Computational Thinking: A Historical View from PL/SE

Dr. Barbara G. Ryder

September 26, 2013

References

- *Prospects for an Engineering Discipline of Software,* Mary Shaw, IEEE Software, Nov 1990
- **The Impact of Abstraction Concerns on Modern Programming Languages*, Mary Shaw, IEEE TSE, Sept 1980
- **Computer Science: Reflections on the Field, Reflections From the Field*, National Research Council, 2004, pp11-23.
- **The Impact of SE Research on Modern PLs*, B. Ryder, M.L. Soffa, M. Burnett, ACM TOSEM, Oct 2005.(my added reference)

Historical Context for PL & SE

- SE and PL were same field until early 1970's
  - Shared NATO SW Confs 1968, 1969
  - First POPL 1973, first ICSE 1975
  - Parnas, Dijkstra, Wirth – all considered experts in both fields
- SW in 1970's going from *programming in the small* to *programming in the large* in the late 1970's-early 1980's
- Mary Shaw (CMU, SEI) leader in software architecture research
  - How to design maintainable, extensible programs
  - Believes SE principles affected PL design and vice-versa
    - Our IMPACT paper sought to prove the influence of SE research on PL design and vice-versa, using academic validation

Software Engineering

- *Hypothesis:* many ideas in evolving PL designs and discussions of SE Body Of Knowledge are relevant for defining a CS perspective on the essentials of Computational Thinking (CT)

Abstraction

"An abstraction is a simplified description or specification of a system that emphasizes some of the system's details or properties while suppressing others"

- Good abstractions emphasize information significant to the user, while ignoring other details
- Called *analytic modeling* in other fields
- For SW, abstraction describes what is to be achieved, not how to do this;
  - Emphasizes functional properties of system
- Abstraction of control, of procedures, of data

Abstraction as Model Building

- Questions to ask
  - What system characteristics are important?
  - What parameters are needed?
  - What formalism to use to build model?
  - How can model be validated?
- Can have hierarchical models
  - Model is system abstraction
  - Specification of a system is abstract description of model
  - Next lower level is implementation
  - Verification is validation that the specification is consistent with implementation

Abstraction - History

- 1960s-1970s:
  - *Control abstraction*
    - *GOTOs considered harmful* (structured programming – Dijkstra vs Knuth);
    - Defined clean information flow in and out of separable blocks of code
      - » single-entry, single-exit control structures (e.g., while – break- continue, if-then-else)
  - *Procedural abstraction*
    - Separable, parameterizable pieces of code with a particular function

Abstraction – History (2)

- Late 1960s-1970s:
  - User-defined datatypes, PL semantics (e.g., loop invariants)
  - Stepwise refinement of code (*top-down programming*) – conceptualizing a program in high-level operations and successively refining them into sequences of PL instructions with same functionality
  - *Abstract datatypes* – information hiding (Parnas)
    - Precursor to objects

Abstraction – History (3)

- Separation of concerns between abstract data types with certain behaviors and their actual implementation in code enabled problem decomposition into smaller and smaller segments

  Problem- Hard to make changes to SW – series of abstraction decisions not documented (unknown invariants)

  Problem- Lack of precision in descriptions of behavior

- Emphasis on program understanding as SW became more complex
  - Program verification – reasoning about state

Abstraction in PLs

- PLs as primary notation for complex ideas in problem solving
  - PL design can influence algorithm development
  - PLs used to communicate between people as well as for writing programs
  - PL design can make some algorithms more 'natural' than others
- 1980s: concerns
  - Keep PL design simple
  - Try to precisely analyze formal specifications
  - Pay attention to long-lived programs
    - Maintenance is longest period in the SW lifecycle

Abstract Data Types

- 1980s-1990s focus
  - Notion of private operations vs public operations on the data type – modules
  - Type checking provides degree of validation of programs
  - Invariants of data types
  - Generic definitions (commonly used aggregate type with its base type as parameter)

Ideas for CT

- CT helps us deal with complex problems by abstracting away non-essential details
- *Top-down programming* offers a process for problem solving by successive refinement, i.e., breaking a problem into smaller and smaller pieces
- *Procedural abstraction* subdivides problem into 'thinkable' pieces
- *Control abstraction* requires/facilitates solution steps which are easy to understand
- *Abstract data types* allow problem solving design in terms of relevant data and operations on it
- *Generics* allow generalization of a particular solution into a family of solutions

Essence of CS (Refl on field…2004)

"CS is the study of computers and what the can do - the inherent powers and limitations of abstract computers, the design and characteristics of real computers, and the innumerable applications of computers to solving problems"

What do Computer Scientists Do?
(From Refl on field…2004, p 12)

- "Seek to understand how to reason about processes and information"
- "Amplify human intellect through the automation of rote tasks and construction of new capabilities"
- "Create abstractions, symbolic representations of information, HW/SW artifacts that embody computing capabilities"
- "Create, study, experiment with real-world artifacts (HW, SW)"

What is CS Research?
(From Refl on field…2004, p 15)

- Involves
  - Creation and manipulation of symbols and abstractions
- Creates
  - Algorithms, Artificial constructs unlimied by physical laws
- Addresses
  - Fundamental limits on what can be computed and exponential growth
- Focus
  - On complex, analytic, rational action associated with human intelligence

Exploring further…

- Computers deal with discrete information
  - Bits – discrete info, real numbers – analogue info
- Use of symbolic representation
  - To permit analysis/processing
  - Sunflowers
    - » For analysis, genetic code diffs with marigolds
    - » For graphical display, describe color, shape, interacting parts
    - » For describing varieties, English words
- Creation and manipulation of abstractions

Exploring further…
(From Refl on field…2004, p 119)

- "Algorithms-precise ways to do a particular task- that perform operations on objects"
  - o Running time, optimization
- Modeling the world "as it is", and "as it could be"
- Dealing with scale – larger, faster, more data
- Idea of fundamental limits of computation
  - o Undecidability
  - o Solvable but not tractable (practically efficient)
- Emulation of human intelligence

Key Ideas for CT

Concepts

- Abstraction
  - Of control (for understanding and simplicity)
  - Of procedures (for efficiency/modularity)
  - Of data types (for organizing/accessing info; for understanding how data is transformed)
- Symbolic representation

Processes

- Stepwise refinement or top-down Programming (problem decomposition into simpler and simpler pieces
- Divide and conquer (recursive problem decomposition with homogeneous solution procedure)
- Generalization of problem solution to family of solutions

Discussion

- What can we take from this history of SE and PLs to get insight as to how computer scientists in these fields viewed problem solving as a computer scientist?
- Does this give us insight into CT?