Extended Online Appendix: Summary of Literature

Summary of literature included in systematic review.

Author	Task	Method	Findings
Shneiderman (1976)	Study, hand-execution	Participants of varying levels of expertise (non-programmers to expert programmers) were given two programs to memorize: a correctly written executable program, and a program with the lines of code randomly reordered. Participants were asked to rewrite the programs verbatim. Novice and experienced programmers were given programs to study and were then asked comprehension questions and to determine the output produced by the program.	Experienced programmers form chunks consisting of multiple statements and treat complex control structures as single units when encoding programs. The results of the study also indicate that experienced programmers recode the syntactic notation of the code to a high-level semantic representation.
Adelson (1981)	Study	Expert and novice programmers were shown one line of code at a time in a random order and were then asked to recall the lines of code.	Novices organize code using syntax whereas experts use an abstract hierarchical organization that is semantically based according to program function.

Continuation of Table			
Author	Task	Method	Findings
McKeithen, Reitman, Rueter, & Hirtle (1981)	Study	Beginner, intermediate, and expert programmers were shown a program with the lines of code in either normal or scrambled order and were asked to recall the program. Participants were then given a stack of cards with the programming language's reserved words which they studied and were encouraged to sort in an order that would be easy to recall. Participants were asked to recall the words for non-cued trials (recall in any order) and cued trials (asked to start with a word, continue with the words that went with it, and recall the remainder).	Experts use functional organization when chunking and their chunks are formed based on programming knowledge. Beginners' associations of common language to programming concepts varied greatly, intermediates show mixtures of programming concepts and common language associations, and experts form associations based on programming knowledge.
Weiser (1981)	Debugging	Experienced programmers were given programs to debug. After finding the bugs participants were shown fragments of algorithms and were asked if they had been used in the programs they had debugged.	Results indicate that programmers mentally construct and use program slices when debugging.
Weiser (1982)	Debugging	Experienced participants' ratings of how typical the bug was, and their debugging time were recorded. Participants were tested on their recognition of program slices relevant to the bugs.	Proposes program slices as an abstract representation of a program that can be formed using information distributed throughout the program.

Continuation of Table			
Author	Task	Method	Findings
Adelson (1984)	Study, debugging	Expert and novice participants were given either code, concrete flow chart of the code, or abstract flow chart of the code to study followed by comprehension questions. Participants were given either an abstract or concrete task (debugging) followed by comprehension questions.	Experts form abstract representation and novices form concrete representations during program comprehension.
Mynatt (1984)	Study, hand-execution	Participants were given programs that performed the same function but varied in semantic complexity to memorize. Participants were then asked to immediately recall the program and perform hand-execution, they were later asked for delayed recall.	Results indicate that semantically complex programs are harder for programmers to encode and chunk.
Soloway & Ehrlich (1984)	Study	Novice, intermediate, and advanced programmers were given code with critical lines left blank and were asked to fill in the blanks with code to complete the program.	Experts programmers use plans which are higher-level structures that allow them to chunk related lines of code. Experts' plan knowledge is more advanced than novices' as a collection of plans is developed with experience.

Continuation of Table			
Author	Task	Method	Findings
Soloway & Ehrlich (1984)	Study	Participants were given two types of programs: plan-like and unplan-like. Plan-like programs were created using a set of programming rules of discourse. Participants were given unfamiliar programs and asked to fill in the missing line of code. Participants were then asked to study a program and recall the code verbatim. Results of novices and experts were compared.	Knowledge of programming plans and programming rules of discourse has a significant effect on program comprehension.
Barfield (1986)	Study	Non-programmers, novice, intermediate, and expert programmers were asked to memorize code presented in either executable order, random lines, or random chunks. Participants recalled the program verbatim and were scored on the number of lines and chunks they recalled.	Experts use chunking when understanding code.

Continuation of Table			
Author	Task	Method	Findings
Schmidt (1986)	Study	Participants studied a meaningful program one line at a time and then attempted to recall the program verbatim. The time to read each line was recorded. Participants were given a distractor task followed by a recognition test. Participants then studied a program of random statements presented one line at a time at a predetermined rate and were asked to recall the program verbatim. Lastly, they answered comprehension questions on the meaningful program. Experience was measured based on number of computer science courses.	Experienced programmers are able to make connections between related statements of meaningful programs more quickly than novices through their ability to recognize algorithms. Supports the knowledge compilation theory and the use of recall as a measure of program comprehension.
Bateson, Alexander, & Murphy (1987)	Study, writing algorithm and code	Expert and novice participants completed four tests designed to measure: syntactic memory (verbatim recall), strategic skill (writing algorithms), tactical skill (writing programs), and semantic memory (general programming knowledge).	Introduces a cognitive processing model that illustrates the relationship between semantic and syntactic memory, and programmer skill/expertise. Semantic memory is the principal factor in determining programmer expertise.

Continuation of Table			
Author	Task	Method	Findings
Boehm-Davis, Holt, & Schultz (1987)	Modification	Expert and novice programmers were tasked with performing either a complex or simple modification to programs written in the same programming language, but using three types of problems, and with three different structures: functional, object oriented, and in-line code. After each modification, participants were asked to recall segments of the code and any relationships between the segments. Using the recalled segments and relationships written on cards, participants were asked to create a structure using the cards. The number of segments and relationships recalled, and the depth and width of the structure were analyzed.	Mental representations of experts are not affected by the surface structure of the code and the content of the program, whereas novices are affected by these aspects of the code. Experts' mental representations are affected by the difficulty of the task: more difficult tasks result in more complex representations. The complexity of the mental representations developed by novices are not affected by the difficulty of the task.

Continuation of Table			
Author	Task	Method	Findings
Letovsky (1987)	Modification	Verbal protocol analysis was performed on professional programmers who were asked to perform a maintenance task on an existing program. The expertise of programmers ranged from expert level to junior level.	Presents a knowledge-based understanding model. Programmers use their knowledge base to form mental models that evolve during the comprehension process as programmers assimilate the program (code and documentation) and their knowledge base. Programmers use an opportunistic approach (switch between top-down and bottom-up strategies) to form mental representations that include: specification (goals), implementations (actions), and annotations (how goals are accomplished).

Continuation of Table			
Author	Task	Method	Findings
Littman, Pinto, Letovsky, & Soloway (1987)	Maintenance	Verbal protocol analysis was performed on experienced programmers who were asked to perform a maintenance task on an existing program. The study also recorded if each participant was successful or unsuccessful at completing the task.	Finding suggest that two strategies are used for program understanding: systematic (symbolic execution used to trace data flow) and as-needed (focus on local components required for task). Two types of knowledge result from the use of these strategies: static (knowledge of objects, actions, and functional components), and casual (knowledge of interactions between functional components). Programmers that use a systematic strategy develop strong mental models consisting of both static and casual knowledge. Programmers that use an as-needed strategy develop weak mental models consisting only of static knowledge.
Pennington (1987b)	Study, modification	Expert participants answered comprehension questions and completed a recognition test after the study for understanding task. Participants also wrote a summary and answered comprehension questions after studying to prepare for a modification task, and after performing a modification task. Comprehension questions were related to control flow, data flow, function, and program state.	The control flow structures of a program (text structure knowledge) are used initially to construct mental representations of programs during the study task. Data flow and functional representations (plan knowledge) form a situational model that is developed later given more time and an appropriate task.

Continuation of Table			
Author	Task	Method	Findings
Vessey (1987)	Study, writing code	After studying a program, expert and novice participants were asked to reproduce a functionally equivalent program (results in the same output). The programs differed in how well the program structure matched the structures used in programming texts. Participants then wrote their own routines to perform a specified function.	Supports previous findings that experts outperform novices on recall tasks. Research found that knowledge structures used by programmers are not based on standardized scripts, instead, programmers were found to have great variation in their knowledge structures. Findings do not support the use of debugging tasks to determine programmers' knowledge structures.
Détienne (1988)	Study	Expert participants studied programs with either meaningful or non-meaningful procedure names, one line at a time. Lines were revealed in two ways: predetermined order and order requested by the participant. Verbal protocol analysis was performed on participants' responses after each line.	Program comprehension requires comprehension of the program and application domains. Schemas are activated top-down when programmers use "signposts" that exist in the code, and bottom-up when the algorithm is unfamiliar. Programmers adapt their existing schema using control and dataflow of the program, and executing the program mentally.

Continuation of Table			
Author	Task	Method	Findings
Gilmore & Green (1988)	Debugging	Expert participants were given the program specification for the problem and a sample of a correct program to study. Participants were then given programs with bugs that met the same specifications as the correct program, but with different structural formats (plan structure, control flow structure). Participants were asked to locate and describe the bugs in the program. The error detection rate of participants was measured.	Results of the study indicate that experts use plan structures that provide a surface level representation of a program. Programming plans are language specific and are used to map problem solving knowledge to implementation of the solution in a programming language.
Vihmalo & Vihmalo (1988)	Study, modification	At timed intervals during think-aloud study task, novice and expert participants gave descriptions of the program. To assess their understanding participants were then asked to: describe from memory what the program does and how it functions, write a portion of the program from memory, and modify the program.	Introduces a compensatory comprehension strategy that is used by expert programmers to compensate for lack of programming language knowledge. The strategy involves reliance on knowledge about the program's application domain and the program type. Results indicate the importance of programming knowledge organization in program comprehension.

Continuation of Table			
Author	Task	Method	Findings
Davies (1990b)	Reconstructing	Novice, intermediate, and expert programmers were given a program with a fragment missing and were then asked to select from a set of program fragments as quickly as possible the fragment that completes the program. One set of program fragments either used typical plan structures or violated plan structures, and another set of program fragments either followed program discourse rules or violated them.	Expert and intermediate programmers both use plan structures in program comprehension, however intermediate programmers are not able to access these plans as easily as experts. Novices do not possess plan structure knowledge. Experts use program discourse rules during program comprehension, whereas intermediate and novice programmers do not.
Davies (1990a)	Debugging	Programmers that all had similar programming experience and either had program design experience or not, were asked to locate and correct bugs in programs. The programs contained bugs related to either: control structure, plan structure, or unrelated to any structure. Cues were used to highlight either the control structure or plan structure, or no cues were provided. Programmers with similar programming experience but either had design experience or not, were shown programs written in either a plan or unplan-like way and were asked to recall the program verbatim.	Programmers with design experience use cues related to plan structures to detect plan related bugs and recall more plan structures. Results indicate that programming plans are used in program comprehension by programmers trained in program design and are not necessarily a characteristic of programming expertise.

Continuation of Table			
Author	Task	Method	Findings
Détienne & Soloway (1990)	Study	Experienced programmers were given plan-like and unplan-like versions of programs with blank lines. Participants were asked to think-aloud while completing the task of filling in the blanks with code to complete the program.	Experts develop two types of representations during program comprehension: goals and plans, and data flow.
Guerin & Matthews (1990)	Study	Measured comprehension and recall of novice and expert participants after studying programs that varied in one of the following ways: the order of lines of code and modules, semantic complexity, or substitution of code with keywords. Comprehension was measured by the participant's description of the program function and operation.	Experts rely more heavily on program functions for program comprehension, compared to novices. Supports the theory that experts use chunking as a comprehension strategy.
Robertson & Yu (1990)	Study, classify	Fortran and Pascal programmers were given programs written in the language coinciding with their background. Participants drew lines in the code to divide it into its different major sections and gave each section a descriptive label. Participants then divide each major section into subsections. Participants were then asked to sort the programs into groups that "work the same way".	Programmers can use multiple structures to represent code that are independent of language. Programmers use plan-based representations and task-based representations when understanding code.

Continuation of Table			
Author	Task	Method	Findings
Bergantz & Hassell (1991)	Study, modification	Protocol analysis from the comprehension phase was analyzed to derive a model.	Supports two-model theory (domain and program) of comprehension for declarative language Prolog. Function and data structure relationships are used to develop a program model.
Corritore & Wiedenbeck (1991)	Study	Novice programmers were asked to study short and long programs, and answer comprehension questions related to each of the five categories of program information: operations, control flow, data flow, state, and function. Participants then wrote a summary of the program.	Results indicate that novices use a bottom-up approach during program comprehension. Novices construct a program model as their mental representation. Novices with better comprehension develop more abstract mental representations based on function information when comprehending short programs but not with long programs.
Koenemann & Robertson (1991)	Study, modification	Verbal protocol analysis was performed on experienced programmers while they completed one of the following modification tasks: functional addition, enhancement, functionality change, or default value change.	Results indicate the comprehension process involves use of beacons to generate hypothesis about the functionality of code. Primarily programmers use a top-down approach to program comprehension but use bottom-up strategies for failing or missing hypothesis or to understand directly relevant code. The scope of the comprehension process is determined by the type of modification task.

Continuation of Table			
Author	Task	Method	Findings
Koubek & Salvendy (1991)	Modification	Expert and super-expert programmers were given a program and a modification task. Audio and visual recordings of participants verbalizing their thought process as they completed the task were analyzed.	Experts use information specific to the modification task to create their initial representations, whereas super-experts initially create a more general abstract representation of the overall program.
Wiedenbeck (1991)	Study	Novice and advanced programmers were given programs to study that either contained beacons or disguised the beacons. In the second study, novice and advanced programmers were given programs that were either prototypical or non-prototypical. In the third study, novice and advanced programmers were given code that either contained a false beacon or did not. In these first three studies participants completed three tasks: described the program's function, rated their confidence in their understanding, and recalled the program. In the final study, four versions of a program, one correct and three incorrect versions (missing lines of code), were given to novice and advanced programmers to study. Participants were asked to describe the function and if they felt the program was incorrect to explain why.	Results indicate that beacons are used in initial comprehension by programmers when understanding code and can reduce the depth of study and simulation required to understand a program. Advanced programmers can make better use of beacons and relied more on beacons than novices. Alternatively, false beacons tend to mislead programmers about the program's function.

Continuation of Table			
Author	Task	Method	Findings
Boehm-Davis, Holt, & Schultz (1992)	Modification	Expert and novice programmers were tasked with performing either a complex or simple modification to programs written in the same programming language, but using three types of problems, and with three different structures: functional, object oriented, and in-line code. After each modification, participants were asked to recall segments of the code and any relationships between the segments. Using the recalled segments and relationships written on cards, participants were asked to create a structure using the cards. The number of segments and relationships recalled, and the depth and width of the structure were analyzed.	Mental representations of experts are primarily affected by the difficulty of the task: more difficult tasks result in more complex representations, whereas novices are primarily affected by the surface structure of the code and the content of the program. The results of the study also indicate that the more time spent thinking about the problem produces more narrow representations, whereas actively exploring and interacting with the program while solving a problem produces mental representations with more breadth and depth.

Continuation of Table			
Author	Task	Method	Findings
Fix, Wiedenbeck, & Scholtz (1993)	Study	Expert and novice programmers studied a program for understanding and were then asked comprehension questions. The questions were related to the five abstract characteristics of mental representations that are formed during program comprehension.	Results support the presence of five abstract characteristics in mental representations formed by expert programmers: hierarchical structure, explicit mapping of code to goals, foundation on recognition of recurring patterns, connection of knowledge, and grounding in the program text. Experts form mental representations containing all five abstract characteristics. Representations formed by novices either do not contain these characteristics or they are poorly developed.
Wiedenbeck, Fix, & Scholtz (1993)	Study	Expert and novice participants answered comprehension questions after studying a program.	Experts' mental representations of programs had more developed abstract characteristics: hierarchical structure, mapping of code to goals, recognition of recurring patterns, connection of knowledge, and grounding in the program text.

Continuation of Table			
Author	Task	Method	Findings
Davies (1994)	Study	Novice, intermediate, and expert programmers were presented with programs that they were asked to memorize. Participants were then presented with either focal (more important) or non-focal lines and asked if they were from the program the participants had memorized.	Novices do not possess complete schematic representational structures. Expert and intermediate programmers use schematic representational structures of programming knowledge to understand programs. In addition, experts use knowledge restructuring by mapping focal structures of a program to an internal representation of schematic programming knowledge.
Teasley (1994)	Study	Novice programmers were given programs that had either meaningful or nonsense variable names to study and were then asked comprehension questions based on four categories of program information: operations, control flow, state, and function.	Results indicate that variable naming style has no affect on experienced programmers but adversely affects novices' acquisition of program function knowledge and not lower-level knowledge. Novice programmers acquire different types of knowledge at similar rates. Experienced programmers acquire function knowledge bottom-up whereas the other types of knowledge are acquired at a similar rate.
von Mayrhauser & Vans (1994)	Study, maintenance	Verbal protocol analysis was used to determine the models formed by programmers while studying the code for understanding in order to perform a maintenance task. Participants had varying levels of expertise, measured by prior knowledge of the code (program knowledge) and domain knowledge.	Presents an integrated model of program comprehension consisting of: program model, situation model, top-down (domain) model, and knowledge base. Programmer switches between levels of abstraction/models.

Continuation of Table			
Author	Task	Method	Findings
Burkhardt & Détienne (1995)	Reuse	Verbal protocol analysis was performed on expert object oriented programmers who were asked to design a solution to a problem. Participants were required to complete the task while alternating between design phases (analyzing the problem and developing a solution) and reuse phases (describing elements the designer would want to re-use from a library).	Results indicate that experts use dynamic mental representations more during the design activity than the reuse activity, and either a top-down or bottom-up approach may be used in the reuse activity.
Davies, Gilmore, & Green (1995)	Study, classify	Expert and novice participants studied and then sorted code fragments into classifications of their choice, providing reasons for their decisions. Participants were then given the option to subdivide any of the classifications and to give their reasoning if they chose to subdivide.	Experts classified mostly based on functional relations and novices classified mostly based on object oriented (OO) classifications. Experts produced more syntactic classifications, whereas novices produced more semantic classifications. Supports findings that experts can form multiple knowledge representations of the same code. Does not support the claims that the OO paradigm is representative of the cognitive structure of programmers.

Continuation of Table			
Author	Task	Method	Findings
Green & Navarro (1995)	Study	Participants had varying levels of programming experience and varying levels of expertise in the three programming paradigms studied: textual, spreadsheet, and visual programming. Participants studied a program one fragment at a time and were then asked comprehension questions to verify their understanding of the program. Participants then asked to rate the closeness of relationship between pair fragments.	Results indicate that different aspects of the schema are emphasized depending on the programming paradigm. Textual paradigms elicit representations that match the goal structure, spreadsheet paradigms elicit representations that match the object structure, and the visual paradigms elicit representations that combine both goal and object structures.

Continuation of Table			
Author	Task	Method	Findings
Schömann (1995)	Study	Advanced programmers had prior knowledge of multiple programming languages, novice programmers only knew the language used in the study. Both groups had the same level of knowledge in the language used in the study. Advanced and novice programmers were shown a program three times and were asked to recall as much of the program as possible after each showing. In a second experiment, advanced and novice programmers were given a program to study, answered comprehension questions about the program, and were then shown segments of code and asked to decide as quickly as possible if they were from one of the programs they studied. Both experiments concluded with an interview of the participants about their encoding and retrieval strategies.	Experts use a schema-driven knowledge organization when understanding programs. Novices use a bottom-up strategy during program comprehension, whereas experts use a top-down strategy and are able to reconstruct the programs instead of relying on the learned program code. Advanced programmers are able to transfer knowledge between programming languages.
Shaft & Vessey (1995)	Study	Verbal protocols were used to analyze the comprehension strategy of expert participants by determining the tracing process. Participants answered comprehension questions related to each abstraction (function, data flow, control flow, and state).	The comprehension strategy used is dependent on familiarity with domain knowledge when it is relevant to understanding the program. Results suggest the use of top-down strategy with familiar domain and the use of bottom-up strategy with unfamiliar domain.

Continuation of Table			
Author	Task	Method	Findings
Snyder (1995)	Modification	Think a-loud was used during the modification task, and a questionnaire was used to measure comprehension of novice and expert participants.	Supports the claim that modification tasks require identifying relationships between four dimensions of program information (data flow, control flow, state, and functionality). Program comprehension is limited to the scope of the modification task. Ability to trace de-localized program information is dependent on expertise.
von Mayrhauser & Vans (1995)	Study, maintenance	Verbal protocol analysis was performed on experienced programmers while they worked on understanding code they would be responsible for maintaining. Participants had varying amounts of previous experience with the code.	Results support the integrated comprehension model. Programmers that maintain code build a mental program model, situation model, and domain model by switching frequently between these three levels of abstraction.
Vans (1996)	Maintenance	A protocol analysis was performed on expert participants' verbalization of their thoughts as they worked on maintenance tasks.	Findings are based on assumptions of the Integrated Code Comprehension Model that has models at different levels of abstraction: program model (low), situation model (intermediate), top-down model (high). Levels that programmers work at and switch between are based on experience and task.

Continuation of Table			
Author	Task	Method	Findings
von Mayrhauser & Vans (1996)	Study, maintenance	Verbal protocol analysis was performed on experienced programmers while they worked on understanding code they would be responsible for maintaining. Participants had varying amounts of previous experience with the code.	Results support the integrated comprehension model where programmers use a multi-level approach to program understanding by switching between program, situation, and domain (top-down) models. Large-scale code requires more knowledge at the domain level.
Ye & Salvendy (1996)	Study	Novice and intermediate programmers were given the code for a program that was divided into numbered segments using a hierarchy of program plans. They were also given a random list of plan goals and were asked to match each program plan (segment of code) to its goal. The sequence in which code segments were matched to goals was observed.	Intermediate and novice programmers use a top-down strategy during program comprehension. Intermediate programmers use a more consistent top-down, depth-first approach whereas novices are less consistent, using more opportunistic strategies.
Barfield (1997)	Recopy	Measured glances and time between glances while expert and novice participants recopied lines of code that were in view. After the recopy task, participants were asked to recall the program verbatim from memory. The program was presented either in executable order, random chunks, or random lines.	Supports the chunking model used to encode programs. Experts create larger chunks than novices allowing them to encode more information. Speculates that experts first encode the plan or algorithm in the chunk, then encode the specific variable names used in the chunk.

Continuation of Table			
Author	Task	Method	Findings
Burkhardt, Détienne, & Wiedenbeck (1997)	Study, documentation, reuse	Verbal protocol analysis was performed on expert and novice object oriented (OO) programmers during every phase. During the first phase participants studied a program and then answered comprehension questions. For the second phase participants completed either a documentation task or reuse task followed by comprehension questions. The comprehension questions were related to the program model and situation model.	Results indicate that OO programmers form a more fully developed situation model early in the comprehension process and the situation model continues to develop over time whereas the program model remains constant. Expert programmers develop a stronger static situation model than novices.
Ramalingam & Wiedenbeck (1997)	Study	Novice participants answered comprehension questions on object oriented (OO) programs and imperative programs. Comprehension questions were related to each of the following knowledge categories: operations, control flow, data flow, state, and function.	Novice programmers formed a program-level mental representation when comprehending imperative programs and a domain-level mental representation when comprehending OO programs.
von Mayrhauser, Vans, & Howe (1997)	Enhancement	Protocol analysis was performed on expert programmers while working on an enhancement task.	Expert programmers perform actions at all three levels of abstraction: domain (top-down), situation, and program models, and switch between these levels. Results support the integrated comprehension model. While performing enhancement tasks, programmers perform more actions at the program and situation model levels.

Continuation of Table			
Author	Task	Method	Findings
Burkhardt, D́tienne, & Wiedenbeck (1998)	Study, documentation, reuse	Expert and novice object oriented programmers were given programs to study and later were asked to complete either a documentation or reuse task. Verbal protocol analysis was performed on the participants and the files accessed and transactions between files were recorded.	Results indicate that experts use a top-down approach initially during program comprehension to form an abstract representation, and later focus on implementation details. Novices do not use a top-down approach until later in the comprehension process.
Furman (1998)	Study	Expert and novice participants were presented with search programs written in different forms where the lines of code were either indented, left-justified, randomly indented. The characters in the code were replaced with X's and the user could select a single line of code to reveal at a time. Measured the study time for individual lines of code, time to select the next line of code to reveal, and number of lines of code revealed to understand the code. For each form, participants completed a comprehension test and gave a subjective rating in terms of their like or dislike, difficulty of the task, and fatigue after completing the task.	Supports the theory that programmers use chunking when understanding code. Experienced programmers like or dislike of a program was more affected by form than novices. Overall, participants had lower look times and chose to reveal fewer lines when the program was normally indented. Results indicate that programmers use indentation to understand the sectioning of code into functional units and that program forms conducive to chunking improve program comprehension.

Continuation of Table			
Author	Task	Method	Findings
Shaft & Vessey (1998)	Study	Protocol analysis was performed on professional programmers while they studied programs written in either a familiar or unfamiliar domain. Participants answered comprehension questions related to each abstraction (function, data flow, control flow, and state).	Results indicate that some programmers use a flexible approach that involves using a top-down process in a familiar domain and bottom-up process in an unfamiliar domain, while others use either a top-down or bottom-up process regardless of familiarity. Programmers who use a flexible approach construct mental representations that contain connections between the domain and program levels.
von Mayrhauser & Vans (1998)	Maintenance	Protocol analysis was performed on expert programmers while working on an adaptive maintenance programming task.	Results support the integrated model of comprehension. Expert programmers performing adaptive maintenance tasks on large scale software, concentrate on the domain model level to a greater extent than the program and situation model level. Programmers switch between all three levels of abstraction, using a combination of top-down and bottom-up approaches during program comprehension.

Continuation of Table			
Author	Task	Method	Findings
Wong, Cheung, & Chen (1998)	Study	Participants, after studying a set of programs for understanding, were shown the programs again in a random order with either semantic changes, surface changes, or no changes, and were asked to identify the programs they recognized. To determine if the same comprehension process is used for English language processing, participants completed a lexical decision task. Participant groups were expert, novice, and control.	Results provide supporting evidence that expert programmers are more likely to form and use semantic representations of programs during program comprehension. Experts' reliance on semantic knowledge is specific to processing computer programs and does not transfer to English language processing.
Corritore & Wiedenbeck (1999)	Study, maintenance	Experts participants studied a program written in a language coinciding with their paradigm of expertise, object oriented (OO) or procedural. Participants answered comprehension questions after the study phase and after completing the modification phase. Comprehension questions were used to measure the following knowledge categories: operations, control flow, data flow, function, and structure. The knowledge categories were grouped to determine the model formed after each phase.	Supports a mixed model representation consisting of equally developed domain and program models. Results indicate that OO programmers initially develop a strong domain model after initial exposure and develop a mixed model after repeated exposure. The repeated exposure assists in developing their program model, however, the emphasis remains on the domain model. Procedural programmers develop a mixed model from the initial exposure with an emphasis on the domain model.

Continuation of Table			
Author	Task	Method	Findings
Vans, von Mayrhauser, & Somlo (1999)	Maintenance	A verbal protocol analysis was performed on participants while carrying out a corrective maintenance task. Participants had varying levels of expertise, measured by prior knowledge of the code (program knowledge) and domain knowledge.	Results support the Integrated Comprehension Model and present hypotheses regarding the relationship between programmer expertise (domain and program knowledge), the level of abstraction comprehension occurs at, and the resulting model that is derived (program, situation, domain). Experts in domain and program knowledge can make connections and switch between all three levels of abstraction.
Wiedenbeck & Ramalingam (1999)	Study	Novice participants, categorized as object oriented (OO) or procedural according to their programming language training, studied a short, simple program and answered comprehension questions from memory. Comprehension questions were from the knowledge categories: operations, control flow, data flow, and function.	Results indicate that less advanced novice OO programmers form a stronger domain model, and less advanced novice procedural programmers form a stronger program model. More advanced novices regardless of programming paradigm form a more balanced representation that includes both program and domain knowledge. Supports the theory that program comprehension requires well developed and connected program and domain models.

Continuation of Table			
Author	Task	Method	Findings
Wiedenbeck, Ramalingam, Sarasamma, & Corritore (1999)	Study	Novice participants, categorized as object oriented (OO) or procedural according to their programming language training, studied a short, simple program and answered comprehension questions from memory. Participants then completed two study sessions with a long, complex program and answered the same comprehension questions after each study session. The code was provided for the final set of comprehension questions. Comprehension questions were from the knowledge categories: operations, control flow, data flow, and function.	Results indicate that novice OO programmers develop a stronger domain model than novice procedural programmers during comprehension of short programs. During the comprehension of long programs, novice procedural programmers developed a stronger program model compared to their domain model. Novice OO programmers did not have a more developed domain model than procedural programmers for the long program.

Continuation of Table			
Author	Task	Method	Findings
Corritore & Wiedenbeck (2001)	Study, maintenance	Expert participants studied and performed maintenance tasks on programs written in a language coinciding with their paradigm of expertise, object oriented (OO) or procedural. Participants completed a study session followed by two modification sessions completed over a longer period of time. The documents participants accessed and their actions during the tasks were analyzed.	The results indicate that the direction of comprehension is affected by the programming paradigm, time, and task. Programmers, and to a greater extent OO programmers, use a top-down strategy during initial comprehension of a program, switching to a bottom-up strategy as their knowledge of the program increases over time, and given a motivating task. Programmers develop a wide breadth of understanding by initially using a broad comprehension strategy, that is more pronounced with procedural programmers, and over time use a more narrow, focused strategy.
Mosemann & Wiedenbeck (2001)	Study	Novice participants studied a program using either sequential, control flow, or data flow navigation. The number of correct responses and response times to comprehension questions answered from memory were measured to determine the mental representation formed. Questions were related to each of the five comprehension categories: module sequential, control flow, data flow, global goals, and operations.	Supports the claim that novice programmers construct mental models of the basic text structures using a bottom-up (sequential navigation) or top-down (control flow navigation) approaches. Data flow navigation is the least effective navigation method for assisting novices in developing mental models.

Continuation of Table			
Author	Task	Method	Findings
Navarro-Prieto & Cañas (2001)	Study, modification	Participants of varying expertise studied or modified programs in the language paradigm of their expertise, visual or procedural. To determine the mental representation developed by the participants, the results of a primed recognition task and a grouping task were analyzed.	Visual programmers develop stronger mental representations than procedural programmers. Procedural programmers' control flow structures are more developed than their data flow structures, whereas visual programmers develop both structures equally well. Procedural programmers focus on control flow information whereas visual programmers focus on data flow information.

Continuation of Table			
Author	Task	Method	Findings
Romero (2001)	Study	Experienced and novice Prolog programmers were asked to study a program and then recall the code and describe the program's function. The recall of key segments and non-key segments of four structural models of comprehension (plans, techniques, data structure schemas, and recursion points), and their ability to identify the function were measured. Experienced and novice Prolog programmers were asked to study a program with either meaningful or cryptic naming styles. Participants were then shown segments of code that related either to focal elements of plans or focal elements of data structure schemas and asked if they appeared in the program they had studied. Finally, participants answered comprehension questions related to functional aspects and data structure issues.	Extends the knowledge restructuring theory that experienced programmers restructure their knowledge according to the type of programming information (plans, function, and data structure). Proposes data structure schemas as a model of structural knowledge for Prolog programmers while recognizing that plan and function information are also important.

Continuation of Table			
Author	Task	Method	Findings
Burkhardt, Détienne, & Wiedenbeck (2002)	Study, documentation, reuse	Novice and expert object oriented (OO) programmers were given either a documentation task or a reuse task. Completion of the tasks were divided into two phases, a study phase followed by the task phase. Participants were asked comprehension questions after each phase related to the program and situation model.	Expertise effects the construction of the situation model but not the program model during a documentation task. Stronger situation models are formed by both experts and novices during reuse task, and the difference in their models decrease over time as novices improve their situation model given a task that requires situation knowledge.
Khazaei & Jackson (2002)	Study	Novice programmers that had experience with both event-driven (ED) and object oriented (OO) programming paradigms were given a program to study a set of comprehension questions to answer. Each participant studied and answered comprehension questions for both ED and OO programs. Comprehension questions were related to elementary operations, control flow, data flow, function, and state.	Novice programmers form stronger control flow, function, and data flow models for ED programs. Novice programmers form weak elementary operations models and strong state models for both ED and OO programs. Overall, novices form a stronger model of data flow.

Continuation of Table			
Author	Task	Method	Findings
Parkin (2004)	Maintenance	Experienced programmers completed either a corrective or enhancement maintenance task. Software was used to track the actions of the programmers while they completed their task.	Experienced programmers completing corrective maintenance tasks initially use top-down comprehension strategies to develop a domain-level model before constructing a program-model. When completing enhancement maintenance tasks, programmers switch from top-down to bottom-up strategies earlier in the program comprehension process.
Romero & Du Boulay (2004)	Debugging	Expert and novice programmers were asked to read the program specification, they were then given the program code and were asked to determine if the code met the specification. The programs each contained three errors: plan, schema, and other.	Mental representations formed by expert Prolog programmers are hierarchically organized based on data structure.
Sajaniemi & Prieto (2005)	Study, modification, classify	Expert programmers were given a program to study for understanding and completed a modification task. Participants were then given cards with the name of each variable from the program and were asked to sort the cards into groups such that similar variables were in the same group. Participants then wrote an explanation for the groups and explained during an interview their sorting criteria and the contents of each group.	Expert programmers use programming knowledge in the form of plans combined with behaviour to develop four categories for representing variables in a program: domain-based, technology-based, execution-based, and strategy-based.

Continuation of Table			
Author	Task	Method	Findings
Fan (2010)	Study, debugging	Participants studied a program and completed a recall test from memory. During another session, participants located errors in a program and provided corrections. The programs were written in two versions; with comments and without comments. Eye-movement data, time, and scores on tasks were recorded.	Results suggest that comments can improve comprehension by assisting programmers with the chunking process and reducing memory load depending on how they are used and the programmer's familiarity of the domain. Findings indicate that the beacons identified by programmers is dependent on the task.
Alardawi & Agil (2015)	Study	Novice object oriented (OO) programmers studied programs written with the use of class structure and without class structure. Participants were asked comprehension questions related to elementary operation, control flow, data flow, function, state, and problem classes' category knowledge.	Programs containing class structure allow novice OO programmers to form stronger mental representations during program comprehension.
Nosál & Porubän (2015)	Study, modification	Programmers with varying levels of expertise were asked to think-a-loud while studying a program for understanding. Participants were then asked to think-a-loud while modifying the program. Participants then answered comprehension questions.	Results support the four-layer mental model created using a hypothesis-based approach to program comprehension given that the programmer possesses the necessary domain knowledge. The four-layer model consists of two layers in the problem domain: problem and features/concepts, and two layers in the solution domain: plans/beacons and source code.
End of Table			