CONCERNLINES: A Timeline View of co-occurring Concerns

Christoph Treude, Margaret-Anne Storey
Dept. of Computer Science, University of Victoria
ctreude@uvic.ca, mstorey@uvic.ca

Abstract

Understanding the evolution of a software system requires understanding how information about the release history, non-functional requirements and project milestones relates to functional requirements on the software components. This short paper describes a new tool, called CONCERNLINES*, that supports this cognitive process by visualizing co-occurring concerns over time.*

1. Introduction and Motivation

Software development processes are among the most complicated tasks performed by humans. Understanding their evolution is a prerequisite for efficient management, process improvements, and the integration of members into development teams. Gaining high level evolutionary information about large software systems is a key challenge in dealing with increasing complexity and decreasing software quality [5]. While several tools for evolutionary information at the source code level have been proposed, research on the visualization of process related information over time is limited. However, process understanding has been identified as an important aspect of software maintenance [19].

One way to understand and explore software evolution is through concerns. According to IEEE, concerns *"are those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders"* [7]. There are different dimensions of potentially overlapping concerns [11], ranging from process-related issues such as milestone releases to requirements such as usability. Determining relevant concerns as the system evolves is not always easy.

In our previous work [17], we discovered that developers document relevant concerns using *tags* on change requests[1]. Tags are user-defined keywords that in this case document and relate process-related, cross-cutting, and component-specific concerns to the change requests. Our goal during this study was to explore how developers tag change requests. To facilitate our exploration of how developers tag using concerns, we created the CONCERNLINES tool. We used the tool to create timelines of how concerns emerged and co-occurred over time. These timelines were shown during interviews with developers to elicit more in depth insights on how tags were used for describing concerns during system development.

The feedback we received from the developers indicated that a tool for viewing information on co-occurring concerns can be very useful. For example, developers wanted to answer questions such as: Which components played a key role during the last beta release? Which non-functional requirements co-occurred with work on the user interface? Which components were affected by the improvements on user experience? Their reaction to the tool and their suggestions for enhancements prompted us to further extend it so that developers could also use it during development to understand the emergence of co-occurring concerns over time.

CONCERNLINES visualizes the evolution and relevance of concerns over the lifetime of a software project and enables the identification of co-occurring concerns. It displays the evolution in a snapshot, using horizontal timelines for individual concerns and mapping the relevance of concerns (e.g. determined through frequency of tag occurrences) onto colour intensities. Co-occurrences can be identified by pivoting on one concern in a selected time range.

Although it may seem challenging to determine which concerns are relevant at given points in a system's evolution, there are various data sources that can be mined to determine concerns, e.g. source code, source code comments, JavaDoc, CVS commits, change requests or email communication. Even though CONCERNLINES uses a generic interface to allow the visualization of concerns extracted from any of these various sources, in this paper we focus on the visualization of concerns derived from change request tags.

The remainder of this paper is structured as follows. Related work is summarized in Section 2 and the tool is described in detail in Section 3. We then present example scenarios in Section 4 and conclude in Section 5.

[1] Depending on the development environment, change requests are also known as bug reports, modification requests, or work items.

ICSE'09, May 16–24, 2009, Vancouver, Canada
978-1-4244-3452-7/09/$25.00 © 2009 IEEE

2. Background and Related Work

Work related to our research can be divided into literature on concerns in software development and literature on visualizations of software evolution. Our work can be seen as the intersection of these areas as it provides a visualization of the evolution of concerns.

The term *concern* is often used synonymously with the term *aspect* in the context of aspect-oriented programming (AOP) [8]. In contrast to that, the definition used here is broader in the sense that it goes beyond source code. Aside from AOP, important tools for concerns in software development include *Concern Graphs* [12] and *ConcernMapper* [13]. They enable developers to associate parts of source code with high level concerns. Opposite to this top-down approach, the tool *TagSEA* [15] takes a bottom-up approach of relating source code to concerns by allowing developers to tag their source code. Mapping of concerns is also supported by *Hyper/J* [11], a tool that focuses on the separation of concerns in multiple dimensions and thus acknowledges the existence of different kinds of concerns.

Literature on visualizations of software evolution can be categorized by the data that is visualized. The main categories are source code, commit events, metrics, and data derived from change requests. On a fine-grained level, Voinea *et al.* look at the evolution of source code on a line-by-line basis with their tools *CVSscan* [19] and *CVSgrab* [18].

A tool for the visualization of commit events was proposed by Telea and Voinea [16]. Their *Project evolution view* maps different files to the y-axis and time to the x-axis. The matrix is then filled using different colours for commit events. Using a visualization like this, clusters of files that are changed together can be identified. Another timeline view of source code files focusing on cluster recognition was proposed by van Rysselberghe and Demeyer [14].

A first step to aggregate information at a higher level of detail and displaying its evolution was done using metrics. The *Evolution Matrix* by Lanza and Ducasse combines metrics with evolution visualization [9] and the *Evolution Spectrographs* by Wu *et al.* provide a metric-based representation of the development history of a software system [20].

Research on the visualization of data derived from change requests is still limited. The *Discrete Time Figure* by D'Ambros and Lanza [3] is a visualization technique in which historical and structural data are embedded into one figure. Software entities are shown as horizontal timelines, with the number of commits regarding these entities and the number of bugs reported mapped onto the timelines. Also leveraging the information available from change requests are *Deep Intellisense* [6] and a tool proposed by Fischer and Gall [4]. They focus on integrating historical information from various sources and the evolution of dependencies between features respectively.

In addition to the static tools discussed above, animations that use video or interactive tooling to display evolution have been proposed. Prominent examples include *Evolution Storyboards* [1] for the visualization of a series of software graphs and *Gevol* [2] based on CVS data.

3. CONCERNLINES

Our tool, CONCERNLINES, aims at visualizing concerns in software evolution over time and at allowing the identification of temporal co-occurrences of concerns. Like most of the tools discussed in Section 2, it utilizes a timeline view to represent time. Before explaining the details of our implementation, we define the visualization using the five dimensions proposed by Maletic *et al.* [10]: task, audience, target, presentation, and medium. The main **task** supported by CONCERNLINES is the exploration of characteristics and interrelations of release history, non-functional requirements, and software components based on concerns. The intended **audience** reaches from management for high level insights to new team members for familiarization with a given project. The data source represented, i.e. the **target**, is a mapping of concerns extracted from software artifacts to time and the **representation** is a timeline-oriented view of concerns over time using a regular display as the **medium**.

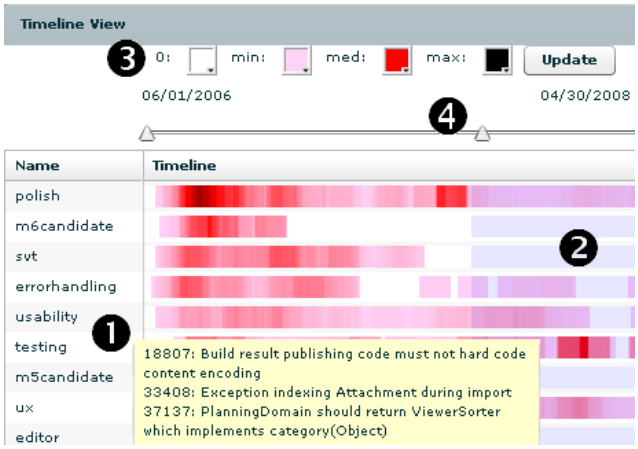

Figure 1. CONCERNLINES **user interface**

Interface. Figure 1 shows the user interface of CONCERNLINES. The main components are the timelines (2) for the concerns listed on the left hand side (1). In the current implementation, this part is enhanced by a tooltip with relevant change requests. The interactive part of the tool consists of support for customizing the colour scheme (3) and a time range selector (4). Unselected time ranges are greyed out in the timeline part (2). CONCERNLINES was

developed using Adobe's Flex[2] and is accessible through a web browser. The tool uses a CSV file with a matrix of concerns and their intensity over time as input data. Such a matrix can be computed by mining repositories of software artifacts such as change requests.

Timelines. A timeline represents each concern (for the examples in this paper, tags on change requests are used as concerns). The timelines are rendered from left to right according to time. As observed by D'Ambros and Lanza [3], timeline visualizations do not necessarily scale well. To ensure the scalability of the visualization up to long periods of time, the width of each time unit is calculated dynamically, i.e. a longer time range results in a narrower display of time units. In addition, the scalability is improved by a default setting of using 30 day averages instead of the actual values in the display. Thus, two time units next to each other are less likely to have values that differ significantly.

Colour intensity is used to represent the relevance of a concern at a particular point in time. This relevance is given by the number of occurrences of a concern on a particular day. Colours are user configurable. With the default settings, white is used to represent that the concern is not relevant at a given point in time and stronger colours represent higher incidence. The suggestion to distinguish more relevant concerns was made by one of the developers viewing the first iteration of our tool.

Time Range Selection. Using a horizontal slider with two end controls, the time range can be narrowed down to a range of interest. Unselected parts are greyed out in the visualization. The time range selection is used for detecting co-occurrences of concerns in the specified time frame.

Detection of Co-occurrences. The detection of co-occurrences is done by pivoting on one concern in the selected time range. The feature of displaying co-occurring concerns was also suggested by one of the developers viewing the earliest version of our tool: *"Api or ui or ux, [...] you might see these line up with these, with the breaks."* Clicking on either the concern name or its timeline, this concern is moved to the top of the list. All other concerns are arranged below it, ordered by the similarity of their timeline to the timeline of the pivot concern in the selected time range. The similarity is calculated by iterating over the days in the selected time frame and summing up the squared differences between the corresponding values, i.e. a difference d between two timelines a and b is defined as $d = \sum_{i=m}^{n}(|a_i - b_i|)^2$ for a time frame from m to n.

[2]http://www.adobe.com/products/flex/.

4. Example Scenarios

This section gives two exemplary scenarios for situations in which we expect CONCERNLINES to be useful. The example data stems from our previous study with a large development team from IBM [17].

Non-functional Requirements during Milestone 6. To get a high level overview of what milestone 6 was about, a project manager can use CONCERNLINES and select *m6candidate* as the pivot concern. All other concerns will be ordered by their timeline-similarity to the pivot concern, as shown in Figure 2. *M5candidate* and *m6candidate* have a big overlap, and *globalization* was the main concern during those milestone releases. This view reveals the major theme without requiring the project manager to look at individual change requests.

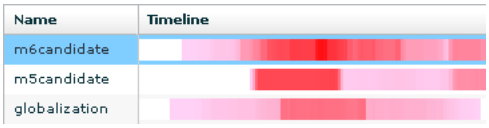

Figure 2. Concerns during Milestone

Concerns regarding UI. To understand the importance of different cross-cutting concerns such as usability or performance regarding the user interface, a software developer can choose the corresponding concern as the pivot element. Figure 3 shows the result. In addition, the time range has been narrowed down to a time frame in which the concern *ui* has high intensity. Several conclusions can be drawn from the visualization: the concern with the highest similarity to *ui* is *svt*, i.e. testing. *Ui* also has a high correlation with *usability* and mainly co-occurs with milestone 6. The developer can conclude that the concern of usability is highly related to work on the user interface and may put a stronger focus on usability during design decisions.

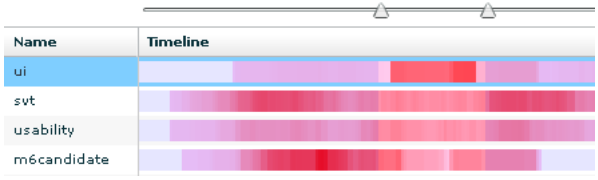

Figure 3. Concerns regarding UI

5. Conclusions and Future Work

With CONCERNLINES, we propose a tool that visualizes the emergence and co-occurrence of concerns over time.

Horizontal timelines are used to represent concerns and the intensities of concerns over time are mapped onto colour. Pivoting on one concern in a specified time frame allows the user to arrange concerns by their timeline-similarity to a pivot concern. This reduces the complexity of identifying co-occurring concerns. The preliminary feedback indicates that CONCERNLINES is a promising approach for understanding the relationship between multiple concerns.

Concerns can be derived from multiple data sources and it will depend on the situation at hand which attributes or characteristics should be interpreted as concerns. Potential sources for concerns include keywords from change requests, JavaDoc and other annotations from source code, and themes extracted from e-mail communication. The feedback we received suggests that the example used in this paper, tags for change requests, is a valuable choice and allows useful insights into the evolution of a software project. Exploring further data sources is part of our future work.

Apart from that, we are looking at evaluating CONCERNLINES by deploying it in an industry setting and logging user interactions. Also, we are exploring how the tool can be used as a way for navigating artifacts based on concerns.

6. Acknowledgements

We wish to thank the team that granted us access to their repositories, conducted interviews with us and gave us tool ideas. This research is supported by a fellowship from IBM.

References

[1] D. Beyer and A. E. Hassan. Animated visualization of software history using evolution storyboards. In *WCRE '06: Proc. of the 13th Working Conf. on Reverse Engineering*, pages 199–210, Washington, DC, 2006. IEEE Computer Society.

[2] C. Collberg, S. Kobourov, J. Nagra, J. Pitts, and K. Wampler. A system for graph-based visualization of the evolution of software. In *SoftVis '03: Proc. of the 2003 ACM symposium on Software visualization*, pages 77–ff, New York, 2003.

[3] M. D'Ambros and M. Lanza. Software bugs and evolution: A visual approach to uncover their relationship. In *CSMR '06: Proc. of the Conf. on Software Maintenance and Reengineering*, pages 229–238, Washington, DC, 2006. IEEE Computer Society.

[4] M. Fischer and H. Gall. Visualizing feature evolution of large-scale software based on problem and modification report data: Research articles. *J. Softw. Maint. Evol.*, 16(6):385–403, 2004.

[5] H. C. Gall and M. Lanza. Software evolution: analysis and visualization. In *ICSE '06: Proc. of the 28th Intl. Conf. on Software engineering*, pages 1055–1056, New York, 2006. ACM.

[6] R. Holmes and A. Begel. Deep intellisense: a tool for rehydrating evaporated information. In *MSR '08: Proc. of the 2008 Intl. working Conf. on Mining software repositories*, pages 23–26, New York, 2008. ACM.

[7] IEEE. Recommended practice for architectural description of software-intensive systems. *IEEE Std 1471-2000*, pages i–23, 2000.

[8] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. *Aspect-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[9] M. Lanza and S. Ducasse. Understanding software evolution using a combination of software visualization and software metrics. In *Proc. of LMO 2002*, pages 135–149, 2002.

[10] J. I. Maletic, A. Marcus, and M. L. Collard. A task oriented view of software visualization. In *VISSOFT '02: Proc. of the 1st Intl. Workshop on Visualizing Software for Understanding and Analysis*, page 32, Washington, DC, 2002. IEEE Computer Society.

[11] H. Ossher and P. Tarr. Hyper/j: multi-dimensional separation of concerns for java. In *ICSE '00: Proc. of the 22nd Intl. Conf. on Software engineering*, pages 734–737, New York, 2000. ACM.

[12] M. P. Robillard and G. C. Murphy. Concern graphs: finding and describing concerns using structural program dependencies. In *ICSE '02: Proc. of the 24th Intl. Conf. on Software Engineering*, pages 406–416, New York, 2002. ACM.

[13] M. P. Robillard and F. Weigand-Warr. Concernmapper: simple view-based separation of scattered concerns. In *eclipse '05: Proc. of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 65–69, New York, 2005. ACM.

[14] F. V. Rysselberghe and S. Demeyer. Studying software evolution information by visualizing the change history. In *ICSM '04: Proc. of the 20th IEEE Intl. Conf. on Software Maintenance*, pages 328–337, Washington, DC, 2004.

[15] M.-A. Storey, L.-T. Cheng, I. Bull, and P. Rigby. Shared waypoints and social tagging to support collaboration in software development. In *CSCW '06: Proc. of the 2006 20th anniversary Conf. on Computer supported cooperative work*, pages 195–198, New York, 2006. ACM.

[16] A. Telea and L. Voinea. Interactive visual mechanisms for exploring source code evolution. In *VISSOFT '05: Proc. of the 3rd IEEE Intl. Workshop on Visualizing Software for Understanding and Analysis*, page 17, Washington, DC, 2005.

[17] C. Treude and M.-A. Storey. How tagging helps bridge the gap between social and technical aspects in software development. In *ICSE '09: Proc. of the 31st Intl. Conf. on Software engineering*, 2009. To appear.

[18] L. Voinea and A. Telea. Mining software repositories with cvsgrab. In *MSR '06: Proc. of the 2006 Intl. workshop on Mining software repositories*, pages 167–168, New York, 2006. ACM.

[19] L. Voinea, A. Telea, and J. J. van Wijk. Cvsscan: visualization of code evolution. In *SoftVis '05: Proc. of the 2005 ACM symposium on Software visualization*, pages 47–56, New York, 2005.

[20] J. Wu, C. W. Spitzer, A. E. Hassan, and R. C. Holt. Evolution spectrographs: Visualizing punctuated change in software evolution. In *IWPSE '04: Proc. of the 7th Intl. Workshop on Principles of Software Evolution*, pages 57–66, Washington, DC, 2004. IEEE Computer Society.