Human-Computer Interaction

Lecture 6: Programming languages

USABILITY OF NOTATIONS
Cognitive Dimensions of Notations

• ‘Discussion tools’ for use when considering alternative designs of programming languages, and other complex information systems
  – Suitable for analytic evaluation before, during and after a design process (e.g. iterative prototyping)
  – But not a checklist of ideal features
• We have to escape ‘superlativism’
  – I think the best programming language is … !
• All real design is about trade-offs
  – What is better – a Lamborghini or a tractor?
  – Do they have design principles in common?

Where do we find information structures?

• Not only programming languages, but anything with internal structure (relationships, dependencies etc)
  – UML diagrams, Spreadsheets, Travel bookings, Musical compositions, Technical manuals, Novels
• Structured information devices involve:
  – a notation
  – an environment
  – a medium
• Consider example dimension: Viscosity
  – simplified preview definition:
    “a viscous system is hard to modify”
Example: modifying structure of text

- **Notations:**
  - inserting text in a novel is easier than in more structured formats like a newspaper
- **Environment:**
  - structures in a word processor are easier to modify, on pencil and paper are harder to modify
- **Media:**
  - any part of a text on paper can be accessed easily, but harder on a dictaphone (example – Philip Pullman).

Definitions

- **Notation:**
  - The perceived marks or symbols (as covered in visual representation lecture), and the correspondence to what they are supposed to mean
- **Environment:**
  - The operations and tools provided for users to navigate, read and manipulate the perceived marks
- **Medium:**
  - Where the marks are being made (screens, paper, Post-Its, tangible objects, augmented reality)
User experience of notational systems

- Interaction is viewed as building, modifying and navigating an information structure
- Usability depends on the structure of the notation and the tools that the environment provides for manipulating marks within the medium
- Dimensions like viscosity draw attention to aspects of user experience when interacting with the information structure
- Different activities have different profiles (e.g. the facilities you need when reading a technical manual are different from those you need when writing one)

Construction activities: building information structure

- Incrementation
  - add a new formula to a spreadsheet
- Transcription
  - convert an equation to a spreadsheet formula
- Modification
  - change spreadsheet for a different problem
- Exploratory design
  - programming on the fly ("hacking")
Interpretation activities: reading information structures

- **Search**
  - find a value specified in a spreadsheet
- **Comparison**
  - fault-finding, checking correctness
- **Exploratory understanding (sensemaking)**
  - analyse a business plan presented in a spreadsheet

Dimensions and Activities are orthogonal

![Diagram showing orthogonal activities and dimensions]
Profiles describe desirable combinations

SOME DETAILED DIMENSIONS
Dimensions covered today:

- Abstraction
  - types and availability of abstraction mechanisms
- Hidden dependencies
  - important links between entities are not visible
- Premature commitment
  - constraints on the order of doing things
- Secondary notation
  - extra information in means other than formal syntax
- Viscosity
  - resistance to change
- Visibility
  - ability to view components easily

Not covered in detail today:

- Closeness of mapping
  - closeness of representation to domain
- Consistency
  - similar semantics expressed in similar forms
- Diffuseness
  - verbosity of language
- Error-proneness
  - notation invites mistakes
- Hard mental operations
  - high demand on cognitive resources
- Progressive evaluation
  - work-to-date checkable any time
- Provisionality
  - degree of commitment to actions or marks
- Role-expressiveness
  - component purpose is readily inferred
- And more ...
  - Research continues to identify new dimensions
Viscosity

• **Resistance to change: the cost of making small changes.**
• Repetition viscosity:
  – e.g. manually changing US spelling to UK spelling throughout a long document
• Domino (was “Knock-On”) viscosity:
  – e.g. inserting a figure in a document means updating all later figure numbers, their cross-references, the list of figures, the index ...

Viscosity features

• Viscosity becomes a problem when you need to change your plan: it is a function of the work required to change a plan element.
• It is a property of the system as a whole
• May be different for different operations
• Often happens when designers assume system use will only involve incrementation, but that users will never change the structure.
Viscosity examples

• Repetition viscosity example:
  – When the user has one document in mind, but it is stored as a collection of files, which must be edited separately to change style in all.

• Domino viscosity example:
  – In structures with high inter-dependency, such as timetables.

• Combinations of the two are the worst!

Combined domino/repetition

• Common in graphic structures, genealogical trees, hypertexts ...
  – e.g. tree showing part of JavaScript hierarchy

```
window
  \___________\___________\
  \___________\___________\
  document frames location navigator
  \___________\___________\
  plugins mimeTypes
```
**Workarounds & trade-offs**

- Separate exploratory & transcription stages
  - e.g. pencil sketch before ink
- Introduce a new abstraction
  - e.g. AutoNumber facility
- Change the notation
  - e.g. quick dial codes for telephone

**Hidden Dependencies**

- *A relationship between components such that one is dependent on the other, but the dependency is not fully visible.*
- The one-way pointer:
  - e.g. your Web page points to someone else’s - how do you know when they move it?
- Local dependency:
  - e.g. which spreadsheet cells use the value in a given cell?
Hidden Dependency features

- Hidden dependencies slow up information finding.
- Tolerable in exploratory design, but not in modification.
- May be responsible for high frequency of errors in spreadsheets.

Hidden Dependency examples

- GOTO statements didn’t have a corresponding COME-FROM.
  - Block structure brings symmetry.
- Data-flow makes dependencies explicit.
Workarounds & trade-offs

• Require explicit cueing
  – e.g. import and export declarations
• Highlight different information
  – e.g. data-flow language
• Provide tools
  – e.g. spreadsheets which highlight all cells that use a particular value

Premature Commitment / Enforced Lookahead

• Constraints on the order of doing things force the user to make a decision before the proper information is available.
Premature commitment features

• Only occur if three conditions hold:
  – target notation contains internal dependencies
  – access to both source and target is order-constrained
  – the constrained order is inappropriate

• Happens when designer’s view of “natural sequence” is at variance with user’s needs

• Results in 2nd and 3rd attempts at task

Premature commitment examples

• Telephone menu systems

• Four-function calculator
  – \((1.2 + 3.4 - 5.6) / ((8.7 - 6.5) + (4.3))\)
More types and examples

- Defining database schemas before the data
- Filing systems (library shelving by Dewey)
- Surreptitious order constraints
  - Provisional relationships in E-R diagram
- Effect of medium
  - Exacerbated when ‘marks’ are transient
    (e.g. in an auditory medium)

Workarounds & trade-offs

- Decoupling
  - e.g. the signwriter paints the sign elsewhere
- Ameliorating
  - premature commitment is not so bad if viscosity is low &
    bad guesses can be corrected
- Deconstraining
  - e.g. GUI interfaces often remove constraints on order of
    actions
Abstractions

- **An abstraction is a class of entities or grouping of elements to be treated as one entity** (thereby lowering viscosity).
- Abstraction barrier:
  - minimum number of new abstractions that must be mastered before using the system (e.g. Z)
- Abstraction hunger:
  - require user to create abstractions

Abstraction features

- Abstraction-tolerant systems:
  - permit but do not require user abstractions (e.g. word processor styles)
- Abstraction-hating systems:
  - do not allow definition of new abstractions (e.g. spreadsheets)
- Abstraction *changes the notation*. 
Abstraction implications

- Abstractions are hard to create and use
- Abstractions must be maintained
  - useful for modification and transcription
  - increasingly used for personalisation
- Involve the introduction of an *abstraction manager* sub-device
  - including its own viscosity, hidden dependencies, juxtaposability etc.

Abstraction examples

- Persistent abstractions:
  - Style sheets, macros, telephone memories
- Definitions and exemplars
  - Powerpoint templates, CAD libraries
- Transient abstractions:
  - Search and replace, selection aggregates
Workarounds & trade-offs

- Incremental abstractions
  - low abstraction barrier, tolerates new additions, provides alternatives (but may confuse)
- Overcoming abstraction-repulsion
  - abstractions decrease viscosity, but increase problems for occasional / end-users
- Programming by example?
  - can introduce abstract hidden dependencies

Secondary Notation

- *Extra information carried by other means than the official syntax.*
- Redundant recoding:
  - e.g. indentation in programs, grouping control knobs by function
- Escape from formalism:
  - e.g. annotation on diagrams
Secondary Notation features

• Redundant recoding ➔ easier comprehension ➔ easier construction.
• Escape from formalism ➔ more information
• Is secondary notation ever bad?  
  – what about the brevity bigots?
• Designers often forget that users need information beyond the “official” syntax.  
  – and even try to block the escapes people use

Secondary Notation examples

• Redundant recoding
  – Telephone number layout
  – Front panel of a car radio
  – Functional grouping

0114 225 5335
or
0 11 42 25 53 35?

Fig. 31: The circuit diagram of a simple audio amplifier.
Secondary Notation examples

• Escape from formalism
  – Usage of calendars and diaries.

Workarounds & trade-offs

• Decoupling (if insufficient secondary notation)
  – e.g. print out hard copy, attack it with a pencil

• Enriched resources
  – e.g. tagging and annotation tools

• But extensive secondary notation introduces added viscosity (it gets out of date).
  – e.g. program comments
Visibility & Juxtaposability

- **Ability to view components easily & to place any two components side by side.**
- Visibility:
  - e.g. searching a telephone directory for the name of a subscriber who has a specified telephone number
- **Juxtaposability:**
  - e.g. trying to compare statistical graphs on different pages of a book

Visibility & Juxtaposability features

- Structure or indexing information is often invisible because designers assumed it wouldn’t be needed.
- Often caused by presenting information in windows, then restricting the number of windows.
- Becomes far worse with small devices (cell-phones, PDAs, wearable computers?).
Visibility & Juxtaposability examples

- Small windows onto invisible control trees:
  - e.g. car radios, fax machines, cameras.
- Shared use displays:
  - e.g. clock-radio: time or alarm or radio station
- Form based systems:

Workarounds & trade-offs

- Working memory
  - refreshed by revisiting items being compared
- External memory
  - e.g. make a hard copy of one component (a new environment that allows side-by-side viewing)
- Adding a browser
  - e.g. class browser, alternative views
- Visibility trades off against clutter, abstraction
Desirable profiles

<table>
<thead>
<tr>
<th></th>
<th>transcription</th>
<th>incrementation</th>
<th>modification</th>
<th>exploration</th>
</tr>
</thead>
<tbody>
<tr>
<td>viscosity</td>
<td>acceptable</td>
<td>acceptable</td>
<td>harmful</td>
<td>harmful for small tasks</td>
</tr>
<tr>
<td>hidden dependencies</td>
<td>harmful</td>
<td>harmful</td>
<td>harmful</td>
<td>harmful</td>
</tr>
<tr>
<td>premature commitment</td>
<td>harmful</td>
<td>harmful</td>
<td>harmful</td>
<td>harmful</td>
</tr>
<tr>
<td>abstraction barrier</td>
<td>useful</td>
<td>useful (?)</td>
<td>useful</td>
<td>harmful</td>
</tr>
<tr>
<td>abstraction hunger</td>
<td>useful (?)</td>
<td>–</td>
<td>v. useful</td>
<td>v. harmful</td>
</tr>
<tr>
<td>secondary notation</td>
<td>not vital</td>
<td>not vital</td>
<td>important</td>
<td>important</td>
</tr>
<tr>
<td>visibility /</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>juxtaposability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notable trade-offs

- Premature commitment
- Viscosity
- Abstraction usage
- Learnability
- Visibility
- Hidden dependencies
- Juxtaposability
- Secondary notation
Some design manoeuvres

• Potential design approaches to:
  – reduce viscosity
  – improve comprehensibility
  – make premature commitment less expensive
  – remove need for lookahead
  – improve visibility

Design manoeuvres (1)

• Aim: to reduce viscosity
• Manoeuvre
  – add abstractions (so one “power command” can change many instances)
• At this cost
  – increased lookahead (to get right abstractions);
  – raises the abstraction barrier;
  – may increase dependencies among abstractions
Design manoeuvres (2)

• Aim: to improve comprehensibility
• Manoeuvre
  – allow secondary notation - let users choose placing, white space, font & colour; allow commenting
• At this cost
  – increases viscosity (because layout, colour etc not usually well catered for by environments)

Design manoeuvres (3)

• Aim: to make premature commitment less expensive
• Manoeuvre
  – reduce viscosity (so that users can easily correct their first guess)
• At this cost
  – see above, re viscosity
Design manoeuvres (4)

• Aim: to remove need for lookahead
• Manoeuvre
  – remove internal dependencies in the notation;
  – allow users to choose an easier decision order
• At this cost
  – may make notation diffuse, or increase errors
  – allowing free order needs a cleverer system

Design manoeuvres (5)

• Aim: to improve visibility
• Manoeuvre
  – add abstractions (so that the notation becomes less diffuse)
• At this cost
  – see above re abstractions
A CASE STUDY
Example – LabView visual programming language

Hidden dependencies

- Visual languages make connections explicit
- But with the trade-off that they need more screen space

BASIC:

```
x = 1
... (possibly many pages of code here...)
y = x + 3
```

LabVIEW:
Premature commitment (1)

- Commitment to layout is a common problem e.g. \( x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \)
- Start with minus b ...

Premature commitment (2)

- ... I’ll need b-squared too ...
Premature commitment (3)

- ... turn that into b-squared minus 4ac ...

Premature commitment (4)

- ... oops, that’s going to be 4ac minus b-squared ... try moving the 4ac chunk down and reconnecting to the ‘minus’ box ...
Premature commitment (5)

• ... OK, now I need plus or minus that ...

• that’s root-b-squared-minus-4ac but I still haven’t used b ... or the rest of the formula!

Secondary notation

• Little support for commenting
  – can only attach comment to a single item
• Spatial layout can’t easily be used for grouping
• All the visual variables (degrees of freedom) are taken up by the formal syntax
Visibility & Juxtaposability

- Visibility of data flow in LabVIEW is excellent
- But control branches in LabVIEW can’t be juxtaposed: