Software tools and implementation support

Shneiderman ch. 5, Dix chs. 6, 8

Introduction

• How does HCI affect programmer?
• Programming focus
  • hardware → interaction-technique
• Layers of development tools
  • design: waterfall model
  • specification methods
  • interface building tools
  • windowing systems
  • interaction toolkits
  • user interface management systems
  • evaluation and critiquing tools
Overview

• design
• specification methods …
• interface building tools …
• programming tools
  • levels of services for programmers
• windowing systems
  • core support for separate and simultaneous user-system activity
• programming application & control of dialogue
• interaction toolkits
  • bring programming closer to level of user perception
• user interface management systems
  • controls relationship between presentation and functionality
• evaluation

The waterfall model

- Requirements specification
- Architectural design
- Detailed design
- Coding and unit testing
- Integration and testing
- Operation and maintenance
Activities in the life cycle

• Requirements specification
  • designer and customer
  • try to capture what system expected to provide
  • can be expressed in
    • natural language
    • more precise specification languages
• Architectural design
  • high-level description of how system will provide services required
  • need to satisfy both
    • functional and
    • nonfunctional requirements
• Detailed design
  • refine architectural components and interrelations
  • identify modules to be implemented separately
  • refinement governed by nonfunctional requirements

Verification and validation

• Verification
  • designing the product right
• Validation
  • designing the right product
• The formality gap
  • validation always rely to some extent on subjective means of proof
• Management and contractual issues
  • design in commercial and legal contexts
life cycle for interactive systems

cannot assume linear sequence of activities as in waterfall model

lots of feedback!

Usability engineering

Ultimate test of usability based on measurement of user experience

Usability engineering demands that specific usability measures be made explicit as requirements

Usability specification
- usability attribute/principle
- measuring: what
- measuring: how
- now level/ worst case/ planned level/ best case

Problems
- usability specification
  - requires level of detail that may not be possible early in design
  - satisfying does not necessarily satisfy usability
part of usability specification for VCR

<table>
<thead>
<tr>
<th>Attribute: Backward recoverability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measuring concept:</td>
</tr>
<tr>
<td>Undo erroneous programming sequence</td>
</tr>
<tr>
<td>Measuring method:</td>
</tr>
<tr>
<td>Number of explicit user actions to undo current program</td>
</tr>
<tr>
<td>Now level:</td>
</tr>
<tr>
<td>No current product allows such undo</td>
</tr>
<tr>
<td>Worst case:</td>
</tr>
<tr>
<td>As many actions as it takes to program-in mistake</td>
</tr>
<tr>
<td>Planned level:</td>
</tr>
<tr>
<td>Maximum of two explicit user actions</td>
</tr>
<tr>
<td>Best case:</td>
</tr>
<tr>
<td>One explicit cancel action</td>
</tr>
</tbody>
</table>

ISO usability standard 9241

adopts traditional usability categories:

- **effectiveness**
  - can you achieve what you want to?
- **efficiency**
  - can you do it without wasting effort?
- **satisfaction**
  - do you enjoy the process?
some metrics from ISO 9241

<table>
<thead>
<tr>
<th>Usability objective</th>
<th>Effectiveness measures</th>
<th>Efficiency measures</th>
<th>Satisfaction measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suitability for the task</td>
<td>Percentage of goals achieved</td>
<td>Time to complete a task</td>
<td>Rating scale for satisfaction</td>
</tr>
<tr>
<td>Appropriate for trained users</td>
<td>Number of power features used</td>
<td>Relative efficiency compared with an expert user</td>
<td>Rating scale for satisfaction with power features</td>
</tr>
<tr>
<td>Learnability</td>
<td>Percentage of functions learned</td>
<td>Time to learn criterion</td>
<td>Rating scale for ease of learning</td>
</tr>
<tr>
<td>Error tolerance</td>
<td>Percentage of errors corrected successfully</td>
<td>Time spent on correcting errors</td>
<td>Rating scale for error handling</td>
</tr>
</tbody>
</table>

Specification methods

- Design requires good notation to record and discuss alternate possibilities:
  - default language for specifications in any field
    - natural language
      - e.g., English
    - communication medium
      - e.g., sketchpad, or blackboard
  - Natural-language specifications
    - lengthy
    - vague
    - ambiguous
  - => often difficult to prove:
    - correct
    - consistent
    - complete
Specification methods (cont.)

• Backus-Naur Form (a.k.a. Backus Normal Form or BNF)
  • high-level components described as nonterminals
  • specific strings described as terminals

• Grammars example
  – <Telephone book entry>::= <Name><Telephone number>
  – <Name> ::= <Last name>, <First name>
  – <Last name> ::= <string>
  – <First name> ::= <string>
  – <string> ::= <character>|<string>
  – <character> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
  – <Telephone number>::= (<area code>) <exchange>-<local number>
  – <area code>::= <digit><digit><digit>
  – <exchange>::= <digit><digit><digit>
  – <local number>::= <digit><digit><digit><digit><digit>::= 0|1|2|3|4|5|6|7|8|9

Examples of acceptable entries
  – WASHINGTON, GEORGE  (301) 555-1234
  – BEEF, STU  (726) 768-7878
  – A, Z  (999) 111-1111

• Multiparty grammars
  – <Session> ::= <U: Opening> <C: Responding>
  – <U: Opening> ::= LOGIN <U: Name>
  – <U: Name> ::= <U: string>
  – <C: Responding> ::= HELLO [<U: Name.]

  U: User C: Computer

• Multiparty grammars effective for text oriented command sequences
Specification methods (cont.)

Transition Diagram
- **nodes**: system states
- **links** between nodes: possible transitions

![Transition Diagram](Image)
Specification methods (cont.)

- State Charts

![State Chart Diagram]

Specification methods (cont.)

![Diagram of SecureSwitch and Operating States]
Interface-building tools

Features of interface-building tools

• User Interface Independence
  • Separate interface design from internals
  • Enable multiple user interface strategies
  • Enable multiple platform support
  • Establish user interface architect role
  • Enforce standards

• Methodology & Notation
  • Develop design procedures
  • Find ways to talk about design
  • Create project management

Interface-building tools (cont.)

• Rapid Prototyping
  • Try out ideas very early
  • Test, revise, test, revise,...
  • Engage end users, managers, and others

• Software Support
  • Increase productivity
  • Offer some constraint & consistency checks
  • Facilitate team approaches
  • Ease maintenance
User interface mockup tools

- Examples
  - Paper and pencil
  - Word processors
  - Slide-show software
  - Macromedia Director, Flash mx, or Dreamweaver

- Visual Editing
  - Microsoft Visual Studio
  - Borland JBuilder
Interface-building tools (cont.)
Interface-building tools (cont.)

Finding right tool = tradeoff between six main criteria:

1. Part of application built using the tool
2. Learning time
3. Building time
4. Methodology imposed or advised
5. Communication with other subsystems
6. Extensibility and modularity

<table>
<thead>
<tr>
<th>Software Layers</th>
<th>Visual Tools</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Application</td>
<td>Model-Based Building</td>
<td>Microsoft Access, Sýbase PowerDesigner</td>
</tr>
<tr>
<td>2. Application Framework/ Specialized Language</td>
<td>Conceptual Building Tools</td>
<td>Macromedia Director, Tcl/Tk, Microsoft MFC</td>
</tr>
<tr>
<td>1. GUI Toolkit</td>
<td>Interface Builder</td>
<td>Borland JBuilder, Microsoft Visual Studio</td>
</tr>
<tr>
<td>1. Windowing System</td>
<td>Resources Editor</td>
<td>Windows Graphical User Interface, Apple Quartz, X11 Windowing System</td>
</tr>
</tbody>
</table>
Interface-building tools (cont.)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Part of the application built</th>
<th>Learning time</th>
<th>Building time</th>
<th>Methodology imposed or advised</th>
<th>Communication with other subsystems</th>
<th>Extensibility and modularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>All for a specific domain</td>
<td>Long</td>
<td>Short</td>
<td>Specification first, then visual, then programming (if required)</td>
<td>Very good for the specific domain of the tool</td>
<td>Very good</td>
</tr>
<tr>
<td>3</td>
<td>Presentation, interaction</td>
<td>Short (days)</td>
<td>Short</td>
<td>Visual first</td>
<td>Depends on the tool</td>
<td>Languages: Bad Frameworks; Good</td>
</tr>
<tr>
<td>2</td>
<td>Presentation</td>
<td>Long (weeks)</td>
<td>Long</td>
<td>Visual first with tools, none otherwise</td>
<td>Good</td>
<td>Medium/good</td>
</tr>
<tr>
<td>1</td>
<td>All</td>
<td>Very long (months)</td>
<td>Very long</td>
<td>None</td>
<td>Very good</td>
<td>Very bad</td>
</tr>
</tbody>
</table>

The windowing system layer
- Sometimes working at low-level is required
- E.g., new platform
- The while(true) main loop

```java
main() {
    InitializeSystem();
    SetInitialState();
    DisplayInitialGraphics();
    while(true) {
        Event event = readNextEvent();
        switch(event.type) {
            case EVENT_HIDE: hideDisplay(); break;
            case EVENT_PUN_DOWN: doPunDown(event.x, event.y); break;
            case EVENT_CHAR: doInputChar(event.detail); break;
            default: doSystemDefault(event); break;
        }
    }
}
Interface-building tools (cont.)

- The GUI toolkit layer
  - Widgets
    - windows
    - scroll bars
    - pull-down / pop-up menu
    - ...
  - Difficult to use without interface
Interface-building tools (cont.)

The application framework and specialized language layer

- Application frameworks are based on object-oriented programming
  - Can quickly build sophisticated interfaces
  - Require intensive learning
- Specialized language layers lighten programming burden
  - Tcl (and its toolkit Tk)
  - Perl/Tk
  - Python/Tk
  - Visual Basic
  - Java Script

```html
<html>
  <head>
    <script>
      /* Definition of Javascript functions inside an HTML document */
      function add() { /* Function to add a new entry to the HTML option list */
        var list = document.forms[0].phoneBookList;
        list.options[list.length] = new Option(document.forms[0].entry);
      }
      function remove() { /* Function to remove a selected entry */
        var list = document.forms[0].phoneBookList;
        for (var i = 0; i < list.length; i++) {
          if (list.options[i].text == document.forms[0].entry.value) {
            list.options[i].null;
            return;
          }
        }
      }
      ...
    </script>
  </head>
  <body>
    <form>
      <input type="text" name="entry" value="">
      <input type="button" value="Add" onClick="add()"/>
      <input type="button" value="Remove" onClick="remove()"/>
      <input type="select" name="phoneBookList">
      ...<input type="select" name="phoneBookList">
    </form>
  </body>
</html>
```
Elements of windowing systems

- Device independence
  - program abstract terminal device drivers
  - image models for output and (partially) input
    - pixels
    - PostScript (MacOS X, NextStep)
    - Graphical Kernel System (GKS)
    - Programmers' Hierarchical Interface to Graphics (PHIGS)
- Resource sharing
  - achieving simultaneity of user tasks
  - window system supports independent processes
  - isolation of individual applications

roles of a windowing system
Architectures of windowing systems

- three possible software architectures
  - all assume device driver is separate
  - differ in how multiple application management implemented

1. each application manages all processes
   - everyone worries about synchronization
   - reduces portability of applications

2. management role within kernel of operating system
   - applications tied to operating system

3. management role as separate application
   - maximum portability

client-server architecture
X Windows architecture

• pixel imaging model with some pointing mechanism

• X protocol defines server-client communication

• separate window manager client enforces policies for input/output:
  • how to change input focus
  • tiled vs. overlapping windows
  • inter-client data transfer
event loop

repeat
  read-event {myevent}
  case myevent.type
    type_1:
      do type_1 processing
    type_2:
      do type_2 processing
    ...
    type_n:
      do type_n processing
  end case
end repeat

notification-based

void main(String[] args) {
  Menu menu = new Menu();
  menu.setOption("Save");
  menu.setOption("Quit");
  menu.setAction("Save", mySave);
  menu.setAction("Quit", myQuit);
  ...
  int mySave(Event e) {
    // save the current file
  }
  int myQuit(Event e) {
    // close down
  }
}
going with the “grain”

- system style affects interfaces
  - modal dialogue box (lock app / entire computer until respond => should avoid)
    - easy with event-loop (just have extra read-event loop)
    - hard with notification (need lots of mode flags)
  - non-modal dialogue box
    - hard with event-loop (very complicated main loop)
    - easy with notification (just add extra handler)

beware!
if you don’t explicitly design, it will just happen
implementation should not drive design

Using toolkits

Interaction objects
  - input and output intrinsically linked

Toolkits provide this level of abstraction
  - programming with interaction objects (or techniques, widgets, gadgets)
  - promote consistency and generalizability through similar look and feel
  - amenable to object-oriented programming
interfaces in Java

- Java toolkit – AWT (abstract windowing toolkit)
- Java classes for buttons, menus, etc.
- Notification based;
  - AWT 1.0 – need to subclass basic widgets
  - AWT 1.1 and beyond — callback objects
- Swing toolkit
  - built on top of AWT – higher level features
  - uses MVC architecture (later)

User Interface Management Systems (UIMS)

- UIMS add another level above toolkits
  - toolkits too difficult for non-programmers
- concerns of UIMS
  - conceptual architecture
  - implementation techniques
  - support infrastructure
- non-UIMS terms:
  - UI development system (UIDS)
  - UI development environment (UIDE)
    - e.g. Visual Basic
UIMS as conceptual architecture

• *separation* between application semantics and presentation

• improves:
  • portability – run on different systems
  • reusability – components reused cut costs
  • multiple interfaces – accessing same functionality
  • customizability – by designer and user

UIMS tradition – interface layers / logical components

• linguistic: lexical/syntactic/semantic

• Seeheim:

• Arch/Slinky
Seeheim model

- Presentation
- Dialogue Control
- Functionality (application interface)
- USER
- APPLICATION

conceptual vs. implementation

Seeheim
- arose out of implementation experience
- but principal contribution is conceptual
- concepts part of ‘normal’ UI language

... because of Seeheim ...
... we think differently!

- e.g. the lower box, the switch
  - needed for implementation
  - but not conceptual
semantic feedback

- different kinds of feedback:
  - lexical – movement of mouse
  - syntactic – menu highlights
  - semantic – sum of numbers changes

- semantic feedback often slower
  - use rapid lexical/syntactic feedback

- but may need rapid semantic feedback
  - freehand drawing
  - highlight trash can or folder when file dragged

what’s this?
the bypass/switch

rapid semantic feedback

direct communication between application and presentation
but regulated by dialogue control

more layers!

dialogue

func. core adaptor

lexical

functional core

physical
Arch/Slinky

- more layers! – distinguishes lexical/physical
- like a ‘slinky’ spring different layers may be thicker (more important) in different systems
- or in different components

monolithic vs. components

- Seeheim has big components
- often easier to use smaller ones
  - esp. if using object-oriented toolkits
- Smalltalk used MVC – model–view–controller
  - model – internal logical state of component
  - view – how it is rendered on screen
  - controller – processes user input
MVC issues

- MVC is largely pipeline model:
  - input ➔ control ➔ model ➔ view ➔ output
- but in graphical interface
  - input only has meaning in relation to output
    - e.g. mouse click
    - need to know what was clicked
    - controller has to decide what to do with click
    - but view knows what is shown where!
- in practice controller ‘talks’ to view
  - separation not complete
PAC model

- PAC model closer to Seeheim
  - abstraction – logical state of component
  - presentation – manages input and output
  - control – mediates between them

- manages hierarchy and multiple views
  - control part of PAC objects communicate

- PAC cleaner in many ways …
  but MVC used more in practice
  (e.g. Java Swing)
Implementation of UIMS

- Techniques for dialogue controller
  - menu networks
  - grammar notations
  - declarative languages
  - graphical specification
  - state transition diagrams
  - event languages
  - constraints

- (for most of these see Dix chapter 16)

- N.B. constraints
  - instead of what happens say what should be true
  - used in groupware as well as single user interfaces
  - (ALV: abstraction-link-view)

  see chapter 16 for more details on several of these

graphical specification

- what it is
  - draw components on screen
  - set actions with script or links to program

- in use
  - with raw programming most popular technique
  - e.g. Visual Basic, Dreamweaver, Flash

- local vs. global
  - hard to 'see' the paths through system
  - focus on what can be seen on one screen
The drift of dialogue control

- internal control
  (e.g., read-event loop)

- external control
  (independent of application semantics or presentation)

- presentation control
  (e.g., graphical specification)

Evaluation & critiquing tools

Tullis' Display Analysis Program, Version 4.0:
- Takes alphanumeric screen designs
- produces display-complexity metrics
  - plus some advice:
    - Upper-case letters: "77% The percentage of upper-case letters is high."
    - "Consider using more lower-case letters, since text printed in normal upper- and lower-case letters is read about 13% faster than text in all upper case. Reserve all upper-case for items that need to attract attention."
Evaluation & critiquing tools (cont.)

- Maximum local density = 89.9% at row 9, column 8.
  - Average local density = 67.0%
  - “area with highest local density is identified...you can reduce local density by distributing characters as evenly as feasible over entire screen.”
- Total layout complexity = 8.02 bits
  - Layout complexity is high.
  - “This means that the display items (labels and data) are not well aligned with each other...Horizontal complexity can be reduced by starting items in fewer different columns on the screen (that is, by aligning them vertically).”

Evaluation & critiquing tools (cont.)

- Doctor HTML - Web Page Analyzer:
  - http://imagiware.com/RxHTML
  - “Did not find the required open and close HEAD tag. You should open and close the HEAD tag in order to get consistent performance on all browsers.”
  - “Found extra close STRONG tags in the document. Please remove them.”
Summary

Levels of programming support tools

- Windowing systems
  - device independence
  - multiple tasks
- Paradigms for programming the application
  - read-evaluation loop
  - notification-based
- Toolkits
  - programming interaction objects
- UIMS
  - conceptual architectures for separation
  - techniques for expressing dialogue