How To Design A Decent UI

Take a look at software
from a user's point of view
and improve your applications

TFM

- All examples are marked as good or bad practice for your convenience
- Good: 
- Bad: 
- The shown applications aren’t particularly worse than others. They just happened to be around, when I was looking for examples ;)

Table of contents

What is a poor UI anyway?

- Users usually want to concentrate on their task and get things done
- Whenever users are forced to think about the UI, they can't think about their actual task
- A poor UI disturbs the workflow
- One example:

Thinking for the machine

Better: Braindump

Wrong flow


Suche starten
Anzeige:
2.
Vollanzeige
Kurzanzeige
Sortieren nach:
Erscheinungsjahr
Titel
Dokumentnummer
Neue Suche
Wildcards: %, _
Suchregeln
Suchtipps
Autoren/Herausgeber:
Nachnamen (ohne von, de etc.)
1.
Institutionen:
Titel/Stichworte:
Schlagworte:
Abkürzung:
Serie:
Verlag:
Erscheinungsjahr:
bis
ISBN/ISSN:
z.B. 3-89864-260-7
Standort:
- Es wurde kein Standort aus der Liste gewählt. -
Doknr:
bis

Jargon

- I would translate both “Stichwort” and “Schlagwort” as “keyword” ...
- In this case it’s jargon from the client’s domain. Computer-jargon is much more frequent in apps. The only admissible jargon is from the user’s **task domain**!

Table of contents

Why, oh why?

- Conspiracy?
- Fnord?

Lack of awareness

- Huge gap in computer experience between those who program and those who is programmed for
- Developers often think they know what users want. Aren’t they themselves users?
- Yes, but far from average ones!
- This is worsened by the inside perseptive one automatically acquires when involved in a project

Lack of training

- Strongly related to the last point
- There is seldom training in usability / interaction issues
- The one to program the UI is often also the one to design it
- Although these two task call for completely different expertise

People are fuzzy

- Computer-people tend to care more about the machines than the users
  - Hard facts from benchmark tests, etc.
- In usability it’s less clear and often about trade-offs

Too little time / money

- Alas, the limitations of the real life
- In private projects as well as in companies usability is seldom a priority ...

Pressure from higher instances

- Sometimes you know better, but ...
- ... someone from management / marketing / the client wants it changed

Table of contents

Why it's worth to do it right

- More and more people use computers that don't know anything about their inner workings
- And they don't even want to know!
- To them a computer is only a tool
  - Not also a hobby, as maybe for many of you

Why it’s worth to do it right

- To the average user the UI **is** the application
  - They’ll never know about your clean code ...
  - They’ll never know about your cleverly designed algorithms ...
- Differences in technology decrease -> others distinctions become more important
- As not many put emphasis on usability, it’s your chance to stand out from the crowd

Table of contents

How to do it right?

- All users have one thing in common:
  They are human beings
- As humans we share some common frailties and limitations
- Take them into account and you're half way there to a decent UI
- Let's look at those "human factors" ...

Table of contents

Timing

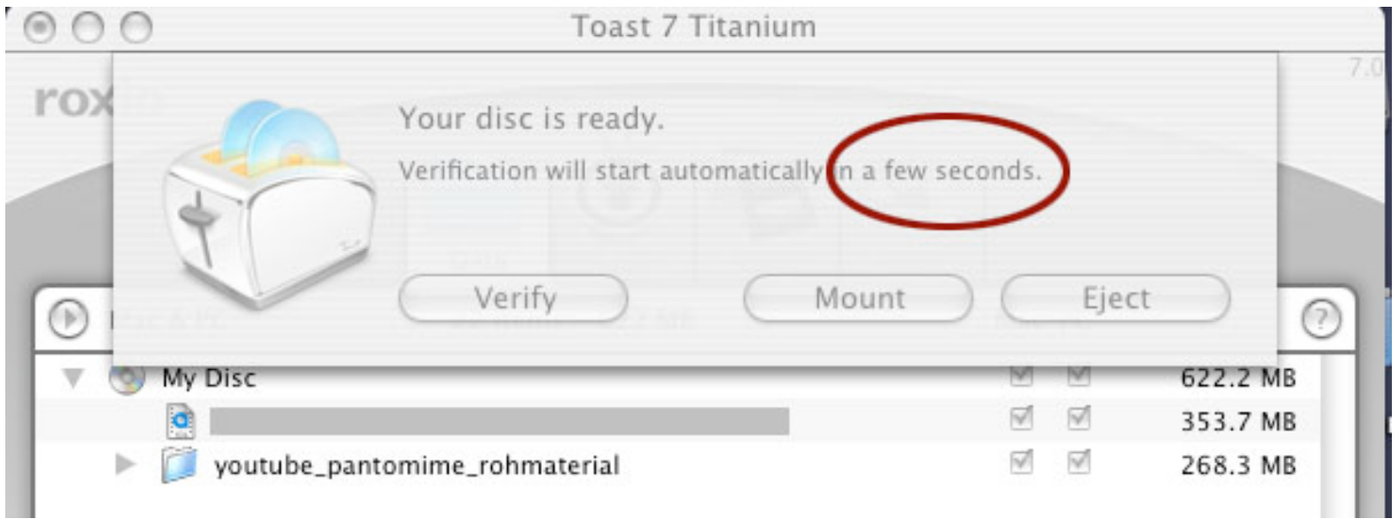
- **100ms - perception of cause and effect**
- For everything that can possibly take longer than 100ms display either
  - a busy cursor (short waiting time expected) or
  - a progress bar (longer waiting time expected)
- If you don’t ...
  - users will wonder whether their request was received
    -> provokes random clicking
  - users are likely to make (wrong) assumptions about causality

Timing

- **1s - minimum reaction time for unexpected events**
- Give users enough time to react to messages etc.
- This is the max time your application has before random clicking sets in
- If you can, don’t require users to react within a given time frame
  - It will always be either to long or to short

Timing

- Example: Toast 7 Titanium (CD burning app)



- Verification starts automatically after “a few seconds”
- No countdown
- I always miss the time frame

Timing

- **10s - typical attention span**
- Upper limit for one step of a task (e.g. one screen of a dialog)

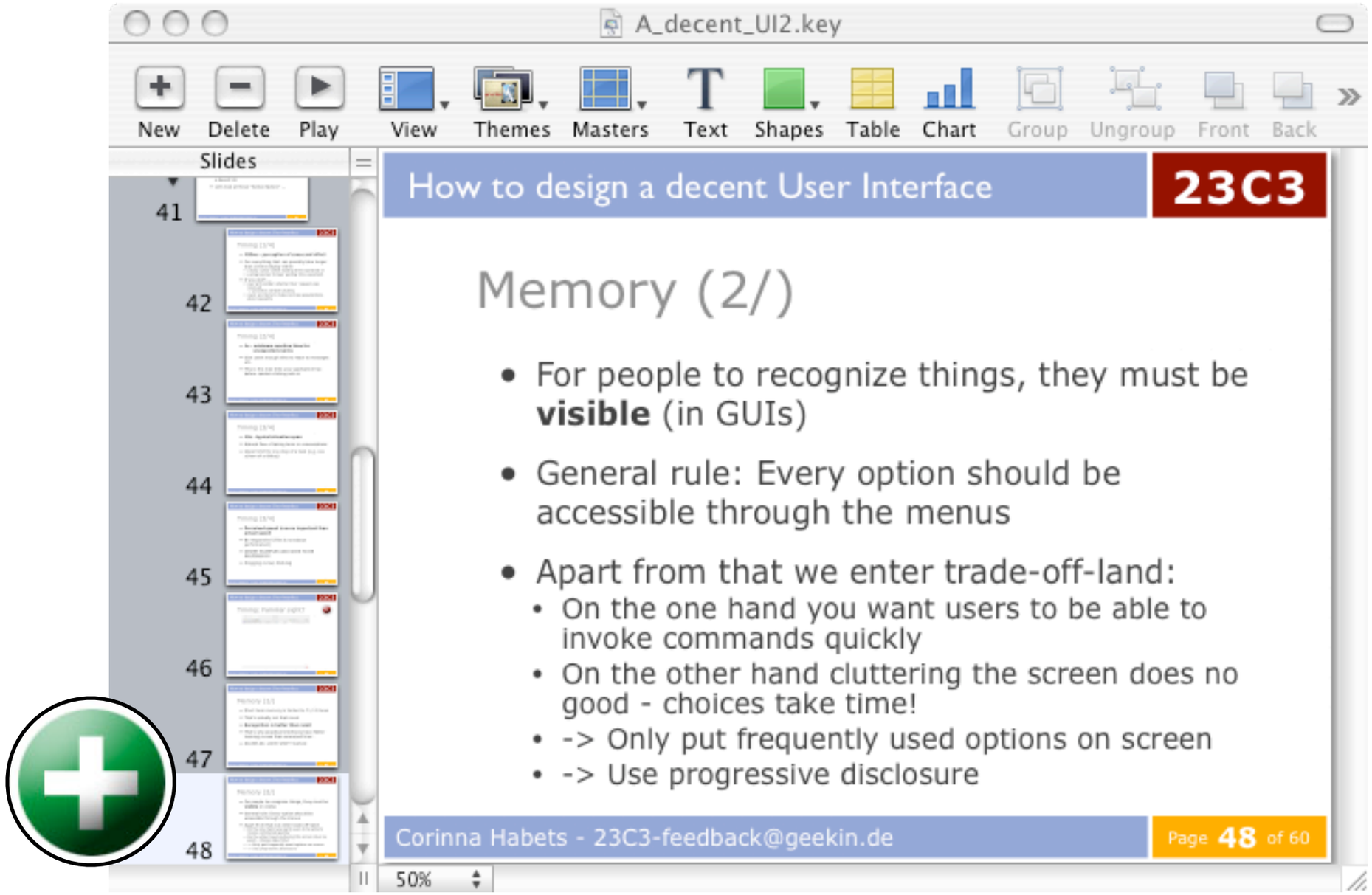
Timing

- **Perceived speed is more important than actual speed**
- Be responsive! (This is not about performance!)
- There are many ways to increase responsiveness:
  - Favour user-perceivable routines in the internal processing queue
  - Preprocess likely requests
  - ...

Memory

- Short term memory is limited to 7+/-2 items
- That's not much
- **Recognition is better than recall**
  - Could you describe your way home?
- That's why graphical interfaces have flatter learning curves than command lines

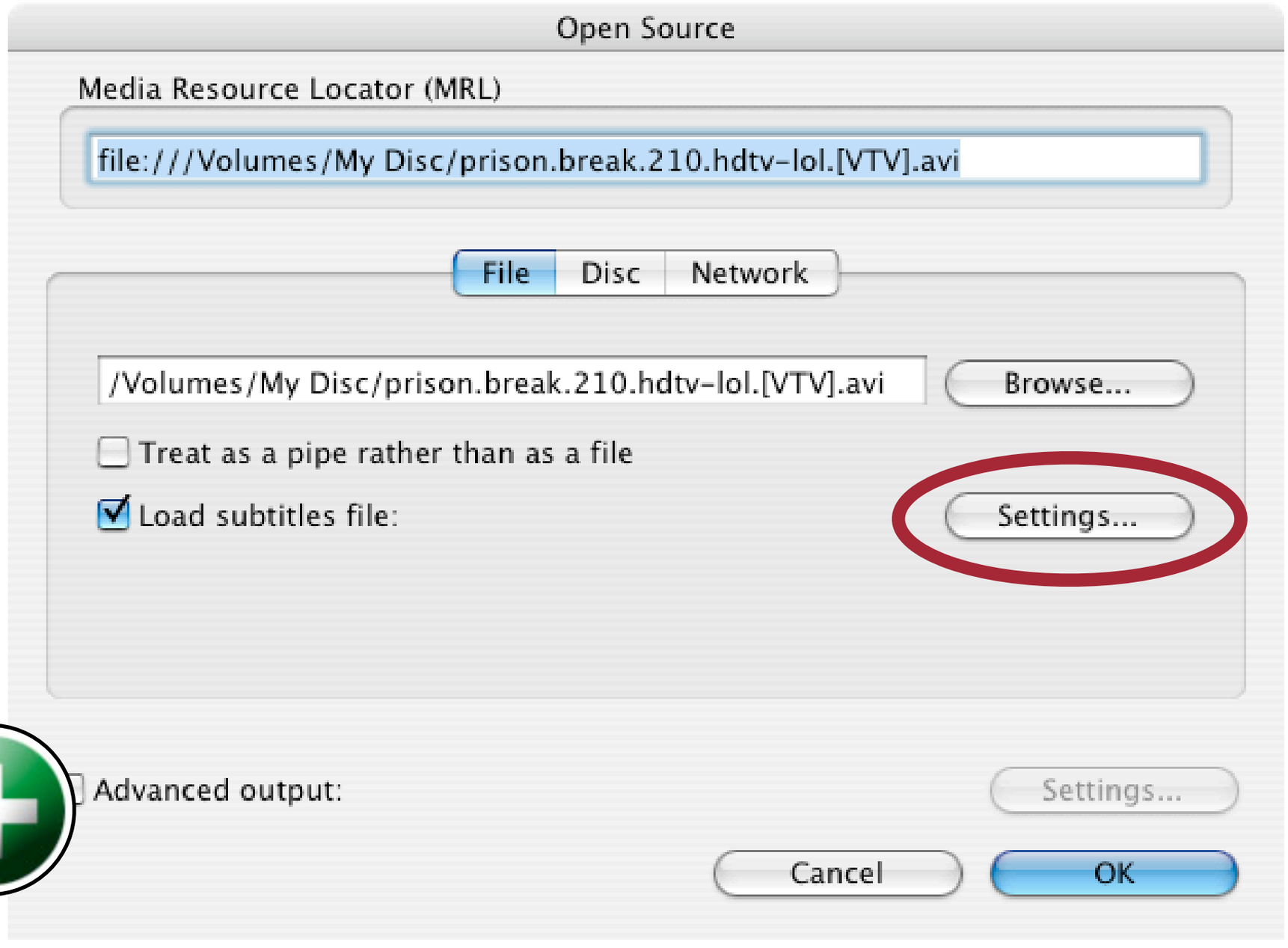
Recognition gives orientation



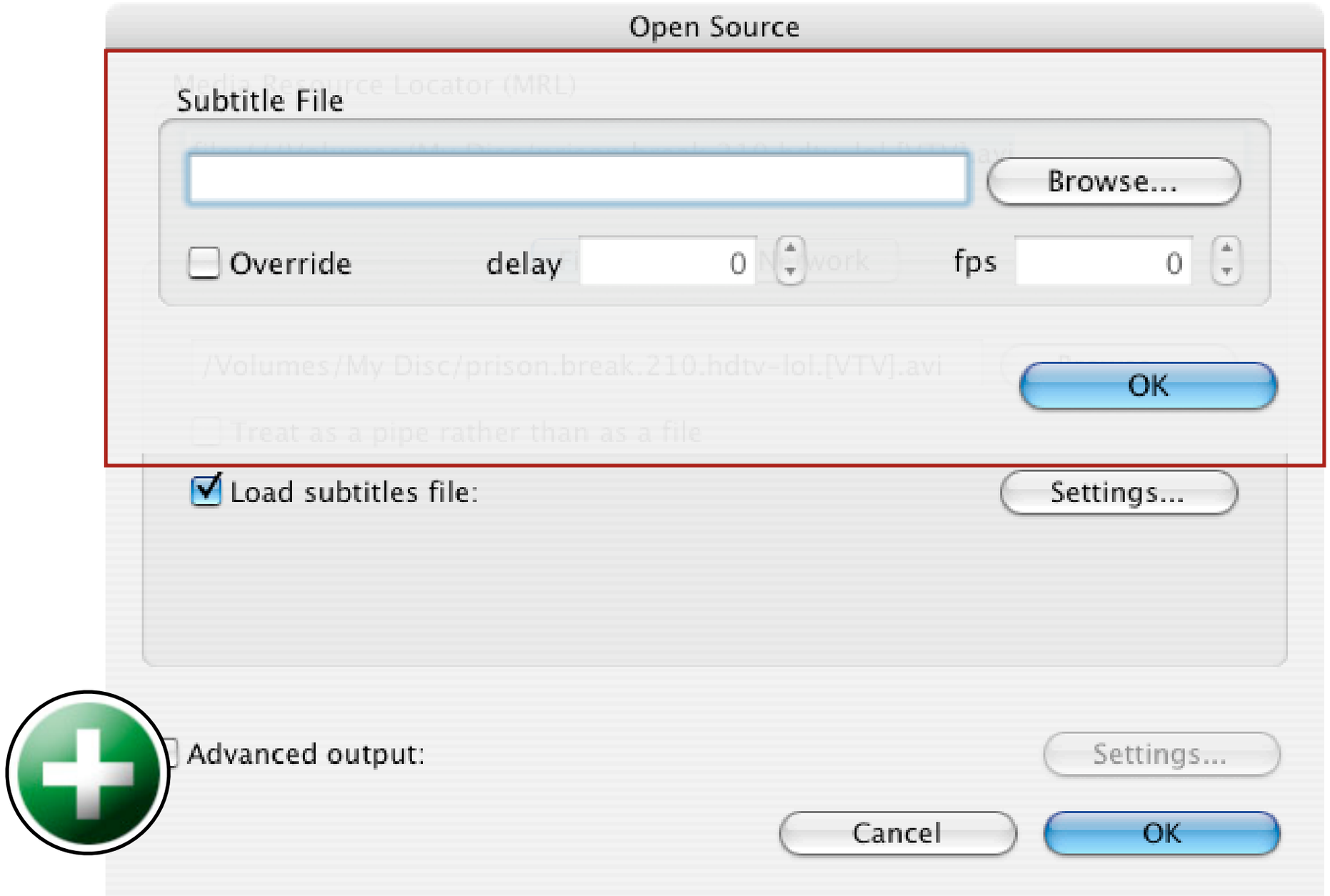
Memory

- To recognize things, they must be **visible**
- General rule: Every option should be accessible through the menus
- Apart from that we enter trade-off-land:
  - On the one hand you want users to be able to invoke commands quickly
  - On the other hand cluttering the screen does no good - choices take time!
  - -> Only put frequently used options on screen
  - -> Use progressive disclosure

Progressive disclosure



Progressive disclosure



Attention

- Only one task can have conscious attention
- If we have to split our attention between tasks our “performance” suffers
- Luckily the brain has a workaround: it can **automate** / form habits
- Automated tasks don’t need conscious attention

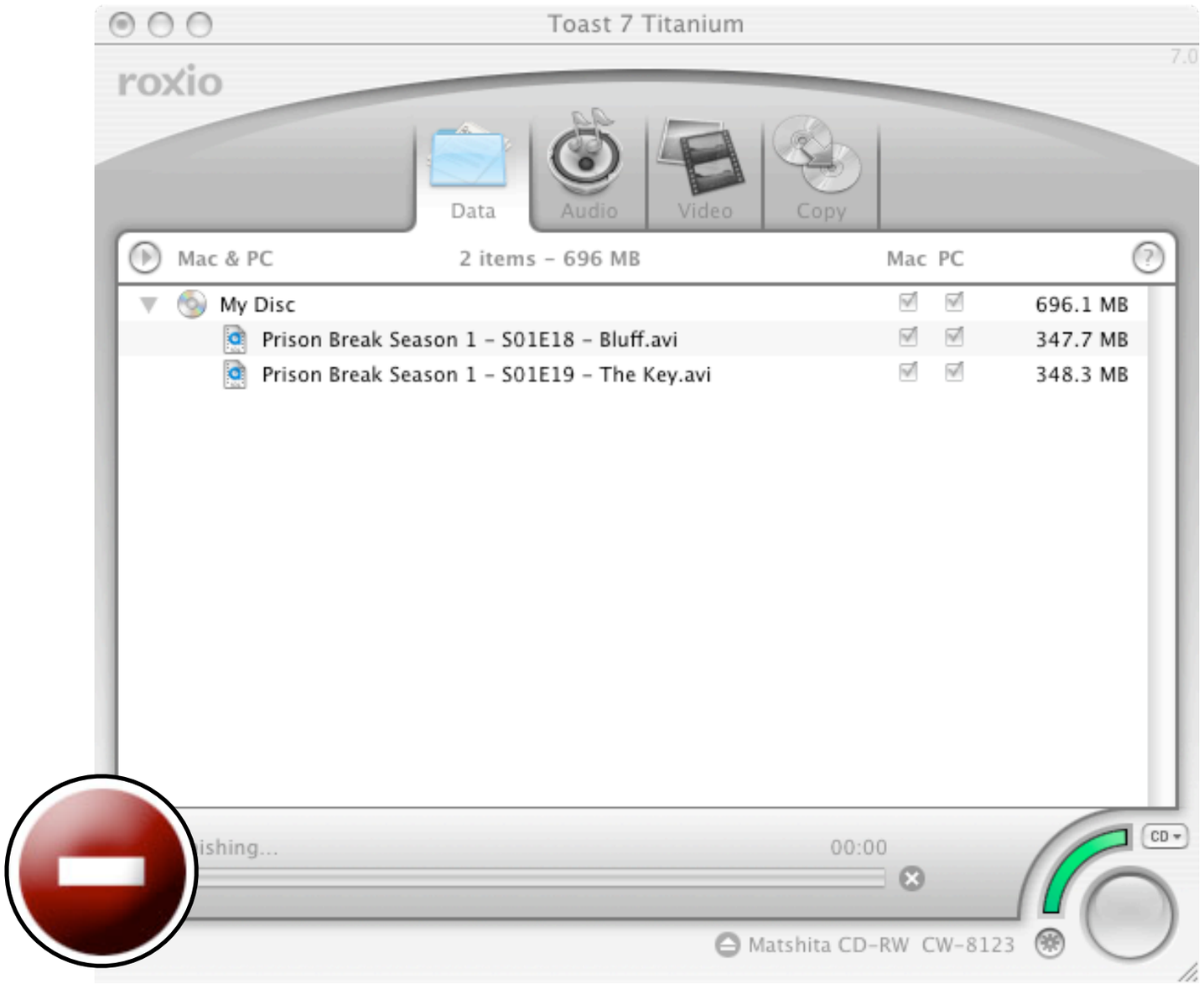
Attention

- We can’t automate “problem-solving” and creative tasks
  - Writing a text
  - Editing an image
  - Programming
- The task that has attention should be the user’s task ...
- ... not figuring out how to operate the application

Consistency

- To be consistent means to be predictable
- Don’t surprise users
- If two objects are the same (from the user’s point of view), they should offer the same functionality
- Let’s look at an example to illustrate **in**consistency

Inconsistent progress feedback

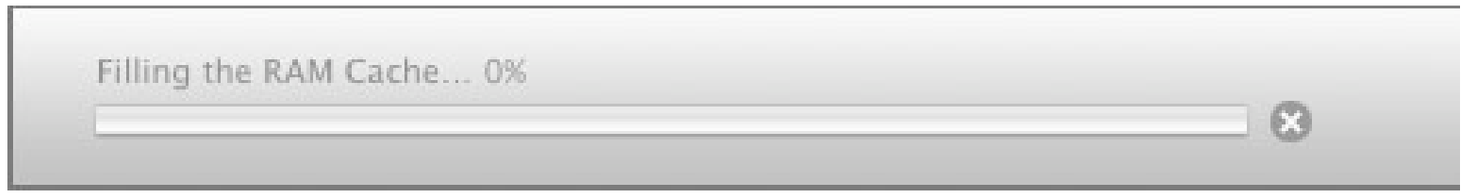


In "Toast 7 Titanium burning a CD has 3 phases:

- Filling the cache
- Writing CD
- Finishing CD

Inconsistent progress feedback

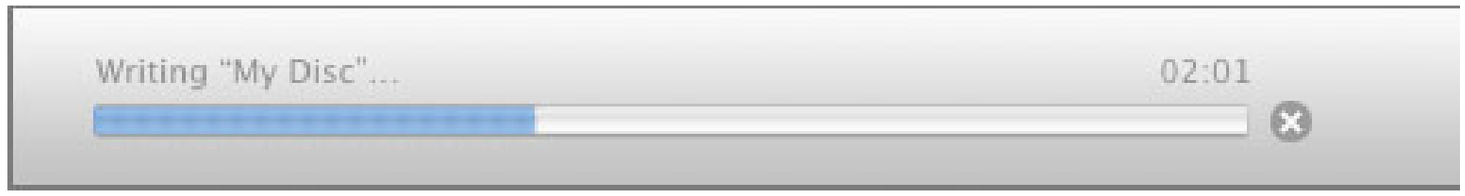
- Filling the cache:



- Giving progress in percents in text form
- No progress bar

Inconsistent progress feedback

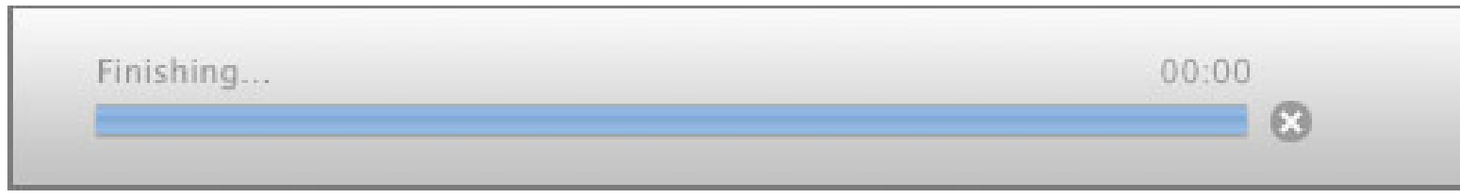
- Writing the CD:



- Giving "estimated time left" in text form
- Progress bar

Inconsistent progress feedback

- Finishing:



- No textual feedback
- No progress bar
- More than half a minute without feedback!

Inconsistent progress feedback

- Three very similar processes
- Three (well, actually just two) different ways to show progress
- How come? Some speculations:
  - Different underlying concepts
  - Lack of communication / management:
    The three parts were programmed by different developers and Alice, Bob, and Eve all did their piece the way they wanted
  - Nobody cared

Consistency

- Consistency is a wide and blurry concept
- Let’s concentrate on **text consistency**
- It’s more concrete
- Already a huge improvement when done right

1. Avoid ambiguity

- Language is full of ambiguity
- Take “cock” for example. What are you thinking about?
- In everyday life there’s lots of context to let us figure out what is meant
- Text in applications often consists of isolated words (e.g. menu commands)
- **Wording** is a difficult task

1. Avoid ambiguity

- Example: In an 3D-rendering-application with multiple camera views on a model
- There's a command "View Settings"
  - What will it do?
    - Show the settings of a view?
    - Show (lets you "view") some other settings?
  - Worst case: Both outcomes in different places of the app

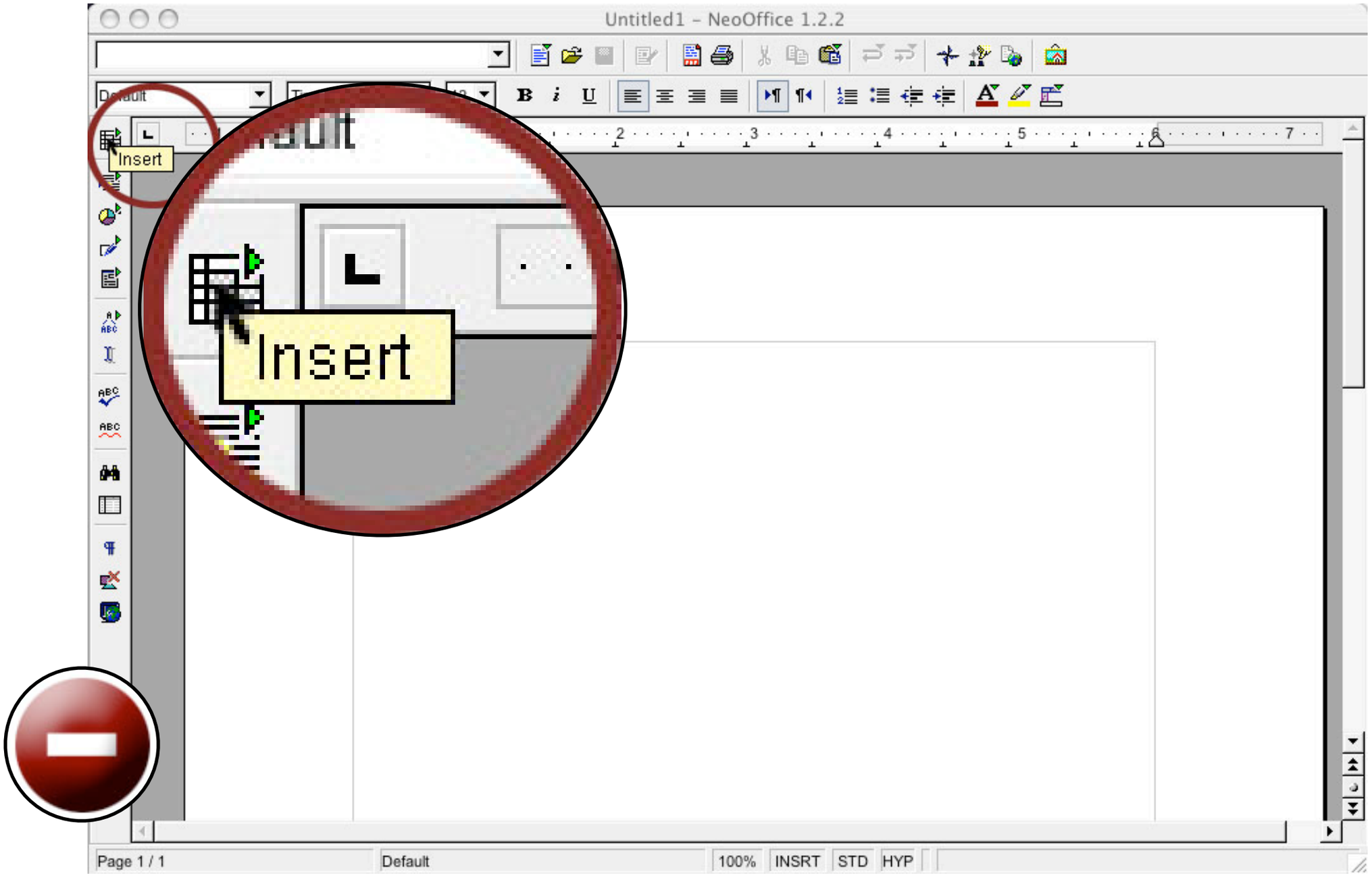
2. Call “x” always “x”

- Choose a unique name for each concept or object in your app and stick to it
- Example: My father got the login-data for his internet connection. We needed it to configure his email-account
  - His documents contained a “passphrase”
  - The login screen asked for a “password”
  - Is “passphrase” and “password” the same?
- To us it’s obvious - to non-computer-people it’s not!

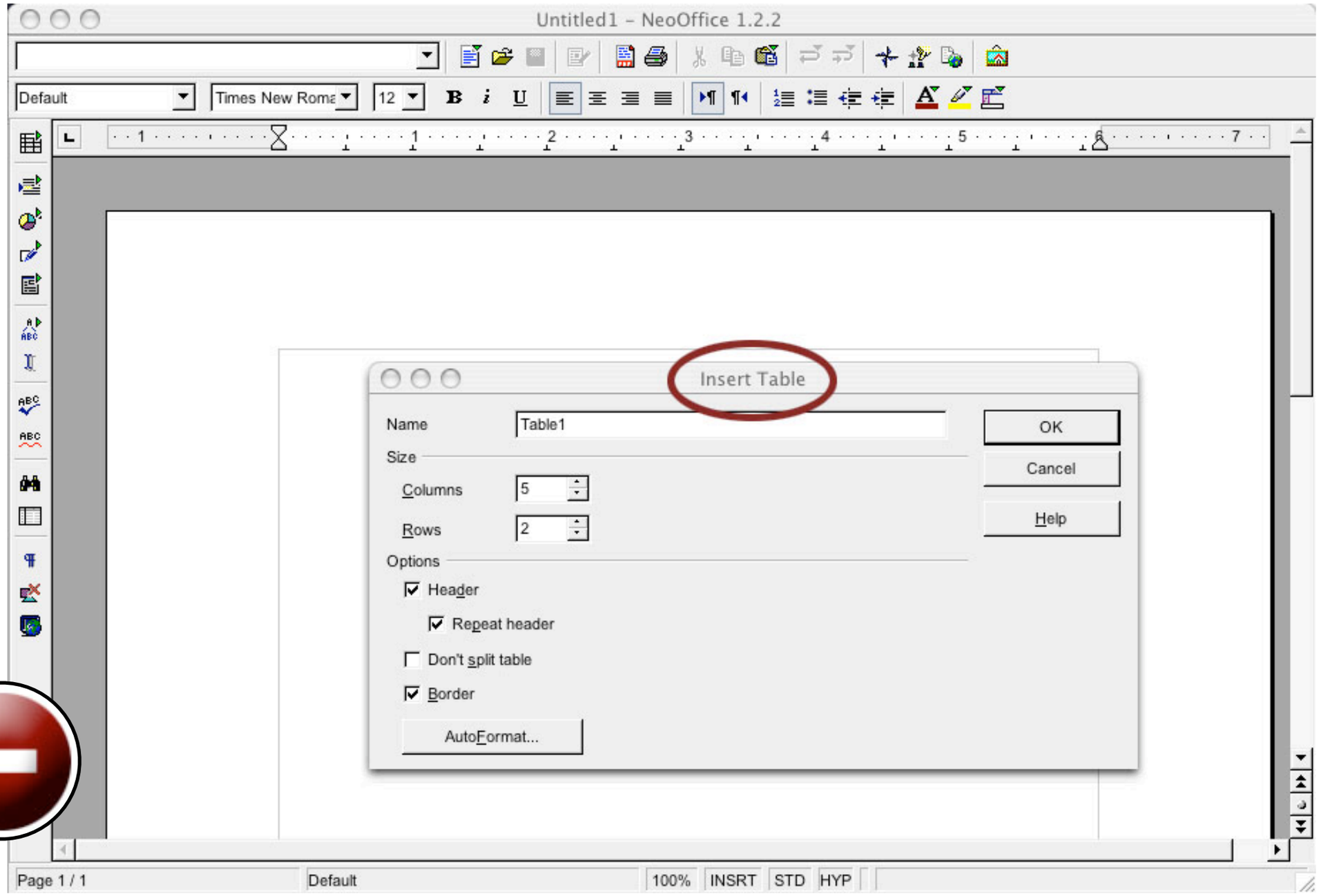
2. Call “x” always “x”

- Especially with more than one developer term consistency is hard to achieve
- You’re best bet is an “application lexicon”
  - It contains all the terms to appear in the app
  - Assign someone to check the consistent use of these terms and these terms only
    -> easier with central message file
  - Get a technical writer if you can

2. Call “x” always “x”



2. Call “x” always “x”



Errors

- Humans make mistakes. Always have, always will
- Still many apps don’t seem to take that into account
- Although it would we quite easy ...

1. Prevent errors

- Don’t let users invoke commands that aren’t applicable
  - Successfully employed in graying out menu items
- Provide sensible defaults. Don’t ask users to take choices they might not know about
- Don’t let users enter freeform text if you could be more specific

1. Prevent errors

Datum (JJJJ-MM-TT):

Datum:

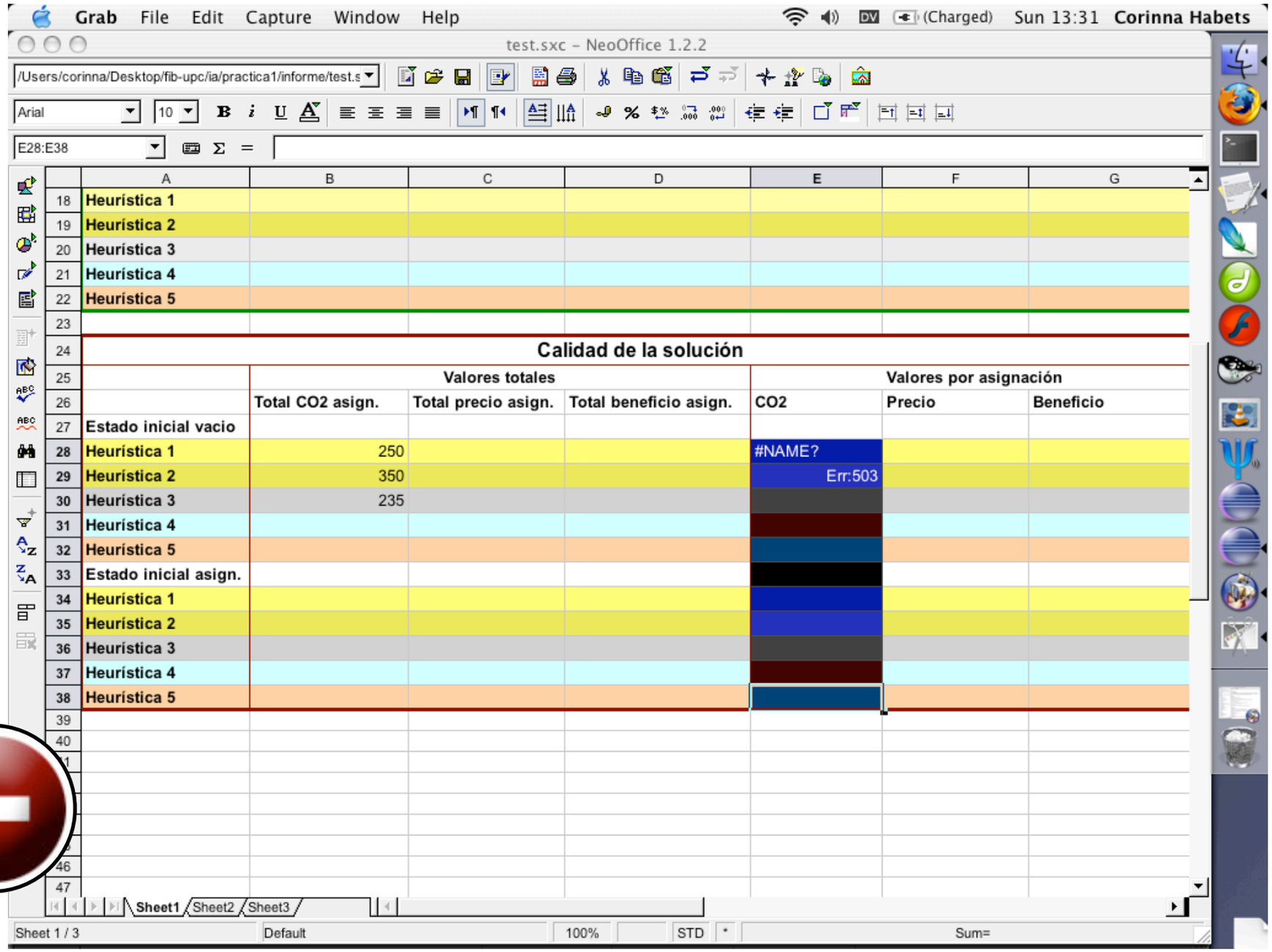
Tag: Monat: Jahr:

01 01 2006

2. Helpful error messages

- Users don’t commit errors on purpose. There’s no need to bark at them!
  -> Be polite!
- Provide hints on how to solve the prob
  - “Caps-lock activated?”
- Error codes are for debugging and debugging only!

Geek "names" for errors



3. Preserve users' work

- **Never** lose data!
- Provide undo-/redo-functionality for everything
  - It needs to work flawlessly, because users will rely on it!
- User's work is also data entered in a dialog

How to do it right?

- Taking common human factors into account is one half of the way
- But users are also individuals
- They vary greatly e.g. in
  - Experience with computers
  - Experience in the task domain
  - Age
  - Gender
  - Cultural background
  - Dissabilites

How to do it right?

- What's right for one user can be wrong for another
- To get the flow of interaction and the interface right, you need to define for **whom**
- **Who's the user?**

Table of contents

Who's the user?

- Sometimes the user is already determined, because the app is for a specific client
- In other cases you have to choose the target users yourself
- But many applications fail to do that
- "For everybody" has a lot of appeal
- But you can't please everybody
  - Mostly "experience task domain" is the distinctive factor

Users and their tasks

- If you know the user, then the next interesting question is:
  What's the task?

What's the task?

- What are the things your target user group has to or wants to do with your app?
- What are frequent tasks? What will they rarely use?
- Finding out about this helps to limit "featuritis"

What's the task?

- Will they receive training?
- How often will they use the app?
  - Once?
  - Every few weeks?
  - Daily?
- Are other applications involved?
  - -> provide import- / export-possibilities

What's the task?

- Some ways to find out about the tasks:
  - **Site visits** - go and observe the users in their natural enviroment
  - **Questionnaires** - ask in text form, mass compatible
  - **Interviews** - ask face to face, possibility to clear uncertain situations
- Think in terms of "What will the user be able to do"
  - Not about the implementation

User-centered design

- Answering our two main questions is the beginning of **user-centered design** (UCD)
- Definition:

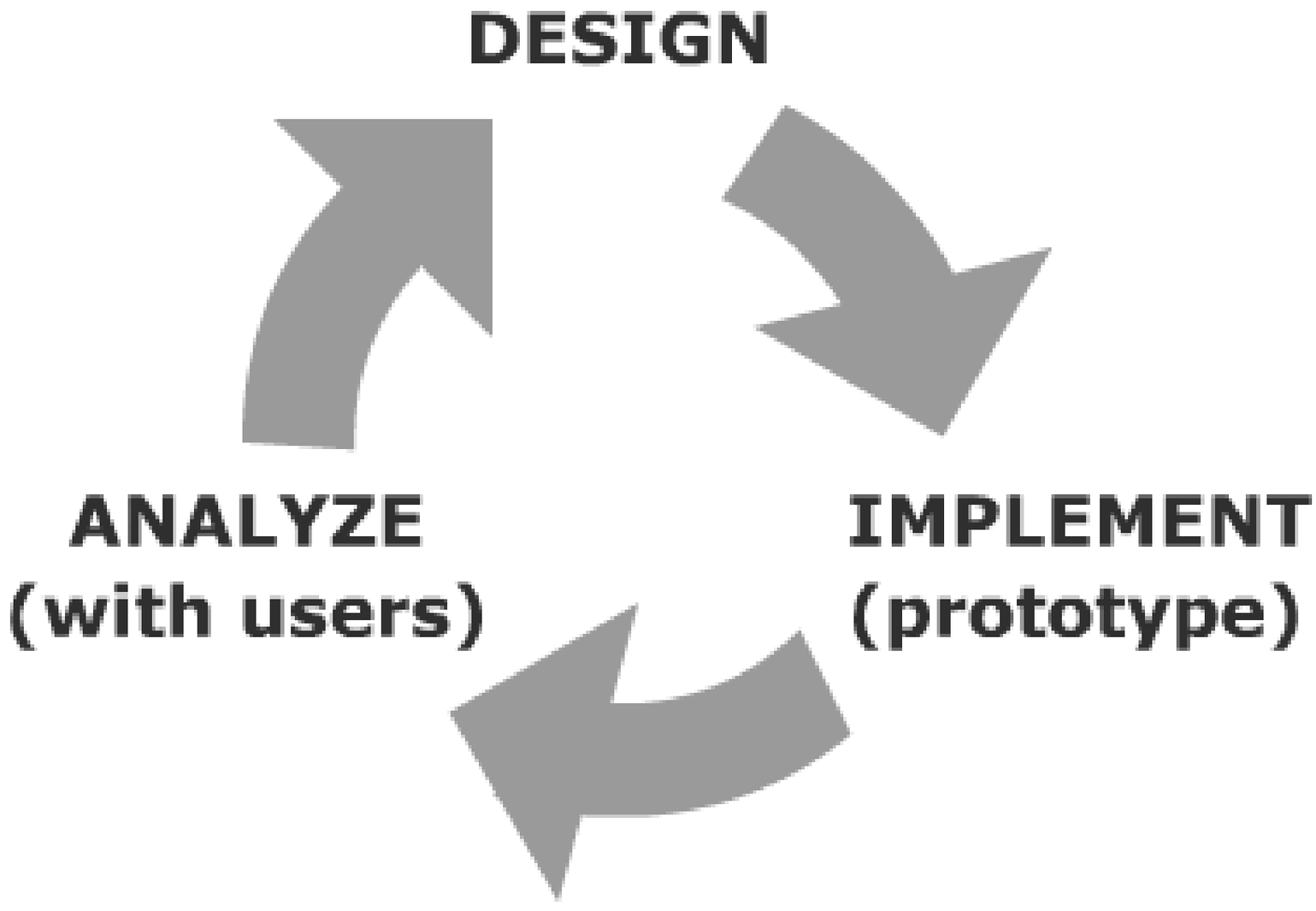
User-centered design tries to optimize the user interface around how people can, want, or need to work, rather than forcing the users to change how they work to accommodate the system or function.

(from Wikipedia)

User-centered design

- The main idea is to include users during the whole development process
- After identifying user and task you enter an iterative process:
  1. You design the application
  2. You implement a prototype
  3. You test the prototype and analyze the results
  4. Go back to 1. until ($final_product_finished)

Iterative design process



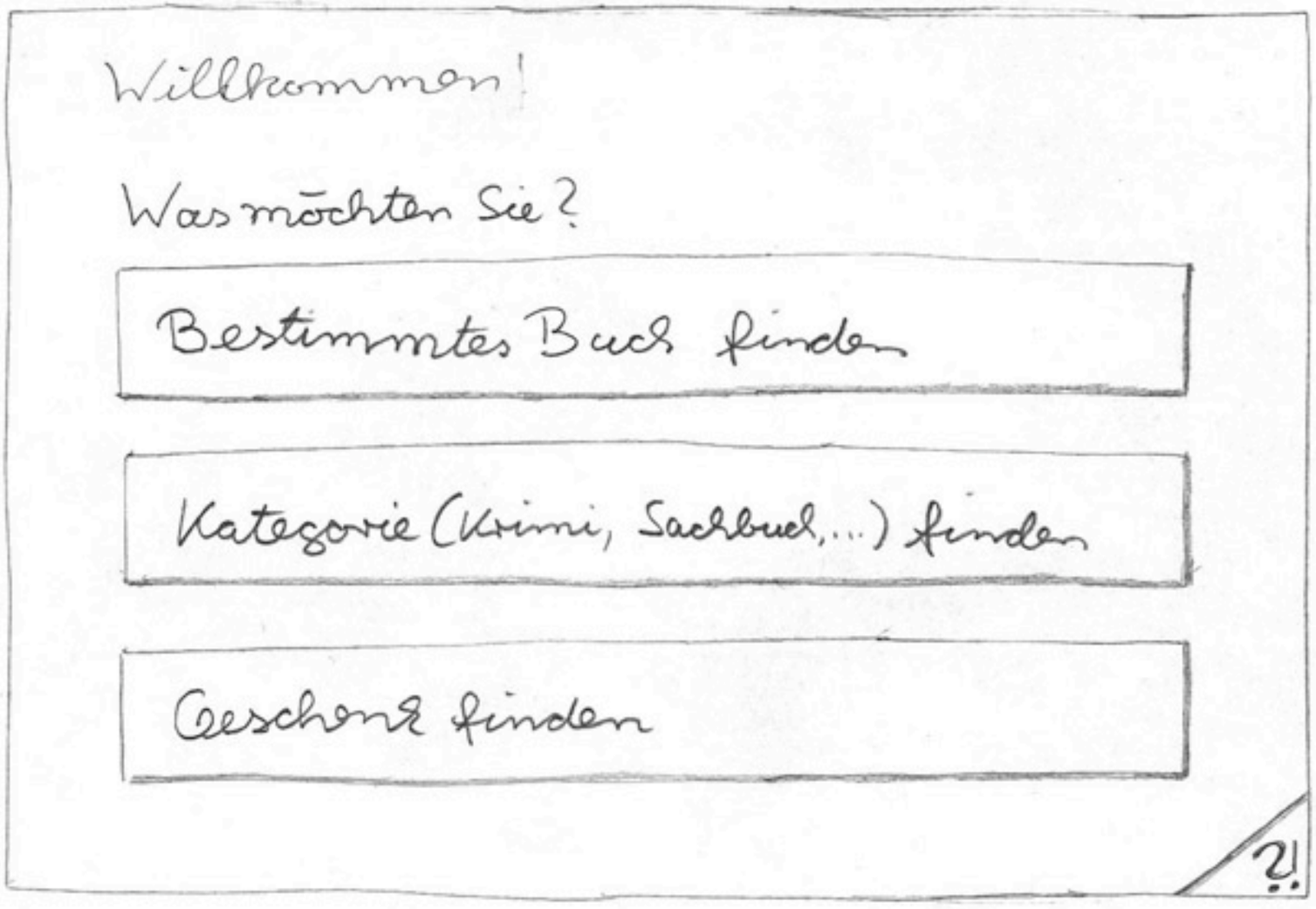
Iterative design process

- The importance of early testing can not be over-estimated!
- Good interaction is like security - you can't just "add" it in the end
- The earlier you test, the easier you can correct misconceptions
- And let go of dear - but inappropriate - ideas

Prototyping

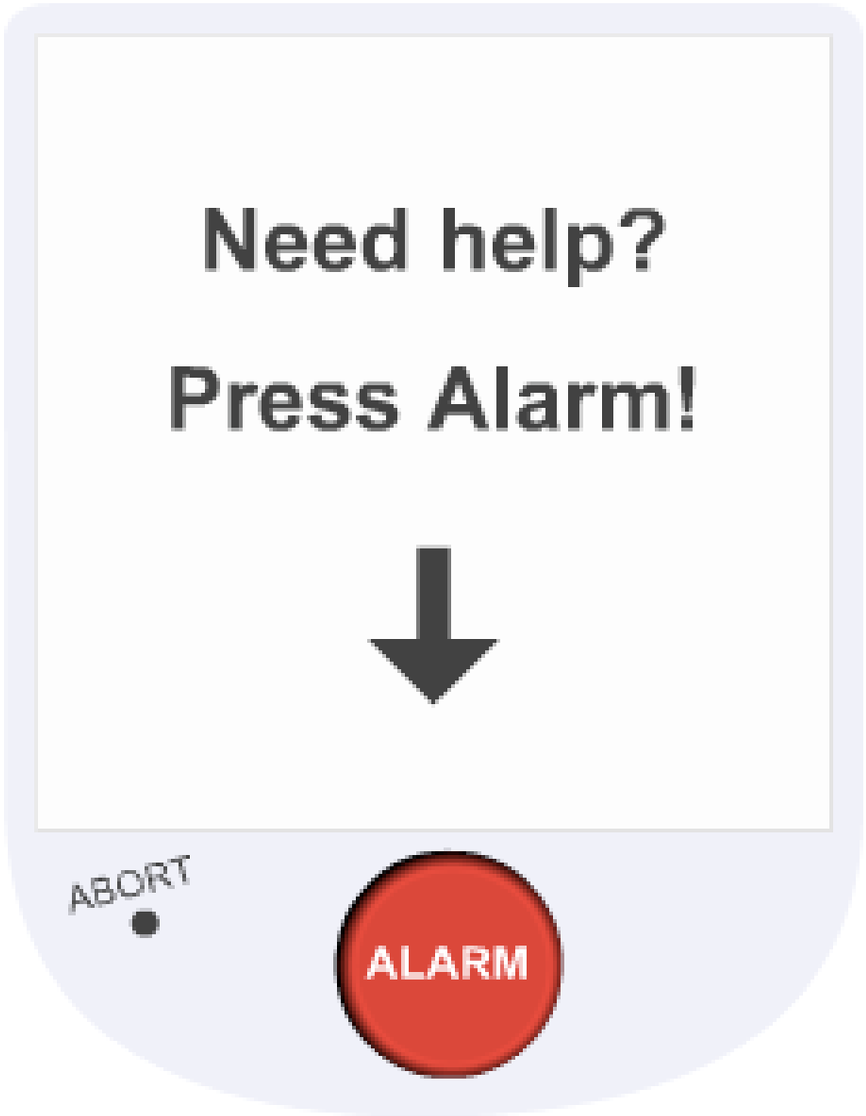
- A prototype needn't be software!
- Consider:
  - **Paper-prototype** - drawings of the screen
  - **Post-it-prototype** - can simulate menus, etc.
  - **Photoshop-/Gimp-MockUp** - realistic but without function
- Or low fidelity software prototypes
  - **Flash prototype** - simulates flow of UI but without real functionality

Prototyping



Paper prototype for a fictive tool to (physically) find a certain book in a large book store

Prototyping



Flash prototype for a fictive handheld device to coordinate medic units / police / firemen during mass events

(shown in interaction)

Prototyping

- The more iterations, the better
- With each iteration the application gets more concrete and detailed

Table of contents

Summary

- Users are often forced by applications to figure them out, rather then concentrate on their work
- Taking common human limitations into account already leads to improvements
- Apart from that many developers think they know what users want
- But they don’t unless they ask their users

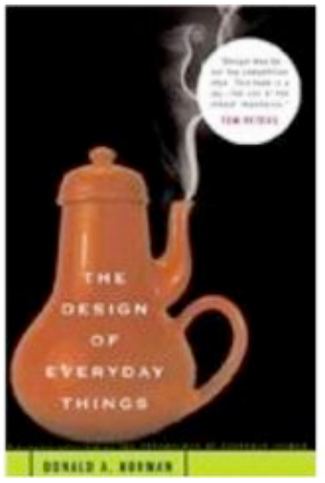
Summary

- The more often and the earlier they ask (i.e. test), the better
- Getting it right from the beginning saves a lot of time and trouble in the end
- And you get a superior product :)

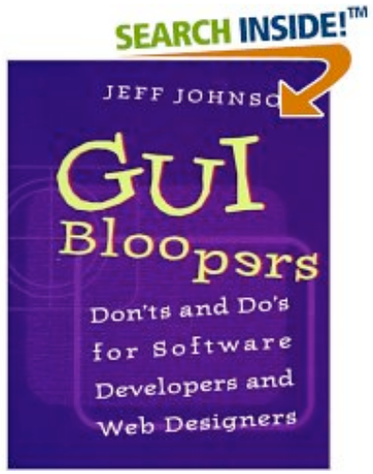
Table of contents

Do you want to know more?

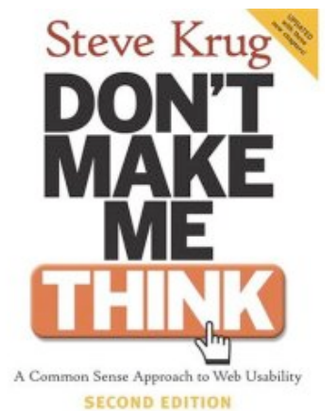
1. The Design of Everyday Things
   by Donald Norman



2. GUI Bloopers
   by Jeff Johnson



3. Don't Make Me Think!
   by Steve Krug



Do you want to know more?

- In the paper accompanying this talk you'll find additional information:
  - Links
  - Passage on how to run a user test

Thanks!

So long and thanks for all the attention :)

Please give feedback to this talk
via the congress feedback system