CSE 291 I: Usability of Programming Languages ("Programmers Are People Too")

Cognitive Dimensions of Notation
Finishing Up: Quantitative Methods
TESTS

• To compare means of two groups (continuous data):
  • T test: assumes normal distribution (parametric)
  • Mann-Whitney U test: does not assume normal distribution (nonparametric)
• For > 2 categories, use ANOVA (ANalysis Of VAriance)
  • ANOVA gives probability that at least two groups are different
• RQ: does garbage collection help Rust novices complete tasks more effectively?

• Assigned participants to use GC or not

• Measured time, scores, and a bunch of Likert-scale questions
Cognitive Dimensions of Notation

Expert Methods

• Recall: expert methods do not require any users

• Pros:
  • Low-cost (don't have to design studies, recruit participants)
  • Applicable at early stages of design

• Cons:
  • Results depend on expert's analytic skills and methods
  • *Cognitive dimensions* serve as one analytic method.
Cognitive Dimensions as Vocabulary

• Programming depends on notation

• What vocabulary can we use to describe properties of notation?

• If we can describe properties, we can analyze tradeoffs
  
  • Every notation highlights some kinds of information at the expense of obscuring other kinds.

  • When seeking information, there must be a cognitive fit between the mental representations and the external representation.
Visual Programming Languages

• Programming is not just text
  • Spreadsheets
  • LabVIEW
  • Scratch
  • IDEs…

• Approaches to analyze notation should generalize to VPLs too
Example Task

• The rocket program computes the vertical and horizontal trajectory of a rocket on which the only forces acting are its thrust and gravity. At time zero the rocket stands stationary and vertical on level ground, with a mass of 104 pounds. Its engine develops a thrust of $4 \times 10^5$ foot-pounds, using up a mass of 50 pounds of fuel per second, until the fuel is exhausted after 100 seconds. It rises vertically for 10 seconds after which it adopts and retains an angle of 0.3941 radians (22.5 degrees) to the vertical. The downwards acceleration of gravity is 32 feet sec$^{-2}$. 
Example Solutions (MS BASIC, LabVIEW)

```
Mass = 10000
Fuel = 50
Force = 400000
Gravity = 32

WHILE Vdist >= 0
    IF Tim = 11 THEN Angle = .3941
    IF Tim > 100 THEN Force = 0 ELSE Mass = Mass - Fuel
    Vaccel = Force * COS(Angle) / Mass - Gravity
    Vveloc = Vveloc + Vaccel
    Vdist = Vdist + Vveloc
    Haccel = Force * SIN(Angle) / Mass
    Hveloc = Hveloc + Haccel
    Hdist = Hdist + Hveloc

    PRINT TIM, Vdist, Holst
    Tim = Tim + 1
WEND

STOP
```
Dimensions Examples

- Viscosity: how hard is it to make changes?
- Which changes?
- Structured editors typically have high viscosity (vs. text editors)
Hidden Dependencies

• Spreadsheets tend to hide dependencies

• Textual languages partially hide dependencies

• Dataflow languages expose dependencies

Figure 10. Hidden and explicit tendencies. Spreadsheets (top) contain hidden dependencies which can interfere with comprehension and debugging: e.g., cell D2 records those cells to which it refers but does not indicate those cells to which it supplies data. The data flow representation (bottom left) makes the dependencies visible but increases the viscosity. Conventional imperative text languages (bottom right) use variables; because their names have to be remembered and matched symbolically rather than perceptually, they could be classed as partially hidden. Notice that the required material is visible in all these cases; it is the dependencies that are hidden.

A form of hidden dependency, in which a function or subroutine surreptitiously altered the value of a global variable, today the classic example is the spreadsheet. The formula in a cell tells which other cells it takes its value from, but does not tell which other cells take their value from it. In the worst case the whole sheet must be scrutinised cell by cell before one can be certain that it is safe to change a cell. Hidden dependencies are apparently responsible for a very high error frequency in spreadsheets, possibly because the spreadsheet programmer hopes for the best rather than scrupulously checking out all of the consequences of a change.

When spreadsheets are viewed in the data-view mode rather than the formula-view mode, the problem becomes still more acute (Figure 10). Other examples include GOTOs and subroutine structures (where is this subroutine used?); in both cases, the absence of a 'come-from' means that it can be risky to make a change to the code. Most VPLs, especially those using data-flow, make GOTO dependencies explicit, but subroutines remain a problem, one that may be amplified by nesting.

Cross-references and call-graphs are remedies for hidden dependencies in conventional languages; antecedents and dependents trees are remedies in data-flow languages. Where computed targets are permitted, such as computed GOTOs, no general remedy is possible. An unfortunate consequence is that it becomes impossible to discover which objects are 'live' (that is, are referred to by some other object) and which are 'dead'. In small systems, exhaustive search may be used to discover which objects are dead. In large systems, search is infeasible and the system accumulates 'fossils' (objects which are presumed dead but which nobody dares throw away just in case they are not dead after all).

Unix filestores typically contain a very large number of hidden dependencies with large numbers of 'dot' files used as data by other scripts, many of which are in fact fossils silting up the system.

Hidden dependencies can be a severe source of difficulty. Eisenstadt recorded

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>item</td>
<td>unit price</td>
<td>units</td>
<td>price</td>
<td>tax</td>
<td>total price</td>
</tr>
<tr>
<td>2</td>
<td>apples</td>
<td>£0.84</td>
<td>2</td>
<td>= B2*C2</td>
<td>0.29</td>
<td>= E2*D2</td>
</tr>
</tbody>
</table>

read (UnitPrice, Units, Tax)
Nett = Units * UnitPrice
Total = Nett + Tax * Nett
Contrast: Nielsen's Heuristics

- 10 heuristics
- Example: *Match between system and the real world*
- Applicable to UIs in general, not just PLs

https://www.nngroup.com/articles/ten-usability-heuristics/
Nielsen's Heuristics

- Recognition rather than recall
  - "What's the capital of Portugal?"
  - "Is Lisbon the capital of Portugal?"
- The latter is much easier for most people!
- GUIs vs. command line
Group Activity

- Analyze (using CDs):
  - Flowcharts
  - Assembly programs
  - Paper/pencil mathematics
  - To-dos on sticky notes
  - Word processor styles
  - Musical notation

- Compare (using CDs): spreadsheets vs. databases

- Identify examples in real life:
  - Premature commitment
  - Viscosity
  - Hidden dependencies
  - Abstraction gradient
  - Closeness of mapping