About Face (3rd Edition)

Summary by Matt Ownby
11 April 2011

Disclaimer

- I use examples from our own product of what not to do
- But I am as guilty as any one else
- So don't feel picked on :o

Locations cited in presentation

- The locations cited in this presentation correspond to the Kindle edition of “About Face 3rd Edition”

Recognizing User Goals

- We often think of our stories as tasks/activities. We can lose sight of the overall goal when we do this.
- Goals change very slowly over time. Tasks/activities may change rapidly.
- Goal: travel to destination, Task: get on airplane, drive car, walk (task depends on available technology)
- See location 521 for another example of goal vs task.
- Goals, not features, are the key to a product success (location 630)

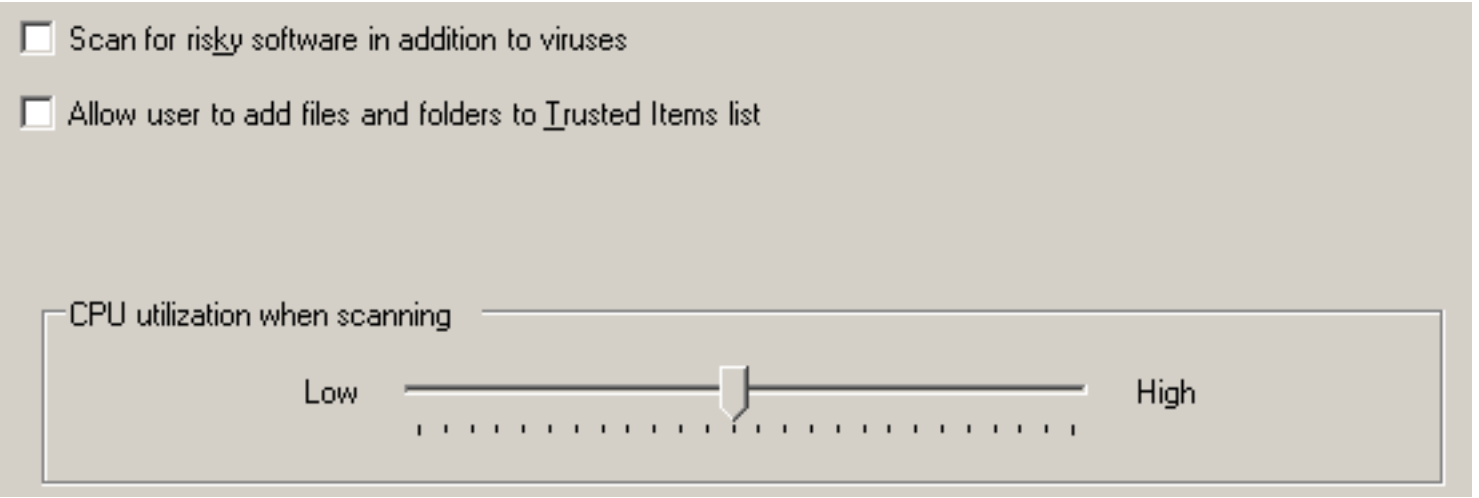
Implementation Model vs Mental Model

- Implementation model means designing the user interface to conform to the way the software is implemented “under the hood”
- Mental model is how a user imagines or visualizes how a product works
- See location 655 first paragraph for description
- **Implementation model is very common in software development**
- Implementation model wastes user's time!

Implementation Model

- Implementation model places undue burden on user by forcing them to understand how a product works under the hood.
- Next slide shows some examples from our own stuff (not to pick on anyone hehehe)

Implementation Model



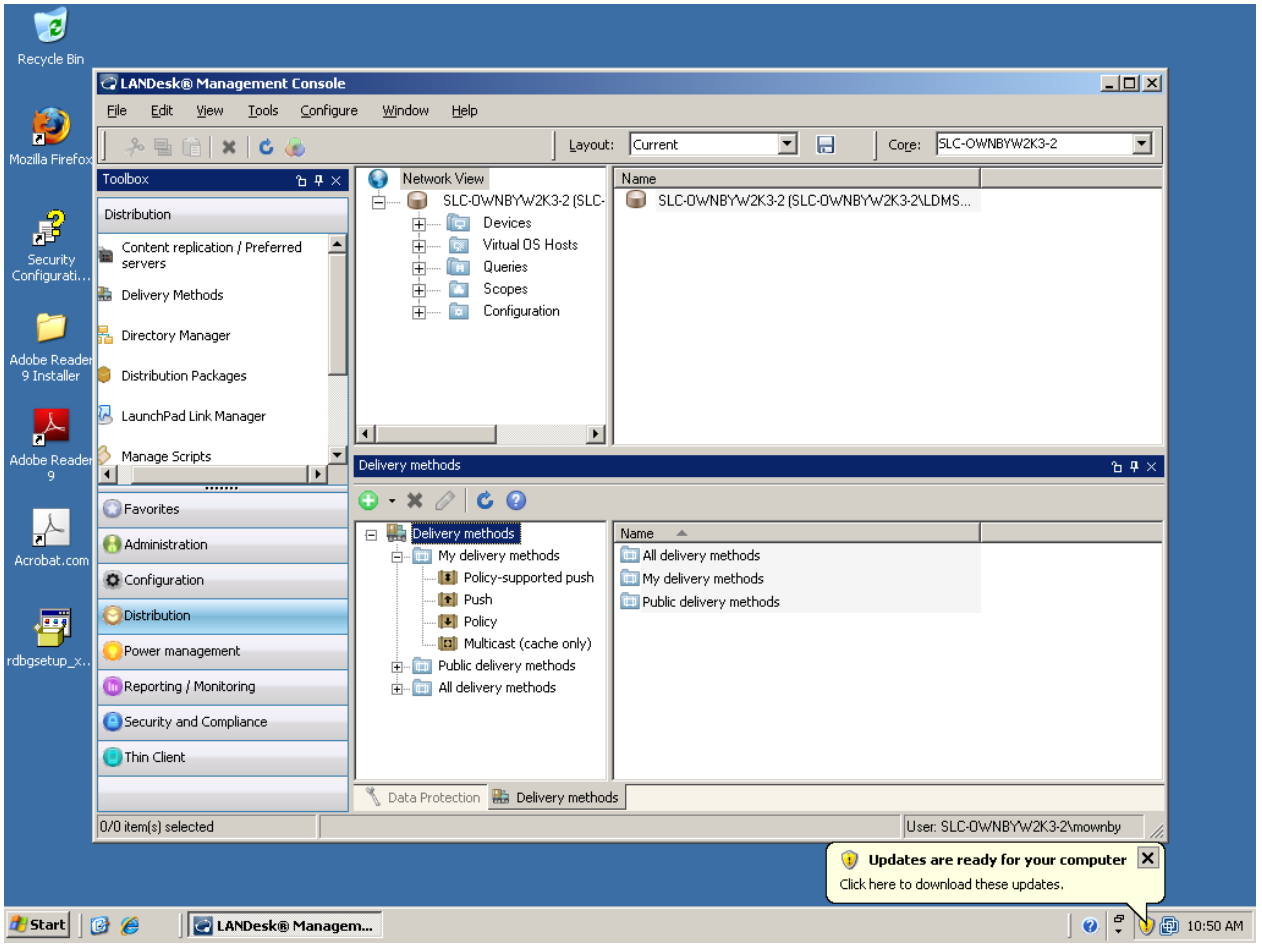
- User must understand the difference between risky software and viruses in order to make a decision here.
- User must understand how the CPU utilization setting works in order to make a decision.
- Neither are user goals.
- What is the user's goal?

Implementation Model

- In order to understand whether they want this option, the user must first understand what it means and how it works.
- This is not a user goal.
- What is the user's goal?

Implementation model continued

- Delivery methods is an example of the implementation model. Not a user goal to understand this stuff. What is the user's goal?



Mental Model

- Mental model is how the user **imagines** that something works.
- Example: electronic device uses Alternating Current but we **imagine** that power just flows like water through the cord.
- Example: when we go to a movie, we don't understand how the projector works. We just **imagine** that it throws the picture on the screen. Same with when we watch TV.

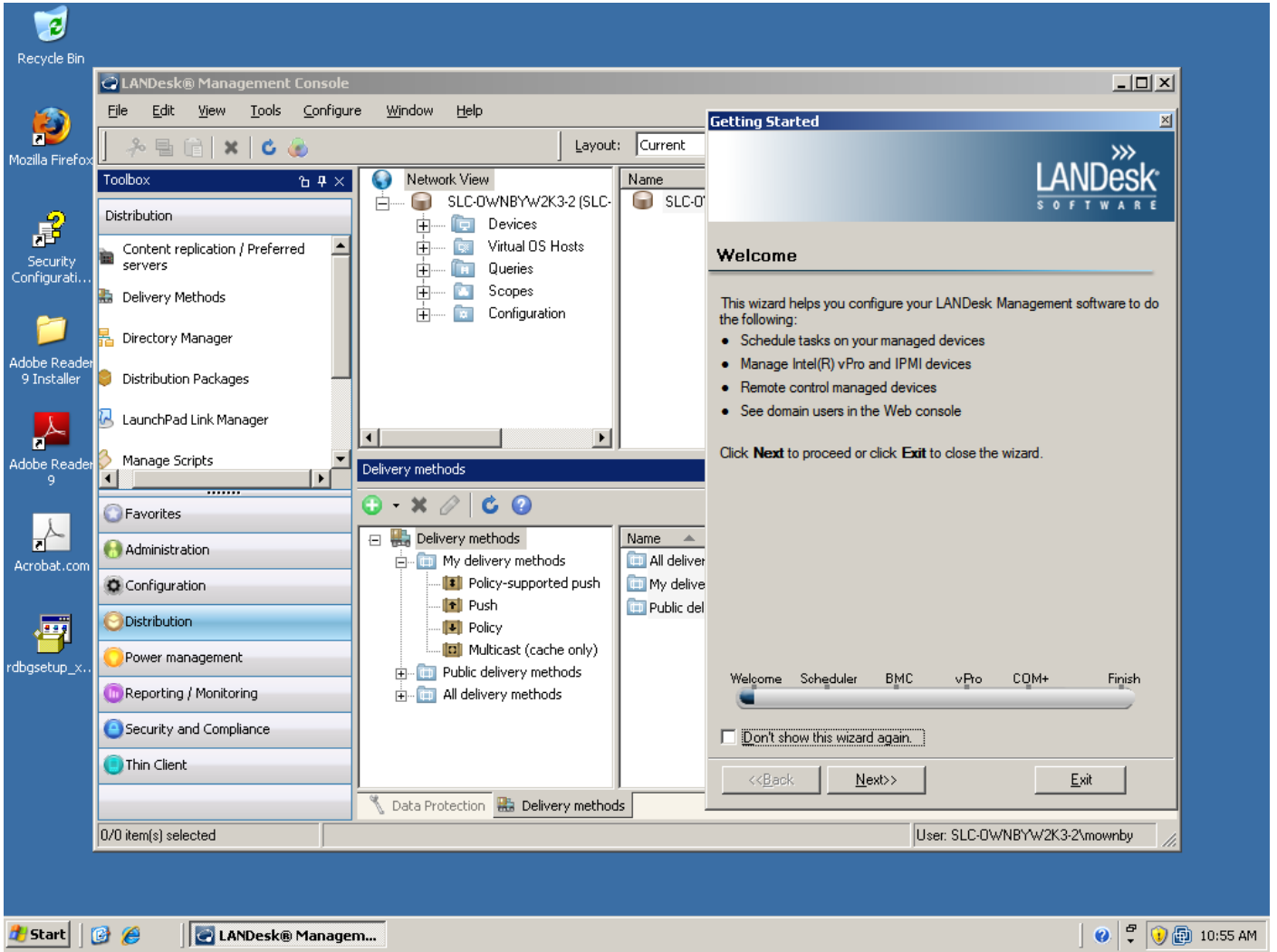
Follow the mental model

- Most software follows the implementation model
- Our interfaces **should follow the user's mental model**, not our implementation model.
- See location 696 in book

Beginners, Experts, and Intermediates

- Most users are intermediates
- Beginners quickly become intermediates
- Software should be **optimized for the "perpetual" intermediate user**
- Experts include users who want shortcut keys and enjoy knowing how things work "under the hood". **Unless people use the software very often, they will not be experts.**
- LDMS tends to be optimized toward experts, not intermediates. (see screenshot on next slide)

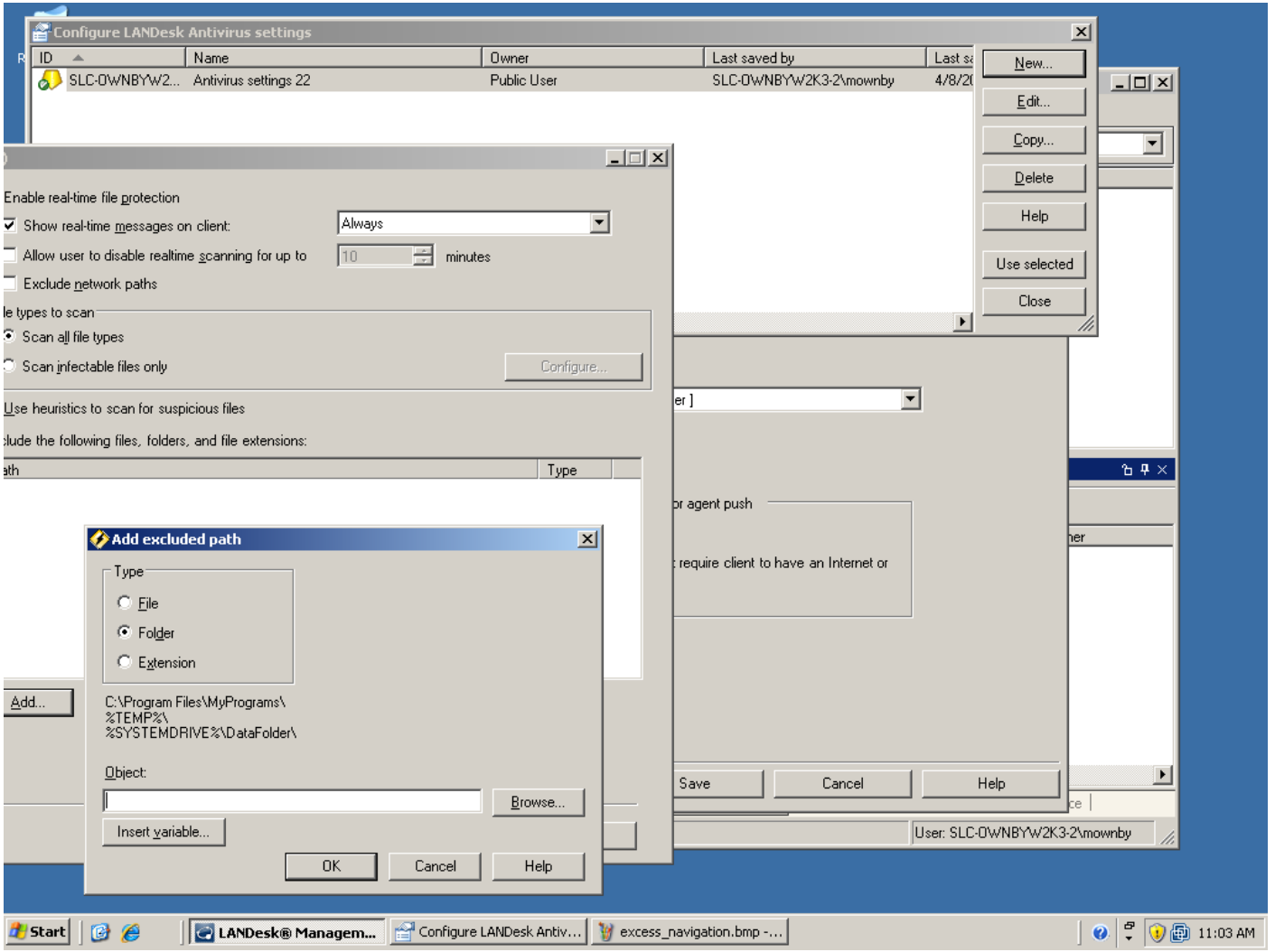
Because our software is optimized for experts, we bolt on “training wheels” to help beginners. The intermediate user is forgotten.



Reducing excise (waste)

- Navigation is excise (location 3004)
- Reduce the number of places to go (location 3064)
- See example on next slide

A random example of navigational excise. Nested dialogs.



Designing Good Behavior

- **Imagine that your product is an extremely considerate human being**
- Design its behavior accordingly
- Anticipate human needs
- Don't ask a lot of questions
- Fail gracefully
- Take responsibility (don't blame the user)
- Know when to bend the rules

Designing smart products

- Put idle CPU cycles to work
- Most people pause while typing to think or do something else. These idle cycles can be used to make their life easier. *For example, in my data protection configuration screen, I use idle cycles to test whether the settings the user has already typed in are correct.*
- Remember settings

Metaphors, Idioms

- Avoid designing your product to conform to a real-world metaphor. *Example is using a phone icon to indicate networking, like Windows 95 did.* Real-world metaphors often do not represent the ideal way to get work done, just the current best way. They become obsolete.
- Idioms don't conform to any real world model but are easily learned and incredibly effective. Example is using a mouse (clicking, dragging, etc).

UNDO

User mental models of mistakes

Users generally don't believe, or at least don't want to believe, that they make mistakes. This is another way of saying that the persona's mental model typically doesn't include error on his part. Following a persona's mental model means absolving him of blame. The implementation model, however, is based on an error-free CPU. Following the implementation model means proposing that all culpability must rest with the user. Thus, most software assumes that it is blameless, and any problems are purely the fault of the user.

The solution is for the user-interface designer to completely abandon the idea that the user can make a mistake — meaning that everything the user does is something he or she considers to be valid and reasonable. Most people don't like to admit to mistakes in their own minds, so the program shouldn't contradict this mindset in its interactions with users.

Files and Save

Every running application exists in two places at once: in memory and on disk. The same is true of every open file. For the time being, this is a necessary state of affairs — our technology has different mechanisms for accessing data in a responsive way (memory) and storing that data for future use (disks). This, however, is not what most people think is going on. Most of our mental models (aside from programmers) are of a single document that we are directly creating and making changes to.

When that Save Changes dialog box, shown in Figure 17-1, opens, users suppress a twinge of fear and confusion and click the Yes button out of habit. A dialog box that is always answered the same way is a redundant dialog box that should be eliminated.

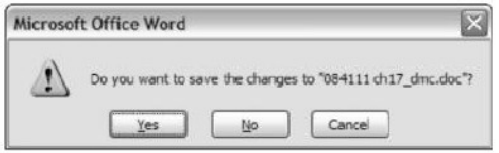


Figure 17-1 This is the question Word asks when you close a file after you have edited it. This dialog is a result of the programmer inflicting the implementation-model of the disk file system on the hapless user. This dialog is so unexpected by new users that they often choose No inadvertently.

The Save Changes dialog box is based on a poor assumption: That saving and not saving are equally probable behaviors. The dialog gives equal weight to these two options even though the Yes button is clicked orders of magnitude more frequently than the No button. As we discussed in Chapter 10, this is a case of confusing possibility and probability. The user might say no, but the user will almost always say yes. Mom is thinking, "If I didn't want those changes, why would I have closed the document with them in there?" To her, the question is absurd.

Save everything

- Automatically save everything as versions
- Make it easy to revert to previous versions
- Make it easy to abandon changes
- Make it easy to compare changes between versions
- Disk space is cheap

Good UI idioms to use

- Toolbars
- Tooltips
- Verbose modeless feedback (text that gives hints about what's going on that doesn't interfere with the user)
- These are highly recommended by the book as effective visual idioms. User only has to learn how to use them once and they instantly become useful thereafter.

Dialogs

- Dialogs should be thought of as going into another room in your house to accomplish a task.
- Dialogs are heavily overused in UI today, mainly because they are so easy to whip up.
- Imagine yourself doing work at a table at home. Ask yourself, “Would I go into another room to accomplish this particular task?” If yes, then use a dialog box. If not, don't. :)

Errors, alerts, and confirmation

- These three dialog boxes are “the most abused components of modern GUI design.” (location 6502)

Error dialogs

- Typically “poorly written, unhelpful, rude, and worst of all, never in time to prevent the error in the first place.” (location 6505)
- “Users never want error messages, they want to avoid the consequences of making errors.”
- “Most error message boxes are informing users of the inability of the program to work flexibly and are an admission of real stupidity on the application's part.” (location 6516)
- People hate error messages.

Error messages continued

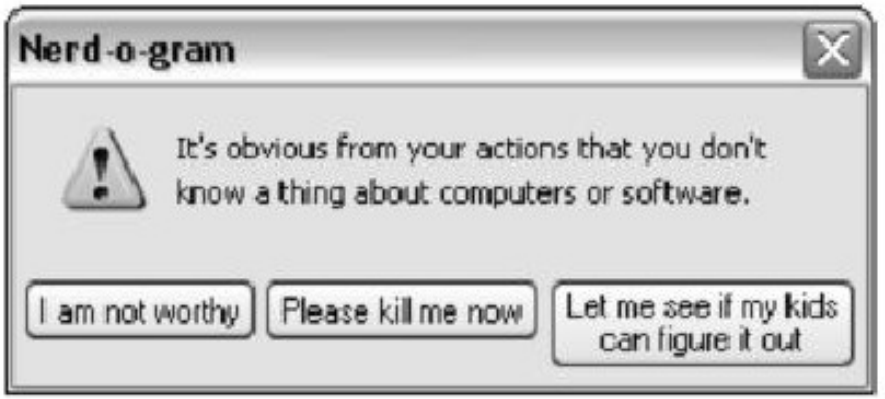


Figure 25-1 No matter how nicely your error messages are phrased, this is how they will be interpreted.

More on error messages

If we asked an assistant to enter a client's phone contact information into our Rolodex, and neglected to mention the area code, he would accept it anyway, expecting that the area code would arrive before its absence was critical. Alternatively, he could look the address up in a directory. Let's say that the client is in Los Angeles so the directory is ambiguous: The area code could be either 213 or 310. If our human assistant rushed into the office in a panic shouting "Stop what you're doing! This client's area code is ambiguous!" we'd be sorely tempted to fire him and hire somebody with a greater-than-room-temperature IQ. Why should software be any different? A human might write *213/310?* into the area code field in this case. The next time we call that client, we'll have to determine which area code is correct, but in the meantime, life can go on.

Eliminating error messages

- Error messages should be eliminated because they don't work (they don't prevent the error)
- Make it impossible for users to make errors.
- Give positive feedback when things go right and be silent when things go wrong.
- Users get humiliated when software tells them they have failed.

Alerts

Figure 25-4 A typical alert dialog box. It is unnecessary, inappropriate, and stops the proceedings with idiocy. Word has finished searching the document. Should reporting that fact be a different facility than the search mechanism itself? If not, why does it use a different dialog?

Alerts

- Main problem with alert dialogs is they take a user into a “different room”.
- Keeping the user informed is important, but it should be done as modeless visual feedback that doesn't require the user to press OK to keep going.

Alerts

Figure 25-5 This dialog, from AirSet Desktop Sync, is unnecessarily obsequious. We tell it to synchronize and are promptly stopped in our tracks by this important message. Do we really need the program to waste our time demanding recognition that it managed to do its job?

Confirmation Dialogs

- Also known as “passing the buck” on to the user
- Application doesn't want to be responsible for its actions
- May be used to avoid implementing robust undo system

> Confirmations pass the buck to users. Users trust the application to do its job, and the application should both do it and ensure that it does it right. The proper solution is to make the action easily reversible and provide enough modeless feedback so that users are not taken off-guard.

Confirmation Dialog (location 6658)

When an application does not feel confident about its actions, it often asks a user for approval with a dialog box, like the one shown in Figure 25-6. This is called a **confirmation**. Sometimes a confirmation is offered because the application second-guesses one of the user's actions. Sometimes the program feels that is not competent to make a decision it faces and uses a confirmation to give the user the choice instead. Confirmations always come from the program and never from the user. This means that they are often a reflection of the implementation model and are not representative of user goals.

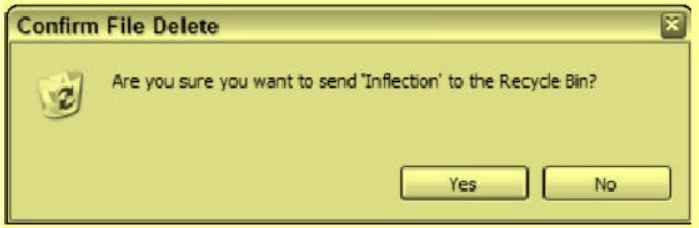


Figure 25-6 Every time we delete a file in Windows, we get this confirmation dialog box asking if we're sure. Yes, we're sure. We're always sure. And if we're wrong, we expect Windows to be able to recover the file for us. Windows lives up to that expectation with its Recycle Bin. So, why does it still issue the confirmation message? When a confirmation box is issued routinely, users get used to approving it routinely. So, when it eventually reports an impending disaster to the user, he goes ahead and approves it anyway, because it is routine. Do your users a favor and never create another confirmation dialog box.

Revealing the implementation model to users is a surefire way to create an unpleasant and inferior product.

When to use confirmations

The dialog that cried "Wolf!"

Confirmations illustrate an interesting quirk of human behavior: They only work when they are unexpected. That doesn't sound remarkable until you examine it in context. If confirmations are offered in routine places, users quickly become inured to them and routinely dismiss them without a glance. Dismissing confirmations thus becomes as routine as issuing them. If, at some point, a truly unexpected and dangerous situation arises — one that *should* be brought to a user's attention — he will, by rote, dismiss the confirmation, exactly because it has become routine. Like the fable of the boy who cried "Wolf," when there is finally real danger, the confirmation box won't work because it cried too many times when there was no danger.

For confirmation dialog boxes to work, they must only appear when a user will almost definitely click the No or Cancel button, and they should *never* appear when a user is likely to click the Yes or OK button. Seen from this perspective, they look rather pointless, don't they?

Eliminating confirmations

- Do, don't ask.
- Make all actions reversible.
- Provide modeless feedback to help users avoid mistakes

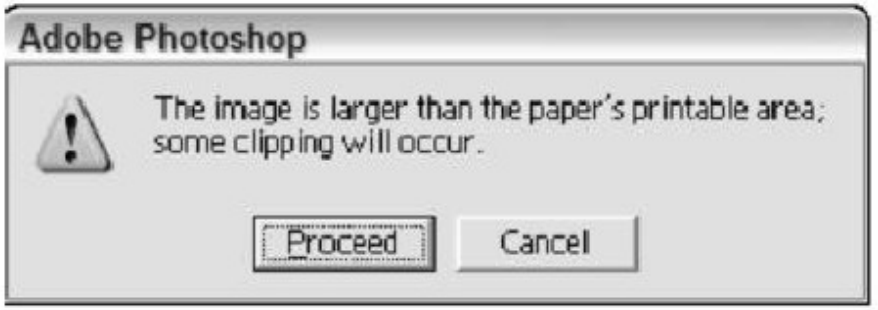


Figure 25-7 This dialog provides too little help too late. What if the program could display the printable region right in the main interface as dotted guides? There's no reason for users to be subjected to dialogs like these.