UNIT — II SOFTWARE PROCESS, MODELS AND THEORIES (09 Hours)

HCI in software process — software life cycle — usability engineering — Prototyping in practice — design rationale. Cognitive models —Socio-Organizational issues and stake holder requirements — Communication and collaboration models. Keystroke level model (KLM), GOMS, CASE STUDIES. Shneiderman's eight golden rules; Norman's seven principles; Norman's model of interaction; Neilsen's ten heuristics with example of use.

1. HCI in the Software Process

Human-Computer Interaction (HCI) should not be treated as just "UI design" — it is integrated into every stage of the Software Development Life Cycle (SDLC).

a) Software Life Cycle (SLC) with HCI perspective

1. Requirements Analysis
    - Collect functional + usability requirements.
    - Use techniques: interviews, surveys, contextual inquiry, ethnographic study.
    - *Real-time Example*: In developing an online exam portal, students want simple navigation, faculty want easy question upload, and admins need secure access control.
2. System Design
    - Create user interface architecture, workflows, prototypes.
    - Apply HCI principles: consistency, feedback, affordances.
    - *Example*: Google Pay app designed with minimal screens to reduce cognitive load.
3. Implementation
    - Developers implement UI + functionality.
    - HCI ensures accessibility (font size, colors for visually impaired, keyboard shortcuts).
    - *Example*: Microsoft Word provides multiple input modes (typing, voice dictation).
4. Testing
    - Functional + usability testing.
    - Methods: Heuristic evaluation, Think-aloud study, Eye-tracking.
    - *Example*: A hospital software tested by doctors → navigation improved after doctors complained about too many clicks to view patient history.
5. Deployment & Maintenance
    - Continuous improvement using user feedback.
    - *Example*: WhatsApp added dark mode after repeated user requests.

b) Usability Engineering

Usability Engineering is a structured approach to ensure usability goals are met.

Key usability attributes (ISO 9241-11):

- Effectiveness: Can users complete tasks correctly?
- Efficiency: How quickly and easily?
- Satisfaction: Are users happy with the experience?

***Real-time Example*:**

- Swiggy/Zomato → Effective (orders delivered correctly), Efficient (few steps to checkout), Satisfying (intuitive app).
- Government portals (like passport services) often fail in efficiency (too many steps, slow servers).

c) Prototyping in Practice

Prototypes = simplified versions of the product to test ideas early.

- Low-fidelity prototypes:
  - Rough sketches or paper-based wireframes.
  - *Example*: Paper prototype of an ATM screen to test withdrawal flow.
- High-fidelity prototypes:
  - Interactive, close to final design (made in Figma, Adobe XD).
  - *Example*: Clickable prototype of an e-commerce app with category browsing, cart, checkout.

Benefits: Saves cost, allows stakeholder feedback, reduces risk of poor design.

d) Design Rationale

Design Rationale = documenting why design choices were made.

- Provides reasoning for decisions → helps in future modifications.

***Example*:**

- A flight booking site may use calendar-based date selection instead of text input because usability tests showed users made fewer errors.
- Recording the rationale prevents future designers from reverting to error-prone designs.

2. Cognitive Models

Cognitive models describe how users think, perceive, and act when interacting with computers.

a) Socio-Organizational Issues & Stakeholder Requirements

Systems exist in organizations → multiple stakeholders with different goals.

- Socio-organizational issues:
    - User resistance to change
    - Training requirements
    - Data security concerns
    - Organizational hierarchy affecting access

Example:

- In a hospital information system:
    - Doctors want quick access to records (speed).
    - Administrators want strict access control (security).
    - Patients want privacy.
- Balancing these needs is part of usability engineering.

b) Communication & Collaboration Models

Software must support teamwork, communication, and collaboration.

- Synchronous communication: Real-time. *(Zoom meetings, Slack calls)*
- Asynchronous communication: Delayed. *(Email, discussion forums)*

Examples:

- Google Docs → supports real-time editing (synchronous) + comments (asynchronous).
- GitHub → developers collaborate via commits, pull requests, issue tracking.

3. Keystroke-Level Model (KLM)

1. Introduction

- The Keystroke-Level Model (KLM) was proposed by **Stuart Card, Thomas Moran, and Allen Newell** (part of the GOMS family of models).
- It is used to **estimate the time** an expert user (who makes no mistakes) will take to perform a task on a computer system.
- Unlike usability testing (which needs real users), KLM is **predictive** — it can be applied by designers before building the system.

👉 It is often used to compare **two different designs** or **interaction methods** (e.g., shortcut vs menu navigation).

2. Basic Idea

The model breaks down a task into a sequence of **primitive actions (operators)**. Each operator has a **standard execution time** (based on experimental data). Adding up the times of all operators = **total predicted task time**.

3. Operators in KLM

There are six main operators:

Operator	Meaning	Average Time	Example
K	Keystroke (keyboard key press)	0.2 sec	Typing "A"
P	Point with mouse to a target	1.1 sec	Moving cursor to "Submit" button
H	Homing (switch between keyboard & mouse)	0.4 sec	Taking hand from mouse to keyboard
M	Mental preparation (thinking before action)	1.35 sec	Planning the next step
R	System response time (waiting for computer)	Variable	Page loading
B	Button press (optional in some versions)	0.1 sec	Pressing mouse button

👉 Times are **average values**; in practice they may vary based on skill level, environment, and device.

4. Assumptions of KLM

1. User is **skilled** (knows the task, no learning time).
2. User works **without errors**.
3. The task is performed in a **routine and predictable way**.
4. Context like distractions, fatigue, or environment is ignored.

👉 This makes KLM best for **expert users doing frequent tasks**.

5. Steps in Applying KLM

1. Identify the **task** (e.g., saving a file, searching on Google).
2. Break it into a sequence of **primitive actions** (operators).
3. Assign **standard times** to each operator.
4. Sum up the times to get **predicted task completion time**.
5. Compare times for **alternative designs**.

6. Examples

Example 1: Saving a file using Menu

Steps:

- Mental preparation (M = 1.35s)
- Point to “File” menu (P = 1.1s)
- Click (K = 0.2s)
- Point to “Save” (P = 1.1s)
- Click (K = 0.2s)
- Wait for system response (R = 0.5s)

Total = 4.45s

Example 2: Saving a file using Shortcut (Ctrl+S)

Steps:

- Mental preparation (M = 1.35s)
- Keystroke Ctrl (K = 0.2s)
- Keystroke S (K = 0.2s)
- System response (R = 0.5s)

Total = 2.25s

👉 Clearly, **shortcut is almost twice as fast** as menu navigation.

Example 3: Google Search

Task: Search for "HCI models" in browser.
Steps:

1. Mental preparation (M = 1.35s)
2. Point to search box (P = 1.1s)
3. Click in search box (K = 0.2s)
4. Type “HCI models” (10 keystrokes × 0.2s = 2.0s)
5. Press Enter (K = 0.2s)
6. Wait for system response (R = 1.0s)

Total = 5.85s

7. Applications of KLM

- **Interface Evaluation**: Compare design alternatives.
- **Usability Prediction**: Estimate efficiency without user testing.
- **Training**: Identify which tasks are better done via shortcuts.
- **Cost-benefit analysis**: Justify investment in design improvements.

8. Limitations of KLM

1. Assumes **expert, error-free performance** → not realistic for novices.
2. Ignores **learning time, errors, emotions, distractions**.
3. Focuses only on **execution time**, not user satisfaction.
4. Does not apply well to **complex, creative tasks** (e.g., writing an essay, designing).

9. Real-Time Industry Examples

- **Text editors**: Designers decide whether to promote menu navigation or keyboard shortcuts.
- **ATM machines**: Sequence of screens analyzed using KLM to minimize time per transaction.
- **E-commerce sites**: Checkout flow optimized (fewer clicks = faster = higher conversion).
- **Call center software**: Operators save time if frequent actions (like opening customer record) are reduced from 6 clicks → 3 clicks.

✅ In short:
KLM = **mathematical stopwatch** for predicting how fast users can perform a task. It helps designers choose between alternatives and optimize for **efficiency**.

4. GOMS Model in HCI

1. Introduction

- The **GOMS model** (developed by Card, Moran, and Newell in 1983) is a **cognitive modeling technique**.
- It helps describe and analyze **how users interact with a system** to achieve goals.
- GOMS is part of the **Model Human Processor (MHP)** framework, which aims to predict human behavior in computer interaction.

👉 GOMS is especially useful for **routine, well-defined tasks** where users are reasonably skilled.

2. What is GOMS?

GOMS = Goals, Operators, Methods, Selection rules

1. **Goals** → What the user wants to achieve.
    - Example: *Print a document, send an email, order food online.*

2. **Operators** → Basic actions needed to achieve the goal.
    - Examples: keystrokes, mouse clicks, mental preparation, eye movements.
3. **Methods** → Procedures (sequences of operators) for achieving a goal.
    - Example:
        - Method 1: Use **Ctrl+P** to print.
        - Method 2: Navigate to **File → Print**.
4. **Selection Rules** → Decide which method to use when there are multiple.
    - Example:
        - *Expert users* → choose shortcut.
        - *Novice users* → choose menu navigation.

3. Types of GOMS Models

There are **different variants** of GOMS depending on detail level:

1. **Keystroke-Level Model (KLM)**
    - Simplest GOMS variant.
    - Only considers execution time of low-level actions.
    - Example: Typing Ctrl+S vs File → Save.
2. **CMN-GOMS (Classic GOMS)**
    - Original form by Card, Moran, and Newell.
    - Hierarchical goals, methods, and operators.
3. **NGOMSL (Natural GOMS Language)**
    - Describes GOMS in structured English.
    - More readable for designers.
4. **CPM-GOMS (Cognitive Perceptual Motor GOMS)**
    - Advanced version that considers parallel actions.
    - Example: Looking at a button while moving the mouse at the same time.

4. Steps in Applying GOMS

1. **Define goals** for the task.
2. **List operators** available to the user (keyboard, mouse, gestures).
3. **Identify methods** (different procedures to achieve the goal).

4. **Establish selection rules** (when to choose which method).
5. **Analyze efficiency** (time, effort, learning required).

5. Examples of GOMS in Action

Example 1: Printing a Document

- **Goal**: Print document.
- **Operators**: Move mouse, click, type keys.
- **Methods**:
    - Method 1: Ctrl+P → Enter.
    - Method 2: File → Print → OK.
- **Selection Rule**:
    - Expert uses Method 1.
    - Novice uses Method 2.

👉 Prediction: Method 1 is faster (fewer operators).

Example 2: ATM Cash Withdrawal

- **Goal**: Withdraw cash.
- **Operators**: Insert card, enter PIN, select menu, enter amount.
- **Methods**:
    - Method 1: Quick withdrawal option (default amount).
    - Method 2: Custom withdrawal (enter different amount).
- **Selection Rule**:
    - If default amount = needed → Method 1.
    - Otherwise → Method 2.

👉 Banks often add “Quick Cash” because GOMS analysis shows it reduces operator count.

Example 3: E-commerce Checkout

- **Goal**: Buy an item.
- **Operators**: Click, type address, select payment.
- **Methods**:
    - Method 1: 1-Click checkout.

- Method 2: Add to cart → Proceed to checkout → Enter address → Payment → Confirm.

- **Selection Rule**:
  - If address & card are saved → Method 1.
  - Otherwise → Method 2.

👉 Amazon introduced **1-Click Checkout** after such analysis, reducing time dramatically.

6. Advantages of GOMS

- Helps **predict efficiency** before implementation.
- Compares alternative designs quantitatively.
- Identifies **shortcuts and optimizations**.
- Provides **structured task analysis**.

7. Limitations of GOMS

1. Works best for **routine, well-defined tasks** (not for creative work like writing or design).
2. Assumes **expert users** (ignores learning curve and errors).
3. Focuses on **execution efficiency**, not satisfaction or emotions.
4. Complex tasks → very large GOMS trees, difficult to manage.

8. Real-Time Applications of GOMS

- **ATM design**: Evaluating different withdrawal flows.
- **Websites**: Reducing number of steps for checkout, login, or search.
- **Call center software**: Optimizing operator workflows (fewer clicks → faster customer service).
- **Airline booking systems**: Reducing keystrokes for frequent users.

✅ **In summary:**

- GOMS is a **theory-based cognitive model** to describe user interaction.
- It breaks tasks into **Goals, Operators, Methods, and Selection rules**.
- It helps designers predict **efficiency**, compare alternatives, and improve workflows.

5. Case Studies

1. ATM Machine
    - Problem: Confusing menu order (e.g., asking for amount before selecting account).
    - Solution: Redesign flow → account selection → amount → confirmation.
2. Amazon One-Click Checkout
    - Reduced steps from cart → payment → address → confirm.
    - Increased efficiency → higher sales.
3. Healthcare System
    - Poor UI → doctors spent more time on system than patients.
    - Solution: Task-focused redesign → patient summary on first screen.

6. Shneiderman's Eight Golden Rules

1. Consistency: Uniform words, actions. *(Ctrl+S saves in all apps)*
2. Shortcuts: For experts. *(Ctrl+C, macros in Excel)*
3. Feedback: Inform users. *("Your order is placed" message)*
4. Dialog closure: Clear task completion. *("Logged out successfully")*
5. Error prevention: Avoid mistakes. *("Are you sure you want to delete?")*
6. Reversal of actions: Undo option. *(Ctrl+Z in Photoshop)*
7. User control: Users should feel in charge. *(Back button in browsers)*
8. Reduce memory load: Recognition > recall. *(Dropdown menus, autofill)*

1. Strive for Consistency

- **Theory:**
  Consistency means **similar tasks are performed in similar ways** across an interface. Users shouldn't have to relearn interactions.
  Consistency applies to:
    - Terminology (use same words everywhere)
    - Colors & fonts
    - Icons & layout
    - Interaction behavior
- **Why important:**
  Inconsistent design → confuses users, increases learning curve, leads to errors.

- **Real-life Example:**
    - Microsoft Office: Same ribbon layout across Word, Excel, PowerPoint.
    - Bad Example: Government websites often mix different button styles ("Submit," "Proceed," "Next") for the same function.

2. Enable Frequent Users to Use Shortcuts

- **Theory:**
  Frequent users should be able to speed up interactions using **accelerators** like shortcuts, macros, function keys, or gestures.
  Beginners rely on menus; experts need shortcuts.
- **Why important:**
  Boosts **efficiency** and keeps experts engaged.
- **Real-life Example:**
    - Ctrl+C (copy), Ctrl+V (paste), Ctrl+S (save).
    - Adobe Photoshop → keyboard shortcuts for professional designers.
    - Bad Example: Apps that force users to always navigate long menus without offering faster alternatives.

3. Offer Informative Feedback

- **Theory:**
  Every user action should produce **immediate and clear feedback**.
  Feedback should be proportional to the action:
    - Small actions → small feedback (button highlight).
    - Big actions → big feedback (confirmation message, progress bar).
- **Why important:**
  Prevents uncertainty, builds trust, helps error recovery.
- **Real-life Example:**
    - WhatsApp → Blue tick marks for delivered/read messages.
    - File upload → Progress bar showing % completed.
    - Bad Example: Clicking “Pay” on a slow website with no feedback, making users click multiple times.

4. Design Dialogs to Yield Closure

- **Theory:**
  Group interactions into **beginning, middle, and end** with clear closure at the end.
  Closure gives users a sense of accomplishment.
- **Why important:**
  Without closure, users feel unsure if a task was completed.
- **Real-life Example:**
  - Amazon → "Your order has been placed" confirmation screen.
  - ATM → "Please take your card and cash. Thank you!"
  - Bad Example: Submitting an online form and returning to a blank page with no confirmation.

5. Offer Error Prevention and Simple Error Handling

- **Theory:**
  Systems should be designed to **prevent errors** and help users **recover easily** if they occur.
  - Error prevention: Restrict invalid input.
  - Error handling: Provide meaningful error messages.
- **Why important:**
  Reduces frustration, avoids critical failures.
- **Real-life Example:**
  - Gmail → Warning: "Did you mean to attach a file?" when you write "Please find attached" but no attachment.
  - Google search → "Did you mean…?" suggestions.
  - Bad Example: "Error 404" with no explanation of what went wrong.

6. Permit Easy Reversal of Actions

- **Theory:**
  Users should be able to **undo or redo** actions easily.
  This encourages exploration and reduces anxiety.
- **Why important:**
  Users feel safe knowing mistakes can be fixed.
- **Real-life Example:**
  - Ctrl+Z (undo) in Word, Photoshop, Excel.
  - Shopping carts → Items can be removed easily before checkout.
  - Bad Example: Accidentally deleting a file with **no undo option**.

7. Support Internal Locus of Control

- **Theory:**
  Users should feel **in control of the system**, not the other way around.
  - Users initiate actions.
  - System should respond to user commands, not surprise them.
- **Why important:**
  Builds trust, confidence, and reduces frustration.
- **Real-life Example:**
  - Google Maps: Users choose route options (avoid tolls, highways).
  - Bad Example: Auto-playing ads or forced software updates without user permission.

8. Reduce Short-Term Memory Load

- **Theory:**
  Human short-term memory is limited (≈ 7 ± 2 items, according to Miller's Law).
  Interfaces should avoid requiring users to **remember information between screens**.
- **Why important:**
  Reduces errors, makes systems easier to use, especially for novices or elderly.
- **Real-life Example:**
  - Dropdown menus (show options instead of remembering them).
  - Auto-complete search suggestions in Google.
  - Bad Example: Filling out long forms where users must remember codes or IDs from earlier steps.

Summary Table: Shneiderman's Eight Golden Rules

Rule	Key Idea	Example
1. Consistency	Keep design uniform	MS Office ribbon
2. Shortcuts	Accelerators for experts	Ctrl+S
3. Feedback	System must respond	WhatsApp ticks
4. Closure	Clear start and end	Amazon order confirmation
5. Error handling	Prevent & recover	Gmail attachment warning
6. Reversal	Allow undo/redo	Ctrl+Z
7. Locus of control	User feels in control	Google Maps route choice
8. Reduce memory load	Don't rely on recall	Dropdown menus, auto-complete

7. Norman's Seven Principles

1. Use natural mapping: Controls resemble results. *(Steering wheel turns car)*
2. Provide feedback: System shows results. *(Mic button glows while recording)*
3. Make things visible: Show possible actions. *(Clickable buttons highlighted)*
4. Constraints: Prevent errors. *(Greyed-out options in menu)*
5. Consistency: Predictable design. *(iOS/Android back button in same place)*
6. Affordances: Design hints usage. *(Door knob = turn, push plate = push)*
7. Conceptual model: Users understand system. *(Shopping cart = checkout metaphor)*

1. Use Natural Mapping

- **Theory:**
  Mapping is the relationship between **controls** and their **effects in the real world**.
  - *Good mapping*: The control directly suggests what it does.
  - *Poor mapping*: Control is confusing and does not indicate its effect.
- **Why important:**
  Good mapping reduces cognitive effort, increases predictability.
- **Real-life Examples:**
  ✅ Stove burners arranged in the same layout as knobs → users instantly know which knob controls which burner.
  ✅ Car steering wheel: Turning left = car turns left.
  ❌ Complex TV remote with dozens of unrelated buttons.

2. Provide Feedback

- **Theory:**
  Users need to know **what happened after they take an action**.
  Feedback must be **immediate, clear, and proportional** to the action.
- **Why important:**
  Without feedback, users feel uncertain and may repeat or abandon actions.
- **Real-life Examples:**
  ✅ WhatsApp → Message sent (one tick), delivered (two ticks), read (blue ticks).
  ✅ Progress bar during file download.
  ❌ Clicking "Pay Now" on a slow website without response → user clicks multiple times, double-paying.

3. Make Things Visible

- **Theory:**
  All possible actions should be **visible or discoverable**.
  Users shouldn't have to guess or remember functions.
- **Why important:**
  Invisibility of options leads to confusion and errors.
- **Real-life Examples:**
  ✅ Elevator: Floor numbers are clearly visible with lighted indicators.
  ✅ Microsoft Word toolbar → icons (bold, italic, underline) are always visible.
  ❌ Hidden options behind hamburger menus that users never find.

4. Exploit Constraints

- **Theory:**
  Constraints **limit possible actions**, preventing errors.
  Types of constraints:
  - **Physical** (shape, size prevents incorrect action)
  - **Cultural** (socially accepted norms)
  - **Logical** (sequence dictates correct action)
- **Why important:**
  Constraints guide users naturally toward correct actions.
- **Real-life Examples:**
  ✅ USB-C plugs → designed to fit only one way (physical constraint).
  ✅ Traffic signals (red = stop, green = go) → cultural constraint.
  ✅ Puzzle apps → logical constraints force step-by-step solving.
  ❌ Early USB (Type-A) often inserted incorrectly, frustrating users.

5. Maintain Consistency

- **Theory:**
  Consistency means **uniform design across functions, screens, or systems**.
  - Internal consistency → within the system.
  - External consistency → with other systems.
- **Why important:**
  Familiarity lowers learning effort and reduces mistakes.
- **Real-life Examples:**
  ✅ Ctrl+C always copies across apps (Word, Excel, browser).
  ✅ Icons: Trash bin = delete everywhere.
  ❌ Different apps using different gestures (swipe left = delete in Gmail but archive in Messages).

6. Use Affordances

- **Theory:**
  Affordance = the design **suggests how an object should be used**.
  - Perceived affordances = what users *think* they can do.
  - Actual affordances = what can actually be done.
- **Why important:**
  Intuitive affordances reduce the need for instructions.
- **Real-life Examples:**
  ✅ Door handle → affords pulling, flat push plate → affords pushing.
  ✅ Buttons in apps look raised → afford clicking.
  ❌ Glass doors with handles but labeled "PUSH" → misleading affordance.

7. Develop a Conceptual Model

- **Theory:**
  A conceptual model is the **mental model** that users form about how a system works.
  Good design supports accurate mental models.
- **Why important:**
  If the conceptual model matches system behavior → users feel in control.
  If not → frustration and errors.
- **Real-life Examples:**
  ✅ Shopping cart in e-commerce → matches physical cart (users know they can add/remove items).
  ✅ Desktop metaphor in computers (files, folders, trash bin).
  ❌ Microwave ovens with complex multi-function buttons → no clear conceptual model.

Summary Table: Norman's Seven Principles

Principle	Key Idea	Good Example	Bad Example
Natural Mapping	Controls resemble results	Steering wheel	Complex stove with misaligned knobs
Feedback	Show results of action	WhatsApp ticks	"Pay" button with no response
Visibility	Show available options	Word toolbar icons	Hidden options in hamburger menu
Constraints	Limit possible errors	USB-C plugs	Old USB (Type-A flips)
Consistency	Uniform design	Ctrl+C works everywhere	Swipe left = archive vs delete
Affordances	Design suggests use	Push plate vs handle	Glass door with pull handle + PUSH label
Conceptual Model	User's mental model	Shopping cart in apps	Confusing microwave interface

8. Norman's Model of Interaction

Two stages:

1. Execution: User forms goal → plans action → executes.
2. Evaluation: System shows feedback → user interprets → compares with goal.

Example: Sending a WhatsApp message

- Goal: Send "Hi" to friend.
- Execution: Open app → type "Hi" → press send.
- Evaluation: Message appears with tick marks → user confirms success.

1. Introduction

- Proposed by **Don Norman** in *The Design of Everyday Things*.
- It describes the **cognitive steps users go through when interacting with technology**.
- Emphasizes that interaction is a **two-way process**:
    - **Execution**: User → System (user acts to achieve a goal).
    - **Evaluation**: System → User (system feedback interpreted by user).

👉 Problems arise when there is a mismatch (called **gulfs**) between what the user wants and what the system shows.

2. Stages of Action in Norman's Model

Norman breaks interaction into **7 stages**:

Execution Phase (User to System)

1. **Forming the Goal**
    - User decides what they want to achieve.
    - *Example:* A student wants to **submit an online assignment**.
2. **Forming the Intention**
    - User plans how to achieve the goal.
    - *Example:* The student decides to **log in to the university portal**.
3. **Specifying the Action Sequence**
    - User identifies the exact steps.
    - *Example:* Open browser → go to portal → enter username & password → upload file → click submit.
4. **Executing the Action**
    - User performs the actions.
    - *Example:* Student types credentials, uploads file, clicks submit.

Evaluation Phase (System to User)

5. **Perceiving the System State**
    - User observes system's response.
    - *Example:* System displays "File uploaded successfully."
6. **Interpreting the System State**
    - User interprets meaning of system feedback.
    - *Example:* Student understands that the assignment was accepted.
7. **Evaluating the Outcome**
    - User compares result with original goal.
    - *Example:* Goal was submission → system confirms → student satisfied.

3. The Two Gulfs

- **Gulf of Execution**
    - Gap between *user's intention* and *system's available actions*.
    - *Example:* A new user wants to format text but doesn't know which button in MS Word does it.
- **Gulf of Evaluation**
    - Gap between *system's output* and *user's ability to interpret it*.
    - *Example:* Error message "0x0000F1" appears → user cannot understand what went wrong.

👉 Good design minimizes these gulfs by making actions **discoverable** and system states **interpretable**.

4. Real-Life Examples

Example 1: ATM Cash Withdrawal

- **Goal**: Withdraw ₹1000.
- **Intention**: Use ATM.
- **Action Sequence**: Insert card → enter PIN → choose withdrawal → enter ₹1000 → confirm.
- **Execution**: User performs steps.
- **System Feedback**: "Transaction Successful. Please collect cash."
- **Evaluation**: User gets cash → Goal achieved.

👉 If the ATM shows an unclear error like "Transaction Denied: Error 45" → **gulf of evaluation**.

Example 2: Online Food Delivery (Zomato/Swiggy)

- **Goal**: Order pizza.
- **Intention**: Use Swiggy app.
- **Action Sequence**: Open app → search pizza → add to cart → pay.
- **Execution**: User taps through steps.
- **System Feedback**: "Order placed. Expected delivery in 30 minutes."
- **Evaluation**: User confirms order matches goal.

👉 If the app crashes during checkout → **gulf of execution** (user can't complete intention).

Example 3: Smart Home Device (Alexa)

- **Goal**: Turn off lights.
- **Intention**: Give Alexa a voice command.
- **Execution**: Say "Alexa, turn off lights."
- **System Feedback**: Alexa responds "Okay" + lights go off.
- **Evaluation**: User sees dark room → goal achieved.

👉 If Alexa says "I didn't understand the command" → **gulf of execution** (user's command not mapped correctly).

5. Strengths of Norman's Model

- Provides a **structured way** to understand interaction.
- Helps designers identify **where users struggle** (execution vs evaluation).
- Useful in **usability testing**: map user failures to a stage.

6. Limitations

- Assumes users are **rational and goal-driven** (ignores emotions, multitasking, distractions).
- Works best for **task-oriented systems**, less for open-ended tasks (like browsing YouTube).
- Sometimes hard to map real-life messy interactions into neat 7 stages.

7. Design Implications

Designers should:

- Minimize the **gulf of execution** → make controls clear, intuitive, and discoverable.
- Minimize the **gulf of evaluation** → give meaningful, human-readable feedback.

- Support users in **both phases** of interaction.

Examples:

- Use icons + text labels (better discoverability).
- Use friendly error messages ("Password too short. Must be at least 8 characters.").
- Provide undo/cancel options to keep users in control.

9. Nielsen's Ten Heuristics

1. Visibility of system status → *Progress bar in file download*.
2. Match with real world → *Trash bin = delete*.
3. User control & freedom → *Undo, back button*.
4. Consistency & standards → *Same icons across apps*.
5. Error prevention → *Confirmation before deleting files*.
6. Recognition over recall → *Recent files list in MS Word*.
7. Flexibility & efficiency → *Shortcuts for experts*.
8. Aesthetic & minimalist design → *Google homepage (just a search box)*.
9. Error recovery → *Spell check suggestions in MS Word*.
10. Help & documentation → *"Need help?" links in software*.

1. Visibility of System Status

- **Theory:**
  The system should **always keep users informed** about what is happening, through appropriate feedback within reasonable time.
- **Why important:**
  Without visibility, users feel lost or think the system has failed.
- **Real-time Examples:**
  ✅ Progress bar while downloading a file.
  ✅ WhatsApp → single tick (sent), double tick (delivered), blue ticks (read).
  ❌ Clicking "Submit" and nothing happens → user re-clicks multiple times.

2. Match Between System and the Real World

- **Theory:**
  The system should **speak the users' language**, using concepts, icons, and metaphors familiar to users (not system jargon).

- **Why important:**
  Users learn faster when design follows **real-world conventions**.
- **Real-time Examples:**
  ✅ Trash bin icon for deleting files.
  ✅ E-commerce sites → "Shopping cart" metaphor.
  ❌ "Error 0x800CCC0E" → incomprehensible to normal users.

3. User Control and Freedom

- **Theory:**
  Users should feel in control. They need **emergency exits** (undo, cancel, back) to leave unwanted states without frustration.
- **Why important:**
  Prevents users from feeling trapped.
- **Real-time Examples:**
  ✅ "Undo Send" in Gmail.
  ✅ Back button in web browsers.
  ❌ Online forms that reset everything if you press "Back."

4. Consistency and Standards

- **Theory:**
  Users should not have to wonder whether different words, icons, or actions mean the same thing. **Follow platform and industry standards**.
- **Why important:**
  Familiarity builds trust and reduces learning effort.
- **Real-time Examples:**
  ✅ Ctrl+S saves in all applications.
  ✅ Hyperlinks are always blue and underlined.
  ❌ Inconsistent terminology: "Cart" in one place, "Bag" in another.

5. Error Prevention

- **Theory:**
  Even better than good error messages is a **careful design that prevents problems** from occurring.
- **Why important:**
  Saves users time and frustration.
- **Real-time Examples:**
  ✅ Gmail → warns if you write "Please find attached" but no attachment is added.

✅ Online forms → disable "Submit" until all required fields are filled.
❌ Airline ticket form that allows invalid dates (e.g., return date before departure).

6. Recognition Rather Than Recall

- **Theory:**
  Reduce memory load by making objects, actions, and options **visible**. Users should not have to remember information between screens.
- **Why important:**
  Recognition is easier than recall, especially for novice users.
- **Real-time Examples:**
  ✅ Recently searched items shown in Google search bar.
  ✅ Dropdown menus (show available options).
  ❌ Requiring users to remember product codes and enter them manually.

7. Flexibility and Efficiency of Use

- **Theory:**
  The interface should cater to both **novices and experts**. Provide accelerators like shortcuts, macros, and customizations.
- **Why important:**
  Novices learn step by step, experts work faster.
- **Real-time Examples:**
  ✅ MS Word → menus for beginners, keyboard shortcuts for experts.
  ✅ Photoshop → allows custom macros and hotkeys.
  ❌ Apps that force everyone through long menus with no faster option.

8. Aesthetic and Minimalist Design

- **Theory:**
  Dialogues should contain **no irrelevant information**. Every extra element competes with relevant ones and diminishes visibility.
- **Why important:**
  Clean design = better focus, less confusion.
- **Real-time Examples:**
  ✅ Google homepage → simple search box.
  ✅ Apple product pages → clean, image-focused.
  ❌ Government websites cluttered with ads, flashing text, and too many links.

9. Help Users Recognize, Diagnose, and Recover from Errors

- **Theory:**
  Error messages should be expressed in **plain language**, precisely indicate the problem, and constructively suggest a solution.
- **Why important:**
  Helps users fix errors instead of panicking.
- **Real-time Examples:**
  ✅ "Your password must contain at least 8 characters including a number and a special character."
  ✅ Browser → "This site can't be reached. Check internet connection."
  ❌ "Error 503" with no explanation.

10. Help and Documentation

- **Theory:**
  Even though the system should be usable without documentation, **help and support must be available**.
- **Why important:**
  Users may need assistance when stuck.
- **Real-time Examples:**
  ✅ Microsoft Office → "Tell me what you want to do" search box.
  ✅ Chatbots or FAQs in banking apps.
  ❌ Complex enterprise apps with no accessible help or tutorials.

Summary Table: Nielsen's Ten Heuristics

Heuristic	Key Idea	Good Example	Bad Example
1. Visibility	System status visible	Progress bar	Silent submit button
2. Real-world match	Use familiar language	Trash bin icon	"Error 0x800CCC0E"
3. Control & freedom	Exit, undo, cancel	Back button	Forms with no back option
4. Consistency	Follow standards	Ctrl+S everywhere	Cart vs Bag inconsistency
5. Error prevention	Avoid mistakes	Gmail attach warning	Invalid date fields allowed
6. Recognition > recall	Visible options	Dropdown menus	Enter product codes manually
7. Flexibility	Novices & experts	Word menus + shortcuts	No accelerators
8. Minimalist	Clean, focused design	Google homepage	Cluttered gov sites
9. Error recovery	Clear error messages	Password guidelines	"Error 503"
10. Help & docs	Provide assistance	MS Office help search	No support in complex apps