Program Comprehension

Moumita Asad

IIT, DU

Program Comprehension

- the process of acquiring knowledge about a computer program
- the precondition of performing any maintenance related activities
- consumes more than half of the maintenance resources

Rugaber, S. (1995). Program comprehension. Encyclopedia of Computer Science and Technology, 35(20), 341-368.

Aims of Program Comprehension

- The ultimate purpose of comprehending programs is to be able successfully to implement requested changes.
- This entails acquiring information about following aspects of a system:
  - Problem domain
  - Execution effect
  - Cause-effect relation
  - Product-environment relation
  - Decision-support features

Problem Domain

- To make change or simply to estimate the resource required for a maintenance task, knowledge of the problem domain in general and the sub-problems in particular is essential to direct maintenance personnel in the choice of suitable algorithms, methodologies and tools
- Information can be obtained from various sources - the system documentation, end-users, or the program source code

Execution Effect

- At a high level of abstraction, the maintenance personnel need to know what results the program will produce for a given input without necessarily knowing which program units contributed to the overall result or how the result was accomplished
- At a low level of abstraction, they need to know the results that individual program units will produce on execution
- Knowledge of data flow, control flow and algorithmic patterns can facilitate the accomplishment of these goals
- During maintenance, this information can assist the maintenance personnel to determine whether an implemented change achieved the desired effect or not

Cause-Effect Relation

- It allows the maintenance personnel to reason about how components of a software product interact during execution
- It enables a programmer to predict the scope of a change and any knock-on effect that may arise from the change
- The cause-effect relation can be used to trace the flow of information through the program. The point in the program where there is an unusual interruption of this flow may signal the source of a bug

Product-Environment Relation

- A product is a software system. An environment is the totality of all conditions and influences which act from outside upon the product, e.g., business rules, government regulations, work patterns, software and hardware operating platforms
- This knowledge can be used to predict how changes in these elements will affect the product in general and the underlying programs in particular

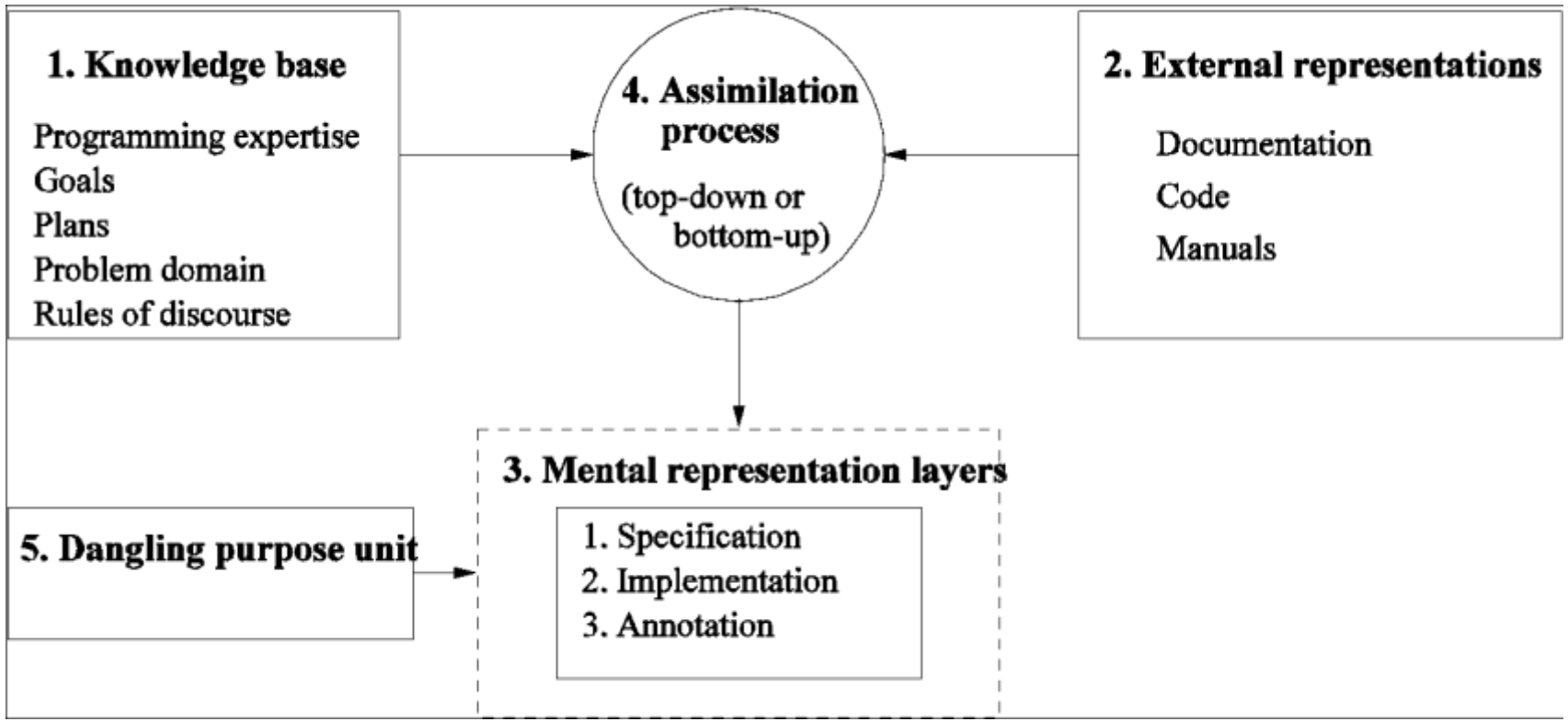
Decision-Support Features

- Software product attributes such as complexity and maintainability are examples that can guide maintenance personnel in technical and management decision-making processes like decision making, budgeting and resource allocation
  - Measures of the complexity of the system can be used to determine which components of the system require more resource for testing
  - The maintainability of the system may be used as an indicator of its quality

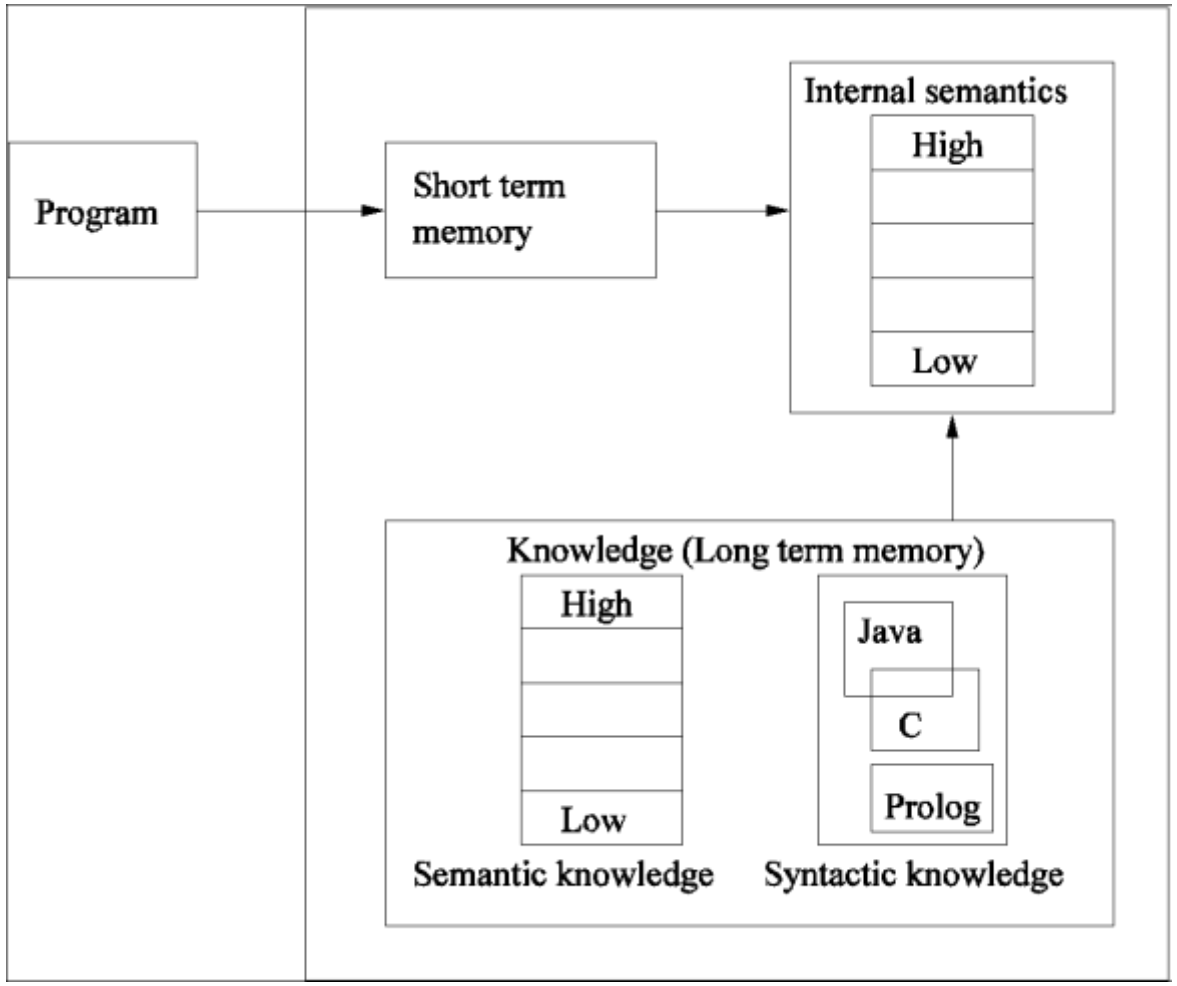
Cognition Models for Program Comprehension

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

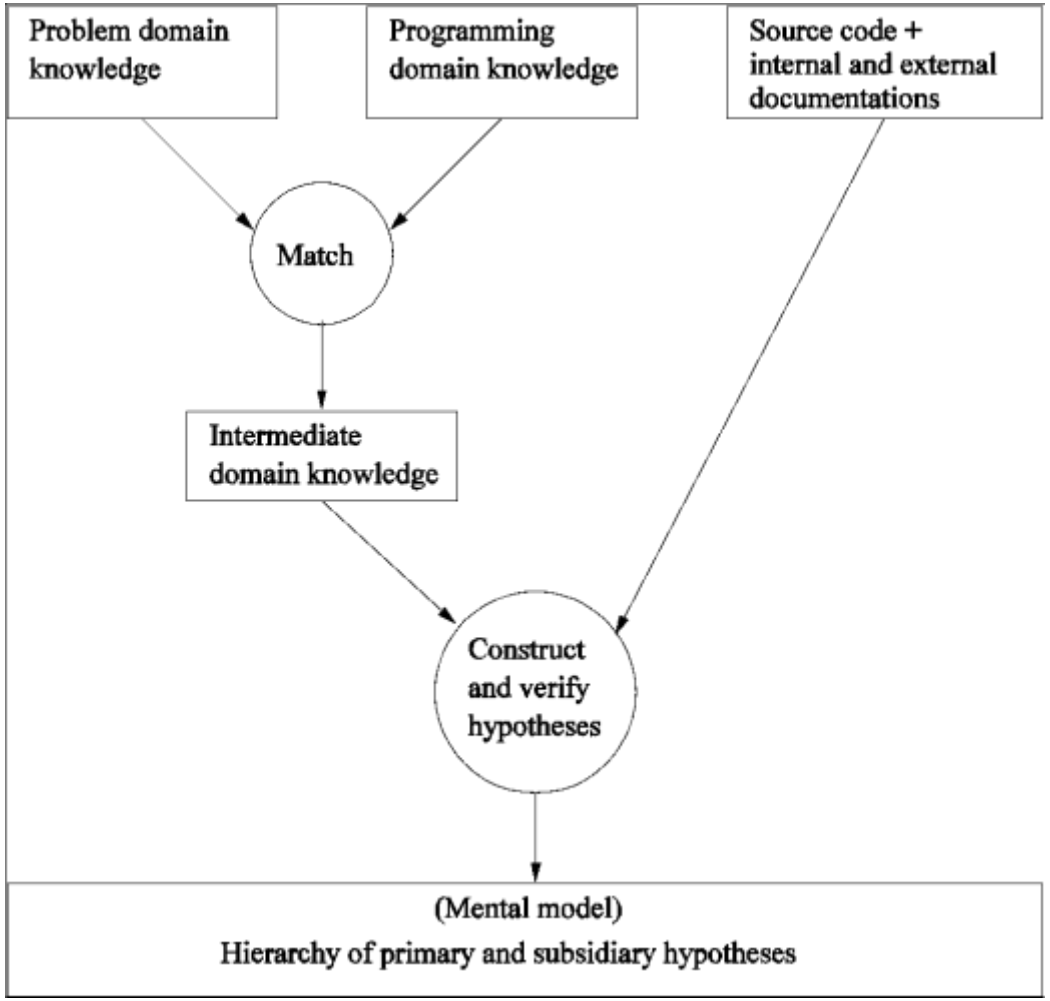
Letovsky Model



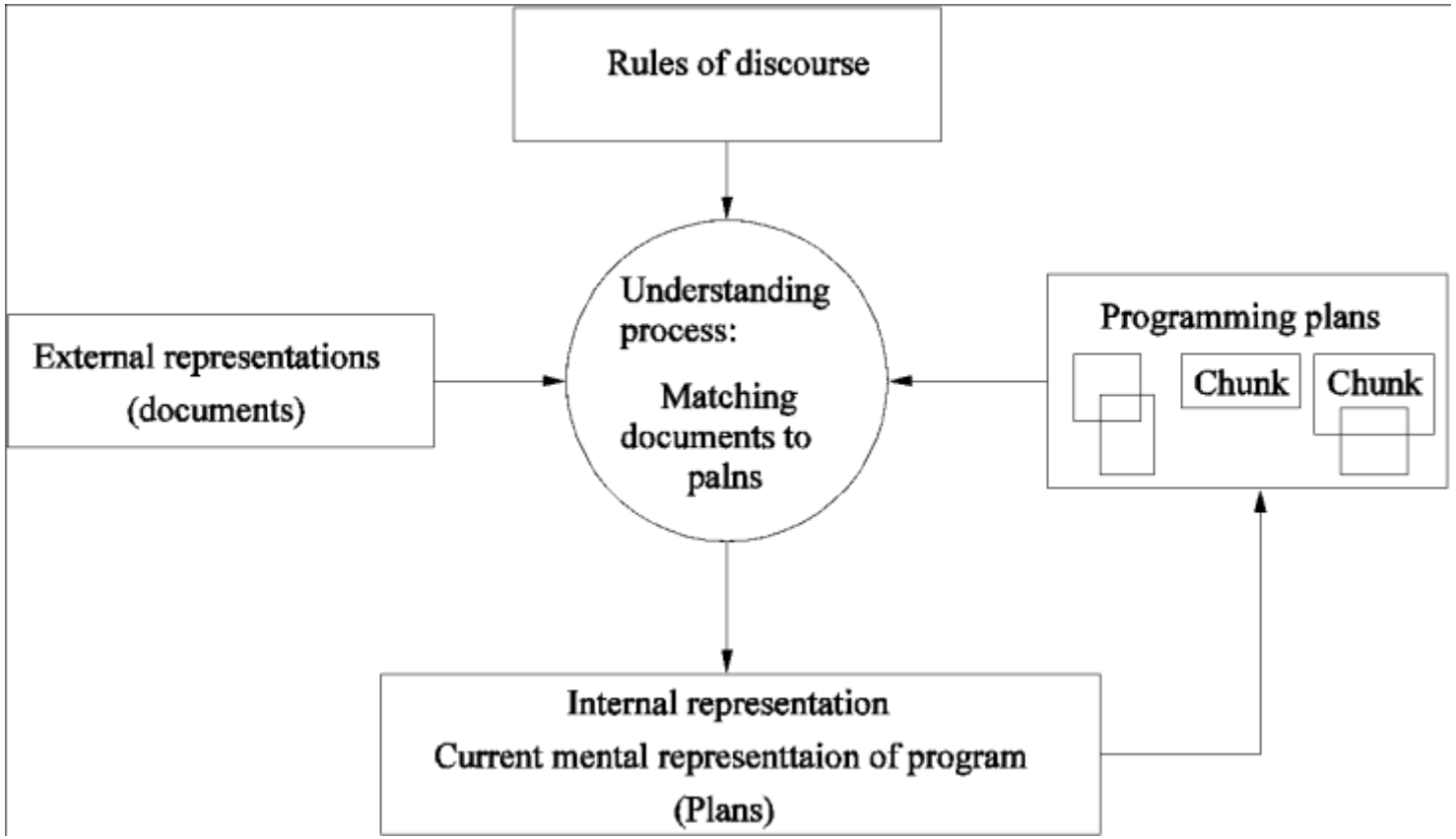
Shneiderman and Mayer Model



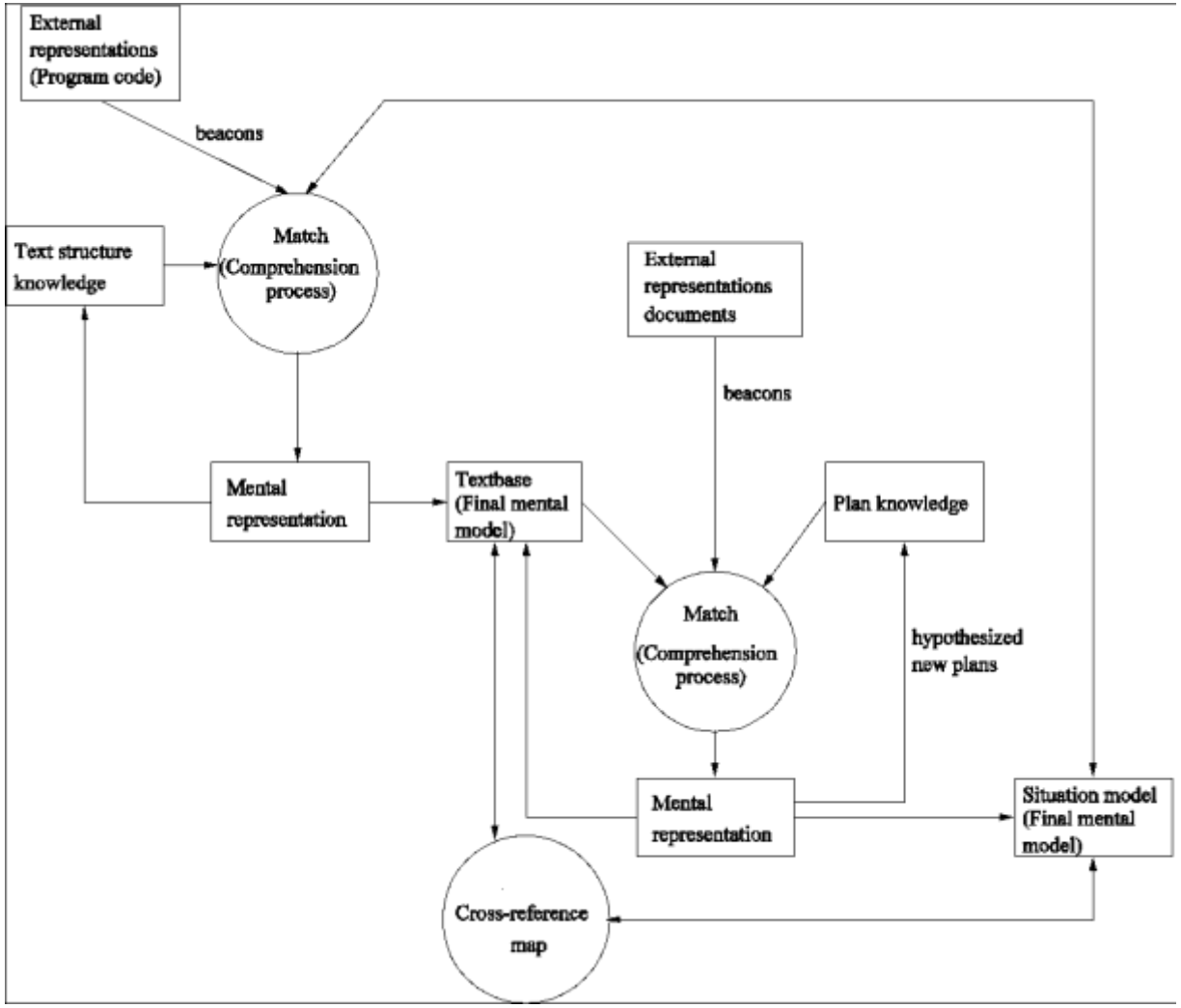
Brooks Model



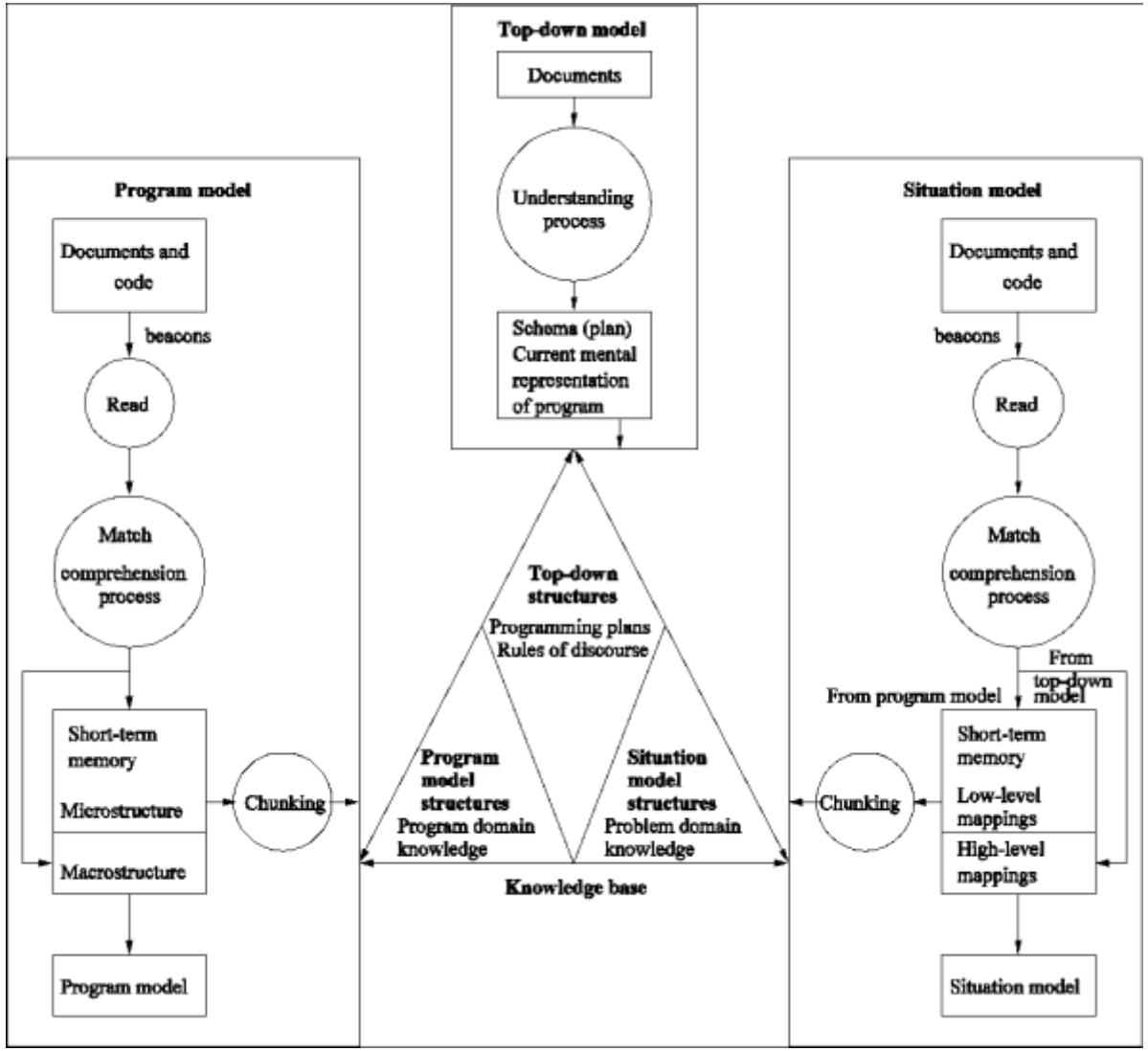
Soloway, Adelson, and Ehrlich Model



Pennington Model



Integrated Metamodel



Factors that Affect Understanding

- Expertise
- Implementation Issues
- Documentation
- Organisation and Presentation of Programs

Expertise

- Programmers become experts in a particular application domain or with a particular programming language by virtue of the repertoire of knowledge and skills they acquire from working in the domain or with the language
- Expertise has a significant impact on comprehension
- The more experienced a programmer is with an application domain or with a programming language, the easier and quicker it is to understand a program and indeed, the whole software system
- Experts differ from novices in both their breadth and their organisation of knowledge: experts store information in larger chunks organised in terms of underlying abstractions. This organization apparently facilitates quick recognition of problem types and recall of associated solution strategies

Implementation Issues

- At the program level, the naming style, comments, level of nesting, clarity, readability, simplicity, decomposition mechanism, information hiding and coding standards can affect comprehension

```
if(marks<40) {
    result = "fail";
}
else {
    if(marks>=80) {
        result = "A";
    }
    else if(marks>=60) {
        result = "B";
    }
    else {
        result = "C";
    }
}
```

Implementation Issues: Naming Style

- Meaningful identifier names can provide clues that assist programmers to invoke appropriate plans during understanding
- Identifier names should be as informative, concise and unambiguous

```
A := FALSE;
WHILE NOT A DO
   IF B.C=B.J THEN
      B.E := B.E+D.F;
      IF G.EOF THEN
         A := TRUE
      ELSE
         ReadBlock (G,D)
      END;
   ELSE
      WriteBlock (H,B);
      ReadBlock (I, B)
   END;
END;
```

Implementation Issues: Comments

- Program comments within and between modules and procedures usually convey information about the program, such as the functionality, design decisions, assumptions, declarations, algorithms, nature of input and output data, and reminder notes
- Prologue comments precede a program or module and describe goals
- In-line comments, within the program code, describe how these goals are achieved
- N.B: The quality of the comment is important, not its presence or absence

```
/**
* @return true or false
*/
protected boolean
    isLoginTicketBased() throws
    Exception {
    ..........
    ..........
}
```

Documentation

- It is not always possible to contact the original authors of the system for information about it. This is partly due to the high turnover of staff within the software industry: they may move to other projects or departments, or to a different company altogether
- Maintainers need to have access to the system documentation to enable them to understand the functionality, design, implementation and other issues that may be relevant for successful maintenance

Organisation and Presentation of Programs

- Well-structured programs take less time to understand
- Indentation is used to emphasise the logical or syntactic relation between statements (or groups of statements) in a program, for example the use of indentation to group together statements belonging to a given control structure
- Automatic program layout tools such as pretty-printers can be used automatically to enforce consistent program layout