Lecture 17

Programming Languages
(S&G, ch. 8)

Read S&G ch. 9
(Compilers and Language Translation)
The Phenomenology of Tools

Phenomenology

- A philosophical method
- A phenomenon is some aspect of concrete human experience of the world
- Phenomenology investigates the invariant structure of some phenomenon by systematic variation of that experience
- Has its own technical terminology
Phenomenology of Tools

• The phenomena of concern are *tools* in the broadest sense
• Developed by Don Ihde
• Here we are concerned with tools related to computers:
  – computers themselves
  – programming languages
  – word processors
  – email
  – etc.

Tools are Ampliative & Reductive

• Example: using stick to knock down fruit
• Ampliative aspects
  – Greater power or performance
  – Extended experience
• Reductive aspects
  – Experience is less immediate
  – Action is channeled by the tool
Examples

- Consider ampliative & reductive aspects of:
  - eye glasses
  - automobiles
  - telephones
  - email
  - recorded music
  - internet
  - word processor

Utopians vs. Dystopians

- Technological *utopians*:
  - focus on ampliative aspects
  - emphasize practical advantages
- Technological *dystopians*:
  - focus on reductive aspects
  - discount practical advantages
- Both attitudes are reduced focuses
- Better:
  - acknowledge essential ambivalence of our experience of the tool
  - “all technology is non-neutral” — Ihde
Phenomenology of Programming Languages

- Ampliative aspect:
  - automation of tedious, error-prone activities
  - error checking
- Reductive aspect:
  - loss of direct control of machine resources
  - possible inefficiency

“Fascination” & “Fear”

- Typical responses to a new technology
- Utopians are fascinated by ampliative aspects
  - embrace & promote the new technology
  - tend to over-apply the new technology
  - inclined to further amplification
- Dystopians fear reductive aspects
  - may view ampliative aspects as dangerous
  - ambivalent feelings of power or helplessness
- Greater familiarity ⇒ balanced understanding of benefits & limitations
Mastery & Embodiment

- Tool replaces immediate (direct) with mediated (indirect) experience
- When tool is mastered, its mediation becomes transparent (not invisible)
- Contrast:
  - bad tool or unskillful use ⇒ experienced as object, relate to it
  - good tool & mastery ⇒ partially embodied, relate through it
- With mastery, objectification becomes embodiment

Focus & Action

- Example: three writing technologies
  - dip pen
  - electric typewriter
  - word processor
- Tools influence focus:
  - makes some aspects of situation salient, hides others
- Tools influence action:
  - makes some actions easy, others awkward
- Tools subtly influence what we notice and do
Cultural Embedding

- All technologies are culturally embedded
- Our reactions to them are influenced by:
  - personal backgrounds
  - collective background
- Stylistic inclinations may vary from user to user

Conclusions

- The phenomenology of tools helps us to:
  - understand people’s experience to tools
  - move beyond our own limited perspectives
  - understand social consequences of tools
  - exercise informed choice about adopting new technologies
  - design better tools
  - understand tool’s effect on focus & action
- “All technology in non-neutral”
Functional Programming

Definition

In functional programming, all programs are mathematical functions:

• The arguments of the function are the inputs to the program
• The computed value of the function is the only output from the program
• There are no side-effects
• The assignment statement is not used!
Functions vs. Procedures with Side-Effects

functional program:

- arguments
- function
- result

procedural program:

- inputs
- procedure
- output
- side-effects

Side-effects are hidden interfaces to a program

Scheme & LISP Syntax

( <function> <argument 1> <argument 2> … )

like a verb

like the objects of the verb
Examples

- \((\log 2)\) — compute the logarithm of 2 and return it
- \((* 2 \, x)\) — compute the product of 2 and \(x\)
- \((\text{list} \, 3 \, 4 \, 5)\) — make a list out of the numbers 3, 4, and 5
- \((\text{define} \, (\text{double} \, x) \, (* \, 2 \, x))\) — define a function with the header “\((\text{double} \, x)\)” and the body “\((* \, 2 \, x)\)”
- \((\text{double} \, 4)\) — apply the user-defined function double to 4

Conditional Function

\[
\begin{align*}
\text{(cond} & \\
& \hspace{1em} (\text{condition} \, \text{value if true}) \\
& \hspace{1em} (\text{else} \, \text{value if false}) \\
& \text{)}
\end{align*}
\]

\[
\begin{align*}
\text{(cond} & \\
& \hspace{1em} (\text{condition} \, \text{1}) \, \text{value if cond 1 true} \\
& \hspace{1em} (\text{condition} \, \text{2}) \, \text{value if cond 2 true} \\
& \hspace{1em} \vdots \\
& \hspace{1em} (\text{else} \, \text{value if all conds false}) \\
& \text{)}
\end{align*}
\]
Run DrScheme