How to Design a Great User Experience

While simply expressed, each of these ideas is profound. We could make each an article, but we'll give a short explanation instead. Fill in any missing details with examples from your own experience.

1. **Nail the basics**
   The core **scenarios**—the primary reasons people use your Windows® program—are far more important than the fringe scenarios—things people might do but probably won't. Nail the basics! (And if you do, users will overlook fringe problems.)
2. **Design experiences, not features**
   Design experiences from beginning to end, not just individual features. And maintain your standards throughout the entire product experience. For example, if your program's setup is hard to use and is buggy, users will assume your program is hard to use and buggy too. Why should they assume otherwise?
3. **Be great at something**
   Think about how *real* users (not the marketing or PR departments) will describe your program. Identify your target users and make sure they can say "I love this program! It does A, B, and C super well!" If users can't say that about your program, what's the point? Today, "good enough" is no longer good enough—make your users love it.
4. **Don't be all things to all people**
   Your program is going to be more successful by delighting its target users than attempting to satisfy everyone. Remember that it is literally impossible to focus on everything.
5. **Make the hard decisions**
   Do you really need that feature, command, or option? If so, do it well. If not, cut it! Don't avoid difficult decisions by making everything optional or configurable.
6. **Make the experience like a friendly conversation**
   Think of your UI as a conversation between you and your target users. Suppose you're looking over a user's shoulder and he or she asks, "What do I do here?" Think about the explanation you would give...the steps, their order, the language you'd use, and the way you explain things. Also think about what you *wouldn't* say. That's what your UI should be—like a conversation between friends—rather than something arcane that users have to decipher.
7. **Do the right thing by default**
   Sure, you can **pile on options** to allow users to change things, but why? Choose safe, secure, convenient default values. Also, make the default experience the right experience for your target users. Don't assume that they will configure their way out of a bad initial experience. They won't.
8. **Make it just work**
   People want to use your program, not configure it or learn a bunch of things. Choose an initial configuration, make it obvious how to do the most common and important tasks, and get your program working right away.
9. **Ask questions carefully**
   Avoid asking unessential questions using **modal dialogs**—prefer modeless alternatives. Better yet, the current context can reveal the user's intent, often eliminating the need to ask at all. If you must ask a question in your UI, express it in terms of users' **goals and tasks, not in terms of technology**. Provide options that users understand (again, phrased in terms of goals and tasks, not technology) and clearly differentiate. Make sure to provide enough information for users to make informed decisions.
10. **Make it a pleasure to use**
    Make sure your program serves its purpose well. Have the right set of features and put the features in the right places. Pay attention to detail, and make sure everything is polished. Don't assume that users won't notice small things. They will.
11. **Make it a pleasure to see**

Use the standard Windows look, including standard **window frames**, **fonts**, **system colors**, **common controls** and **dialog boxes**, and standard **layout**. Avoid custom UI and use **branding** with restraint. Use standard Windows **icons**, **graphics**, and **animations** whenever possible (and legal!) For your own graphics and icons, use a professional designer. (If you can't afford one, use a few simple graphics—or even none at all.) And don't assume that providing skins will compensate for an unexciting look. Most users won't bother with them and having one great look makes a much better impression than having dozens of not-so-great ones.

12. **Make it responsive**
    Your program's responsiveness is crucial to its overall experience—users find unnecessarily slow and unresponsive programs unusable. For every feature where performance is an issue, first understand your users' goals and expectations, then choose the lightest weight design that achieves these goals. Generally, tasks that can take longer than 10 seconds need more informative feedback and the ability to cancel. Keep in mind that users' perception of speed is just as important as the actual speed, and the perception of speed is primarily determined by how quickly a program becomes responsive.

13. **Keep it simple**
    Strive for the **simplest design** that does the job well. Expand the design beyond that only as *required*. Don't have three ways to do something when one will do. Eliminate or reduce all that unnecessary junk!

14. **Avoid bad experiences**
    Easier said than done, but users' overall perception of your program is more often determined by the quality of the bad experiences than of the good ones.

15. **Design for common problems**
    Is your design great—until the user makes a mistake or the network connection is lost? Anticipate and design for common problems, user mistakes, and other errors. Consider things like the network being slow or unavailable, devices being not installed or unavailable, and users giving incorrect input or skipping steps. At each step in your program, ask yourself: What are the worst likely things that could happen? Then see how well your program behaves when they do happen. Make sure all **error messages** clearly explain the problem and give an actionable solution.

16. **Don't be annoying**
    Most likely, anything users routinely dismiss without performing any action should be redesigned or removed. This is especially true for anything users see repeatedly, such as error messages, **warnings**, **confirmations**, and **notifications**. Never interrupt something users care about with something they don't care about. Never cover something beautiful with something ugly. Use **sound** with extreme restraint. UI related to security and legal issues (for example, consent or license terms) are possible exceptions.

17. **Reduce effort, knowledge, and thought**
    To reduce the effort, knowledge, and thought required to use your program:
    - **Explicit is better than implicit.** Put the information users need to know directly on the screen. Carefully craft the **main instruction** on windows and pages to clearly communicate the purpose of the UI.
    - **Automatic is better than manual.** Try to help users by doing things automatically whenever practical and desirable. A simple test: Close your program, then restart it and perform the most common task. How much manual effort can you eliminate?
    - **Concise is better than verbose.** Put it on the screen, but concisely. Get right to the point! Design text for scanning, not immersive reading. Use **Help links** for helpful, supplemental, but not essential information.
    - **Constrained is better than unconstrained.** When choosing controls, the control constrained to valid input is usually the best choice.
    - **Enabled is better than disabled.** **Disabled controls** are often confusing, so use them only when users can easily deduce why the control is disabled. Otherwise, remove the control if it doesn't apply or leave it enabled and give helpful feedback.
    - **Remembered is better than forgotten.** Except for situations that involve security and privacy, it's

better to remember users' previous input and actions and make them easy to do again than to make users start over each time.

- **Feedback is better than being clueless.** Give clear feedback to indicate whether a task is being done or has failed. Don't make the user guess.

18. **Follow the guidelines**
    Of course! Consider UX Guide to be the minimum quality and consistency bar for Windows-based programs. Use it to follow best practices, make routine decisions, and to just make your job easier. Focus your creative energy on the important things—whatever your program is all about—not the routine. Don't create that weird program that nobody can figure out how to use. Follow the guidelines and make your experience stand out while fitting in.
19. **Test your UI**
    You won't know if you've got it right until you've tested your program with real target users with a **usability study**. Most likely, you'll be (unpleasantly) surprised by the results. Be glad to have your UI criticized—that's required for you to do your best work. And be sure to collect feedback after your program ships.