HCI Evaluation I : Qualitative Techniques:

Tags

What is HCI Evaluation:

- Evaluation is a crucial part of the user-centred development process
- This can involve two parts:
    - Think Aloud Evaluation
    - Heuristic Evaluation

Why is evaluation important ?

- “Iterative design, with its repeating cycle of design and testing is only validated methodology in existence that will consistently produce successful results” - Bruce Tognazzini

Think Aloud Evaluation

- Users are asked to verbalise what they are thinking whilst completing task that have been assigned to them
- Think Aloud Evaluation provides insights into the user experiance of usign your software
- It can identify issues wwith teh software
- Think aloud evaluation is part of the software development process which allows you to itterativly improve software

Benifits of Think Aloud

- Cheap
- Relatively easy
- Can be carried out with low numbers of participants

Drawbacks

- Subjective experiance
- Sociable Deseriability bias - Users will tell you what they think you want to hear
- Unusual experiance for users

Planning a think aloud evaluation

- Decide questions you want your study to answer
- Wirte downs tasks you want the user to complete while using your software
- Decide how many pariticipants you want to recruit and how longyou want the sessions to last
- You have one facilitator who provides basic information to tell the faciliate the user using the software
- You also have two observers which record information about the game

Analysis of think aloud evaluation

- Organise the notes you have made into discrete catagories. e.g. useful features (what participants liked), broken features / bugs (what the pariticipants didn't like), and what the user would like (additional features)
- Count the number of times usest comment about different catagories to identify the biggest issues.

Heuristic evaluation

Heuristic evaluation is a method for identifying problems with a product by testing them against a set of guidlines (called heuristics)

How it works

- **Select Heuristics:** First, the evaluators choose a set heuristics. These are generally accepted guidlines that describe common properties of usable interfaces based on Jakob Nielsen's 10 heuristics for user interface design.
- **Recruit Experts:** Generally 3-5 usability **experts** are recruited. The evaluators use **analytical** evaluation (because the experts have knowledg of the heuistics to test the software)
- **Evaluation Session:** Each evaluator goes through the interface several times, first to get familiar with it and then examine it more thoroughly against each heuritcs
- **List Usability Problems:** As the experts examine the interface they note dows the issues that violate the herutistics, indicatin the potential problem and which heuristic it violates.
- **Severity Rating:** Each identified issue is rated in terms of severtiy based on the impact to user experience

Benefit of Heuristic Evaluation

- Cheap

- Relatively easy to carry out (experts need to be familiar with the heuristics)
- **Instant gratification -** list of problems are are available immediately

10 Principles of Heuristics:

1. **Visibility of System Status (feedback):** The design should always keep users informed about what is going on, through appropriate feedback within reasonable time.

- This means that the user should always have feedback on an action e.g. if you click on a object that loads a new screen then either there needs to be a loading bar telling the user that the screen is loading or an error message display
- Do not show blank screens
- do not show static load or progress messages

example:

Password strength. When you put in a password for a website, some times websites will show you dynamically (through color change) that a password is either strong or weak. e.g when a strong password is entered there will be a green tick, if a weak password is entered there might be a red tick.

2. **Match between System and Real World (metaphor):** The system should follow real world conventions in its language i.e. use words / concepts / phrases familiar to the average user

- This means that your software should follow existing conventions. e.g. normally the right arrow on a keyboard moves the player to the right, it would not be following real world convention if your software had the right key moving the player left

examples:

In iTunes all of the users media, the music, movies, podcasts, audiobooks etc are contained within a “Library” which is how the user would normally think of when organising real world physical media

3. **User Control and freedom (navigation):** Users often choose system functions by mistake and will need a clearly marked “emergency exist” to leave the unwanted state without having to go through an extended process. Support undo and redo options

- This means have the ability to ‘go back’, e.g. when you go to a webpage there will always certainly be an option to go back to the homepage

4. **Consistency and standards:** Users should not have to wonder whether different words, situations of action mean the same thing. Follow platform and industry conventions

example:

Gmail: Designed the interface with like ‘inbox’, ‘spam’, ‘Drafts’ etc. these were industry standards and users knew what they meant.

5. **Error Prevention:** Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before commit to the action.

example:

When you want to cancel your subscription on Netflix, you have to go through a series of yes and no questions to get to the final step where you can cancel your subscription. And once you get there, the primary button which draws your attention is the “Cancel” button and the secondary button “Yes, I am sure cancel my subscription” less visually astethic and plain, preventing users from accidentally cancelling their memberships.

6. **Recognition rather then recall :** Minimise the users memory load. Make objects, actions, and options and visible. Instructions for use of the system should be visible or easily retrievable.

example:

- Many IDEs auto complete the syntax you are using.
- In a game instructions for using power ups can appear whenever the user is next to a power up, if there are loads of power ups, having visual instructions

that come on and off can mean the user does not have to remember every mechanic of the game.

7. **Flexibility and efficiency of use:** Accelerators — unseen by the novice user — may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

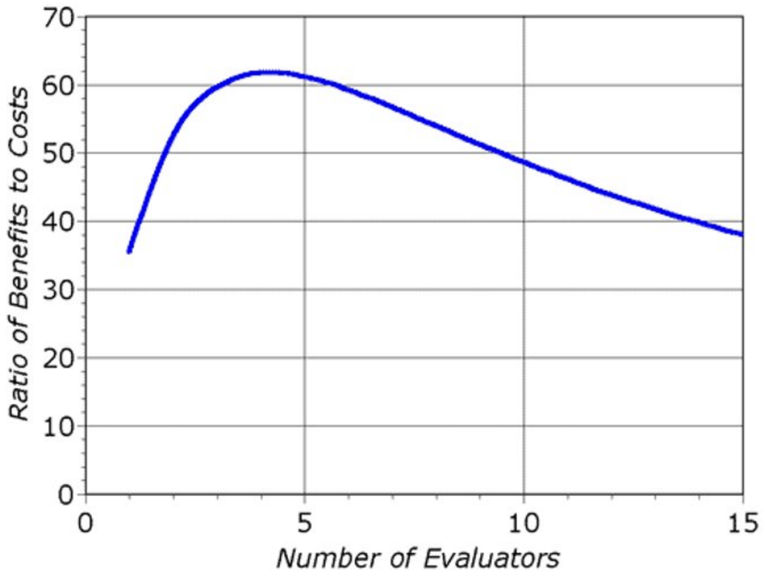
example: Shortcuts on keyboards

8. **Aesthetic and minimalistic design:** Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility
9. **Help users recognise, diagnose and recover from errors**

example: When you fill in a from and something is missing you are told that there is an error.

10. **Help and Documentation:** Even though it is better for software can be used without documentation, however sometimes this may be necessary

- **You need about 4 / 5 evaluators to get a good report of your software, use the reference to this graph to show by you need only 4 / 5:**


Ratio of Benefits to Costs
Number of Evaluators
0
10
20
30
40
50
60
70
0
5
10
15