Software Development

HCI in Software Development

Motivation for good design
Goals for the designer
How design fits into the software life cycle

Case study: Microsoft Windows 95

Motivation for Good Design

It is easier to point to what happens when the design is poor:

- Under-use of expensive computer technology
  - People use alternative (unofficial?) systems
- Inefficient use of systems
- Irritated/frustrated users
- Physical danger (safety-critical systems)

Motivation for Good Design

Concentrating on the benefits of good HCI:

Customer satisfaction (customers often prefer usability to functionality) and hence £££

Good reputation

Reliability (users don' t make as many errors)

Allows users to focus on their work, not the interface

- Better turnover
- Greater flexibility
- Better use of staff

Goals for the Developer

The big picture:

- Adequate functionality
- System reliability
- Good interface design

Also other features such as standardization, portability, consistency, integration

Shneiderman Section 1.2

Specific Goals for the Designer

General aims:

- Improving quality of work produced
  - Discouraging errors, for example
- Increasing throughput
  - Making it easy to do things quickly
- Minimization of training time
  - Making it easy to learn

…but we need measures in order to see whether the goals are achieved!

Shneiderman, Section 1.3
Preece, Section 1.3

Goals for the Designer

Specific measurable goals:

- Subjective satisfaction (use surveys with satisfaction scales, e.g. like feedback questionnaires for lectures)
- Rate of errors (how many, what kind?)
- Speed of performance (use benchmark tests)
- Time to learn (measure for a specific set of tasks)
- Retention over time (how easy is it to remember how to use?)

There are certain trade-offs that might have to be made.

The Mud-flinging Approach

How?

- Throw mud at wall, hope it sticks; if it slides off it's no good.
- Analogous: throw design out into user community, hope it is usable; if not, oops! try again!

Advantage: new designs are rapidly developed

Major disadvantages:

- Loss of customers (the best ones who are most eager and come early to your site/product)
- Loss of reputation even if you fix it the next time - web start up companies are a good example, e.g. boo.com

Testing - Crucial for good design

Some software developers think that feedback after the product's release will be sufficient. It isn't (see mud-flinging).

Perhaps the most obvious way that next comes to mind is to do all the development, then test before releasing, then fix bugs, but that's not a good way to do it!

Suppose a fundamental design flaw was discovered at a late stage? Lots of wasted effort…

Good design requires feedback on the design, so testing must occur much earlier in the development.

The Software Life Cycle

Different models all include these main areas of activity to a greater/lesser degree:

- Specification
- Analysis
- Design
- Implementation
- Testing
- Maintenance

Where should the designer be considering good design?

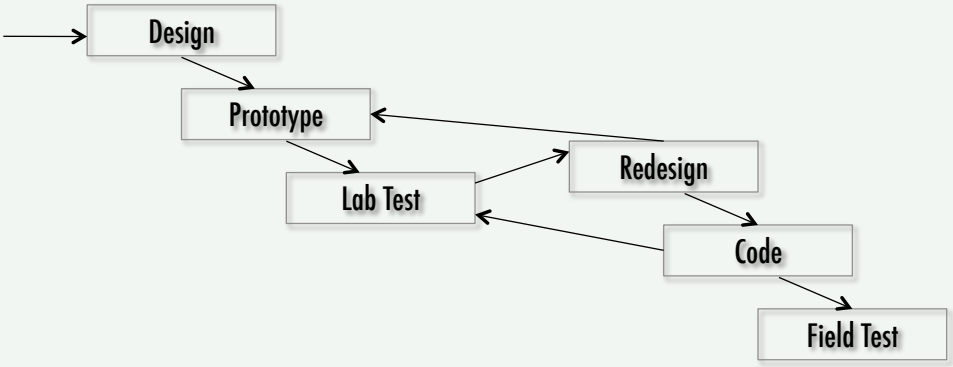
Shneiderman, Section 3.4

Usability Engineering Model

It is possible to test from very early on. Possible places to test during the development are:

- Early design testing (of previous versions or competitors' products)
  - **provides goals for the designers**
- Middle stages design testing
  - **validates designers' choices, provides feedback for improvement**
- Later stages
  - **ensures that product meets the design objectives**

Rapid Iterative Design



Involves a lot of prototypes and testing
Gives fast feedback about design flaws

Scenarios

These are a special kind of prototyping to assist with making rapid prototyping simpler

The idea behind prototyping is to cut down on costs by implementing only parts of the system

- Horizontal prototypes
  - **full range of features**
  - **functionality reduced**
- Vertical prototypes
  - **limited range of features**
  - **each feature implemented has full functionality**

Scenarios combine horizontal/vertical ideas in the extreme by implementing a limited range of features with reduced functionality

Case Study: Microsoft Windows 95

Scenario:

To produce a user interface for Windows 95 (an upgrade to Windows 3.1 and Windows for Workgroups 3.1), in just *18 months* for the design *and* coding

See http://acm.org/sigchi/chi96/proceedings/desbrief/Sullivan/kds_txt.htm

Goals:

To make Windows easier to learn for beginners

To make Windows easier to use for computer users

Implementers (12), Design team (12):

Interdisciplinary, including product designers, graphic designers, usability testers and computer scientists

The Task Ahead

Careful design and testing were needed to avoid worsening the design for millions of users

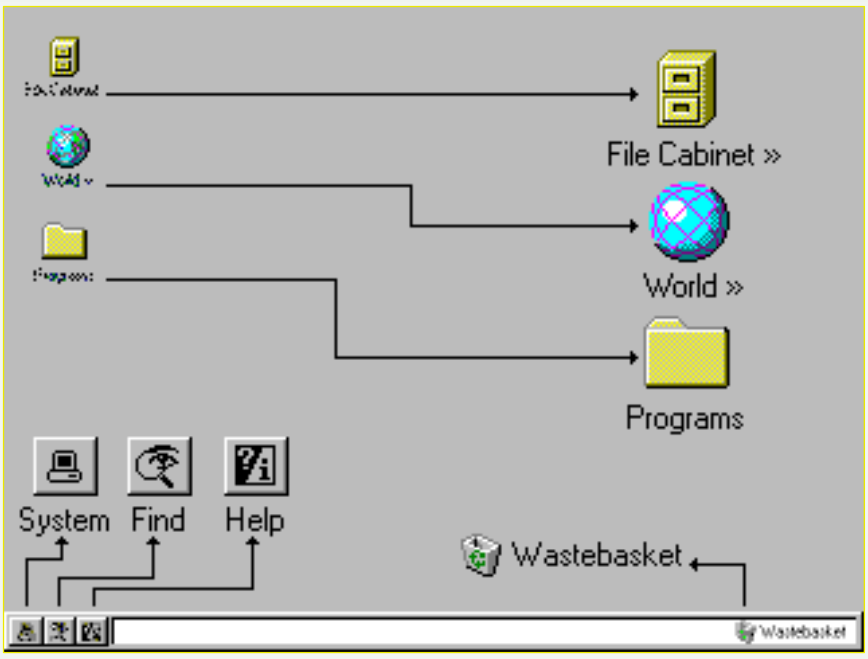
With a largely-untapped home computers market, the product had to be usable for beginners

Very little data on beginners - existing data only concerned intermediate/expert users

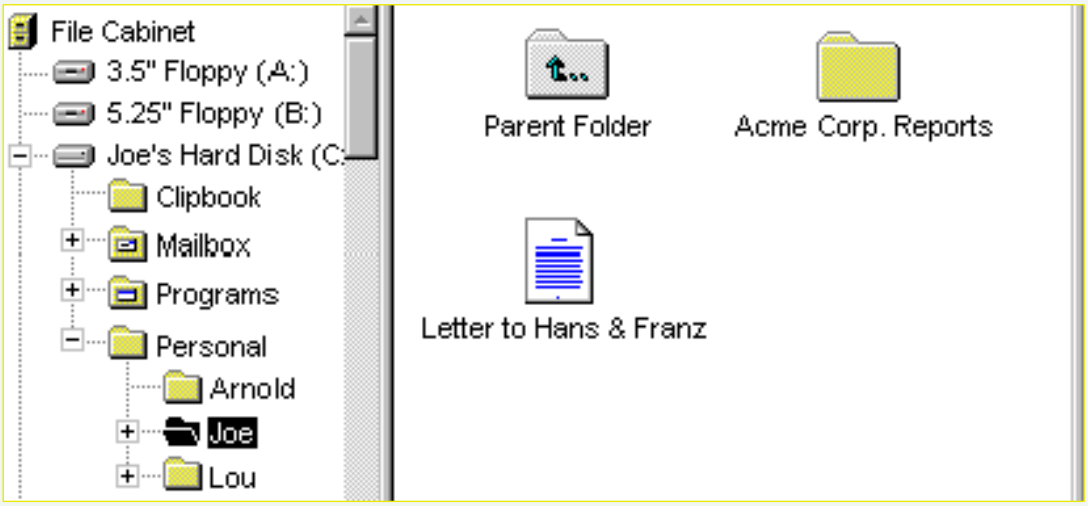
Very tight schedule - a traditional "waterfall" model leaves testing until near the end - not usable

Iterative design was opted for - previous examples of its use already existed elsewhere in the company

A First Attempt



Within File Cabinet



Early Results

Early results from iterative design showed that a baseline from Windows 3.1 was required

User testing provided data on frequently repeated tasks in Windows 3.1

Comparisons of Windows 3.1 showed that the early Windows 95 prototypes were different, not better

Consistency with Windows 3.1 was over-focused on and the team needed to adopt a whole different approach

Specification Process

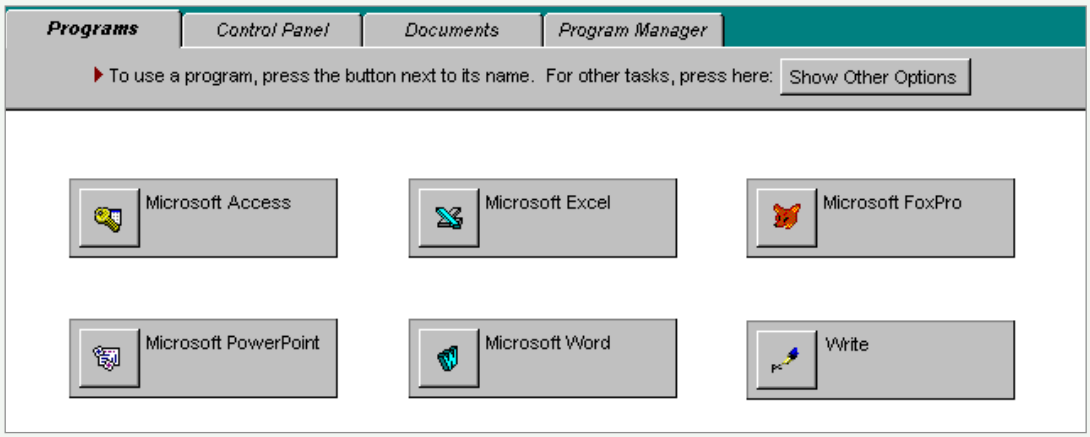
Results and improvements were coming in thick and fast. A major decision loomed:

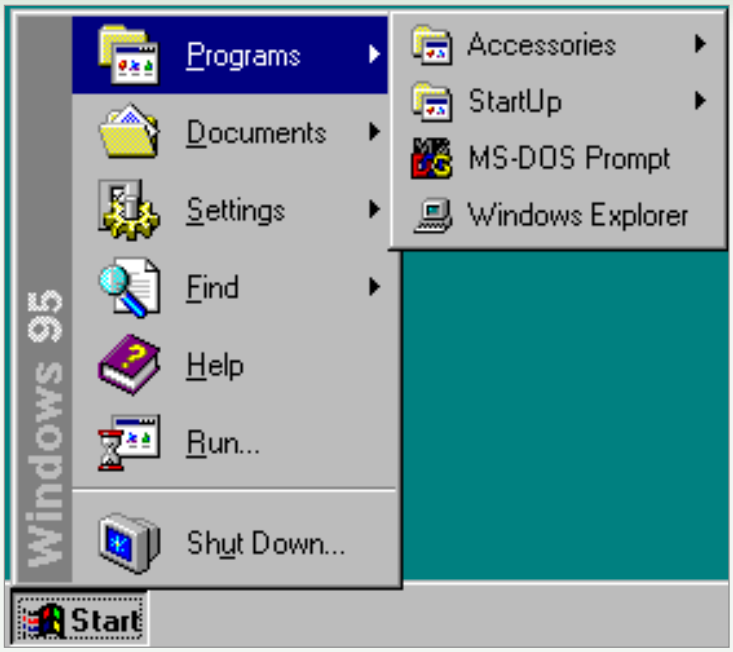
Spend weeks updating the specification and lose valuable iterating time, or stop updating the spec and let the prototypes serve as the spec

The latter approach was taken, and this required clear information to everyone:

- Regular staff meetings for the design team, to keep informed
- Usability test schedules/results regularly broadcasted
- Regular design presentations for outsiders
- Formal tracking of all usability issues, using a database

More Iterations





How to see minimized windows?

Testing and Problem Tracking

64 phases of lab testing, using 560 subjects

699 different comments on usability, 148 positive

Identification

- Level 1: Unable to continue with task (15%)
- Level 2: Considerable difficulty completing task (43%)
- Level 3: Minor difficulty completing task (42%)

Problem Resolution

1. Addressed (fixed and tested successfully)
2. Planned (fix designed, implementation awaits)
3. Undecided (not sure if fix desirable/feasible)
4. Somewhat (fix designed, tested, issues remain)
5. Not addressed (fix not going to happen)

81% of problems were addressed (most not addressed were due to technical or schedule limitations)

Review

Iterative Design:

- No detail of the initial design survived in the final product
- The scope and volume of changes were not envisaged at the start of the project
- Iterative design and constant testing allowed many different solutions to problems to be explored rapidly
- Designers felt rushed towards the end and would have liked to keep iterating and testing and improving

Specification Process

- The "prototype/code is the spec" approach worked well
- Practical organizational details refined

Review

Usability Testing

- This was crucial in "making the pieces fit"
- Good changes to wording in the interface and the Help documents were suggested by testers

Problem Tracking

- The tracking database was crucial to make sure issues didn' t get pushed aside
- It was important that the team believed they weren' t going to get it right first time and indeed not getting it right was very useful
- Unaddressed issues were rolled over into a new database for the next version of Windows