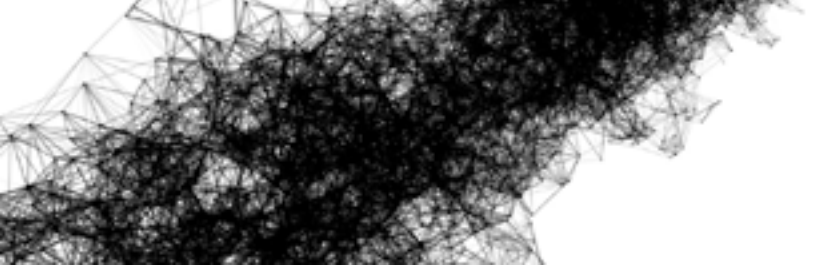
Bachelor Arbeit von
Fabian Morón Zirfas



1 Einleitung

Diese Arbeit setzt sich mit der Automation von Design-Prozessen auseinander. Sie ist an Designer gerichtet, die einen rohen Überblick bekommen wollen, wie Arbeitsprozesse automatisiert werden können und soll als Hilfestellung dienen, einen Einstieg in das Schreiben von Skripten für Grafikanwendungen zu finden. Es ist keine komplette Anleitung zum Erlernen von Programmier-Grundkenntnissen. Diese können für viele verschiedene Sprachen im Internet gefunden werden. Vielmehr ist es eine Auseinandersetzung mit der Fragestellung, was Automation durch Programmierung für Designer leisten kann und welche Bereiche sie nicht abdeckt.
Im ersten Abschnitt soll dem Leser kurz erläutert werden, was er benötigt, um zu Skripten, um dann im zweiten Teil zu analysieren, wann dies sich lohnt. Hier wird an kurzen Beispielen erläutert, wo Probleme auftreten können und wie solche Probleme klug umgangen werden können. Im dritten Teil folgt ein Szenario an dem Schritt für Schritt die Logik und Funktionsweise eines Skriptes erklärt wird, sowie ein weiteres Szenario mit einer einzeiligen und doch nützlichen Lösung. Der vierte Teil behandelt reelle Anwendungsmöglichkeiten. Anhand dieser wird das Ausmaß, das diese annehmen können, gezeigt. Es folgt im fünften Teil ein Versuch die Frage zu beantworten, warum es bei Nicht-Programmierer/innen teils große Berührungsängste mit Computersprachen gibt. Im sechsten Teil, dem Fazit, wird der Erkenntniszuwachs aus all dem zusammengefasst. Am Ende der Arbeit folgt ein Abschitt mit Erläuterungen zur verwendeten Terminologie. Dieser Abschnitt sollte jedoch nicht linear gelesen werden, sondern als Erläuterung dienen.

2 Wann Soll Ich Skripten?

> *„Scripting languages assume that there already exists a collection of useful components written in other languages. Scripting languages aren't intended for writing applications from scratch; they are intended primarily for plugging together components."*
> *Scripting: Higher Level Programming for the 21st Century*
> *by John K. Ousterhout*

Auch wenn Aufgaben auf unterschiedliche Weise mit verschiedenen Programmiersprachen gelöst werden können, haben sich spezielle Anwendungsgebiete für die einzelnen Sprachen ergeben. Ganz unabhängig davon, dass sich in unserem Fall Adobe Anwendungen mit JavaScript ansprechen lassen, macht es Sinn, eine Skriptsprache zu verwenden, um die bereits in höheren Sprachen implementierten Funktionen zu verbinden. JavaScript ist unser „Kleber". Wir können jedoch unseren Arbeitsablauf durch gezielte Befehlsketten von repetitiven Aufgaben befreien. Ich bin mir sicher, dass ein Grossteil aller Gestalter die vorgefertigte Software für ihre Arbeit verwenden, schon an den Punkt kamen, wo sie sich dachten: „Warum kann mein Programm DAS* nicht, es ist doch alles da. Der Knopf und danach diesen Knopf!".

DAS steht hier für eine gewünschte Funktionsweise

„Scripting" erlaubt es uns, diese beiden Knöpfe miteinander zu verbinden. Das bedeutet dann, dass wir unsere Arbeit um einen „Klick" reduziert haben. Wir haben zwei Knöpfe durch Verkettung auf einen neuen Knopf gelegt. Natürlich klingt die Reduktion um einen „Klick" vernachlässigbar. Wenn jedoch diese zwei „Klicks" 100-mal ausgeführt werden müssen und wir durch logische Anweisung diese ebenfalls auf nur einen Knopf zusammenführen können, ist der Zeitgewinn enorm. Ebenfalls muss hier erwähnt werden, dass viele der Probleme, die in einem Gestaltungsprozess auftreten, nicht zum ersten Mal bei eben dieser Person auftreten. Für den Bereich „Scripting" von Adobe-Anwendungen gibt es im Netz viele Seiten und Foren, die sich mit diesem Thema befassen.

Im Bereich JavaScript gibt es noch viele mehr, da JavaScript auch verwendet wird beziehungsweise entwickelt wurde, um Browser zu steuern. Aufgrund dessen ist die Dokumentation mehr als ausgiebig. Es bedarf nur etwas Übung, um die gefundenen Beispiele zu lesen und auf die eigene Problemstellung zu abstrahieren.

Hierbei Sei Zu Beachten!

„Scripting" kann keine Design-Entscheidungen fällen. Es existiert kein Algorithmus, der Ästhetik simuliert.* Um eine spannende Komposition zu schaffen, braucht der Gestalter „nur" drei geometrische Grundformen zu erzeugen und diese im richtigen Verhältnis zu einander anzuordnen. Um dies programmatisch zu lösen, müsste ein Skript mehrere hundert Mal ausgeführt werden. Jedes Mal mit einer kleinen Veränderung der Koordinaten, der oben genannten drei Objekte. Abgesehen davon, dass der Autor irgendwann entscheiden muss, welche Komposition spannend ist. Was das Skript leisten kann, ist anhand von bestimmten Rahmenparametern eine Fülle von Varianten zu liefern, die manuell ausgeführt Sehnenscheidenentzündungen hervorrufen würden. Programmieren ist nicht einfach. Es ist, wie eine neue Sprache zu erlernen. Stellen sie sich vor, sie sind in einem fremden Land, dessen Sprache sie nicht beherrschen. Sie werden zuerst Probleme haben, sich zu verständigen. Dann lernen sie, ihre Grundbedürfnisse zu decken. Ab einem gewissen Punkt können sie Tageszeitungen lesen, Inhalte erfassen und abstrahieren. Eines Tages werden sie feststellen, dass sie in der Sprache träumen. Der Vorteil an Computersprachen im Vergleich zu „Menschensprachen" ist, dass in der Computersprache kein Raum für Interpretation vorhanden ist. Jede Aussage **MUSS**, im Gegensatz zur zwischenmenschlichen Kommunikation, eindeutig sein. In Quellcode ist kein Raum für Interpretation. Die Syntax muss valide sein. Dies ist ein Vorteil, da es Genauigkeit bedingt. Dennoch, eine Sprache lernen, ist keine leichte Aufgabe.

Ein Algorithmus ist ein logische Verkettung von Operationen siehe Abschnitt 7.03.

2.0.5 Wie oft muss diese Tätigkeit ausgeführt werden?

Wenn das Skript nur ein einziges Mal ausgeführt werden soll, sollte man sich fragen, wo darin der Nutzen liegt. Es kann natürlich sein, dass dies Sinn und Zweck hat und sollte nur bedingt ausschlaggebend sein. Hierbei gilt es zu unterscheiden, dass einmalig 1000-mal einen Knopf drücken, bereits eine Hilfe sein kann. Einmalig 1000-mal unterschiedliche Knöpfe drücken, ist eine Aufgabe, die doch besser manuell geschieht.

2.0.6 Lässt sich die Automation auch auf andere, ähnliche Bereiche anwenden oder mit geringem Aufwand abstrahieren?

Wenn dies der Fall ist, steigt der Nutzen der Arbeit. Der einmalige Aufwand ein Programm oder Skript für eine immer wieder kehrende beziehungsweise ähnliche Aufgabe zu schreiben, rentiert sich durch jede weitere Ausführung. Dies soll heißen: Durch das Schreiben des Programms, das ich immer wieder verwende, spare ich Zeit in der Zukunft.

2.0.7 Wie sehr ist sie von Umgebungsvariablen abhängig?

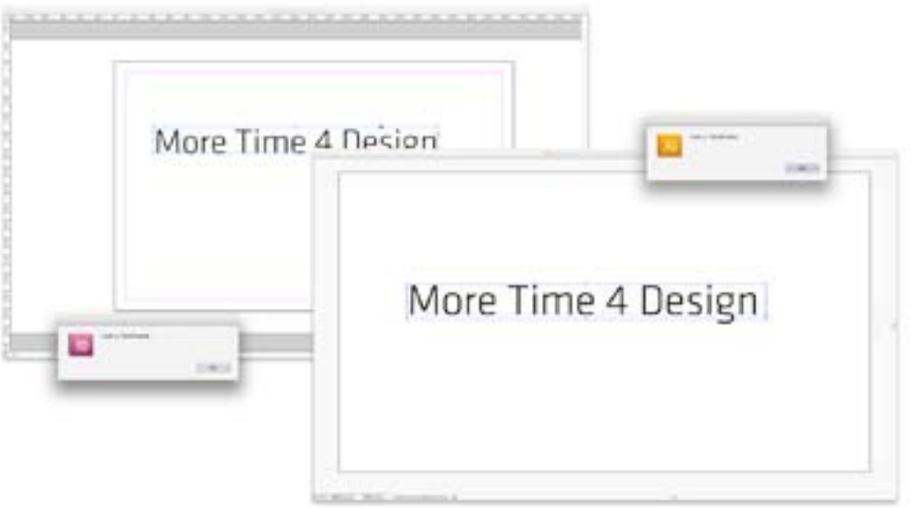
Kann das Skript unabhängig von allen Variablen, die der Benutzer setzen kann, ausgeführt werden, vereinfacht das den Aufwand. Bei einer Abhängigkeit erfordert es immer erst einer Abfrage des „Ist-Status“. Ein kleines Beispiel: In Illustrator oder InDesign wird die aktuelle Auswahl des aktiven Dokuments in einer Liste, genannt „selection“, geführt.

```
app.activeDocument.selection;
```

In dieser Liste liegen einzelne Objekte, die beispielsweise Text, eine Vektor-Form oder ein Bild sein können. All diese haben gemeinsame Eigenschaften, aber auch spezielle. Es muss also, bevor eine Eigenschaft genutzt oder verändert werden kann, eine Abfrage stattfinden, welche Art von Objekt enthalten ist. Das nachfolgende Beispiel funktioniert in InDesign und Illustrator gleich. Es wird der Name des ersten Objekts in der Selektion abgefragt, wenn dies eine Textbox ist, gibt das Skript eine Meldung zurück.


```
// InDesign & Illustrator
var firstListItem = app.activeDocument.selection[0];
// first object in selection
if(firstListItem instanceof TextFrame){
/* check the type */
alert(„I am a „ + firstListItem.constructor.name);
// and name it
}
```

Viele solcher Abfragen können ein Skript schnell komplex werden lassen. Oder anders ausgedrückt, je universeller der Nutzen sein soll, desto mehr Umgebungsvariablen müssen beachtet werden. Dies benötigt Zeit.



https://gist.github.com/2654645.js?file=appSelection.jsx

```
{ // START SCRIPT
main();
// you need a function to be able to cancel a script
function main(){ // all is in here
var w = 25; // the image sizes
var allImages = loadFiles(„*.jpg“); // opens a prompt
and lets the user choose a folder
if(allImages == null) return;
// if that what the function returns is nul
// cancelthe script
var pw = Math.round(
    Math.sqrt(allImages.length)
    ) * w + (w*2); // this will hold the page width
var ph = pw;
if(
    Math.round(Math.sqrt(allImages.length)) != Math.
sqrt(allImages.length)
    ){
          ph = pw + w;
    }
var doc = app.documents.add();
//build a basic document
doc.documentPreferences.pageWidth = pw;
// set the width
doc.documentPreferences.pageHeight = ph;
// set the height
var page = doc.pages.item(0);
// finally - get the first page
var y = w; // the upper left corner
var x = w; // the upper left corner
for(var i = 0; i < allImages.length;i++){
// loop thru all images
var rect = page.rectangles.add({
geometricBounds:[y,x,y + w,x + w]
}); // add a rectangle to the page
rect.place(allImages[i]); // place the image
```



```
rect.fit(FitOptions.FILL_PROPORTIONALLY);
// fit it to the frame
rect.fit(FitOptions.CENTER_CONTENT);
// fit it to the frame
x +=w; // increase x by an image width

if(x >= pw - (w)){
// if x is the width minus an image width
    x = w; // reset x
    y+= w; // and increment y by an image width
    }; //end increase x and y conditional
};//close allImages loop
};// end main function

function loadFiles(type){
// the function that loads the files
var theFolder = Folder.selectDialog („Choose the
Folder“);// user select a folder
if(!theFolder){
// if the folder is not a folder cancel the script
return;
// this cancels the whole function image_loadFiles
}; // end folder check

var allImages = theFolder.getFiles(type);
// get the files by type
if((allImages.length < 1)||(allImages == null) ){
// again if check if there are images
alert(„There are no images of the type: „ + type);
// hm something went wrong? User error
return null; // so we cancel the function
}else{
return allImages; // give back the images. Success!
};// end all images check
};// end function loadFiles
} // END OF SCRIPT
```

Selbst wenn eine vernünftige Vektor-Form einer Karte existiert und griffbereit ist wie können die beiden Länder darin gefunden werden? Was ist wenn eine Grenzänderung stattfindet? Und und und. Um diesen Problemen zu entgehen, kann AEMap.jsx eine Weltkarte in Rektangularprojektion erzeugen. Dabei entsteht eine After Effects Komposition in einer gewählten Skalierung (immer 2:1) in der 178 Prä-Kompositionen enthalten sind, die 286 einzelne Polygon-Formen beinhalten.
Der Nutzer kann zwischen verschiedenen Einstellungen wählen, zum Beispiel die Karte mit Kontur oder ohne zu zeichnen. Er kann entscheiden, ob alle Polygone auf eine Ebene gezeichnet werden sollen oder ob in die oben genannten Kompositionen gesplittet werden soll. Ebenfalls können 3D-Einstellungen definiert werden und ähnliches mehr. Die Daten bestehen auf einem GeoJson Datensatz, der zum freien Gebrauch ins Netz gestellt wurde. Der gesamte Funktionsumfang ist auf dieser Webseite* dokumentiert. Dieses Skript spart nicht nur mir Zeit, sondern auch anderen. Die Resonanz in der After Effects Community“ ist groß. Daher hat das Skript seit seiner Veröffentlichung auf AEScripts.com am 10 April 2012 bereits über 550 Downloads gehabt (heute 4 Mai 2012). Das Tutorial und das Demo wurden bereits über 5000 mal auf Youtube geladen. Aber genug der Selbstbeweihräucherung. Der Vorteil an solchen und ähnlichen Werkzeugen, die zum Beispiel auf AEScripts.com bereit gestellt werden, ist, dass diese meist aus dem Zwang heraus entstanden, sind einen Arbeitsablauf zu automatisieren, um Zeit zu sparen.

http://fabiantheblind.github.com/AEMap/

4.2 createBook

Ein weiteres Werkzeug zum Multipublishing, das ich noch nicht umgesetzt habe und auch vielleicht niemals umsetzen werde, wäre folgendes. Zum Schreiben dieser Arbeit benutze ich eine Auszeichnungssprache genannt Markdown*, in einem Editor genannt iAWriter.

Die Auszeichnungssprache Markdown wurde von John Gruber entwickelt und erlaubt es in reinen Textdokumenten, Auszeichnungen wie Überschrift 1 oder Zitat zu verwenden, ohne dass der Lesefluss maßgeblich beeinträchtigt wird.

```
#Überschrift 1
##Überschrift 2
**Fett** oder __Fett__
*Italic* oder _Italic_
```

Der obere Text wird nach dem Übersetzen, durch mein CSS (Cascading Style Sheets) gesteuert, so dargestellt:

Überschrift 1
Überschrift 2
Fett oder **Fett**
Italic oder Italic

Mit iAWriter habe ich einen weißen Bildschirm und bin frei von jedweder Ablenkung. Mit diesem Set, die Markdown Auszeichnung und iAWriter, bin ich voll auf das Schreiben konzentriert und lasse mich nicht von Layoutaufgaben stören. Um diese Texte in ein lesbare und mit CSS gestaltbare Form zu bringen, benutze ich Jekyll. Jekyll ist ein in Ruby geschriebener Markup-Übersetzer, der statische .html Dokumente exportiert. Mit einem in Ruby geschriebenem Jekyll Plugin **könnte**, neben dem .html, auch eine für InDesign verständliche Datei erzeugt werden ein: .idml (InDesign Markup Language). Ebenfalls könnte ein .epub (Electronic Publication) oder ein RSS-Feed (Rich Site Summary) erzeugt werden. Alle drei sind offene Standards und basieren auf der .xml (EXtensible Markup Language) Formatdefinition.

Diese Pipeline kann mit unterschiedlichen Templates für verschiedenste Formate gesteuert werden. Auch hier gilt, die erzeugten Ergebnisse der Automation sind nicht das Endergebnis, sondern müssen noch weiterverabeitet werden. Es würde jedoch die Datenübertragung erleichtern und mit einer Anweisung in der Kommandozeile `jekyll --server` Rohdaten für verschiedenste Medien erzeugen. Diese müssten dann widerum einen Gestaltungsprozess durchlaufen. Leider ist dies nur eine Idee. Um es umzusetzen, müsste ich Ruby, das idml Schema, das epub Schema, mehr über xml, dtd, xslt, xPath, mehr über html und css und wie ich Plugins für Jekyll schreibe, lernen. Bis zum Ende dieser Arbeit ist das nicht zu bewerkstelligen. Aber toll wäre es.

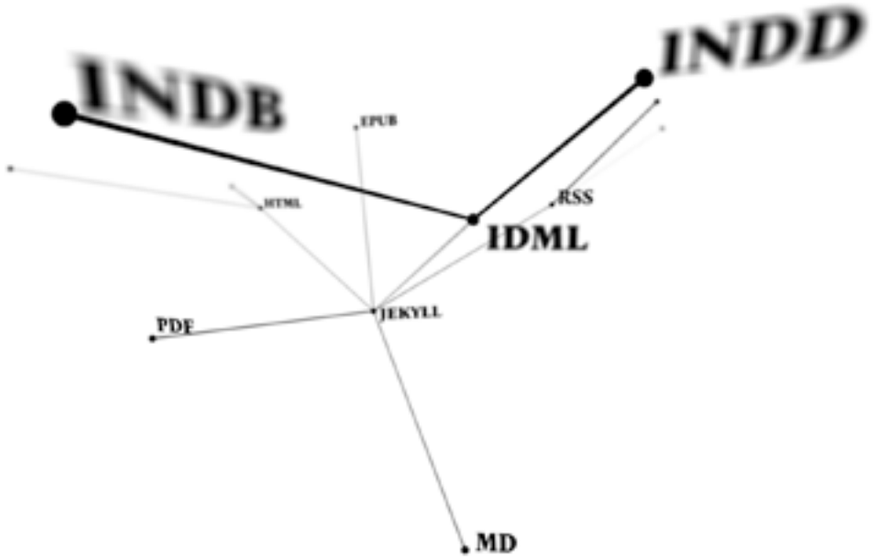
Mit 3 Befehlen könnte dies schon auf einem Webserver bereitgestellt werden.

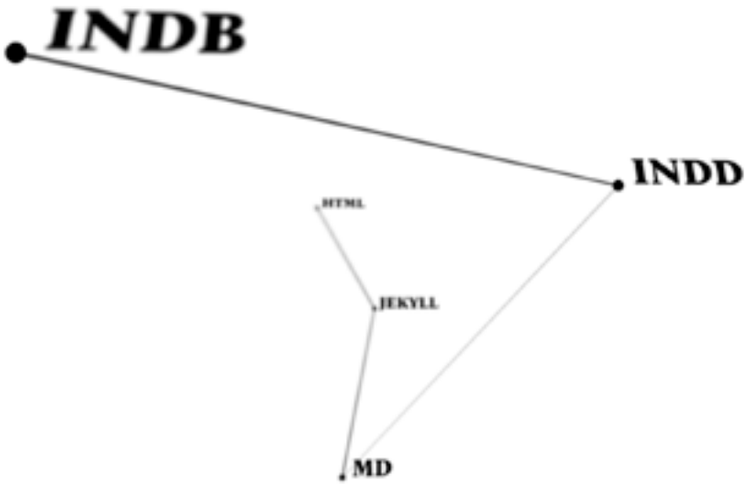
```
git add --all
git commit -m „This is the mes-
sage describing the commit“
git push origin gh-pages
```

Damit sind die Daten im Netz verfügbar und können eingesehen werden. Hier könnte ein online Korrektursystem mit differenzierten Nutzerrechten eingebunden werden und und und…

Der aktuelle Prozess orientiert sich an dieser Idee, ist jedoch nicht komplett automatisert. Die Markdown Dateien werden mit einem Skript zu einem InDesign Buch zusammengeführt und die Auszeichnungen werden in Absatzformate übersetzt. Alle referenzierten Bilder werden auf den entsprechenden Seiten platziert. Ab diesem Punkt wird der „Feinschliff“ des Layouts manuell durchgeführt. Das Übersetzen der Auszeichnung wird durch „Suchen und Ersetzen“-Routinen bewerkstelligt. Dies ist eine rohe Art der Transformation. Es wäre sinnvoller, die Übersetzung nicht mit einem Workaround zu erledigen. Wie bereits oben erwähnt, fehlt jedoch dafür die Zeit.

https://gist.github.com/2659939





6 Fazit

Programmieren ist nicht leicht, aber leichter als man denkt.
Gerade der Anfang hat eine flache Lernkurve und der Lernende wird auch noch mit einem neuen und komplizierten Wortschatz konfrontiert. Ab einem gewissen Punkt ändert sich dies, aber der muss erst erreicht werden. Das Problem ist dabei, dass wir gerne von komplexen Systemen träumen, um solche umzusetzen, fehlt uns dann das Können. Denn zuerst gilt es, die einfachsten Schritte zu machen. Wir sind in ein Entwicklungsstadium zurückgeworfen, in dem wir noch nicht unsere Bedürfnisse formulieren können, aber möchten. Dies kann demotivierend sein. Wenn der flache Teil der Lernkurve endlich überwunden ist, kann diese Fähigkeit wie ein weiteres Werkzeug verwendet werden. Ab diesem Punkt eröffnet die Automation durch Programmierung für uns neue Welten in der Gestaltung. Aufgaben die einmal unmöglich schienen, können nun angegangen werden. Tausend Seiten mit tausend Bildern sind kein Problem. Hinzu kommt, dass beim Programmieren lernen sich ein neues Verständnis für die Logik von Computern bildet und deren Eigenheiten nicht mehr magisch sind. Dies nimmt die Angst vor der direkten Interaktion mit der Maschine. Ich denke nicht mehr, dass jeder programmieren lernen muss, um die Rechner, die wir haben, voll ausnutzen zu können, ist es jedoch unerlässlich. Denn mit den Programmen, die wir benutzen, kratzen wir nur an der Oberfläche ihrer Leistungsfähigkeit. Wir sind beschränkt auf die Funktionsweise, die ein anderer für uns entworfen hat, obwohl wir der Meinung sind, dass wir es eigentlich besser wissen. Ein weiterer Vorzug ist folgender. Nur wenn ich die Grundlagen einer Idee verstanden habe, kann ich darüber kommunizieren. Wie kann ein Gestalter eine Webseite planen, ohne ein wenig Verständnis für die Vorgänge und Funktionsweisen zu besitzen? Nur durch ein Grundverständnis können die Möglichkeiten, die dieses Medium bietet, gezielt eingeschätzt und ausgenutzt werden.

Der Idealfall zum Erlernen von Computersprachen ist unter Zwang, beispielsweise wenn sie vor einem Problem stehen, das manuell nicht gelöst werden kann, aber gelöst werden muss. Alle anderen müssen sich selbst überwinden.
Es kann vieles mit einer Automation durch Programmierung bewerkstelligt werden. Die Automation ist jedoch nicht die Lösung, sondern die Entschlackung unseres Arbeitsprozesses von geisttötenden Aufgaben. Jedes Programm oder Skript, welches wir schreiben, ist ein kreativer Prozess an dem wir unsere Fähigkeiten verbessern. Abschließen möchte ich mit einem Zitat von Marshall McLuhan um noch eine weiter Betrachtungsweise der Automation zu eröffnen.

> *Automation ist Information, und sie macht nun nicht nur den Spezialaufgaben im Bereich der Arbeit ein Ende, sondern auch der Auffächerung im Bereich des Lernens und Wissens. In Zukunft besteht die Arbeit nicht mehr darin, seinen Lebensunterhalt zu verdienen, sondern darin, im Zeitalter der Automation leben zu lernen. Das ist ein ganz allgemeines Verhaltensmuster im Zeitalter der Elektrizität. Es beendet die alte Dichotomie von Kultur und Technik, von Kunst und Handel und von Arbeit und Freizeit. Während im mechanischen Zeitalter der Fragmentierung Freizeit die Abwesenheit von Arbeit bedeutet oder bloßes Müßigsein, gilt im Zeitalter der Elektrizität gerade das Gegenteil.*
> *Marshall McLuhan*
> *„Die magischen Kanäle - Understanding Media“*