User Interface Design Principles Evaluation for the TAN(x) Calculator GUI

Jayanth Apagundi(40291184) | SOEN-6011:Eternity Version:1.0.0

The TAN(x) Calculator GUI was evaluated against seven established user interface design principles. For each principle, we provide a **Statement**, a **Description/Discussion**, and concrete **Examples** drawn from the current application.

1 The Principle of User Profiling

Statement. The interface should reflect the target users' knowledge, experience, and goals.
Description/Discussion. The application targets learners and practitioners who need a quick $\tan(x)$ value without advanced configuration. The UI favors clarity over configurability: plain-language prompts, minimal steps, and familiar control types. No domain jargon is required (e.g., users need not manually convert degrees to radians).
Examples.

- Prompts such as "Enter angle (x)" and "Select Unit" are concise and novice-friendly.
- The unit combo box (Degrees/Radians) avoids forcing users to know conversion formulas.
- Error messages use plain language (e.g., "Invalid input! Please enter a valid number.").

2 The Principle of Humility

Statement. The interface should stay out of the way and help users accomplish their task.
Description/Discussion. The GUI is intentionally minimal: one input, one unit selector, one action button, and a results label. Visual noise (animations, banners, branding) is avoided. Guidance appears as unobtrusive prompt text, tooltips, and accessible text.
Examples.

- A compact single-view layout with only essential components.
- Context help via tooltips; no pop-ups or modal interruptions.
- The window title and version label inform without distracting.

3 The Principle of Metaphor

Statement. Use familiar concepts so users can transfer real-world knowledge.
Description/Discussion. The calculator metaphor is preserved: enter a value, choose units, press "Calculate," read the result. Standard controls (text field, combo box, button, label) behave predictably and mirror common calculator workflows.
Examples.

- Unit selection mimics a physical calculator's degree/radian toggle.
- The "Calculate tan(x)" button mirrors pressing an operation key.
- Labels and prompts map directly to what appears on calculators and worksheets.

4 The Principle of Feature Exposure

Statement. Core features should be visible and accessible without hunting.
Description/Discussion. All primary functionality is on one screen: input, unit selection, action, and result. There are no hidden menus, tabs, or advanced dialogs for the core task. Users can complete the workflow at a glance.
Examples.

- The angle input, unit dropdown, and calculate button are visible on launch.
- Results appear immediately in a dedicated label beneath the controls.
- No nested navigation is required to compute $\tan(x)$.

5 The Principle of State Visualization

Statement. Make system state and outcomes visible at all times.
Description/Discussion. Prompt text communicates the initial state (awaiting input); the result label reflects outcomes: success, missing unit, invalid input, or mathematically undefined cases. This reduces uncertainty and supports quick correction.
Examples.

- If no unit is selected, the message "Please select a unit (Degrees or Radians)." appears.
- Non-numeric input triggers "Invalid input! Please enter a valid number."
- When $\cos(x) \approx 0$, an explicit error explains that $\tan(x)$ is undefined at that angle.

6 The Principle of Coherence

Statement. Maintain consistency in terminology, layout, and behavior.
Description/Discussion. The interface uses consistent spacing, capitalization, and tone. Field prompts, button text, and labels align semantically. Error handling is centralized in the result label to provide a single place to read system feedback.
Examples.

- Uniform vertical spacing (`VBox(10)`) and padding (`Insets(20)`).
- Consistent wording across prompt texts, tooltips, and messages.
- A single result label communicates all outcomes, avoiding fragmented feedback.

7 The Principle of Safety

Statement. Prevent errors where possible and support safe recovery when they occur.
Description/Discussion. Input validation and exception handling prevent crashes and undefined behavior. Users can correct inputs without losing context. Mathematically undefined cases are intercepted and explained.
Examples.

- `NumberFormatException` caught for non-numeric input; the app stays responsive.
- `ArithmeticException` caught when $\cos(x)$ is near zero; a clear message is shown.
- Users can re-enter a value, change units, and press "Calculate" again without restarting.