Welcome to SENG 371 Software Evolution Spring 2013

A Core Course of the BSEng Program

Hausi A. Müller, PhD PEng
Professor, Department of Computer Science
Associate Dean Research, Faculty of Engineering
University of Victoria

Announcements

- Lab attendance
  - Has been a problem as of late — needs to change
  - Several questions on labs on final exam
- Final exam
  - Sat, April 13 — 7:00 -10:00 pm
- Marking
  - Midterm and A1 graded
  - Marks posted
- Course website
  - http://www.engr.uvic.ca/~seng371
  - Lecture notes posted
  - Lab slides and activities are posted
- Assignment 3
  - Due Thu, April 4
  - Cite your sources
  - Submit by e-mail to seng371@uvic.ca

2

Reading Assignment

- Murphy, Notkin, Lan: An empirical study of static call graph extractors, *ACM Transactions on Software Engineering and Methodology (TOSEM)* 7(2):158-191 (1998)
  - http://dl.acm.org/citation.cfm?id=279314
- Müller, Jahnke, Smith, Storey, Tilley, Wong: Reverse Engineering: A Roadmap, in *The Future of Software Engineering*, pp. 47-60 (2000)
  - http://dl.acm.org/citation.cfm?id=336526
- Storey: Theories, tools and research methods in program comprehension: past, present and future, *Software Quality Journal* 14:187-208 (2006)
  - http://webhome.cs.uvic.ca/~chisel/pubs/storey-pc-journal.pdf
- Brown, Malveau, McCormick III, Mowbray: *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley (1998)
- AntiPatterns Tutorial and Website
  - http://www.antipatterns.com/briefing/index.htm
  - http://www.antipatterns.com

3

Views: Separation of concerns

- Interfaces
- APIs
- COTS, middleware
- Scripting layers
- Extensibility, genericity
- Event handling
- Performance
- Platforms, product lines
- Persistence, storage
- Electrical diagrams
- Plumbing diagrams
- Perspective views
- Front, top, side views
- Levels of indirection
- Autonomic manager versus managed element

4

Views: Separation of Concerns

- UI, DB, algorithms
- Data structures
- Fire walls
- Security architecture
- Middleware components
- Different platforms (i.e., OS)
- Platform dependent/independent parts

5

Views: Design Patterns

- Iterator
- Wrapper façade
- Monitor
- Event handling patterns
- Mediator
- Collection, container
- MVC (Model, View, Controller)
- Serialization
- Exception, error handling
- Algorithms and data structures

6

Views: Architecture Patterns

- Architectural styles
- Event driven architecture (event handling)
- Pipes and filters
- Publish – subscribe

7

Program understanding Learning objectives

- Learn different models of program understanding
- Understand implications of the models on how we write programs and how we use and design maintenance tools

Program understanding

- What strategies do you follow when trying to understand a program written by someone else?
- Describe the kinds of information you use to arrive at an understanding of how it works.

9

Overview

- Program comprehension models
  - Bottom up
  - Top down
  - Integrated meta-model
  - Opportunistic, Systematic etc.
- Theories about tool support
  - Cognitive support
  - Improving flow

10

Different kinds of models

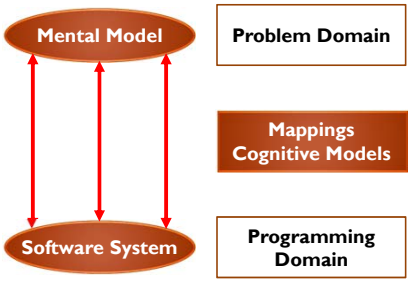
- A mental model describes the maintainer's mental representation of the program to be understood
- A cognitive model describes the processes and information structures used to form the mental model

11

What does this code remind you of?

```
m := (x + y) div 2;
```

Developing mental models using cognitive models



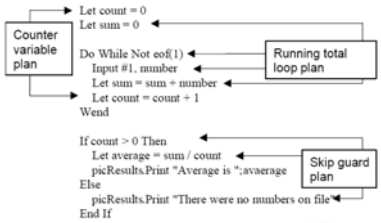
13

Terminology (1)

- Beacons
  - Recognizable or familiar features in the code that act as cues to the presence of certain structures
    - Swapping of two variables is a beacon for a sort routine
- Plans
  - Knowledge elements for developing and validating expectations, interpretations and inferences
  - Programming plans and domain plans
  - Often referred to as clichés, schemas, idioms
  - Low-level pattern
  - Slot types (generalized templates) and slot fillers (specific solutions)

14

Examples of programming plans, clichés, schemas, or slices



- Reading input
- Counting input
- Running total
- Computing average
- Handling exceptions

Slot types and fillers (plans)

- An example of a slot type could be a function such as a sort routine
- A slot filler could be a particular implementation of a sort routine, for example quicksort
- Slot fillers are related to slot types via either a *Kind-of* or an *Is-a* relationship.

Terminology (2)

- Rules of discourse
  - Rules or conventions of programming
    - naming standards, indentation, white space, in-line documentation standards, exception handling style, use of include files
  - Can be imposed by programming language
    - Python
  - Rules of discourse set up expectations in the mind of programmer
- Cross referencing
  - Relates different abstraction levels such as a control flow and functional view by mapping program parts to functional descriptions

17

cxref command

- Analyzes C files and builds a cross-reference table
- Uses a special version of cc to include #define'd information in its symbol table.
- Generates a list of all symbols (auto, static, and global) in each individual file.
- Includes four fields: NAME, FILE, FUNCTION, LINE
  - The line numbers appearing in the LINE field also show reference marks as appropriate. The reference marks include:
    - assignment =
    - declaration –
    - definition *
    - general reference <no mark>

18

Cognitive models of program comprehension

- Bottom-up comprehension
- Top-down comprehension
- Knowledge based understanding model
- Systematic and as-needed strategies
- Integrated meta-model of program comprehension

19

What does this piece code do?

```
maxValue := table[0];
for k := 1 to MAXINDEX do
  if table [k] > maxValue then
      maxValue := table [k]
  end
end
```

- Experts do this much faster than novices, so bottom up comprehension is much more successful for experts than novices.

Bottom-up comprehension (1)

- Starts understanding from the source code, constructing higher level abstractions using chunking and concept assignment
  - Shneiderman and Mayer 79
  - Pennington 87
  - Biggerstaff, et al. 93
- Chunking creates new higher level abstractions from lower level structures
- When higher level structures are recognized, they replace more detailed lower level ones
- This helps to overcome the limitations of the human memory when confronted with too many pieces of information

21

Bottom-up comprehension (2)

This theory suggests that programmers understand programs by reading the source code and documentation, and mentally chunking this information into progressively larger chunks until an understanding of the entire program is achieved, uses both syntactic and semantic knowledge

- Chunks are syntactic or semantic mental abstractions of text structures within the source code
- Syntactic knowledge: language syntax, available library routines
- Semantic knowledge: general and task related

Proposed by Shneiderman & Mayer

22

Bottom-up comprehension (3)

Pennington also proposed a bottom up model and suggests that maintainers first develop a program model, and then a situation model:

- Program model
  - Based on control flow abstractions
  - Developed when code is completely new to programmers
  - Developed bottom up via beacons – identification of code control primes in the program
- Situation model
  - data flow and functional abstractions
  - Also developed bottom-up – requires knowledge of real world domains (domain plans)
  - Cross referencing is used to arrive at the overall program goal

23

Top-down comprehension (1)

- Tries to reconstruct the mappings from the problem domain into the programming domain that were made when programming the system
  - Brooks 83
  - Soloway and Ehrlich 84
- Reconstruction is expectation-driven
  - Understanding starts with some pre-existing hypotheses about the functionality of the system and the engineer investigates whether they hold, should be rejected, or refined in a hierarchical way

24

Top-down comprehension (2)

- According to Brookes
  - Programmer develops a hierarchy of hypotheses
  - Make heavy use of beacons (cues)
  - Understanding is complete when a complete set of mappings can be made from the problem domain to the programming domain

25

Examples of beacons

- Internal to the program:
  - Prologue comments
  - Variable, method, procedure, package names
  - Data declarations
  - In-line comments
  - Subroutine or file structure
  - I/O formats
  - XML schemas
- External to the program
  - User manuals
  - Cross reference listings
  - Documentation

26

Top-down comprehension (3)

According to Soloway & Ehrlich

- Three types of programming plans:
  - Strategic plans – describe a global strategy, domain independent
  - Tactical plans – local strategies for solving a problem, language independent
  - Implementation plans – how to implement tactical plans, language dependent – may contain code fragments
- Rules of programming discourse and beacons are used to decompose plans into lower level plans
- Delocalized plans are plans which are implemented in a distributed manner throughout the program and complicate program comprehension
- Separation of concerns, aspects oriented programs

27

Opportunistic approach

- There is no such thing as a pure top-down or pure bottom-up approach
- To create mental representations of the software system programmers frequently change between top-down and bottom-up approaches
  - Letovsky 86
- Or even combine them
  - Mayrhauser and Vans 95, 96, 97

28

Knowledge-based understanding (1)

- Letovsky 86
- Describes programmers as opportunistic processors capable of exploiting either bottom-up or top-down cues as they become available.
- Three components to his model:
  - Knowledge base: encodes a programmer's expertise and knowledge before the task
  - Mental model: encodes the current understanding of the program
  - Assimilation process: describes how the mental model is formed using the programmers knowledge and source code and other documentation
- His study involved
  - Programmers with unfamiliar code
  - Ask these programmers to do a task
  - Asked them to use think-aloud

Knowledge-based understanding (2)

- Knowledge base
- Mental model – 3 layers:
  - Specification—high level abstract view
  - Implementation
  - Annotation
- Assimilation process
  - May occur bottom-up or top-down or some combination of the two in an opportunistic manner
  - Makes use of existing knowledge and any external help such as source code and documentation
  - Conjectures
    - Why: hypothesize the purpose of a function or design choice
    - How: hypothesize the method for accomplishing a program goal
    - What: hypothesize classification (e.g. variable or function)

Systematic/as-needed strategies

- Littman et al.
  - Microstrategies
    - inquiry episodes and delocalized plans
  - Macrostrategies
    - systematic and as-needed

31

Integrated meta-model

- von Mayrhauser/Vans – components:
  - top-down model (domain)
  - program model (control flow)
  - situation model (data flow)

  } Comprehension processes

  - knowledge base (programmer background)
- Experiments show that programmers switch between all three comprehension models

32

Static program analysis

- Def. The process of inferring results about the nature of a program according to some model without executing the subject program
- Syntactic analysis, type checking and inference
- Control and data flow analysis
- Structural analysis
- Slicing and dicing
- Cross references
- Complexity measures
- Navigation

33