Exploring the Iterative Nature of Corrective SQL Maintenance: An Eye Tracking Study

Conference Paper · December 2014

DOI: 10.13140/2.1.4362.4008

4 authors, including:

Binny M. Samuel
University of Cincinnati
19 PUBLICATIONS **30** CITATIONS
SEE PROFILE

Lingyao Yuan
Iowa State University
8 PUBLICATIONS **51** CITATIONS
SEE PROFILE

Exploring the Iterative Nature of Corrective SQL Maintenance: An Eye Tracking Study

Completed Research Paper

Randy Minas (rminas@hawaii.edu), University of Hawaii
Binny Samuel (bsamuel@ivey.uwo.ca), Ivey Business School
Lingyao Yuan (yuanl@indiana.edu) and Vijay Khatri (vkhatri@indiana.edu), Kelley School of Business

Introduction

Software maintenance, a critical component of the software development process (Dennis et al. 2014), is defined as a set of problem-solving tasks which keep systems operational (Swanson 1976). Early studies on software maintenance estimated that as much as 40 to 60 percent of total system and programming resources are spent in the software maintenance stage (Lientz et al. 1978). More recent estimates place the time and cost of software maintenance at nearly 80 percent of the total software development process (Hatton 1998). While previous research has sought to understand cognitive underpinnings of software maintenance, few studies have attempted to use the tools of cognitive neuroscience to understand the problem solving process during software maintenance.

Prior research in maintenance has focused primarily on maintenance of third-generation procedural programming languages, e.g., COBOL (Shaft and Vessey 2006). SQL (structured query language), a fourth-generation declarative programming language is the lingua franca for relational DBMSs (database management systems), which according to IDC were worth a little over $24 billion worldwide in 2012.[1] Unlike program maintenance, SQL maintenance requires smaller edits in the SQL code but often requires an analyst to integrate multiple artifacts, for example, the SQL code, the logical schema and the data dictionary.

Problem solving in general (Newell et al. 1958) and technical problem solving in particular has trial-and-error as a prominent feature (Allen 1966). Problem solving is often iterative in nature, and prior research indicates that there are differences in the processes individuals undertake in solving problems (Brand-Gruwel et al. 2005; Newell et al. 1958). Research has also shown that there are differences resulting from expertise on problem solving, with high performers spending more time in the initial phase, for example, the problem definition, than low performers (Brand-Gruwel et al. 2005). Brooks (1983) suggests that mental models are iteratively refined from an application to a specific code in the program. However, prior research in query composition suggests a sequential three-stage cognitive model that proceeds from query formulation to query translation and culminates with query writing (Ogden 1985). In the context of SQL maintenance, we anticipate that these stages would intertwine and be iterative in nature. To date there has been little theoretical development surrounding the iterative nature of problem solving in the SQL maintenance process.

This study seeks to make a contribution to software maintenance by clarifying search processes that professional data analysts undertake to debug SQL code. Many different tasks are associated with maintenance: adapting to new needs (50% of maintenance projects (Pigoski 1996)),

[1] http://itknowledgeexchange.techtarget.com/eye-on-oracle/oracle-the-clear-leader-in-24-billion-rdbms-market/

adapting to changing environment (25%) and correcting errors, either preventively (5%) or as a result of actual problem (20%) (von Mayrhauser and Vans 1995); we focus on corrective tasks, which involve understanding the existing code as well as the problem, repairing the code and finally conducting tests on the repaired code. As such the research question that we explore is: What is the information search process that is employed by professional data analysts in conducting corrective SQL tasks?

The rest of the paper is organized as follows. We present the cognitive theoretical foundations for this research. We then present the research methodology. In Section 4 we present our findings. We discuss our findings in Section 5 and round out the research in Section 6.

Theoretical Foundation

Early database query languages were based on Codd's relational algebra, e.g., PRTV (Peterlee Relational Test Vehicle) (Jarke and Vassiliou 1985). Very quickly, Quel and SQL were adopted, with SQL now prevalent. Prior studies indicate that complexity in a SQL query is related to joins as well as confusion associated with the WHERE, GROUP BY, and HAVING clauses (Chan et al. 1993). Considering confusion associated with the WHERE, GROUP BY and HAVING clause (Chan et al. 1993), we anticipate that query complexity will play a role in the query maintenance process (Allen and Parsons 2010; Bowen et al. 2006; Bowen et al. 2009).

Proposition 1: Performance on the task of SQL query correction will be contingent on the complexity of the corrective SQL task.

Ogden (1985) proposed a sequential three-stage cognitive model of database query composition: 1) *query formulation*, in which the end users articulate the data they need; 2) *query translation*, in which data analysts take the end users' requirements as reflected in the representational diagrams; and 3) *query writing*, in which the data analysts arrange the output from the previous stage into the format required by the query language. We integrate this perspective with prior literature on software maintenance that delineates three stages of maintenance: comprehension, modification, and verification (Shaft and Vessey 2006).

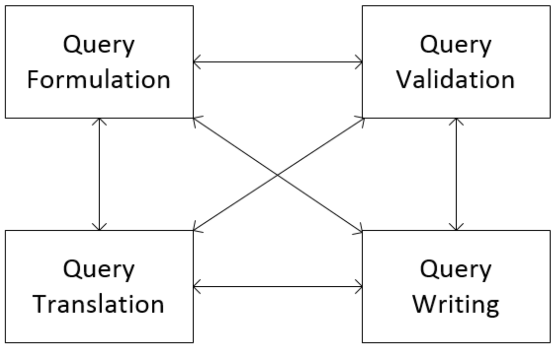


Figure 1: An iterative conceptual model of the query corrective process

Our *query formulation* and *query translation* stages overlaps with the software maintenance comprehension stage as it entails understanding the query maintenance task. Similarly, our *query writing* stage refers to the modification stage that involves changing, deleting from, or adding to the SQL code. Finally, in the *query verification* stage, which is closely tied to software quality (Adrion et al. 1982), a programmer checks the SQL code for errors or omissions by testing to ensure that the SQL code operates as intended (Adrion et al. 1982). Therefore, in order to gain a holistic picture of the corrective query maintenance process, it is not only important to understand the processes data analysts undertake in each of the stages of maintaining a SQL query but also how these stages intertwine with each other.

Since Ogden's cognitive model treats problem-solving as having distinct sequential steps, it does not account for the fact that the software maintenance process is often iterative in nature (Newell et al. 1958). Our conceptualization of a corrective query task involves the problem solvers moving between four stages in a non-sequential and iterative manner (see Figure 1).

Proposition 2: The task of query correction will be iterative in nature between the stages of query formulation, query translation, query writing, and query validation.

Finally, we expect an interaction between query complexity and the process that problem solvers will undertake in query correction. Research has often focused on differences between expert and novice programmers, in that experts employ strong problem-solving strategies adaptable to the complexity of the task at hand (Mayer 1992). However, some research has indicated that even experts have divergent abilities, especially as the task complexity increases (Pennington 1982; Vessey 1985). We conceive of corrective query problem solving as composed of two phases, early and late (Brand-Gruwel et al. 2005). We expect that high performers will undertake a majority of transitions between query formulation, query translation, and query writing stages during the early phase of problem-solving, and between query writing and query validation during later phase. Transitions among the query formulation, query translation, and query writing stages during the early phase of problem-solving allows for broad strategizing to understand the problem space and limit the potential solutions or generate elements of the solution (Mayer 1997), which has been linked to discovering the solution more efficiently (Newell et al. 1958); however in the later phase of problem-solving an expert should not need to spend effort, as they did during the early phase of the problem-solving process, forming an appropriate mental model of the solution (Brooks 1983). Instead, the later phase of problem-solving should be spent transitioning between writing and validation to verify/perform small corrections to the formulated solution. Thus, we conceptualize the aforementioned transitions as effective in the early (see Figure 2) and late phases (Figure 3) of corrective query problem solving. We also expect expertise to be manifested in *both* early as well as late phases of problem solving.

Proposition 3: As query complexity increases, the degree of a problem solver's expertise will be reflected in different phases – early and late – of the corrective SQL task.

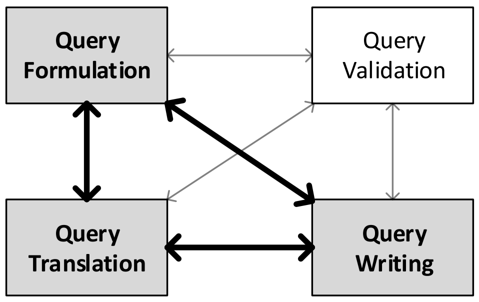


Figure 2: Effective Transition in the Early Phase

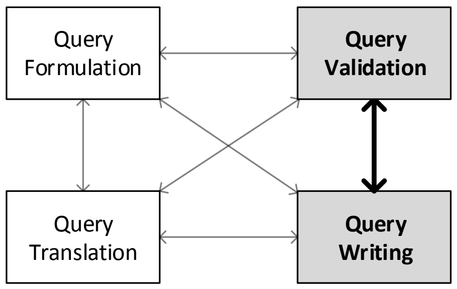


Figure 3: Effective Transitions in the Late Phase

Research Methodology

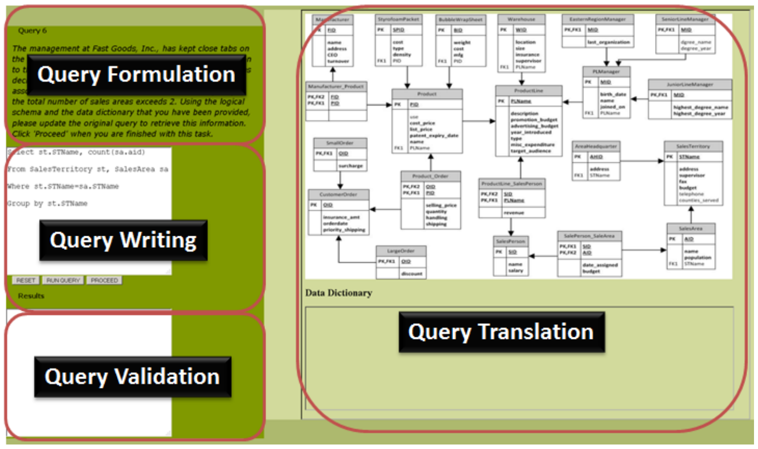
We examined our research question using a lab experiment with eye tracking data of professional data analysts responding to corrective SQL tasks, which involved both simple and complex queries.

Participants

The participants for this study were 8 professional data analysts who had several years of work experience with databases and degrees from computer science, informatics, or MIS fields. Each analyst reported at least five years of experience in writing SQL and over 10 years of total work experience.

Experiment Design

We used a 2 x 1 design with the complexity of the SQL task as the within-subject factor. Both the tasks employed a sales domain as the context for our tasks. Subjects completed the tasks using a web interface as shown in Figure 4. The interface provided: 1) The *query formulation* section with directions for the analysts that allowed them to identify the requirements of the query; 2) The *query* translation section provided the analysts with a logical schema and data dictionary to understand end users' data; 3) The *query writing* section allowed the analysts to correct a problematic SQL query; and 4) *query validation* section allowed the analysts to see if the query that was written was providing the desired results; it connected to the database server.


Query 6

The management at Fast Goods, Inc., has kept close tabs on
the total number of sales areas exceeds 2. Using the logical schema and the data dictionary that you have been provided, please update the original query to retrieve this information. Click 'Proceed' when you are finished with this task.

```
Select st.STName, count(sa.aid)
From SalesTerritory st, SalesArea sa
Where st.STName=sa.STName
Group by st.STName
```

Figure 4: Web interface used by participants during experiment

Tasks

Two queries were used to operationalize simple and complex corrective SQL tasks.

	Simple Query	**Complex Query**
Initial Query (Query Writing of Figure 4)	Select supervisor From SalesTerritory Where STName= 'CCA';	Select s.name, sum(ps.revenue) From SalesPerson s, ProductLine_SalesPerson ps Where s.SID=ps.SID Group By s.name;
Query Correction Task (Query Formulation of Figure 4)	Besides monitoring the supervisor for the CCA sales territory, Fast Goods, Inc., would like to have better visibility over the sales territories for the entire management team. Thus, they have decided that the report should show the budget and supervisor for every sales territory. Using the logical schema and the data dictionary that you have been provided, please update the original query to retrieve this information.	The management at Fast Goods, Inc., have closely watched the revenue generated by sales persons via a report. Recently, they have decided that they report should display the sales person name, product line name, description, and total revenue for product lines that are associated with a sales person. Furthermore, the report should only display sales persons associated with product lines that have total revenue of more than $500,000. Using the logical schema and the data dictionary that you have been provided, please update the original query to retrieve this information.
Query Solution	Select supervisor, budget From SalesTerritory;	Select s.name, p.PLName, p.description, sum(ps.revenue) From SalesPerson s, ProductLine p, ProductLine_SalesPerson ps Where s.SID=ps.SID and p.PLName=ps.PLName Group By s.name, p.PLName, p.description Having sum(ps.revenue) >500000;

Table 1: Simple and Complex Query Tasks

Table 1 shows the initial query that was populated in the "query writing" portion of the web interface, the modification task that was shown on the "query formulation" section of the web interface. Table 1 also shows the final query solution for both simple and complex queries. Consistent with prior research, the complex query required the use of a JOIN, GROUP BY, HAVING, and aggregate function in the SELECT clause, whereas the simple query only required modification to the WHERE and SELECT clauses of SQL (Bowen et al. 2009; Chan et al. 1993).

Eye-tracking

To understand processes that were employed by professional analysts in conducting corrective SQL tasks, we employed eye tracking. Eye tracking is a psychophysiological recording technique that offers a unique ability to track where an individual is directing his or her visual attention (Stern et al. 2001). Eye tracking involves the use of a camera or a set of cameras to capture the movement and fixations of an individual's eyes. Eye movement is characterized by both saccadic eye movements and fixations (Stern et al. 2001). *Saccadic eye movements* are high velocity movements of the eye towards a target, followed by deceleration in velocity once the eye reaches the target (Stern et al. 2001). During *fixation*, the eye becomes fixed on a target for some period of time, generally on the order of hundreds of milliseconds (Just and Carpenter 1980). In eye tracking recordings, the eye position, the direction of gaze, and the eye movements are measured (Stern et al. 2001). By integrating these measurements, a researcher can disentangle saccadic eye movements from fixations, thereby elucidating where an individual is directing his or her visual attention (Desimone and Duncan 1995). These measures can aide in understanding the process by which individuals traverse a given landscape.

Variables of Interest

The primary outcome measures we examined were: 1) the number of errors committed (#Errors); 2) the amount of time taken (Time); and 3) the number of iterations (#Iterations). Our independent variables were how the subjects undertook problem solving, including how the subject's eyes traversed our defined areas of interest, fixations at different areas of interest, and dwell time among the areas of interest. These data enabled us to assess our propositions and address our research question.

Findings

In the following, we first present our findings related to the effectiveness of corrective SQL performance and then our findings related to the efficiency in the information search process.

Performance Effectiveness

We first present the performance in terms of the total number of errors each subject made during the tasks (see Table 2). The solution of each task was analyzed by the researchers to identify #Errors across all iterations. #Errors indicates the total number of errors made by a subject for a given task. With respect to #Errors, Table 2 indicates that in general the subjects made more errors in the complex than simple task. Based on the absolute change in the number of errors that each subject made on the simple and complex queries, we classified them as low and high performers. We construe subjects with greater or less than the median score (5) for | Change in

#Errors | as high and low performing, respectively. Thus, H-1, H-2, H-3, and H-4 are referred to as high performing subjects; the others are referred to as low performing.

Subject	#Errors		| Change in #Errors |
	Simple Query	Complex Query	
H-1	1	3	2
H-2	2	4	2
H-3	2	1	1
H-4	0	3	3
L-1	7	14	7
L-2	2	12	10
L-3	1	65	64
L-4	0	9	9

Table 2: Overall Performance Effectiveness

Performance Efficiency

For each subject, we also examined the efficiency of their problem solving in terms of the time taken as well as the number of iterations in solving the problem. We found that in general all of our subjects took more time to solve the complex query than the simple query, with subject L-1 being an exception. We also found that in general the #Iterations[2] for the high performing subjects did not substantially change as for the low performing subjects.

Subject	Simple Query		Complex Query	
	#Iterations	Time[a]	#Iterations	Time[a]
H-1	2	66.52	3	229.62
H-2	2	96.16	3	225.65
H-3	2	95.27	1	229.84
H-4	1	42.56	3	201.17
L-1	3	301.99	6	252.94
L-2	3	129.98	8	665.95
L-3	1	207.45	12	229.16
L-4	1	48.49	5	176.04

[a]Time is shown in seconds

Table 3: Overall Performance Efficiency

To analyze how our respondents performed across multiple iterations, we segregated early and late problem solving phases as: 1) first; and 2) subsequent iterations. We next normalized the number of errors the subjects made in their subsequent iterations. The normalized errors were computed by dividing the number of errors made in the subsequent iterations by the total number of subsequent iterations. The results of this analysis can be seen in Table 4. Our findings suggest that while our high and low performing subjects had similar error scores in the first phase, their scores were divergent in the subsequent phase.

Table 4 also provides some insights into how performance proceeded over first vs. subsequent iterations. First, we highlight that there were not too many differences in simple query performance. However, the complex query does begin to delineate individuals who performed better. Specifically, it appears the high performing subjects, H-1, H-2, H-3, and H-4, committed less than one normalized error in the subsequent iteration. Comparatively, the low performing

[2]Subjects could click the "Run Query" button in the Query Writing section of the web interface (see Figure 4) at any point to see the results of their current SQL code. We only counted an iteration if the query writing portion of the web interface had changed since the last time the "Run Query" button had been clicked. This also meant that if a subject chose to start their problem-solving by observing the results of the initial query (see Table 1) then this was not counted as iteration either. Only capturing modifications to the query writing allowed us a more discernable view in the actual trial and error process subjects used during problem solving.

subjects, L-1, L-2, L-3, and L-4, committed more normalized errors in the subsequent iteration. To better understand the differences in performance, we next turned to the eye tracking data to evaluate the process that subjects in these groups undertook.

In order to understand the search process that subjects undertook during problem solving, we examined their eye-tracking data to analyze their movement between the query formulation, query translation, query writing, and query validation sections of the web interface as shown in Figure 4. We were specifically interested in each subject's saccadic eye movements between these areas over time (i.e., as they were problem-solving). We created transition graphs for each subject to display where the subjects were looking on the screen at each point in time. A transition graph placed time (in milliseconds) on the X-axis, and the section of the screen that was viewed on the Y-axis. We placed a red vertical line at the point in time when the subject completed their first iteration, and a green (grey) vertical line to show that the subject had moved to the next task. Prior research has shown the searching for information is a proxy for information processing (Hungerford et al. 2004) and the transition graphs enabled us to observe what type of information processing subjects were undertaking in the first and subsequent iterations. We first established a coding scheme consistent with the stages of a corrective query task conceptualized in this research and then display the results visually. Due to space limitations, we show a simple and complex transition graph for one subject each from the high (i.e., H-1 in Figure 5 & Figure 7) and low (i.e., L-1 in Figure 6 & Figure 8) performing group.

	Simple Query		Complex Query	
Subject	First	Subsequent (Normalized)	First	Subsequent (Normalized)
H-1	1	0 (0)	2	1 (0.5)
H-2	1	1 (1)	3	1 (0.5)
H-3	1	1 (1)	1	-
H-4	0	-	2	1 (0.5)
L-1	1	6 (3)	3	11 (2.2)
L-2	1	1 (0.5)	1	11 (1.6)
L-3	1	-[a]	7	58 (5.3)
L-4	0	-	4	5 (1.3)

[a]The dash symbol (i.e., -) indicates that the subject did not have a subsequent iteration

Table 4: Error Performance – Number of Errors Normalized by Iterations

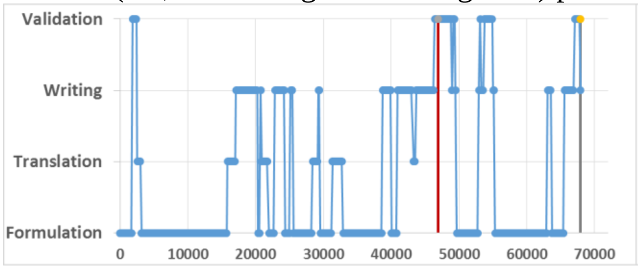


Figure 5: Transition Graph for a Subject from High Performing Group (H-1) for the Simple Query

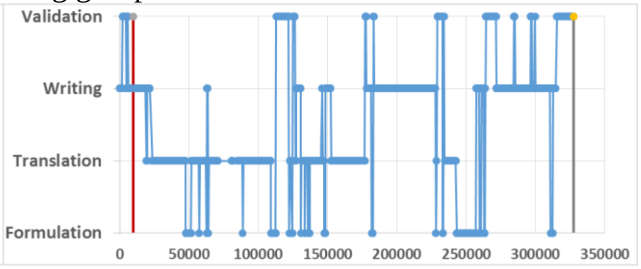


Figure 6: Transition Graph for a Subject from Low Performing Group (L-1) for the Simple Query

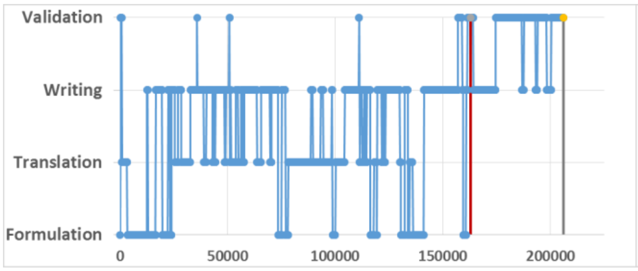


Figure 7: Transition Graph for a Subject from High Performing Group (H-1) for the Complex Query

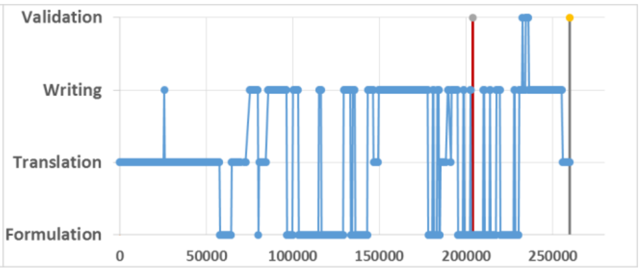


Figure 8: Transition Graph for a Subject from Low Performing Group (L-1) for the Complex Query

Figure 5 and Figure 6 indicate that the two representative subjects undertook different processes during problem solving. Subject H-1 (Figure 5) started with query formulation at time zero and made several transitions that were primarily between query writing and query formulation during the first iteration. After the first iteration, H-1 transitioned between query validation, query writing, and query formulation. We also notice that their first iteration lasted for over half of their problem-solving effort (see the red line). Subject L-1 (Figure 6) transitioned between query writing and query validation during their first iteration, which accounted for a small portion of their total problem-solving effort. L-1's subseqent iterations moved between all four phases and overall they made more transitions than H-1. In general, we found no discernable pattern that distinguished the high and low performer for the simple query. Despite these differences in the search process undertaken by our subjects, it seems that there are many viable search strategies for simple corrective query tasks.

Using Figure 7 and Figure 8, we compared the processes of the subjects in the high performing group vs. those in the low performing group for the complex task. It seems that the high performer (H-1 in Figure 7) placed more emphasis on transitions among query formulation, query translation, and query writing during the first iteration, and between query writing and query formulation in the subsequent iterations. However, low and high performers did not necessarily spend the majority of their transitions in first and subsequent transitions in a similar fashion. For example, L-1 in Figure 8 did emphasize transitions between query formulation, query translation, and query writing during the first iteration, but continued to make those areas of interest a priority in subsequent iterations.

We present a summary of the transitions that occurred for all subjects during the first and subsequent iterations (See Table 5). Effective transitions during the first iteration were construed to be the percentage of all transitions between: 1) Formulation – Translation; 2) Formulation – Writing; and 3) Translation – Writing, and ineffective were construed as transitions between Writing – Validation. The converse applies for subsequent iterations. Table 5 shows that for the simple query there were a variety of effective vs. ineffective transitions during the first and subsequent iterations, but as we pointed out earlier, this did not impair performance. However, for the complex query our findings indicate that the high performing group transitioned effectively the majority of the time in both the first and subsequent iterations.

	Simple Query				Complex Query			
	First		Subsequent		First		Subsequent	
Subject	Effective	Ineffective	Effective	Ineffective	Effective	Ineffective	Effective	Ineffective
H-1	87.5%	4.2%	56.3%	43.8%	85.3%	9.3%	100.0%	0.0%
H-2	100.0%	0.0%	21.4%	78.6%	89.2%	9.2%	52.9%	41.2%
H-3	100.0%	0.0%	0.0%	100.0%	91.7%	8.3%	-[a]	-
H-4	80.0%	13.3%	0.0%	80.0%	86.8%	13.2%	83.3%	16.7%
L-1	0.0%	100.0%	0.0%	100.0%	100.0%	0.0%	22.2%	77.8%
L-2	66.7%	33.3%	27.3%	63.6%	61.3%	30.1%	55.9%	39.2%
L-3	100.0%	0.0%	-[a]	-	80.0%	10.0%	30.9%	60.0%
L-4	76.9%	23.1%	-	-	88.9%	8.3%	43.5%	52.2%

[a]The dash symbol (i.e., -) indicates that the subject did not have a subsequent iteration

Table 5: Percentage of Effective Transitions during Simple and Complex Query

We also note that two of these subjects (i.e., H-2 and H-3) even altered their problem-solving process between simple and complex queries. Although they did not have effective transitions as their major emphasis in both the first and subsequent iterations of the simply query, they did make this shift for the complex query. On the other hand, two subjects (i.e., L-3 and L-4) in the low performing group made effective transitions in their first iteration for the simple query, but emphasized ineffective transitions in their subsequent iterations in solving the complex query.

Discussion

We summarize the findings of this current research. First, consistent with prior research on problem-solving, we conceptualized query correction as an iterative process between four stages. Further, we differentiate between two phases of corrective query problem solving which we refer to as the first and subsequent iteration. While prior research on query composition suggests a linear process that excludes query validation, our findings indicate that query correction is actually a complicated non-sequential and iterative process that entails shifts between query formulation, query translation, query writing, and query validation over multiple iterations.

Second, while query composition requires an analyst to integrate multiple artifacts, for example, the SQL code, the logical schema and the data dictionary, their significance varies in the early vs. late phases of query maintenance. We found that especially for complex tasks, carefully examining and studying these artifacts was critical for query performance. Although these artifacts *could* be referenced at any time during the problem-solving process, this current research suggests that they be carefully considered during the early phase of the corrective query maintenance task.

Third, our findings suggest that expert behavior has a more pronounced effect on performance during complex tasks as opposed to simple tasks. Table 5 suggests that while there were a variety of behaviors (i.e., search processes) that all of our analysts undertook for simple tasks (i.e., portion of effective vs. ineffective transitions), their behavior had little impact on performance (see Table 4). However, only our high performing data analysts employed effective transitions in both the first and subsequent iterations in solving the complex query. Thus, although expertise may be within the grasp of an individual, sometimes they may not engage in behavior that will result in better performance during problem-solving.

Conclusion

Software maintenance is a critical component of the software development process. This research employed eye-tracking to understand the cognitive underpinnings during a corrective SQL query task. Our findings indicate that SQL maintenance is an iterative process that transitions between formulation, translation, writing, and validation. Furthermore, when tasks become complex, how an analyst approaches the problem is critical in both the early and late phase of the task. Future research can help further elucidate the criticality of the early phase of problem-solving and how different approaches to it impact the late phase SQL maintenance.

References

Adrion, W. R., Branstad, M. A., and Cherniavsky, J. C. "Validation, Verification, and Testing of Computer Software," *ACM Comput. Surv.* (14:2) 1982, pp 159-192.

Allen, G. and Parsons, J. "Is query reuse potentially harmful? Anchoring and adjustment in adapting existing database queries," *Information Systems Research* (21:1) 2010, pp 56-77.

Allen, T. J. "Studies of the problem-solving process in engineering design," *IEEE Transactions on Engineering Management* (13:2) 1966, pp 72-83.

Bowen, P. L., O'Farrell, R. A., and Rohde, F. H. "Analysis of competing data structures: Does ontological clarity produce better end user query performance," *Journal of the Association for Information Systems* (7:8) 2006, p 22.

Bowen, P. L., O'Farrell, R. A., and Rohde, F. H. "An Empirical Investigation of End-User Query Development: The Effects of Improved Model Expressiveness vs. Complexity," *Information Systems Research* (20:4), Dec 2009, pp 565-584.

Brand-Gruwel, S., Wopereis, I., and Vermetten, Y. "Information problem solving by experts and novices: analysis of a complex cognitive skill," *Computers in Human Behavior* (21:3), 5// 2005, pp 487-508.

Brooks, R. "Towards a theory of the comprehension of computer programs," *International Journal of Man-Machine Studies* (18) 1983, pp 543-554.

Chan, H. C., Wei, K. K., and Siau, K. L. "User-Database Interface: The Effect of Abstraction Level on Query Performance: A Field Experiment " *MIS Quarterly* (17:4) 1993, pp 441-464.

Dennis, A. R., Samuel, B. M., and McNamara, K. "Design for maintenance: how KMS document linking decisions affect maintenance effort and use," *Journal of Information Technology*) 2014.

Desimone, R. and Duncan, J. "Neural Mechanisms of Selective Visual Attention," *Annual Review of Neuroscience* (18:1) 1995, pp 193-222.

Hatton, L. "Does OO sync with how we think?," *Software, IEEE* (15:3) 1998, pp 46-54.

Hungerford, B., Hevner, A., and Collins, R. "Reviewing Software Diagrams: A Cognitive Study," *IEEE Transactions on Software Engineering* (30:2) 2004, pp 82-96.

Jarke, M. and Vassiliou, Y. "A Framework for Choosing a Database Query Language " *ACM Computing Surveys* (17) 1985, pp 313–340.

Just, M. A. and Carpenter, P. A. "A theory of reading: From eye fixations to comprehension," *Psychological Review* (87:4) 1980, pp 329-354.

Lientz, B. P., Swanson, E. B., and Tompkins, G. E. "Characteristics of application software maintenance," *Commun. ACM* (21:6) 1978, pp 466-471.

Mayer, R. E. *Thinking, Problem Solving, Cognition*, (Second Edition ed.) W.H. Freeman and Company, New York, NY, 1992, p. 560.

Mayer, R. E. "From Novice to Expert " in: *Handbook of Human-Computer Interaction* M. Helander, T. Landauer and P. Prabbu (eds.), Elsevier Science London, 1997, pp. 781-795.

Newell, A., Shaw, J. C., and Simon, H. A. "Elements of a theory of human problem solving," *Psychological Review* (65:3) 1958, p 151.

Ogden, W. C. "Implications of a cognitive model of database query: Comparison of a natural language, a formal language, and direct manipulation interface " *ACM SIGCHI Bulletin* (18:2) 1985, pp 51–54.

Pennington, N. *Cognitive components of expertise in computer programming: A review of the literature* Center for Decision Research, Graduate School of Business, the University of Chicago, 1982.

Pigoski, T. M. *Practical Software Maintenance: Best Practices for Software Investment* John Wiley & Sons, Inc., 1996.

Shaft, T. M. and Vessey, I. "The Role of Cognitive Fit in the Relationship between Software Comprehension and Modification," *MIS Quarterly* (30:1) 2006, pp 29-55.

Stern, R. M., Ray, W. J., and Quigley, K. S. *Psychophysiological Recording*, (2nd Edition ed.) Oxford University Press, New York, 2001, p. 282.

Swanson, E. B. "The dimensions of maintenance," in: *Proceedings of the 2nd international conference on Software engineering*, IEEE Computer Society Press, San Francisco, California, United States, 1976, pp. 492-497.

Vessey, I. "Expertise in debugging computer programs: A process analysis," *International Journal of Man-Machine Studies* (23:5), 11// 1985, pp 459-494.

von Mayrhauser, A. and Vans, A. M. "Program comprehension during software maintenance and evolution," *Computer* (28:8) 1995, pp 44-55.