Bachelorarbeit

Informatik-Ingenieurwesen

Reverse Engineering eines Kaffeevollautomaten

von

Niklas Joachim Eberhard Krüger

Februar 2019

Betreut von

Florian Meyer
Institute für Telematik, Technische Universität Hamburg

Erstprüfer	Prof. Dr. Volker Turau Institute für Telematik Technische Universität Hamburg
Zweitprüfer	Florian Meyer Institute für Telematik Technische Universität Hamburg

Institut für Telematik
Technische Universität Hamburg
TUHH

Eidesstattliche Erklärung

Ich, NIKLAS JOACHIM EBERHARD KRÜGER (Student im Studiengang Informatik-Ingenieurwesen an der Technischen Universität Hamburg, Matr.-Nr. 21491319), versichere an Eides statt, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Hamburg, 22. Februar 2019

Niklas Joachim Eberhard Krüger

Inhaltsverzeichnis

Abkürzungsverzeichnis

EEPROM Electrically Erasable Programmable Read-Only Memory 1
RAM Ramdom-Access Memory 1
UART Universal Asynchronous Receiver Transmitter 4
MCU Microcontroller Unit 4
SPI Serial Peripheral Interface 4

Kapitel 1

Einführung

1.1 Motivation

Heutige Geräte versprechen viel Komfort und eine einfache Handhabung. Über das Internet werden die Geräte zunehmend vernetzt und smart. Diese Arbeit befasst sich mit einer Kaffeemaschine und in erster Linie mit ihrem Speicher. Dieser merkt sich nicht nur Betriebszustände, sondern auch Einstellungen, Zählstände und evtl. vieles mehr. Diese Informationen können nicht nur gelesen, sondern zum Teil auch verändert werden. Moderne Funktionen stecken ohne eine entsprechende graphische Schnittstelle aber bereits in älteren Maschinen, unter der Voraussetzung über das entsprechende Wissen zu verfügen.

1.2 Aufgabenstellung

Anhand eines *Jura Impressa S9 Kaffeevollautomaten* soll der Speicher untersucht werden. Welches Wort / Byte / Bit speichert welche Information? Welche Bedeutung haben diese auf den Betrieb? Hierfür sollen Skripte erstellt werden und der Speicher systematisch untersucht werden.

Wenn die Denkweise der Kaffeemaschine bekannt ist, sollen Werte im Electrically Erasable Programmable Read-Only Memory (EEPROM) abgefragt, aber auch gezielt verändert werden, sowie Statusinformationen aus dem Ramdom-Access Memory (RAM) ausgelesen werden. Die erhaltenen Rohinformationen werden nach den gewonnen Erkenntnissen aufbereitet und stehen so für weitere Projekte zur Verfügung.

Als kleine Demonstrationen entsteht am Ende ein Programm, welches Profile anlegen kann, sodass jeder Nutzer auf Knopfdruck einen Kaffee nach seinen Lieblings Präferenzen zubereitet bekommt.

Kapitel 2

Grundlagen

2.1 Begrifflichkeiten

Der Titel dieser Arbeit lautet `Reverse Engineering eines Kaffeevollautomaten`, aber hinter der Terminologie des Begriffs *Reverse Engineering* steckt ein ganzes Spektrum an Bedeutungen. E. J. Chikofsky und J. H. Cross differenzieren in ihrem Paper [CC90] die Begriffe *Forward Engineering*, *Reverse Engineering*, *Redocumentation*, *Design Recovery*, *Restructuring*, und *Reengineering*.

Zugrunde liegt ein Produkt, welches während seiner Entwicklung mehrere Lebenszyklen durchlaufen hat. Kassem A. Saleh beschreibt in seinem Buch [Sal09] ausführlicher Entwicklungsaktivitäten, wie die Anforderungsanalyse, das Design, die Implementation, die Tests, die Installation und den Einsatz. Während der Implementation nennt er Vorgehensmodelle der Softwareentwicklung, wie das Wasserfallmodell, den Prototypenbau, das Spiralmodell, den Objektorientierten Ansatz, das inkrementelle und iterative Modell, sowie das agile Modell. Jede (Hardware- und) Softwareentwicklung durchläuft dabei, unabhängig vom Modell, die verschiedenen Entwicklungsaktivitäten. Das voranschreiten zur nächsten Stufe, bis zur Fertigstellung des Produkts, stellt das *Forward Engineering* dar.

Der zweite Begriff des *Reverse Engineering* führt in die Gegenrichtung. Aus dem implementierten Produkt wird auf das Design, bzw. aus dem Design auf die Spezifikationen geschlossen. Dabei werden zum einen die Komponenten und ihr Zusammenspiel identifiziert, zum anderen wird aber immer eine höhere Abstraktionsebene, eine vorherige Stufe, rekonstruiert. Ein wichtiges Zitat, welches in Abschnitt 6.3 aufgegriffen wird, lautet: „Reverse engineering in and of itself does not involve changing the subject system or creating a new system based on the reverse-engineered subject system. It is a process of examination, not a process of change or replication.“[CC90]

Redocumentation arbeitet auf einer Ebene und bringt primär eine andere Darstellung. Das Paper nennt „dataflow“, „data structure“ und „control flow“ als Beispiele, die über Werkzeuge wie Diagramm Generatoren, Syntaxhervorhebung und Querverweis Generatoren erzeugt werden können. Das Kernziel sei es die Zusammenhänge und Ablaufpfade hervorzuheben.

Breiter angelegt ist das *Design Recovery*, das für die Designwiederherstellung externe Informationen, Schlussfolgerungen, und Unschärfelogik mit einbezieht, um sinnvolle Abstraktionen auf höherer Ebene zu identifizieren, welche nicht aus dem System selbst hätten gewonnen werden können.

Restructuring umfasst das Nachbauen einer Darstellung innerhalb einer Ebene. Funktionalität und Semantik bleiben erhalten, während die Darstellung umgestaltet wird. Als Beispiel wird die Umstellung von unstrukturiertem Spaghetti Code zu strukturiertem „goto“ freien Code genannt. Der Begriff umfasst aber auch Datenmodelle, Entwurfsmuster und Anforderungsstrukturen. Dabei genügt das Wissen über die Struktur, ohne die Bedeutung dahinter zu verstehen, beispielsweise können „if“ und „case“ Ausdrücke ineinander überführt werden, ohne zu verstehen wann welcher Fall eintritt. Normalerweise werden daher ohne Anpassung der Spezifikation keine Veränderungen vorgenommen. *Restructuring* ist oft eine Form der präventiven Instandhaltung.

Zuletzt wird der Begriff *Reengineering* oder auch „Renovation and Reclamation“ als nachträgliche, neu erstellte Form beschrieben. Dafür geht *Reverse Engineering* zum abstrakten Verständnis dem *Forward Engineering* oder *Restructuring* voraus. Beim *Reengineering* sind Anpassung der Spezifikation und Veränderungen durchaus möglich. Ähnlich zum *Restructuring* verändert das *Reengineering* die unterliegende Struktur ohne die Funktionalität zu beeinträchtigen. Jedoch passiert es selten, dass beim *Reengineering* keine weiteren Funktionalitäten hinzugefügt werden. Damit ist der Begriff *Reengineering* allgemeiner gefasst als das *Restructuring*. Aber *Reengineering* ist kein Überbegriff für *Reverse Engineering* und *Forward Engineering*, nur weil es beides beinhaltet. Beide Disziplinen entwickeln sich unabhängig vom *Reengineering* weiter.

2.2 Stand der Technik

Eine Bachelorarbeit der Universität Magdeburg zum Thema „Reverse-engineering a De'Longhi Coffee Maker to precisely bill Coffee Consumption“[Off18] behandelt eine De'Longhi Caffee Maschine. Das Ziel ist es den Verbrauch exakt zu bestimmen und das System damit um ein Abrechnungssystem zu erweitern. Dafür wird eine Microcontroller Unit (MCU) über das Serial Peripheral Interface (SPI) zwischen Master und Slave Einheiten geschaltet und Informationen aus dem proritären De'Longhi Protokoll ausgelesen. Das Ergebnis der Arbeit ist unter anderem ein Verständnis über das interne Bus Protokoll der Maschine.

Diese Arbeit hingegen nutzt als Ansatz eine gegebene serielle Schnittstelle des Jura Kaffeevollautomaten mit einem ebenfalls propritären Universal Asynchronous Receiver Transmitter (UART) Protokoll.

!!!ToDo!!!

Aprilscherz: Hyper Text Coffee Pot Control Protocol (HTCPCP), RFC2324 -> RFC7168 ???

http://www.rfc-editor.org/info/rfc2324

https://www.rfc-editor.org/rfc/rfc2324.txt

https://www.ietf.org/rfc/rfc2324.txt

https://de.wikipedia.org/wiki/Hyper_Text_Coffee_Pot_Control_Protocol

https://tools.ietf.org/html/rfc7168

Kapitel 3

Hardware und Software

Dieses Kapitel führt die gegebenen und verwendeten Hard- und Software Komponenten in dieser Arbeit auf.

3.1 Der Kaffeevollautomat

3.1.1 Aufbau und Verkabelung

Der „Jura Impressa S9“, siehe Abbildung 3.1, ist ein Kaffeevollautomat mit fünf Kaffeebezugstasten (Spezialkaffee, 1 große Tasse Kaffee, 2 große Tassen Kaffee, 1 kleine Tasse Kaffee und 2 kleinen Tassen Kaffee). Auf der rechten Seite befinden sich Tasten für heißes (Tee-)Wasser und Wasserdampf zum Milch Aufschäumen. Interessanter wird es hinter der linken Wartungsklappe. Dort befindet sich neben mehreren Menü-Tasten auch eine serielle Schnittstelle. Über ein eigenes UART Protokoll, siehe Kapitel 3.2.1, kann hierüber mit der Maschine kommuniziert werden. Ein Arduino Uno übersetzt als „Man in the middle“ die bekannten Kommandos in das Format der Kaffeemaschine. Über eine weitere serielle Verbindung über USB lässt sich der Arduino ansteuern. Aus Sicht des Computers ist der Arduino ein Gerätelaufwerk unter `/dev/ttyACM0` mit einer Baudrate von 9600. Diese Zahl findet sich zu Beginn des Arduino Skrips wieder. !!!ToDo!!! verweis aufs Ardunio Skript, Projekt CoffeeMachine.

3.1.2 Kommandos

Die Tabelle 3.1 zeigt die Befehlsgruppen, sowie ausgewählte Befehle, die der Kaffeevollautomat versteht.

Kommando	Beschreibung	Rückgabewert
Betriebszustand (AN:<id>)		
AN:01	Einschalten	ok:
AN:02	Ausschalten	ok:
AN:03	Display Test	ok:
AN:<id>	u.v.m.	ok:
Bezugstaste (FA:<id>)		
FA:02	Gerät spülen	ok:
FA:0C	Spezialkaffee	ok:
FA:<id>	u.v.m.	ok:
Steuerungskomponenten (FN:<id>)		
FN:<id>	Pumpen, Heizung, u.v.m	ok:
Eingabestatus*		
IC:	Eingaben auslesen*	
Spiele Musik (easter egg)*		
PM:	Play music*	
Speicherzugriff		
RE:<address>	Liest 2 Byte EEPROM Speicher	re:[0000-FFFF]
WE:<address>,<value>	Schreibt 2 Byte in EEPROM	ok:
RT:<address>	Liest eine Zeile EEPROM	rt:[0-F 64x]
RR:<address>	Liest eine Zeile RAM	rr:[0-F 32x]
Sperrung		
?M3	Aktiviert den Inkassomodus	?ok
?M1	Deaktiviert den Inkassomodus	?ok
Display		
?D0	Standard Displaytext zurück setzen	?ok
?D1[A-Z 8-11x]	Displaytext Zeile 1	?ok
?D2[A-Z 8-11x]	Displaytext Zeile 2	?ok
Aktion an der Kaffeemaschine		
	1 kleiner Kaffee (Produkt 1)	?PAE
	2 kleine Kaffees (Produkt 2)	?PAF
	1 großer Kaffee (Produkt 3)	?PAA
	2 große Kaffees (Produkt 4)	?PAB
	Spezialkaffe (Produkt 7)	?PAG
	Dampf (Produkt 6)	?PAI
	1 Dampfportion (Produkt 5)	?PAJ
	1 Tasse Tee (Produkt 8)	?PAK
*** leider nicht alles in der Jura Impressa S9 implementiert, siehe weitere Geräte der S-Reihe**		

Tabelle 3.1: Befehlsübersicht der Jura Kaffeevollautomaten (S-Reihe)

Abbildung 3.1: Jura Impressa S9 – Kaffeevollautomat

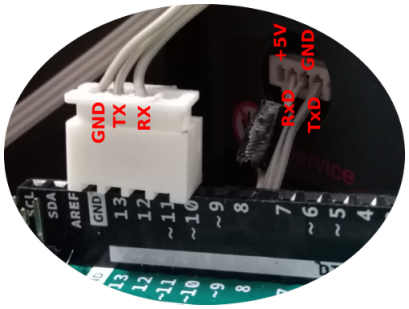

Abbildung 3.2: Pinbelegung: Jura Impressa S9 – Arduino Uno

3.2 Bibliotheken

3.2.1 Serielle Kommunikation

Aus dem Projekt „CoffeeMachine“[1] stammt das Programm des Arduino UNO zum Kodieren der UART Kommandos. Die an den Kaffeevollautomaten adressierten Bytes (ASCII-Zeichen für Zeichen der Befehle aus der Kommando Tabelle 3.1), bestehend aus seinen acht Bits, werden in je vier Bytes aufgeteilt. Die dritte und sechste Stelle der neuen Bytes repräsentieren je zwei Bits des ursprünglichen Bytes. Bei der Übertragung an den Kaffeevollautomaten werden die restlichen Bits mit Nullen aufgefüllt. Abbildung 3.3 veranschaulicht dies an dem ersten Byte des Einschaltbefehls. Die Kodierung der, von dem Kaffeevollautomaten

[1] https://collaborating.tuhh.de/e-17/General/CoffeeMachine

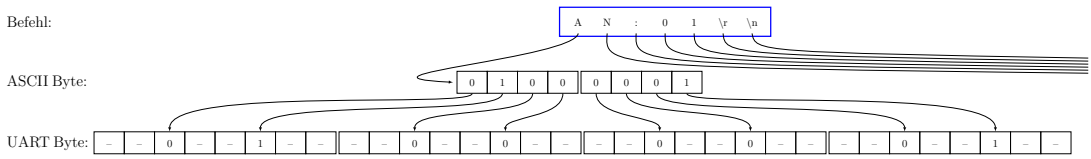

Abbildung 3.3: Umrechnung

kommenden, Bytes erfolgt analog. Laut „Protocoljura“[2] bestehen die irrelevanten Bits der ankommenden Bytes aus einer Null und weiteren fünf Einsen pro Byte.

Vier Bytes kodieren ein ASCII Zeichen und werden fortan als Gruppe bezeichnet. Zwischen jeder Gruppe gibt es eine Verzögerung von 8ms. Diese Verzögerung begrenzt hauptsächlich die Übertragungsgeschwindigkeit, was in Abschnitt 6.1.1 diskutiert wird.

Die jetzt vorhandene Umgebung kann bereits über den seriellen Monitor der Arduino IDE genutzt werden.

libserial

3.2.2 Speicherformat

libjsoncpp

[2] http://protocoljura.wiki-site.com/index.php/Protocol_to_coffeemaker

Kapitel 4

Methodik und Implementierung

4.1 Vorgehen

Die Speicher der Kaffeemaschine, der EEPROM und der RAM, werden Zeilenweise von einem Skript ausgelesen. Dieses Skript setzt die einzelnen Informationen zu einem Speicherauszug zusammen. Aufeinander folgende Speicherauszüge können auf Veränderungen verglichen werden. Die von Hand angestoßenen Veränderungen sollten möglichst elementar sein, um gezielte Aktionen den Wertänderungen bestimmter Speicherzellen zuprdnen zu können. Der dafür nötige Zugriff kann auf zwei Arten erfolgen: direkt am Speicherstein auf der Hauptplanine oder seriell über die vorhandene UART Schnittstelle, siehe 6.1.1. Im Folgenden erfolgt die Kommunikation über die UART Schnittstelle und mithilfe der *libserial*-Library, siehe 6.1.2.

4.2 ...

Kapitel 5

Ergebnisse

5.1 Bedeutung der Speicherstellen

Um den Speicher zu verstehen, klären wir vorab das Rückgabeformat der Speicherauszugs Befehle, sowie den Aufbau des Speichers der Jura Impressa S9. Dieser Kaffevollautomat besitzt einen RAM für die Status, Messwerte und Zwischenberechnungen im Betrieb und einen EEPROM für Einstellungen und Zählstände, die auch nach einer Stromunterbrechung erhalten bleiben.

5.1.1 EEPROM

Der Electrically Erasable Programmable Read-Only Memory (EEPROM) umfasst 512 Bytes. Über den Befehl `RT:<address>`, siehe Tabelle 3.1, lässt sich eine Zeile EEPROM Speicher abfragen. Die Adresse reicht von 0x00 bis 0xF0 in sechzehner Sprüngen. Als Antwort erhält man hinter dem kleingeschriebenen Kommando eine Zeichenkette bestehend aus 64 Hexadezimalzeichen. Da immer zwei Hexadezimalzahlen eine Byte (8 Bit mit Werten von 0-255) repräsentieren umfasst eine EEPROM Zeile 32 Bytes. Über den Befehl `RE:<address>` können direkt Wörter im EEPROM abgefragt werden. Die Adresse reicht von 0x00 bis 0xFF. Die kleinste adressierbare Einheit, das Wort, besteht daher aus zwei Bytes, also vier Hexadezimalzeichen.

!!!ToDo!!! Abbildung aus dem Antrittsvortrag: Seite 8/13

5.1.2 RAM

Der Ramdom-Access Memory (RAM) hingegen umfasst 256 Bytes. Der Lesebefehl für eine RAM Zeile lautet: `RR:<address>`. Die Adresse reicht hier von 0x00 bis 0xFF, sodass hier die kleinste Einheit ein Byte, bestehend aus zwei Hexadezimalzeichen, ist.

!!!ToDo!!! Abbildung aus dem Antrittsvortrag: Seite 9/13

5.2 Aktionsauswirkungen

Reinigung setzt etliche Zähler zurück; Entnahme der Schale setzt min. 2 Trester zähler zurück; Bits im RAM variieren auch im ausgeschalteten Betriebszustand, einige Informationen nur im eingeschalteten Zustand auslesbar, ander aber auch immer (solange Maschine mit Strom versongt wird)

Kapitel 6

Diskussion und Ausblick

6.1 Probleme

6.1.1 Kommunikation mit der Kaffeemaschine

Der direkte Zugang ist wahrscheinlich am schnellsten, im laufenden Betrieb aber schlecht möglich, da die Speicheradressen aktiv abgefragt werden müssten und damit Kurzschlüsse in der Elektronik produziert werden. Die serielle Schnittstelle ermöglicht das Abfragen im laufenden Betrieb, also auch im RAM. Jedoch benötigt das Auslesen seine Zeit (7-9 Sekunden für den RAM und 10-15 Sekunden für den EEPROM). Gerade im RAM gibt es viele Veränderungen im Ruhezustand, sodass man auf diesem Weg nie einen zusammenhängenden Speicherauszug zu einem festen Zeitpunkt auslesen kann.

6.1.2 Serielle Kommunikation

Gerätedatei direkt ansprechen Um aufbauend Skripte ausführen zu können war der erste Versuch den Dateideskriptor `/dev/ttyACM0` sowohl lesend als auch schreibend zu nutzen. Zunächst funktionierte dies sehr gut, bis nach wenigen Tagen kaum reduzierbares Fehlverhalten auftrat. Gerade Antworten der Kaffeemaschine kamen nur noch in Bruchstücken an. Dies wurde bei der Einrichtung bereits im CoffeeMachine-Projekt (1) in der Readme.md für die Ausführung auf einem Raspberry Pi beschrieben.

Versuche dies direkt zu lösen, in dem ein funktionierende Umgebung nachgebaut wurde, schlugen fehl. !!!ToDo!!! Unternommene Befehle und Versuche aufzeigen

C++ und die *libserial*-Library Für eine sichere Verbindung wird in dem C++ Programm auf die im Linux Repository erhältliche Library *libserial* zurückgegriffen. Beim Verbindungsaufbau über die *connect()* Methode der *SerialConnection* Klasse wird die Schnittstelle initialisiert. Hier werden der Gerätepfad, die Baudrate und eine erwartete Mindestlänge festgelegt.

Kommandos gehen zuverlässig an den Arduino raus und ausgelesene Antworten kommen nun im Ganzen an. Der *read* Befehl ist ein blockierender Aufruf, der das Programm anhält bis eine Antwort vorliegt. Dies stellt für die Speicherauslesung keine Problem da, da die ordnungsgemäßen Befehle immer ein `ok:` oder `xx:0-F` Speicherauszug zurück geben.

6.1.3 Unbekannte Speicherorte

Unbekannter Speicherort des Einstellungsmenü-Zählerstands (überlebt Stromtrennung?!) und des Standarddisplaytextes (im RAM?!); dafür Blick auf den Mikrocontroller in der Hardware [Hol17] und externes Auslesen ggf. erforderlich!

6.2 Aussagekraft der Ergebnisse

EEPROM Menüeinstellungen recht sicher, einige partielle Zubereitungszähler aber weniger (Firmware unbekannt)

RAM mehrere Status Bits einigermaßen sicher (vor allem die im ausgeschalteten Betriebszustand); durch viele unregelmäßige Veränderungen wurden aber mehrere Bytes außer acht gelassen, die an bestimmten Bit Positionen wichtige Flags enthalten könnten!

Es fehlen (wichtige?!), im Benutzerhandbuch dokumentierte, Meldungen wie: Gerät verkalkt, Störung 2, Störung 8, sowie etliche Menümeldungen während der internen Programmabläufe.

6.3 Einordnung zur Terminologie des Reverse Engineerings

Diese Arbeit besteht im wesentlichen aus den zwei Teilen, den Speicher zu analysieren und zu verstehen, bzw. das Wissen darüber in einer neuen Oberfläche einfach zur Verfügung zu stellen. Ein kurzer Rückblick in Abschnitt 2.1 erinnert an die Begriffe *Forward Engineering*, *Reverse Engineering*, *Redocumentation*, *Design Recovery*, *Restructuring*, und *Reengineering*.

Für den ersten Teil wurden eine gegebene serielle Schnittstelle, sowie bereits implementierte Befehle ausgenutzt, um den Speicher auszulesen. Viele Aufnahmen, mit kleinen Veränderungen zwischendurch, ließen Rückschlüsse auf die Bedeutung der Werte an bestimmten Speicherstellen zu. Gerade mit den Überlegungen, was während einer Aktion am Kaffeevollautomaten alles im Speicher passiert, lässt sich diese Tätigkeit dem Begriff *Reverse Engineering* zuordnen. Aus dem fertig implementierten und gegebenen System werden Rückschlüsse auf das Design der Maschine gezogen. *Redocumentation* trifft aufgrund fehlender Spezifikationen und Quellcodeauszügen wenig zu. Betrachtet man aber das Benutzerhandbuch und Interneteinträge über Erfahrungen mit der Maschine als externe Informationen, sowie Schlussfolgerungen

und Unschärfelogik über den RAM, kann dieser Part auch als *Design Recovery* über den Speicher angesehen werden.

Mit diesem Wissen des ersten Teils wurde dann im zweiten Teil etwas Neues geschaffen. Das Abfragen der aktuellen Einstellungen und Zustände bietet eine neue Oberfläche, die als *Redocumentation* oder *Restructuring* angesehen werden kann. Für die *Redocumentation* spricht das neue Ausgabeformat auf der Ebene der Implementierung. Ein eigenes Gerät ist zur Alternative (Ausnahmen siehe 6.2) für das kleine Display des Geräts geworden und gibt den aktuellen Zustand in ausführlicherer Weise aus. Ohne eine präventive Instandhaltung beabsichtigt zu haben ist *Restructuring* aber auch zutreffend, da Wissen aus dem ersten Teil hier eingeflossen ist. Wenn die Bedeutung bestimmter Speicherstellen nicht bekannt gewesen wäre, hätten die eingestellten Werte keine Bedeutung gehabt, was für den Begriff *Restructuring* spricht.

Für eine individuelle Zubereitung, wird nur die Datengrundlage kurz zuvor mit einem Impuls von außen verändert, der Kaffee wird dabei nach wie vor von den Standard Abläufen der Kaffeemaschine zubereitet. Die neue Oberfläche kann man als *Restructuring* oder dem allgemeineren *Reengineering* angesehen werden. Da die neue Oberfläche, über die serielle Schnittstelle in Verbindung mit einem eigenen Endgerät, aber sehr von den Bordmitteln der Maschine abweicht passt vielleicht das *Reengineering* etwas besser. Die Funktionalitäten der Maschine sind aber immer noch die Gleichen, das Programmmenü muss nun jedoch nicht mehr betreten werden, um die allgemeinen Geräte Einstellungen für den eigenen Kaffee anzupassen.

Im Detail ist die genau Zuordnung der Begrifflichkeiten immer eine persönliche Auslegung dieser Begriffe. Sehr zutreffend ist das Zitat aus Abschnitt 2.1, denn diese Arbeit hat den Speicher intensiv betrachtet und mittels der Ergebnisse aus dem zweiten Teil um neue Perspektiven bereichert, ohne das Ziel verfolgt zu haben die Maschine zu klonen oder umzubauen.

6.4 Ausblick

Mit unendlicher Zeit und unendlich vielem Geld wäre folgendes noch möglich: evtl. Firmware aus ROM auslesen und disassemblieren -> Vorgänge wirklich verstehen Für folgendes hat die Zeit gefehlt / der Schwerpunkt ging nicht in diese Richtung: Display Texte und Menü-Zählerstand wurden im RAM nicht wiedergefunden, ...

Literaturverzeichnis

[CC90] CHIKOFSKY, E. J. ; CROSS, J. H.: Reverse engineering and design recovery: a taxonomy. In: *IEEE Software* 7 (1990), Jan, Nr. 1, S. 13–17. http://dx.doi.org/10.1109/52.43044. – DOI 10.1109/52.43044. – ISSN 0740–7459 3

[Hol17] HOLTEK SEMICONDUCTOR INC. (Hrsg.): *HT46F46E/HT46F47E/HT46F48E/HT46F49E Cost-Effective A/D Flash MCU with EEPROM*. Rev. 1.50. Hsinchu, Taiwan: Holtek Semiconductor Inc., April 2017. http://www.holtek.com.tw/documents/10179/11842/HT46f4xev150.pdf. – http://www.holtek.com/productdetail/-/vg/46f4xe 16

[Off18] OFF, Fabian: *Reverse-engineering a De'Longhi Coffee Maker to precisely bill Coffee Consumption*, Universität Magdeburg, Diplomarbeit, Februar 2018 4

[Sal09] SALEH, Kassem A.: *Software engineering*. Boca Raton, Fla. : Ross, 2009. – ISBN 1932159940 9781932159943. – Literaturangaben 3

Content of the DVD

In this chapter, you should explain the content of your DVD.