
Program Comprehension

Program Comprehension
During Software Maintenance and Evolution
IEEE Computer, August 1995
Von Mayrhauser, A. and Vans, A. M.

Outline

- Overview
- Mental models
- Cognitive models
- Tools
- Research topics

Program comprehension

- Program comprehension is the study of how software engineers understand programs.
- Program comprehension is needed for:
 - Debugging
 - Code inspection
 - Test case design
 - Re-documentation
 - Design recovery
 - Code revisions

Program comprehension process

- Involves the use of existing knowledge to acquire new knowledge about a program.
- Existing knowledge:
 - Programming languages
 - Computing environment
 - Programming principles
 - Architectural models
 - Possible algorithms and solution approaches
 - Domain-specific information
 - Any previous knowledge about the code
- New knowledge:
 - Code functionality
 - Architecture
 - Algorithm implementation details
 - Control flow
 - Data flow

Comprehension techniques

- Reading by step-wise abstraction
 - Determine the function of critical subroutines, work through the program hierarchy until the function of the program is determined.
- Checklist-based reading
 - Readers are given a checklist to focus their attention on particular issues within the document.
 - Different readers were given different checklists, therefore each reader would concentrate on different aspects of the document.
- Defect-based reading
 - Defects are categorized and characterized (e.g., data type inconsistency, incorrect functionality, missing functionality, etc.)
 - A set of steps (a scenario) is then developed for each defect class to guide the reader to find those defects.
- Perspective-based reading
 - Similar to defect-based reading, but instead of different defect classes, readers have different roles (tester, designer and user) to guide them in reading.

Sources of variation

- Aside from the issue of how comprehension occurs, comprehension performance and effectiveness are affected by many factors:
 - Maintainer characteristics
 - Program characteristics
 - Task characteristics

Maintainer characteristics

- Familiarity with code base
- Application domain knowledge
- Programming language knowledge
- Programming expertise
- Tool expertise
- Individual differences

Program characteristics

- Application domain
- Programming domain
- Quality of problem to be understood
- Program size and complexity
- Availability and accuracy of documentation

Task characteristics

- Task type
 - Experimental: recall, modification
 - Perfective, corrective, adaptive, reuse, extension.
- Task size and complexity
- Time constraints
- Environmental factors

Outline

- Overview
- **Mental models**
- Cognitive models
- Tools
- Research topics

Models

- Mental models
 - Internal working representation of the software under consideration.
- Cognitive models
 - Theories of the processes by which software engineers arrive at a mental model.



Mental models

- Static elements
 - Text structure knowledge
 - Microstructure
 - Chunks (macrostructure)
 - Plans (objects)
 - Hypotheses
- Dynamic elements
 - Strategies (chunking and cross-referencing)
- Supporting elements
 - Beacons
 - Rules of discourse

Text structure

- The program text and its structure
 - Control structure: iterations, sequences, conditional constructs
 - Variable definitions
 - Calling hierarchies
 - Parameter definitions
- Microstructure – actual program statements and their relationships.

Chunks

- Contain various levels of text structure abstractions.
- Also called macrostructure.
- Can be identified by a descriptive label.
- Can be composed into higher level chunks.

The Sort Chunk	
microstructure	macrostructure
Input: array a Result: array in sorted order for i = 0 to n-1 for j = i to n-1 if a[i]>a[j] swap(a[i],a[j])	sort(list)

Plans (objects)

- Knowledge elements for developing and validating expectations, interpretations, and inferences.
- Include causal knowledge about information flow and relationships between parts of a program.
- Programming plans
 - Based on programming concepts.
 - Low level: iteration and conditional code segments.
 - Intermediate level: searching, sorting, summing algorithms; linked lists and trees.
 - High level
- Domain plans
 - All knowledge about the problem area.
 - Examples: problem domain objects, system environment, domain-specific solutions and architectures.

Hypotheses

- Conjectures that are results of comprehension activities that can take seconds or minutes to occur.
- Three types:
 - Why – hypothesize the purpose/rationale of a function or design choice.
 - How – hypothesize the method for accomplishing a certain goal.
 - What – hypothesize classification.
- Hypotheses are drivers of cognition. They help to define the direction of further investigation.
- Code cognition formulates hypotheses, checks them whether they are true or false, and revises them when necessary.
- Hypotheses fail for several reasons:
 - Can't find code to support a hypothesis.
 - Confusion due to one piece of code satisfying different hypothesis.
 - Code cannot be explained.

Supporting elements

- Beacons
 - Cues that index into existing knowledge.
 - A swap routine can be a beacon for a sorting function.
 - Experienced programmers recognize beacons much faster than novice programmers.
 - Used commonly in top-down comprehension.
- Rules of discourse
 - Rules that specify programming conventions.
 - Examples: coding standards, algorithm implementations, expected use of data structures.

Mental models – dynamic elements

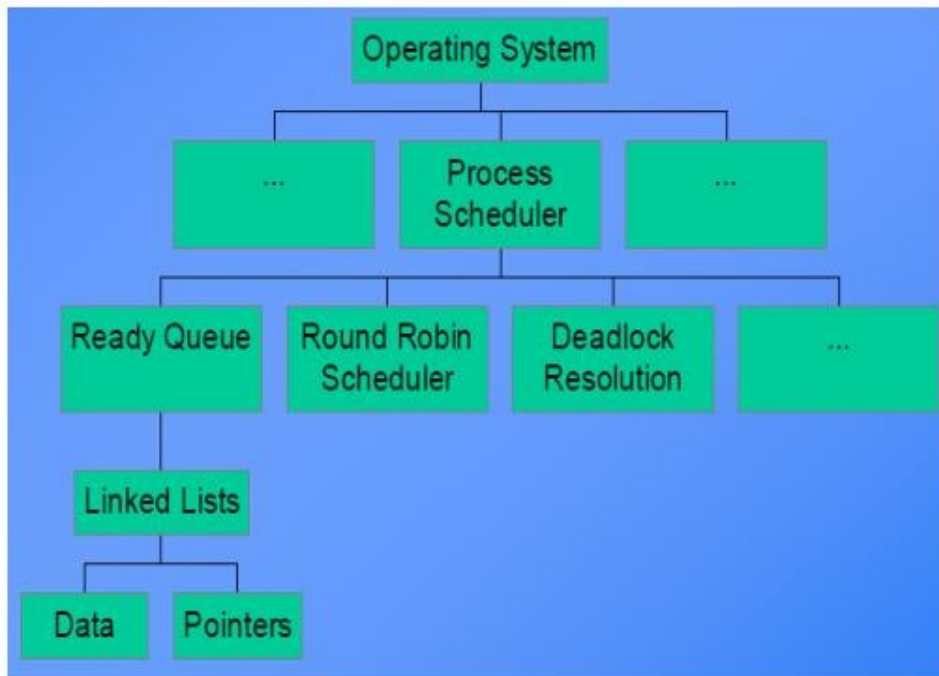
- Strategies
 - Sequences of actions that lead to a particular goal.
- Actions
 - Classify programmer activities implicitly and explicitly during a maintenance task.
- Episodes
 - Sequences of actions.
- Processes
 - Aggregations of episodes.

Strategies

- Guide the sequence of actions while following a plan to reach a goal.
- Match programming plans to code.
 - Shallow reasoning – do not perform in-depth analysis; stop upon recognition of familiar idioms and programming plans.
 - Deep reasoning – perform detailed analysis.
- Mechanisms for understanding
 - Chunking
 - Cross-referencing

Chunking

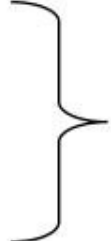
- Creates new, higher-level abstraction structures
- Labels replace the detail of the lower level chunks.



Cross-referencing

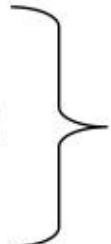
- Map program parts to functional descriptions

```
temp = a;  
a = b;  
b = temp;
```



swap

```
for (i=0; i<size; i++)  
    if (array[i]==target)  
        return true;
```



sequential search

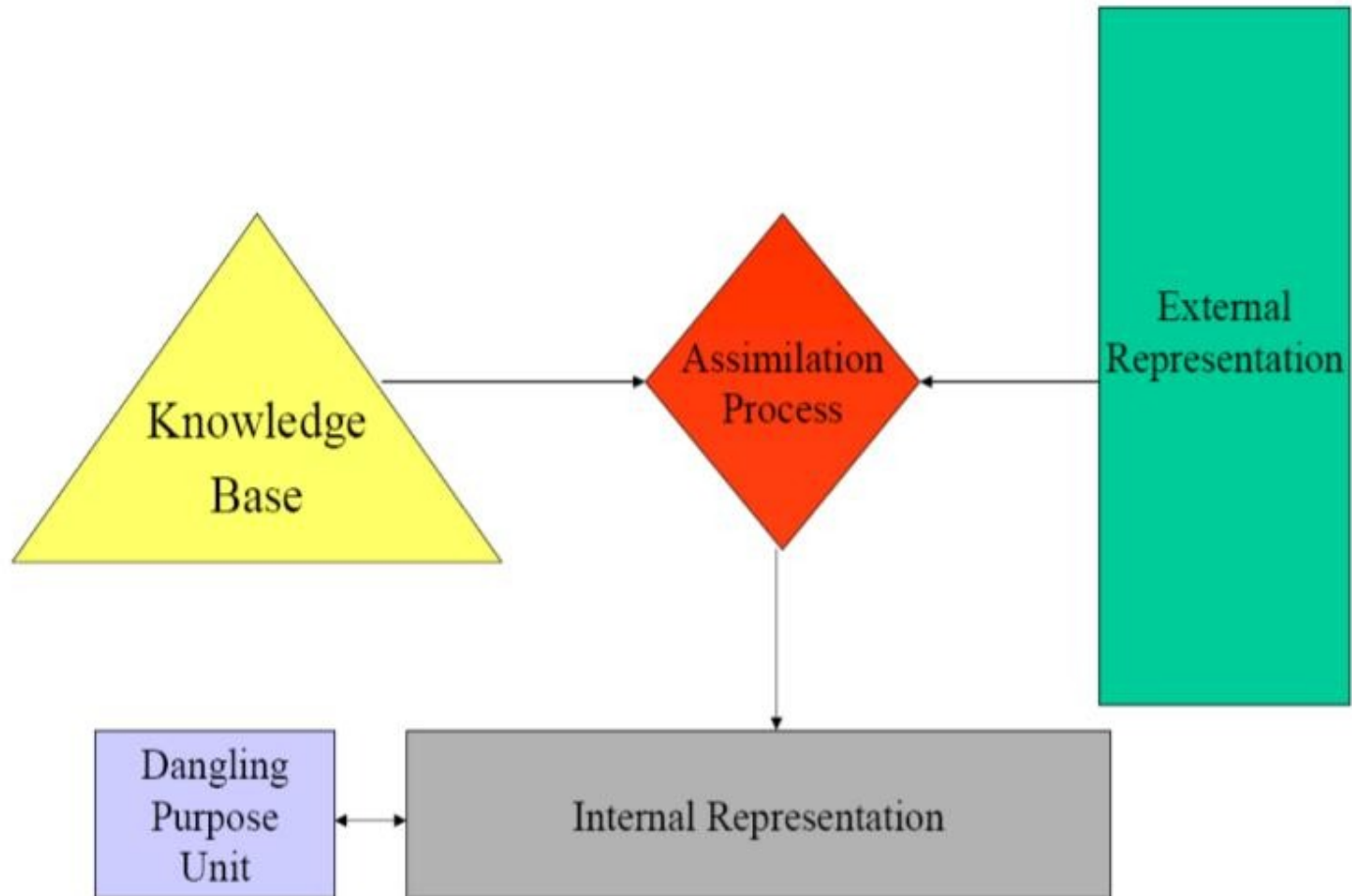
Outline

- Overview
- Mental models
- **Cognitive models**
- Tools
- Research topics

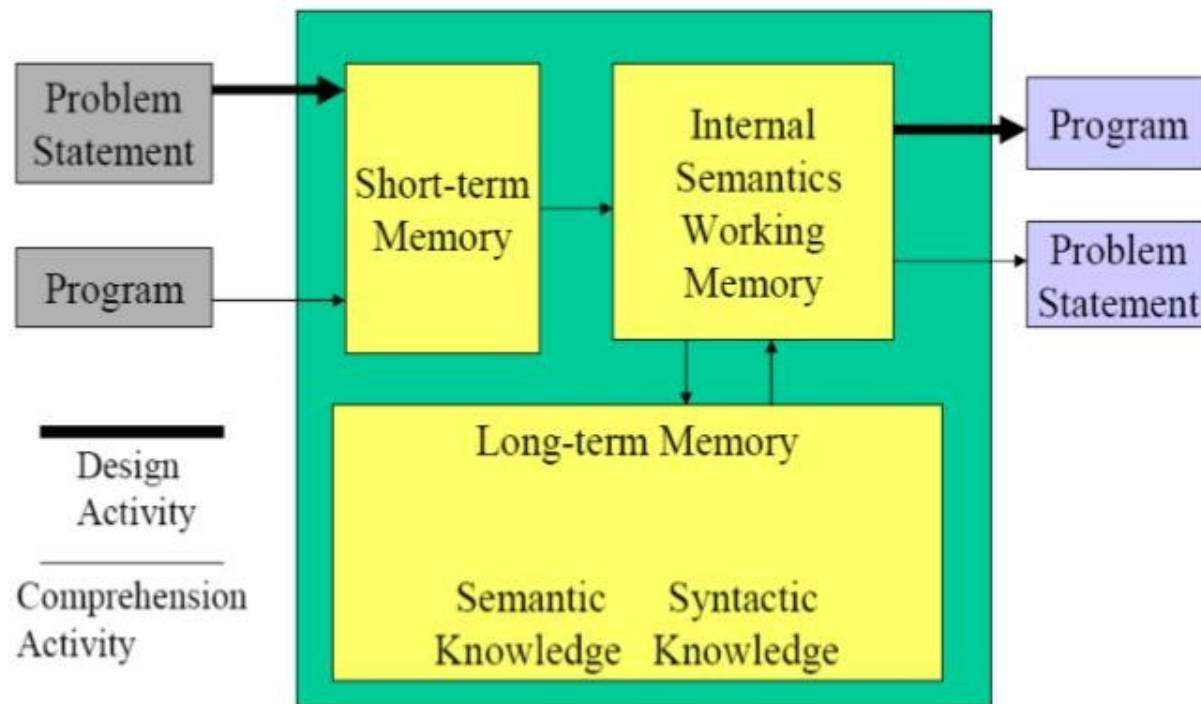
Cognitive models

- Letovsky
- Shneiderman and Mayer
- Brooks
- Soloway, Adelson and Ehrlich
- Pennington
- Mayrhauser and Vans (Integrated)

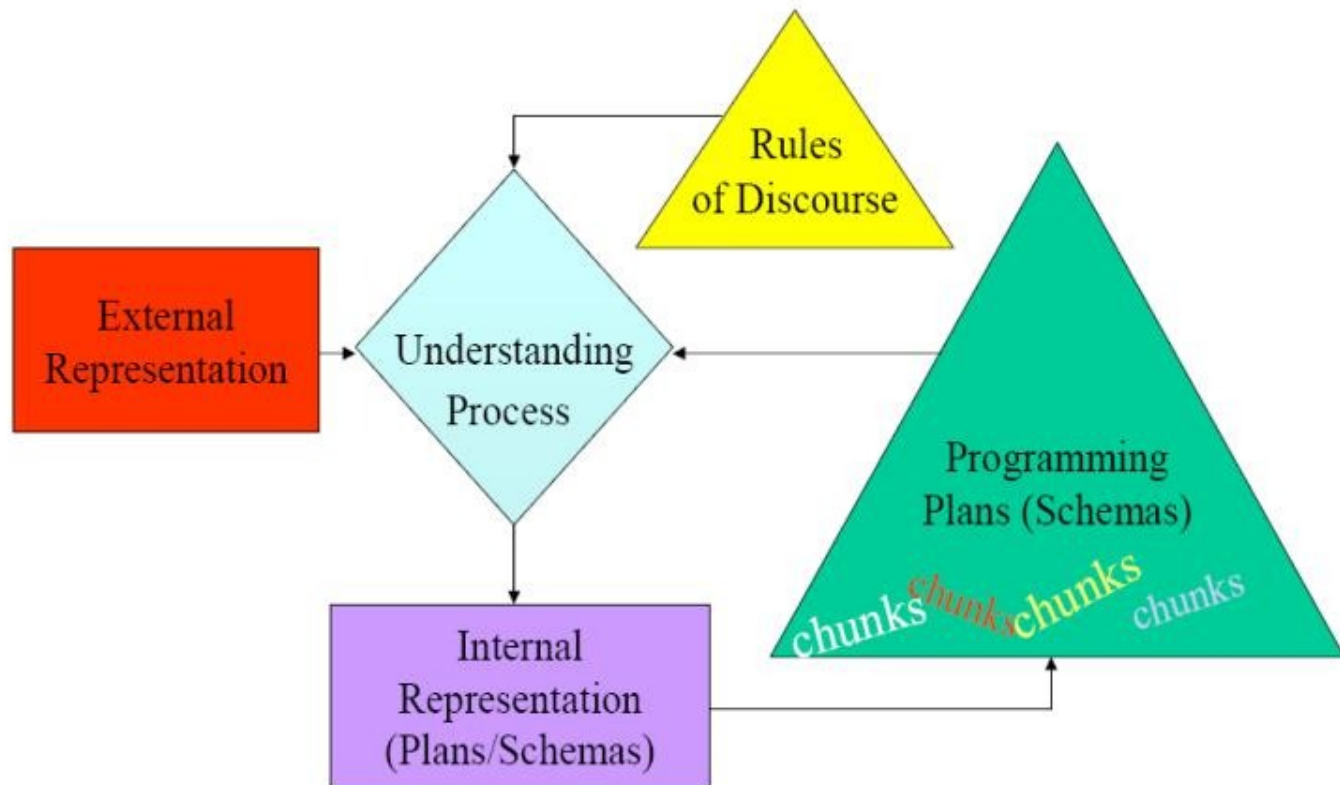
Letovsky model



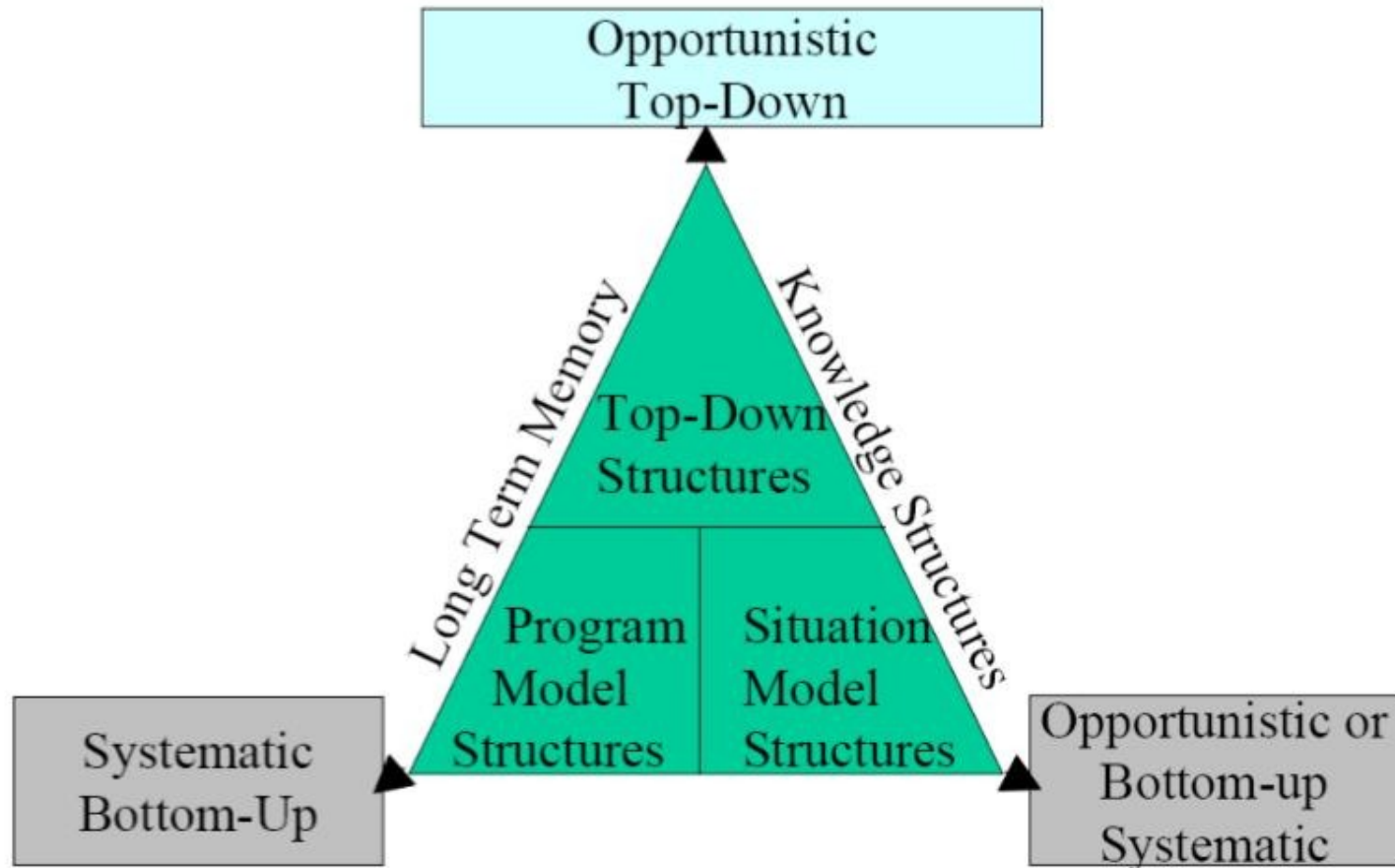
Shneiderman model



Soloway model



Integrated model



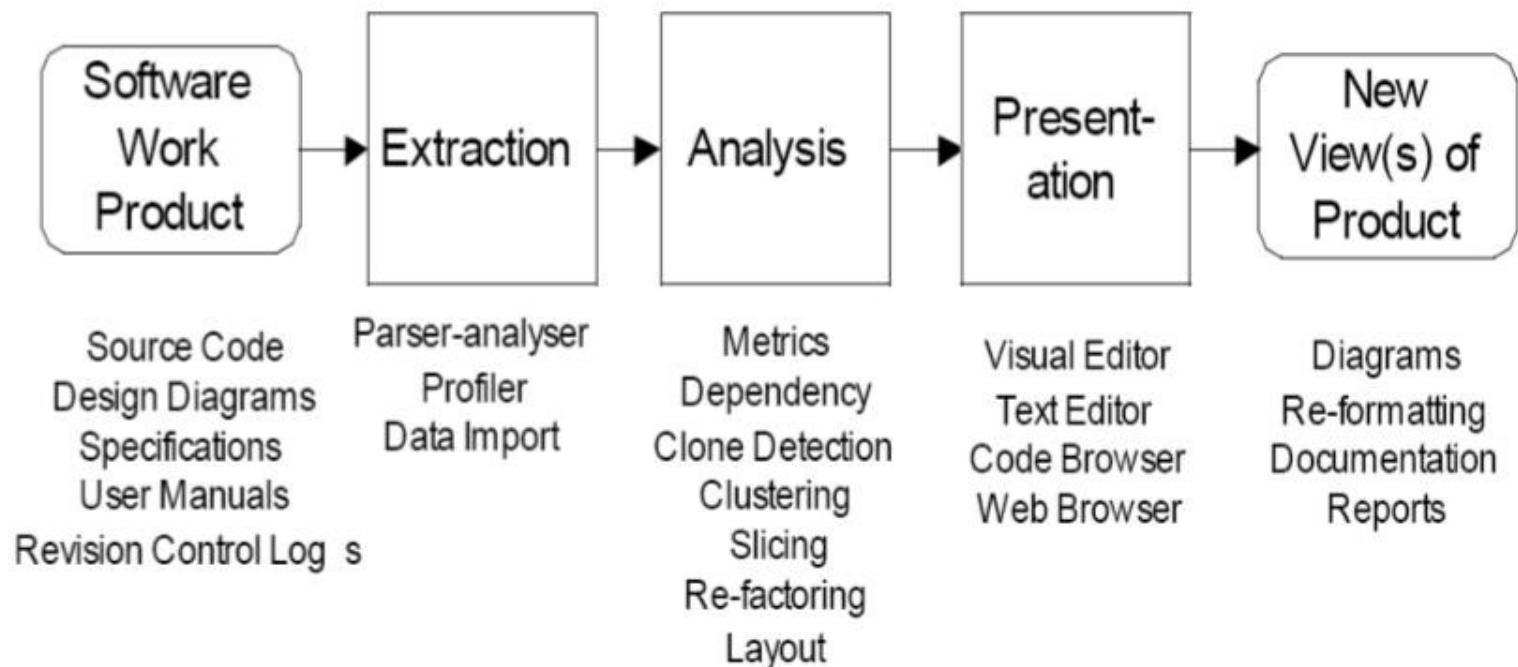
Distributed cognition

- Traditional cognitive models deal the cognitive processes inside one person's brain.
- On real projects, software developers:
 - Work in teams
 - Can ask people questions
 - Can surf the web for answers
- How do these affect the cognitive process?

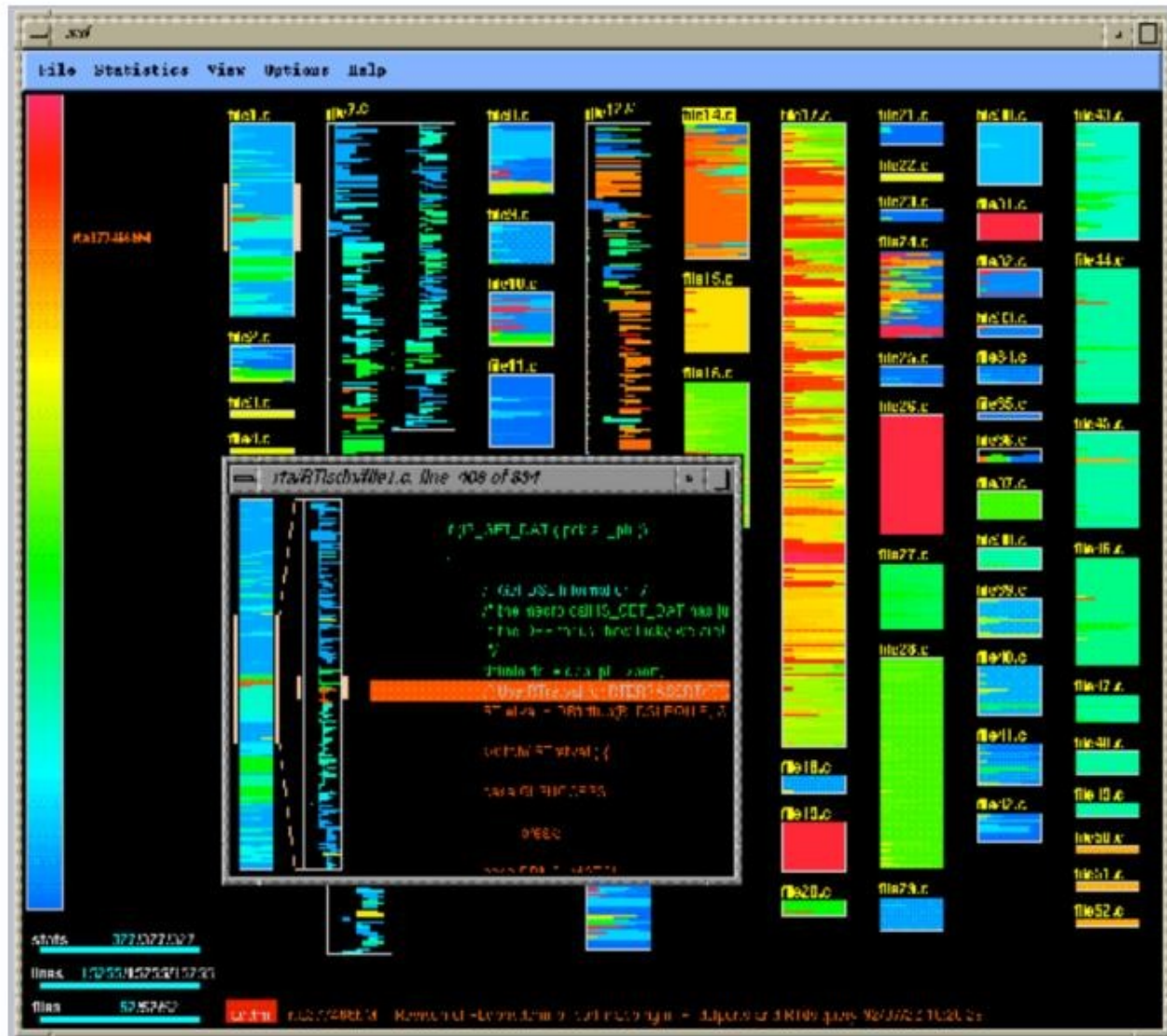
Outline

- Overview
- Mental models
- Cognitive models
- **Tools**
- Research topics

Generic tool pipeline



Code browser example: SeeSoft



Outline

- Overview
- Mental models
- Cognitive models
- Tools
- **Research topics**

Research topics

- Empirical studies of cognitive models
 - Larger systems (multiple programmers)
 - Modern architectures
 - Borrow techniques form psychology
 - Use of recognition and recall as dependent variables
- Tool support for comprehension tasks
 - Information foraging
 - Automated suggestions for program investigation
 - Text mining application to code exploration
 - Source code analysis
 - Support for newer languages

Additional references

- Susan Elliott Sim. **Research in Program Comprehension** (UC Irvine lecture notes).
- Jonathan Maletic. **Program Comprehension & The Psychology of Programming** (Kent State University lecture notes).
- Richard Upchurch. **Program Comprehension.** From Software Process Resource Collection. UMass – Dartmouth, 1996.
<http://www2.umassd.edu/swpi/1docs/comprehension.html>.