Heuristische Evaluation von ColecoVision-Entwicklungswerkzeugen

Philipp Klaus Krause

8. Juli 2007

Inhaltsverzeichnis

1 Einführung **1**

2 Heuristische Evaluation **2**
2.1 Überblick 2
2.2 Phasen 3
2.3 Schwere 3
2.4 Heuristiken 3

3 Der Untersuchungsgegenstand **4**
3.1 Überblick 4
3.2 Umfang der Evaluation 5

4 Durchführung der Evaluation **5**
4.1 Vorgehen 5

5 Ergebnisse **5**
5.1 Zusammenfassung 5
5.2 Gefundene Probleme 6

6 Ausblick **10**

1 Einführung

Diese Arbeit behandelt eine heuristische Evaluation von ColecoVision-Entwicklungswerkzeugen. Sie enthält eine ausführliche Vorstellung der Methode, einen kurzen Überblick über den Untersuchungsgegenstand, sowie die Ergebnisse der Evaluation, wobei dort auch die Methode kritisch gewürdigt wird. Vorab sei bereits erwähnt, daß obwohl die Evaluation nur mit einem einzige Evaluator durchgeführt wurde, nützliche Erkenntnisse gewonnen wurden.

2 Heuristische Evaluation

2.1 Überblick

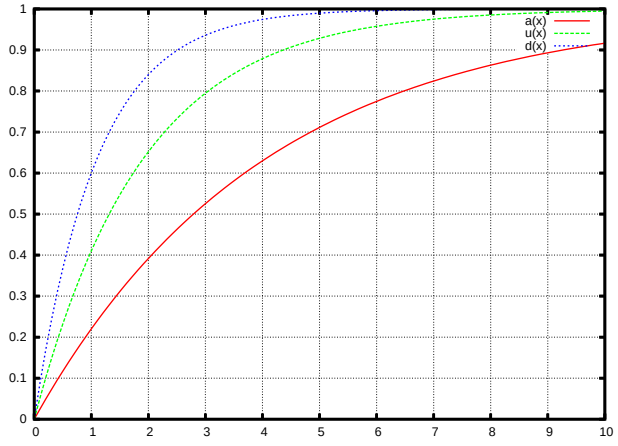
Die heuristische Evaluation ist eine Usability Inspection Method: Im Gegensatz zu Benutzerorientierten Verfahren wird sie von Usability-Experten durchgeführt, Endanwender werden dabei nicht benötigt.

Sie wurde von Rolf Molich und Jakob Nielsen [9] entdeckt.

Heuristische Evaluation ist eine einfache, kostengünstige Methode zur Beurteilung der Benutzerfreundlichkeit von Benutzeroberflächen. Evaluatoren versuchen anhand einer Liste von Heuristiken Probleme zu finden und beurteilen deren Schwere. Abgesehen von der Liste der Heuristiken und den Schweregraden sind die Evaluatoren dabei frei; es gibt aber auch Varianten der heuristischen Evaluation, bei denen die Evaluatoren vorgegebene Aufgaben lösen (heuristischer Walkthrough). Eine Variante bei der die Freiheit der Evaluatoren sehr stark eingeschränkt ist, ist die formale Usability Inspektion, wie sie bei Hewlett-Packard für den internen Gebrauch aus der heuristischen Evaluation entwickelt wurde: Neben den Aufgaben sind auch Benutzerprofile sowie ein festes Benutzermodell vorgegeben, die Evaluatoren arbeiten in einem Team mit fest verteilten Rollen [6].

Die heuristische Evaluation kann zu jedem Zeitpunkt der Entwicklung angewandt werden, es braucht nicht einmal ein Prototyp der zu evaluierenden Anwendung zu existieren. Im Vergleich zu anderen Methoden der Usability Inspection sind Kosten und Aufwand geringer und es werden mehr Fehler gefunden [8].

Heuristische Evaluation findet schwere und leichte Probleme gleichermaßen [4, Seite 56]. Sie findet auch Probleme, die von den Benutzern selbst nicht ohne weiteres erkannt werden können, beispielsweise Wartezeiten, die zu kurz sind, um vom Benutzer als Problem wahrgenommen zu werden, aber dennoch die Produktivität mindern [4, Seite 46].



Die Graphik zeigt den Anteil gefundener Probleme unter der Annahme, daß das Finden von Problemen durch verschiedene Evaluatoren unabhängig ist. Dabei wurden die Wahrscheinlichkeiten für das Auffinden einzelner Fehler durch Anfänger (a), Usability-Experten (u) und Evaluatoren, die sowohl

Usability-Experten als auch Experten im Bereich der Anwendung sind (e) von J.N. übernommen [3].

2.2 Phasen

Jakob Nielsen, der Entdecker der Methode, zerlegt die Heuristische Evaluation in vier Phasen [4, Seite 38] (auch detailliertere Zerlegungen werden empfohlen, beispielsweise durch Jens Heuer [5]).

- Training
- Eigentliche Evaluation
- Nachbesprechung
- Beurteilung der Schwere der gefundenen Probleme

Beim Training werden die Evaluatoren in die Methode der heuristischen Evaluation sowie das Fachgebiet der zu evaluierenden Anwendung eingeführt. In der Nachbesprechung werden die gefundenen Probleme für die nachfolgende Beurteilung der Schwere ausgetauscht; außerdem können hier die Stärken der evaluierten Anwendung angesprochen werden; diese finden ansonsten bei der heuristischen Evaluation keine Berücksichtigung. Ein weiterer wichtiger Aspekt der Nachbesprechung ist die Erarbeitung von Lösungsvorschlägen [4, Seite 42].

2.3 Schwere

Jeder einzelne Evaluator beurteilt die Schwere aller gefundenen Probleme. Nach J.N. [4, Seite 47] hängt diese von Häufigkeit, Auswirkungen und Persistenz der Fehler ab.

Die Beurteilung erfolgt auf einer Skala [4, Seite 49], von 0 bis 4, um die Berechnung von Mittelwerten zu ermöglichen:

- 0 Kein Problem
- 1 Unbedeutend - wird nur behoben wenn sonst nichts zu tun ist.
- 2 Leicht
- 3 Schwer
- 4 Katastrophal - muß vor Einsatz des Produkts behoben werden.

2.4 Heuristiken

1994 überarbeitete J.N. die ursprünglichen Heuristiken [9] aufgrund der Analyse hunderter Usabilityprobleme [4, Seite 30]. Aktuell sind:

- Sichtbarkeit des Systemzustands

- Begriffe aus gewohnter Umgebung des Benutzers
- Benutzerkontrolle und Freiheit
- Konsistenz und Einhalten von Standards
- Fehlervermeidung
- Erkennen statt Erinnern
- Flexibilität und Effizienz
- Ästhetisches, minimalistisches Design
- Fehlermeldung und Fehlertoleranz
- Dokumentation und Hilfefunktion

Allerdings will J.N. dieses Jahr eine nochmals überarbeitete Liste vorstellen [2].

3 Der Untersuchungsgegenstand

3.1 Überblick

Untersucht werden soll eine Auswahl von Entwicklungswerkzeugen für die Videospielkonsole ColecoVision.

ColecoVision ist eine Videospielkonsole, die erstmals im August 1982 von Coleco Industries in den USA auf den Markt gebracht wurde. Für den europäischen Markt wurde ein im wesentlichen identisches Gerät unter dem Namen „CBS ColecoVision“ von CBS Electronics hergestellt. Kompatible Systeme wurden unter den Namen „SpliceVision“ von Splice in Brasilien und „Dina“ von BitCorp bzw. „Personal Arcade“ von Telegames in den USA hergestellt. Technisch war das Gerät den Hauptkonkurrenten Atari 2600 und Intellivision überlegen:

- 3,58 Mhz 8 Bit Z80 Prozessor
- 1 KB RAM
- 16 KB Graphikspeicher
- 15 Farben

Heute gibt es wohl nur noch einige wenige hundert Nutzer der Konsole (nach Daniel Bienvenu wurden von keinem seit den 90er Jahren erschienenen Spiel mehr als 200 Kopien verkauft [1]) . Beginnend mit Kevin Horton 1997 entwickelt eine kleine Schar von Hobbyprogrammierern ColecoVision-Spiele. Soweit diese nicht vollständig in Assembler geschrieben werden, kommt dabei die Programmiersprache C zum Einsatz.

Bisher wurde hierbei meist der HITECH-C-Compiler verwendet, eine neuere Alternative ist sdcc, ein freier C Compiler, der auf 8 Bit-Prozessoren ausgelegt ist [10].

Tabelle 1: Probleme nach Programm

Anzahl	Programm
1	Windows XP
3	Cygwin
0	sdcc
10	libcv/libcvu
3	png2cv
7	Sonstige

3.2 Umfang der Evaluation

Die Evaluation beschränkte sich darauf, die Werkzeuge, mit denen ein Einsteiger in Berührung käme, zu betrachten und auch von diesen nur die für das Erstellen einer einfachen, ersten Anwendung relevanten Teile:

- Cygwin - Unixartige Umgebung für Windows
- sdcc - C compiler
- libcv, libcvu - ColecoVision-spezifische Bibliotheken
- png2cv, png2cvs - Umwandlung von Graphiken in ColecoVision-Formate.

4 Durchführung der Evaluation

4.1 Vorgehen

Auf einem Windows-XP-System wurden die Entwicklungswerkzeuge installiert, kleine Beispielprogramme kompiliert. Die bei der Installation von Cygwin, sdcc, libcv/libcvu, png2cv und png2cvs sowie dem Kompilieren der libcv beiliegenden Beispielprogramme aufgetretenen Probleme wurden den oben aufgeführten Heuristiken von Jakob Nielsen zugeordnet.

5 Ergebnisse

5.1 Zusammenfassung

Es wurden 24 Usability-Probleme gefunden. Erwartungsgemäß handelt es sich hauptsächlich um Probleme in den ColecoVision-spezifischen Werkzeugen. Überraschend war, daß selbst in Cygwin (3 Probleme) und Windows XP (1 Problem) Probleme gefunden wurden, obwohl es sich um vielfach eingesetze Standardsoftware handelt; dagegen wurde kein Problem in sdcc gefunden. Eine ausführlichere Aufschlüsselung der Probleme nach Programmen zeigt Tabelle 1.

Alle aufgefundenen Probleme ließen sich nach Nielsens Heuristiken zuordnen (Tabelle 2). Es wurden allerdings keine Probleme bezüglich der Sichtbarkeit des Systemzustands, der Verwendung von Begriffen aus der gewohnten Umgebung

Tabelle 2: Art der Probleme

Anzahl	Prinzip
3	Konsistenz
3	Fehlervermeidung
2	Flexibilität und Effizienz
1	Minimalismus
2	Fehlertoleranz
13	Dokumentation

Tabelle 3: Schwere der Probleme

Anzahl	Schwere
0	Kein Problem - 0
4	Unbedeutend - 1
13	Leicht - 2
5	Schwer - 3
2	Katastrophal - 4

des Benutzers, Benutzerkontrolle und Freiheit, Erkennen statt Erinnern festgestellt.

Da die Evaluation mit nur einem Evaluator durchgeführt wurde, haben alle Probleme eine ganzzahlige Schwere größer Null (Tabelle 3). Die arithmetische mittlere Schwere beträgt $\approx 2,2$.

Die Methode der heuristischen Evaluation hat sich bewährt. Obwohl nur mit einem einzigen Evaluator durchgeführt, lieferte sie nützliche Erkenntnisse. Der Zeitaufwand war mit 1-2 Stunden für die eigentliche Evaluation gering, es wurde eine Vielzahl von Problemen gefunden. Auch damit, daß kein einzelnes Programm, sondern eine lose Sammlung solcher evaluiert wurde, kam die Methode gut zurecht. Die gewöhnlichen Heuristiken ließen sich erfolgreich auf Programmierwerkzeuge ohne graphische Benutzeroberfläche anwenden.

5.2 Gefundene Probleme

Es folgt eine Liste aller aufgefundenen Probleme; wo nötig sind in den Beschreibungen auch kurze Erläuterungen enthalten.

Der Benutzer weiß nicht, welche Software er installieren muß (Cygwin, sdcc, libcv/libcvu, png2cv, png2cvs).

- Verletztes Prinzip: Dokumentation
- Schwere: 4 (auf der Skala von 0 bis 4)

Der Benutzer weiß nicht, welche Pakete er bei der Installation von Cygwin auswählen muß (gcc-g++, make, sed, binutils, gcc-corem automake, libpng12-devel, bison, flex).

- Verletztes Prinzip: Dokumentation

- Schwere: 4 (auf der Skala von 0 bis 4)

Cygwin ist eine unixoide Umgebung für Windows; Sie ermöglicht es, für Unix gedachte Programme unter Windows zu nutzen. Fast tausend auf Unix-Systemen normalerweise vorhandene Pakete mit Unix-Programmen sind bereits in Cygwin enthalten und können bei der Installation ausgewählt werden. Bei der Auswahl werden diese alphabetisch sortiert oder nach Kategorien geordnet präsentiert. Es gibt keine Suchfunktion.

- Verletztes Prinzip: Flexibilität und Effizienz
- Schwere: 3 (auf der Skala von 0 bis 4)

Fehler beim Download der einzelnen Pakete führen zum Absturz des Cygwin-Installationsprogramms. Dem Benutzer sollte im Fehlerfalle ermöglicht werden, einen anderen Server für den Download auszuwählen.

- Verletztes Prinzip: Fehlermeldung und -toleranz
- Schwere: 3 (auf der Skala von 0 bis 4)

Cygwin hat keine automatische Deinstallation. Um cygwin zu deinstallieren, muß der Benutzer Dateien, Registryeinträge und Umgebungsvariablen an verschiedenen Stellen entfernen.

- Verletztes Prinzip: Flexibilität und Effizienz
- Schwere: 2 (auf der Skala von 0 bis 4)

Der Benutzer weiß eventuell nicht, wie auf unixoiden Systemen Software aus den Quellen, enthalten in sogenannten Tarballs, installiert wird.

- Verletztes Prinzip: Dokumentation
- Schwere: 2 (auf der Skala von 0 bis 4)

Windows ändert beim Speichern der heruntergeladenen Tarballs mit Dateiendungen „.tar.gz“ und „.tar.bz2“ die Dateiendungen auf „.tar.tar“, selbst wenn man manuell im Speichern-Dialog die richtige Endung angibt, wird danach von Windows noch ein „.tar“ angehängt (was dann zu den Dateiendungen „.tar.gz.tar“ und „.tar.bz2.tar“ führt).

- Verletztes Prinzip: Konsistent und Standardkonformität
- Schwere: 2 (auf der Skala von 0 bis 4)

png2cv hat zur Verarbeitung monochromer Graphiken eine redundante „–black“-Option.

- Verletztes Prinzip: Ästhetisches, minimalistisches Design
- Schwere: 1 (auf der Skala von 0 bis 4)

Das png2cv-Archiv entpackt sich in ein Verzeichnis „png2cv-0.10“, während es von den libcv/libcvu-Demoprogrammen in einem Verzeichnis namens „png2cv“ erwartet wird.

- Verletztes Prinzip: Fehlervermeidung
- Schwere: 2 (auf der Skala von 0 bis 4)

libcv/libcvu liegt keine README-Datei bei. Zwar gibt es README-Dateien zu den 3 Demos und die Bibliotheken selbst sind in Kommentaren in den Header-Dateien dokumentiert, aber dies muß der Benutzer selbst herausfinden.

- Verletztes Prinzip: Dokumentation
- Schwere: 2 (auf der Skala von 0 bis 4)

libcv/libcvu kann nicht installiert werden.

- Verletztes Prinzip: Konsistenz und Standardkonformität
- Schwere: 2 (auf der Skala von 0 bis 4)

libcv/libcvu-Demos erwarten, daß png2cv, png2cvs und Komprimierungsprogramme in bestimmten Verzeichnissen liegen.

- Verletztes Prinzip: Fehlervermeidung
- Schwere: 2 (auf der Skala von 0 bis 4)

README-Dateien der libcv/libcvu-Demos werden wegen unterschiedlicher DOS/Unix Zeilenenden nicht korrekt dargestellt.

- Verletztes Prinzip: Dokumentation
- Schwere: 2 (auf der Skala von 0 bis 4)

Die README-Datei der cursor libcv/libcvu-Demo beschreibt das Kompilieren der Demo als wäre png2cv installiert, während alles andere davon ausgeht, daß dieses in einem bestimmten Verzeichnis liegt.

- Verletztes Prinzip: Dokumentation
- Schwere: 2 (auf der Skala von 0 bis 4)

Die README-Datei der cursor libcv/libcvu-Demo beschreibt einen einzugebenden Befehl falsch: Es sind zu viele Minuszeichen angegeben („–“ vor einem Argument, das nicht optional ist).

- Verletztes Prinzip: Dokumentation
- Schwere: 2 (auf der Skala von 0 bis 4)

Die README-Datei der cursor libcv/libcvu-Demo beschreibt bei einem einzugebenden Befehl den Pfad zu den libcv/libcvu Bibliotheken falsch.

- Verletztes Prinzip: Dokumentation
- Schwere: 2 (auf der Skala von 0 bis 4)

Die README-Datei der Komprimierungsprogramme enthält Rechtschreibfehler.

- Verletztes Prinzip: Dokumentation
- Schwere: 1 (auf der Skala von 0 bis 4)

Die Komprimierungsprogramme lassen sich nicht automatisch installieren.

- Verletztes Prinzip: Konsistenz und Standardkonfomität
- Schwere: 2 (auf der Skala von 0 bis 4)

Das Makefile der compression libcv/libcvu-Demo erwartet die Komprimierungsprogramme und png2cv an einem anderen Ort, als dem, an dem sie nach einfachem Entpacken der Tarballs liegen.

- Verletztes Prinzip: Fehlervermeidung
- Schwere: 2 (auf der Skala von 0 bis 4)

Die Dokumentation der Funktionen in „cv_graphics.h“, die Einstellungen des Graphikchips ändern, enthält keine Informationen über die bei Programmstart vorliegenden Einstellungen.

- Verletztes Prinzip: Dokumentation
- Schwere: 2 (auf der Skala von 0 bis 4)

In den Bibliotheksfunktionen dokumentierenden Kommantaren in cv_sound.h finden sich Rechtschreibfehler.

- Verletztes Prinzip: Dokumentation
- Schwere: 1 (auf der Skala von 0 bis 4)

„–black“-Option in png2cv fehlerhaft

- Verletztes Prinzip: Fehlermeldung und -toleranz.
- Schwere: 1 (auf der Skala von 0 bis 4)

cvmtuning-Archiv enthält ausschließlich den Quellcode.

- Verletztes Prinzip: Dokumentation.
- Schwere: 3 (auf der Skala von 0 bis 4)

abc2cvm-Archiv enthält ausschließlich den Quellcode sowie Dokumentation des abc-Musikformats.

- Verletztes Prinzip: Dokumentation.
- Schwere: 3 (auf der Skala von 0 bis 4)

6 Ausblick

Die meisten Probleme sind leicht zu beheben: Eine Überarbeitung der Dokumentation, bei der auch eine README-Datei für libcv/libcvu erstellt wird behebt die Dokumentationsprobleme bis Schwere 3. Die beiden katastrophalen sind durch Erstellung eines Tutorials [7] zu beheben. png2cv, png2cvs, etc sollten nach /usr/local installiert werden, so daß sie wie gewöhnliche Programme aufgerufen werden können. Bei png2cv ist die fehlerhafte und redundante „–black“-Option zu entfernen. Windows XP solte die Dateierweiterungen beim Speichern heruntergeladener Dateien nicht ändern. Das Cygwin-Installationsprogramm sollte um eine Suche erweitert werden und stabiler werden.

Literatur

[1] Daniel Bienvenu. `http://www.geocities.com/newcoleco/collection.txt`.

[2] Jakob Nielsen. Ten Usability Heuristics. `http://www.useit.com/papers/heuristic/heuristic_list.html`.

[3] Jakob Nielsen. Finding usability problems through heuristic evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 373 – 380, 1992.

[4] Jakob Nielsen. Heuristic Evaluation. In *Usability Inspection Methods*. John Wiley & Sons, Inc., 1994.

[5] Jens Heuer. Checkliste „Heuristischer Walkthrough“. `http://www.usability-umsetzen.de/Checkliste%20Heuristischer%20Walkthrough.pdf`.

[6] Michael J. Kahn, Amanda Prail. Formal Usability Inspections. In *Usability Inspection Methods*. John Wiley & Sons, Inc., 1994.

[7] Philipp Klaus Krause. An introduction to ColecoVision development using sdcc. `http://www.colecovision.eu/ColecoVision/development/tutorial0.shtml`.

[8] Robin Jeffris et al. User interface evaluation in the real world: a comparison of four techniques. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 119 – 124, 1991.

[9] Rolf Molich, Jakob Nielsen. Improving a Human-Computer Dialogue. In *Communications of the ACM archive Volume 33 , Issue 3*, pages 338 – 348. ACM Press, 1990.

[10] Sandeep Dutta. Anatomy of a Compiler. *Circuit Cellar*, 121, August 2000.