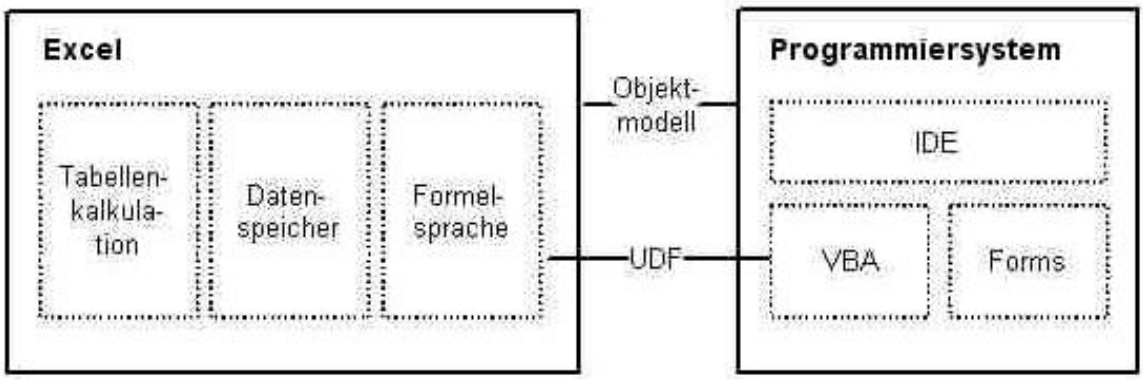
1 Einführung

In diesem Kapitel geht es zunächst um die Möglichkeiten, die Excel in Verbindung mit VBA (Excel-VBA) zur Entwicklung von Software bietet. Darauf aufbauend beschäftigen wir uns mit der Frage, für welche Arten von Software sich Excel-VBA eignet und in welcher Form wir diese Software an die Abnehmer weitergeben sollen. Anschließend gehen wir der Frage nach, was die Qualität einer Software ausmacht, und ob die Verwendung von Excel-VBA besondere Konsequenzen in Bezug auf die Softwarequalität hat. Zum Schluss werden dann die Ziele des Kurses präzisiert.

1.1 Was ist Excel?

Mit einer kurzen Definition lässt sich nur unzureichend ausdrücken, welche Möglichkeiten Excel einem Softwareentwickler bietet. Excel ist ein äußerst vielseitiges Werkzeug. Man kann darin zwei Bereiche unterscheiden (s. auch Bild):

- Das klassische Excel als Endbenutzersystem, das auch ohne VBA-Programmierung benutzt werden kann
- Das Programmiersystem mit den beiden Kernbestandteilen VBA und Forms und der Entwicklungsumgebung IDE (Integrated Development Environment).



Die einzelnen Bestandteile sind im Folgenden kurz beschrieben:

Excel

Tabellenkalkulation und Präsentationsinstrument
In der einfachsten Form der Benutzung ist Excel ein Instrument zur Auswertung von Daten, gewöhnlich in Tabellenform eingegeben, und zur Präsentation der Ergebnisse.

Einfacher Datenspeicher
In den Tabellenblättern einer Excel-Arbeitsmappe kann man Daten beträchtlichen Umfangs speichern. Die Form der Speicherung ist die der Tabelle, wie bei relationalen Datenbanken. Allerdings bietet Excel nicht dieselben Möglichkeiten, Tabellen zu verknüpfen, so dass bei datenintensiven Anwendungen Zugriff auf Datenbanken notwendig ist.

Formelsprache (Funktionale Programmiersprache)

Auch ohne VBA-Programme zu schreiben, kann man mit Excel viele Aufgaben lösen. Als Basis dienen die vielen eingebauten **Funktionen** (Worksheet Functions), die man in **Formeln** flexibel kombinieren und schachteln kann. Die gebildeten Formeln sind ähnlich aufgebaut wie Ausdrücke in funktionalen Programmiersprachen. Aber obwohl die Excel-Formelsprache sehr mächtige Funktionen enthält, ist sie keine vollständige Programmiersprache.

Programmiersystem

Entwicklungsumgebung IDE

Die Entwicklungsumgebung IDE, manchmal auch (zu) stark vereinfacht nur VB-Editor genannt, bildet die Klammer um das gesamte Programmiersystem. Sie erlaubt unter anderem, Programmtexte einzugeben und zu bearbeiten, Formulare (Bildschirmmasken) anzulegen und in die Programme zu integrieren, Programme zu übersetzen, ablaufen zu lassen und auszutesten.

VBA

VBA (Visual Basic for Applications) ist eine vollständige Programmiersprache, welche sowohl die klassische imperative Programmierung als auch die objektorientierte Programmierung unterstützt.

Forms

Die Komponente Forms erlaubt dem Programmierer, Bildschirmmasken („Formulare“) zu entwerfen und in VBA-Programme einzubinden. Es lassen sich damit Programme realisieren, die als Benutzeroberfläche entweder nur Masken oder eine Kombination aus Masken und Tabellenblättern verwenden.

Verbindende Elemente

Das Konzept der benutzerdefinierten Funktionen (UDF)

Viele der in VBA programmierten Funktionen lassen sich von den Excel-Tabellenblättern aus benutzen. Man bezeichnet solche Funktionen als benutzerdefinierte Funktionen, im Englischen User-Defined Functions (UDF). Solche Funktionen stehen dem Excel-Benutzer innerhalb des Funktionsassistenten zur Verfügung, genau so wie die eingebauten Excel-Funktionen (Worksheet Functions) auch. Durch die Entwicklung von UDF kann die oben erwähnte Excel-Formelsprache fast beliebig erweitert werden.

Excel-Objektmodell

Das Excel-Objektmodell vermittelt den Zugriff auf nahezu alle Excel-Objekte aus VBA heraus. Damit kann man auch aus der Programmiersprache all das gestalten, was der Benutzer direkt am Tabellenblatt gestalten kann. Mit anderen Worten: das Objektmodell „verheiratet“ VBA und Excel.

Ausgehend von den beschriebenen Elementen lassen sich vor allem drei Arten der Aufgabenlösung unterscheiden:

Aufgabenlösung nur mit Endbenutzersystem und Formelsprache
Auch mit dieser einfachsten Form lassen sich umfangreiche Aufgabenstellungen lösen. Die Lösung schwieriger Aufgaben mit Hilfe von geschachtelten Funktionen in der Formelsprache kann allerdings Anforderungen an den Benutzer stellen, die jenen der Programmierung nicht nachstehen.

Aufgabenlösung mit Endbenutzersystem und erweiterter Formelsprache
Bei dieser Form stehen dem Endbenutzer in der Formelsprache nicht nur die eingebauten Funktionen (Worksheet Functions) zur Verfügung, sondern weitere, die als benutzerdefinierte Funktionen entwickelt wurden. Der Benutzer kann diese zusätzlichen Funktionen entweder selbst entwickeln, von Fachleuten entwickeln lassen oder in Form von Funktionsbibliotheken kaufen.

Aufgabenlösung mit Benutzeroberfläche
Hier umfasst die Aufgabenlösung neben dem Programm auch eine Benutzeroberfläche, entweder in Form von Formularen (Bildschirmmasken) oder von besonders gestalteten Arbeitsblättern, oder auch aus einer Kombination aus beidem.

Nur die letzten beiden Arten der Problemlösung erfordern VBA-Programmierung; sie sind der Gegenstand dieses Kurses. Man wird eine dieser Formen verwenden, wenn die erste Form nicht ausreicht. Hierfür kann es unterschiedliche Gründe geben. Besonders häufig sind die beiden folgenden:

- Zur Aufgabenlösung werden weitere Funktionen gebraucht, also solche, die über die eingebauten Funktionen hinausgehen.
- Die zu lösende Aufgabe ist sehr umfangreich. Eine reine Endbenutzerlösung wäre zwar machbar, aber nicht übersichtlich genug und nicht hinreichend wartbar.

1.2 Welche Arten von Software kann man mit Excel/VBA entwickeln? Welche soll man mit Excel/VBA entwickeln?

Da VBA eine vollständige universelle Programmiersprache ist, kann man im Prinzip alle Arten von Software mit Excel-VBA entwickeln. Da es aber eine Reihe anderer leistungsfähiger Programmiersprachen gibt, wie z.B. Java, C++ und VB.net, sollte man vor Beginn einer Entwicklung immer prüfen, welche Sprache bzw. Entwicklungsumgebung für die betreffende Aufgabe am besten geeignet ist.

Gegen Excel/VBA spricht, dass man ein damit erstelltes Programm nur im Rahmen von Excel durchführen kann. Man muss also Excel starten, um das Programm starten zu können. Hinzu kommt, dass VBA aus der Sicht des Software Engineering, insbesondere der Qualitätssicherung, keine ideale Programmiersprache ist (s. hierzu auch Abschnitt 1.5).

Man wird demnach Excel-VBA nur dann einsetzen, wenn es im Zusammenhang mit der jeweiligen Entwicklungsaufgabe wesentliche Gründe dafür gibt. Zwei typische Fälle sind:

- Man erspart sich einen großen Teil des Entwicklungsaufwands durch Nutzung mächtiger Excel- und VBA-Funktionen. Beispiele für solche mächtigen Funktionen sind die Matrixfunktionen, die statistischen Funktionen, die Textbearbeitungsfunktionen und die Funktionen des Solvers.
- Die Benutzeroberfläche von Excel (Arbeitsblätter und Diagramme), lässt sich in der Anwendung gut einsetzen. Dies ist z.B. der Fall, wenn die Anwender große Tabellen pflegen müssen.

Wenn einer der genannten Gründe vorliegt und der Einsatz von Excel-VBA naheliegt, bleibt noch die Frage zu klären, welche der beiden im vorhergehenden Abschnitt beschriebenen Problemlösungsformen man wählt, Endbenutzersystem mit erweiterter Formelsprache oder Programm mit Benutzerführung.

Die Verwendung des Endbenutzersystems mit erweiterter Formelsprache („UDF-Lösung“) erfordert gewöhnlich weniger Entwicklungsaufwand. Außerdem sind Wartbarkeit und die Wiederverwendbarkeit (s. unten: Was ist gute Software?) ohne große Anstrengungen erreichbar, weil das Programm nur aus benutzerdefinierten Funktionen (UDF) besteht. Diese sind völlig in sich geschlossene Einheiten mit klar definiertem Input und Output.

Programme mit Benutzerführung („UI-Lösung“) sind in Bezug auf Wartbarkeit und Wiederverwendbarkeit erheblich problematischer, denn die Benutzerführung ist vorwiegend auf die konkrete Anwendung zugeschnitten. Um Wartbarkeit und Wiederverwendbarkeit herzustellen, müssen die Entwickler im Code das allgemein Verwendbare vom anwendungsspezifischen trennen. Dies ist keine triviale Aufgabe. Während die UI-Lösung bei Entwicklungsaufwand, Wartbarkeit und Wiederverwendbarkeit also im Nachteil ist, hat sie in einem anderen wesentlichen Punkt klare Vorteile gegenüber der UDF-Lösung: Gut programmiert ist sicherer und robuster, weil sie den Benutzer besser vor Fehlern und Irrwegen bewahren kann.

Wir kommen damit zu folgenden Schlüssen hinsichtlich der Eignung der Lösungsformen:

- Je umfangreicher und komplexer die Aufgabe ist, umso mehr eignet sich die UI-Lösung.
- Je besser die Benutzer mit dem Lösungsweg vertraut sind, umso besser eignet sich die UDF-Lösung

Das folgende Bild fasst diese Schlüsse zu einer Anwendungsempfehlung zusammen. Diese ist nur als Tendenzaussage zu verstehen. Eindeutig ist die Sachlage nur in den folgenden Fällen:

- Die Aufgabe ist sehr komplex und die Vertrautheit der Benutzer mit dem Lösungsweg ist sehr gering
- Die Aufgabe ist wenig komplex und die Vertrautheit der Benutzer mit dem Lösungsweg ist sehr groß.

In allen anderen Fällen kommt es auf eine genaue Abwägung der Einflussfaktoren an.

1.3 In welcher Form soll die Software weitergegeben werden?

Programme, welche **in anderen Sprachen** als VBA geschrieben sind, werden gewöhnlich in übersetzter Form (kompiliert) an die Anwender ausgeliefert. Der Quelltext bleibt damit dem Abnehmer verborgen. Das Geheimhalten des Quelltexts dient dabei nicht nur dem Schutz der Investitionen, welche die Anbieter durch Entwicklung der Software getätigt haben, sondern sorgt auch dafür, dass die Software wartbar bleibt. Wenn die Anwender die Möglichkeit hätten, nach Belieben in der gelieferten Software „herumzuprogrammieren“, könnte keine sinnvolle Wartung mehr gewährleistet werden.

Wird die Software weiter entwickelt, so geschieht dies nicht durch die Anwender, sondern die Entwickler. Die Entwickler sorgen auch dafür, dass die neue Fassung mit den bereits vorhandenen Daten kompatibel ist. Kann dies in besonderen Fällen nicht durchgehalten werden, weil die Änderungen zu groß sind, so müssen die Entwickler ein Transferprogramm bereit stellen, mit dessen Hilfe die Daten in die Form überführt werden, welche die neue Software-Version benötigt.

Benutzt man **Excel/VBA** als Entwicklungsumgebung, so ist die Lage etwas komplizierter. Die Software befindet sich nach der Entwicklung in einer Arbeitsmappe, die nicht en bloc übersetzt werden kann. Liefert man die Arbeitsmappe so an die Abnehmer aus, so können diese leicht den Quellcode einsehen und verändern. Ein zweites Problem ergibt sich daraus, dass die Daten der Anwender ebenfalls in diese Arbeitsmappe eingegeben werden. Wenn später eine Änderung der Software notwendig ist, müsste diese Änderung in allen Arbeitsmappen durchgeführt werden, die irgendwo in Gebrauch sind.

Excel bietet verschiedene Speicheroptionen bzw. Formate an, welche diese Probleme ganz oder teilweise lösen sollen:

Schützen des Quellcodes durch ein Passwort

In der ansonsten unveränderten Arbeitsmappe wird der Quellcode durch ein Passwort geschützt, das nur die Entwickler kennen. Programmklau und unkontrolliertes Herumprogrammieren werden gewöhnlich dadurch verhindert. Es bleibt das Problem, dass die Arbeitsmappe Daten und Programm vereint. Änderungen des Programms beim Anwender bleiben, wenn überhaupt möglich, sehr aufwändig und fehleranfällig.

Vorlage (Template)

Eine Vorlage entsteht, wenn man eine Arbeitsmappe in dem für Vorlagen gedachten Format speichert; die Dateien tragen den Zusatz xltx oder, wenn sie VBA-Code enthalten, xltm. Legt man eine neue Arbeitsmappe auf der Basis einer solchen Vorlage an, so übernimmt diese Arbeitsmappe alle Formatierungen und sonstigen Einzelheiten der Vorlage, also auch den darin enthaltenen VBA-Code. Dieser kann auch in der Vorlage durch ein Passwort geschützt werden.

Liefert man die Software in Form einer Vorlage aus, so kann der Abnehmer beliebig viele Arbeitsmappen anlegen, welche die Software enthalten. Gleichzeitig wird die Gefahr gemindert, dass er eine Mappe versehentlich überschreibt. Es bleibt aber das grundsätzliche Problem bestehen, dass Daten und Code in den Arbeitsmappen vereint sind, und der Code nicht mit vertretbarem Aufwand verändert werden kann.

Add-In

Ein Add-In ist eine Datei, die ein Benutzer seinem Excel-System hinzufügen kann, und die diesem System zusätzliche Funktionalität verleiht. Das Hinzufügen geschieht menügesteuert mit Hilfe des Add-In-Managers und ist schnell durchgeführt.

Ein Add-In ist die beste Möglichkeit, mit Excel/VBA gestaltete Software auszuliefern. Die Entwickler programmieren wie gewohnt in einer ganz normalen Arbeitsmappe. Ist die Software fertig gestellt und ausgetestet, so wird sie mit Hilfe eines einfachen Verfahrens in ein Add-In verwandelt. Dieses Add-In kann dann an alle Bezieher der Software verteilt werden und wird von diesen mittels Add-In-Manager installiert. Ein Add-In enthält grundsätzlich keine Daten. Es ist möglich (und üblich), den Quellcode der Software im Add-In gegenüber den Erwerbern zu verbergen und die Software auf diese Weise gegen Missbrauch zu schützen.

1.4 Was ist gute Software?

Etwas pauschal kann man auf diese Frage antworten: die Güte einer Software hängt davon ab, wie gut sie ihren Zweck erfüllt. Vorausschauende und ökonomisch denkende Menschen werden anfügen: sie sollte diesen Zweck über längere Zeit hinweg erfüllen können und sich den Bedürfnissen der Benutzer leicht anpassen lassen, wenn diese sich ändern.

Wenn wir gezielt auf die Softwaregüte einwirken wollen, genügt eine solche pauschale Betrachtung allerdings nicht. Wir müssen den Begriff der Güte in einzelne Komponenten aufspalten, an denen wir uns orientieren können. Man bezeichnet solche Komponenten als **Qualitätsziele** oder **Softwarequalitätsmerkmale**. Bei der folgenden Betrachtung dieser Komponenten setzen wir stillschweigend voraus, dass die Software die gewünschte **Funktionalität** besitzt. Ohne diese Funktionalität nützen alle Qualitätsmerkmale nichts. Ein Benutzer, der ein Programm zur Verwaltung seiner Briefmarkensammlung wünscht, wird mit einem Programm zur Verwaltung des Weinkellers nicht zufrieden sein, auch wenn es noch so gut funktioniert. Nun aber die wichtigsten Qualitätsziele:

Benutzerfreundlichkeit

Ein Programm ist benutzerfreundlich, wenn der Benutzer damit bequem arbeiten kann. Er soll in jeder Phase seiner Arbeit genau wissen, was das Programm von ihm erwartet. Ein gutes Benutzerhandbuch hilft anfangs dabei, aber im Idealfall ist ein Programm so weit selbsterklärend, dass der Benutzer nach kurzer Einarbeitungszeit auch so zurechtkommt. Daneben soll die Software auch ästhetischen Ansprüchen genügen: die Benutzeroberfläche soll sachdienlich und harmonisch aufgebaut sein, so dass es ein Vergnügen ist, damit zu arbeiten.

Zuverlässigkeit

Wir nennen ein Programm zuverlässig, wenn Fehler selten auftreten und nur geringe Auswirkungen haben. Zuverlässigkeit ist natürlich besonders von korrekter Programmierung abhängig, jedoch sind **Korrektheit** und Zuverlässigkeit nicht dasselbe. Zum einen braucht es für eine hinreichende Zuverlässigkeit keine völlige Korrektheit (man erreicht sie ohnehin so gut wie nie), zum anderen wäre auch völlige Korrektheit nicht ausreichend, um Zuverlässigkeit zu gewährleisten. Wichtig ist, dass das Programm in seinen wesentlichen Funktionen korrekt arbeitet. Neben dem Aspekt der Korrektheit ist die **Robustheit** der zweite wichtige Aspekt der Zuverlässigkeit. Ein Programm sollte auf keine Eingabe „bösartig“ reagieren, also z.B. mit einem Absturz, so unsinnig die Eingabe des Benutzers auch sein mag. Bessere Reaktionen auf Fehleingaben sind aussagefähige Fehlerhinweise, verbunden mit der Möglichkeit, die Eingaben zu wiederholen.

Wiederverwendbarkeit

Damit ist gemeint, dass man Teile der Software herauslösen und im Rahmen anderer Software verwenden kann. Offensichtlich ist dies am ehesten möglich, wenn die Software aus in sich geschlossenen Bausteinen zusammengesetzt ist. Man spricht dann auch von einem **modularen Aufbau**. Ein modularer Aufbau harmoniert im Übrigen gut mit einer **Schichtenarchitektur** (s. dort). Eine spezielle Variante der Wiederverwendbarkeit liegt vor, wenn das gesamte Programm in einer anderen Umgebung verwendet werden soll als ursprünglich vorgesehen. Man bezeichnet dies als **Portabilität**. Auch die Portabilität profitiert von einem modularen Aufbau.

Wartbarkeit

Nicht nur Geräte müssen gewartet werden, sondern auch Programme, obwohl sie sich im Gegensatz zu ersteren nicht im Laufe der Zeit abnutzen. Anlass für die Softwarewartung sind zum einen die Änderungswünsche der Benutzer, zum anderen die Programmierfehler, die nach und nach zu Tage treten. Wichtige Bestimmungsfaktoren der Wartbarkeit sind Architektur, modularer Aufbau, Entwicklungsdokumentation und Art der Codierung (Kommentierungen, Übersichtlichkeit).

Effizienz

Wenn man von Effizienz einer Software spricht, meint man vor allem ihre **Schnelligkeit**. Ein langsames Programm wird besonders zum Ärgernis, wenn der Benutzer eines Dialogprogramms lange vor dem Bildschirm sitzt, bevor er die Rückmeldung auf seine Aktionen sieht. Dies kann am Ende dazu führen, dass ein solches Programm von den Benutzern nicht akzeptiert wird.

Aber auch wenn kein Benutzer vor dem Bildschirm sitzt, um auf die Antwort zu warten, kann Langsamkeit ein Hinderungsgrund sein, ein Programm einzusetzen. Dies gilt vor allem, wenn aufwändige Rechnungen auf einem großen Datenumfang durchzuführen sind.

Manche Programme kann man als Programmierer dramatisch beschleunigen, indem man den Großteil der Daten ständig im Arbeitsspeicher hält, anstatt die benötigten Daten erst bei Bedarf vom Externspeicher zu laden. Man fördert in diesem Fall die Geschwindigkeit zu Lasten des Arbeitsspeicherbedarfs. Im Einzelfall muss man in solchen Fällen prüfen, was wichtiger ist, Geschwindigkeit oder geringer Speicherbedarf.

1.5 VBA und Softwarequalität

Gibt es bei VBA im Hinblick auf die Softwarequalität besondere Dinge zu berücksichtigen, welche in anderen Programmiersprachen nicht relevant sind? Ja, die gibt es. Wir können diese besonderen Probleme vor allem an zwei Dingen festmachen:

VBA ist nicht die ideale Programmiersprache

VBA ist keine Sprache aus der Retorte, sondern eine gewachsene Sprache, die über Generationen von Office-Paketen immer wieder erweitert und verändert worden ist. Deshalb sind in VBA Sprachelemente enthalten, die ganz unterschiedliche Philosophien der Programmierung verkörpern. Am Anfang versuchte man, es dem Programmierer so leicht wie möglich zu machen. Man wollte ihm nicht einmal den Zwang zu präzisem Ausdruck auferlegen, ja machte ihm sogar einen solchen präzisen Ausdruck unmöglich (die Benutzung desselben Symbols für Zuweisung und Gleichheitsprüfung ist ein besonders krasses Beispiel dafür). Später kam dann mehr Präzision in die Sprache, aber die Relikte der Anfangszeit sind im Hinblick auf die Kompatibilität geblieben. Die alten Programme sollen auch auf den neuen Office-Versionen noch funktionieren.

Interaktion mit Office

Der einzige Grund, mit VBA zu arbeiten, ist die Nutzung der mächtigen Ressourcen von Office. Diese Nutzung hat jedoch bedeutsame Konsequenzen:

- Der Entwickler formt sein System von zwei Enden her: durch VBA-Programmierung und durch menügesteuerte Festlegungen in Office. Er muss in beiden Bereichen gestalten, beide Bereiche aufeinander abstimmen, und er muss (müsste) in seiner Dokumentation beide Bereiche berücksichtigen.
- Der Zustand von Office kann das Funktionieren des VBA-Programms beeinträchtigen. Dies ist insbesondere dann der Fall, wenn der Programmierer beim Formulieren von Anweisungen einen bestimmten Zustand stillschweigend voraussetzt und dieser nicht gegeben ist. Beispiel: Zugriff auf das aktive Excel-Tabellenblatt. Ist gerade ein anderes Tabellenblatt aktiv als das gewünschte, so misslingt der Zugriff.
- Der Zugriff von VBA auf Office geschieht über die sog. Objektmodelle. Diese Objektmodelle sind sehr umfangreich, teilweise schlecht dokumentiert, und sie besitzen zum Teil auch einen recht barocken Aufbau. Dies erklärt im Übrigen auch den Erfolg der Bücher, in denen Office-Gurus ihre intimen Kenntnisse der Objektmodelle in Form von „Tricks“ anbieten.

1.6 Lernziele

Es geht in diesem Kurs nicht darum, dass Sie alle Feinheiten der Programmiersprache VBA kennenlernen, sondern darum, dass Sie lernen, gute Programme zu schreiben, also solche, die benutzerfreundlich, zuverlässig, effizient und gut wartbar sind und die einen hohen Grad an Wiederverwendbarkeit aufweisen. Das ist auch mit begrenzten Kenntnissen der Sprache möglich. Wer sich die Prinzipien guter Programmierung zueigen gemacht hat, kann fehlende Details nachschlagen. In umgekehrter Richtung funktioniert es nicht. Es gibt Programmierer zuhauf, die trotz umfangreicher VBA-Kenntnisse fürchterliche Programme schreiben.

Was Sie erreichen, hängt auch davon ab, ob Sie gefestigte Grundkenntnisse der Programmierung haben bzw. bereit sind, sich diese in kurzer Zeit anzueignen. Und natürlich von Ihrem Interesse. Im Idealfall können Sie nach diesem Kurs auch größere Anwendungsprogramme so gestalten, dass Sie sich vor einem Profi nicht zu schämen brauchen.