Irene Weber

VBA für Office-Automatisierung und Digitalisierung

Springer Vieweg

VBA für Office-Automatisierung und Digitalisierung

Irene Weber

VBA für Office-Automatisierung und Digitalisierung

Springer Vieweg

Irene Weber
Hochschule für Angewandte Wissenschaften
Kempten
Kempten, Deutschland

ISBN 978-3-658-42716-0 ISBN 978-3-658-42717-7 (eBook)
https://doi.org/10.1007/978-3-658-42717-7

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://portal.dnb.de abrufbar.

Planung/Lektorat: Petra Steinmueller
Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Fachmedien Wiesbaden GmbH und ist ein Teil von Springer Nature.
Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

Das Papier dieses Produkts ist recycelbar.

Vorwort

Den Anlass dafür, dieses Buch zu schreiben, gaben meine VBA-Kurse in der Fakultät Maschinenbau an der Hochschule für Angewandte Wissenschaften Kempten. Als ich VBA in das Lehrprogramm aufnahm, war das Interesse an Programmierung und Informatik-Themen allgemein in der Studierendenschaft noch nicht besonders groß und viele Studierende befassten sich in der Lehrveranstaltung nur widerstrebend damit. Wenige Semester später, als die ersten aus den Industriepraktika zurückkehrten, war das Feedback dann sehr positiv. Schon mit den Grundkenntnissen aus einem einsemestrigen VBA-Kurs waren Studierende in der Lage, wertvolle Beiträge im Unternehmen zu leisten. Seitdem haben viele unserer Studierenden VBA in Studienprojekten, Industriepraktika, Abschlussarbeiten und als Werkstudenten erfolgreich eingesetzt und Projekte damit realisiert. Diese Erfahrungen zeigen den praktischen Nutzen, den VBA in der Industrie hat. Sowohl die Erfolge wie auch die Schwierigkeiten, die in den Berichten von Studierenden zum Ausdruck kamen, haben zur Themenauswahl dieses Buchs angeregt.

Einige Studierende vertiefen ihre VBA-Kenntnisse in einem fortgeschrittenen Kurs. Hier entwickeln sie Lösungen für echte Anwendungssituationen, die sie aus der Praxis als Werkstudierende, Selbständige und oft auch aus dem Vereinsehrenamt mitbringen. So entstehen nützliche, auf die jeweiligen Aufgaben spezialisierte Verwaltungstools und Automatisierungen. Andere befassen sich aus persönlichem Interesse mit innovativen und experimentellen Aufgabenstellungen. Um einige Beispiele nennen: J. Schädler hat die CAD-Software CATIA automatisiert. M. Hoffmann steuert seine automatische Hobby-Bierbrauanlage aus einem in Excel realisierten Rezeptkonfigurator, der ein Ablaufprogramm generiert und außerdem Rezepte, Vorräte und Bierbewertungen verwaltet und Flaschenetiketten druckt. S. Benischke und M. Lankau haben ein legendär schönes Brettspiel mit exzellenter Usability auf Excel-Basis realisiert. Ein Routenplaner, der die OpenRouteService-API verwendet und auch Fahrtkosten berechnet, stammt von A. Zipfel. J. Raffler hat experimentell die Möglichkeiten und Grenzen der Integration von Excel und Python evaluiert. L. Grotz hat ein professionelles Tool zur Berechnung der nötigen Kollektorfläche für die Beheizung von Freischwimmbädern entwickelt, das die monatlichen Durchschnittstemperaturen für den geplanten Pool-Standort und Globalstrahlungsdaten aus Webdiensten importiert. E. Altun hat das Tool „Digitaler Urlaubsantrag®“

eingebracht. Mit der Nutzung der PayPal-API aus VBA hat sich M. Nakov befasst. V. Koffler ist es gelungen, eine für Experimente benötigte, über USB angeschlossene Konstantstromquelle per VBA zu konfigurieren, Rückgabewerte auszulesen und diese in einem Exceldiagramm in Echtzeit zu visualisieren. T. Wolfram und M. Heitzer haben sich daran gemacht, zu einem Reiseziel und Erlebniswünschen per VBA eine Broschüre mit einem Reisevorschlag zu generieren. Der SOLIDWORKS Schraubengenerator von M. Probst erzeugt aus einem Satz Parameter auf Knopfdruck ein normgerechtes Modell einer Schraube. Y. Üzüm hat gezeigt, wie gut eine auf einem Excel-Arbeitsblatt gestaltete Bedienoberfläche aussehen und funktionieren kann. Danke für Euer Engagement und Euren Ideenreichtum! Mit diesem Buch möchte ich einen Beitrag dazu leisten, dass weitere solche Projekte schnell und mit wenig Mühe in guter Qualität verwirklicht werden.

Gosheim Irene Weber
im Dezember 2023

MARKEN/WARENZEICHEN DRITTER

Microsoft, Microsoft Office, Microsoft Word, Microsoft PowerPoint, Microsoft Excel, Microsoft Outlook sind Warenzeichen oder eingetragene Warenzeichen der Microsoft Corporation. SAP® ist eine Marke oder eingetragene Marke der SAP SE oder ihrer verbundenen Unternehmen in Deutschland und anderen Ländern. SOLIDWORKS™ ist eine Marke von Dassault Systèmes oder ihrer verbundenen Unternehmen in Frankreich und in anderen Ländern.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Einleitung

1

Digitalisierung ist in vielen Bereichen des privaten und öffentlichen Lebens angekommen. Für Unternehmen ist Digitalisierung wichtig, um wettbewerbsfähig zu bleiben und die Anforderungen des Marktes und der Kunden zu erfüllen. Auch Mitarbeiter stellen wachsende Ansprüche an die digitalen Werkzeuge an ihrem Arbeitsplatz. Wirksame digitale Tools reduzieren die Arbeitsbelastung, machen die Arbeit produktiver und letztendlich zeigen sie auch, dass die Leistung der Mitarbeiter wertgeschätzt wird. Die Unternehmen treiben Digitalisierung voran, doch oft bremsen Kapazitätsengpässe in den IT-Abteilungen den Fortschritt. Eine Lösung stellt hier das End User Computing dar: Die User selbst entwickeln Anwendungen und Tools, um sie später für ihre Arbeit zu nutzen. Visual Basic for Applications (VBA) ist seit Langem bekannt und bewährt als Werkzeug, mit dem Endnutzer Lösungen entwickeln, um ihre eigene Arbeit und die Arbeit der Kollegen in den Fachabteilungen komfortabler und effizienter machen. Damit befasst sich „VBA für Office-Automatisierung und Digitalisierung".

VBA ist eine vollwertige Programmiersprache, die meist innerhalb einer bestehenden Anwendungssoftware, der sogenannten Wirtsanwendung, verwendet wird. Daraus leiten sich viele ihrer Stärken ab. Mit VBA entwickelte Lösungen bauen auf die Wirtsanwendungen auf und können mit deren Funktionen, Bedienoberflächen und Datenspeichermechanismen arbeiten. Dies spart, im Vergleich zu völlig neu entwickelten Lösungen, sehr viel Entwicklungsaufwand. Der Einstieg in die VBA-Entwicklung ist unaufwendig, denn mit den Wirtsanwendungen ist auch VBA bei Entwicklern und Nutzern von VBA-basierten Lösungen bereits installiert und ohne zusätzliche Kosten direkt verwendbar. Da die Nutzer von VBA-Lösungen mit der Wirtssoftware vertraut sind, lernen sie meist schnell, mit solchen Tools umzugehen.

Zudem macht VBA es leicht, mehrere VBA-fähige Anwendungssysteme zusammenarbeiten zu lassen. Dadurch erschließen sich weitreichende Möglichkeiten, denn neben der Office-Software von Microsoft sind auch viele Softwarepakete anderer Hersteller mit

I. Weber, *VBA für Office-Automatisierung und Digitalisierung,*
https://doi.org/10.1007/978-3-658-42717-7_1

VBA ausgestattet. VBA bietet damit das Potenzial, effektive Digitalisierungslösungen mit geringem Aufwand zu realisieren. Diese können auch größere, von der Unternehmens-IT gesteuerte Digitalisierungsprojekte als Prototypen vorbereiten, sie ergänzen oder die Zeit bis zu ihrer Fertigstellung überbrücken.

Besonders häufig wird VBA im Unternehmenskontext zusammen mit Microsoft Excel eingesetzt. Demzufolge sind viele Beispiele und Informationsquellen zu Excel-VBA zu finden. „VBA für Office-Automatisierung und Digitalisierung“ zeigt dagegen Methoden und gibt Anregungen für Digitalisierungslösungen mit VBA über Excel hinaus. Ein Schwerpunkt liegt auf der Software-Integration, also der gleichzeitigen Verwendung mehrerer Softwarepakete in einer VBA-Lösung. Ein eigenes Kapitel befasst sich damit, wie VBA-Lösungen auch Cloud-basierte Dienste nutzen können.

Auch wenn sie durch eine gemeinsame Sprache VBA angesteuert werden, bringen die verschiedenen Anwendungssysteme doch meist eine eigene innere Logik mit, die jeweils ein etwas anderes Herangehen beim Programmieren erfordert. „VBA für Office-Automatisierung und Digitalisierung“ führt in das Objektmodell des jeweiligen Softwaresystems ein und machst seine innere Logik verständlich. Es erläutert die VBA-Entwicklung an kurzen Code-Beispielen und an etwas umfangreicheren Demo-Anwendungen. Da End User Computing nicht nur Vorteile hat, sondern auch Nachteile und Risiken mit sich bringen kann, nennt „VBA für Office-Automatisierung und Digitalisierung“ Maßnahmen, um diese zu minimieren, und thematisiert auch Aspekte der IT-Sicherheit.

Einleitende Kapitel führen in Visual Basic for Applications und in objektbasiertes Programmieren ein und geben einen Einblick in die Welt der Programmiersprachen. Auch in den folgenden Kapiteln sind immer wieder Infos und Tipps zu bewährten Programmierpraktiken zu finden.

Code. Die Code-Beispiele und Demo-Anwendungen sind online in einem Github-Repository verfügbar: https://github.com/weberi/vba-buch. Sie sind unter Windows 10 und mit Office 2016 entwickelt und zum großen Teil auch unter Office 365 getestet. Alle Code-Beispiele sind dafür ausgewählt und konzipiert, nützlich und gleichzeitig leicht verständlich zu sein. Dazu gehören auch Kürze und Übersicht. Die Behandlung von Fehlern ist daher oft nur angedeutet. Der Einsatz von Makros, Automatisierungen und IT ganz allgemein bringt Risiken mit sich, die von der jeweiligen konkreten Verwendung und der Umgebung abhängen. Weder Verlag noch Autorin geben irgendeine Zusicherung oder Gewähr, dass die Code-Beispiele wie erwartet funktionieren, sich für einen bestimmten Anwendungszweck eignen oder einen bestimmten konkreten Nutzen stiften. Die Verwendung aller Code-Beispiele, die nicht bereits frei verfügbar sind, wird unter der Unlicense gestattet, erfolgt aber auf eigenes Risiko.

Anmerkungen zur Sprache. Viele Informationen aus dem Bereich der IT sind auf Englisch geschrieben. Auch die Programmiersprache VBA selbst besteht weitgehend aus englischen Wörtern. Ins Deutsche übersetzte Fachausdrücke lassen sich oft den englischen Originalbegriffen nur schwer wieder zuordnen. Dies führt zu Missverständnissen und erschwert weiterführende Recherchen im Netz. Das Buch verwendet deshalb häufig englische Begriffe im deutschen Text. Es soll auch ohne vertiefte IT-Kenntnisse gut zu

lesen und verständlich sein und verzichtet deshalb möglichst auf Fachjargon. Der Text ist im generischen Maskulin verfasst. Verweise auf Webseiten, zum Beispiel mit Dokumentation oder Downloads, sind unter den Literaturangaben aufgeführt, um lange URLs im Text zu vermeiden.

Einführung in VBA

2

Inhaltsverzeichnis

I. Weber, *VBA für Office-Automatisierung und Digitalisierung*,
https://doi.org/10.1007/978-3-658-42717-7_2

Im Lauf der Jahrzehnte hat die Informatik viele Programmiersprachen entwickelt. Je nach Kontext und Anwendungsbereich sind unterschiedliche Programmiersprachen populär und gebräuchlich. Dabei unterscheiden sich die sprachlichen Grundelemente der verbreiteten Programmiersprachen nur wenig. In der Regel bestimmen andere Faktoren, welche Programmiersprache in einem Projekt zum Einsatz kommt: die Verfügbarkeit geeigneter vorgefertigter Programmbausteine und Code-Bibliotheken, vorhandene Entwicklungswerkzeuge und Entwicklungs-Infrastruktur und nicht zuletzt das Know-How und die Vorlieben der Entwickler. Dieses Kapitel ordnet VBA in das Spektrum der Programmiersprachen ein und gibt dann einen Einstieg in die Praxis des Programmierens mit VBA: wie und womit man VBA-Programme erstellt, wie sie aussehen und wie man sie ausführt und speichert. Dies schafft die Voraussetzungen dafür, Code-Beispiele praktisch umzusetzen und auszuprobieren. Der anschließende Überblick über die wichtigsten Sprachelemente von VBA richtet sich an Programmieranfänger wie auch an erfahrene Softwareentwickler, die sonst mit anderen Programmiersprachen arbeiten. Sie schafft die Grundlage dafür, die Code-Beispiele zu verstehen, anzupassen und eigene Programme zu erstellen.

2.1 Die Programmiersprache VBA

VBA ist die Abkürzung von Visual Basic for Applications. BASIC steht für „Beginner's All-purpose Symbolic Instruction Code". Es wurde 1964 als besonders leicht zu erlernende Sprache entwickelt mit dem Ziel, sie in Programmierkursen einzusetzen. Wie viele verbreitete Programmiersprachen ist auch BASIC eine imperative Programmiersprache. Ihre Grundbausteine sind Befehle, mit denen Programmierer angeben, was ein Programm tun soll. Nach 1964 wurde BASIC ständig weiterentwickelt. 1991 führte

Microsoft die Programmiersprache Visual Basic ein, mit der man Anwendungen für das Betriebssystem Windows programmieren kann. Im Jahr 1995 begann Microsoft, in seinen Office-Programmen die bis dahin verwendeten Makrosprachen durch Visual Basic for Applications zu ersetzen.

Microsoft Visual Basic for Applications gehört zu den interpretierten Programmiersprachen, im Unterschied zu den kompilierten Programmiersprachen. Interpretierte und kompilierte Programmiersprachen unterscheiden sich darin, wie und wann aus dem Quellcode eines Programms maschinenausführbarer Code wird. Bei einer kompilierten Programmiersprache erzeugt ein Compiler (Übersetzer) eine Datei mit maschinenausführbarem Code, die dann auf Rechnern mit passendem Betriebssystem immer wieder ausgeführt werden kann. Eine interpretierte Programmiersprache benötigt statt eines Compilers einen Interpreter, und zwar bei jeder Ausführung eines Programms. Der Interpreter liest die Befehle als Quellcode ein und führt sie sofort aus.

Interpretierte Programmiersprachen bezeichnet man auch als Skriptsprachen und die damit erstellten Programme als Skripte. Interpretierte Programme laufen langsamer als übersetzte Programme, doch sie sind einfacher zu verbreiten und anzuwenden, weil der Übersetzungsvorgang entfällt und ein Skript überall, wo ein passender Interpreter installiert ist, sofort einsatzfähig ist. Visual Basic for Applications ist in Microsoft Word, Excel, Project, PowerPoint, Access, Visio, Outlook und viele weitere Software-Produkte anderer Hersteller integriert, vor allem in Geschäfts- und Ingenieursanwendungen. Mit diesen Softwaresystemen ist zugleich auch ein Interpreter für VBA auf einem Computer installiert. Mit VBA entwickelt man meist keine eigenständigen Programme, sondern Programmteile, die innerhalb eines dieser Anwendungssoftwaresysteme laufen, es ansteuern und seine Funktionalitäten nutzen. Das umgebende Softwaresystem (die „Application“) bezeichnet man als Wirtsanwendung oder auch Host Application. Neben der jeweiligen Wirtsanwendung lassen sich zugleich auch weitere VBA-fähige Softwaresysteme mit einem Skript ansteuern, sodass diese Systeme nahtlos zusammenarbeiten und Daten austauschen können.

Die technische Basis dafür ist das Component Object Model (COM), das von Microsoft entwickelt und 1992 in das Betriebssystem Windows integriert wurde. COM ist eine objektorientierte Softwarearchitektur, die die Funktionalität von Softwarekomponenten in Objekten kapselt. COM ist plattformunabhängig, was bedeutet, dass die Softwarekomponenten mit beliebigen Programmiersprachen und Entwicklungswerkzeugen entwickelt sein können. Sie werden als Bibliotheken oder als ausführbare Programme bereitgestellt. Die darin implementierten Objekte machen ihre Funktionen über Schnittstellen verfügbar und können von anderen Programmen so aufgerufen und genutzt werden, als seien sie Teil des aufrufenden Programms. Dies ist mit mehreren Programmiersprachen möglich, darunter C++, C#, Delphi, Java, Python und natürlich VBA.

VBA wird oft als objekt*basierte* Programmiersprache bezeichnet. Charakteristisch für objekt*orientierte* Programmierung ist, dass sie das entwickelte Softwaresystem durch Klassen und Objekte abbildet, wobei sich die Objekte aus Klassen ableiten. VBA unterstützt es sehr gut, eigene Klassen zu programmieren und daraus Objekte zu erzeugen.

Für einige weiterführende Programmierkonzepte der objektorientierten Programmierung wie Vererbung oder Abstraktion stellt VBA jedoch keine speziellen Sprachmittel zur Verfügung. Während in objektorientierten Programmiersprachen wie Java, C++ oder auch Visual Basic (ohne „Applications“) solche Programmierkonzepte sehr einfach umgesetzt werden können, lassen sie sich in VBA nur umständlich mit anderen Befehlen und Konstruktionen nachbilden. Insofern grenzt sich VBA als objektbasierte Programmiersprache von objektorientierten Programmiersprachen ab. Bei der Entwicklung mit VBA entwickelt man jedoch eher selten eigene Klassen, sondern arbeitet mit den Klassen und Objekten, die die Wirtsanwendungen mitbringen.

Weiterhin unterstützt VBA die ereignisgesteuerte Programmierung. Ereignissteuerung ist wie Objektorientierung ein Programmieransatz, der es ermöglicht, die komplexen Softwaresysteme zu realisieren, die es heute gibt. Statt eine zentrale Ablaufsteuerung zu implementieren, verteilt ereignisgesteuerte Programmierung Zuständigkeiten auf einzelne Softwarekomponenten, in einer objektorientierten Programmiersprache typischerweise auf Objekte. Ereignisse sind dabei zum Beispiel Benutzeraktivitäten wie etwa ein Mausklick auf eine Befehlsschaltfläche oder eine Dateneingabe in ein bestimmtes Eingabefeld, aber auch Nachrichten, die Systemkomponenten untereinander austauschen. Zuständige Softwarekomponenten werden über ein Ereignis informiert und verfügen über Prozeduren und Methoden, um es zu verarbeiten.

VBA ist eine mächtige Programmiersprache mit allen Möglichkeiten, die man von einer modernen, allgemein einsetzbaren Programmiersprache erwarten kann. Sein Sprachkern ist gleichwertig zu dem von C, Python, PhP, C++, Java und so weiter, wobei jede dieser Programmiersprachen ihre speziellen Stärken mitbringt und sich für bestimmte Anwendungszwecke besonders eignet. Für umfangreiche und komplexe Softwareprojekte, bei denen vielleicht sogar mehrere Entwickler mitarbeiten, eignet sich VBA weniger. Die Stärke von VBA ist seine Integration als Makrosprache in funktionsmächtige Anwendungssoftwaresysteme aus dem Office-, Business- und technischen Bereich. VBA eignet sich sehr gut dafür, diese Softwaresysteme zu erschließen, sie zu automatisieren und eigene Tools für spezifische Aufgaben darauf aufzusetzen.

2.2 VBA-Programme entwickeln und ausführen

Ein großes Plus von VBA ist die besonders niedrige Einstiegshürde in die Software-Entwicklung. VBA macht es sehr einfach, Programme zu erstellen und auszuführen. Dazu trägt auch die integrierte Software-Entwicklungsumgebung (IDE) bei, die VBA mitbringt. Im Vergleich zu mächtigen, professionellen Software-Entwicklungswerkzeugen wie Microsoft Visual Studio oder Eclipse hat sie einen geringen Funktionsumfang. Gerade das macht sie aber auch übersichtlich und leicht zu bedienen. Sie lässt sich in Verbindung mit Excel, Word und vielen weiteren VBA-fähigen Anwendungen nutzen. Die jetzt folgenden Erklärungen zur VBA-Entwicklung gelten für Excel als Wirtsanwendung und funktionieren weitgehend genauso mit Word und anderen Office-Wirtsanwendungen.

2.2.1 Die Entwicklungsumgebung

Die VBA-Entwicklungsumgebung lässt sich in Excel mit **Alt-F11** oder aus dem Menüband öffnen. Der Befehl heißt **Visual Basic** und ist auf der Registerkarte **Entwicklertools** zu finden, sofern die **Entwicklertools** aktiviert sind. Falls die Registerkarte **Entwicklertools** nicht im Menüband erscheint, kann man sie sichtbar machen, indem man auf der Registerkarte **Datei** den Menüpunkt **Optionen** anklickt und im sich öffnenden **Optionen**-Dialogfenster den Eintrag **Menüband anpassen** wählt. Dann hakt man unter **Hauptregisterkarten** das Kontrollkästchen **Entwicklertools** an und verlässt das Dialogfenster mit **Ok**, siehe Abb. 2.1.

Abb. 2.2 zeigt eine Ansicht der VBA-Entwicklungsumgebung mit vier Fenstern. Weitere oder andere Fenster können im DropDown-Menü **Ansicht** sichtbar gemacht werden. In Abb. 2.2 ist rechts das große **Code**-Fenster zu sehen. Das Fenster darunter ist der **Direktbereich.** In der linken Spalte sind der **Projekt-Explorer** und darunter das **Eigenschaften**-Fenster angeordnet.

Der **Projekt-Explorer** listet die Komponenten von VBA-Projekten auf. Bei der VBA-Entwicklung beginnt man meist damit, in den **Projekt-Explorer** zu klicken und im Kontextmenü **Einfügen > Modul** zu wählen. Der **Projekt-Explorer** legt dann ein neues Modul an und öffnet es im **Code**-Fenster. Ein solches Modul wird als Code-Modul oder Standard-Modul bezeichnet. Hier kann man VBA-Code eingeben. Das Modul „Modul1“ in Abb. 2.2 enthält bereits ein kleines Skript mit einer Sub `HalloWelt`. Im Feld **(Name)** des **Eigenschaften**-Fensters lässt sich das Modul umbenennen.

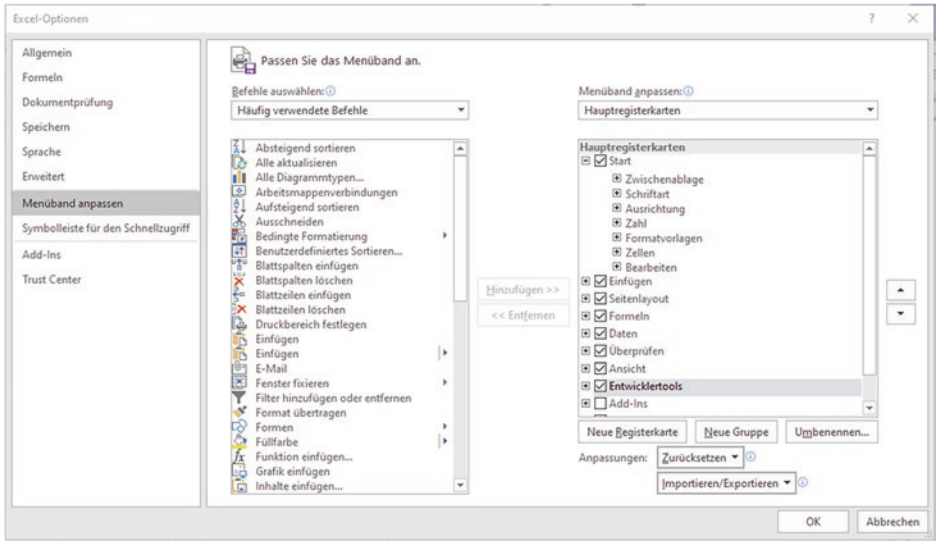

Abb. 2.1 Aktivieren der Entwicklertools in den Excel-Optionen

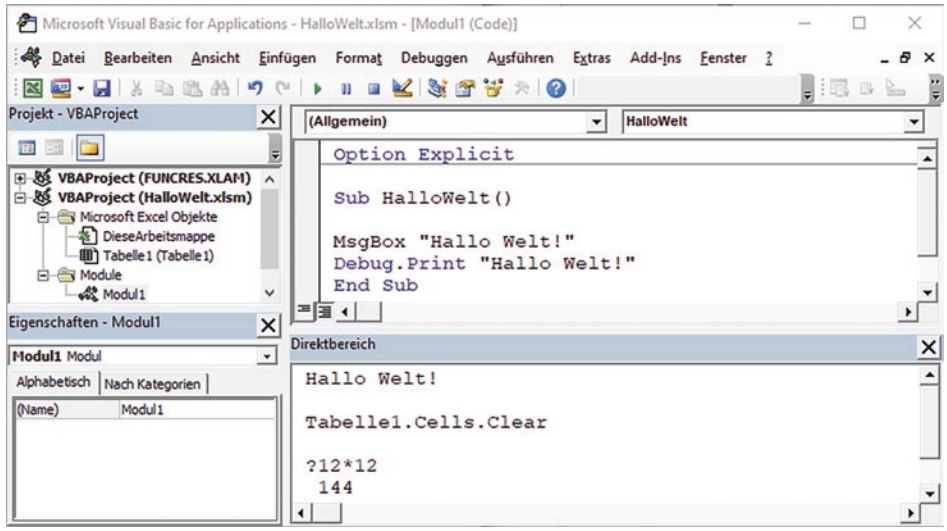

Abb. 2.2 Die VBA-Entwicklungsumgebung

Wenn man eines der **Microsoft Excel Objekte** im **Projekt-Explorer** anklickt (zum Beispiel „Tabelle1“ oder „DieseArbeitsmappe“), öffnet sich ebenfalls ein Modul, ein sogenanntes Objektmodul. Auch ein Objektmodul kann Code aufnehmen. Es ist jedoch nicht für allgemein verwendete Programmteile vorgesehen, sondern für objektspezifischen Code. Weitere Infos zu Modulen gibt Abschn. 2.7.4.

Der **Direktbereich** dient als Eingabe- und Ausgabefenster für den VBA-Interpreter. Hier erscheinen Ausgaben von `Debug.Print`-Befehlen. In Abb. 2.2 zeigt der Direktbereich oben die Ausgabe des `Debug.Print`-Befehls aus der Sub `HalloWelt`. Wenn man einen Befehl in den **Direktbereich** schreibt und mit der Eingabetaste abschließt, führt ihn der VBA-Interpreter sofort aus. Hier kann man Befehle ausprobieren, den Interpreter einen Ausdruck auswerten lassen oder Aktionen in Excel ausführen, zum Beispiel schnell Zelleninhalte und Formatierungen von einem Arbeitsblatt löschen, die beim Entwickeln und Testen eines Skripts entstanden sind. Um einen Ausdruck auszuwerten, tippt man ein Fragezeichen und den Ausdruck (zum Beispiel „12*12“) in den Direktbereich und schließt mit der Eingabetaste ab.

Nützlich sind auch das **Lokal**-Fenster und der **Objektkatalog,** die beide in Abb. 2.2 nicht sichtbar sind. Sie lassen sich im DropDown-Menü **Ansicht** öffnen. Das **Lokal**-Fenster zeigt Variablen und ihren Wert, während der Interpreter ein Skript ausführt. Der **Objektkatalog,** der auch mit der Funktionstaste **F2** zu öffnen ist, listet die jeweils verfügbaren Objekte auf. Er überlagert das **Code**-Fenster. Die Funktionstaste **F7** bringt das **Code**-Fenster wieder nach oben.

2.2.2 VBA-Code schreiben

VBA ist eine zeilenorientierte Sprache. Während andere Programmiersprachen das Ende eines Befehls zum Beispiel mit einem Strichpunkt „;“ markieren, endet ein VBA-Befehl mit der Zeile, in der er steht. Sehr lange Befehle lassen sich auf mehrere Zeilen umbrechen, indem man eine Zeile mit einem Leerzeichen und einem darauffolgenden Unterstrich beendet und den Befehl auf der folgenden Zeile fortsetzt. Getrennt von einem Doppelpunkt lassen sich mehrere Befehle in eine Zeile schreiben. Man sieht diese Schreibweise gelegentlich bei Variablendeklarationen mit anschließender Initialisierung der Variablen. Sie macht den Code kompakter, aber auch weniger übersichtlich. Der folgende Code-Schnipsel zeigt Beispiele für VBA-Befehle.

```
Sub SyntaxDemo()

Dim i As Integer: i = 1
Dim info As String

info = "ich bin ein ziemlich " _
  & "langer Infotext."

' Debug.Print info   ' auskommentiert!
Debug.Print info     ' nicht auskommentiert!
End Sub
```

Kommentare sind Beschreibungen, Erklärungen, Notizen oder auch ungenutzte Code-Teile, die bei der Ausführung des Programms ignoriert werden sollen. Kommentare markiert man in VBA durch ein Hochkomma. Das Hochkomma kann am Anfang einer Zeile stehen oder auch hinter einem Befehl. Alle Zeichen, die in der betreffenden Zeile auf das Hochkomma folgen, sind Kommentar. VBA kennt keine weiteren Möglichkeiten, Text als Kommentar auszuzeichnen. Wenn man mit Kommentarzeichen vorübergehend Code-Teile „stilllegt“, nennt man dies auch „Auskommentieren“. Auskommentieren hilft bei der Programmentwicklung, um Fehler einzugrenzen, noch nicht vollständig programmierte Code-Teile zu überspringen oder um beim testweisen Programmausführen langdauernde, unkritische Berechnungen auszulassen.

2.2.3 VBA-Code in der Entwicklungsumgebung ausführen

Alle VBA-Befehle in einem Code-Modul müssen in einer Prozedur stehen, also zwischen `Sub` und `End Sub`. Prozeduren wie die beiden Code-Beispiele `HalloWelt` in

Abb. 2.2 und die eben gezeigte `SyntaxDemo` kann man in der Entwicklungsumgebung direkt ausführen. Dazu platziert man den Cursor in den Bereich zwischen `Sub` und `End Sub` und drückt die Funktionstaste **F5** oder wählt den Befehl **Sub/UserForm ausführen** im Menüband der Entwicklertools oder im DropDown-Menü **Ausführen.** Für die Fehlersuche oder um nachzuvollziehen, was ein Codestück genau bewirkt, führt man den Code in Einzelschritten aus. Statt der Funktionstaste **F5** drückt man dazu für jeden auszuführenden Befehl die Funktionstaste **F8** oder wählt die Funktion **Einzelschritt** im DropDown-Menü **Debuggen.**

Direkt ausführbare Prozeduren oder „Subs" bezeichnet man in VBA-Sprechweise auch als Makro. Sie beginnen mit dem Schlüsselwort `Sub` und das Klammerpaar `()` hinter ihrem Bezeichner ist leer.

2.2.4 VBA-Code speichern

Wenn man VBA in einer Office-Wirtsanwendung entwickelt, speichert man den VBA-Code in einer Datei der Wirtsanwendung. Office-Dateien mit Makros sind an ihrer Dateiendung zu erkennen: „.xlsm" für Excel-Arbeitsmappen mit VBA-Code, „.docm" für Word-Dokumente mit VBA-Code, „.pptm" für PowerPoint-Präsentationen mit VBA-Code und so weiter. Das „m" in der Dateiendung steht dabei für „Makros". Eine Office-Datei mit Makros hat ein anderes Format als eine Office-Datei ohne Makros. Beim Speichern der Datei muss man das Dateiformat entsprechend umstellen, sonst geht der VBA-Code verloren.

Wirtsanwendungen anderer Hersteller gehen möglicherweise anders mit VBA-Code um. In jedem Fall besteht ein VBA-Quellprogramm aus ganz gewöhnlichem Text und kann mit allen üblichen Texteditoren bearbeitet und als Textdatei gespeichert werden. VBA-Quellcode lässt sich wie anderer Text auch per Copy&Paste zwischen der Entwicklungsumgebung und anderen Dateien übertragen. Die Entwicklungsumgebung bietet außerdem eine Funktion **Datei exportieren,** die ein VBA-Modul in eine Textdatei exportiert. VBA-Dateien haben üblicherweise die Dateiendung „.bas". Die Funktion **Datei exportieren** findet man in dem Kontextmenü, das sich bei Rechtsklick auf ein Modul im **Projekt-Explorer** öffnet, vergleiche Abb. 2.2. Auch eine Funktion **Datei importieren** ist dort zu finden.

Weitere Funktionen zum Verwalten von VBA-Code bietet das Dialogfenster **Projekt-Eigenschaften**, das man im DropDown-Menü **Extras** der VBA-Entwicklungsumgebung öffnen kann. Hier kann man unter anderem einen Namen und eine Beschreibung für das VBA-Projekt festlegen und ein Kennwort angeben, wenn man den VBA-Code verstecken möchte.

2.2.5 VBA-Prozeduren in Anwendungen starten

In echten Anwendungen startet man VBA-Code natürlich nicht aus der Entwicklungsumgebung. VBA bietet vielfältige Möglichkeiten, um VBA-Skripte in einer Anwendung zugänglich zu machen.

- In Kap. 9 wird beschrieben, wie man eine mit VBA realisierte Funktion in das Menüband einer Office-Anwendung einbinden kann.
- Ein Symbol für eine solche Funktion lässt sich auch in die Symbolleiste für den Schnellzugriff integrieren. Wie das geht, erklärt ein Beispiel in Abschn. 13.6.
- Weiterhin kann man ein VBA-Skript automatisch anstoßen, wenn der Benutzer eine bestimmte Aktion ausführt, zum Beispiel beim Öffnen einer Arbeitsmappe. Diesen Ansatz nutzt und demonstriert ein Beispiel in Abschn. 13.5.
- Wenn man in Excel als Wirtsanwendung entwickelt, ist ein Button (Schaltfläche) eine sehr gute Lösung, um ein VBA-Skript zu starten. Dass sie auch sehr schnell und einfach zu realisieren ist, zeigt der folgende Abschnitt.

Auch ein Tastaturkürzel ist schnell als Starter für ein VBA-Skript eingerichtet. Dies empfiehlt sich für VBA-basierte Tools jedoch weniger, weil Benutzer es auf der Bedienoberfläche nicht finden können. Statt irgendwo einen Text zu platzieren, der auf die Funktion und das Tastaturkürzel hinweist, ist ein klickbares Bedienelement wie die bereits genannte Schaltfläche meist bedienfreundlicher.

2.2.6 VBA-Code mit einer Schaltfläche starten

Um ein Skript mit einer Schaltfläche zu verbinden, braucht man zunächst ein Skript. Das folgende kleine Demo-Skript genügt zum Ausprobieren. Man schreibt es wie in Abb. 2.2 in der Entwicklungsumgebung von Excel in ein Code-Modul.

```
Sub HalloWelt()
MsgBox "Hallo Welt!"
End Sub
```

Schaltflächen und weitere Steuerelemente findet man auf der Registerkarte **Entwicklertools** unter dem Werkzeugkoffer-Symbol mit der Beschriftung **Einfügen** in der Befehlsgruppe **Steuerelemente**. Am einfachsten ist die Schaltfläche zu benutzen, die im Bereich **Formularsteuerelemente** oben links angeordnet ist.

Nachdem man auf dem Arbeitsblatt eine solche Schaltfläche aufgezogen hat, öffnet sich sofort das Dialogfenster **Makro zuweisen** und bietet alle ausführbaren Subs an. In Abb. 2.3 gibt es lediglich das Makro „HalloWelt“**,** das man der Schaltfläche zuweisen kann**.** „Schaltfläche4_Klicken“ ist ein Vorschlag der Entwicklungsumgebung, um ein Makro zu bezeichnen, das man mit **Neu** oder **Aufzeichnen** erst generieren würde. **Neu** führt dabei in ein Codefenster der Entwicklungsumgebung und **Aufzeichnen** startet den Makrorecorder, der in Abschn. 2.2.7 beschrieben wird. Den Bezeichner „Schaltfläche4_Klicken“ sollte man nur für einmalig verwendeten Wegwerfcode beibehalten. Für dauerhaft genutzte Makros lohnt sich ein sprechenderer Bezeichner.