by subscribing to feeds. As a developer's feed subscriptions accumulate over time, he gets flooded with information, a lot of which is not relevant to his work. To find out if a feed is relevant to a developer's work, the developer has to read through the feed and find the relation to his work, often by following keywords or links in the feed.

Some existing approaches ease this problem for certain kinds of feeds. Palantir [8, 7], for example, makes developers aware if any of the source code in a developer's workspace is currently changed by someone else. These approaches provide some fixed context for the new information; in the case of Palantir, the context is the source code loaded in the developer's workspace. However, they provide little to no flexibility for the developer to choose the context.

In previous work [4], we have shown how one can compose different kinds of information, such as source code, bugs, teams and change sets, to answer a developer's question. In this work, we extend the underlying model of our previous research with the concept of feeds to find relevant information a developer should stay aware of. Figure 1(a) shows a typical non-contextual list-based feed reader. A developer must scroll and read each feed item to understand what might be relevant. Our approach allows a developer to compose the feeds with other information from a developer's workspace, providing a context for interpreting the feeds and supporting various means of ranking and filtering the feeds. Figure 1(b) displays our Fragment Explorer view in which a developer has composed the source code of his workspace with feeds, in this case web feeds and feeds on change set events. The kinds of information being composed, i.e., source code, web feeds and change set feeds, and the order of composition is represented by the icons in the top right corner of the view. The developer can also choose the time frame for what he considers relevant information by using the slider in the awareness picker on the bottom of the view. Now, as soon as, for example, a new change set event comes in from a change set feed, it is automatically put in context of the source code of the developer's workspace and in case the change set affects the source code, it is presented in the view. An example of this is shown in Figure 1(b) in which Mike delivered a change set that affects class CountSelectionAction.

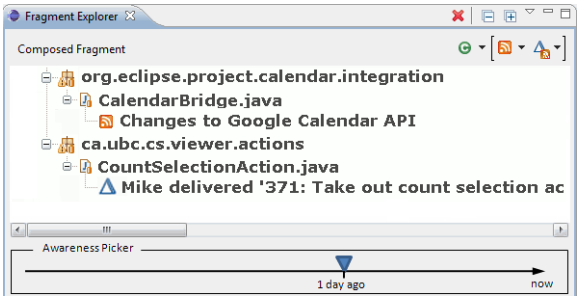
2. BACKGROUND AND RELATED WORK

"Awareness is an understanding of the activities of others, which provides a context for your own activity." [3]. As described in Section 1, several approaches (e.g., Palantir [8, 7] and an awareness environment built at IBM [1]) provide awareness for limited kinds of information in limited contexts.

Other approaches have tried to recommend information that is relevant to the current piece of information a developer is working with. For instance, Hipikat [9] establishes links between artifacts based on a fixed schema and then uses these links to infer which artifacts might be relevant based on a developer's query. Deep Intellisense [6] presents

ICSE '10, May 2-8 2010, Cape Town, South Africa
.

(a) All Feeds



(b) Relevant Feeds in Context

Figure 1: Finding Relevant Feeds

information related to the source code currently selected, such as the bugs it is involved in. Neither of these two approaches focuses on providing awareness to a developer, and both are based on a fixed schema.

In other previous work [5], we have introduced a degree-of-knowledge model to capture a developer's familiarity with source code and applied the model to find bugs that could be relevant to a developer. Our previous work does not attempt to provide awareness to a developer.

3. APPROACH AND EVALUATION

Our approach is based on an information fragment model that supports the composition of different kinds of information and its presentation. Previously, information fragments were defined as a static set of items of information, such as bugs or classes and methods, to answer a specific question. Feeds are a dynamic concept and can not be captured by our original *static information fragments*. As a feed is a stream of information, it constantly changes with new incoming information items. Furthermore, a feed has a certain lifetime aspect—information items that are too old are not relevant anymore and discarded. To incorporate feeds into our model, we expand it with the notion of a *dynamic information fragment*, which is continuously evaluated.

In practice, a developer forms a dynamic information fragment by selecting feeds of interest, such as feeds on change sets or web feeds. He can then compose these information fragments with other information fragments, such as a fragment comprising the source code of certain packages. Figure 1(b) shows the result of such a composition. Previously, the composition was done once and then displayed in our view. With a dynamic information fragment, the composition is done continuously so that every time a feed changes, the view is updated accordingly. To reflect the lifetime aspect of feeds, the view also contains a slider, called the awareness picker, to pick the relative start time. In our example in Figure 1(b), the start time is one day ago. Every item older than that will be discarded. For now, the awareness picker applies to all dynamic fragments in the view, but it might be necessary to specify it separately. Finally, our new approach also provides the option to stack information fragments as can be seen with the two feeds fragments in Figure 1(b). By stacking fragments, they are presented on the same hierarchical level in the tree viewer.

To evaluate our approach and its flexibility, we apply our approach to three different scenarios. First, we show that our approach can be used to stay aware of relevant changes to the source code by applying it to feeds on source code changes similar to most existent approaches. Second, we apply our approach to web feeds on changes to APIs and evaluate the false positive and false negative rate. Finally, we use our approach to find feeds that talk about bugs relevant to a developer's work. In the last scenario, we determine relevancy in terms of bugs that were later on changed again by the developer and discuss the false positive and false negative rate.

4. RESULTS AND CONTRIBUTIONS

The result of our work is a fluid integration of dynamic and static development information in a developer's integrated development environment. This improves how different kinds of information can be integrated. In particular, it allows a developer to stay aware of relevant information from feeds not limited to a particular kind of information and have them presented in his work context.

5. REFERENCES

[1] L.-T. Cheng, S. Hupfer, S. Ross, and J. Patterson. Jazzing up eclipse with collaborative tools. In *eTX'03*.

[2] D. Damian, L. Izquierdo, J. Singer, and I. Kwan. Awareness in the wild: Why communication breakdowns occur. In *ICGSE'07*.

[3] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *CSCW'92*.

[4] T. Fritz and G. C. Murphy. Using information fragments to answer the questions developers ask. In *ICSE'10*. to appear.

[5] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill. A degree-of-knowledge model to capture source code familiarity. In *ICSE'10*. to appear.

[6] R. Holmes and A. Begel. Deep intellisense: a tool for rehydrating evaporated information. In *MSR'08*.

[7] A. Sarma, G. Bortis, and A. van der Hoek. Towards supporting awareness of indirect conflicts across software configuration management workspaces. In *ASE'07*.

[8] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantír: raising awareness among configuration management workspaces. In *ICSE'03*.

[9] D. Čubranić, G. C. Murphy, J. Singer, and K. S. Booth. Hipikat: A project memory for software development. *IEEE TSE*.