Thesis Proposal

Wendy E. Mackay

17 March 1989

Research Issues

How can a user modify software to meet his or her own needs?

How can the system tailor itself to adapt to perceived user needs?

Why is this hard?

Assumptions about what is possible may be hard to detect. (Implicit not explicit knowledge)

Assumptions may be hard to change.

Users may be unable to articulate needs in a way that maps to potential solutions.

Intuitions

Observe difference between users' perception of their own behavior and their actual behavior.

Provide feedback to users about their behavior.

Ask users to predict their own behavior with a new system and compare to actual use.

Observe changes in use over time, particularly in response to crises.

Software: What’s Different About It?

Error Measurement
    "Mean time between Failures" vs. Bug counts

Interaction among different parts of the system
    New operating system can cause software to fail.

Manufacturing
    Incremental cost for different features has no appreciable effect in manufacturing or distribution costs. (A tape is a tape, a floppy is a floppy.)

Programmers
    Can have order of magnitude differences in programmer ability.

Tools affect solutions
    Different programming languages and operating systems have different assumptions

User Interface
    Orthogonal to system development issues

Language

It provides a language for expressing ideas

Interaction

The interaction seems fundamentally different than with a machine.

Making Assumptions Explicit

Experimental method: *look* for them.

User descriptions: to someone with a
different set of assumptions.

Comparisions: systematic comparison of
competing assumptions.

User feedback loops: user challenges
own assumptions.

Accidents: what happens?

New assumptions: present them and see
what happens.

Proposed Definitions

Adaptive Software:

Software changes itself in response to user behavior.

Tailorable Software:

User selects values of pre-determined parameters to meet user characteristics.

Modifiable Software:

User changes software in response to own needs.

Stories: Lucy

Experienced email user with existing, not completely successful sorting strategy.

Described Lens: Lucy thought about her mail differently.

Lucy predicted how she would use it (and how she would not.

"Worked fine" for several months; not a big help.

Crisis point: went on a trip; had to make new rules. Made two and didn’t have time to do the rest.

Discovered that these rules worked much better and solved real problems for her. Kept the rules.

Lessons?

Lucy couldn’t predict how her use would change.

I couldn’t predict it either.

Once the change was made, others could use the same strategy.

Can artifically set up "crisis" situations and see what happens.

Stories: Rulesets

Orginial assumption: rules are for pre-sorting (prioritizing) mail.

One ruleset always fires before user reads mail.

Many users refused to use it.

One user discovered he could run rulesets after.

He used it for sorting already seen messages.

My assumption: Two kinds of users...

Wrong: New way of adding semi-implicit information about user’s work context.

Learning-by-doing

Why is it critical?

Can we enhance its effectiveness?
(ie influence what is learned?)

Can we predict areas that will change?
(if not precise changes)

Can we change the design of the software
to enhance modifiability?

Repair Theory

Perception that a system is easy to use
is based on ability to repair breakdowns
rather than lack of breakdowns.

Stories: Spreadsheets

Designed to automate filling out columns of numbers in budgets.

Used for anything with columns, numbers or text.

Changed organizational structure.

Changed kinds of financial analysis.

Introduced new kinds of errors.

A Networked Community of People and Computers

Software changes what people do and who does it

People change what the software is and does

Together: co-adaptive communication networks

Co-Adaptive Communication Networks

Co-Evolution:

Living organisms change the environment
The environment changes living organisms

Co-Adaptation:

People change the communication network
The communication network changes people

Evolution implies change across generations

Learning implies change in an individual

Adaptation can refer to both

Co-Adaptive Information Networks

In practice:

- Software is never finished
- Information needs change
- People never stop learning
- People learn best in context

Build this into the design:

- Build software that adapts to users
- Help users adapt to the software

Co-Adaptive Information Networks

- Capture information:
  - What an individual learns
  - What a group learns
  - What the system discovers
- Embody information:
  - Generalize iterative design techniques
  - Adaptive software
  - Communication among people
- Access information:
  - Wide bandwidth under user control
  - Use context to help determine information needs
  - Accommodate changes in needs over time

Why do info needs change over time?

Changes in people’s:

- work activities
- interests
- knowledge
- experience

Changes in the external environment

Time scale ranges from minutes to years