Stichwörter Empirische Forschungsmethoden – Software Evaluation

Drei englische Texte

LANDAUER

Behavioral Research Methods in Human – Computer Interaction

Ziel ist es die Benutzungsqualität zu verbessern

Die meisten Bedienschnittstellen haben alleine im Schnitt ca. 40 fehler

Oftmals werden Usability Studien sogar unterbunden

Problem des Egocentric Bias => Fehleinschätzung von dem was für andere gut ist

Aufgaben:

1. Evaluation oder vergleich bestehender Systeme
2. Entwicklung neuer Systeme oder Features
3. Die relevanten wissenschaftlichen Prinzipien erforschen
4. Guidelines und Standards entwickeln

Hauptaufgabe: Gute Messmethoden finden und Datenerfassungsmethoden entwickeln die einem bei der differenzierten Betrachtung weiterhelfen

Was soll ein System tun können ?
Nicht alles was technisch möglich ist ist auch sinnvoll, man sollte sich auf die Probleme der User konzentrieren.

Relevante Informationen sammeln, wie, welche Probleme bei bestimmten Kognitiven oder sozialen Aufgaben entstehen. Hilft den Designer sehr viel Weiter !

Die Entwicklung expliziter Standards ist auch sehr wichtig.

Spezielle Probleme bei der Untersuchung:
Intentionen sind meist falsch !
Designer finden ihre Produkte oft einfach und verständlich, die Benutzer hingegen nicht!

Wie jemand die Welt wahrnimmt hängt stark von dem ab was dieser bereits weiss.

Benutzer mit gewisser Expertise können nicht mehr über evtl. Probleme ihrer benutzten Software sagen

Features dürfen nicht isoliert betrachtet werden, sondern immer nur im realen Umfeld !

Oft entscheided der Markt darüber was gut ist und was nicht, noch bevor irgendwelche Studien kommen.

Bias

Problem: Wenn man ein bestimmtes Ergebnis erwartet, neigt man dazu in diese Richtung zu interpretieren

Bei der Vorstellung eines neuen Systems neigen die User dazu dem Autor positiv entgegenzutreten, das heisst das Produkt pos. Überzubewerten, denn man will ihm gutes tun.

Viele Leute neigen dazu neue Technologien prinzipiell als „toll“ zu interpretieren unabhängig vom Nutzen.

Enthusiasmus des Autors reisst oft mit. Weshalb auch oft die Einführung neuer Lehrmethoden mit der ersten Versuchsgruppe, von zB Kindern und Freunden der Kinder des Erfinder, von Erfolg gekrönt sind, bei der weiteren Verwendung aber nicht mehr funktionieren

Untersuchung

Problem bei Untersuchungen: Oft wird nur „First Use“ getestet, zB eine ½ Stunde. Bei längerer Benutzung zeigt sich das Programm aber als unpraktisch.

- ⇨ Soviele wie mögliche Variationen ausprobieren um möglichst viele Design-Fehler zu finden.

Oft sind neue Systeme schlechter als die altgedienten.

Teilweise erweisen sich Systeme als schlechter wie die bisherigen im Fehlerfall (zB Barcode)

Wie messen ?

1. Tastatur mitloggen
2. Representative Stichprobe an Versuchspersonen ermitteln
3. Möglichst viel Hintergrundinformationen einholen
4. Bisherige Studien beachten und deren Fehler vermeiden
5. nicht störende Beobachtung der teilnehmer (weg. Reaktivität)
6. Die Teilnehmer nach zusätzlich ideen und Informationen befragen
7. Mehrere Fragen in einer Studie kombinieren (wg. Interaktionseffekten)

Human-problem oriented invention

1. Ziele der Benutzer herausfinden mit denen sie Probleme haben
2. herausfinden warum Sie damit probleme haben
3. Finde einen Weg die Schwierigkeit zu meistern
4. Einbinden in das System

Fehler Analyse:
Herausfinden wo Leute langsam sind oder Probleme haben
Analyse immer mit echten Benutzern und keinen Experten die ggf Workarounds nutzen

Treten Individuelle probleme bei unfähigeren Leuten auf kann man diese meist beseitigen ohne die fähigeren Leute mit der Änderung zu beeinflussen

Ein Ziel ist es Komponenten zu finden für die zu hohe Fähigkeiten nötig sind, die vereinfacht werden könnten.

Zeitanalyse:

Wieviel Zeit für einzelne Tasks gebraucht wird. Also für:

- ⇨ Training
- ⇨ Manual lesen
- ⇨ Geschwindigkeit einzelner features
- ⇨ Gemachte Fehler beheben

Full Scale Evaluation

Testen nicht nur 1h sondern zB ein ganzes Arbeitsjahr. Problem: Experten nutzen möglicherweise auf Sie zugeschnittene Software, den eigenen Fähigkeiten angepasst

Manche Features sind möglicherweis sehr gut,. Aber werden nicht benutz => Redesign !!

Iterative Testen !

Immerwieder testen während der Entwicklung !

Sehr gut ist Beobachten, zählen der gemachten Fehler, Zeitmessungen…aber wie viel Beobachtung ist notwendig?

Poweranalyse
Wie in Beller beschrieben (?)

Qualität der Daten
Fehler können „willkommen“ sein, das heisst man sieht sie nicht als fehler wohingegen ungewöhnliches schneller als Fehler wahrgenommen wird. Fehler gibt es immer !!!

Reliabilität
Ggf Messmethoden verbessern

Statistische Analyse

- ⇨ Zusammenfasung der wichtigen Daten
- ⇨ Effektgröße bestimmen
- ⇨ Datenmuster erkennen
- ⇨ In ein quantitaives Modell pressen

Karat

User-Centered Software Evaluation Methodologies

Evaluationen haben alle ein Ziel

Es ist unmöglich alle Aspekte mtienzubeziehen

Evaluationen können Subjektiv (zB Erfahrungen eines Einzelnen) oder Objetkiv (d.h. empirisch gemessen) sein.

Evaluation kann Zustimmung erfahren oder Ablehung und sie kann auch Verbesserungsvorschläge hervorbringen

Bei Evaluationsstudien ist es oft leider nicht einfach nur Messen (mit Lineal etc.) sondern einschätzungen. Was zB bedeutet „die Oberflache ist schön". Oft folgen kurzen Statements längere Erklärungen, wie zB bei Filmkritiken („Ich mag den Film, weil…..")

Messemthoden sind schwer zu finden, man kann oft schon froh sein eine Methode gefunden zu haben die wenigstens einigermasse mit dem zu Messenden korreliert. Es muss auch immer das zu messende Ziel vor Augen gehalten werden

Evaluating Software for Usability

Es gibt viele Möglichkeiten ein Softwaresystem zu evaluieren !

Besonders wichtig ist die Usability von Software, bestimmt durch das Design

Def von Usability

Usability von Software ist der Umfang in dem ein Produkt von einem speziellen Benutzer genutzt werden kann um spezifische Ziele, effektiv und zu vollster Zufriedenhet in spezifischem Kontext zu erreichen.

Effektivität, Effizienz und Zufriedenheit zu messen ist recht Zeitaufwendig

Warum Usability evaluieren ?

Oftmals kein Geld für Redesign vorhanden, nachdem evaluiert wurde

Folgende Dimensionen sind wichtig bei der Evaluation:

- ⇨ Was will man überhaupt beabsichtigen mit der Evaluation
- ⇨ Wer macht die Evaluation
- ⇨ Welche Informationen werden gesammelt
- ⇨ Wer wird untersucht
- ⇨ Wieviele Ressourcen stehen zur Verfügung (Zeit und Geld)

Wichtig ist die Evaluation des gesamten Systems in dem die Software genutzt wird

Ergebnis der Studie sollte sein, wie leicht oder schwer bestimmt Aufgaben zu erledigen sind

Informationsquellen

- ⇨ Direkt von den Benutzern
- ⇨ Von potentiellen Usern oder usability Experten die eine Inspektion oder einen Durchlauf (Walktrough) durch das System machen

Dabei gibt es user based Information durch
- ⇨ mündliche Reports
- ⇨ Fragebögen
- ⇨ Aufgezeichnete Daten

Informatonen von Experten Teams durch
- ⇨ Walktroughs
- ⇨ Heuristic reviews

Informationen aus der Theory
- ⇨ Durch theoretische Analyse (GOMS)

Benutzer zentrierte Evaluation

Informationen über das Interface zB über mündliche Reports, walktroughs oder direktem Test

Das Verhalten der Benutze einfach loggen ist ein sehr elementarer Level der Untersuchung, aber dennoch sehr sinnvoll

Leider findet man die Fehler oft erst in fertigen Systemen nach extensiver Nutzung

Mündliche Reports und Lautes denken

Grundsätzlich sollte immer ein gesundes Misstrauen dabei sein, wenn es darum geht was Leute sagen. Mit enstprechender Vorsicht eine excellente Methode für Informationen.

Die Informationen sollten direkt während der Anwendung der software am besten durch lautes denken kommen.

Rückblickende Erzählungen sind immer sehr suspekt, denn es ist schwer im nachhineinzu sagen was man in einer bestimmten Situatuin gedacht hat.

Am besten alles Aufzeichnen, sowohl ein Keyboardlogging als auch das was laut gedacht wird.

Es ist nicht für jeden leicht laut zu denken, ein guter Einstieg kann dadurch erfolgen laut zu denken wie viele Fenster das eigene Haus hat. Damit kann man im erklären dass mann im prinzip die Infos eines Walkthroughs durch das Haus alles hören will. Er weiss dann was wichtig ist.

Dennoch sollte man die Benutzer auch während der Untersuchung immer wieder daran erinnern laut zu denken wenn sie dies eisntellen.

Die Daten die durch das laut denken kommen sind nicht konstant, bei Automatisierten Prozessen oder Zugriffen auf das Langzeitgedächtnis, kommt sehr wenig „Output“ beim laut denken.

Obwohl die formalen Theorien (GOMS) auf dem Vormarsch sind, sind die Reports sehr erfolgreich

Umfragen und Fragebögen

Der Fragebogen muss letztendlich die Frage der Einfachheit der Benutzung eines Systems beantworten.

Mehr Spezifische Fragen als generelle Fragen sind besser. Ausserdem lieber Fragen stellen die das aktuelle System betreffen und keine hypothetischen Fragen über mögliche Veränderungen. Also ganz konkrete Fragen stellen.

Use Data Collection (Durchführung)

- ⇨ Es muss erst einmal eine Frage im Raum stehen
- ⇨ Art und weise diese Frage zu stellen
- ⇨ Wenn die Frage klar definiert ist können die empirischen Daten dazu designed werden
- ⇨ Versuchsplan
- ⇨ Representative Personen finden und Daten sammeln

Die Informationen die man erhält können viel verlässlicher sein wie die eines mündlichen Reports.

„Testing in the Field“ ist sehr kosten und Zeit effektiv

Design Walkthroughs

Man benötigt Paipier und Bleistift und im Extremfall nur eine Attrappe (oder Prototyp) des Systems: Das Untersuchungsteam sollte bestehen aus:

- ⇨ End Usern
- ⇨ Programmierer
- ⇨ Oberflächendesigner
- ⇨ Usability Engeneers
- ⇨ Andere Teilnehmer des Entwicklerteams

Alle relevanten Teile des Systems müssen im Walkthrough abgedeckt werden. Notwendige Redesigns könne sofort an der Attrappe geändert werden und erneut getestet werden.

Der Walktrough sollte möglichst real sein, also soviel wie möglich Benutungstypische Sachen testen.

Leider fehlen bei den Walkthroughs die nötigen Guidelines um es zum Common Sense zu machen.

Aber es ist eine gute Vorstufe um die Fehler in anderen Studien gering zu halten. Der Erfolg des Walkthroughs hängt in erheblichem Masse vom guten Gefühl für Software der Teammitglieder

Heuristic Reviews

Wird nicht genauer drauf eingegangen

Theory based Reviews

GOMS

Vergleich Use und Inspection Based Techniques - Wann sollte man was nehmen ?

Usability Testing

- ⇨ bringt mehr Usability Probleme zum Vorschein
- ⇨ schlecht wenn menschliche Faktoren das Ergebnis verfälschen
- ⇨ ungeeignet for Low-Level Design Phase

Inspection Based:

- ⇨ Gefundene Fehler treten möglicherweise im realen Umfeld gar nicht auf.
- ⇨ Es werden nicht alle wichtigen Punkte ermittelt
- ⇨ Schlecht für High Level Design und Komplettuntersuchung
- ⇨ Bringt keine Vorschläge für Redesign

Beide haben

- ⇨ Kosteneffektive Methoden
- ⇨ Gute Akzeptanz der Entwicklerfirmen
- ⇨ Gute Relabilität

TABELLE SEITE 701

Beide Methoden ergänzen sich gut, auf Usability Testing kann man aber nicht verzichten

Kieras

A Guide to GOMS Model usability Evaluation using NGOMSL

Das Goms Modell ersetzt das User Testing nicht !

Aber die formalen Methoden helfen bei schnellem Redesign !

GOMS= GOALS, OPERATORS, METHOS, SELECTION RULES

Methoden
- ⇨ Serie von Schritten
- ⇨ Rufen evtl. Subroutinen für Subgoal auf
- ⇨ Hierarchische Struktur

Selection Rules
- ⇨ Wenn es mehr al seine Methode für ein Ziel gibt

Einfachstes GOMS Model ist das Keystroke Model
Komplexestes Modell ist das CPM-GOMS

Dazwischen liegt NGOMSL !!!
- ⇨ Lernzeit und Ausführungszeit in einer Programmähnlichen Representation
- ⇨ Natural GOMS Language
- ⇨ Der Algorithmus für das Problem wird so beschrieben dass er ausgeführt werden kann, enweder von einem Analyst oder einem Computer

Stärken und Grenzen den GOMS Models

- ⇨ GOMS fängt nach der Taskanalyse an !
- ⇨ GOMS betrachtet nur die Prozeduralen Aspekt der Usability (Betrachtet zB keine grafischen eidnrücke des Users und auch im High Level Design in Bezug auf das Konzeptuelle Wissen des Benutzers ist es nicht geeignet
- ⇨ GOMS Modelle sind anwendbar und effektiv

Was ist GOMS Task Analyse

Wichtig ist als erstes das Erkennen der Ziele die ein User erreichen will

Was soll man beschreiben und was nicht ?
- ⇨ Mentale Prozesse kann man, da zu komplex, als Black Box definieren
- ⇨ Es ist wichtig zu wissen wenn etwas gelesen wird und warum es gelesen wird, aber nicht wie die involvierten Prozesse Funktionieren, dafür setzen wir Placeholder bzw. Dummies

Ziele können zB sein:
- ⇨ Löschen einer Datei
- ⇨ Verschieben einer Datei
- ⇨ Löschen eines Verzeichnisses
- ⇨ Verschieben eines Verzeichnisses

Diese Ziele könnte man, da sie sehr ähnlich sind ändern in

- ⇨ Löschen eines Objektes
- ⇨ Verschieben eines Objektes

Vergleich von „PC Dos“ und „Macintosh Finder“
PC Dos: 12 Methoden und 68 Schritte
Macintosh Finder: 3 Methoden und 18 Schritte => Sehr Konsistent !!!

NGOMSL
- ⇨ GOMS Beschreibungssprache
- ⇨ Einfach zu lesen, nicht kryptisch
- ⇨ Nahe an einer formalen GOMS Sprache die man auf einem Computer laufen lassen könnte
- ⇨ Programmiersprache des Geistes
- ⇨ Enthält einerseits Daten für „Human Performanz“ und anderseits theoretische ideen in Kognitiver Psychologie

Goals
Etwas das der User erreichen will
zB „Wort löschen“

Operatoren
Etwas das ausgführt wird um ein Ziel zu erreichen

Es gibt externe operatoren (die die man beobachten kann) und mentale Operatoren (nicht beobachtbar, nur hypotetisch erfassbar oder durch theorie)

Primitive und High Level Operatoren
- ⇨ Primitive, wenn ein Operator nicht weiter in eine Sequenz von Lower Level Operatoren aufgeteilt wird
- ⇨ High Level: Wenn er in eine sequenze von Primitiven Operatoren aufgeteilt wird

Standard Primitive externale Operatoren
Z:b: „Home Hand to Mouse“

Standard Primitive Mental Operatoren
- ⇨ “Accomplish goal: <goal description>” entspricht CALL statement
- ⇨ “Return with goal accomplished” entspricht RETURN
- ⇨ “Decide operator” entspricht If-Then bzw If-Then Else
- ⇨ “Goto Step <number> entspricht Goto (sollte aber nicht verwendet warden !
- ⇨ Speicherzugriffe mittel Recall, retain und Forget aus dem Arbeitsgedächtnis
- ⇨ „Mit Retrieve –from-LTM that <LTM-Object-desc>“ aus dem Langezitgedächtnis

zu komplizierte Mentale Prozesse kann der Analyst zB durch Placeholder wie
- ⇨ Get-from-task …
- ⇨ Get-next-edit-location …
- ⇨ Think-of …
- ⇨ Etc.

Beschrieben werden

Methoden

Sequenz von Schritten die ein Ziel erfüllen

```
Method for goal: <goal desc>
     Step1. operator
     Step2. operator
     …
     step n. Return with goal accomplished
```

natürlich kann auch ein Subgoal innerhalb der Sequenz aufgerufen werden !

Selection Rules

Zur Auswahl zwischen verschiedenen Methoden mittel If-Then Else

Eine einfache if then Anweisung nennt man Decide !

Task Beschreibungen und Task Instanzen
Beschreibung: Beschreibt die Aufgabe, also das Ziel und den Weg dorthin
Instanz:. Enthält bereits spezifische Zahlen (zB „Gehe zu Zeile 23“)

NGOMSL Statements
Verschiedene Arten von Statements haben unterschiedliche „Gewichte“ und zählen mehr oder weniger“

Steps, Methods und Selections Rules zB zählen alle „1“

If-Then Else zB als „2“

Ein komplettes Selection Rule Set zählt aber “3”
Eines für das „Select“, eines für das „If-Then“ und eines für das „RETURN“

Judgment Calls
Jenachdem wie der Anaylist sich entscheided wie ein User irgendetwas macht nennt man „Judgement Call“

Diese entscheidunge könne Falsch sein !

Der Analyst kann auch mehrere Fälle entscheiden, d.h. mehrere Entscheidungen ausprobieren

Gemeine Fußangeln

- ⇨ User wissen leider oft selber nicht welche Ziele sie haben
- ⇨ Was diese gerade tun ist oft was anderes als sie denken sie tun gerade (zB E-Mail Adresse in Browser-Adressleiste schreiben um eine E-Mail zu versenden)
- ⇨ User Missbrauchen die System oft auch (Zweckentfremdung)
- ⇨ User nehmen nicht immer die effektivsten Methoden (zB Gesamter Text markieren durch doppelklick am rand, stattdessen markieren Sie alles von hand mit der Mouse. => toll bei 200 Seiten…)

Prozesse die durch Placeholder ausgespart wurden

- ⇨ Kann problematisch sein und in die Irre führen

- ⇨ Man kann am Ende zählen wie oft ein bestimmter Placeholder aufgerufen wurde und wie wichtig dieser damit ist
- ⇨ In der Taskdescription sollte dann das Ergbnis des Placeholders genau drin stehen
- ⇨ Dies nennt man Yellow Pad Heuristic
- ⇨ Wird der Complexe Operator nicht genau spezifiziert wird er bei der Auswertung möglicherweise unterschiedlich bewertet (z.B. 5 statt 3 Ausführungszeiteinheiten beim Zählen)

Wann kann man eine GOMS Analyse machen ?

Nach der Implementation – bei Existierenden Systemen

- ⇨ Am einfachsten
- ⇨ man muss nur noch herausfinden was die intendierten Ziele der programmierer waren und welches die Ziele der User waren
- ⇨ Mit den Usern reden kann helfen aber vorsicht vor den Fußangeln die oben beschrieben waren)

Nach der Design Spezifikation – Während der Entwicklung

- ⇨ Das System muss noch nicht implementiert sind, hauptsache es ist fertig designt
- ⇨ Keine User zum Befragen ! Aber Designer zum Befragen nach den Intentionen
- ⇨ Der Analyst schlüpft in die Rolle des zukünftigen Users

Während des Designs GOMS als Guide für das Design

- ⇨ Die Designer und der Analyst reden darüber wie die Ziele und Methoden sein SOLLTEN

Erstellen eines GOMS Modells

- ⇨ Breadth first !

- ⇨ Schritt A:Die Hauptziele der User heraussuchen

- ⇨ Schritt B: Wiederhole das folgende rekursiv

- ⇨ B1: Entwerfe für jedes Ziel eine Methode um dieses zu erreichen

- ⇨ B2: Überprüfe und revidiere ggf. diese Entwürfe entsprechend der Konsistenz und der Guidelines

- ⇨ B3: Ersetze die High Level Operator ggf. durch Subgoals und erstelle die zugehörigen Methoden

- ⇨ Schritt C: Dokumentiere und Checke die Analyse

- ⇨ Schritt D: Checke die empfindlichkeit (?) der Judgment Calls und Annahmen

Qualitative Aspekte eines GOMS - Modells :

- Natürlichkeit
- Vollständigkeit
- Eindeutigkeit (selektion Rules)
- Konsistenz
- Effizienz

geschätzte Ausführungszeit :

statements + ext. O. + ment. O. + Antwortzeit

Verbesserung des Systems :

- häufige Ziele ...
- Lernzeit reduzieren
- evt. elimin. v. Alternativmethoden
- elemin. Notwendigkeit LZG zu benutzen
- Eliminierung von Arbeitsgedächtnisproblemen
- Highlevelbypassing - Operatoren eliminieren
- Methoden verkürzen