Designing Interactive Systems I

Ten Golden Rules: Timeouts, Responsiveness, Notations

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

Winter Semester '20/'21

https://hci.rwth-aachen.de/dis
Review

- What are the Ten Golden Rules of Interface Design?
Ten Golden Rules: Timeouts
Timeouts are Evil!
Timeouts are Evil!
Timeouts: Station Search
Timeouts: Station Search
Timeouts: Writing “Hi” on T9
Timeouts: Writing “Hi” on T9
The Top 5 Performance Hits
Hard Disk Access
Large Memory Footprint
Interlocking Threads
Unsuitable Data and Control Structures
Reinventing the Wheel
Reinventing the Wheel
Notations
Interaction Design Notations

- Alan Dix et al.: Human-Computer Interaction, 3rd ed. (2003), Chapter 16
What are the problems with using such a notation to specify a dialog?

Print "Please enter a number"
INPUT n
Print "The square of",n,"is",n*n
Why UI Specification Languages
Why UI Specification Languages

- In normal programming languages, UI and algorithms are mixed up
Why UI Specification Languages

• In normal programming languages, UI and algorithms are mixed up

• System and user decisions are hard to distinguish
Why UI Specification Languages

- In normal programming languages, UI and algorithms are mixed up
- System and user decisions are hard to distinguish
- Error checking on inputs dominates and complicates code
Why UI Specification Languages

• In normal programming languages, UI and algorithms are mixed up
• System and user decisions are hard to distinguish
• Error checking on inputs dominates and complicates code
• First step: bundling I/O in classes/procedures
Why UI Specification Languages

• In normal programming languages, UI and algorithms are mixed up

• System and user decisions are hard to distinguish

• Error checking on inputs dominates and complicates code

• First step: bundling I/O in classes/procedures

• Second step: Use a more efficient, readable language to specify the dialog
Why UI Specification Languages

- In normal programming languages, UI and algorithms are mixed up
- System and user decisions are hard to distinguish
- Error checking on inputs dominates and complicates code
- First step: bundling I/O in classes/procedures
- Second step: Use a more efficient, readable language to specify the dialog
  - A priori to design the dialog
Why UI Specification Languages

• In normal programming languages, UI and algorithms are mixed up

• System and user decisions are hard to distinguish

• Error checking on inputs dominates and complicates code

• First step: bundling I/O in classes/procedures

• Second step: Use a more efficient, readable language to specify the dialog
  • A priori to design the dialog
  • As part of the implementation (executable spec.)
Specifying User Interfaces

• Problem: Describe the proposed design of a user interface

• Approach: natural/semi-formal/formal languages

• Many standard computer science techniques apply

• The more modern the UI, the harder to describe textually, depending on modality and UI style
Grammars

- Mostly BNF-like

\[
expr ::= \text{empty} \mid \text{atom expr} \mid (\text{expr})' expr
\]
Grammars

- Mostly BNF-like

\[
expr ::= \text{empty} | \text{atom } expr | '( \text{expr} )' \text{expr}
\]

- E.g., Shneiderman's multiparty grammar

\[
\begin{align*}
\langle \text{Session} \rangle & ::= \langle \text{U: Opening} \rangle \langle \text{C: Responding} \rangle \\
\langle \text{U: Opening} \rangle & ::= \text{LOGIN} \langle \text{U: Name} \rangle \\
\langle \text{U: Name} \rangle & ::= \langle \text{U: string} \rangle \\
\langle \text{C: Responding} \rangle & ::= \text{HELLO} [\langle \text{U: Name} \rangle]
\end{align*}
\]
Grammars

• Mostly BNF-like

\[ \text{expr ::= empty} \mid \text{atom expr} \mid '\(' \text{ expr } '\)' \text{ expr} \]

• E.g., Shneiderman's multiparty grammar

\[
\begin{align*}
\langle \text{Session} \rangle & ::= \langle \text{U: Opening} \rangle \langle \text{C: Responding} \rangle \\
\langle \text{U: Opening} \rangle & ::= \text{LOGIN} \langle \text{U: Name} \rangle \\
\langle \text{U: Name} \rangle & ::= \langle \text{U: string} \rangle \\
\langle \text{C: Responding} \rangle & ::= \text{HELLO} [\langle \text{U: Name} \rangle]
\end{align*}
\]

• Great for command-line UIs, e.g., banking ATMs, Unix commands
Grammars

• Mostly BNF-like

\[ expr ::= \text{empty} \mid \text{atom expr} \mid '('< \text{expr} ')'> \text{expr} \]

• E.g., Shneiderman's multiparty grammar

\[ \text{<Session>} ::= \text{<U: Opening>} \text{<C: Responding>} \]
\[ \text{<U: Opening>} ::= \text{LOGIN} \text{<U: Name>} \]
\[ \text{<U: Name>} ::= \text{<U: string>} \]
\[ \text{<C: Responding>} ::= \text{HELLO} [\text{<U: Name>}] \]

• Great for command-line UIs, e.g., banking ATMs, Unix commands

• Less suitable for GUIs
Grammars

• Regular expressions
  • select-line click click* double-click
Grammars

• Regular expressions
  
  • select-line click click* double-click

• E.g., Unix “copy” command synopsis:

  cp [-R [-H | -L | -P]] [-f | -i | -n] [-pv] source_file target_file
  cp [-R [-H | -L | -P]] [-f | -i | -n] [-pv] source_file ... target_dir
Grammars

• Regular expressions
  • select-line click click* double-click

• E.g., Unix “copy” command synopsis:

\[
\begin{align*}
\text{cp} & \ [\text{-R} \ [\text{-H} \mid \text{-L} \mid \text{-P}]] \ [\text{-f} \mid \text{-i} \mid \text{-n}] \ [\text{-pv}] \ \text{source\_file} \ \text{target\_file} \\
\text{cp} & \ [\text{-R} \ [\text{-H} \mid \text{-L} \mid \text{-P}]] \ [\text{-f} \mid \text{-i} \mid \text{-n}] \ [\text{-pv}] \ \text{source\_file} \ \ldots \ \text{target\_dir}
\end{align*}
\]

\[\begin{align*}
\text{recursion policies} & \hspace{1cm} \text{overwrite policies}
\end{align*}\]
Grammars

• Regular expressions
  • select-line click click* double-click

• E.g., Unix “copy” command synopsis:

  cp [-R [-H | -L | -P]] [-f | -i | -n] [-pv] source_file target_file
  cp [-R [-H | -L | -P]] [-f | -i | -n] [-pv] source_file ... target_dir

  recursion policies \hspace{1cm} overwrite policies

• Short and precise, but hard to read, requires additional information about semantics
Production Rules

- Unordered list of rules: \textit{if condition then action}
  - Condition based on state or pending events
  - Every rule always potentially active

- Good for concurrency

- Bad for sequence
Event-based Production Rules
Event-based Production Rules

select-line → first
Event-based Production Rules

select-line → first
click first → rest
Event-based Production Rules

select-line → first
click first → rest
click rest → rest
Event-based Production Rules

select-line → first
click first → rest
click rest → rest
double-click rest → <draw line>
Event-based Production Rules

select-line → first
click first → rest
click rest → rest
double-click rest → <draw line>

• Note:
  • Events added to list of pending events
  • ‘first’ and ‘rest’ are internally generated events
Event-based Production Rules

select-line → first
click first → rest
click rest → rest
double-click rest → <draw line>

• Note:
  • Events added to list of pending events
  • ‘first’ and ‘rest’ are internally generated events
  • Bad at state!
Graph Notations: STNs

- State Transition Networks (STNs)
  - Most common tool to specify dialogs
  - Established format (since 1960s)

- Consisting of:
  - **States** (usually the system waiting for some user action)
  - **Transitions** (which have a **user action** and a **system response** associated with them)

- Describes **sequences** of user actions and system responses
Graph Notations: STNs

Start → Menu

Select 'circle'

Menu → Circle 1

Click on centre
Rubber circle

Line 1

Select 'line'
Click on first point
Rubber hand

Circle 1 → Circle 2
Click on circumference
Draw circle

Circle 2 → Finish

Click on centre
Rubber circle

Line 2

Double click
Draw last line

Click on point

Line 1 → Line 2
Draw line
Rubber hand

Finish
Example: STN for Personal Orchestra Dialog

Legend:

STATE NAME
Description of what users do

program start / show rehearsal video loop

FSM_INTROLOOP

FSM_INTROLOOPENDS

FSM_LANGCHOSEWAIT

FSM_PIECECHOOSEWAIT

FSM_INFOPAGE

FSM_WAITFORCONDUCTOR

FSM_CONDUCTINGSTARTED

FSM_CONDUCTING

FSM_COMPLAINT

FSM_CONDUCTINGENDS

baton button clicked / wait for video loop to end

video loop ends / switch to language selection

baton button clicked / switch to piece selection

choose preferred language

choose musical piece or info page

baton button clicked / determine selection

yes / switch to info page

no / switch to orchestra waiting

requested info page?

baton button clicked / switch to piece selection

baton button clicked / switch to piece selection

baton button clicked / switch to piece selection

complaint ends / display current piece selection

complaint ends / display current piece selection

conducted first beat / –

conducting has started / start audio and video

conducted too fast or too slow / jump to complaint sequence

completed successfully / display congratulations page

input / output

input / output

input / output

input / output

input / output

Example: STN for Personal Orchestra Dialog

Program start / show rehearsal video loop

1. FSM_INTROLOOP
   - Baton button clicked / wait for video loop to end

2. FSM_INTROLOOPENDS
   - Video loop ends / switch to language selection

3. FSM_LANGCHOSEWAIT
   - Baton button clicked / switch to piece selection

4. FSM_PIECECHOOSEWAIT
   - Baton button clicked / determine selection

5. FSM_INFOPAGE
   - Check if info page is requested

6. FSM_WAITFORCONDUCTOR
   - Conducting has started / start audio and video

7. FSM_CONDUCTINGSTARTED
   - Conducting too fast or too slow / jump to complaint sequence

8. FSM_COMPLAINT
   - Completed successfully / display congratulations page

9. FSM_CONDUCTINGENDS
   - Switch back to orchestra waiting

Legend:

STATE NAME
Description of what users do
program start | show rehearsal video loop

FSM_INTROLOOP

baton button clicked / wait for video loop to end

idle timeout | switch to rehearsal video

FSM_INTROLOOPENDS

video loop ends / switch to language selection

idle timeout | switch to rehearsal video

FSM_LANGCHOOSEWAIT

choose preferred language

idle timeout | switch to rehearsal video

FSM_LANGCHOOSE

-b | continuously display current piece selection

baton button clicked / switch to piece selection

idle timeout | switch to rehearsal video

FSM_PIECECHOOSEWAIT

choose musical piece or info page

baton button clicked / determine selection

thanx Timeout | switch to rehearsal video

FSM_PIECECHOSSEN
Please stand on the footprints
Bitte auf die Fußabdrücke stellen
请大家站好位置
Please stand on the footprints
Bitte auf die Fußabdrücke stellen
请大家站好位置
Silhouettes: Server STN

- Unconventional notation (agreed upon in the team)

Legend:
- : operator pushes button

Note:
- server is responsible to fade sound in each state
Prof. Dr. Jan Borchers: Designing Interactive Systems I • WS 2020/21

Silhouettes: Client STN

(any state other than maintenance states)  

Init (maintenance state)  
- run POST  
- start Fl.  
- connect to Fl.  
- look for server  
- look for web server

WaitForServer  
- query state from server  
- display error on client

Error (maintenance state)  
- display error on client

srv. msg. "GotoStateM"  

(stateM)  

ElectricityPosing  
- send state → Fl.  
- display shadow ©

ElectricityCountdown  
- overlay 3-2-1  
- display shadow ©

ElectricityResult  
- send "illuminate building" → Fl.

Reconnecting (maintenance state)  
- reconnect to server ©

srv. msg. "GotoStateElectricityPosing"

electricityPosing  
- send state → Fl.  
- display shadow ©

ElectricityCountdown  
- overlay 3-2-1  
- display shadow ©

ElectricityResult  
- send "illuminate building" → Fl.

client timer

srv. msg. "GotoStateElectricityCountdown"

got server connection

got server connection

connection to server lost

any failure
<table>
<thead>
<tr>
<th>Visuals</th>
<th>Description</th>
<th>Server State/Action</th>
<th>Client State/Light</th>
</tr>
</thead>
</table>
| Waiting people will be asked to move up to the front and take their positions on the indicated spots. | PleaseMoveUp | • generate game ID  
• spotlight on  
• play sound (operator fill backseats) | DMX msg.  
- fade in spots on indicated footprints on the floor |
| Playing people will be presented a welcome screen. | Welcome | • send state → Fl. | srv. msg.  
"GotoStatePleaseMoveUp" |
| People from the outer waiting queue will take their seats in the inside waiting booths. | Welcome | • send state → Fl.  
• display mnemonics | DMX msg.  
- fade out light in the waiting booths |
Iconic illustrations will remind the player how to create buildings by their shadows.

People will see their shadows while posing in front of the screens.

A countdown will indicate that shadows will be "frozen".

People will see the representation of their shadows as buildings.

BuildingPosing
- spotlight off
- booth light off

BuildingCountdown
- overlay 3-2-1
- display shadow ©
- send feature vector → Fl. ©

BuildingResult
- send "assemble building" → Fl.

BuildingDescription
- send state → Fl.

GotoStateBuildingPosing
- srv. msg. "GotoStateBuildingPosing"

GotoStateBuildingCountdown
- srv. msg. "GotoStateBuildingCountdown"

GotoStateBuildingDescription
- srv. msg. "GotoStateBuildingDescription"
A countdown will indicate that shadows will be "frozen".

People will see illuminated buildings in 4 different versions.

Iconic illustrations will remind the player how to pose in order to have their shadows overlap and thus create a power line.

People will see their shadows while posing in front of the screens.
Iconic illustrations will remind the player how to create trees by their shadows.

People will see their shadows while posing in front of the screens.

A countdown will indicate that shadows will be “frozen”.

People will see the representation of their shadows as trees.
Iconic illustrations will remind the player how to pose in order to have their shadows overlap and thus create a water line.

People will see their shadows while posing in front of the screens.

A countdown will indicate that shadows will be "frozen".

People will see the trees that they created with different green colours/trees with leaves?
People will see an animation of what will happen with their skyline.

People will see their city block with the block number and the URL of the DuC Website.
People will be asked to move out.

Beyond the playing area, visitors can deepen their understanding of city concepts at individual interactive stations before leaving the pavilion.
Hierarchical STNs

- **Start** and **Finish** states serve to glue an STN for a sub dialog (e.g., a certain menu selection) into a larger dialog (e.g., operating the application in general)

- Same expressive power as STNs, just more convenient

- The dialog structure of an entire system can be specified this way
Using STNs in Prototyping

• Create a simple STN for the dialogs envisioned

• Create one UI snapshot (sketch if paper prototype) per state (label it with the state name)

• Include offscreen area for annotations and to include extra buttons simulating user actions that do not correspond to simple clicks on the current screen

• When walking the user through your paper prototype, consult the STN to find out how to respond to each user action
Using STNs in Prototyping

- Alternative: Let the computer “execute” the STN to run the prototype
- Use tools such as Keynote, PowerPoint
select 'graphics'

Main Menu

select 'text'

select 'paint'

Graphics Submenu

Text Submenu

Paint Submenu

Start

Menu

select 'circle'

Circle 1

click on centre
rubber circle
click on
first point
rubber hand
click on point
draw line
rubber hand
double click
draw last line

Line 1

Line 2

Circle 2

click on circumference
draw circle

Finish

Finish
Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state: Main Menu
Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state:  Main Menu

<table>
<thead>
<tr>
<th>graphics</th>
<th>text</th>
<th>paint</th>
</tr>
</thead>
<tbody>
<tr>
<td>circle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>line</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Drawing example:
- Circle
- Line
Using STNs in Prototyping
Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state: Circle 1

Click this button to simulate a click on the drawing area.

Click on centre
Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state: Circle 2

Click this button to simulate a click on the drawing area.

Click on circumference
Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state: End of drawing

The circle is drawn now.
Click the button to go back to main menu.

end of sub-state
Using STNs in Prototyping

Adapted from “Human–Computer Interaction” by Dix, Finlay, Abowd, and Beale, Chapter 8

Current state:
- Main Menu
- Circle 1
- Circle 2
- End of drawing

Current state:
- graphics
- text
- paint

Drawing:
- Circle 1
- Circle 2
- End of drawing
Checking STN Properties: States
Checking STN Properties: States

• Completeness
  • Can you get anywhere from anywhere?
  • Are all possible actions covered in every state?
  • How easily?
Checking STN Properties: States

- Completeness
  - Can you get anywhere from anywhere?
  - Are all possible actions covered in every state?
  - How easily?

- Reversibility
  - Can you get to the previous state?
  - But NOT undo
Checking STN Properties: States

• Completeness
  • Can you get anywhere from anywhere?
  • Are all possible actions covered in every state?
  • How easily?

• Reversibility
  • Can you get to the previous state?
  • But NOT undo

• Dangerous states
  • Some states you don’t want to get to
Checking Transition Properties: Completeness

- Missing arcs indicate unspecified user input
- What happens when the user double-clicks in the circle states?
Checking Transition Properties: Completeness

- Missing arcs indicate unspecified user input
- What happens when the user double-clicks in the circle states?
Checking Transition Properties: Completeness

• Missing arcs indicate unspecified user input
  • What happens when the user double-clicks in the circle states?
Checking Transition Properties: Reversibility

- E.g., reversing select ‘line’ requires
  \textit{Click} - \textit{double click} - select ‘graphics’ (3 actions)
- Note: Reverse means just getting back to a state, \textbf{not} to “undo” its effect
Checking Transition Properties: Reversibility

• E.g., reversing select ‘line’ requires
  *Click - double click - select ‘graphics’* (3 actions)

• Note: Reverse means just getting back to a state, **not** to “undo” its effect
Checking Transition Properties: Reversibility

• E.g., reversing select ‘line’ requires
  \textit{Click - double click - select ‘graphics’} (3 actions)

• Note: Reverse means just getting back to a state, \textbf{not} to “undo” its effect
Checking Transition Properties: Reversibility

• E.g., reversing select ‘line’ requires
  \textit{Click - double click - select ‘graphics’} (3 actions)

• Note: Reverse means just getting back to a state, \textbf{not} to “undo” its effect

---

Prof. Dr. Jan Borchers: Designing Interactive Systems I • WS 2020/21
Checking Transition Properties: Reversibility

• E.g., reversing select ‘line’ requires
  Click - double click - select ‘graphics’ (3 actions)

• Note: Reverse means just getting back to a state, **not** to “undo” its effect
Dangerous States Example

• Word processor: two modes and exit
  • F1 - changes mode
  • F2 - exit (and save)
  • Esc - no mode change

• But ... Esc resets autosave
Dangerous States Example

• Exit with/without save ⇒ dangerous states

• Duplicate states - semantic distinction

• F1-F2 - exit with save

• F1-Esc-F2 - exit with no save
Dangerous States Example: Layout Matters

old keyboard - OK

edit \[\xrightarrow{F1}\] menu \[\xrightarrow{F2}\] exit

edit \[\xrightarrow{F1}\] menu

any update

exit

F2

Esc

F1

Esc

F1

F2
Dangerous States Example: Layout Matters

new keyboard layout

Intend F1-F2 (save)

Finger catches Esc

F1-Esc-F2 - disaster!
Checking STN Properties: Other Transition Properties

- Determinism
  - Several arcs for one action
    - Deliberate: application decides
    - Accidental: production rules
- Nested escapes
- Consistency
  - Same action, same effect?
  - Modes and visibility
In-Class Exercise: STN

- Simple dialog to select bold, italics, and/or underline

- Draw the state diagram for:
  - Only Bold checkbox
  - Bold and italics checkboxes
  - All three checkboxes
**Bold Checkbox**

![Diagram showing a transition from regular style to bold text]

- regular style
- "bold"
- bold

![Screenshot of a text style dialog box with options for Bold, Italic, and Underline]
Bold & Italic Combined

regular style → bold
italic → bold italic
bold → italic
bold italic → italic

Preview

Text Style

- Bold
- Italic
- Underline
All Three Options
Adding Another Option…

Normal case

CAPITALIZED
STNs: State Explosion

- Problem: Combining two concurrent STNs with N and M states leads to new STN with $N \times M$ states
- STN hides clear structure of the dialog
- Especially problematic with modern GUIs
- Similar problems with “Escape” and “Help” options
  - ESC can be modeled as special second “Finish” exit active throughout subdialog
  - Help can be modeled as little subdialog hanging off every single state in the STN
- Gets messy
Example: ESC & Help in STNs

- **Menu**
  - select ‘graphics’
  - select ‘text’
  - select ‘paint’

- **Graphics Submenu**
  - ESC
  - normal finish

- **Text Submenu**
  - ESC
  - normal finish

- **Paint Submenu**
  - ESC
  - normal finish

- **Help Subsystem**
  - press HELP button

- **Main Menu**
  - press HELP button

- **Finish**
  - click on circumference
  - draw circle
  - ESC
  - normal finish

- **Circle 1**
  - click on centre
  - rubber band

- **Circle 2**
  - press HELP button
  - click on circumference
  - draw circle

- **Help Subsystem**
  - press HELP button

- **from Menu**

---

Prof. Dr. Jan Borchers: Designing Interactive Systems I • WS 2020/21
Petri Nets

- Better approach to dialogs that have several states at once
- But not better for sequential dialogs and mutually exclusive UI elements (radio buttons)
- Relatively old formalism to model concurrency
Petri Nets

- Transition fires when all input places have one or more token
  - A token is produced in each output place
- Positions of all tokens represent the current state
  - NOTE: This is different from state machines
Petri Nets

- Transition fires when all input places have one or more token
  - A token is produced in each output place
- Positions of all tokens represent the current state
  - NOTE: This is different from state machines
Petri Nets

- Transition fires when all input places have one or more token
  - A token is produced in each output place

- Positions of all tokens represent the current state
  - NOTE: This is different from state machines
Petri Nets

- Transition fires when all input places have one or more token
  - A token is produced in each output place
- Positions of all tokens represent the current state
  - NOTE: This is different from state machines

Tokens disappear from input places
Petri Nets

• Transition fires when all input places have one or more token
  • A token is produced in each output place
• Positions of all tokens represent the current state
  • NOTE: This is different from state machines
In-Class Exercise

Draw the Petri net for our dialog box with concurrent “Bold” and “Italic” options (ignore “Underline” for now)
Petri Net For “Bold & Italic” Dialog

User presses ‘Bold’

Bold On

T1

T2

Bold Off
Petri Net For “Bold & Italic” Dialog

Ellipse: place where user input can occur

User presses ‘Bold’

Circle: UI place

Rectangle: transition

Token

Bold On

Bold Off

T1

T2
User presses ‘Bold’

User actions represented as a new token

Petri Net For “Bold & Italic” Dialog
Petri Net For “Bold & Italic” Dialog

User presses ‘Bold’

Bold On

Bold Off

Transition ‘fires’ when all input places have tokens
A token is consumed from each input place

User presses ‘Bold’

T1

T2

Petri Net For “Bold & Italic” Dialog

Bold On

Bold Off

Bold On

Bold Off

User presses ‘Bold’

T1

T2
Petri Net For “Bold & Italic” Dialog

User presses ‘Bold’

A token is produced in each output place
Petri Net For “Bold & Italic” Dialog

User presses ‘Bold’

User presses ‘Italic’
State Charts

- By Harel; used in UML
- Example: TV Control Panel
- State Charts extend STNs
  - Hierarchy
  - Concurrent sub-nets
    - ON resumes both state machines
  - Escapes
    - OFF always active
  - History
    - Link marked “H” goes back to last state on re-entering subdialog
Diagrams For User Documentation

• Some dialog descriptions are clear enough to serve as user documentation (similar to GOMS)

• Especially if description uses screen shots and is semi-formal
Digital Watch – User Instructions

- Two main modes
- Limited interface
  - 3 buttons
- Button A changes mode
Dangerous states

Completeness

Distinguish depress A and release A

What do they do in all modes?
Digital Watch – User Instructions

and… that’s just one button
Semantics - Raw Code

- Event loop for word processor
- Dialogue description: very distributed
- Syntactic/semantic trade-off: terrible!

```c
switch ( ev.type ) {
    case button_down:
        if ( in_text ( ev.pos ) ) {
            mode = selecting;
            mark_selection_start(ev.pos);
        }
        ...
    case button_up:
        if ( in_text ( ev.pos )
            && mode == selecting ) {
            mode = normal;
            mark_selection_end(ev.pos);
        }
        ...
    case mouse_move:
        if (mode == selecting ) {
            extend_selection(ev.pos);
        }
        ...
} /* end of switch */
```
Further Reading

• Alan Dix et al.: Human-Computer Interaction, 3rd ed. (2003), Chapter 16

• Ben Shneiderman: Designing The User Interface, 5th ed. (2009), esp. Chapter 5
Roadmap

Human

- Performance
- Models of interaction
  - Affordances
  - Mappings
  - Constraints
  - Types of knowledge
- Errors
- Visual Design

Case Studies

- History of HCI
- Visions
- Phases of Technology

Development Process

- Iterative design
- User observation
- Ideation
- Prototyping
- User studies and evaluation
- Interaction design notation

Diagram:
- Goal
- Plan
- Compare
- Specify
- Interpret
- Perform
- Perceive
- World

Prof. Dr. Jan Borchers: Designing Interactive Systems I • WS 2020/21
What’s Next?

• **Designing Interactive Systems 2** (6 ECTS)
  https://hci.rwth-aachen.de/dis2
  - What makes a UI tick?
  - Technical concepts, software paradigms and technologies behind HCI and user interface development

• **Current Topics in HCI** (6 ECTS)
  https://hci.rwth-aachen.de/cthci
  - Understand & practice ways to do research in HCI
  - Learn about up-to-date developments in HCI and interactive multimedia from new books and recent conference/journal articles

• **Self-study in Summer 2021!**
What’s Next?

• **Designing Interactive Systems 2** (6 ECTS)
  https://hci.rwth-aachen.de/dis2
  
  • What makes a UI tick?
  
  • Technical concepts, software paradigms and technologies behind HCI and user interface development

• **Current Topics in HCI** (6 ECTS)
  https://hci.rwth-aachen.de/cthci
  
  • Understand & practice ways to do research in HCI
  
  • Learn about up-to-date developments in HCI and interactive multimedia from new books and recent conference/journal articles

• **Self-study in Summer 2021!**

Interested in a HiWi position or B.Sc./M.Sc. thesis?
https://hci.rwth-aachen.de/jobs