Inhaltsverzeichnis:

1 Vorwort 2
2 Meine Motivation 3
3 Gedanken zur Programmierung 4
3.1 …für den Anfänger 5
3.1.1 Was ist Programmierung (für mich) 5
3.1.2 Und wie geht jetzt programmieren? 7
3.1.3 Was brauche ich zum Programmieren? Womit geht es los? 8
3.1.4 Von Konzepten und Tools 11
3.1.5 Von Konzepten und Tools 13
3.2 …für den Fortgeschrittenen 17
3.3 …für den Profi 18

1 Vorwort

Jeder hat so seine eigenen Visionen. Meine ist, dass ich irgendwann ein Programm erstellen werde, das ein Archiv für Klassik-CDs ist.

Als gelernter Host-Entwickler habe ich es mit COBOL, IMS, JCL und DB2 zu tun gehabt, alles Dinge, für die man einen Großrechner braucht. Da mir der für den Hausgebrauch nicht zur Verfügung steht, habe ich mich auf die Entwicklung mit frei zur Verfügung stehenden Programmiersprachen und Entwicklungswerkzeugen konzentriert. Für die Auswahl der geeigneten Mittel habe ich mich viel mit Kollegen unterhalten, Tipps bekommen, gegoogelt, ausprobiert, verworfen, Neues ausprobiert, auch das verworfen, bin von „Hölzchen auf Stöckchen" gekommen, habe die Lust verloren, hingeschmissen, liegen gelassen und schließlich doch irgendwie weiter gemacht, die Scherben aufgesammelt, neu zusammengesetzt und festgestellt, dass es immer noch ein langer Weg ist, bis ich meine Vision umgesetzt haben werde.

Was ich damit sagen will, Programmieren bedeutet Arbeit. Und viel Zeit. Es ist total frustrierend, wenn etwas nicht so klappt wie man das will, oder man sogar feststellen muss, dass man die letzten Wochen komplett in eine Sackgasse investiert hat. Aber es ist auch total faszinierend, wenn man den entscheidenden Hebel umgelegt hat und dann läuft der Kram! Hurra, es läuft wirklich! Wahnsinn! Toller Augenblick, der für vieles entschädigt!

Und so dachte ich mir, ich versuche mein angesammeltes Wissen in dieser Form zugänglich zu machen, vielleicht hilft das ja der Einen oder dem Anderen über die ersten Hürden hinweg.

Die nachfolgenden Kapitel sind also eine Mischung aus Erfahrungsbericht und dem Philosophieren über Programmierung.

Ich bin selbst mal gespannt, was ich so alles zusammentragen werde…

Rückmeldungen dazu natürlich immer gerne per Mail. Jetzt erstmal viel Spaß beim Lesen. Viele Grüße,

Papa

2 Meine Motivation

Warum ein Archiv für Klassik-CDs? Ich glaube, es könnte ganz interessant sein, ein Programm zu entwickeln, das dem Nutzer die Gelegenheit gibt, seine Klassik-CDs an einem Ort zu verwalten, sodass er oder sie alle auf der CD verfügbaren Informationen speichern und abfragen kann.

Eine Klassik-CD unterscheidet sich von einer POP- oder Rock-CD darin, dass die auf der CD enthaltenen Informationen viel umfangreicher sind. So ist für den Klassik-Liebhaber interessant zu wissen, welches Orchester auf der CD zu hören ist, welcher Dirigent gerade am Pult steht, wo die Aufführung stattgefunden hat, wer die Solisten sind, und so weiter.

All diese Daten kann man in den mir bekannten gängigen Archiv-Programmen nicht eingeben. So glaube ich, dass das ein ganz lohnendes Betätigungsfeld wäre, um zum einen die Programmierung zu erlernen, zum anderen die erste Million in Angriff zu nehmen – also Probleme, nicht Euro…

Ich habe mich daher seit 2009 durch diverse unterschiedliche Programmier-sprachen, Konzepte und Methoden gewühlt. Dabei habe ich immer wieder Stunde um Stunde im Internet in diversen Foren nach Antworten gesucht, teilweise hatte ich dann mehr Fragen als Antworten, sodass ich wie oben schon erwähnt wieder von vorne angefangen habe.

Deshalb habe ich mich dazu entschlossen, dieses Dokument anzugehen und auf meiner Homepage online zu stellen, vielleicht liest das ja jemand und findet über das Dokument zur Programmierung. Das würde mich freuen, denn ich glaube, dass die Programmierung wirklich Zukunft hat.

3 Gedanken zur Programmierung

Nachfolgende Kapitel habe ich für Anfänger und Fortgeschrittene aufgeteilt. Den Profis unter Euch – und sicher auch manchem Fortgeschrittenen – kann ich vermutlich nichts beibringen, ich habe trotzdem ein Kapitel aufgemacht. Vielleicht gibt es ja noch den ein oder anderen Gedanken, der in dieser Form noch nicht bekannt ist.

Was erwarte ich von einem Anfänger? Nichts. Ich versuche sie oder ihn ganz am Anfang abzuholen. Wenn das für den Leser zu Basic ist, tut mir leid, steigt später ein.

Was setze ich bei einem Fortgeschrittenen voraus? Eigentlich auch nichts. Themen wie Unterschied zwischen Interpreter und Compiler, IDE, MVC, vielleicht sogar die ersten Schritte in Datenmodellierung, lauter so Kram halt sollten Euch geläufig sein.

Um es hier einleitend noch einmal zu sagen, Programmieren bedeutet Arbeit. Viel Arbeit. Und Lesen. Viel Lesen. Deutsch und Englisch. Ohne geht’s nicht.

Es wird einem nichts geschenkt, *die* Lösung für *das* Problem mag es irgendwo im Internet geben, im Zweifel findet man aber den Link dahin nicht, er ist kaputt oder die Beschreibung ist so missverständlich, dass man nicht weiterkommt. Oder es ist mit Kosten verbunden.

Foren lohnen sich auch immer, man sollte sich aber mit *einem* Problem für *ein* Forum entscheiden. Eine allgemeine Frage zu HTML in 27 Foren zu platzieren macht keinen Sinn. Dann lieber erst nach einem Spezial-Forum schauen, sich anmelden und die Frage stellen. Und gebt immer auch die Source mit, die Ihr gerade am Wickel habt. Die anderen wissen nicht, was Ihr gerade macht, sie müssen sich auf Euren Kommentar einlassen, da sagt der Source-Code mehr als 1.000 Worte.

Wenn man es dann aber geschafft hat, mit den zur Verfügung stehenden Mitteln einen funktionierenden Prototypen entwickelt zu haben, gibt einem das ein echtes Glücksgefühl. Es macht riesig Spaß sich tief in die Materie zu graben um dann am Ende ein solches Erfolgserlebnis zu genießen.

Wie einleitend erwähnt, komme ich aus der Welt des Großrechners. Das ist für die Entwicklung im privaten Umfeld nicht schädlich, hilft aber auch nur bedingt. So sind mir die Grundzüge guter (und damit auch schlechter) Programmierung natürlich vertraut, das hilft mir allerdings nur bedingt, da ich beispielweise ein

COBOL-Code strukturieren kann, allerding keine Ahnung von der korrekten Darstellung der HTML-Tags hatte. Das gehörte (und gehört immer noch, das hört nie auf!) zum Lernprozess dazu.

So, jetzt die Vertiefung…

3.1 …für den Anfänger

Hey, Ihr habt Lust auf Programmierung, wisst aber nicht so genau wie anfangen? Oder Ihr habt schon in der Schule „mit C rumgemacht", das ist aber schon lange her und da ist vieles verschütt gegangen? Perfekt, damit seid Ihr hier vermutlich richtig.

Ich habe viel Blabla zusammengestellt, vielleicht hilft Euch davon ja was. Und immer daran denken, es ist noch kein Meister vom Himmel gefallen, Ihr braucht viel Zeit um das richtig zu verstehen.

3.1.1 Was ist Programmierung (für mich)

Zunächst glaube ich, dass man ganz gut mit der Vorstellung fährt, dass Programmierung der Vorgang ist, meinem Computer das beizubringen, was ich gerne von ihm hätte, das er im Rahmen seiner Möglichkeiten für mich erledigt.

Das muss ich in einer für ihn verständlichen Sprache tun, sonst versteht er mich nicht. Der Begriff Programmier-„Sprache" passt für mich auch ganz gut, wenn ich mich mit jemandem unterhalten will, müssen wir uns auch als erstes auf eine gemeinsame Sprache einigen. Im gleichen Land ist das in der Regel die Landessprache. International hat sich natürlich Englisch durchgesetzt. Das heißt, wenn ich, sagen wir mal in Aserbaidschan, ein Bier bestellen möchte, ist es hilfreich zu wissen, was „bitte ein Bier" auf Englisch heißt. Damit, dass ich in Aserbaidschan an der Bar den Satz „one beer please" ausspreche, ist die Wahrscheinlichkeit höher, dass mich der Mensch hinter dem Tresen versteht, als wenn ich das auf Deutsch sage.

Okay, das Beispiel ist banal, soll aber auch nur verdeutlichen, dass ohne gemeinsame Sprache die Verständigung ziemlich schwierig wird.

Der Mensch hinter der Theke kann zur Not noch meine Mimik und Gestik interpretieren, das kann der Rechner schon nicht mehr, bzw. wenn er es kann, hat man ihm das mühsam beibringen müssen. Ich gehe aber mal davon aus, dass der

Rechner auf dem dieses Dokument gerade betrachtet wird, diesen Transfer nicht leisten kann.

Das Bespiel zeigt aber auch, dass ich die Sprache in ihren Grundzügen verstanden haben muss, also die Grammatik verstehe. Wenn mir der Barkeeper mitteilt, dass er momentan kein Bier hat, ich das aber nicht verstehe, muss ich durstig ins Bett gehen.

Mit diesen Analogien wieder zurück zur Programmierung.

Wie gesagt, wenn ich möchte, dass der Rechner das macht, was ich von ihm erwarte, muss ich das in einer für ihn verständlichen Sprache mit den in dieser Sprache geltenden Regeln und Worten tun, sonst „reden" wir aneinander vorbei.

Ich habe keine Ahnung, wie viele unterschiedliche Programmiersprachen es auf der Welt gibt. Ich gehe aber mal von über einhundert bekannteren und hunderte weiterer Spezialsprachen aus. Bei den Spezialsprachen handelt es sich oft um Weiterentwicklungen bekannterer Sprachen, man spricht hier tatsächlich auch von „Dialekten". Auf Wikipedia gibt es eine Seite, die Beispiele gibt, wie in den sogenannten „Hochsprachen" die Worte „Hello World!" ausgegeben werden sollen.

Dieses „Hello World!" ist eine tolle Idee. Ursprünglich ist das wohl zuerst in einem Tutorial für das Erlernen der Programmiersprache „C" irgendwann um 1975 aufgetaucht, hat sich dann aber schnell verselbständigt. Großartig! Mehr dazu später.

Im Gegensatz zu der „Hochsprache" steht die „Maschinensprache". Die Maschinensprache ist tatsächlich die Sprache, in und mit der eine Maschine gesteuert werden kann. Die Sprache ist in der Regel auf die Befehle beschränkt, die sie ausführen kann und enthält keine weiteren Funktionen. Eine Hochsprache besitzt einen deutlich komplexeren Aufbau und ist in der Regel auch nicht nur auf einer speziellen Maschine lauffähig. Damit sie von der Maschine „verstanden" werden kann, muss die noch in die Maschinensprache „übersetzt" werden. Auch das vertiefen wir später.

Zusammenfassung (tldr):

Programmieren ist für mich der Vorgang der Softwareerstellung um einen Computer anzuweisen, eine bestimmte Aktion auszuführen. Das muss in einer für ihn verständlichen Sprache geschehen.

3.1.2 Und wie geht jetzt programmieren?

Wie wir oben herausgearbeitet haben, besteht die Kunst des Programmierens darin, in einer für uns verständlichen Sprache aufzuschreiben, was wir wollen, das der Computer für uns macht. Das wird dann in die Maschinensprache übersetzt und die Maschine führt die Befehle dann aus.

Soweit so gut – beim Klavierspielen muss man auch nur zur richtigen Zeit auf die richtige Taste drücken, also alles ganz einfach?

Von wegen. Wer sich schon einmal eingehend mit einer anderen Sprache als seiner Muttersprache befasst hat, wird bestätigen können, dass der Teufel oft im Detail steckt. Schon zweit kleine Punkte können den Sinn komplett verändern. Wenn man aus der Vogelperspektive auf die beiden Worte „Achtung" und „Ächtung" schaut, unterscheiden sie sich nur durch 2 Punkte. Inhaltlich sind das 2 völlig verschiedene Dinge. Aber auch das gleiche Wort kann unterschiedliche Bedeutungen haben, dafür haben wir im Deutschen sogar ein Spiel - „Teekesselchen". Ein Wort mit 2 oder mehr Bedeutungen, wie „Bank" oder „Schloss".

Auch die Interpunktion ist wichtig, wie heißt es in dem blöden Witz – Interpunktion rettet Leben – „Wir essen Papa" und „Wir essen, Papa"…

Warum reite ich so auf der Sprache herum – das Erlernen einer Programmiersprache ist zum ganz großen Teil geprägt vom Erlernen der Syntax der Sprache, der Grammatik, der Interpunktion, also dem theoretischen Teil der Sprache.

Das Wichtigste beim Programmieren ist aus meiner Sicht das Machen von Fehlern. Nur wer Fehler macht, kann daraus lernen und macht sie vielleicht nicht noch ein zweites Mal. Und wenn man doch das zweite Mal auf den gleichen Fehler hereingefallen ist, ihn gefunden hat, ist ein schöner „Stirnklatscher" echt entspannend. Oder um es mit den Worten von Homer Simpson zu sagen „D'oh!".

Welche Programmiersprache ist jetzt die richtige für mich?

Das ist die alles entscheidende Frage. Sozusagen die Frage der Fragen. Frei nach Douglas Adams ist die Antwort „42", die Frage dazu müssen wir allerdings erst noch finden. Also nicht sehr hilfreich.

Um auch das gleich zu sagen, ich habe in diesem Dokument auch nicht die Antwort, ich kann aber vielleicht helfen, Anstöße für Fragen zu formulieren damit Ihr Euch selber die richtigen Fragen stellt um dann Antworten geben könnt.

Bei der Vielzahl von unterschiedlichen Programmiersprachen gibt es sicher für die Lösung Eures Problems durchaus mehrere Programmiersprachen, die eine Lösung bieten können. Da ich selber auch nicht alle Sprachen kenne, kann ich Euch auch nur einen kleinen und damit unvollständigen Ausschnitt bieten, die Herausforderung wird sein, die richtige Programmiersprache für das anstehende Problem zu finden. Vielleicht ist es auch eine Kombination aus mehreren Programmiersprachen, die dann zum Ziel führt.

Jetzt haben wir aber immer noch nichts programmiert, sondern immer nur philosophiert, wann geht das denn jetzt endlich los? Nächstes Kapitel?

Zusammenfassung (tldr):

Suche Dir eine Sprache aus, wende sie an und lerne aus Deinen Fehlern.

3.1.3 Was brauche ich zum Programmieren? Womit geht es los?

Und wieder wird es philosophisch – die Antwort lautet, „Das kommt darauf an." Es kommt tatsächlich auf die Problemstellung an. Im einfachsten Fall reicht mir der Computer, an dem ich gerade sitze vollkommen aus, weil er alle notwendigen Programme zur Programmierung bereits enthält, manchmal reicht das aber nicht aus und ich muss mir noch weitere Programme installieren. Die können dann vielleicht auch kostenpflichtig sein, vielleicht gibt es aber auch eine freie Lösung dafür. Wir versuchen das in diesem Kapiteln mal näher zu beleuchten.

Fangen wir also einfach mal an mit der Programmierung, oder?

Eigentlich schon, aber was sollen wir machen? Was soll das Programm tun? Gute Fragen, gell? Wir brauchen also zuerst immer Idee oder Vorstellung von dem, was unser zukünftiges Programm machen soll. Den Schritt nennt man „Anforderungsanalyse" oder „Designphase" oder auch „Erstellen der Vision", je nachdem wen man fragt und nach welcher Methodik man vorgeht. Nach den neueren Methodiken geht man dazu über, den Auftraggeber der Software zu fragen, was soll das Ding denn können. Dazu gibt ihm der Entwickler eine Idee, wie er die Anforderung formulieren soll, nämlich als „use case", übersetzt sowas wie „Anwendungsfall". Dabei wird eine Art Fragekorsett zusammengestellt. Man bittet den Auftraggeber den nachfolgenden Satz zu vervollständigen – das nennt sich dann „user story":

„Als [Rolle] möchte ich [Ziel], um [Nutzen]"

Okay, das muss man erklären. Mit Rolle ist in diesem Fall „eine Person oder Sache in der Rolle als" gemeint. Also „Als Nutzer…", „Als Kunde…" oder „Als Prozess zum Durchführen von Überweisungen…"

Als Ziel soll der Auftraggeber etwas angeben, das es zu erreichen gilt. „Als Kunde möchte ich beim Start der Anwendung mit meinem Namen begrüßt werden…".

Als letztes noch der Nutzen. Das hat den praktischen Hintergrund, dass der Auftraggeber sich über den Sinn und Zweck seiner Anforderung Gedanken machen soll. „Als Kunde möchte ich beim Start der Anwendung mit meinem Namen begrüßt werden, da diese Form der Personalisierung der Kundenbindung an das Unternehmen erhöht.". Coole User-Story. Im Internet finden sich dazu viele interessante Seiten, es lohnt sich da die Suchmaschine mit den Begriffen zu füttern!

Aber wir haben immer noch nicht programmiert, wann geht das denn endlich los? Gaaanz ruhig…

Also, was können wir aus der User-Story für uns ableiten. Offensichtlich soll ein Programm gebaut werden, das unter anderem den Namen des Kunden ausgibt. Daraus lässt sich schließen, dass es einen Kunden in unserem System geben muss, der sich dann auch noch als solcher zu erkennen geben muss, und wir ihm dann am Bildschirm anzeigen, „Hallo Papa, schön dass Du mal wieder vorbeischaust". Oder so. Details sind dann mit dem Auftraggeber abzustimmen.

Sonst noch was? Nö, aus der User-Story können wir aktuell nicht mehr rausholen. Aus meiner Sicht ist die Anforderung mit dieser einzigen User-Story aber noch nicht vollständig. Wenn sich der Kunde die Mühe gemacht hat, sich bei uns anzumelden, also Nutzernamen und Passwort zu hinterlegen um sich dann bei uns einzuloggen, nur, damit er seinen Benutzernamen sieht, macht nicht wirklich Sinn.

Sieht man das allerdings im großen Kontext beispielsweise eines Web-Shops macht das schon wieder Sinn, ist dann aber „nur" die Phase der Anmeldung. Die Bestellung ist damit nicht abgedeckt, ebenso wenig der Geschäftszweck unseres Web-Shops.

Zurück zu der Anforderung, in unserem Beispiel soll also eine Anmeldung an das System gebaut werden. Ich als zuständiger Entwickler stimme die Einzelheiten als nächstes mit dem Auftraggeber ab.

Dabei interessieren mich Dinge wie

- wie kommen wir an unsere Kunden

- wo werden die im System abgelegt
- haben wir nur eine Webseite, oder müssen das Gleiche auch in unsere App einbauen
- wieviel Aufwand darf ich in die Analyse stecken

Und wenn ich weiter nachdenke, fallen mir sicher noch duzende andere Fragen ein. Sobald die Fragen beantwortet sind, habe ich ein Bild, wie sich unsere Anmeldung in das Gesamtbild einfügt.

Als Programmierer für Eure eigenen Programme seid Ihr natürlich für die Anwendung selber verantwortlich, es sagt Euch niemand, was Ihr tun sollt oder wie die Oberfläche aussieht. Das müsst Ihr selbst mit Euch ausmachen. Das Wissen um die Vorgänge rund um die Use-Cases hilft aber, eine eigene Struktur zu entwickeln. Im Zweifel habt Ihr Euch für eine Programmiersprache entschieden, am Ende stellt ihr aber fest, huch, so hatte ich mir das nicht gedacht, dass passt ja gar nicht, jetzt muss ich das nochmal überarbeiten… Ist dann blöd gelaufen, aber auch das ist nicht tragisch, ich erinnere an den Absatz mit dem „Fehler machen".

Zusammenfassung (tldr):

Vor dem Hacken muss man sich mit den Anforderungen an das zu erstellende Programm befassen. Daraus lassen sich dann Rückschlüsse über die zu verwendende Programmiersprache ziehen.

3.1.4 Von Konzepten und Tools

Und, haben wir schon eine Zeile programmiert? Nein, immer noch nicht, das wird sich aber jetzt ändern.

Oben habe ich angesprochen, dass manchmal schon das Vorhandene auf dem Rechner ausreicht, um zu programmieren. Was meine ich damit? Okay machen wir unser erstes Beispiel.

Bitte erstellt bei Euch auf dem Computer ein einfaches Text-Dokument. Nehmt nicht Microsofts Word oder den Writer von LibreOffice oder ein anderes Office-Programm, nehmt einen Editor ohne die Möglichkeit von Text-Formatierung. Bei Windows heißt das Gerät einfach nur „Editor". Oder Ihr installiert Euch Notepad++, das ist mein Lieblings-Texteditor!

Das Dokument nennt bitte „Hallo.html" und es sollte nur die Zeile

```
Hello World!
```

enthalten. Mehr noch nicht. Ich weiß natürlich nicht, was für ein Betriebssystem Ihr habt, von daher kann ich Euch hier keine weiteren Tipps geben, das muss aber klappen, das schafft Ihr schon!

Ihr habt also jetzt ein HTML-Dokument erstellt. HTML steht für „Hypertext Markup Language" und ist die Sprache des Internet. Wenn Ihr jetzt einen Doppelklick auf die Datei Hallo.html macht, sollte ein Browserfenster in Eurem Standard-Browser aufgehen und der Text „Hello World!" sollte erscheinen.

Haben wir denn jetzt programmiert? Wir haben doch nur einen kleinen Text in einer Datei abgelegt, und diese irgendwie benannt, was ist denn daran programmieren? Oder ist das tatsächlich so einfach?

Die Antwort lautet, in diesem Fall ist das tatsächlich so einfach. Allerdings nicht, weil wir nicht programmiert haben, sondern weil die Erfinder der HTML-Syntax gesagt haben, sie wollen die Sprache so tolerant gegenüber Fehlern machen, wie nur irgend möglich. Und da in unserem Fall alle Steuerelemente der HTML-Syntax fehlen (was das ist sehen wir gleich), sagt sich der Browser bei der Interpretation „hey, ich habe keine anders lautenden Steuerinformationen, also zeige ich das mal als Text an".

Wie sehen also, die Fehlertoleranz des Einen ist die Freude des Anderen, also in diesem Fall unsere Freude!

Wenn wir ein wirkliches HTML-Dokument erstellen wollen würden, brauchen wir noch weitere Infos in der Datei, den oben stehenden „Steuerelementen der HTML-Syntax". Das Grundgerüst für HTML sieht wie folgt aus:

```
<!DOCTYPE html>
<html>
	<head>
		<title></title>
	</head>
	<body>
	</body>
</html>
```

Eine ausführliche Erklärung dazu gibt es im Internet bei selfhtml.org, der Link dazu ist:

https://wiki.selfhtml.org/wiki/HTML/Tutorials/HTML5/Grundger%C3%BCst

Die Tutorials sind prima, da kann man eine Menge lernen. Deshalb gehe ich an dieser Stelle auch nicht weiter auf die Syntax ein, nur so viel, wenn wir unser Dokument „Hallo.html" erweitern um:

```
<!DOCTYPE html>
<html>
	<head>
		<title></title>
	</head>
	<body>
		<p>Hello World!</p>
	</body>
</html>
```

es wieder speichern und dann mit Doppelklick auf das Programm öffnen, sehen wir was? Häh? Das ist doch das Gleiche wie ohne den ganzen Kladderadatsch, warum haben wir das denn jetzt geschrieben?

Na, weil jetzt der Trick kommt. Jetzt *weiß* der Browser, dass er einen Text ausgeben soll, weil wir das so programmiert haben. Die Macher von HTML haben festgelegt, dass der Browser alles was zwischen `<p>` und `</p>` steht als Text auszugeben hat.

Ohne den ganzen Klammer-Wirrwarr sagt sich der Browser „ich habe keine Anweisungen, also stelle ich das mal als Text dar". Das was der Browser machen

soll („ich soll einen Text ausgeben, das sagt mir die Anweisung <p>") und was er *tatsächlich macht* obwohl er nicht weiß was er tun soll („da ich es nicht besser weiß, gebe ich das mal als Text aus"), ist nur zufällig identisch. Klingt komisch, ist aber so.

Okay, wir wollten einen Text, wir haben auch einen Text bekommen, aber das war, wie gesagt nur Zufall. Wenn wir beispielweise eine Überschrift gewollt hätten, wäre das so nicht gegangen.

Ersetzt spaßeshalber mal in die Zeile

```
<p>Hello World!</p>
```

durch

```
<h1>Hello World!</h1>
```

Na, was seht Ihr? Aha, da der Browser *weiß*, dass er alles was zwischen `<h1>` und `</h1>` steht in der größten Überschriftscodierung anzuzeigen hat, ist der Text jetzt viel größer dargestellt.

Haben wir jetzt programmiert? Ja, spätestens jetzt können wir von uns behaupten programmiert zu haben – Glückwunsch!

Und wie geht es jetzt weiter? Tja, wenn ich das so genau wüßte…

Auch hier gilt, es hängt davon ab. Sorry, wieder keine wirklich tolle Antwort. Aber das war 42 auch nicht, oder?

3.1.5 Von Konzepten und Tools

Was ich allerdings sagen kann, wir haben eben das Konzept des Interpreters kennen gelernt. Ein Interpreter ist ein Programm, das in eine Datei schaut und Zeichenketten darin „interpretiert". Ein Browser ist ein solcher Interpreter. Dateien, die die Endung .html oder .htm haben, werden beim Öffnen vom Browser Zeichen für Zeichen von oben nach unten durchsucht und die Anweisungen darin ausgeführt.

Ich greife den Tutorials ein wenig vor, weil ich das hier für die Erklärung brauche. Der Interpreter hält Ausschau nach den Zeichen „<" und „>". Alles was zwischen diesen Zeichen steht ist eine Anweisung. Sie weist den Browser an, eine bestimmte Aktion durchzuführen. Nach dem „<" kann als nächstes Zeichen auch ein"/" stehen, das ist die Ende-Kennung. Es gibt also (fast) immer eine Anfangs-Anweisung und eine Ende-Anweisung.

Gehen wir einen Teil unseres Beispiels durch. Die Zeile

```
<h1>Hello World!</h1>
```

wird wie folgt interpretiert: die erste Anweisung (`<h1>`) bedeutet, dass alles was nach der Klammer zu „`>`" steht als Überschrift 1 interpretiert werden soll. Das gilt solange, bis eine neue Anweisung auftaucht. Das tut sie in diesem Fall in der Form `</h1>` also der Ende-Anweisung für die Überschrift 1.

War jetzt nicht so schwer, oder?

Okay, versuchen wir eine weitere Betrachtung.

Worin unterscheidet sich HTML von anderen Sprachen wie zum Beispiel Java? Ich habe es schon erwähnt, HTML ist die Sprache des Internet. Dabei ist wichtig zu verstehen, wie die Komponenten zusammenhängen. Was passiert, wenn wir eine Webseite aufrufen? Wer liefert wem wann welche Inhalte? Um diese Fragen zu klären müssen wir wieder neue Begriffe einführen. Das Konzept „Client-Server" ist für den Einen oder die Andere vielleicht neu.

Das bezeichnet grob gesagt die Verbindung zwischen 2 oder mehr Rechnern, wobei auf nur einem Rechner ein sogenannter Dienst oder englisch „Service" bereitgestellt wird, das ist dann der „Server". Dieser Dienst wird in der Regel von mehreren Rechnern angefordert, diese Rechner sind dann die „Clients". Wenn wir wie geschehen auf die Web-Seite www.papa-macht-hausaufgaben.de klicken, sind wir der Client, der Browser auf unserem Rechner verbindet sich mit dem Rechner auf dem die Webseite gehalten wird (man sagt „gehostet"), somit fungiert dieser Rechner als Server.

Als Ausblick für die Fortgeschrittenen unter Euch – Datenbank-Operationen finden immer auf dem Server statt, wenn ich also einen einzigen Rechner habe, auf dem die Datenbank und die Abfragen darauf angezeigt werden, muss ich einen Teil meines Rechners zum Server machen, sodass die Datenbankabfragen überhaupt möglich sind.

Wenn Ihr mit dem Begriff Datenbanken noch nichts Konkretes verbindet, wartet bis zum Kapitel für Fortgeschrittene, da lasse ich mich darüber aus. Für jetzt nur so viel, Datenbanken sind sowas wie Kalkulationstabellen. Damit kann man auch jede Menge Zeit verbringen…

Ich habe auf meiner Homepage ein Beispiel für eine CRUD-Anwendung abgelegt, wenn das jemanden interessiert, kann er oder sie sich da gerne bedienen.

CRUD? Wasndas? CRUD steht für die Datenbankoperationen Create also Erstellen, Read also Lesen, Update also Änderung und Delete also Löschen. Der Server stellt

diese Dienste oder Funktionalitäten auf die Datenbank bereit, der Client wählt aus was davon er machen möchte. Die Daten liegen immer an einem Ort (auf dem Server), die Pflege erfolgt durch den Client.

Schöne Sache. Kann ich nur jedem empfehlen mal auszuprobieren. Dann ist aber nichts mehr mit Notepad++ oder UltraEdit, dann braucht Ihr eine richtige IDE. IDE? IDE steht für Integrated Development Environment also integrierte Entwicklungsumgebung. Aha, nu sind wir schlauer.

Integriert überspringen wir mal, eine Entwicklungsumgebung ist ein Stück Software, das mehrere Funktionen an einer Stelle bereitstellt. Sie ist gewissermaßen der Werkzeugkasten, indem sich unsere (Entwicklungs-) Werkzeuge – englisch „tools" – befinden.

Davon gibt es auch ganz viele im Netz, einige von ihnen sind sogar kostenlos. Googelt mal nach NetBeans, Eclipse oder IntelliJ. Die ersten beiden sind frei, bei IntelliJ gibt es eine freie Version, die kann aber nicht alles. Für das was spannend ist muss man bezahlen. Leider, die IDE ist toll! Oder wenn Ihr „was mit mobiler App-Entwicklung" machen wollt das Android Studio, das basiert auf IntelliJ.

Aber auch hier sind wir schon wieder an einem Punkt, wo sich neue Fragen auftun, die ich Euch nicht eindeutig beantworten kann. Ihr müsst Euch vor dem Download der IDE entscheiden, welche Version Ihr haben wollt. Wenn Ihr auf der sicheren Seite sein wollt, nehmt die Version die alle kann. Damit habt Ihr Euch im übertragenen Sinn ein ganzes Fitness-Studio installiert, obwohl Ihr nur ein Hantelpaar haben wolltet.

Und wenn Ihr das jetzt ausprobieren wollt, schaut Euch die Hilfe-Seiten an und installiert alles nach den dort enthaltenen Anweisungen.

Bei den meisten IDEs ist Voraussetzung, dass Ihr eine Java-Laufzeitumgebung installiert habt. Und was ist das jetzt? Ihr habt sicher schon gehört, dass Java eine plattformunabhängige Programmiersprache ist. Was bedeutet das jetzt? Egal welches Betriebssystem auf meinem Rechner läuft, auf (fast) allen Rechnern kann ich ein einmal geschriebenes Java-Programm laufen lassen.

Die Unabhängigkeit wird dadurch erreicht, dass das Java-Programm in einer plattformunabhängigen Form, dem sogenannten Bytecode zur Verfügung gestellt wird. Dieser läuft dann auf einer eigenen virtuellen Maschine, die ist dann wieder plattformspezifisch, sie kennt also die Umgebung und Eigenheiten des Rechners bzw. des Betriebssystems. Cooles Konzept.

Ist aber schon ganz schön fortgeschritten, für jetzt sollten wir uns merken, dass wir das Java Development Kid (JDK) irgendwann runterladen müssen. Lest Euch dazu gerne im Internet schlau.

Wir haben aber mit dem ganzen Java-Geraffel jetzt die zweite Säule der Entwicklung neben dem Interpreter angesprochen. Zur Wiederholung, der Interpreter – wie zum Beispiel ein Browser – interpretiert den von uns geschriebenen Code Zeichen für Zeichen von oben nach unten.

Bei Java passiert etwas anderes. Ich habe zuerst mein Programm, das muss ich in einem bestimmten Ordner mit einer bestimmten Datei-Endung (diesmal .java, nicht .html) ablegen.

Dann starte ich den Compiler und sage ihm, wandele bitte meinen Source-Code in der .java-Datei in Bytecode um. Der Compiler „javac" schnappt sich jetzt die .java-Datei und erstellt eine .class-Datei in die er den Bytecode ablegt. Die .class-Datei kann dann in der virtuellen Maschine gestartet werden.

Eine IDE für die Entwicklung von Java nimmt Euch den ganzen Kram ab, Ihr drückt einfach auf den „Run"-Button und schon geht das Compilieren automatisch los und die .class-Datei startet in der virtuellen Maschine. Perfekt.

So. Langsam kommen wir in die Fortgeschrittenen-Ecke.

Zusammenfassung (tldr):

Ein Interpreter ist kein Compiler, googel nach IDE und installiere den Bums.

3.2 …für den Fortgeschrittenen

Ich habe es bisher noch nicht geschafft, die Texte zu schreiben. Als Themen für den Fortgeschrittenen geistern mir durch den Kopf:

PHP
Datenbanken
JavaFX
SceneBuilder
UML
Scrum

Und sicher noch mehr, wenn ich mal am Schreiben bin.

3.3 …für den Profi

Ganz ehrlich, einem Vollprofi kann ich nichts mehr mitgeben. Wirklich nicht…

Im Gegenteil – falls dies hier irgendjemand liest, der wirklich weiß wo der Hase langläuft, wäre ich über Feedback dankbar.

Wie einleitend dargelegt, kann ich hier nur über die Arbeit als Entwickler und als Business Analyst in Unternehmen philosophieren.

Mir ist aufgefallen, dass die Kluft zwischen den Entwicklern und den Auftraggebern also den Fachabteilungen der Unternehmen immer tiefer wird. So ist das Verständnis für die andere Seite immer weniger vorhanden, was eine vernünftige Kommunikation nahezu unmöglich macht. Woran das liegt kann ich nicht genau sagen, ich habe nur eine Vermutung.

Heute spielen „time-to-market"-Aspekte und andere KPIs eine immer größere Rolle. Dem Einzelnen wird seitens der Unternehmensleitung oder dem Management nicht mehr die Möglichkeit geboten, sich ein umfassendes Bild der gesamten Landschaft zu machen.

So kennt jeder nur noch seinen kleinen Ausschnitt, das „Große Ganze" wird aus den Augen verloren.

Die Aufgabe des Auftraggebers ist das „Was" zu beschreiben. Was soll entwickelt werden, was geändert, was gelöscht. Der Entwickler hat dann die Aufgabe das „Wie" festzulegen. Dabei muss er aber nicht nur die Systemlandschaft kennen, sondern auch wissen, wohin die strategische Reise des Unternehmens geht. Das wird aber immer schwieriger, wenn das Management selber keine Vision mehr hat. Ich hoffe, dass das nur ein Trend ist, den man umkehren kann. Ich glaube nicht, dass das auf Dauer gut geht.

Momentan habe ich leider nicht die Zeit, weitere Gedanken niederzuschreiben, ich hoffe aber, dass dies Dokument nicht allzu lange ruhen muss.

Bis hierhin also meine Gedanken zum Programmieren, ich hoffe, dies interessiert jemanden.

Liebe Grüße,

Papa