Program Comprehension

Idaho State University | Computer Science

Isaac Griffith

CS 4423 and CS 5523
Department of Computer Science
Idaho State University

ROAR

Idaho State University
Computer Science

Outcomes

After today's lecture you will be able to:

- Understand and describe the various cognition models for program understanding, including
  - Letovsky Model
  - Shneiderman and Mayer Model
  - Brooks Model
  - Soloway, Adelson, and Ehrlich Model
  - Pennington Model
  - Integrated Metamodel

ROAR

Cognition Models

CS 4423/5523

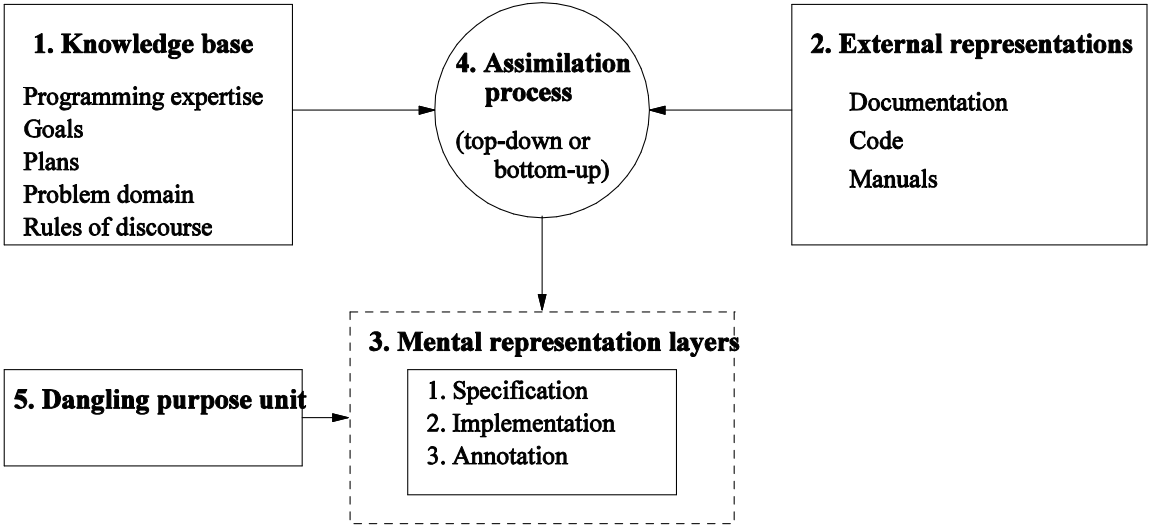
ROAR

Cognition Models for Program Understanding

- Letovsky model
- Shneiderman and Mayer model
- Brooks model
- Soloway, Adelson, and Ehrlich model (top-down model)
- Pennington model (bottom-up model)
- Integrated metamodel

Idaho State University
Computer Science

Letovksy Model



ROAR

1. Knowledge base

Programming expertise

- Programming expertise helps in asking questions, making conjectures, and searching for specific information in the code.
- The questions are grouped into five categories:
  - **Why** questions are designed to know about the purpose of actions and design choices.
  - **How** questions assist the programmer to learn about the way some goal of the code is accomplished.
  - **What** questions are used to find out what a variable or code fragment is.
  - **Whether** questions are asked to know if the code behaves in a certain way.
  - **Discrepancy** questions are meant for resolving confusions and apparent inconsistencies in the code.
- Some example questions are as follows:
  - Why is the variable being reset to zero?
  - What is done to the memory block after the data is transmitted?
  - Why is the memory block being deleted in two places?

1. Knowledge Base

- **Goals**
  - A programmer may find recurring computational code blocks, such as sort, search, delete, connect (to servers), start timers, transmit, and receive.
  - It is useful to know the meaning of those recurring computational goals in the program, to be able to create a higher level of abstraction.
- **Problem domain**
  - A good understanding of the application domain serves as a backdrop for clearly and quickly understanding code segments in order to identify their goals and creating abstractions.
- **Plans**
  - Programmers have their own ways (also called plans) of finding solutions to problems.
  - They use widely used solutions to some common problems.

1. Knowledge Base

- Rules of discourse
  - Programmers have knowledge of stylistic conventions in writing code, which assist them in recognizing the goals of procedures and interpreting variables.
  - For example, if a constant is called MAX_RECORDS and is used in a record processing loop, the programmer quickly recognizes that the loop is going to iterate for a maximum count of MAX_RECORDS.

2. External Representations

- The external representations of a program include its source code, documentation in the form of some comments, and manuals.
- The manuals are useful in understanding the high-level goals of the code, whereas the in-line comments are useful in understanding the low-level details.

3. Mental Representation

By reading code and documents, a programmer may create the followings:

- Specification of the program
  - Here, a specification means a complete and unambiguous description of the goals of the program. This is done by identifying the user-level functions, attributes of the functions, and program constraints.
- High-level implementation of the program
  - This means producing a complete and unambiguous description of the actions and data structures of the program.
- Annotation of the program
  - Make a two-way association between the goals and the actions and data structures, by annotating the program as follows:
    - How each goal in the specification is accomplished and by which actions and data structures.
    - What goals use the services of a given action or a data structure.
  - In other words, establish a traceability matrix between program goals and actions and data structures.

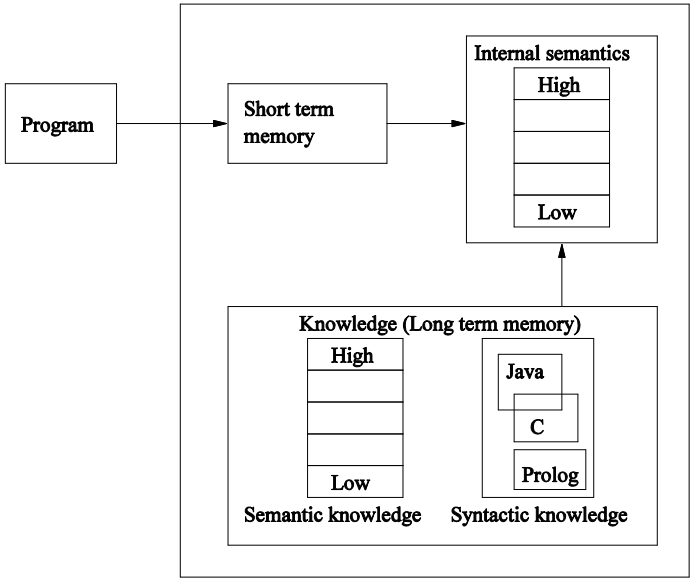
4. Assimilation Process

- Programmers combine their knowledge base and the external representations to create their mental models. This process is known as assimilation.
  - The assimilation process can work in three ways: top-down, bottom-up, and opportunistic.
    - Top-down: Begin with a goal, followed by possible implementations of the goal.
    - Bottom-up: Identify program plans from code, make annotations, and move up to the top.
    - Opportunistic: Combine both top-down and bottom-up in an opportunistic manner.

5. Dangling Purpose Unit

- This unit captures those goals whose implementations have not been clearly understood.

Shneiderman and Mayer Model



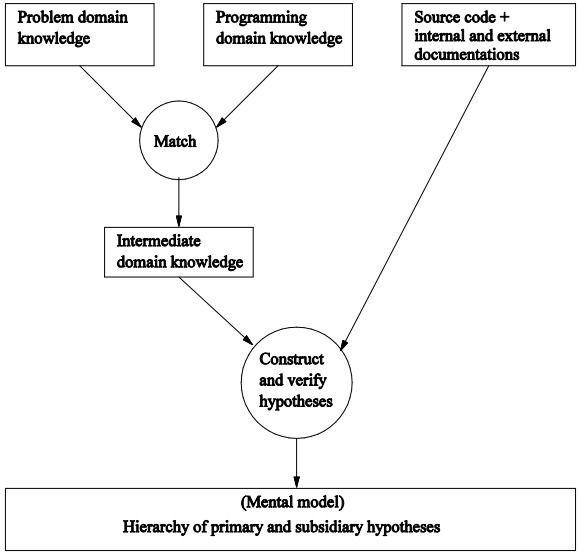
Shneiderman and Mayer Model

- The model comprises three key components
  - Short term memory of the programmer
  - The programmer's knowledge to understand the code
  - Internal semantics of the code as understood by the programmer
- Internal semantics
  - An internal semantics lies between the top-level goals of the program (aka the what aspects) and their detailed implementations.
  - In between the two extremes, programmers develop an internal semantic structure to represent the program.
  - An example of intermediate level abstraction is the concept of call graphs.

Shneiderman and Mayer Model

- Knowledge
  - Here knowledge refers to the application domain knowledge and programming knowledge (both syntactic knowledge and semantic knowledge), which are stored in the long term memory of the programmer.
    - Syntactic knowledge concern individual programming languages.
    - Semantic knowledge means programming concepts and techniques at different levels of abstractions.
- Short term memory
  - The capacity of the short term memory is very limited, and, therefore, the programmer must be able to quickly identify chunks, create their abstractions, and represent those abstractions in some internal form.

Brooks Comprehension Model



Brooks Comprehension Model

- The three key elements of the model are
  - Code viewed as performing mappings from a problem domain to the programming domain
  - Understanding the mappings in terms of hypotheses
  - Verification and refinement of hypotheses

Idaho State University
Computer Science

Brooks Comprehension Model

- The three key elements of the model are
  - Code viewed as performing mappings from a problem domain to the programming domain
    - Developing a software system can be seen as performing a series of mappings from one domain to the next, starting from the problem domain and finishing in the programming domain.
    - The results of the mappings are documented with varying degree of details, whereas the thought processes that perform the mappings are generally missing from the documentations.
    - For correct and complete understanding of the program, it is important to understand the mappings and their relationships, which is realized by means of constructing hypotheses and validating them.

ROAR

Idaho State University
Computer Science

Brooks Comprehension Model

- Understanding the mappings in terms of hypotheses
  - Programmers read all available documentations and try to reconstruct the mappings as much as they can.
  - They are said to have truly comprehended the program if all the mappings are exactly reconstructed.
  - Programmers try to understand the program by formulating hypotheses in terms of what they find from the available documentations, their expectations, their current level of understanding of the program, and their knowledge of the problem domain and programming in general.
  - Hypothesis construction begins with the generation of a primary hypothesis concerning the global structure of the program in terms of inputs, outputs, major data structures, and the processing sequences.
  - Hypotheses can be organized in a hierarchical manner to represent both the breadth and depth of comprehending the program.

Brooks Comprehension Model

- Verification and refinement of hypotheses
  - A hypothesis represents a programmer's understanding of a certain aspect of the program – and that understanding may be correct, incorrect, or partially correct.
  - A hypothesis must be verified or refined by means of further understanding of the program.
  - Programmers verify a hypothesis by searching the program text and related documentations for beacons that confirm the hypothesis.
  - While reading code to find beacons, programmers try to develop a broad understanding of the program by having an open mind about the system, rather than stay focused only on the hypothesis under consideration.
  - A programmer can continue constructing and validating hypothesis, thereby creating a hierarchical structure of hypotheses, where the top one is the primary hypothesis and the others are subsidiary hypotheses, and code segments are bound to specific hypothesis.

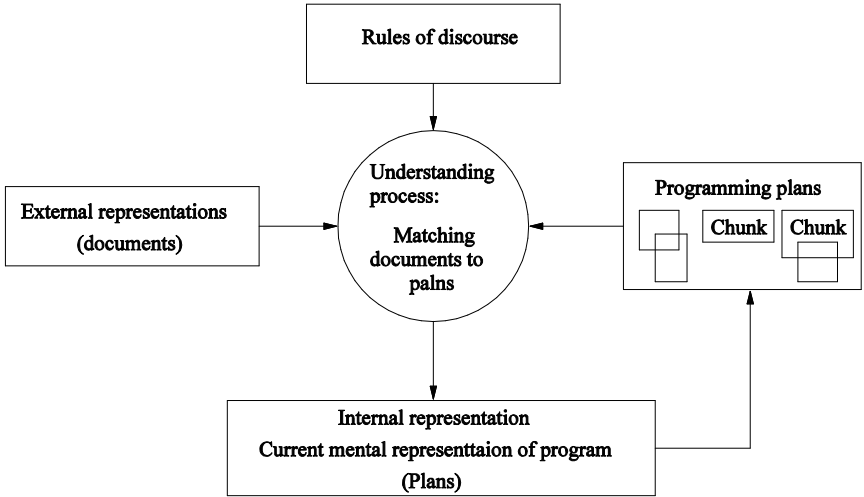
Brooks Comprehension Model

- Verification and refinement of hypotheses
  - Programmers may encounter a number of problems while verifying hypotheses:
    - The programmer fails to find code to bind to a subsidiary hypothesis.
    - The same code is bound to multiple subsidiary hypotheses.
    - The programmer fails to bind a code segment to any hypothesis.
  - The above problems can be resolved by adopting new hypotheses, refining the existing hypotheses, and altering and adding to the bindings of code segments to hypotheses.

Brooks Comprehension Model

- Brooks has identified a number of factors having an impact on program comprehension:
  - Characteristics of source code
  - Quality of documentation
  - Task differences affect comprehension
  - Programmers differ in their ability to comprehend programs

Soloway, Adelson, and Ehrlich Model



Soloway, Adelson, and Ehrlich Model

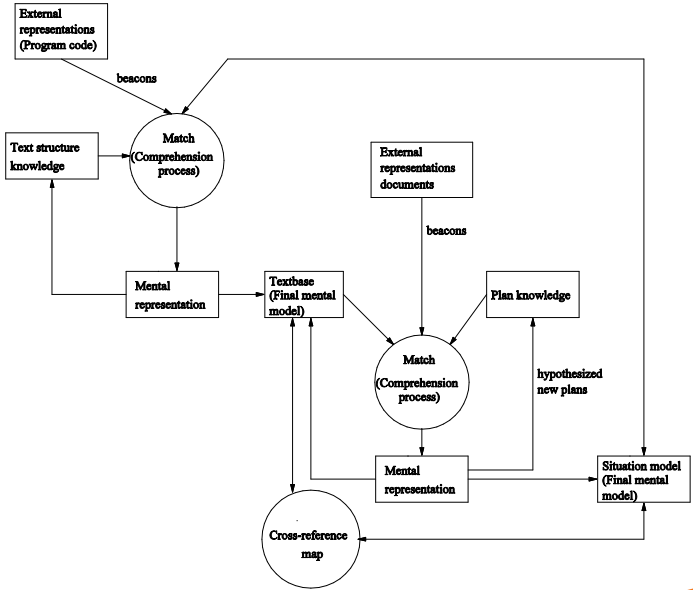
- This model works in a top-down manner, and it applies when the code is familiar to the programmer.
- Two fundamental concepts in the model are
  - programming plans (aka schemas) and
  - programming rules of discourse.
- Programmers have and use specific programming plans and rules of programming discourse to comprehend programs.
- Some concrete rules of programming discourse are:
  - The names of the variables reflect their purpose.
  - Code that is not going to be executed is not included.
  - A tested condition must have the potential of evaluating to true.
  - A variable that is initialized by means of an assignment statement is subsequently updated with assignment statements.
  - Use an `if` statement to execute a code segment once, whereas `for()` and `while()` loops are used to repeatedly execute code segments.

Soloway, Adelson, and Ehrlich Model

- Similar to the other models, documentations play a key role in program understanding in this model.
- The understanding process matches programming plans found in source code with external documentations using rules of discourse.
- During the understanding process, the programmer creates a hierarchical knowledge structure representing their understanding of the code.
  - Comprehension begins with a high level program goal, and finer, lower level subgoals are generated to realize the upper level goals.
- Comprehension is an iterative process; in each iteration, the programmer expands their understanding of the code by refining the already identified subgoals and identifying new subgoals.
- The process is said to complete when the programmer has associated all the programming plans with the goal hierarchy.

Idaho State University
Computer Science

Pennington Model



ROAR

Pennington Model

- The model applies two concepts:
  - textbase and
  - situation model. is important to
- Pay attention to the loop:
  - {Match – Mental representation – Text structure knowledge} followed by the Textbase.
- The programmer iterates through the loop, thereby incrementally creating the mental representation.
- Finally, when the programmer stops iterating through the loop, the final mental representation is known as the textbase.
- There is a similar relationship between the second mental representation box and the situation model.

Pennington Model

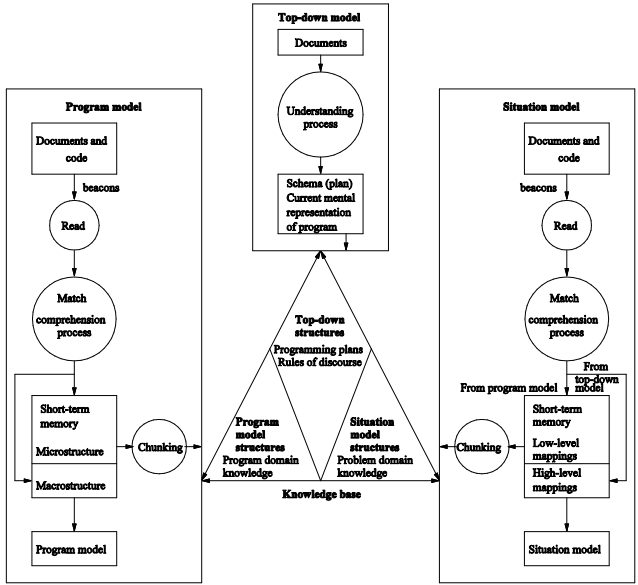
- The concepts of textbase and situation model are explained as follows:
  - Textbase (Program model)
    - A textbase represents information that the reader of a text can recall from memory after reading the text.
    - A textbase includes a hierarchy of representations comprising a surface-level knowledge of the text, a microstructure of relationships among text propositions, and a macrostructure organizing the text representation.
    - The textbase basically describes a program model in terms of the control flow of the program, because when programmers read new code they build a control flow abstraction of the code.
  - Situation model
    - A situation model represents what the text is about.
    - The model requires knowledge of the real-world domains and objects.
    - Situation models are built via cross-referencing and chunking.

Pennington Model

- A high level description of the model is as follows:
  - The programmer assimilates their understanding of the code, knowledge of the text structure, and the situation model to create a mental model in the form of a textbase.
  - The programmer assimilates the documentations, plan knowledge, and the textbase to create the situation model.
  - The textbase and the situation model are cross-referenced to refine and update the two models.
- While reading code, programmers gain knowledge about the following aspects of code:
  - Operations
  - Control Flow
  - Data Flow
  - State
  - Function

Idaho State University
Computer Science

Integrated Metamodel



ROAR

Integrated Metamodel

- The four major components of the model are explained in the following:
  - **Top-down model**
    - It is also known as domain model and it represents domain knowledge about the program.
  - **Program model**
    - If the programmer understands what the program is doing from the control flow aspect, then he has a program model of the code.
  - **Situation model** (See the Pennington's model)
    - After building a program model, a programmer builds a situation model by using control flow and data flow information in a bottom-up manner.

Idaho State University
Computer Science

Integrated Metamodel

Knowledge base

- The knowledge base provides a medium for interactions among the above three models, and comprises the following three kinds of knowledge structures. Knowledge base (contd.)
- The following three kinds of knowledge structures.
  - Top-down structures
    - These include programming plans and rules of discourse.
    - The programming plans are categorized into: strategic plans, tactical plans, and implementation plans.
  - Program model structures
    - These include two kinds of knowledge: text-structure knowledge and plan knowledge.
  - Situation model structures
    - These are described in terms of problem domain knowledge and functional knowledge.

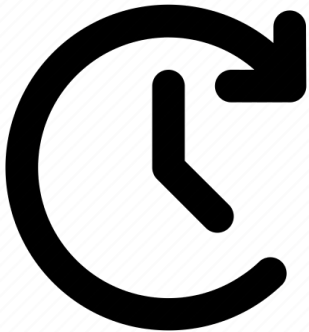
ROAR

Integrated Metamodel

- In the integrated metamodel, all the three models are simultaneously built.
- The programmer takes an opportunistic approach and simultaneously builds all the three models by updating the knowledge base with the understanding of one model and applying the knowledge to further build another model.
  - For example, the knowledge generated from the program model can be used to expand and/or refine the situation model, and vice versa.
  - The programmer uses the knowledge base for guidance to make a transition from one model to another.
- The knowledge base is also called long-term memory, and it is generally organized into schemas (aka plans).

For Next Time

- Review EVO Chapter 8.3
- Read EVO Chapter 8.4 - 8.6
- Watch Lecture 22



Are there any questions?