Research in Program Comprehension

Susan Elliott Sim
ses@ics.uci.edu
November 7, 2006

Overview

• Program Comprehension

• People Comprehending Programs
  – Strategies
  – Models
  – Factors

• Tools to Help People Comprehend Programs
  – Generic Pipeline
Program Comprehension

- Help programmers understand code more quickly
  - Improve general understanding
  - To complete a task
- Examples of program comprehension tools?

- Related fields: reverse engineering, source code analysis, software visualization

People Comprehending Programs

- Models
- Strategies
- Factors
Studies of Program Comprehension

- Cognitive models
  - A single programmer working alone to understand a program
    - Mental model is the representation created by a programmer during the process of understanding
- Social and distributed cognition
  - A programmer working as part of a team to develop or maintain a software system
- Frameworks for designing program comprehension tools

Cognitive Models

- Bottom-up
- Top-down
- Hybrid
  - Systematic and As-Needed
  - Knowledge-based
  - Integrated
Bottom-Up Models

• Build larger abstractions from details
  – *Chunking* is the process of building larger units
• Allows programmer to hold more information in short-term memory
  – Miller's magic number: $7 \pm 2$
• Larger chunks corresponds to more meaningful understanding of larger parts of the program

  – Lexical -> Syntactic -> Semantic
  – Program model -> situation model

Top-Down Models

• Features in the program are mapped onto expectations
  – Hypotheses are formed at higher levels and direct investigation of details
  – Expectations are called schemas or plans

  – Programmers look for beacons in the code
  – Two types of programming knowledge: programming plans and rules of programming discourse
Hybrid Models

- Programmers don’t use just one approach all the time.
  - Can choose an appropriate one for the task
- Switch between them during a single task

Systematic and As-Needed Models

- **Systematic** = read the code methodically, tracing control flow and data flow
  - Provides static (structural) and causal information
- **As-needed** = focus only on the code relevant to immediate task
  - Also called Just In Time Comprehension
  - Provides only static information (weaker mental model)
Merging Systematic and As-Needed

- The two strategies were later merged into a single model


- Micro-strategies
  - Inquiry episodes: read, question, conjecture, search
- Macro-strategies
  - Systematic
  - As-Needed

Knowledge-Based Models

- Programmers are opportunistic processors; they use what is available to get the job done.


- Comprehension is a goal-oriented, hypothesis-driven, problem solving process


- Inquiries = asking questions, conjecturing answers, verifying answers
- Three Components in Model: Knowledge Base, Assimilation Process, Mental Model
Integrated

- The Integrated Metamodel

Sources of Variation

- Aside from the issue of how comprehension occurs, researchers agree that programmer performance is affected by a number of 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 of documentation
Task Characteristics

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

Social and Distributed Cognition

- On typical industrial projects, software developers:
  - Work in teams
  - Have access to documentation
  - Can ask people questions
  - Can surf the web for answers
  - Can go home and think about the problem
  - Learn!
- Work on cognitive models don’t really address program comprehension as a open system
**Distributed Cognition**

- Cognition is an interactive process between a person and the environment
  - Plans are incomplete
  - External world is used to help cognitive process
  - Examples: wayfinding, GUI desktops, to do lists
  - See work in HCI by Hutchins and by Suchman

  - Developed a distributed cognition model for program comprehension
  - Created a framework for designing and evaluating tools

---

**Teams**

  - On large software projects, it takes many months to become a contributing team member

  - A significant part of learning about the system is identifying expertise and ownership

  - While comprehension is a critical part of programming, software engineers actually spend a minority of their time coding
Work Practices

• An approach more than a framework
• Study work practices in order to tailoring tools (or methods) to local organization and environment
• Example: source code searching with grep
  – Why do people use it?
  – What does it do well? (Keep these features)
  – What does it not do well? (Ideas for improvements)
  – Result: SEE (Software Exploration Environment)
  – Does the resulting tool fit with work practices?


Research Opportunities

• New research is needed on larger systems, with modern architectures, involving multiple programmers
  – Past research conducted on small programs
    • A few hundred or thousands of lines of code
  – Single programmer
• Past studies did not use right mix of psychology and software engineering
  – Example: use of recognition and recall as dependent variable
  – Example: Integrated metamodel
Recent Research

- Navigation through source code
  - Information foraging
    - Developers spent on average 35% of their time performing the mechanics of finding information
  - Graph-based model of traversal
    - In a web application, the layered architecture graph is an effective model for developer’s navigation behavior
    - Developers who successfully completed a perfective maintenance task made fewer traversals and were more likely to make traversals of distance 1

Tools to Help People Comprehend

- Generic Pipeline
- Environments
Analyses in Reverse Engineering

- Assumption: Legacy code
- Redocumentation
- Design Recovery
- Restructuring
- Reengineering
Things to Notice

- Architecture influenced by UNIX and compilers
- Graph-based representation
  - Take a graph theory course
- Tools, analyses, and representations that are useful for other areas of software engineering
- Each element in the diagram (including the arrows) is a research problem

Generic Pipeline

Software Work Product
  Source Code Design Diagrams Specifications User Manuals Revision Control Logs

Extraction
  Parser-analyser Profiler Data Import

Analysis
  Metrics Dependency Clone Detection Clustering Slicing Re-factoring Layout

Presentation
  Visual Editor Text Editor Code Browser Web Browser

New View(s) of Product
  Diagrams Re-formatting Documentation Reports
Fact Extraction for C++

```cpp
1 template <class T, int Size>
2 class Array
3 {
4    T arr[Size];
5 public:
6    virtual const T& get(int idx)
7    {
8      T& t = arr[idx];
9      return t;
10    }
11    /* set()...*/
12    
```
Generic Pipeline

Analysis Tools

- Clustering
- Program summaries, fingerprints
- Design pattern detection
- Slicing
Slicing

• Program analysis technique for reducing a program
  – Eliminates all lines that are not currently of interest
• Choose a variable, choose a line of code, slice forward (or backward)

Backward Slice

```c
void main () {
  int i = 1;
  int sum = 0;
  while (i<11) {
    sum = add(sum, i);
    i = add(i, 1);
  }
  printf("sum = %d\n", sum);
  printf("i = %d\n", i);
}
static int add(int a, int b) {
  return(a+b);
}
Backward slice from printf("i = %d\n", i);
```
Backward Slice

```c
void main () {
    int i = 1;
    int sum = 0;
    while (i<11) {
        sum = add(sum, i);
        i = add(i, 1);
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
static int add(int a, int b) {
    return(a+b);
}
```

Forward slice from `printf("i = %d\n", i);`

Forward Slice

```c
void main() {
    int i = 1;
    int sum = 0;
    while (i<11) {
        sum = add(sum, i);
        i = add(i, 1);
    }
    printf("sum = %d\n",sum);
    printf("i = %d\n", i);
}
static int add(int a, int b){
    return(a+b);
}
```

Forward slice from `sum = 0;`
void main() { 
    int i = 1;
    int sum = 0;
    while (i<11) {
        sum = add(sum, i);
        i = add(i, 1);
    }
    printf("sum = %d\n", sum);
    printf("i = %d\n", i);
}
static int add(int a, int b){
    return(a+b);
}
Forward slice from sum = 0;

**Generic Pipeline**

- **Software Work Product**: Source Code, Design Diagrams, Specifications, User Manuals, Revision Control Logs
- **Extraction**: Parser-analysers, Profiler, Data Import
- **Analysis**: Metrics, Dependency, Clone Detection, Clustering, Slicing, Re-factoring, Layout
- **Presentation**: Visual Editor, Text Editor, Code Browser, Web Browser
- **New View(s) of Product**: Diagrams, Re-formatting, Documentation, Reports
Presentation Tools

- SNiFF+
- Seesoft™
- SHriMP/Creole
- PBS (Portable Bookshelf)
PBS

Creole
### Standard Exchange Format

- **GXL (Graph eXchange Language)**
  - An XML sub-language for exchanging graphs
    - Joint work by Sim, Andreas Winter, Ric Holt, Andy Schürr
    - Mechanism for data interoperability between reverse engineering, reengineering, and graph transformation tools
    - In use by ~40 research groups around the world
    - Similar to MDA currently promoted by OMG
GXL Features

- Labeled, typed graph model
  - Nodes, (directed) edges, hyperedges, attributes
  - Unique identifiers on nodes, optional identifiers on others
- Transmit schema along with data
  - Use single DTD for both
- Provides a common syntax for data interchange, so problem shifts to schema interchange

```
<?xml version="1.0"?>
<!DOCTYPE gxl SYSTEM "gxl.dtd">
<gxl>
  <graph id="schema">
    ...
  </graph>
  <graph id="instance">
    ...
  </graph>
</gxl>
```

Schemas

- Converting from one syntax to another is relatively easy
- Mapping between schemas is the most difficult aspect of conversion
  - GXL helps by making them explicit
  - More tools are needed to support schema exchange
- Related to research into model management in databases
  - Idea: translating data is isomorphic to translating between their respective schemas
Reference Schemas

- A canonical schema for a particular domain or task
  - Examples: graph drawing, C++ fact extraction
- Contains an agreed-upon set of modeling entities and relations
  - Instead of writing mappings between every local schema, write a mapping between local and reference schema

Research Opportunities

- Programming Environments
  - UNIX pipelines vs. Eclipse
  - New tools, new opportunities
- Languages
  - Previous: COBOL, PL/I
  - Current emphasis: Java
  - Future? Scripting languages, JSP, Visual Basic, C#
- Systems
  - Web applications, plug-ins, games
- Mining Software Repositories
- Applications of Reverse Engineering