Cognition and Distributed Cognition in Software Engineering Research[WIP]

Marjahan Begum
Lecturer
Department of Computer Science
City University London
Marjahan.Begum@City.ac.uk

Abstract

The purpose of this paper is to conduct an exploratory literature review in the area of cognition and distributed cognition within software engineering (SE) research. Over arching aims of this literature review are to understand the how individual cognition and distributed cognition are discussed in software engineering research. This is to support empirical studies on the role cognition (individual or team) plays during software development process.

The finding suggest that most of the research to date is empirical but there is also some theoretical research. The research focus has been on: program comprehension, design and review, distributed cognition and how tools can reduce cognitive load for programmers. This analysis provides opportunities for further research in terms of cognitive models and methodological insights.

1. Introduction

The purpose of this paper is to conduct an exploratory literature review in the area of cognition and distributed cognition in software engineering research. Over arching aims of this literature review are to understand the how individual cognition and distributed cognition are discussed in software engineering research. This is to support empirical studies on the role cognition (individual or team) play during software development process. Even though there are advances in technologies and in software engineering processes, fundamentally individuals within teams will design and will continue to develop software. This will continue to be so in a collaborative and in a distributed fashion. For the purpose of this research an all encompassing definition of cognition is used here. It is higher order cognition processes such as generating hypothesis and lower order cognitive processes such a recalling(Renumol, Janakiram, & Jayaprakash, 2010). Cognitive processes are associated with the way information is processed, retrieved and stored for later use. It is not the intention of this paper to make any claims about a pure definition of cognition in the context of software engineering but rather to explore how the concepts around cognition are discussed in the literature in the software engineering. To the best of the researcher's knowledge there is no comprehensive literature review on how cognition is discussed in software engineering literature.

2. Methodology

A semi-structured and exploratory approach to literature review was done using inspiration from Snowballing technique(Wohlin, 2014). Such an exploratory literature study was also used in studies in software engineering(Helgesson, Engstrom, Runeson, & Bjarnason, 2019).

It was semi-structured because key research databases, journals and conferences were identified. Each of the databases was then searched separately.

2.1. Databases and Sources

Combinations of search strings were used that consisted of the following keywords or phrases: "abstraction", "awareness", "chunking", "cognitive", "cognition", "distributed cognition", "mental imagery", "reason/reasoning" , "inspection", "recognition", "sense making", and "tracing"

Combination of these search strings were applied in two major searches followed by some 'adhoc' searches in the databases Scoupus, Science Direct, IEEE Xplore, ACM Digital Library specific journal and conference avenues. Journals and conferences were identified from an established grounded theory literature review(Stol, Ralph, & Fitzgerald, 2016). They are listed below.

- Information and Software Technology
- Journal of Systems and Software
- IEEE Transactions on Software Engineering
- Empirical Software Engineering Journal
- Software Process: Improvement and Practice
- Journal of Software: Evolution and Process
- Software Quality Journal
- ACM Trans. Software Engineering and Methodology
- Journal of Software Maintenance and Evolution: Research and Practice
- International Symposium on Empirical Software Engineering an Measurement
- International Conference on Software Engineering
- Journal IEEE transaction on Software Engineering

2.2. Inclusion and Exclusion Criteria

Their inclusion was confirmed by reading their titles and abstracts.

In order to narrow down the articles for an in depth study, inclusion and exclusion criteria were developed iteratively by reading the abstracts of the papers and making sense of the research. The main criteria were that followings:

- Peer reviewed conferences and journals
- Empirical software engineering
- Software engineers behaviour, team and social aspects

Peer-reviewed papers were chosen as they went through quality assurance with peers working in the similar areas. Team and social aspects were important as software development is a team activity and practice is always within a social and organisational structure. This resulted in 24 articles.

3. Results

24 papers were read in detail and were then reduced to only 13 research papers. These are presented in this paper and briefly summariezed in Table 1.

Table 1 – Research related Cognition in Software Engineering

No.	Study	Brief Overview
1.	(Cant, Jeffery, & Henderson-Sellers, 1995)	This research makes the connections between software complexity and cognitive complexity based on cognitive processes used by programmers. It proposes a cognitive model which describes cognitive processes during maintaining, modifying, and extending, testing and understanding code. These are chunking (code comprehension) and tracing (searching). It offers a hypothesis driven by scanning the code with a combination of forward/backward tracing. These two cognitive processes require syntactic and semantic knowledge. The contribution of this research is modelling of complexity of programmers' processes

2.	(Arunachalam & Sasso, 1996)	The study explores cognitive processes engaged in the program comprehension. The study introduces explanatory generation, confirmatory association and explanation validity. It also includes cognitive processes related to design documents such as pseudocode or flow charts or mapping to business domain.
3.	(Zhuge, Ma, & Shi, 1997)	This paper argues that software process is a special case of human problem solving and calls cognitive space. It presents a cognitive based the problem solving framework. The framework includes: analogy, abstraction, knowledge content, control strategies, top-down and bottom up.
4.	(Shaft & Vessey, 1998)	This study shows that top-down and bottom-up theories of program comprehension due to differences in the knowledge of application domain and how meta-cognition are used.
5.	(Robillard, D'Astous, Detienne, & Visser, 1998)	Investigates cognitive activities in software team work during a technical review meeting. The activities were evenly distributed among hypothesising, evaluating, justifying and informing.
6.	(Storey, Fracchia, & Müller, 1999)	Provides a cognitive framework for a tool to aid exploration and comprehension. The framework includes top-down, bottom up and integrated model. It concludes that cognitive load needs to be considered in the design of tools.
7.	(Herbsleb, 1999)	Based on content analysis of conversation during nine domain analysis meetings, the research looks at how metaphorical representations are used in domain analysis. The result shows that 70% data was metaphorical. For example systems behaviour described as physical movement of objects.
8.	(Dunsmore, Roper, & Wood, 2000)	The research conducted an experiment with bottom up and as needed strategy to identify defects but it is not conclusive about which approach is better.
9.	(Karahasanović, Levine, & Thomas, 2007)	Explores strategies and difficulties when conducting a maintenance task of a software system. The research found that there are two level of difficulties: development environment and difficulties with software development process (mainly comprehension). In the research programmers were required to use use systematic and as needed strategy.
10.	(Ko, Deline, & Venolia, 2007)	The research shows how developers maintain awareness for their tasks. This was done through actively seeking it, through scheduled meetings and through ad-hoc meetings and through collaborative tools
11.	(Petre, 2010)	The research examines the relationship between software visualization and effective software developers' reasoning specifically during design and generation. It specifically discusses roles of mental imagery and external representation during design.
12.	(Omoronyia, Ferguson, Roper, & Wood, 2009)	Introduces a 'Continuum of Relevance Index' (CRI) model to help with building awareness in distributed software development team in their collaborative space. The model tries to show only relevant/dependent tasks and required artefacts for particular task in the software engineering process.
13.	(Lavallee, Robillard, & Paul, 2013)	This is a mapping study in the area of cognition and software engineering. It argues for team meta-cognition, team situation awareness and team problem solving in software development process

Software engineering processes involve a number of processes such as program comprehension, debugging, design and testing. These activities are carried out by software developers or engineers using cognitive activities and knowledge (internal or distributed). Cognitive processes are higher level processes as defined by Renumo et al (Renumol et al., 2010). Some examples of these processes are recognition, imagery, comprehension, learning, reasoning, deduction, induction, decision making, problem solving, explanation, analysis, synthesis, creation, analogy, planning, and quantification. This literature review shows that most of research in SE discusses cognition and distributed cognition in the context of program comprehension which occurs during maintenance, adding features, debugging or simply to understand the program. This is corroborated by the finding of a research on program development which concluded that programmers spent 70% of their time in program comprehension (Minelli, Mocci, & Lanza, 2015). The general conclusion is that there are cognitive difficulties with such activities and there is a body of research argues for better design of tools to support software engineers in these processes (Petre, 2010; Storey et al., 1999). See Section 3.4.

3.1. Cognitive Processes in Software Engineering

Two opposing cognitive processes dominate in research related to program comprehension, top-down vs bottom-up.

Top-down is the hypothesis driven understanding of the code by recognising its syntactic or semantic structure which is followed by validating or invalidating the hypothesis (Cant et al., 1995; Shaft & Vessey, 1998; Storey et al., 1999; Littman, Pinto, Letovsky, & Soloway, 1986; Arunachalam & Sasso, 1996). Hypothesis building involves reconstruction of knowledge or mental models held by the programmers. In the literature it is generally argued that the top-down approach is used when programmers are working on familiar domains or technologies. The Top-down approach is also associated with tracing of code and systematic strategy (Cant et al., 1995; Littman et al., 1986).

Bottom-up processes involve making inferences from small lines of code and then building to form higher levels of abstraction (Arunachalam & Sasso, 1996)(Storey et al., 1999). This approach is associated with the situation where a code is new to the programmers and sometimes called chunking (Cant et al., 1995). Bottom-up processes build the abstraction of knowledge while top down rebuilds the previously held knowledge. Top-down is often associated with experts and bottom up approach is associated with novices (Burkhardt, JM., Détienne, F. & Wiedenbeck, 2002; Cant et al., 1995; Shaft & Vessey, 1998; Storey et al., 1999).

Another approach is called integrated or as-needed strategy in the context of program comprehension and is often associated with opportunistic behaviour. Based on the task at hand and knowledge base, the programmer decides which approach to use (Storey et al., 1999; Dunsmore et al., 2000).

Often programmers use patterns of activities and patterns of cognitive processes to achieve a goal, be it program comprehension, design or problem solving. For example, during program comprehension there is evidence of explanatory generation followed by confirmatory association and then again explanatory generation(Arunachalam & Sasso, 1996). In the problem solving context, control strategies are used to transform a problem from its initial state to a goal state using a combination of analogy and abstraction(Zhuge et al., 1997). In an empirical setting of a technical review meeting, the team distributed their activities among hypothesising, evaluating, justifying and informing (Robillard et al., 1998).

It should be noted that all the research argues that no one model can explain the behaviour underlying programming in a unified way.

3.2. Internal and External Representations

The cognitive processes described above interact with representation in the form of mental models or patterns held by programmers based on their previous experience. Mental models are formed iteratively through various levels of abstractions. The knowledge required is called syntactic or semantic knowledge (Burkhardt, Detienne, & Wiedenbeck, 1998) . Mental models are progressively formed through using the semantic knowledge which is language independent and can also be about the domain knowledge (Shaft & Vessey, 1998). Syntactic knowledge is language dependent and is about the structural elements of the program.

In the problem solving context, cognitive space consists of analogy and abstraction as illustrated above, but it also consists of knowledge content (Zhuge et al., 1997). In this context knowledge appears to be explicitly about concepts, relationships, rules, experiences and theory.

Mental imagery is another form of representation that has been studied (Petre, 2010). Mental imagery takes many forms such as dancing symbols, auditory images, visual images, machine/artifacts with their own minds, surfaces and landscapes. When these images are externalised they become focal points for design discussions. Complementary research shows that during domain analysis, software engineer use metaphorical representations 70% of the time (Herbsleb, 1999). Software engineers describes the behaviour of systems as physical movement of objects,(”go”, ”give”,”come back”) as perceptual processes (”look”, ”see”, ”search”) or in anthropomorphic terms of the system having beliefs and desire (”knows”, ”trying to do”).

3.3. Awareness and Distributed Cognition

In its simplest form, awareness is about knowing who, what, when, where and how. But in the literature there are four types of context awareness: informal awareness, group-structural awareness, social awareness and work-space awareness (Omoronyia et al., 2009; Gutwin, Penner, & Schneider, 2004). These types of awareness are important for enhancing the coordination and efficiency of teamwork. Researchers have developed the Continuum of Relevance Index (CRI). They used it empirically to enhance developers' awareness about other developers and about tasks and artefacts that are most relevant in their particular work contexts. Awareness is particularly crucial in distributed software development (Omoronyia et al., 2009; Gutwin et al., 2004) where the traditional notion of knowledge being stored in the software developer's head does not entirely solve the modern challenges in software engineering. An example of how awareness is maintained is through the use of text based communication channels i.e version control systems like Git and email forums.

A discussion of the theory of distributed cognition is very relevant here. The theory can explain how software developers need to consider systems outside the immediate context in terms of people, processes and artifacts to solve modern software engineering challenges(Hollan, Hutchins, & Kirsh, 2000).

Distributed Cognition suggests that

- Cognitive processes may be distributed across the members of a social group;
- Cognitive processes may be distributed between internal and external (material or environmental) structure; and
- Cognitive processes may be distributed through time in such a way that the products of earlier events can transform the nature of later events.

In a mapping study of distributed cognition in software engineering, the research identified five major concepts(Lavallee et al., 2013). Two of the relevant concepts here are community (team-meta cognition and task awareness) and cognitive support tools to support task awareness. The community concept resonates with the awareness concept illustrated at the beginning of this section(Omoronyia et al., 2009). Cognitive support tools are to help with collaboration and communication in team contexts.

3.4. Tools to support software engineering

Due to the nature of software development, software engineers have to work with multiple artifacts and at multiple levels of abstraction. This creates cognitive overhead and researchers have tried to address these issues (Karahasanović et al., 2007; Arunachalam & Sasso, 1996; Storey et al., 1999; Dunsmore et al., 2000; Petre, 2010; Lavallee et al., 2013). Computer mediated knowledge collaboration tools should address socio-technical challenges. Difficulties associated with the programming environment can be reduced through the design of tools that reduce cognitive overheadsduring browsing, navigating, and visualising of the software architecture(Storey et al., 1999; Karahasanović et al., 2007). Research informed design of the tool is advocated so that top down, bottom up and integrated approaches to software development are supported (Petre, 2010). Visualisation tools that support understanding of software architecture and domain knowledge are a step in the right direction. Difficulties associated with programming logic and testing procedures are addressed through specific tools for testing and debugging(Karahasanović et al., 2007). It is often software engineers who develop their own task specific tools as available tools do not either reduce the cognitive load or do not support the development process(Petre, 2010).

4. Discussion

This is work in progress research to help design empirical studies on cognitive processes used by developers and engineers during the software development process. By reviewing the literature it is intended that, future empirical studies will have some basis for theoretical foundation (cognitive models explored in previous literature) and types of methods used to study engineers and developers.

Should a study be designed to study individual cognition in software engineering and or from a distributed and collaborative perspectives?

If individual cognition is to be studied, perhaps in the design of the empirical work it is useful to know when software developers debug or extend a software what kind of the knowledge they use during chunking and tracing(Cant et al., 1995). So the question stands as to the type of syntactic and semantic knowledge that they use? Where do they get this knowledge from (individual or distributed)? How do they make use of the "project memory"? What tools do they use to help them with their tasks. If domain knowledge is important where do they get it from. What are their difficulties and how do they overcome these difficulties? What role does an individual's knowledge and previous experiences play?

If team cognition and software engineering is to be studied how should the empirical study be designed? What methods should one use and what analytical framework could be used?

More broadly is the question of cognition and software engineering an important research avenue to explore? Would such a study have impact on SE practices and research?

The findings in this literature review provide some basis for the design of the empirical research to answer the above questions. It provides basis for developing a framework for the empirical work and for developing research tools such as observational framework, interview design (group and individual).

For example, for future empirical work, CRI(Omoronyia et al., 2009) and the workspace awareness framework (Gutwin et al., 2004) could be used to frame observations of and interviews with software engineers. Different cognitive models identified can be used to provide an initial theoretical foundation to design interview protocols or even inform a coding scheme for analysing software development activity.

It should be noted that this literature review is carried out to understand the current state of play in how cognition is discussed in software engineering to help carried out empirical research in industrial context in this area.

5. Conclusion and Limitation

The main conclusion that can be drawn is that in cognition and software engineering not much research has been done beyond code comprehension, with limited research in modern software engineering environment (Helgesson et al., 2019). Though there is established understanding that distributed cognition is an appropriate framework to investigate software engineering practice there are limited studies in this area.

This project does not claim to cover all the research in this area. The following literature have not be searched

- Global Software Engineering
- Open source knowledge sharing community
- Psychology of Programming
- Computer Supported Collaboration Learning

6. References

Arunachalam, V., & Sasso, W. (1996). Cognitive processes in program comprehension: An empirical analysis in the context of software reengineering. *Journal of Systems and Software*, *34*(3), 177–189. doi: 10.1016/0164-1212(95)00074-7

Burkhardt, J. M., Detienne, F., & Wiedenbeck, S. (1998). Effect of object-oriented programming expertise in several dimensions of comprehension strategies. *Program Comprehension, Workshop Proceedings*(July), 82–89. doi: 10.1109/wpc.1998.693294

Burkhardt, JM., Détienne, F. & Wiedenbeck, S. E. S. E. (2002). Object-Oriented Program Comprehension: Effect of Expertise, Task and Phase. *Empirical Software Engineering*, *7*, 115.

Cant, S. N., Jeffery, D. R., & Henderson-Sellers, B. (1995). A conceptual model of cognitive complexity of elements of the programming process. *Information and Software Technology*, *37*(7), 351–362. doi: 10.1016/0950-5849(95)91491-H

Dunsmore, A., Roper, M., & Wood, M. (2000). Role of comprehension in software inspection. *Journal of Systems and Software*, *52*(2), 121–129. doi: 10.1016/S0164-1212(99)00138-7

Gutwin, C., Penner, R., & Schneider, K. (2004). Group awareness in distributed software development. In *Proceedings of the acm conference on computer supported cooperative work, cscw* (pp. 72–81). New York, New York, USA: ACM Press. Retrieved from `http://portal.acm.org/citation.cfm?doid=1031607.1031621` doi: 10.1145/1031607.1031621

Helgesson, D., Engstrom, E., Runeson, P., & Bjarnason, E. (2019). Cognitive load drivers in large scale software development. *Proceedings - 2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2019*, 91–94. doi: 10.1109/CHASE.2019.00030

Herbsleb, J. D. (1999). Metaphorical representation in collaborative software engineering. *Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration, WACC 1999*, 117–126. doi: 10.1145/295665.295679

Hollan, J., Hutchins, E., & Kirsh, D. (2000, jun). Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research. *ACM Transactions on Computer-Human Interaction*, *7*(2), 174–196. Retrieved from `http://dl.acm.org/doi/10.1145/353485.353487` doi: 10.1145/353485.353487

Karahasanović, A., Levine, A. K., & Thomas, R. (2007). Comprehension strategies and difficulties in maintaining object-oriented systems: An explorative study. *Journal of Systems and Software*, *80*(9), 1541–1559. doi: 10.1016/j.jss.2006.10.041

Ko, A. J., Deline, R., & Venolia, G. (2007). *Information N eeds in Collocated Software Dev elopment Teams* (Tech. Rep.).

Lavallee, M., Robillard, P., & Paul, S. (2013). Distributed Cognition in Software Engineering. *eKNOW 2013, The Fifth ...* (c), 57–62. Retrieved from `http://www.thinkmind.org/index.php?view=article{\&}articleid=eknow{\_}2013{\_}4{\_}20{\_}60013`

Littman, D., Pinto, J., Letovsky, S., & Soloway, E. (1986). Mental models and software maintenance. In *In empirical studies of programmers* (pp. 80–98). Ablex Pub. Corp.

Minelli, R., Mocci, A., & Lanza, M. (2015). I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. *IEEE International Conference on Program Comprehension*, *2015-Augus*, 25–35. doi: 10.1109/ICPC.2015.12

Omoronyia, I., Ferguson, J., Roper, M., & Wood, M. (2009). Using developer activity data to enhance awareness during collaborative software development. *Computer Supported Cooperative Work*, *18*(5-6), 509–558. doi: 10.1007/s10606-009-9104-0

Petre, M. (2010). Mental imagery and software visualization in high-performance software development teams. *Journal of Visual Languages and Computing*, *21*(3), 171–183. Retrieved from `http://dx.doi.org/10.1016/j.jvlc.2009.11.001` doi: 10.1016/j.jvlc.2009.11.001

Renumol, V. G., Janakiram, D., & Jayaprakash, S. (2010). Identification of cognitive processes of effective and ineffective students during computer programming. *ACM Transactions on Computing Education*, *10*(10). Retrieved from `http://doi.acm.org/10.1145/1821996.1821998.`

doi: 10.1145/1821996.1821998

Robillard, P. N., D'Astous, P., Detienne, F., & Visser, W. (1998). Measuring cognitive activities in software engineering. *Proceedings - International Conference on Software Engineering*, 292–300. doi: 10.1109/icse.1998.671342

Shaft, T. M., & Vessey, I. (1998, jun). The Relevance of Application Domain Knowledge: Characterizing the Computer Program Comprehension Process. *Journal of Management Information Systems*, *15*(1), 51–78. Retrieved from `https://www.tandfonline.com/doi/full/10.1080/07421222.1998.11518196` doi: 10.1080/07421222.1998.11518196

Stol, K. J., Ralph, P., & Fitzgerald, B. (2016). Grounded theory in software engineering research: A critical review and guidelines. *Proceedings - International Conference on Software Engineering*, *14-22-May*-(Aug 2015), 120–131. doi: 10.1145/2884781.2884833

Storey, M. A., Fracchia, F. D., & Müller, H. A. (1999). Cognitive design elements to support the construction of a mental model during software exploration. *Journal of Systems and Software*, *44*(3), 171–185. doi: 10.1016/S0164-1212(98)10055-9

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. *ACM International Conference Proceeding Series*. doi: 10.1145/2601248.2601268

Zhuge, H., Ma, J., & Shi, X. (1997). Abstraction and analogy in cognitive space: A software process model. *Information and Software Technology*, *39*(7), 463–468. doi: 10.1016/S0950-5849(96)00008-0