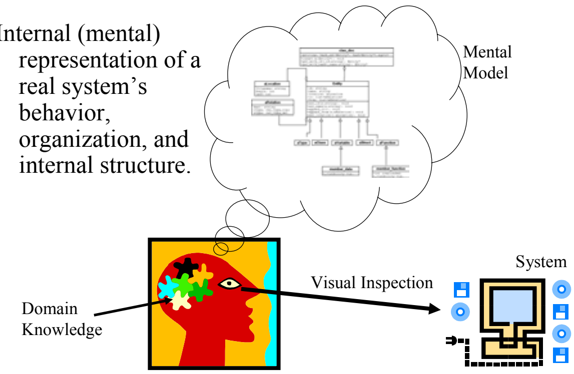
Mental Models for Program Understanding

Dr. Jonathan I. Maletic
<SDML>
Computer Science Department
Kent State University

What is a Mental Model?

Internal (mental) representation of a real system's behavior, organization, and internal structure.



Mental Models

- Must construct a mental model of a system in order to use the system (product)
- The goal is to understand how a system works well enough to support a given usage
- Use:
  - Visual inspection, reading
  - Knowledge about the problem domain, the system, past experience, heuristics

Example

- Using a car versus fixing one.
  - Only need to understand that the gas petal on a car is pressed down to make the car go faster – if all you want to do is drive the car.
  - If the goal is to fix a sticky accelerator then you need to look under the hood and (maybe) in a technical manual.

"You don't have to know how to rebuild a motor to drive a car"

Complexity of Model

The accuracy and complexity of the model depends on the task or usage scenario

A relatively simple mental model of an automobile is needed for driving

A complex and accurate mental model of an automobile is necessary to repair or build one

Mental Models of Software

For many years researchers have tried to understand how programmers comprehend programs (software) during:

- Software Development
- Software maintenance/evolution

Novice versus Expert

What Purpose Does a Mental Model Serve?

Mental models allow researchers a way to analyze the cognitive processes behind software development and maintenance

What Makes up a Mental Model?

- static elements
- dynamic elements

Static Elements

- Text Structure
- Chunks
- Plans
- Hypotheses
- Beacons
- Rules of Discourse

Text Structure

The program text and its structure

- if-then-else
- loops
- variable definitions
- parameter definitions

Chunks

Knowledge structures containing different levels of abstractions of text structures.

- macro-structure
- micro-structure

Plans

Knowledge elements for developing and validating expectations, interpretations, and inferences. They correspond to a vocabulary of intermediate level programming concepts such as a counter. An average plan would include a counter plan.

Hypothesis

Conjectures that are results of comprehension activities that can take seconds or minutes to occur. They are drivers of cognition. They help to define the direction of further investigation.

- why
- how
- what

Beacons

Signals that index into knowledge. An example of a beacon is a swap. It has been proven experienced programmers recall beacon lines much faster than novice programmers. They are used most commonly in top-down comprehension.

Rules of Discourse

Rules that specify the conventions in programming.They set the expectations of the programmer. Examples:

- Variables should reflect function
- Don’t include text that won’t be used
- If there is a test for a condition, the condition should have the potential to be true.

Dynamic Elements

- Strategies
- Actions
- Episodes
- Processes

Strategies

A sequence of actions that lead to a particular goal.

– opportunistic strategy
– systematic strategy

Actions

Classify programmer activities implicitly and explicitly during a specific maintenance task.

Episodes

Are made up of a sequence of actions.

Processes

An aggregation of episodes.

Maintenance Tasks

- adaptive
- perfective
- corrective
- reuse
- code leverage

Adaptive

- Understand the system
- Define requirements
- Develop preliminary and detailed design
- Code changes
- Debug
- Regression tests

Perfective

- Understand the system
- Diagnosis and requirement definition for improvements
- Develop and design preliminary design
- Code changes and/or additions
- Debug
- Regression tests

Corrective

- Understand the system
- Generate and/or evaluate hypotheses concerning the problem
- Repair the code
- Regression tests

Reuse

- Understand the problem, find solution based on close fit with predefined components
- Obtain predefined the components
- Integrate predefined components

Code Leverage

- Understand the problem, find solution based on predefined components
- Reconfigure solution to increase likelihood of using predefined components
- Obtain and modify predefined components
- Integrate modified components

Mental Model

The type of mental model a programmer uses is determined by the type of development/ maintenance task he has to perform.

Proposed Mental Models

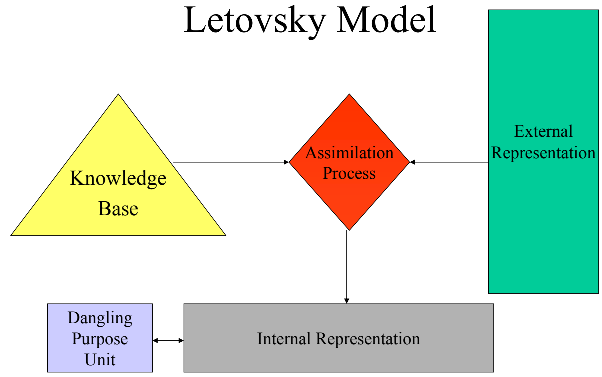
- Letovsky ‘86
- Shneiderman ’79, ‘80
- Brooks ’77, ‘83
- Soloway / Ehrlich ’83, ’84, ‘88
- Pennington ‘87
- Integrated (Von Mayrhauser ’94, ’95, ’97)

Letovsky Model

Opportunistic approach. This model has three main parts:

- knowledge base
- mental model
- assimilation process (bottom-up/top-down)

Letovsky Model

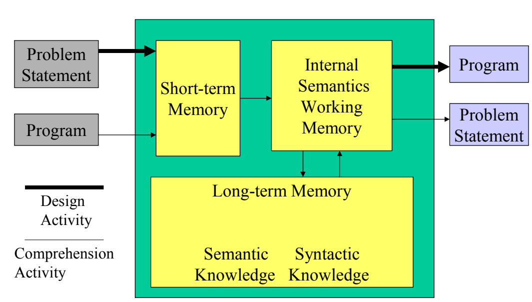


Shneiderman Model

The main parts of this model are:

- short-term memory (uses chunking)
- internal semantics (working memory)
- long-term memory

Shneiderman Model

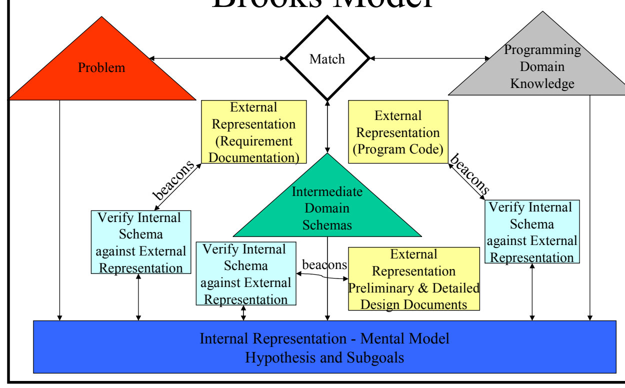


Brooks Model

Top-down model.This model uses:

- hypotheses
- beacons

Brooks Model

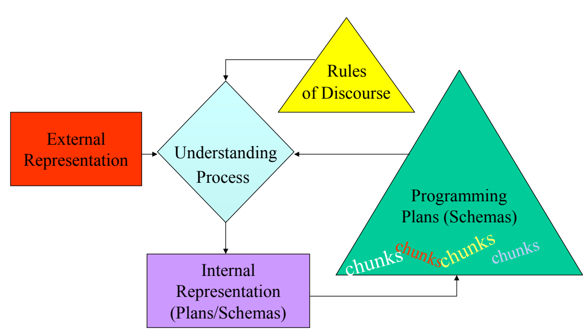


Soloway / Ehrlich Model

Top down approach. Also known as *domain model.* This model uses:

- plans
- rules of discourse
- chunks

Soloway / Ehrlich Model

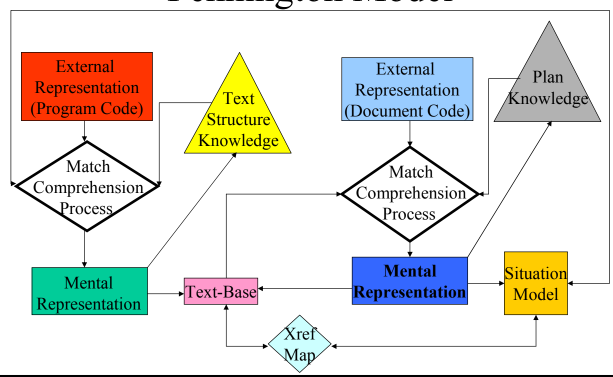


Pennington Model

Bottom-up approach. This model uses:

- beacons
- text structures
- chunks
- plans

Pennington Model

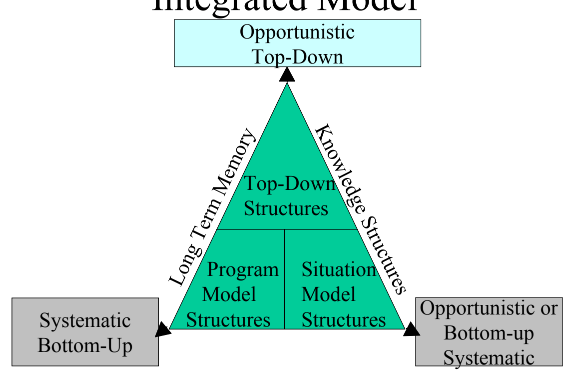


Integrated Model

Top-down, bottom-up approach. This model contains the following:

- top-down model
- bottom-up model
- program model
- knowledge base

Integrated Model



Common Elements of Mental Model

- Knowledge
  - general knowledge
  - software specific knowledge

Comparison of the Six Models

- Letovsky Model - general
- Shneiderman Model - hierarchical organization
- Brooks Model - hypothesis driven
- Soloway / Ehrlich Model - knowledge similar to Letovsky Model
- Pennington Model - detailed, lacks higher level knowledge
- Integrated Model - combination of the other 5 models.

Conclusion

It is important to learn how programmers understand code. This could lead to better tools, better maintenance guidelines and documentation.