Interaction and Design Concepts

# Interaction Types:

* Communication – The user is giving commands or conversing with the computer.
  + Interaction as *instruction*.
    - CLI, Button Pushing, Speech Commands, Menu Selection etc.
  + Interaction as *conversation*.
    - Phone Menu Systems, Siri etc.
    - Computer feedback may be substantial and computer may not directly follow instructions; It may request clarification, counter-propose etc.
    - Speech Act Theory:
      * Assert – state an existing truth
      * Direct – request/command
      * Commit – make a promise
      * Express – state attitude/emotion
      * Declare – define a new truth
* Manipulation – The user is manipulating virtual objects or moving virtual self.
  + Interaction as *object manipulation*.
    - Users manipulate virtual objects, items or tokens. Typically limited to touch-based manipulation.
    - Characterized by continuous, immediate feedback to indicate changing states of the system.
    - Drag & Drop, Two-Finger Pinch, Rotation etc.
  + Interaction as *ego manipulation*.
    - Users move their mental selves around.
    - Using Hyperlinks, 3D Games, Car Simulators etc.
* Peripheral Feedback
  + Interfaces can provide peripheral feedback, which need little attention and may be ignored.
  + Modeless Feedback – Feedback that does not switch interface mode. Status Bar Information, Tooltips etc.
  + Augmented Reality – Camera video augmented by labeled info. Car HUD, Building Names in Live Video, HoloLens etc.
  + Ambient Devices / Calm Computing – Glanceable output, handled pre-attentively in user’s peripheral vision. Power Aware Cord, Ambient Orb etc.
* Interaction styles can be differentiated by:
  + Interactivity – How actively does the interface engage the user?
    - Related to rate of the human-computer interaction cycle. Depends on input & response time.
    - Communication-based interaction has a slow, turn-taking cycle between users & computers.
    - Manipulation-based interaction has a very rapid cycle as feedback is continuous.
    - Greater interactivity is more engaging but also computationally costly.
    - Communication-based Interaction:
      * 0 – 0.1 secs: Instantaneous
      * 0.1 – 1 secs: Responsive
      * 1 – 10 secs: Slow
      * > 10 secs: Non-Interactive
    - Manipulation-based Interaction:
      * Response time is also the latency (delay).
      * Users experience distracting lag from around 170ms.
  + Intuitiveness – How quickly & easily can a user learn to use the interface or carry out tasks with it?
    - Related to the ease of using (or learning to use) of an interface.
    - Often about users’ familiarity with previous software.
    - UIs should apply for as wide a range of past experiences as possible.
      * Tennis with the Wiimote is easily picked up by many diverse users versus gamepad controls.
    - Expert users can usually do well with less intuitive interfaces as compared to novice or normal users.

# Software Behavior:

* Software Posture:
  + Sovereign – Monopolize user’s attention for extended duration. Most major applications do this.
  + Transient – Briefly capture the user’s attention from time to time. Chat Notification, Sidebar Gadgets etc.
  + Daemonic – Mostly silent/invisible, very rarely capturing the user’s attention. Network & Volume Icons in System Tray etc.
* Empathetic Software Behavior:
  + Frustration/Anger – due to “excise” (unnecessary effort or difficulty).
  + Fear/Distress – due to having done a wrong but irreversible action.
  + Best solution is to minimize such affects, however, if unavoidable, mitigation is still possible via empathetic error messages.
* Considerate Software Behavior:
  + Better to design considerate software behavior in the first place.
  + Proactive – know user’s habits & anticipate needs. (auto-fill, preloading etc.)
  + Flexible – easily reversible, don’t force the user, be adaptive & fail gracefully.
  + Deferential – avoid informing user of non-critical internal problems & asking unnecessary questions.
* Anthropomorphism:
  + Giving human attributes to software.
  + Suitable for some groups of users and some application types.
  + Gender and personality must be considered.
  + Examples: Chatbots, Avatars, Virtual Assistants etc.

# Design Concepts:

* Affordances – attribute of an item, object or structure, permitting a user to perform some basic action.
  + Well-designed items have clearly perceived affordance. For eg. button = push, switch = flip, slider = slide etc. Users instinctively know how to use them.
  + Affordance does not define effect.
* Metaphors – computer interactions mapping to real-world interactions that users are more familiar with.
  + Advantages:
    - Reduces gulfs of execution and evaluation.
    - Users understand better what can or cannot be done.
  + Disadvantages:
    - Bad metaphors can confuse users.
  + Examples:
    - Order List in eCommerce 🡪 Shopping Cart
    - Recycle Bin 🡪 Icon of Trash Can
* Idioms – very widely-used actions that span different platforms/software.
  + Examples:
    - Resize windows by dragging on borders.
    - Scroll window views via scroll bars.
    - WASD keys in 3D FPS games.
* Choice Limitation –
  + Keyboard and pointer actions are very unconstrained so how does a novice user know what to do or how to avoid errors/mistakes.
  + Either get users to learn idiomatic actions or provide noticeable constraints to lead users towards allowable actions, such as checkboxes or radio buttons.
  + Games have evolved from having extremely limited allowable actions to *infinite* choices.
* Context Awareness –
  + Choice limitation may not be enough. The interface can use context to help distinguish between similar actions.

# Design Patterns:

* Reusable past solutions to design problems. Not just the visuals, but also the core ideas/techniques.
  + Design patterns arise from organic, collective knowledge of what works and what doesn’t, from past experiences of other designers.
  + Users tend to be familiar with some interface design patterns and implementing them makes the UI easier to use.
* Templates:
  + Near-complete design prototypes.
  + Easiest to be used directly, hardest to adapt to different situations.
* Widgets/Controls:
  + Basic high-level GUI components.
  + Buttons, sliders, textboxes, radio buttons, checkboxes etc.
  + Allows designers to think with a higher-level vocabulary (of widgets).
  + Graphical User Interface Builders:
    - Software to visually design UIs using widgets from a library.
    - Able to generate some automatic code but usually doesn’t enforce how a good UI can be built.
* Design Language & Guidelines:
  + Scheme to define/design a consistent look and feel for UIs.
  + Guidelines determine the design language.
    - Prescribe expected look and feel of app UIs.
    - Recommend when and how widgets should be used and combined.
* Pattern Language:
  + Integrated set of “patterns” to solve complex design problems. Higher level patterns tend to “unfold” into lower level patterns for sub-problems.
  + Each pattern is usually presented in sections:
    - Context – Where is the problem encountered?
    - Problem – What is the problem?
    - Solution(s) – Set of solution steps & key considerations.
  + Non-experts can use this to generate good design.
  + Pattern languages now widely used in software engineering and user interaction design.