Overview of Program Comprehension

CPRE 556 Lecture 12

Program Comprehension

- International Workshop on Program Comprehension http://www.ieee-iwpc.org/iwpc2005/
- Annotated bibliography http://www2.umassd.edu/swpi/processbibliography/bibcodereading2.html#Deimel90

Understanding Programs

- Goes far beyond the ability to read syntax.
- Serious economic issue for the industry.
- Program comprehension is required for:
  - Defect identification
  - Tracing the defect source
  - Code inspection
  - Preparation of test cases
  - Good documentation
  - Code revisions and enhancements

Macro vs. Micro Level Understanding

- Macro-level: understanding software at large with focus on the global and cross-cutting characteristics (e.g. class relationships).
- Micro-level: understanding a specific part of software in great detail (e.g. implementation of a data structure such as a tree).
- We will focus on macro-level understanding.

Code Reading Types

- Reading by step-wise abstraction
- Defect-based Reading
- Perspective-based Reading

Source: http://www2.umassd.edu/swpi/1docs/comprehension.html

Reading: step-wise abstraction

- Determine the function of critical subroutines, works through the program hierarchy until the function of the program is determined.
- A bottom-up strategy- map the code to suggested problem domain activity.
- Basili & Selby investigated the effectiveness:
  - the technique detects more software faults, and has a higher fault detection rate than functional or structural testing.

Source: http://www2.umassd.edu/swpi/1docs/comprehension.html

Defect-based Reading

- Defects are categorized and characterized, a set of questions developed for each defect class to guide the reader.
- Experiments conducted at the University of Maryland suggest that defect-based reading is more effective to ad hoc reading.

Source: http://www2.umassd.edu/swpi/1docs/comprehension.html

Perspective-based Reading

- Similar to defect-based reading, but instead of defects readers have different roles (tester, designer and user) to guide them in reading.
- Experiments conducted at the University of Maryland suggest that defect-based reading is more effective to ad hoc reading.
- Perspective-based reading has been applied to the inspection of requirements documents.

Source: http://www2.umassd.edu/swpi/1docs/comprehension.html

Cognitive Processes in Program Comprehension

- A *mental model* describes an engineer's mental representation of the program to be understood. A *cognitive model* describes the cognitive processes and information structures used to form the mental model. Three cognitive processes:
  - Expectation-based comprehension (Brooks 1983).
  - Inference-based Comprehension (Soloway 1984).
  - Bottom-up processing (Schneiderman & Mayer, 1979).
- Which strategy would be more useful in familiar domain?

Empirical Studies

- Empirical studies of cognitive processes using “Talk-Aloud Protocol”.
- Subjects are asked to verbalize their thought process of program understanding.
- Analysis schemas have been developed.
- Results from different subjects are compared to check for consistency of results.

Talk-aloud Excerpt

ACCOUNTING – COMPREHENSION
SUBJECT ID: SUBJECT1.ACC
PHRASES: 81 TO 100

81. CHECK-TWENTY-ONE-OUT
82. This looks like ID numbers
83. Or looks like we're printing a check
84. HEADLINE
85. REPORT-NAME,
86. WORK-STUDY ROSTER
87. TITLELINE
88. NAME, SOCIAL-SECURITY-NUMBER, BUDGET, GROSS-PAY, NET-PAY, LOCATION-LINE, DATE, LOCATION, PAGE
89. Next page
90. DEPARTMENT-LINE
91. TOTALS FOR DEPARTMENT, NUMBER OF STUDENTS
92. DEPARTMENT-CT, looks like a count
93. DEPT-GROSS-AC, DEPT-NO, DEPT-TYPE, something to do with ordering perhaps
94. DEPT-NET
95. And there are flags beside the COUNT, the GROSS, and the NET
96. UNIVERSITY-LINE looks like the same thing,
97. GRAND TOTAL
98. So up above we have looks like a DEPARTMENT-LINE and then the UNIVERSITY-LINE is a GRAND-TOTAL-LINE
99. MAILING is the LABLES bit
100. FIRST-NAME,

Expectation-based Comprehension

- What would be verbalization?

```
/* Code Segment */
for (i=0; i<n-1; i++) {
  for (j=0; j<n-1-i; j++)
    if (a[j+1] < a[j]) {
      tmp = a[j];
      a[j] = a[j+1];
      a[j+1] = tmp;
  }
}
```

Program Slicing

- Given a set of program elements S, a slice is a projection of the program that includes only program elements that might affect (either directly or transitively) the values of the variables used at members of S.
- A technique for visualizing dependencies and restricting attention to just the components of a program.
- Two main types: *backward* slicing and *forward* slicing.
- Project: http://www.cs.wisc.edu/wpis/html/

Effort Estimation for Program Comprehension

- Econometric model - http://portal.acm.org/citation.cfm?id=837837
- Case study: a subset of 26 programs from a banking application written in COBOL; 31,981 lines of code (locs), overall effort for *restoration* required about 170 man/hours.
- Efforts depend on: the objective of restoration, adequacy and capability of the tools used, engineer's experience, the knowledge of the applicative domain available etc.
- The model provides a way quantify and estimate the efforts.

Restoration

- The restoration process considered in the study included:
  - Classify data as applicative domain data, control data, structural data.
  - Rename variables using meaningful names.
  - Extract modules with high internal cohesion.
  - Localize variables declared to be global but used locally.

Reverse Engineering

- Identify software components, their interrelationships, and represent these entities at a higher level of abstraction.
  - *Redocumentation:* Perhaps the weakest form of reverse engineering.
  - *Design Rediscovery:* use domain knowledge and other external information to create a model of the system at a higher level of abstraction.
  - *Restructuring:* Transform the system within the same level of abstraction maintaining the same functionality and semantics.
  - *Reengineering: M*ost radical, involves both reverse and forward engineering to reexamine which functionalities need to be retained, deleted or added.

Difficulties

- Gap between the application model and the program.
- Computer science education is largely about mapping from the abstract to the detailed implementation, but there is little to assist in the reverse mapping.
- Over time, program structure drifts from the original specification. It becomes difficult to reconcile and synchronize the documented design and the current implemented design.

Tools for Program Comprehension

1. Source code comprehension tools: http://grok2.tripod.com/code_comprehension.html
2. A Survey of Program Comprehension and Reverse Engineering Tools by Nelson, http://arxiv.org/ftp/cs/papers/0503/0503068.pdf

Approaches for Automated

- Textual, lexical and syntactic analysis.
- Graphing program artifacts.
- Execution and testing.

Using Electronic Library

- IEEE Xplore: http://www.lib.iastate.edu/collections/db/ieeexx.html
- Process:
  - Suppose you get following reference after searching on Google http://portal.acm.org/citation.cfm?id=837837
  - Google search shows that the paper appeared in International Workshop on Program Comprehension (IWPC) in 96.
  - Click on Xplore, click on conferences, then type IWPC in the search box and go.
  - You will get a yearly listing of all IWPC proceedings.
  - Click on the appropriate year, the Table of Content comes up.
  - Click on the PDF link for the paper.
- WCRE is another conference with several relevant papers for this course.

References

1. Brooks, R., (1983) Towards a Theory of the Comprehension of Computer Programs. *International Journal of Man-Machine Studies*, Vol. 18.
2. Soloway, E., (1984) Empirical Studies of Programming Knowledge. *IEEE Transactions on Software Engineering*, IEEE Computer Society, Vol. SE-10, No. 5.
3. Schneiderman, B., Mayer, R., (1979) Syntactic / Semantic Interactions in Programmer Behavior. *International Journal of Computer and Information Sciences*, Vol. 8, No. 3.
4. Von Mayrhauser, A., Lang, S., (1999) A Coding Scheme to Support Analysis of Software Comprehension. *IEEE Transactions on Software Engineering*, Vol. 25, No. 4, July/August.

