Software Developers' Information Needs: Towards the Development of Intelligent Recommender Systems

Adam Grzywaczewski, Rahat Iqbal, Anne James, John Halloran
Faculty of Engineering and Computing,
Coventry University, CV1 5 FB,
Coventry, UK
{grzywaca; r.iqbal; a.james; j.halloran} @coventry.ac.uk

Software development is a process which is significantly reliable on information and in the context of the Internet on Information Retrieval (IR) tools. Approximately 20%-30% of work time of software developers is spent on information retrieval and this proportion would be significantly higher if it were not for time constraints and pressure to deliver code. Even though a number of IR solutions exist, 86% of work-related search sessions start with a general purpose search engine. Therefore there exists a significant potential for research and development of ubiquitous, domain specific tools to support the IR process. This paper discusses how the knowledge of work tasks and information needs of software developers can be used to deliver ubiquitous, highly context-sensitive search and intelligent recommendation tools. We present a detailed review of software developers' work related tasks and habits. We also discuss factors that can be used as implicit feedback indicators for further collaborative filtering.

Personalised information retrieval, retention actions, copy and paste, software development, information seeking behaviour.

1. INTRODUCTION

Software development is a cognitively complex task (Détienne & Bott, 2002; Xu, 2004). As a consequence, even before the widespread use of the internet, software developers spent a significant proportion of their work-related time searching for various sources of information (Curtis et al. 1988; King & Griffiths, 1991; Seaman, 2002; Liu, Chen, Lakshminarayanan & Perry, 2005; Singer et al. 2010). The introduction of the internet in the software developers' work place did not change this proportion significantly (Freund, Toms & Waterhouse, 2005; Brandt et al., 2009) and still the ability to find and use various sources of information is one of the key abilities deciding about engineers efficiency and success (Tenopir & King, 2004). The availability of the internet changed the way developers acquire information which previously was significantly dependant on communication with co-workers (Hertzum & Pejtersen, 2000). The research suggests that the fact that online information is so easily available changes the social settings of work and discourages developers from interacting with their colleagues if it is not essential (Ye, Yamamoto & Nakakoji, 2007). On the other

hand it allows developers to distribute their cognitive resources (Hollan et al. 2000) across the web and engage in practices of opportunistic programming (Hartmann, Doorley & Klemmer, 2008, Brandt et al. 2009), reusing sections of code varying in length and complexity from single lines of text to entire frameworks and libraries. With the increasing numbers of open source projects and frameworks, and popularisation of technical blogs and libraries, software developers have started to perceive the internet as their "key information resource" (Brandt et al. 2009) and feel that if it were not for the commercial constraints they would spend an even higher proportion of time searching it for project-related information (Freund et al. 2005).

Table 1: Average amount of time software developers spend on information retrieval / communication

Paper	From	To	Research method
(Liu et al. 2005)	60%	60%	Questionnaire supported interview carried out on 9 participants
(Brandt et al. 2009)	14%	41%	Observation of 20 students

			<i>participating in a case study</i>
(Singer et al. 2010).	24%	45%	<i>Questionnaire on 13 participants, shadowing a single participant for 6 months</i>
(King & Griffiths, 1991)	20%	80%	-
(Freund et al. 2005)	20%	30%	<i>Interviews with 14 software engineers / consultants</i>
(LaToza et al. 2006)	5%	20%	<i>Survey carried out on 187 Microsoft corporation software developers</i>
(Perry et al. 1994)	16%	16%	<i>A yearlong diary study carried out on 13 participants</i>
(Tenopir & King, 2004; King et al. 1994)	40%	66%	<i>Very wide literature review specific to information use of engineers</i>

The results illustrated in Table 1 along with the reports of continuous feeling of “time famine” among software developers (Perlow, 1999) are motivation for the research of domain specific IR tools for software developers. Even though a significant number of search tools and utilities designed for software developers exist (Poshyvanyk et al. 2006; Begel, 2007 Hoffmann et al. 2007; Gallardo-Valencia et al. 2009; Bajracharya et al. 2009; Brandt et al. 2010) they tend to be not used by the software developers who rely in most of the cases on the general purpose search engine (Umarji et al. 2008). The reason for such software developers’ behaviour is not fully understood but there exist indications that it is due to insufficient understanding of the tasks the software developer engages in. This on the other hand makes it very difficult not only to design usable and task-focused user interfaces but also to control the results of the IR system to meet the temporary task at hand.

In order to deliver fully effective domain / task specific IR tools, the design process has to take all of the characteristics of the task and technological limitations into account (Jrvelin & Ingwersen, 2004). According to the Cognitive Approach to IR (Ingwersen & Jrvelin, 2005), to be able to design usable IR solutions we have to take into account the individual user’s cognitive space, the user’s social-organizational environment, the IR interface / intermediary, information objects and the IR system settings.

Even though a significant amount of literature exists aiming to understand software developers’ information behaviour, in the domain of Psychology

of Software Development (Sim et al. 1998; LaToza et al. 2006; Umarji et al. 2008; Janjic et al. 2010;) and Human Information Behaviour research (Ellis & Haugan, 1997; Freund et al. 2005; Buckley et al. 2006), it does not provide a consistent description of software developers’ task distribution and does not address the variations of factors laid out in the Cognitive Approach Theory.

This paper will provide a detailed overview of the distribution of the tasks and information needs of the software developer / software engineer. It will argue that a very detailed understanding of the tasks of the software developer can lead to development of very efficient and highly usable IR solutions. The argument will be supported by the detailed analysis of the task of software delivery and especially with regard to the practice of opportunistic programming. The paper will conclude with the analysis of two user study results supporting the feasibility of dedicated IR tool development for this particular activity.

2. SOFTWARE DEVELOPERS’ TASK DISTRIBUTION

One of the most comprehensive investigations of Software Engineers’ task distribution in the context of IR was carried out by Freund et al. (2005). The research provides an overview of the changing information needs and strategies used to find the required information. Freund et al. stress that the Software Engineers’ information behavior is highly shaped by the work context in which the information need is created and in which the information is searched and found. Their work suggests an existence of a causable relationship between the work task, context in which it is being performed and information behavior / strategies engaged in searching. The research highlights the fact that apart from engaging in code maintenance and delivery, the wider group of Software Engineers also engages in a variety of consulting tasks which are in most of the cases much more abstract and require different types of information and are also presented in a different format.

Software Developers also have significant information needs across their work-related tasks and use the internet even more frequently than the more general group of Software Engineers. They use the internet not only as a knowledge source but also as a source of snippets of various complexity and length that are being reused in the process of opportunistic programming (Brandt et al. 2009). Their work distribution is significantly different from widely understood Software Engineers as they focus mainly on: low level analysis and design, implementation, testing, optimization and

debugging, code deployment, and code maintenance.

To understand the task distribution as well as tools and most common practices of Software Developers, LaToza et al. (2006) conducted two surveys and eleven interviews on overall 187 software developers from Microsoft Corporation. The results revealed the existence of 9 main development tasks. The results revealed also the correlation between the stage of the project and the amount of time spent on particular tasks. The overall distribution of the tasks and their quantity remained the same across the stages. Most of the developers engage simultaneously in more than one activity during the week with the majority engaging in all 9 activities all the time. The results pointed out an existence of statistically significant correlations, both positive and negative, between the various tasks revealing the sequential nature of many of them.

The research carried out by Brandt et al. (2009) suggests that the way software developers search for code can be classified on a linear scale depending on the novelty of the information which is required. The position of the information on that scale impacts not only the way users search for code, the type of code, but also the search process characteristics.

Similar classification was reported by UmarjiSim & Lopes (2008) who also classified the web searching behavior of coders based on the same spectrum (code for reuse/reference example). An additional dimension was added to the classification that is the size of the code related content that is being investigated. The experiment was carried out on a group of 69 professional software developers and based on the results and the proposed classification, 9 archetypes of behavior were identified spanning across most of the code development related IR cases. The above archetypal classification was then further extended to support the variation of developers' behavior across different stages of the project (Janjic et. 2010). The results illustrate that the different tasks / archetypes of behavior are not equally probable during different stages of the process. It is also important to notice that the stage of the process has also a direct impact on the information needs decreasing their abstraction level with the maturity of the project or project iteration. This supports the results from LaToza et al. (2006). Their empirical results also clearly demonstrate the variation of the importance of different tasks across different stages of the development lifecycle. The results of their study show that along with the maturity of the project, developers spend less time delivering new features and focus more on bug fixes and code maintenance.

Additionally their results suggest that most of the specific information, related to the usage of code, library or framework is being reported during the implementation and testing stages of the project. Other stages of the project involve the retrieval of more abstract information. In this research the notion of Software Developer was used quite arbitrarily though, and in a significant proportion of projects the phases of analysis and design as well as deployment and maintenance would be assigned to different engineers in the company.

The questionnaire based user study carried out on a group of 67 software developers by Sim et al. (1998) also identified a similar set of work-related tasks. The tasks were classified based on the search motivation and were reported to span over the majority of software development activities. The most common search tasks reported were: defect repair, code reuse, program understanding, feature addition, and impact analysis. On top of this classification, 11 archetypes of software development behavior were designed and used for an in-depth analysis of the search process, search behavior and the decision making process involved in search.

Overall the literature is quite consistent in terms of the description of the Software Developers' work tasks and similar or even identical tasks are observed. The literature is also quite consistent as far as the impact of the project role/project stage has on the set of software-related responsibilities and as a consequence on the search requirements. In all of the cases, architectural, code maintenance, code testing and development tasks required the usage of different data sources, different search strategies and also resulted in different use of the captured information. The next section of this paper will focus exclusively on tasks related to code delivery and more precisely on an opportunistic programming approach to code development.

3. SOFTWARE DELIVERY AND OPPORTUNISTIC PROGRAMMING

The proportion of time software developers spend on code delivery varies with the development team, distribution of responsibilities and the project. The code delivery is obstructed by multiple interruptions and gaps in their knowledge which require the use of various forms of communication. For all of them though the delivery of code is the key activity and the overall goal of their work. Software engineers are significantly results-driven and if it is not necessary then in many cases they are not interested in the full understanding of the problem or piece of code that is being implemented. As a consequence software engineers often copy the solutions, even very loosely matching their problem

and, “see what happens”, engaging in the process of programming by example/iterative programming (Brandt et al. 2008; Yeh et al. 2008;). The ability to search, copy and paste and then experiment on the sections of third party code is very valuable to the programmers because it allows them to distribute their cognitive resources across the internet (Hollan et al. 2000). Due to the availability of the online resources they can ignore the details of implementation of simple snippets of code, bigger sections and modules focusing exclusively on the task of gluing them together. The literature suggests that the extent of the code reuse through copy and paste is very high and reaches up to 30% of the code in the solution (Brandt et al. 2009). The research carried out by Kim et al. identified that the average user performs approximately 16 copy and paste operations per hour (Kim et al, 2004) out of which 2 are non-trivial sections of code and all of them carry relevant information. Similarly the research carried out by Umarji et al. (2008) reveals that 73% of search sessions could be classified as “searching for code for reuse”. From the perspective of this discussion, the most interesting type of search identified, so the one that relates to non-trivial copy and paste operations in the previously discussed research, is the search for the subsystem components (41%).

The information listed above provides us with a significant potential for development of ubiquitous search and recommendation tools aiming to solve subsystem related code retrieval problems. From this information a number of design related conclusions can be made:

- (i) The act of copy and pasting is an implicit indicator of relevance. The ability to capture the information related to copy and paste allows us to create a knowledge containing keywords used for search, content that was copied and the ID of the user that copied the information. The above knowledge can be quite easily used with existing collaborative filtering approaches to provide recommendations of related content.
- (ii) The information regarding the subsystem level code should not necessarily be provided directly in the development environment as it cannot be simply copy and pasted to the solution but instead will require significant modifications to contextualize it. Instead it should be accessible through the browser or even the search engine page providing a single and consistent search and recommendation of user interface

- (iii) The summary information related to the recommended content should be more focused on the functionality and the textual content used to describe the problem by other developers.

The next sections of this paper will describe the user studies designed to validate those design conclusions.

4. METHODOLOGY

The goal of the experiment was to assess the following:

- Whether the copy and paste action indeed is an indicator of relevance.
- Whether the copy and paste occurs frequently enough to make it possible to quickly and effectively build a comprehensive knowledge base.
- What is the nature of the copy and pasted content (whether block, subsystem or system information is most frequently copy and pasted?)

In order to answer the following questions a two stage experimental process was designed.

4.1 Feasibility assessment

Feasibility assessment stage was carried out on 12 professional software developers. All participants were men with at least 5 years of professional software development experience. All of the participants were specialized in Microsoft .NET technologies with a special emphasis on MOSS2007 development. All of the participants were engaged almost exclusively in code delivery and tests. The user study had a form of a questionnaire with a variety of open-ended and close-ended questions followed by a short interview. Most of the questions provided the user with a set of exemplary answers which was followed by a number of empty text fields requiring more than one answer from the user.

4.2 Automated observation of user behaviour

A user study was carried out on the same group of software developers. The user study was carried out using a number of browser add-ins (Chrome, Fire Fox, IE) and a development environment add-in (Visual Studio 2003 and upwards) designed to automatically capture user interaction with information.

The automated observation lasted for a different period of time depending on the participant's availability. For the purpose of the data analysis a

common period of 2 months was chosen between the 4th of January 2011 and 1st of March 2011. The results reported in this paper are based mainly on the output of the development environment add-ins. The output of the add-ins is composed of a number of reports. A report with the detailed description of user interaction is sent when the document on which the developer worked is closed. For the purpose of this analysis 665 non-trivial reports were selected. The report is classified as non-trivial if: the time the user spent working on the document is above the selected threshold (20 seconds) or the report contains either 'copy & paste' or 'focus change' information.

5. RESULTS

5.1 Feasibility assessment

The first goal of the user study was to identify the frequency with which software developers used the internet to solve their work-related problems and how frequently they have been reusing the online content in their applications. Table 2 demonstrates the normalized frequencies on a per-day and per-working-year basis. It is important to notice that those results are self reported and therefore had to be verified by the second stage of the user study.

Table 2: Frequency of Internet resource usage

Description	Actions per/day	Actions per/260 days
How often does your work related problems require from you the use of Internet/Intranet?	5.15	1338.1
How often do you look for code documentation and code examples?	1.97	511.7
How often do you reuse the code snippets of code from external sources to rework them in your projects	0.37	96.8

The second goal of the feasibility study was to identify what happens to the relevant information once it is found. Table 3 summarizes the most frequently reported actions.

Table 3: The reported usage of relevant information

Description	Answer count per 14	Answer %
Once you find the information you need what do you usually do with it?		
Just read it and use the knowledge	12	85.71%
Copy the code snippet or code example	12	85.71%
Read the information in the same time solving the problem	12	85.71
Print the information	4	28.57%
Bookmark, email or save the information in any way	7	50.00%
Share the information with other people	6	42.86%

All of the participants reported that they often copy and paste the relevant information or alternatively directly and immediately use it to solve their development problems. Two responders when discussing the question during the short interview reported that they often copy even very loosely related code to be able to obtain the template for further work.

The third goal was to identify the information that is being searched for in the internet. Table 4 lists the most frequently reported information needs. We can see that only the search for code examples / snippets / modules was selected by all of the users but still there is a very high consistency in the answers provided by the users.

Table 4: The most frequently required information

Description	Answer count per 14	Answer %
What work related information do you look for on the internet		
Code documentation.	8	57.14%
Code examples.	12	85.71%
Solutions to some code related problems.	11	78.57%
General ideas regarding the approach to take.	10	71.43%
Technological news.	6	42.86%
Technical Material.	6	42.86%

During the post-questionnaire interview participants highlighted the fact that they most frequently search for highly processed information. If the code examples / code solutions are not available or not appropriate to the problem, they usually search through highly processed information sources such

as blogs containing other people descriptions of the problems.

5.2 Automated observation of user behaviour

We have recorded a total of 1296 non-trivial copy and pastes out of which 379 originated from outside of the development environment. Our results indicate that the presence of copy and pasting from the browser is an indicator of higher problem complexity what is manifested by much longer average active time across sessions that involved copy and pasting from the browser.

Table 5: Active times of interaction: general statistics

Description	Q1	Q2	Q3	Average
Browser	40.25	133	242.25	203.69
Non-browser	25.25	87	117.25	98.40
Development	24	69	74	74.82

Figure 1 illustrates the distribution of copy and paste actions in time. The results show that even though copy and pasting from the browser is not as frequent as in internal development environment operations, it is much more frequent than reported in the feasibility study. The distribution of browser interaction seems to have a normal shape with an average of 120 minutes with Q1=26 min and Q3 = 132 min (as shown in Table 5).

Table 6 shows the results of the analysis of the copy and pasted text depending on the source of copy and paste. The average number of characters copy and pasted from the browser was more than two times higher than the average number of characters copied from other sources. On the other hand users copied the information from the browser much less frequently than internally in the development environment. Single copy and pastes account for 60% of copied data, with the maximum of 8 copy and pastes in a single report (a single interaction with a source code file).

Figure1: The average time between the Copy and Paste actions.

The data for development environment copy and pasting also contains a lot of isolated events with 60% of copy and pastes occurring only 1-3 times in the report (a single interaction with a source code file). The remainder of the reports contains significantly higher values ranging 44-46 indicating significant amount of work carried out on the document.

Finally the relationship between the copy and pasting and project building was investigated:

- When the user copy and pasted the information from the browser, the build action was executed in 57.5% of cases after the paste and with the focus in the same window
- When copy and pasted the information from the development environment the build took place in 11.14% of cases after the paste and with the focus in the same window

This indicates that the user was not certain about the results of copy and pasting and used the build option to validate the correctness of the code and to further test it.

Table 6: Copy and pasted content analysis

Description	Average
Average number of characters copied from the browser	230.05
Average number of characters copied internally in the development environment.	107
Average number of copy and pastes from the browser in a single report	2.24
Average number of copy and pastes from the browser in a single report (the reports having just one copy and paste were excluded from the calculation)	3.86
Average number of copy and pastes internally in the development environment	5

4.2 Summary

Overall the copy and paste action was indeed reported by all of the participants as an indicator of relevance. The results of our feasibility study and the automated observation were further supported by the interviews that were concluding the feasibility study. On the other hand the perception of relevance reported by the participants is quite complex and related not only to the similarity of the code to the problem they want to solve but also completeness of code and its usage license.

Secondly the copy and paste action of subsystem components did occur frequently enough to consider its usage as a source of implicit relevance feedback with a non-trivial copy and paste occurring every 120 minutes.

The code that was copy and pasted from the browser was complex and averaged in length to 230.05 characters (with values ranging from 6 to 2031 characters).

6. CONCLUSIONS

The literature review presented in this paper illustrates that the software development process is composed of a fixed amount of repeatable development tasks. Across each of those tasks and also across different stages of the software development lifecycle, software developer faces a fixed number of information need types. One of such needs is the need to reuse the code found on the internet to decrease the cognitive burden of software development and increase the speed of development. The further analysis and the user studies indicate that it is possible to infer content relevance from the copy and paste action which can be perceived as a strong implicit feedback indicator. Additionally the analysis of the software development tasks allowed us to create a number of user interface guidelines for the process of retrieving non-trivial code snippets.

What is more important, the paper provided an example of the process that can be undertaken in order to design and then deliver domain specific recommendation systems. The process is based on a detailed analysis of the user group task distribution, isolation of the information intensive tasks that are characteristic for the user group and step by step analysis of user behaviour during the execution of the task in the work environment. By application of this process, the results of this user study can be further generalised to other domains.

It is also valuable to notice that such, domain specific, recommendation systems are already designed and successfully used especially by e-commerce even though a detailed design methodology does not yet exist:

- Amazon uses the event of purchase and product visit as an indicator of relevance.
- Last.FM uses the event of music playback as a source of implicit information
- Gmail uses a combination of email related indicators, such as reading or replying, to infer relevance / priority of new incoming email

Therefore further formalisation of the process has a significant potential and can lead to development of numerous domain/task specific IR tools supporting business critical tasks in the modern enterprise.

7. REFERENCES

- Bajracharya, S., Ossher, J., & Lopes, C. (2009). Sourcerer: An Internet-scale Software Repository In Proceedings of the 2009 ICSE Workshop on Search-Driven Development -Users, Infrastructure, Tools and Evaluation (SUITE '09). IEEE Computer Society, Washington, DC, USA, 1-4.
- Baldwin, V. A., & Hallmark, J. (2003). Information and the Professional Scientist and Engineer. Routledge.
- Begel, A. (2007). Codifier: a Programmer-centric Search User Interface. Paper presented at the Workshop on Human-Computer Interaction and Information Retrieval. Cambridge, Massachusetts, USA, 23-24
- Brandt, J., Dontcheva, M., Weskamp, M., & Klemmer, S. (2010). Example-centric Programming: Integrating Web Search into the Development Environment. In Proceedings of the 28th International Conference on Human factors in Computing Systems (CHI '10). ACM, New York, NY, USA, 513-522.
- Brandt, J., Guo, P., Lewenstein, J., Dontcheva, M., & Klemmer, S. (2009). Two studies of Opportunistic Programming: interleaving Web Foraging, Learning, and Writing Code. In Proceedings of the 27th International Conference on Human factors in Computing Systems (CHI '09). ACM, New York, NY, USA, 1589-1598
- Brandt, J., Guo, P., Lewenstein, J., & Klemmer, S. (2008). Opportunistic programming: how rapid ideation and prototyping occur in practice. In Proceedings of the 4th International Workshop on End-user software engineering (WEUSE '08). ACM, New York, NY, USA, 1-5.
- Buckley, J., P. O'Brien, M., & Power, N. (2006). Empirically Refining a Model of Programmers' Information-Seeking Behavior during Software Maintenance. Paper presented at the 18th Annual Psychology of Programming Interest Group (PPIG) Workshop Brighton, UK:
- Curtis, B., Krasner, H., & Iscoe, N. (1988). A Field Study of the Software Design Process for Large Systems. Communications of the ACM, 31(11), 1268-1287.
- Détienne, F., & Bott, F. (2002). Software design--cognitive aspects. Springer Verlag.
- Ellis, D., & Haugan, M. (1997). Modelling the Information Seeking Patterns of Engineers and Research scientists in an Industrial Environment. Journal of Documentation, 53(4), 384-403.

- Freund, L., Toms, E., & Waterhouse, J. (2005). Modeling the Information Behaviour of Software Engineers using a Work-task Framework. *Proceedings of the American Society for Information Science and Technology*, 42(1), n-a.
- Gallardo-Valencia, R., & Elliott Sim, S. (2009). Internet-Scale Code Search. In *Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation (SUITE '09)*. IEEE Computer Society, Washington, DC, USA, 49-52.
- Hartmann, B., Doorley, S., & Klemmer, S. (2008). Hacking, Mashing, Gluing: Understanding Opportunistic Design. *IEEE Pervasive Computing* 7(3), 46-54.
- Hertzum, M., & Pejtersen, A. (2000). The Information-seeking Practices of Engineers: searching for Documents as well as for People. *Information Processing & Management*, 36(5), 761-778.
- Hoffmann, R., Fogarty, J., & Weld, D. (2007). In *Proceedings of UIST 2007: ACM Symposium on User Interface Software and Technology*, pages 13–22, Newport, Rhode Island, 2007.
- Hollan, J., Hutchins, E., & Kirsh, D. (2000). Distributed Cognition: toward a New Foundation for Human-computer Interaction Research. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(2), 174-174.
- Ingwersen, P., & Jrvlin, K. (2005). *The Turn*. Kluwer Academic Pub.
- Janjic, W., Hummel, O., & Atkinson, C. (2010). More Archetypal Usage Scenarios for Software Search Engines. In *Proceedings of 2010 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation (SUITE '10)*. ACM, New York, NY, USA, 21-24.
- Jrvlin, K., & Ingwersen, P. (2004). Information seeking Research needs Extension towards Tasks and Technology. *Information Research*, 10(1), 361-382.
- KiM, J. (2008). Task as a Context of Information Seeking: An Investigation of Daily Life Tasks on the Web. *Libri*, 58(3), 172-181.
- Kim, K. S., & Allen, B. (2002). Cognitive and Task Influences on Web Searching Behavior. *Journal of the American Society for Information Science and Technology*, 53(2), 109-119.
- Kim, M., Bergman, L., Lau, T., & Notkin, D. (2004). An Ethnographic Study of Copy and Paste Programming Practices in OOPL. In *Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE '04)*. IEEE Computer Society, Washington, DC, USA, 83-92.
- King, D. W., Casto, J., & Jones, H. (1994). *Communication by Engineers*.
- King, D., & Griffiths, J. (1991). Indicators of the Use, Usefulness and Value of Scientific and Technical Information. In D. Raitt (Ed.), *Proceedings of the 15th International Online Meeting London*, December 10-12. Oxford, UK: Learned Information Ltd., 361-377
- LaToza, T., Venolia, G., & DeLine, R. (2006). Maintaining Mental Models: a Study of Developer Work Habits. In *Proceedings of the 28th International Conference on Software engineering (ICSE '06)*. ACM, New York, NY, USA, 492-501.
- Leckie, G., Pettigrew, K., & Sylvain, C. (1996). Modeling the Information Seeking of Professionals: a General Model derived from Research on Engineers, Health Care Professionals, and Lawyers. *The Library Quarterly*, 161-193.
- Leckie, G. J. (2005). General Model of the Information Seeking of Professionals. In *Theories of information behavior Information Today*, 158 - 163Liu, W., Chen, C., Lakshminarayanan, V., & Perry, D. (2005). A Design for Evidence-based Software Architecture Research.
- Perlow, L. (1999). The Time Famine: Toward a Sociology of Work Time. *Administrative Science Quarterly*, 44(1), 57-81.
- Perry, D., Staudenmayer, N., & Votta, L. (1994). People, Organizations, and Process Improvement. *Software, IEEE*, 11(4), 36-36.
- Poshyvanyk, D., Marcus, A., & Dong, Y. (2006). JIRiSS - an Eclipse Plug-in for Source Code Exploration. In *Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC '06)*. IEEE Computer Society, Washington, DC, USA, 252-255.
- Seaman, C. (2002). The Information Gathering Strategies of Software Maintainers. In *Proceedings of the International Conference on Software Maintenance (ICSM'02) (ICSM '02)*. IEEE Computer Society, Washington, DC, USA, 141-150.
- Sim, S., Clarke, C., & Holt, R. (1998). Archetypal Source Code Searches: A survey of software developers and maintainers. In *Proceedings of the 6th International Workshop on Program Comprehension (IWPC '98)*. IEEE Computer Society, Washington, DC, USA, 180-187.

- Singer, J., Lethbridge, T., Vinson, N., & Anquetil, N. (1997). An examination of Software Engineering Work Practices. In Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative research (CASCON '97), IBM Press, 174-188.
- Tenopir, C., & King, D. W. (2004). Communication Patterns of Engineers. Wiley-IEEE Press.
- Umarji, M., Sim, S., & Lopes, C. (2008). Archetypal Internet-scale Source Code Searching. Open Source Development, Communities and Quality, 257-263.
- Xu, S. Rajlich, V. (2004) Cognitive Process during Program Debugging, In Proceedings of the Third Conference on Cognitive Informatics (ICCI'04), 176-182.
- Ye, Y., Yamamoto, Y., & Nakakoji, K. (2007). A Socio-technical Framework for Supporting Programmers. In Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering (ESEC-FSE '07). ACM, New York, NY, USA, 351-360.
- Yeh, R., Paepcke, A., & Klemmer, S. (2008). Iterative Design and Evaluation of an Event Architecture for Pen-and-paper Interfaces. In Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology (UIST '08). ACM, New York, NY, USA, 111-120.