Interactive User-Oriented Views for Better Understanding Software Systems

Truong Ho-Quang, Michel R.V. Chaudron
Department of Computer Science and Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden
{truongh, chaudron}@chalmers.se

Abstract

Understanding software artefacts is a crucial task for people who want to participate in any software development process. However, because of the large amount of detailed and scattered information in software artefacts, understanding them is usually time-consuming and vulnerable to human errors and subjectivities. A system that aids practitioners to investigate understanding about software artefacts could reduce the vulnerabilities and speed up software development/maintenance process. Our research focuses on building a comprehensive view of software system in order for developers to achieve the two goals: (i) to save the time spending on searching and navigating on source code; and (ii) to gain better understanding about software artefacts regarding to domain-specific tasks. To achieve these goals, we propose an emprical approach in which the visualisation and the generation of high-level design and architectural views from source code and design documentations have been played central roles. The research is on-going and could potentially be extended to different software artefacts (such as requirements, use-cases, test-cases, revision logs).

1 Introduction

Software artefacts are created, maintained and evolved as part of a software development project. Understanding software artefacts is a crucial task for every person who wants to participate in any phase of software development life cycle [26]. However, because of the large amount of detailed and scattered information in software artefacts, understanding them is usually very time-consuming and vulnerable to human errors and subjectivities [9][20]. The task becomes even more difficult when it comes to large-size software projects, which contain a huge amount of code, designs and documentation.

Recently, a significant number of research and tools has been conducted in order to investigate better understanding of software artefacts. It can be listed as reverse engineering [3], feature location [8], document summarization, etc. However, there is lack of automatic approaches were proposed. With regards to those that can automatically perform the task, the accuracy is not high [12]. Soh et.al, with an observation on 2408

Proceedings of SATToSE, Mons, Belgium, 6-7-2015, published at http://ceur-ws.org

developers interaction logs, has pointed out that 62% of files explored during the implementation of a task are not significantly relevant to the final implementation of the task [20].

In addition, the approaches are usually applied on a single software artefact (e.g. source code, revision history), resulting in a single view(navigation or search results). This fact, at some points, limits developers ability to obtain the overview of the whole (OR a part of) system, which is an essential part of understanding, with regards to the development/maintenance task. Thus, creating of a comprehensible view which can automatically navigate and generate suitable views on different software artefacts would be very beneficial.

To this end, our research has been focusing on the visualisation and the generation of high-level design and architectural views from source code and design documentations. The research could potentially be extended to different software artefacts (such as requirements, use-cases, test-cases, revision logs [6]).

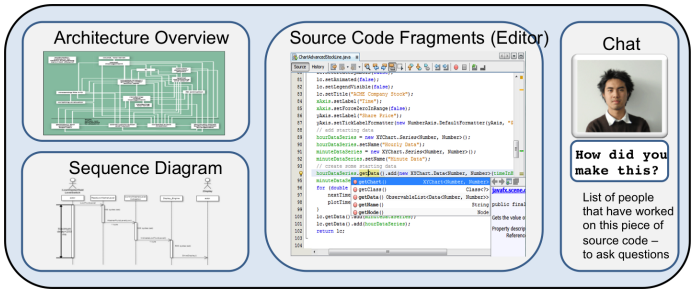

Figure 1: A comprehensible view for task "Maintain a feature"

Figure 1 shows a prototype design of such the view. Given a maintenance task, the sub-views show the task-related parts on different software artefacts. Sub-view Architecture Overview shows an overview of the system with highlights on the task-related components. Sub-view Editor locates to the relevant part of source code. Sub-view Chat shows the list of the responsible developers and the historical chat regarding the observed source code. Sub-views are linkable between themselves and automatically or manually updated. We take developers as the main audience of our research. The following example reveals how the developers perform the understanding task using the view. We consider it as the motivation in our research.

Motivation example. Developer ***X*** has to conduct a task: Maintain Feature A. Software artefacts are stored in projects database. ***X*** starts by logging into his work space (e.g. an IDE), then performs searching for the Feature A by key words. Sub-view Editor will automatically address the related parts of source code which could potentially be changed during the maintenance work. Sub-views Architecture Overview and Sequence Diagram will provide the developer with an overview about the code structure and probably a suggestion about which parts could be subsequently changed. The developer can ask for recommendations from responsible persons through Chat space. Using such the view could allow the developer to better investigate understanding about the task and the system, and to reduce the implementation and maintenance time.

The rest of this paper is organized as follows. In Section 2, we dicuss the problem definition and formulate two main research questions. Section 3 presents the outlines of our approach.

2 Problem Statement and Research Questions

2.1 Problem Statement

Lack of empirical research on developers cognitive task during software maintenance phase. Despite the large body of work on software maintenance [26][20][2], there are very few studies that empirically investigated how developers achieve the understanding of software artefacts during software development/maintenance activities. Ko et.al [13][14] has revealed a number of issues that cause developers more time on navigation between source files. The authors have suggested ideas for tools that help developers seek, relate, and collect information in a more effective and explicit manner. On the other hand, it seems that there is a huge gap between state-of-the-art research and practice in software comprehension. An observational study by Roehm et.al shows that no one in the 28 professional developers (from seven software companies) observes any use of state-of-the-art comprehensive tools [18].

Hard to collect relevant task-based information effectively and automatically. For most tasks, developers begin by searching then navigating by search results. However, traditional searching methods seem

not very effective. A. J. Ko et.al have revealed that an average of 88 percent (±11) of developers searches led to nothing of later use in the task. Those failed searches were at least partially responsible for approximately 36 percent of their time spent on inspecting irrelevant code [14]. Recently, a number of task-specific searching methods (such as features location, program slicing, UML slicing, etc.) has been introduced. However, they are not easy to apply and sensitive to inputs quality [8]. Thus, developing an easy-to-use solution could improve searching and navigating efficiencies.

Lack of visualisation of relevant information in understandable manners. Apart from searching and navigation, visualisation of software artefacts is widely used in the areas of software maintenance, reverse engineering, and re-engineering, where typically large amounts of complex data need to be understood and a high degree of interaction between software engineers and automatic analyses is required. Over the past few years, software visualisation has greatly evolved. However, despite the fact that software visualisation tools have a great potential, when it comes to contextual information, finding a suitable solution is not an easy work [21][10] (e.g. UML design layouting, personalised view, etc.).

2.2 Research Questions

Our research focuses on the visualisation and the generation of high-level design and architectural views from source code and design documentations. In order to come up with a systematic answer for the question, it's necessary to find out what is practitioner's mind when performing the understanding task. Thus, we would think about the following research questions:

RQ1. *What are practitioners needs in order to understand a part of system with regards to a specific task?*

RQ2. *How to generate and present the information by an effective way?*

3 Research Approaches

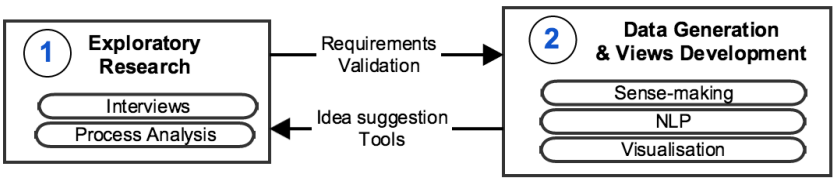

Figure 2: Research Activities

In order to investigate the two research questions, we conduct two research activities as shown in the Figure 2. In Research Activity 1, we use both qualitative and quantitative approaches to learn practitioners needs and strategies during the understanding phase. In particular, by conducting interviews with industrial and academic practitioners, we could achieve better understanding on what is their cognitive thinking and possibly the strategy that was used to understand the system. By logging practitioners activities and analysing the logging file, we could statistically investigate their unconscious behaviors and the difficulties performing the understanding task. This approach is discussed in detail in subsection A.

Research Activity 2 aims at answering RQ2 with a focus on generating high-level abstraction of design and architectural views from source code. We have been studying sense-making and software architectural visualisation. The approaches are discussed in subsection B.

Activity 1 and Activity 2 are concurrently performed. On one hand, outcomes of Activity 1 can be considered as requirements for Activity 2. On the other hand, research ideas and the views that are generated from Activity 2 will be introduced to pratitioners. Validation will be made during the iterations of the two activities.

3.1 An exploratory study of practitioners

3.1.1 Conduct interviews

Developers are often not up-to-date with state-of-the-art comprehension tools. On the other hand, academia has a limited knowledge about industrial practitioners. For example, when it comes to questions like: *How could we*

understand a (part of a) software system? Referring to software design seems to be an obvious answer. However, none of the observed research has mentioned the use of architecture design as part of the understanding process.

Thefore, *semi-structured interviews* will be used to get a better understanding of software practitioners. We consider both academic and industrial developers as targeted interviewees. We split them into groups by several ways: level of software comprehension expertise; familiar with a specific software system/software maintenance task. Shedding some lights in the differences between groups in understanding software systems could be beneficial for us in order to generate suitable views for each group. We take our colleagues and Software Engineering students at the University of Gothenburg and Chalmers University of Technology as academic candidates. We have been inviting a number of local companies (such as Volvo Cars, Ericsson) and out-of-border companies (which locate in Vietnam, The Netherland) to involve in this research.

3.1.2 Process Mining

In the quest for knowledge about strategies and struggles that practitioners have found during the understanding phase, process mining is possible research tool. Process mining techniques make use of historical data to graphically represent and analyse a particular process [23].

Blikstein reported of the use of a logging module for programming tasks [1] and identified student strategies that could help lecturers identify student problems in an early stage of the task. Ko et al. conducted a study in which they used the combination of a logging file and a visual interpretation tool to analyse the behaviour of software developers during a maintenance task [14]. They successfully identified different strategies the developers used. Claes, Pingerra et.al [4][17] logged students events during business process modeling sessions. They used visual analysis [24] and found different styles and related them to model quality.

By using a process mining approach, we have conducted an exploratory study on students strategies performing software modeling tasks [5]. We found out that students use different strategies for solving the tasks. We categorised these strategies into four main strategies: Depthless, Depth First, Breadth First and Ad-Hoc. From our results Depth First indicates to support better layout and richness (detail). We wanted to examine our insights by conducting this experiment on a bigger sample size of students, and possibly on industrial side.

3.2 Data generation and development of the views

3.2.1 Sense-making on source code

Sensemaking, as described by Weick [7], literally means making sense of events. According A. von Mayrhauser, sensemaking is a term used to refer to humans capability to actively comprehend the significance of ambiguous events and data [25]. To our point of view, sense-making is considered as a process where software artefacts are manipulated and presented in a higher level of abstraction. With the focus on high-level design and architectural concepts from source code, we take software reverse engineering (RE) and natural language processing (NLP) as the main drivers for the sense-making process.

Reverse Engineering. Reverse engineering aims to analyse the source code of a system and create design representations of the system [3]. Open source and commercial tools have been developed to generate software design from source code. However, the reverse engineered presentation often contains too much details. When a RE class diagram becomes too large, it provides little benefit towards program. We have been working on possible solutions to present the RE diagrams in a more informative way.

We take the prior research done by Osman et.al as inspiration. The authors have proposed a supervised machine learning approach to condense RE class diagrams into another class diagram that is close to forward design diagram [15][16]. The authors compute values of a number of design metrics from source code and use those to predict classes as important or not. The condensed diagram is then constructed from the reverse-engineered diagram by keeping the important classes and eliminating unimportant ones. Thung et.al have extended Osmans work by using networks metrics as predictors [22]. This work could be extended by considering more predictor features, i.e. dynamic metrics (from execution traces), text mining metrics (from use cases, requirements).

Natural Language Processing. NLP can be considered as a process of extracting information from human or natural language inputs. By adapting NLP to source code analysis, one could be able to extract semantically-related parts of source code, which in the end results in the reducing of maintenance cost. Shepherd et.al has introduced a Find-Concept search process which makes use of NLP analysis that captures the relations between actions (verbs) and the objects (nouns) that these actions act upon [19]. Hill et.al propose a technique and a tool to score method relevance with respect to natural language descriptions of a specific maintenance task

[11]. Starting with a seed method and a natural language description of the bug to be fixed or feature to be added, the tool automatically generate and show a reduced version of the call-graph of the method by pruning irrelevant structure edges from consideration. We have been working on automatic recognition of classs roles (such as UI, security, persistence, etc.) by using text-analysis on source code. The result could then be displayed in a role-based view of reverse engineering design.

3.2.2 Data visualisation

Visualisation depends on target audience and its information needs which are not available at the first phases of the research. Therefore, visualisation is not our main focus at current time. So far, we have been working on the two main directions: 1) With regards to the visualisation of RE diagrams: We are considering different visualisation strategies for class's roles; 2) Regarding presentation of activity logging data: We have developed the tool LogViz which is capable of showing multiple logging files and filtering the logging by activities and architectural elements [5]. In the time to come, we tend to contact with companies for validating our approaches.

References

[1] P. Blikstein. Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, LAK '11, pages 110–116, New York, NY, USA, 2011. ACM.

[2] B. W. Boehm. Software engineering. *IEEE Trans. Comput.*, 25(12):1226–1241, Dec. 1976.

[3] E. Chikofsky and I. Cross, J.H. Reverse engineering and design recovery: a taxonomy. *Software, IEEE*, 7(1):13–17, Jan 1990.

[4] J. Claes, I. Vanderfeesten, J. Pinggera, H. Reijers, B. Weber, and G. Poels. A visual analysis of the process of process modeling. *Information Systems and e-Business Management*, 13(1):147–190, 2015.

[5] M. R. C. Dave R. Stikkolorum, Truong Ho-Quang. Revealing students uml class diagram modelling strategies with webuml and logviz. In *(accepted for presentation at the Euromicro SEAA conference, August 26-28, 2015 and publication in the conference proceedings).*

[6] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting &Amp; Designing for Pervasive Information*, SIGDOC '05, pages 68–75, New York, NY, USA, 2005. ACM.

[7] C. A. Decker. Sensemaking in organizations, by k. e. weick. (1995). thousand oaks, ca: Sage. 321 pp., 44.00*cloth*,19.95 paper. *Human Resource Development Quarterly*, 9(2):198–201, 1998.

[8] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95, 2013.

[9] A. Dunsmore, M. Roper, and M. Wood. The role of comprehension in software inspection. *Journal of Systems and Software*, 52(23):121 – 129, 2000.

[10] Q. Gan, M. Zhu, M. Li, T. Liang, Y. Cao, and B. Zhou. Document visualization: an overview of current research. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(1):19–36, 2014.

[11] E. Hill, L. Pollock, and K. Vijay-Shanker. Exploring the neighborhood with dora to expedite software maintenance. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 14–23, New York, NY, USA, 2007. ACM.

[12] T. Ishio, S. Hayashi, H. Kazato, and T. Oshima. On the effectiveness of accuracy of automated feature location technique. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 381–390, Oct 2013.

[13] A. J. Ko, H. Aung, and B. A. Myers. Eliciting design requirements for maintenance-oriented ides: A detailed study of corrective and perfective maintenance tasks. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 126–135, New York, NY, USA, 2005. ACM.

[14] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Trans. Softw. Eng.*, 32(12):971–987, Dec. 2006.

[15] M. Osman, M. Chaudron, and P. Van Der Putten. An analysis of machine learning algorithms for condensing reverse engineered class diagrams. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 140–149, Sept 2013.

[16] M. Osman, M. Chaudron, P. Van Der Putten, and T. Ho-Quang. Condensing reverse engineered class diagrams through class name based abstraction. In *Information and Communication Technologies (WICT), 2014 Fourth World Congress on*, pages 158–163, Dec 2014.

[17] J. Pinggera, P. Soffer, S. Zugal, B. Weber, M. Weidlich, D. Fahland, H. Reijers, and J. Mendling. Modeling styles in business process modeling. In I. Bider, T. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, and S. Wrycza, editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 113 of *Lecture Notes in Business Information Processing*, pages 151–166. Springer Berlin Heidelberg, 2012.

[18] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej. How do professional developers comprehend software? In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 255–265, Piscataway, NJ, USA, 2012. IEEE Press.

[19] D. Shepherd, Z. P. Fry, E. Hill, L. Pollock, and K. Vijay-Shanker. Using natural language program analysis to locate and understand action-oriented concerns. In *Proceedings of the 6th International Conference on Aspect-oriented Software Development*, AOSD '07, pages 212–224, New York, NY, USA, 2007. ACM.

[20] Z. Soh, F. Khomh, Y.-G. Gueheneuc, and G. Antoniol. Towards understanding how developers spend their effort during maintenance activities. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 152–161, Oct 2013.

[21] M.-A. D. Storey, D. Čubranić, and 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, pages 193–202, New York, NY, USA, 2005. ACM.

[22] F. Thung, D. Lo, M. H. Osman, and M. R. V. Chaudron. Condensing class diagrams by analyzing design and network metrics using optimistic classification. In *Proceedings of the 22Nd International Conference on Program Comprehension*, ICPC 2014, pages 110–121, New York, NY, USA, 2014. ACM.

[23] W. M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[24] B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters, and W. van der Aalst. The prom framework: A new era in process mining tool support. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer Berlin Heidelberg, 2005.

[25] A. von Mayrhauser and A. Vans. From code understanding needs to reverse engineering tool capabilities. In *Computer-Aided Software Engineering, 1993. CASE '93., Proceeding of the Sixth International Workshop on*, pages 230–239, Jul 1993.

[26] A. von Mayrhauser and A. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, Aug 1995.